Федеральное государственное образовательное бюджетное
учреждение высшего образования
«Финансовый университет при Правительстве Российской Федерации»
(Финуниверситет)

Самарский финансово-экономический колледж
(Самарский филиал Финуниверситета)

УТВЕРЖДАЮ
Заместитель директора по учебно-методической работе
_____Л.А Косенкова
«21» февраля 20 22 г.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ОРГАНИЗАЦИИ И ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ ЗАНЯТИЙ ПО ПРОФЕССИОНАЛЬНОМУ МОДУЛЮ «ПМ.01 РАЗРАБОТКА МОДУЛЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ КОМПЬЮТЕРНЫХ СИСТЕМ»

СПЕЦИАЛЬНОСТЬ: 09.02.07 ИНФОРМАЦИОННЫЕ СИСТЕМЫ И ПРОГРАММИРОВАНИЕ

Самара – 2022

Методические указания по организации и выполнению практических занятий разработаны на основе рабочей программы по профессиональному модулю «Разработка модулей программного обеспечения для компьютерных систем», в соответствии с федеральным государственным образовательным стандартом среднего профессионального образования по специальности 09.02.07 Информационные системы и программирование, утвержденного приказом Министерства образования науки Российской Федерации от 09.12.2016 года № 1547, с учетом Профессионального стандарта, утвержденного приказом Министерства труда и социальной защиты Российской Федерации от 11 февраля 2014 г. № 647н «Об утверждении профессионального стандарта 06.011 Администратор баз данных» (зарегистрирован Министерством юстиции Российской Федерации 24 ноября 2014 г., регистрационный № 34846)
Присваиваемая квалификация: администратор баз данных

Разработчики:

Платковская Е.А. _____ Преподаватель Самарского филиала Финуниверситета

Методические указания по организации и выполнению практических занятий рассмотрены и рекомендованы к утверждению на заседании предметной (цикловой) комиссии естественно-математических дисциплин

Протокол от «24» сентября 20 22 г. № 5

Председатель ПЦК _____ М.В. Писцова

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Методические указания по выполнению практических работ по профессиональному модулю ПМ.01 Разработка модулей программного обеспечения для компьютерных систем разработаны с целью оказания помощи студентам специальности 09.02.07 Информационные системы и программирование и преподавателям по организации практических занятий по изучаемой дисциплине, в соответствии с требованиями федерального государственного стандарта среднего профессионального образования.

Методические разработка включает в себя краткие теоретические сведения, указания по выполнению практических работ, контрольные вопросы, формы контроля.

В соответствии с учебным планом на практические работы для студентов отводится 54 часа.

В результате изучения профессионального модуля студент должен освоить основной вид деятельности Осуществление интеграции программных модулей и соответствующие ему общие компетенции и профессиональные компетенции:

Перечень общих компетенций

Код	**Наименование общих компетенций**
ОК 01	Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам
ОК 02	Осуществлять поиск, анализ и интерпретацию информации, необходимой для выполнения задач профессиональной деятельности.
ОК 03	Планировать и реализовывать собственное профессиональное и личностное развитие
ОК 04	Работать в коллективе и команде, эффективно взаимодействовать с коллегами, руководством, клиентами.
ОК 05	Осуществлять устную и письменную коммуникацию на государственном языке с учетом особенностей социального и культурного контекста.
ОК 06	Проявлять гражданско-патриотическую позицию, демонстрировать осознанное поведение на основе традиционных общечеловеческих ценностей
ОК 07	Содействовать сохранению окружающей среды, ресурсосбережению, эффективно действовать в чрезвычайных ситуациях.
ОК 08	Использовать средства физической культуры для сохранения и укрепления здоровья в процессе профессиональной деятельности и поддержания необходимого уровня физической подготовленности
ОК 09	Использовать информационные технологии в профессиональной деятельности.
ОК 10	Пользоваться профессиональной документацией на государственном и иностранном языках

Перечень профессиональных компетенций

Код	**Наименование видов деятельности и профессиональных компетенций**
ВД 1	Разработка модулей программного обеспечения для компьютерных систем
ПК 1.1	Формировать алгоритмы разработки программных модулей в соответствии с техническим заданием
ПК 1.2	Разрабатывать программные модули в соответствии с техническим заданием
ПК 1.3	Выполнять отладку программных модулей с использованием специализированных программных средств
ПК 1.4	Выполнять тестирование программных модулей

ПК 1.5	Осуществлять рефакторинг и оптимизацию программного кода
ПК 1.6	Разрабатывать модули программного обеспечения для мобильных платформ

1.1.3. В результате освоения профессионального модуля студент должен:

Иметь практический опыт	В участии в соадминистрировании серверов; разработке политики безопасности SQL сервера, базы данных и отдельных объектов базы данных; применении законодательства Российской Федерации в области сертификации программных средств информационных технологий
уметь	проектировать и создавать базы данных; выполнять запросы по обработке данных на языке SQL; осуществлять основные функции по администрированию баз данных; разрабатывать политику безопасности SQL сервера, базы данных и отдельных объектов базы данных; владеть технологиями проведения сертификации программного средства
знать	модели данных, основные операции и ограничения; технологию установки и настройки сервера баз данных; требования к безопасности сервера базы данных; государственные стандарты и требования к обслуживанию баз данных

Количество практических работ по разделам

Раздел 1. Выполнение разработки программных модулей	16
Раздел 2. Поддержка и тестирование программных модулей	12
Раздел 3. Разработка мобильных приложений мобильных приложений	12
Раздел 4. Системное программирование	14

Перечень практических занятий

№п.п.	**НАЗВАНИЕ ПРАКТИЧЕСКИХ ЗАНЯТИЙ**
	МДК 01.01 Разработка программных модулей МДК.01.02 Поддержка и тестирование программных модулей
1	**Практическое занятие № 1.** Оценка сложности алгоритмов сортировки, писка, рекурсивных и эвристических алгоритмов.
2	**Практическое занятие № 2.** Работа с классами. Определение операций в классе. Создание наследованных классов
3	**Практическое занятие № 3.** Использование основных шаблонов.
4	**Практическое занятие № 4.** Использование порождающих, структурных, поведенческих шаблонов.
5	**Практическое занятие № 5.** Разработка приложения с использованием текстовых компонентов
6	**Практическое занятие № 6.** Оптимизация и рефакторинг кода.
7	**Практическое занятие № 7.** Разработка интерфейса пользователя.
8	**Практическое занятие №8.** Создание приложения с БД. Создание запросов, хранимых процедур к БД
	МДК.01.02 Поддержка и тестирование программных модулей
9	**Практическое занятие № 1.** Тестирование «белым ящиком»
10	**Практическое занятие № 2.** Тестирование «черным ящиком»

11	**Практическое занятие № 3.** Модульное тестирование
12	**Практическое занятие № 4.** Интеграционное тестирование
13	**Практическое занятие № 5.** Оформление документации на программные средства с использованием инструментальных средств.
	МДК.01.03 Разработка мобильных приложений
14	**Практическое занятие № 1.** Установка инструментария и настройка среды для разработки мобильных приложений
15	**Практическое занятие № 2.** Установка среды разработки мобильных приложений с применением виртуальной машины
16	**Практическое занятие № 3.** Создание эмуляторов и подключение устройств. Настройка режима терминала
17	**Практическое занятие № 4.** Создание нового проекта. Изучение и комментирование кода. Изменение элементов дизайна
18	**Практическое занятие № 5.** Подготовка стандартных модулей. Передача данных между модулями. Тестирование и оптимизация мобильного приложения
	МДК 01.04 Системное программирование
19	**Практическое занятие № 1.** Использование потоков.
20	**Практическое занятие № 2.** Обмен данными.
21	**Практическое занятие № 3.** Сетевое программирование сокетов.
22	**Практическое занятие №4.** Работы с буфером экрана.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ СТУДЕНТОВ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

Практическая работа №1 Оценка сложности алгоритмов сортировки

Цель работы: изучить основные алгоритмы внутренних сортировок и научиться решать задачи сортировок массивов различными методами (бинарная *пирамидальная сортировка*,метод Шелла, быстрая *сортировка* Хоара, *сортировка слиянием*).

При выполнении лабораторной работы для каждого задания требуется написать программу на языке С++, которая получает на входе числовые данные, выполняет генерацию и *вывод* массива *указанного типа* в зависимости от постановки задачи. В каждой задаче необходимо выполнить сортировку данных и реализовать один из алгоритмов: бинарной *пирамидальной сортировки*, сортировки по методу Шелла, быстройсортировки Хоара и *сортировки слиянием* в виде отдельных функций. Ввод данных осуществляется с клавиатуры или из файла с учетом требований к *входным данным*, содержащихся в постановке задачи. Ограничениями на *входные данные* является *диапазон* используемого *числового типа* данных в языке С++ и максимально допустимый размер объявляемого одномерного массива.

Теоретические сведения. Ознакомьтесь с материалом лекции.

Задания к лабораторной работе.

Выполните приведенные ниже задания.

1. На основании приведенных в лекции функций реализуйте алгоритмы внутренних сортировок.
2. Даны два целочисленных файла, упорядоченных по возрастанию. Сформировать третий файл на основе данных, который также упорядочен и представляет операцию с элементами исходных файлов:
   - объединение (содержит числа, принадлежащие хотя бы одному из множеств);
   - перечисление (числа, принадлежащие обоим множествам);
   - разность (числа, принадлежащие первому множеству, но не второму);
   - симметричную разность (объединение *разностей множеств*).
3. Заданы N (N<=5000) попарно различных длин отрезков. Вычислить количествоспособов, которыми из отрезков можно сложить треугольник.
4. Дана целочисленная квадратная матрица размером n. Упорядочить значения так,чтобы $a_{11}<=a_{12}<=<=a_{1n}<=a_{21}<=a_{22}<=<=a_{2n}<=<=a_{n1}<=a_{n2}<=<=a_{nn}$.
5. Дан целочисленный массив. Выполните проверку уникальности. Удалите измассива повторные вхождения чисел.

Указания к выполнению работы.

Каждое задание необходимо решить в соответствии с изученными алгоритмами внутренних сортировок: бинарной *пирамидальной сортировки*, сортировки по методу Шелла, быстрой сортировки Хоара и *сортировки слиянием*. Программные коды следует реализовать на языке С++. Рекомендуется воспользоваться материалами лекции 42, где подробно рассматриваются описание используемых в работе алгоритмов, примеры их реализации на языке С++. Программу для решения каждого задания необходимо разработать методом процедурной абстракции, используя функции. Этапы решения сопроводить комментариями в коде. В отчете следует отразить разработку и обоснование математической модели решения задачи, и примеры входных и выходных файлов.

Следует реализовать каждое *задание в соответствии* с приведенными этапами:

- изучить словесную постановку задачи, выделив при этом все виды данных;
- сформулировать математическую постановку задачи;
- выбрать *метод решения* задачи, если это необходимо;
- разработать графическую схему алгоритма;
- записать разработанный алгоритм на языке С++; разработать контрольный

тест к программе;

- отладить программу;
- представить отчет по работе.

Требования к отчету.

Отчет по лабораторной работе должен соответствовать следующей структуре.

Титульный лист.

Словесная постановка задачи. В этом подразделе проводится полное описание задачи. Описывается суть задачи, анализ входящих в нее физических величин, область их допустимых значений, единицы их измерения, возможные ограничения, анализ условий при которых задача имеет решение (не имеет решения), анализ ожидаемых результатов.

Математическая модель. В этом подразделе вводятся математические описания физических величин и математическое описание их взаимодействий. Цель подраздела – представить решаемую задачу в математической формулировке.

Алгоритм решения задачи. В подразделе описывается разработка структуры алгоритма, обосновывается *абстракция данных*, задача разбивается на подзадачи. Схема алгоритма выполняется по ЕСПД (ГОСТ 19.003-80 и ГОСТ 19.002-80).

Листинг программы. Подраздел должен содержать текст программы на языке программирования C++, реализованный в среде MS Visual Studio 2010.

Контрольный тест. Подраздел содержит наборы исходных данных и полученные в ходе выполнения программы результаты.

Выводы по лабораторной работе.

Ответы на контрольные вопросы.

Задача 5.4. Напишите программу, печатающую n-ое *число Фибоначчи*, которая имелабы *линейную сложность*.

Вот решение этой задачи, в котором переменные j и k содержат значения двух последовательных чисел Фибоначчи.

Текст программы

```
public class FibIv1 {
public static void main(String[] args) throws Exception {int n = Xterm.inputInt("Введите n -
> "); Xterm.print("f(" + n + ")");
if (n < 0) {
Xterm.print(" не определено\n");
} else if (n < 2) { Xterm.println(" = " + n);
} else {
long i = 0;long j = 1;long k;
int m = n;
while (--m > 0) {k  = j;
j += i;i  = k;
}
Xterm.println(" = " + j);
}
}
}
```

Следующий вопрос вполне естественен — а можно ли находить *числа Фибоначчи* ещебыстрее?

После изучения определенных разделов математики совсем просто вывести следующую формулу

$$f_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right].$$

f_n n легко проверить для небольших
Может показаться, что основываясь на ней, легко написать программу сосложностью $\Theta(1)$, не использующую итерации или рекурсии.

Текст программы

```
public class FibIv2 {
public static void main(String[] args) throws Exception {int n = Xterm.inputInt("Введите n -
> ");
double f = ( 1.0 + Math.sqrt(5.) ) / 2.0;
int j = (int)( Math.pow(f,n) / Math.sqrt(5.) + 0.5 );
Xterm.println("f(" + n + ") = " + j);
}
}
```

На самом деле эта *программа* использует *вызов функции* возведения в степень
{ Math.pow(f,n) }, которая не может быть реализована быстрее, чем за логарифмическое время ($\log_2 n$). Про алгоритмы, в которых количество операций примерно пропорционально $\log n$ (в информатике принято не указывать *основание* двоичного логарифма) говорят, что они имеет ***логарифмическую сложность*** ($\Theta(\log n)$).
Для вычисления n-го *числа Фибоначчи* существует такой *алгоритм*, программную реализацию которого мы приведем без дополнительных комментариев, — иначе нужно объяснять слишком много (*связь* чисел Фибоначчи со степенями некоторой матрицы порядка два, использование классов для работы с матрицами, *алгоритм* быстрого возведения матрицы в степень).

Задача . Напишите программу, печатающую n-ое n *число Фибоначчи*, которая имела бы *логарифмическую сложность*.

Текст программы

```
public class FibIv3 {
public static void main(String[] args) throws Exception {int n = Xterm.inputInt("Введите n -
> ");
Xterm.print("f(" +  + ")");if (n < 0) {
Xterm.println(" не определено");
} else if (n < 2) { Xterm.println(" = " + n);
} else {
Matrix b = new Matrix(1, 0, 0, 1);
Matrix c = new Matrix(1, 1, 1, 0);while (n>0) {
if ((n&1) == 0) {
n >>>= 1; c.square();
} else {
n  -=  1; b.mul(c);
}
}
Xterm.println(" = " + b.fib());
}
}
}
class Matrix {
private long a, b, c, d;
public Matrix(long a, long b, long c, long d) {this.a = a; this.b = b; this.c = c; this.d = d;
}
public void mul(Matrix m) {
```

```
long a1 = a*m.a+b*m.c; long b1 = a*m.b+b*m.d;long c1 = c*m.a+d*m.c; long d1 =
c*m.b+d*m.d;a = a1; b = b1; c = c1; d = d1;
}
public void square() {mul(this);
}
public long fib() {return b;
}
}
```

Если попробовать посчитать десятимиллионное *число Фибоначчи* с помощью этой и предыдущей программ, то разница во времени счета будет вполне очевидной. К сожалению, результат будет неверным (в обоих случаях) в силу ограниченности диапазона чисел типа long.

В заключение приведем сравнительную таблицу времен выполнения алгоритмов с различной сложностью и объясним, почему с увеличением быстродействия компьютеров важность использования быстрых алгоритмов значительно возрастает.

Рассмотрим четыре алгоритма решения одной и той же задачи, имеющие сложности $\log n$, n, n^2 и 2^n соответственно. Предположим, что второй из этих алгоритмов требует для своего выполнения на некотором компьютере при значении параметра $n = 10^3$ ровно одну минуту времени. Тогда времена выполнения всех этих четырех алгоритмов на том же компьютере при различных значениях параметра будут примерно такими, как в таблице 5.1.

Таблица Сравнительная таблица времен выполнения алгоритмов

Сложность алгоритма	**n=10**	**$n=10^3$**	**$n=10^6$**
log n	0.2 сек.	0.6 сек.	1.2 сек.
n	0.6 сек.	1 мин.	16.6 час.
n^2	6 сек.	16.6 час.	1902 года
2^n	1 мин.	10^{295} лет	10^{300000} лет

Когда начинающие программисты тестируют свои программы, то значения параметров, откоторых они зависят, обычно невелики. Поэтому даже если при написании программыбыл применен неэффективный *алгоритм*, это может остаться незамеченным. Однако, еслиподобную программу попытаться применить в реальных условиях, то ее практическая непригодность проявится незамедлительно.

С увеличением быстродействия компьютеров возрастают и значения параметров, для которых работа того или иного алгоритма завершается за приемлемое время. Таким образом, увеличивается среднее *значение* величины n, и, следовательно, возрастает величина отношения времен выполнения быстрого и медленного алгоритмов. **Чем быстрее компьютер, тем больше относительный проигрыш при использовании плохого алгоритма**!

Рекурсия.

Вычислить сумму элементов линейного массива.

При решении задачи используем следующее соображение: сумма равна нулю, если количество элементов равно нулю, и сумме всех предыдущих элементов плюс последний, если количество элементов не равно нулю.

```
Program Rec2;
Type LinMas = Array[1..100] Of Integer;Var A : LinMas;
I, N : Byte;
{Рекурсивная функция}
Function Summa(N : Byte; A: LinMas) : Integer;Begin
If N = 0 Then Summa := 0
```

```
Else Summa := A[N] + Summa(N - 1, A)End;
{Основная программа}Begin
Write('Количество элементов массива? ');ReadLn(N);
Randomize;
For I := 1 To N DoBegin
A[I] := -10 + Random(21); Write(A[I] : 4)
End;
WriteLn;
WriteLn('Сумма: ', Summa(N, A))End.
```

Контрольные вопросы

1. Чем можно объяснить многообразие алгоритмов сортировок?
2. Почему на данный момент не существует универсального алгоритма сортировки?
3. Как соблюдение свойств устойчивости и естественности влияет на трудоемкость алгоритма сортировки?
4. За счет чего в алгоритмах быстрых сортировок происходит выигрыш при выполнении операций сравнения и перестановок?
5. Какие из перечисленных алгоритмов наиболее эффективны на почти отсортированных массивах: бинарная *пирамидальная сортировка*, *сортировка слиянием,* сортировка Шелла и сортировка Хоара? За счет чего происходит выигрыш?
6. Почему алгоритмы быстрых сортировок не дают большого выигрыша при малых размерах массивов?
7. В чем преимущества и недостатки по отношению друг к другу следующих алгоритмов сортировок: бинарная *пирамидальная сортировка*, *сортировка слиянием*, сортировка Шелла и сортировка Хоара?
8. Как определить, какому алгоритму сортировки отдать предпочтение при решении задачи?

Практическая работа № 2 Работа с классами. Определение операций в классе. Создание наследованных классов

Цель работы: получение практических навыков работы с классами и объектами

Теоретический материал

Описанием объекта является класс, а объект представляет экземпляр этого класса. Можноеще провести следующую аналогию. У нас у всех есть некоторое представление о человеке, у которого есть имя, возраст, какие-то другие характеристики. То есть некоторый шаблон - этот шаблон можно назвать классом. Конкрет4н7ое воплощение этого шаблона может отличаться,

например, одни люди имеют одно имя, другие - другое имя. И реально существующий человек (фактически экземпляр данного класса) будет представлять объект этого класса.

По умолчанию проект консольного приложения уже содержит один класс Program, скоторого и начинается выполнение программы.

По сути класс представляет новый тип, который определяется пользователем. Классопределяется с помощью ключевого слова class:

class Person

Где определяется класс? Класс можно определять внутри пространства имен, вне пространства имен, внутри другого класса. Как правило, классы помещаются в отдельные файлы.Но в данном случае поместим новый класс в файле, где располагается класс Program. То есть файл Program.cs будет выглядеть следующим образом:

using System;

```
namespace HelloApp
{
class Person
{

}
class Program
{
static void Main(string[] args)
{
}
}
}
```

Вся функциональность класса представлена его членами - полями (полями называются переменные класса), свойствами, методами, событиями. Например, определим в классе Person поля и метод:

```
using System; namespace HelloApp
{
class Person
{
public string name; // имя public int age = 18;        // возраст

public void GetInfo()
{
Console.WriteLine($"Имя: {name}  Возраст: {age}");
}
}
class Program
{
static void Main(string[] args)
{
Person tom;
}                                                    }
}
```

В данном случае класс Person представляет человека. Поле name хранит имя, а поле age -возраст человека. А метод GetInfo выводит все данные на консоль. Чтобы все данные были доступны вне класса Person переменные и метод определены с модификатором public. Посколькуполя фактически те же переменные, им можно присвоить начальные значения, как в случае выше,поле age инициализировано значением 18.

Так как класс представляет собой новый тип, то в программе мы можем определять переменные, которые представляют данный тип. Так, здесь в методе Main определена переменная tom, которая представляет класс Person. Но пока эта переменная не указывает ни на какой объект и по умолчанию она имеет значение null. Поэтому вначале необходимо создать объект класса Person.

Конструкторы

Кроме обычных методов в классах используются также и специальные методы, которые называются конструкторами. Конструкторы вызываются при создании нового объекта данного класса. Конструкторы выполняют инициализацию объекта.

Конструктор по умолчанию

Если в классе не определено ни одного конструктора, то для этого класса автоматически создается конструктор по умолчанию. Такой конструктор не имеет параметров и не имеет тела.

Выше класс Person не имеет никаких конструкторов. Поэтому для него автоматически создается конструктор по умолчанию. И мы можем использовать этот конструктор. В частности,создадим один объект класса Person:

```
class Person
{
public string name; // имя public int age;      // возраст

public void GetInfo()
{
Console.WriteLine($"Имя: {name}  Возраст: {age}");
}
}
class Program
{
static void Main(string[] args)
{
Person tom = new Person(); tom.GetInfo();  // Имя: Возраст: 0

tom.name = "Tom";tom.age = 34;
tom.GetInfo(); // Имя: Tom Возраст: 34Console.ReadKey();
}
}
```

Для создания объекта Person используется выражение new Person(). Оператор new выделяет память для объекта Person. И затем вызывается конструктор по умолчанию, который не принимает никаких параметров. В итоге после выполнения данного выражения в памяти будетвыделен участок, где будут храниться все данные объекта Person. А переменная tom получит ссылку на созданный объект.

Если конструктор не инициализирует значения переменных объекта, то они получают значения по умолчанию. Для переменных числовых типов это число 0, а для типа string и классов

- это значение null (то есть фактически отсутствие значения).

После создания объекта мы можем обратиться к переменным объекта Person черезпеременную tom и установить или получить их значения, например, tom.name = "Tom";.

Консольный вывод данной программы

```
Имя:              Возраст: 0
Имя: Tom                  Возраст: 34Создание конструкторов
```

Выше для инициализации объекта использовался конструктор по умолчанию. Однако мысами можем определить свои конструкторы:

```
class Person
{
public string name;public int age;
public Person() { name = "Неизвестно"; age = 18; }// 1 конструкторpublic
Person(string n) { name = n; age = 18; }                                    // 2 конструктор
public Person(string n, int a) { name = n; age = a; } // 3 конструктор public void GetInfo()
{
Console.WriteLine($"Имя: {name}  Возраст: {age}");
}
}
```

Теперь в классе определено три конструктора, каждый из которых принимает различноеколичество параметров и устанавливает значения полей класса. Используем эти конструкторы:

```
static void Main(string[] args)
```

```
{
Person tom = new Person();                    // вызов 1-ого конструктора без параметров
Person bob = new Person("Bob");               //вызов 2-ого конструктора с одним параметром Person sam = new Person("Sam", 25); // вызов 3-его конструктора с двумя параметрами


bob.GetInfo();                    // Имя: Bob Возраст: 18 tom.GetInfo();   // Имя: Неизвестно Возраст: 18sam.GetInfo();       // Имя: Sam  Возраст: 25
}
```

Консольный вывод данной программы:

```
Имя: Неизвестно  Возраст: 18Имя: Bob Возраст: 18
Имя: Sam  Возраст: 25
```

При этом если в классе определены конструкторы, то при создании объекта необходимоиспользовать один из этих конструкторов.

Стоит отметить, что начиная с версии C# 9.0 мы можем сократить вызов конструктора,убрав из него название типа:

```
Person tom = new ();                    // аналогично new Person();
Person bob = new ("Bob");               // аналогично new Person("Bob"); Person sam = new ("Sam", 25); // аналогично new Person("Sam", 25);Ключевое слово this
```

Ключевое слово this представляет ссылку на текущий экземпляр класса. В каких ситуациях оно нам может пригодиться? В примере выше определены три конструктора. Все триконструктора выполняют однотипные действия - устанавливают значения полей name и age. Но этих повторяющихся действий могло быть больше. И мы можем не дублировать функциональность конструкторов, а просто обращаться из одного конструктора к другому через ключевое слово this, передавая нужные значения для параметров:

```
class Person
{                                                    50
public string name;
public int age;

public Person() : this("Неизвестно")
{
}
public Person(string name) : this(name, 18)
{
}
public Person(string name, int age)
{
this.name = name;this.age = age;
}
public void GetInfo()
{
Console.WriteLine($"Имя: {name}  Возраст: {age}");
}
}
```

В данном случае первый конструктор вызывает второй, а второй конструктор вызывает третий. По количеству и типу параметров компилятор узнает, какой именно конструктор вызывается. Например, во втором конструкторе:

```
public Person(string name) : this(name, 18)
{
}
```

идет обращение к третьему конструктору, которому передаются два значения. Причем вначале будет выполняться именно третий конструктор, и только потом код второго конструктора.

Также стоит отметить, что в третьем конструкторе параметры называются также, как иполя класса.

```
public Person(string name, int age)
{
this.name = name;this.age = age;
}
```

И чтобы разграничить параметры и поля класса, к полям класса обращение идет через ключевое слово this. Так, в выражении this.name = name; первая часть this.name означает, что name - это поле текущего класса, а не название параметра name. Если бы у нас параметры и поля назывались по-разному, то использовать слово this было бы необязательно. Также через ключевое слово this можно обращаться к любому полю или методу.

Инициализаторы объектов

Для инициализации объектов классов можно применять инициализаторы. Инициализаторы представляют передачу в фигурных скобках значений доступным полям и свойствам объекта:

```
Person tom = new Person { name = "Tom", age=31 };tom.GetInfo();        // Имя: Tom Возраст: 31
```

С помощью инициализатора объектов можно присваивать значения всем доступным полям и свойствам объекта в момент создания без явного вызова конструктора.

При использовании инициализаторов следует учитывать следующие моменты:

С помощью инициализатора мы можем установить значения только доступных из внешнего кода полей и свойств объекта. Например, в примере выше поля name и age имеют модификатор доступа public, поэтому они доступны из любой части программы.

Инициализатор выполняется после конструктора, поэтому если и в конструкторе, и в инициализаторе устанавливаются значения одних и тех же полей и свойств, то значения, устанавливаемые в конструкторе, заменяют5с1я значениями из инициализатора.

C# является полноценным объектно-ориентированным языком. Это значит, что программу на C# можно представить в виде взаимосвязанных взаимодействующих между собой объектов.

Описанием объекта является **класс**, а объект представляет экземпляр этого класса. Можно еще провести следующую аналогию. У нас у всех есть некоторое представление о человеке, у которого есть имя, возраст, какие-то другие характеристики. То есть некоторый шаблон - этот шаблон можно назвать классом. Конкретное воплощение этого шаблона может отличаться, например, одни люди имеют одно имя, другие - другое имя. Иреально существующий человек (фактически экземпляр данного класса) будет представлять объект этого класса.

По умолчанию проект консольного приложения уже содержит один класс Program, с которого и начинается выполнение программы.

По сути класс представляет новый тип, который определяется пользователем. Класс определяется с помощью ключевого слова class:

```
1 class Person
2 {
3       }
```

Где определяется класс? Класс можно определять внутри пространства имен, в пространства имен, внутри другого класса. Как правило, классы помещаются в отдельн файлы. Но в данном случае поместим новый класс в файле, где располагается класс Program. есть файл Program.cs будет выглядеть следующим образом:

```
1  using System;
2
3  namespace HelloApp4     {
5      class Person
6      {
7
8      }
9      class Program
10     {
11         static void Main(string[] args)
12         {
13
14         }
15     }
16 }
```

Вся функциональность класса представлена его членами - полями (полями называются переменные класса), свойствами, методами, событиями. Например, определим в классе Person поля и метод:

```
1  using System;
2
3  namespace HelloApp4     {
5      class Person
6      {
7          public string name; // имя
8          public int age = 18;     // возраст9
10         public void GetInfo()
11
12
13
14     }
{
Console.WriteLine($"Имя: {name}  Возраст: {age}");
}
15 }
class Program
{
static void Main(string[] args)
{
Person tom;
}
}
```

В данном случае класс Person представляет человека. Поле name хранит имя, а поле age - возраст человека. А метод GetInfo выводит все данные на консоль. Чтобы все данные были доступны вне класса Person переменные и метод определены с модификатором public. Поскольку поля фактически те же переменные, им можно присвоить начальные значения, как в случае выше, поле age инициализировано значением 18.

Так как класс представляет собой новый тип, то в программе мы можем определять переменные, которые представляют данный тип. Так, здесь в методе Main определена переменная tom, которая представляет класс Person. Но пока эта переменная не указывает ни на какой объект и по умолчанию она имеет значение **null**. Поэтому вначале необходимо создать объект класса Person.

Конструкторы

Кроме обычных методов в классах используются также и специальные методы, которые называются **конструкторами**. Конструкторы вызываются при создании нового объекта данного класса. Конструкторы выполняют инициализацию объекта.

Конструктор по умолчанию

```
1  class Person
2  {
3      public string name; // имя
4      public int age;     // возраст5
6      public void GetInfo()
7      {
8          Console.WriteLine($"Имя: {name} Возраст: {age}");9    }
10 }
11 class Program12    {
13     static void Main(string[] args)14          {
15         Person tom = new Person();
16         tom.GetInfo();      // Имя: Возраст: 017
18         tom.name = "Tom";
19         tom.age = 34;
20        tom.GetInfo(); // Имя: Tom Возраст: 34Console.ReadKey();
21        }
22        }
```

Для создания объекта Person используется выражение new Person(). Оператор **new** выделяет память для объекта Person. И затем вызывается конструктор по умолчанию, который не принимает никаких параметров. В итоге после выполнения данного выражения в памяти будет выделен участок, где будут храниться все данные объекта Person. А переменная tom получит ссылку на созданный объект.

Если конструктор не инициализирует значения переменных объекта, то они получают значения по умолчанию. Для переменных числовых типов это число 0, а для типа string и классов - это значение **null** (то есть фактически отсутствие значения).

После создания объекта мы можем обратиться к переменным объекта Person через переменную tom и установить или получить их значения, например, tom.name = "Tom";.

Консольный вывод данной программы:

```
Имя:    Возраст: 0
Имя: Tom       Возраст: 34
```

Создание конструкторов

Выше для инициализации объекта использовался конструктор по умолчанию. Однако мы сами можем определить свои конструкторы:

```
1  class Person
2  {
3      public string name;
4      public int age;5
6      public Person() { name = "Неизвестно"; age = 18; }       // 1 конструктор7
8      public Person(string n) { name = n; age = 18; }          // 2 конструктор9
10     public Person(string n, int a) { name = n; age = a; } // 3 конструктор11
12     public void GetInfo()
13     {
14         Console.WriteLine($"Имя: {name} Возраст: {age}");15  }
```

```
16 }
```

Теперь в классе определено три конструктора, каждый из которых принимает различное количество параметров и устанавливает значения полей класса. Используем эти конструкторы:

```
1  static void Main(string[] args)
2  {
3      Person tom = new Person();          // вызов 1-ого конструктора без параметров
4      Person bob = new Person("Bob");     //вызов 2-ого конструктора с одним
параметром
5      Person sam = new Person("Sam", 25); // вызов 3-его конструктора с
двумя
6  параметрами
7
8      bob.GetInfo();          // Имя: Bob  Возраст: 18
9      tom.GetInfo();          // Имя: Неизвестно  Возраст: 18
10 sam.GetInfo();              // Имя: Sam Возраст: 2511
                                                    }
```

Консольный вывод данной программы:

Имя: Неизвестно Возраст: 18 Имя: Bob Возраст: 18

Имя: Sam Возраст: 25

При этом если в классе определены конструкторы, то при создании объекта необходимо использовать один из этих конструкторов.

Стоит отметить, что начиная с версии C# 9.0 мы можем сократить вызов конструктора, убрав из него название типа:

```
1 Person tom = new ();            // аналогично new Person();
2 Person bob = new ("Bob");       // аналогично new Person("Bob");
3 Person sam = new ("Sam", 25);  // аналогично new Person("Sam", 25);
```

Ключевое слово this

Ключевое слово this представляет ссылку на текущий экземпляр класса. В каких ситуациях оно нам может пригодиться? В примере выше определены три конструктора. Все три конструктора выполняют однотипные действия - устанавливают значения полей name и age. Но этих повторяющихся действий могло быть больше. И мы можем не дублировать функциональность конструкторов, а просто обращаться из одного конструктора к другому через ключевое слово this, передавая нужные значения для параметров:

```
1  class Person
2  {
3      public string name;
4      public int age;5
6      public Person() : this("Неизвестно")7 {
8      }
9      public Person(string name) : this(name, 18)10          {
11     }
12     public Person(string name, int age)13  {
14         this.name = name;
15     this.age = age;16          }
17     public void GetInfo()
18     {
19         Console.WriteLine($"Имя: {name} Возраст: {age}");20  }
```

```
21 }
```

В данном случае первый конструктор вызывает второй, а второй конструктор вызывает третий. По количеству и типу параметров компилятор узнает, какой именно конструктор вызывается. Например, во втором конструкторе:

```
1 public Person(string name) : this(name, 18)
2 {
3 }
```

идет обращение к третьему конструктору, которому передаются два значения. Причем в начале будет выполняться именно третий конструктор, и только потом код второго конструктора.

Также стоит отметить, что в третьем конструкторе параметры называются также, как и поля класса.

```
1 public Person(string name, int age)
2 {
3     this.name = name;
4 this.age =  age;5      }
```

И чтобы разграничить параметры и поля класса, к полям класса обращение идет через ключевое слово this. Так, в выражении this.name = name; первая часть this.name означает, что name - это поле текущего класса, а не название параметра name. Если бы у нас параметры и поля назывались по-разному, то использовать слово this было бы необязательно. Также через ключевое слово this можно обращаться к любому полю или методу.

Инициализаторы объектов

Для инициализации объектов классов можно применять инициализаторы. Инициализаторы представляют передачу в фигурных скобках значений доступным полям и свойствам объекта:

```
Person tom = new Person { name = "Tom", age=31 };
tom.GetInfo(); // Имя: Tom  Возраст: 31
```

С помощью инициализатора объектов можно присваивать значения всем доступным полям и свойствам объекта в момент создания без явного вызова конструктора.

При использовании инициализаторов следует учитывать следующие моменты:

С помощью инициализатора мы можем установить значения только доступных из внешнего кода полей и свойств объекта. Например, в примере выше поля name и age имеют модификатор доступа public, поэтому они доступны из любой части программы.

Инициализатор выполняется после конструктора, поэтому если и в конструкторе, и в инициализаторе устанавливаются значения одних и тех же полей и свойств, то значения,устанавливаемые в конструкторе, заменяются значениями из инициализатора.

Практическое занятие № 3. Использование основных шаблонов.

Цель работы: область применения основных шаблонов

Теоретический материал

Любая структура данных и алгоритмы имеют дополнительную ценность, если они могут хранить и работать с данными различных типов. Такая универсальность (или что одно и то же, независимость от данных) может быть достигнута в Си++ различными способами:

- в обычном Си (см. **9.3**) «переход от типа к типу» и абстрагирование от конкретного хранимого типа возможно на основе преобразования типов указателей и использования указателя **void*,** олицетворяющего «память вообще». Совместно с механизмом динамического связывания возможно создание алгоритмо8в9, в которых фрагмент, ответственный за работу с конкретным типом данных, передается через

указатель на функцию;

· в Си++ на основе **полиморфизма** (виртуальных функций) возможно создание интерфейсных классов, способных объединять единым механизмом доступа различные классы. Если эти классы являются «обертками» известных типов данных, то независимость от типов хранимых данных можно обеспечить ссылками на интерфейсный класс.

Однако приведенные выше способы не обеспечивают **синтаксической** совместимости, т.е. они реализуются как технологические приемы, а не как элементы языка. По аналогии с переопределением операций (см. **10.3**) хотелось бы использование естественного синтаксиса, гдевместо **int, double** и т.п. фигурировало бы абстрактное обозначение типа, например, **T.**

Такое средство, позволяющее создавать заготовку функции или целого класса, в которомвместо конкретного имени типа данных будет фигурировать его символическое обозначение, называется **шаблоном**. В первом приближении смоделировать шаблон в обычном Си можно с использованием директив препроцессора для подстановки имен – **define.** Обозначив именами **T** и **N** тип и размерность массива, можно создать класс с использованием этих имен везде, где это необходимо.

```
//-----------------------------------------------------------------
// Имитация шаблона в обычном Си
#define T int                    // Параметры заданы через подстановку имен
#define N 100                    // Тип элементов и размерность массива struct
Array{
T Data[N];
int k;                                  // Текущее кол-во элементов Array(){ k=0; }
                                        // Конструктор - массив пустvoid add(T
&v){                                    // Добавление элемента
if (k>=N) return;
Data[k++]=v;}
T remove(int m){ // Удаление элемента по номеру
T foo;
if (m>=k) return foo;foo=Data[m];
for(int i=m;i<k-1;i++)
Data[i]=Data[i+1];k--; return foo; }
};
void main(){
Array A;
int B[10]={6,2,8,3,56,7,89,5,7,9};
for (int i=0;i<10;i++) A.add(B[i]);cout << A.remove(2) << endl; }
```

Чтобы применить ее для другого типа данных, нужно отредактировать директиву **define,** заменив, например, имя **int** на **double.** Но все же основным недостатком этой модели является однократность ее использования. В Си++ текстовая заготовка класса позволяет из одного описания создавать множество классов, отличающихся типом используемых объектови размерностями данных.

Итак, шаблон можно определить как *текстовую заготовку определения класса с параметром, обозначающим тип используемых внутри переменных.* Сразу же, не дожидаясь описания синтаксиса, отметим особенности трансляции шаблона:

· шаблон является описанием *группы классов,* отличающихся используемым типом данных;

· при создании (определении) объекта шаблонного класса указывается тип данных, для которого он создается;

· при определении объекта шаблонного класса конкретный тип

подставляется в шаблон вместо параметра и создается текстовая заготовка *экземпляра класса,* которая транслируется и дает собственный оригинальный программный код;

· и только затем происходит трансляция определения самого объекта.

Самое главное, в отличие от обычных объектов, объект шаблонного класса требуетотдельного экземпляра класса для того типа, который обозначен в объекте.

Замечание: при чтении транслятором шаблона заголовка класса и шаблонов встроенных в него функций и методов их трансляция **не производится.** Это происходит в другое время: притрансляции определения объекта шаблонного класса генерируется текстовая заготовка экземпляра класса. Отсюда некоторый нюансы:

· чтобы проверить, как транслируется шаблон, нужно описать хотя бы один объект шаблонного класса;

· весь шаблон, в том числе и шаблоны методов, следует размещать в заголовочном файле проекта;

· на каждый тип данных – параметр шаблона создается отдельный класс и в сегментекоманд размещается программный код его методов.

Следующим примером попробуем «убить двух зайцев». Во-первых, пояснить довольно витиеватый синтаксис шаблона, а во-вторых, выделить особенности реализации структур данных и использованием технологии ООП. Основной принцип шаблона, добавление к имени класса

«довеска» в виде имени – параметра (например, **vector<T>**). Это имя обозначает внутренний тип данных, который может использоваться в любом месте класса: как указатель, ссылка, формальный параметр, результат, локальная или статическая переменная. Во всем остальном шаблон не отличается от обычного класса. Само имя шаблона (**vector**) теперь обозначает не один класс, а группу классов, отличающихся внутренним типом данных.

```
//.....
// ----------Шаблон СД - динамический массив указателей
template <class T> class vector{
int sz,n;                                   // Размерность ДМУ и кол-ко
элементов
T **obj;                                  // Массив указателей на
параметризованныеpublic:                  // объекты типа T
T *operator[](int);                         // оператор [int] возвращает указатель на
// параметризованный объект типа T
operator int();                           // Возвращает текущее количество
указателейint append(T*);                  // Добавление указателя на объект типа T
int index(T*);                             // Поиск индекса хранимого объекта
vector(int);                              // Конструктор
~vector(){ delete []obj; } // Деструктор
};
```

Данный шаблон может использоваться для порождения объектов-векторов, каждый из которых хранит указатели объекты определенного типа. Имя класса при этом составляется из имени шаблона **vector** и имени типа данных (класса), который подставляется вместо параметра **T.**

```
vector<int>                 a; vector<double> b; extern class time;vector<time>
                            c;
```

При определении каждого вектора с новым типом объектов транслятором генерируется описание нового класса по заданному шаблону (естественно, неявно в процессе трансляции). Например, для типа **int** транслятор получит.

```
class vector<int>{
```

```
        int sz,n;                                   // Размерность ДМУ и кол-ко
элементов
        int **obj;                                  // Массив указателей на
параметризованныеpublic:                            // объекты типа T
        int *operator[](int);                       // оператор [int] возвращает указатель на
        // параметризованный объект типа T
        operator int();                             // Возв9р1ащает текущее количество
указателейint append(int *);                        // Добавление указателя на объект типа T
        int index(int *);                           // Поиск индекса хранимого объекта
vector(int);                                        // Конструктор
        ~vector(){ delete []obj; } // Деструктор
        };
```

Обратите внимание, что это *иллюстрация* принципа подстановки, а не фрагмент программы с синтаксисом Си++. Далее следует утверждение типа «масло масляное»: встроенные функции (методы) шаблонного класса – есть шаблонные функции. Это означает, что методы класса, включенные в шаблон, также должны «заготовками» с тем же самым параметром, то естьгенерироваться для каждого нового типа данных. То же самое касается переопределяемыхоператоров.

```
        //.....105-02.cpptemplate <class T>
vector<T>::operator int()
        { return n; }
        template <class T> T* vector<T>::operator[](int k){return k>=n ? NULL : obj[k]; }
        template <class T> int vector<T>::index(T *pobj){for ( int i=0; i<n; i++)
        if (pobj == obj[i]) return i;return -1;}
        template <class T> vector<T>::vector(int sz0){sz=sz0; n=0; obj=new T*[sz]; }
        template <class T> int vector<T>::append(T *pobj){if (n>=sz) return 0;
        obj[n++]=pobj;return 1;}
```

Приведенный пример касается только методов, «вынесенных» из заголовка класса. Для каждого из них пишется отдельный шаблон, а сам класс фигурирует в нем под именем вида **vector<T>.** Возможность непосредственного определения методов в заголовке шаблонногокласса (inline-методов) также остается.

Шаблоны могут иметь также и параметры-константы, которые используются для статического определения размерностей внутренних структур данных. Кроме того, шаблон может использоваться для размещения не только указателей на параметризованные объекты, но и самих объектов. В качестве примера рассмотрим шаблон для построения циклической очереди ограниченного размера на основе статического массива (см. **6.1**), хранящей непосредственно сами объекты (значения, а не указатели).

```
        //.....105-03.cpp
        // ----------------Шаблон с параметром-константой
        template <class T,int size> class FIFO{
        int fst,lst;                                // Индексы начала и конца очереди
        T queue[size];                              // Массив объектов класса T
размерности size
        public:
        T from();                                   // Функции включения-исключения
        int into(T);                                //
        FIFO(){ fst=lst=0; }                        // Конструктор
        };
        template <class T,int size> T FIFO<T,size>::from(){T work=0;
```

```
        if (fst !=lst){
        work = queue[lst++];if (lst==size) lst=0;
        }
        return work;}                                   92
        template <class T,int size> int FIFO<T,size>::into(T obj) {
        if ((fst+1)%size==lst)
        return 0;                                       // Проверка на переполнение
queue[fst++] = obj;
        if (fst==size) fst=0;return 1; }
```

Объекты такого шаблонного класса при определении имеют два параметра: тип данных иконстанту – статическую размерность.

```
        struct x {…}; FIFO<double,100>     a;
        FIFO<int,20>                       b;
        FIFO<x,50>                         c;
```

Особенности разработки шаблонов структур данных

Шаблон является исключительно средством подстановки, он ничего не меняет в ни в существе класса структуры данных, для которой строится шаблон, ни в принципах передачи объектов класса – параметров шаблона. Во-первых, это касается способа хранения объектов в структуре данных: она может содержать указатели, а может и сами объекты (копии, значения):

· если шаблон хранит указатели на объекты, то он не касается проблем корректного копирования объектов и «не отвечает» за их создание и уничтожение. Деструктор шаблона обязан уничтожить динамические компоненты структуры данных (динамические массивы указателей, элементы списка), но он обычно не уничтожает хранимые объекты;

· если шаблон хранит сами объекты, то он «должен быть уверен» в корректном копировании объектов при их записи и чтении из структуры данных (конструктор копирования о переопределение присваивания для объектов, содержащих динамические данные). При разрушении структуры данных разрушаются и копии хранимых в ней объектов.

Во-вторых, шаблон может использовать ряд стандартных операций по отношению к типуданных – параметру шаблона: *присваивание (копирование), сравнение*, а также *ввод и вывод* в стандартные потоки. При использовании шаблона с параметром – не базовым типом, а классом,необходимо следить, чтобы эти операции были в нем переопределены.

Шаблоны классов списков

Для представления списка или дерева необходимы две сущности: элементы списка (вершины дерева), связанные между собой и заголовок – указатель на первый элемент списка (корневую вершину). В технологии ООП есть два равноправных решения:

· разрабатывается один класс, объекты которого играют разную роль в процессеработы класса. Первый объект – заголовок, создается программой (статически или динамически),доступен извне и не содержит данных (по крайней мере, в момент конструирования). Остальные объекты, содержащие данные, создаются динамически методами, работающими с первым объектом. Этот вариант имеет некоторые тонкости, связанные с тем, что программа должна уметь различать, где она работает с элементом списка, а где с элементом-заголовком;

· разрабатывается два класса – класс элементов списка и класс списка как такового, содержащего, как минимум, его заголовок. Объекты первого (вспомогательного) класса пользователем (программой, работающей с классом) не создаются. Они все – динамические, и ихпорождают методы второго (основного) класса;

Ход работы

Рассмотрим первый вариант на примере шаблона односвязного списка. Коллизий, связанных с распознаванием заголовочного элемента и элемента, содержащего данные, не возникает. Все методы, описанные в заголовке класса, применяются только по отношению к заголовочному элементу (в контексте все время фигурирует установка на первый элемент спискав виде **p=next**, либо установка на «предыдущий»-заголовочный в виде **p=this**). Часть простых методов реализована в самом заголовке класса, в них для указателей на элементы спискаиспользуется сокращенный контекст вида **list *p**, хотя класс **list** является шаблонным.

```
//.....105-04.cpp
template <class T> class list{                    93
        list<T> *next;          // Указатель на следующий в списке
данными
T data;                         // Элемент списка хранит сам объект
list(T& v)  { data=v; next=NULL;}  // Скрытый конструктор для элементов
с
public:  list() { next=NULL; }                    // Конструктор для элемента -
заголовка
~list(){                                          // Деструктор рекурсивныйif
(next!=NULL) delete next;
}                                                 // рекурсивное удаление следующего
void front(T& v){                                 // Включение в начало
list *q=new list(v);
q->next=next; next=q;
}
void end(T &v){                                   // Включение в конец list
*p,*q=new list(v);                                // Дойти до последнего
for(p=this;p->nezt!=NULL;p=p->next);
p->next=q;                                        // Следующий за последним - новый
}
void insert(T&,int);                              // Включение по логическому
номеруvoid insert(T&);                            // Включение с сохранением
порядка
T remove(int);                                    // Извлечение по ЛН
operator int(){                                   // Приведение к int = размер списка
list *q; int n;
for(q=next,n=0; q!=NULL; n++,q=q->next);return n;
}
friend ostream &operator<<(ostream&, list<T>&);friend istream
&operator>>(istream&, list<T>&);
};                                                // Переопределенный вывод в поток
```

Единственная коллизия возникает в конструкторе и деструкторе. Для заголовочного элемента используется конструктор без параметров. Методы списка, создавая динамические элементы – объекты того же класса, используют закрытый конструктор с параметром – значением, хранимым в элементе списка. В методе удаления элемента из списка кроме логического исключения выбранного элемента из цепочки у него обнуляется указатель на следующего (**p->next=NULL**). Это делается для того, чтобы исключить рекурсивный вызов деструктора при удалении одного элемента списка.

Специфика односвязного списка проявляется в реализации методов вставки по номеру, с сохранением порядка и удаления. Текущий указатель в цикле ссылается на

предыдущий элемент списка по отношению к искомому (поэтому, например, используется выражение **q->next->data** для сравнения значения в искомом элементе).

```
//-------------------------------------------------105-04.cpp template <class T>
void list<T>::insert(T &newdata,int n){
    list<T> *p,*q;
    p=new list<T>(newdata);
    for (q=this; q->next!=NULL && n!=0; n--,q=q->next);
    p->next=q-> next;              // Вставка после текущего
    q->next=p; }                   // Отсчет от заголовка
    //-------------------------------------------------------------------------
template <class T> void list<T>::insert(T &newdata){
    list<T> *p,*q;
    p=new list<T>(newdata);                    // Сравнивать СО СЛЕДУЮЩИМ
for (q=this; q->next!=NULL && newdata>q->next->data; q=q->next);
    p->next =q->next;                          // Вставка ПОСЛЕ текущегоq-
>next=p;}
    //-------------------------------------------------------------------------
template <class T> T list<T>::remove(int n){
    list<T> *q,*p;                             // Указатель на ПРЕДЫДУЩИЙ
    for (q=this; q->next!=NULL && n!=0; n--,q=q->next);T val;
    if (q->next==NULL) return val;             // Такого нетval=q->next->data;
    p=q->next; q->next=p->next;                // Исключить СЛЕДУЮЩИЙ из
цепочки
    p->next=NULL;                              // Для исключения рекурсивного
деструкторadelete p;
    return val;}
```

Переопределенные операции ввода-вывода в стандартные потоки поддерживают саморазворачивающийся симметричный формат представления данных, использующий счетчик элементов. При вводе в соответствии с прочитанным значением счетчика многократно читается объект класса **T,** значение которого каждый раз включается в список.

```
//-------------------------------------------------105-04.cpp
template <class T> ostream &operator<<(ostream &O, list<T> &R){list<T> *q;
O << (int)R << endl;
for (q=R.next; q!=NULL; q=q->next) O << q->data << endl;return O; }
template <class T> istream &operator>>(ostream &I, list<T> &R){T val; int n;
I >> n;
while(n--!=0){ I >> val; R.front(val); }return I; }
```

Практическое занятие № 4. Использование порождающих, структурных, поведенческих шаблонов.

Цель работы: ознакомиться с порождающими шаблонами

Теоретический материал

Порождающие шаблоны

Абстрактная фабрика — порождающий шаблон проектирования, позволяющий изменять поведение системы, варьируя создаваемыми объектами, при этом сохраняя интерфейсы. Он позволяет создавать целые группы взаимосвязанных объектов, которые, будучи созданными одной фабрикой, реализуют общее поведение. Шаблон реализуется

созданием абстрактного класса Factory, который представляет собой интерфейс для создания компонентов системы (например, для оконного интерфейса он может создавать окна и кнопки). Затем пишутся классы,реализующие этот интерфейс.

Этот шаблон предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов.

Фабричный метод (Factory Method)

Фабричный метод — порождающий шаблон проектирования, предоставляющий подклассам интерфейс для создания экземпляров некоторого класса. В момент создания наследники могут определить, какой класс создавать. Иными словами, Фабрика делегирует создание объектов наследникам

родительского класса. Это позволяет использовать в коде программы не специфические классы, а манипулировать абстрактными объектами на более высоком уровне.

Шаблон определяет интерфейс для создания объекта, но оставляет подклассам решение о том, какой класс инстанцировать. Фабричный метод позволяет классу делегировать создание подклассов. Используется, когда:

- классу заранее неизвестно, объекты каких подклассов ему нужно создавать.
- класс спроектирован так, чтобы объекты, которые он создаёт, специфицировалисьподклассами.
- класс делегирует свои обязанности одному из нескольких вспомогательных подклассов,и планируется локализовать знание о том, к9а5кой класс принимает эти обязанности на себя.

Структура:

Product — продукт; определяет интерфейс объектов, создаваемых абстрактным методом.

ConcreteProduct — конкретный продукт, реализует интерфейс Product.

Creator — создатель; объявляет фабричный метод, который возвращает объект типаProduct. Может также содержать реализацию этого метода «по умолчанию»; может вызывать фабричный метод для создания объекта типа Product.

ConcreteCreator — конкретный создатель; переопределяет фабричный метод таким образом, чтобы он создавал и возвращал объект класса ConcreteProduct.

Одиночка (Singleton)

Одиночка — порождающий шаблон проектирования, гарантирующий, что воднопоточном приложении будет единственный экземпляр класса с глобальной точкой доступа.

Существенно то, что можно пользоваться именно экземпляром класса, так как при этом во многих случаях становится доступной более широкая функциональность.

Глобальный «одинокий» объект — именно объект, а не набор процедур, не привязанных ни к какому объекту — бывает нужен:

- если используется существующая объектно-ориентированная библиотека;
- если есть шансы, что один объект когда-нибудь превратится в несколько;
- если интерфейс объекта (например, игрового мира) слишком сложен, и не стоит засорятьосновное пространство имён большим количеством функций;
- если, в зависимости от каких-нибудь условий и настроек, создаётся один из несколькихобъектов. Например, в зависимости от того, ведётся лог или нет, создаётся или настоящий объект,пишущий в файл, или «заглушка», ничего не делающая.

Поведенческие шаблоныСтратегия (Strategy)

Стратегия — поведенческий шаблон проектирования, предназначенный для определения семейства алгоритмов, инкапсуляции каждого из них и обеспечения их взаимозаменяемости. Это позволяет выбирать алгоритм путем определения

соответствующего класса. Шаблон Strategy позволяет менять выбранный алгоритм независимо от объектов-клиентов, которые егоиспользуют.

Проблема: по типу клиента (или по типу обрабатываемых данных) выбрать подходящий алгоритм, который следует применить. Если используется правило, которое не подвержено изменениям, нет необходимости обращаться к шаблону «стратегия».

Решение: отделение процедуры выбора алгоритма от его реализации. Это позволяет сделать выбор на основании контекста.

Класс Strategy определяет, как будут использоваться различные алгоритмы. Конкретные классы ConcreteStrategy реализуют эти различные алгоритмы. Класс Context использует конкретные классы ConcreteStrategy посредством ссылки на конкретный тип абстрактного класса Strategy. Классы Strategy и Context взаимодействуют с целью реализации выбранного алгоритма (в некоторых случаях классу Strategy требуется посылать запросы классу Context). Класс Context пересылает классу Strategy запрос, поступивший от его класса-клиента.

Наблюдатель (Observer)

Наблюдатель — поведенческий шаблон проектирования. Создает механизм у класса, который позволяет получать экземпляру объекта этого класса оповещения от других объектов обизменении их состояния, тем самым наблюдая за ними.

Определяет зависимость типа «один ко многим» между объектами таким образом, что приизменении состояния одного объекта все зависящие от него оповещаются об этом событии.

При реализации шаблона «наблюдатель» обычно используются следующие классы:

- Observable — интерфейс, определяющий методы для добавления, удаления иоповещения наблюдателей;
- Observer — интерфейс, с помощью которого наблюдатель получает оповещение;
- ConcreteObservable — конкретный класс, который реализует интерфейс Observable;
- ConcreteObserver — конкретный класс, который реализует интерфейс Observer.

Шаблон «наблюдатель» применяется в тех случаях, когда система обладает следующимисвойствами:

- существует, как минимум, один объект, рассылающий сообщения;
- имеется не менее одного получателя сообщений, причём их количество и состав могутизменяться во время работы приложения;
- нет надобности очень сильно связывать взаимодействующие объекты, что полезно дляповторного использования.

Данный шаблон часто применяют в ситуациях, в которых отправителя сообщений не интересует, что делают получатели с предоставленной им информацией.

Команда (Command)

Команда — поведенческий шаблон проектирования, используемый при объектно-ориентированном программировании, представляющий действие. Объект командызаключает в себе само действие и его параметры.

Паттерн обеспечивает обработку команды в виде объекта, что позволяет сохранять её, передавать в качестве параметра методам, а также возвращать её в виде результата, как и любойдругой объект.

Паттерн Command преобразовывает запрос на выполнение действия в отдельный объект-команду. Такая инкапсуляция позволяет передавать эти действия другим функциям и объектам в качестве параметра, приказывая им выполнить запрошенную

операцию. Команда – это объект, поэтому над ней допустимы любые операции, что и над объектом.

Интерфейс командного объекта определяется абстрактным базовым классом Command и в самом простом случае имеет единственный метод execute(). Производные классы определяют получателя

запроса (указатель на объект-получатель) и необходимую для выполнения операцию (метод этого объекта). Метод execute() подклассов Command просто вызывает нужную операциюполучателя.

Впаттерне Command может быть до трех участников:

- Клиент, создающий экземпляр командного объекта.
- Инициатор запроса, использующий командный объект.
- Получатель запроса.

Сначала клиент создает объект ConcreteCommand, конфигурируя его получателем запроса. Этот объект также доступен инициатору. Инициатор использует его при отправке запроса, вызывая метод execute(). Этот алгоритм напоминает работу функции обратного вызовав процедурном программировании

– функция регистрируется, чтобы быть вызванной позднее.

Паттерн Command отделяет объект, инициирующий операцию, от объекта, который знает,как ее выполнить. Единственное, что должен знать инициатор, это как отправить команду. Это придает системе гибкость: позволяет осуществлять динамическую замену команд, использоватьсложные составные команды, осуществлять отмену операций.

Структура – это объединение одного либо более объектов (переменных, массивов, указателей, других структур). Как и массив, она представляет собой совокупность данных, но отличается от него тем, что к ее элементам необходимо обращаться по имени, и ее различные элементы не обязательно должны принадлежать одному типу.

Структуры удобно использовать там, где разнообразные данные, относящиеся к одному итому же объекту, необходимо объединять. Например, ученика средней школы характеризуют следующие данные: фамилия, имя, дата рождения, класс, возраст.

Объявление структуры осуществляется с помощью ключевого слова struct, за которым следует ее тип, список элементов, заключенных в фигурные скобки. Ее можно представить в следующем общем виде:

struct тип {тип элемента 1 имя элемента 1;тип элемента n имя элемента n; };

Именем элемента может быть любой идентификатор. В одной строке можно записывать через запятую несколько идентификаторов одного типа.

Например:

```
struct date { int day;int month;
int year;} ;
```

Русские буквы использовать в идентификаторе в языке СИ нельзя.

Следом за фигурной скобкой, заканчивающей список элементов, могут записываться переменные данного типа, например:

struct date {…} a, b, c;

При этом выделяется соответствующая память.

Выведенное имя типа можно использовать для объявления записи, например: struct date day;. Теперь переменная day имеет тип date.

Разрешается вкладывать структуры одна на другую. Для лучшего восприятия структуры используем русские буквы в идентификаторах, в языке СИ этого делать нельзя.

Ход работы

1. Нарисовать в PlantUML диаграмму классов реализуемой программы. (проектирование)
2. Реализовать программу на Java. (реализация)

Для каждого из шаблонов, предложенных в вариантах можно найти пример реализации UMLсхемы и кода в приложенной книге “Паттерны проектирования”. Также указана глава, где подробно описан данный шаблон.

Шаблон “Стратегия”. Проект “Принтеры”. В проекте должны быть реализованы разные модели принтеров, которые выполняют разные виды печати.

Шаблон “Наблюдатель”. Проект “Оповещение постов ГАИ”. В проекте должна быть реализована отправка сообщений всем постам ГАИ.

Шаблон “Декоратор”. Проект “Универсальная электронная карта”. В проекте должна быть реализована универсальная электронная карта, в которой есть функции паспорта, страхового полиса, банковской карты и т. д.

Структурные шаблоны.

Задание 1: Написать программу, проанализировать результатstruct УЧЕНИК { char
Фамилия [15];
имя [15];
struct DATA ДАТА РОЖДЕНИЯ;
int класс, возраст;};

Определенный выше тип DATA включает три элемента: День, Месяц, Год, содержащие целые значения (int). Запись УЧЕНИК включает элементы: ФАМИЛИЯ [15]; ИМЯ[15]; ДАТА РОЖДЕНИЯ, КЛАСС, ВОЗРАСТ. ФАМИЛИЯ [15] и ИМЯ [15] – это символьные массивы из 15 компонент каждый. Переменная ДАТА РОЖДЕНИЯ представлена составным элементом (вложенной структурой) ДАТА. Каждой дате рождения соответствуют день месяца, месяц и год. Элементы КЛАСС и ВОЗРАСТ содержат значения целого типа (int). После введения типов ДАТА и УЧЕНИК можно объявить переменные, значения которых принадлежат этим типам.

Задание 2: Написать программу, проанализировать результат struct УЧЕНИК УЧЕНИКИ
[50];
массив УЧЕНИКИ состоит из 50 элементов типа УЧЕНИК.

В языке СИ разрешено использовать массивы структуры; записи могут состоять из
массивов и других записей.

Чтобы обратиться к отдельному компоненту структуры, необходимо указать ее имя, поставить точку и сразу за ней написать имя нужного элемента.

Задание 3: Написать программу, проанализировать результат Ученики [1]. КЛАСС = 3; Ученики [1]. ДАТА РОЖДЕНИЯ. ДЕНЬ=5;

Ученики [1]. ДАТА РОЖДЕНИЯ. МЕСЯЦ=4; Ученики [1]. ДАТА РОЖДЕНИЯ. ГОД=1979;

Первая строка указывает, что 1-й ученик учится в третьем классе, а последующие строки – его дату рождения: 5.04.79.

Каждый тип элемента структуры определяется соответствующей строкой объявления в фигурных скобках. Например, массив УЧЕНИКИ имеет тип УЧЕНИК, год является целым числом. Так как каждый элемент записи о9т8носится к определенному типу, его составное имя может появляться везде, где разрешено использовать значение этого типа. Рассмотрим пример программы:

```
/* Демонстрация записи */#include < stdio.h >
struct computer { int mem;int sp;
char model [20]; };
/* Объявление записи типа computer, состоящей из трех элементов: mem, sp, model */
struct computer pibm =
{512, 1, “ПЭВМЕС 1840.05”}
/* Объявление и инициализация переменной pibm типа computer */main ( )
{ printf (“ персональная ЭВМ % s\n\n “, pibm.model);
printf ( “объем оперативной памяти - % d К байт \n”, pibm.mem);
```

```
printf (“производительность - % d млн. операций в секунду \n”, pibm.sp);
/* вывод на экран значений элементов структуры */
}
```

В данной программе объявляется запись computer, которая состоит из трех элементов: mem (память ЭВМ), sp (быстродействие), model [20] (модель ПЭВМ). Переменная pibm имеет тип computer и является глобальной. Строки pibm.model, pibm.mem, pibm. sp в операторе printf вызывают обращение к соответствующим элементам записи pibm типа computer, которым ранее были присвоены определенные значения.

Результат работы программы имеет вид:

персональная ЭВМ ПЭВМ ЕС 1840.05 объем оперативной памяти – 512 К байт
производительность – 1 млн. операций в секунду

Рассмотрим использование в программе вложенных структур:

```
/* Демонстрация вложенных структур*/# include < stdio.h >
struct date { int day;int month;
int year; };
/* Объявление записи типа date*/struct person { char fam [20];
char im [20];
char ot [20]; struct date f1;};
/* Объявление структуры типа person; одним из элементов записи person является
запись
f1
типа date */main ( )
{ struct person ind1;
/* обьявление переменной ind1 типа person */
printf ( “Укажите фамилию, имя, отчество, день, \n месяц”“ и год рождения гражданина
ind1\n”);
scanf (“ % S % S % S %d %d”, &ind1.fam, &ind1.im, &ind1.ot,& ind1.f1.day,
&ind1.f1.month, &ind1.f1.year );
/* Ввод сведений о гражданине ind1 */
printf (“ Фамилия, имя, отчество: % S % S % S \n”, ind1.fam, ind1.im, ind1.ot);printf (“
Годрождения - % d \n”, ind1.f1.year);
printf (“ Месяцрождения - % d -й \n”, ind1.f1.month);printf (“ День рождения - % d -й \n”,
ind1.f1.day);
/* Вывод сведений о гражданине ind1 */
}
```

Структура типа date (дата) содержит три элемента: day (день), month (месяц), year (год). Структура типа person (человек) содержит четыре элемента: fam[20] (фамилия), im[20] (имя) , ot[20] (отчество), f1 (дата рождения). Последний из них (f1) – это вложенная запись типа date.

Результаты работы программы:

Укажите фамилию, имя, отчество, день, месяц и год рождения гражданина
ind1Алексеев Сергей Петрович 3.5.1978

Подчеркнутая информация вводится пользователем.Сведения о гражданине ind1
Фамилия, имя, отчество: Алексеев Сергей ПетровичГод рождения – 1978
Месяц рождения – 5-й
День рождения – 3-й

В следующей программе рассмотрим использование структуры в виде элементов массива pibm. Каждый элемент состоит из следующих компонентов: mem (память), sp (объем винчестера), model [20] (модель ПЭВМ):

```
/* Массивы записей */#include < stdio.h >
struct computer { int mem, sp;char model [20];
pibm [10];};
```

```
/* объявлениезаписиитипа computer; объявление массива pibm типа computer */main ( )
{ int i, j, k, priz;
for ( i=0; i<10; i++)
{ printf (“Введите сведения о ПЭВМ %d и признак (0-конец;
\n другая цифра- продолжение)\n”, i);printf (“ модельПЭВМ - ”);
scanf (“%S”, &pibm [i].model );
printf ( “объемоперативнойпамяти -”);scanf (“%d”, &pibm[i].mem);
printf (“ объемвинчестера - ”);scanf ( “%d , &pibm[i].sp ”); printf (“признак - ”);
scanf (“ %d ”, &priz );k=i;
if (!priz) break; }
/* Здесь !priz – операцияотрицания priz; break – выходизцикла for, если priz=0 */for (i=0;
i<10, i++);
{
printf ( “\n О какой ПЭВМ Вы хотите получить сведения?\n (Введите номер от 0 до
9)\n”
);
scanf ( “%d ”,&j );if (j>k)
{ printf (“Нет сведений об этой ПЭВМ \n”);continue; }
printf (“ персональнаяЭВМ %s\n ”, pibm[j].model);
printf (“объем оперативной памяти - % d Мб \n ”, pibm[j].mem);printf (“объем
винчестера - % d Мб \n ”, pibm[j].sp);
printf (“ признак – “ );
scanf ( “ %d ”, &priz);
```



```
if (!priz) break; }
```

/* Ввод сведений о ПЭВМ и занесение в массив pibm записей типа computer (первый цикл for); вывод на экран сведений о ПЭВМ (второй цикл for) */

}

Результаты работы программы:

Введите сведения о ПЭВМ и признак (0-конец; другая цифра – продолжение)модель ПЭВМ – AT 486 SX

объем оперативной памяти – 32объем винчестера – 4 Гбайта признак – 1

Введите сведения о ПЭВМ и признак (0-конец; другая цифра – продолжение)модель ПЭВМ – AT 386 DX

объем оперативной памяти – 64объем винчестера – 14 Гбайт признак – 0

О какой ПЭВМ Вы хотите получить сведения? (Введитн номер от 0 до 9)1

модель ПЭВМ – AT 386 DX

объем оперативной памяти – 16 Мбобъем винчестера – 2,5 Гбайт признак – 0

Практическое занятие № 5. Разработка приложения с использованием текстовых компонентов

Цель работы: изучить правила работы текстовыми компонентами

Теоретический материал

В языках С и С++ файл рассматривается как поток (stream), представляющий собой последовательность считываемых или записываемых байтов. При этом последовательность записи определяется самой программой.

С++ Builder позволяет работать с файлами тремя различными способами:

Работа с текстовыми компонентами

Каждый файл в программе на С++ должен быть связан с некоторым указателем. Этот указательимеет тип FILE (определен в stdio.h) и используется во всех операциях с файлами. Синтаксис операциисследующий:

include < stdio.h >FILE *fin, *fout;

Для работы с файлом его необходимо открыть функцией **fopen,** первый параметр которой содержит имя файла и путь к нему, а второй - режим открытия файла. Функция возвращает указатель на файл. Часто используемые режимы открытия файла:

r	Открывает файл для чтения
r+	Открывает существующий файл для чтения и записи
w	Создает файл для записи. Если файл уже существует, его содержимое уничтожается
w+	Создает файл для чтения и записи. Если файл уже существует, его содержимое уничтожается
a	Открывает файл для записи данных в конец файла. Если файл отсутствовал, он создается
a+	Открывает файл для чтения или записи данных в конец файла. Если файл отсутствовал, онсоздается

После режима может добавляться символ "t" - текстовый файл или "Ъ" - бинарный файл. Еслисимвол не указан, то по умолчанию считается что файл текстовый. Например:

```
fin=fopen("a:\\dat.txt","r");
fout=fopen("a:\\out.txt","w");
```

При записи обмен происходит не непосредственно с файлом, а с некоторым буфером. Информацияиз буфера переписывается в файл только при его переполнении или при закрытии файла.

После окончания работы с файлом он обязательно должен быть закрыт функцией **fclose(FILE** *). Если файл не удалось открыть, то возвращается нулевой указатель (NULL). Например:

```
# include < stdio.h >FILE *lw;
if (lw=fopen("a.text","r")) == NULL){
Memo1->Lines->Add("Файл не открыт"); return; }fclose(lw);
```

Работа с текстовыми файлами

Для записи в текстовый файл наиболее часто используется функция fprintf**:** int *fprintf(FILE *stream, const char *format[]);

где параметр format определяет строку форматирования аргументов**,** заданных своими адресами. Обычно эта строка состоит из последовательности символов "%", после которых следует символ типа данных:

I или i	Десятичное, восьмеричное или шестнадцатеричное целое
D или d	Десятичное целое
U или u	Десятичное целое без знака
E или e	Действительное с плавающей точкой
s	Строка символов
c	Символ

Из отрытого текстового файла можно читать информацию как по строкам. так и посимвольно.

Чтение строки осуществляется функцией fgets :char *fgets(char *st, int n, FILE *stream);

где st - указатель на буфер, в который считается строка; п - число читаемых символов; stream -указатель на файл. Строка читается до тех пор, пока не будет прочитано п-1 символов, или до конца строки

\n. В конце прочитанной строки ставится нулевой символ.

Для проверки достижения конца файла используется функция feof(F).

Чтение форматированных данных можно осуществлять с помощью функции fscanf:int *fscanf(FILE *stream, const char *format[]);

Строка форматирования строится аналогично fprintf.

Следует обратить внимание на то, что при чтении данных всегда указываются адреса переменных(&), а не сами переменные.

Пример записи и чтения данных из файла:# include < stdio.h >

Memo1->Clear(); FILE *lw;

// Запись данных в файл

if ((lw=fopen("a.text","wf)) == NULL) {

Memo1->Lines->Add("Файл не удалось создать");return; }

int num=10; char st[12] = "ИНФОРМАЦИЯ",sr[30]; fprintf(lw,"%s \n В группе %i человек",st, num); fclose(lw);

// Чтение данных из файла

if ((lw=fopen("a.text","rt")) == NULL) {

Memo1->Lines->Add("Файл не удалось открыть "); return;return; }

while (!feof(lw)) { fgets(sr,30,lw);

if (sr[strlen(sr)-1] =='\n') sr[strlen(sr)-1]=0; Memo1->Lines->Add(sr);

} fclose(lw);

В C++ определены три класса файлового ввода/вывода: **ifstream** - входные данные для чтения;

ofstream - выходные файлы для записи; **fstream** - файлы для чтения и записи.

Очень удобно применять следующие операции: поместить в поток (<<) и извлечь из потока (>>).

Пример программы:

#include <fstream.h>Memo1->Clear();

// Ввод данных в файлofstream lw ("a.text");

if (!lw) {Memo1->Lines->Add("Файл не удалось создать "); return;}int num=10; double k=5.67; char s[20] = "ИНФОРМАЦИЯ";

lw << num << " <<k << " << s << endl;lw.close();

// Чтение данных из файлаifstream lx ("a.text");

if (!lx){ Memo1->Lines->Add("Файл не удалось открыть ");return;}Ix » num » k » s;

Memo1->Lines->Add(IlntToStr(num));Memo1->Lines->Add(FloatToStr(k)); Memo1->Lines->Add(s); lx.close();

Помимо этих операции поместить в поток можно еще с помощью функций put и write.

Возможности ввода/вывода можно существенно расширить, используя манипуляторы потока.

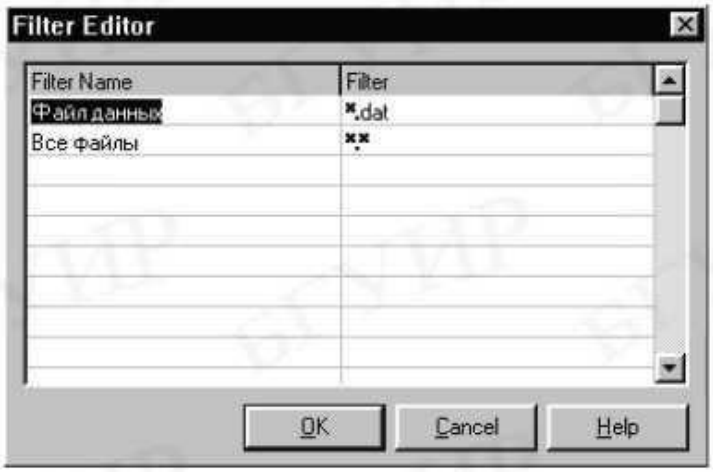
Компоненты TOpenDialog и TSaveDialog

Компоненты TOpenDialog и TSaveDialog находятся на странице DIALOGS. Все компоненты этой страницы являются невизуальными, т.е. не видны в момент работы программы. Поэтому их можно разместить в любом удобном месте формы. Оба рассматриваемых компонента имеют идентичные свойства и различаются только внешним видом. После вызова компонента появляется диалоговое окно, с помощью которого выбирается имя программы и путь к ней. В случае успешного завершения диалога имявыбранного файла и маршрут поиска содержатся в свойстве FileName. Для фильтрации файлов, отображаемых в окне просмотра, используется свойство Filter, а для

задания расширения файла, в случае, если оно не задано пользователем, - свойство DefaultExt. Если необходимо изменить заголовок диалогового окна, используется свойство Title.

Для установки компонентов TOpenDialog и TSaveDialog на форму необходимо на страницеDialogs меню компонентов щелкнуть мышью по пиктограммаме и поставить её в любое свободное место формы.

Установка фильтра производится следующим образом. Выбрав соответствующий компонент, дважды щелкнуть по правой части свойства Filter инспектора объектов. Появится окно Filter Editor, в левой части которого записывается текст, характеризующий соответствующий фильтр, а в правой части -маска. Для OpenDialogl установим значения маски, как показано на рис. 7.1. Формат *.dat означает, что будут видны все файлы с расширением dat, а формат *.* - что будут видны все файлы (с любым именеми с любым расширением).

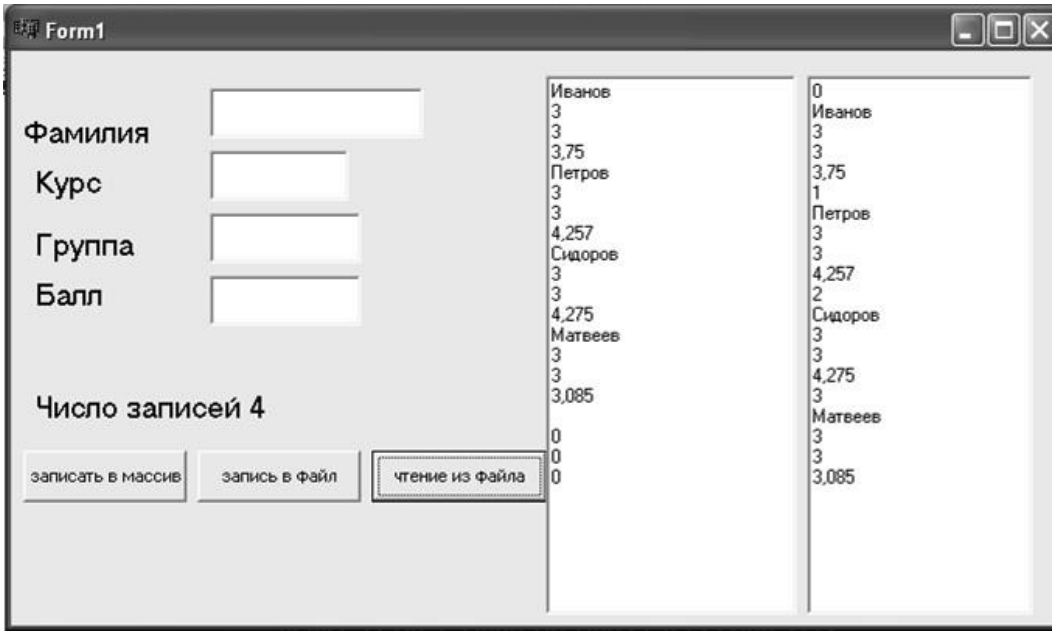


Для того чтобы файл автоматически записывался с расширением .dat, в свойстве DefaultExt запишем требуемое расширение - .dat.

Аналогичным образом настроим SaveDialogl для текстового файла (расширение .txt).

Ход работы

Составить программу, вводящую в файл или читающую из файла ведомость абитуриентов, сдавших вступительные экзамены. Каждая запись должна содержать фамилию, курс, группу и средний балл. Вывести список абитуриентов, записанный в файл и прочитанный из файла. Запись произвести в стиле C++ в текстовый файл.



Unit1.cpp

```
#define n 5                                  //ограничим число студентовstruct stud {
char fam[15];int kurs;
int grup; float ball;};
stud st[n],sz[n];int kol=0;
//---------------------------------------------------------------------------- void__fastcall
TForm1::Button1Click(TObject *Sender){ if(kol<n){
strcpy(st[kol].fam,Edit1->Text.c_str());st[kol].kurs=StrToInt(Edit2->Text);
st[kol].grup=StrToInt(Edit2->Text); st[kol].ball=StrToFloat(Edit4->Text); Edit1->Text="";
Edit2->Text="";
Edit3->Text=""; Edit4->Text="";
112
Label5->Caption="Число записей "+IntToStr(kol+1);kol++; }
else Button1->Enabled=false;
}
//---------------------------------------------------------------------------- void__fastcall
TForm1::Button3Click(TObject *Sender)
{int i=0; Memo1->Clear();ifstream in("z1.txt");
if(!in) {ShowMessage("Не удается открыть файл."); }while(!in.eof())
{
in>>sz[i].fam; in>>sz[i].kurs; in>>sz[i].grup; in>>sz[i].ball;if (in) {
Memo1->Lines->Add(IntToStr(i));Memo1->Lines->Add(sz[i].fam);
Memo1->Lines->Add(IntToStr(sz[i].kurs)); Memo1->Lines->Add(IntToStr(sz[i].grup));
Memo1->Lines->Add(FloatToStrF(sz[i].ball,ffGeneral,5,3)); i++; }
}
in.close();                    }
//---------------------------------------------------------------------------- void__fastcall
TForm1::Button2Click(TObject *Sender)
{int i;  AnsiString aaa;  Memo2->Clear();
//ofstream out("z1.txt"); ofstream out("z1.txt",ios::app);for (i=0; i<kol; i++){
out<<st[i].fam<<" "<<st[i].kurs<<" "<<st[i].grup
<<" "<<st[i].ball<<endl; }for (i=0; i<5; i++){
Memo2->Lines->Add(st[i].fam);
Memo2->Lines->Add(IntToStr(st[i].kurs)); Memo2->Lines->Add(IntToStr(st[i].grup));
```

```
Memo2->Lines->Add(FloatToStrF(st[i].ball,ffGeneral,5,3));
}
out.close();  }
//---------------------------------------------------------------------------
```

Практическое занятие № 6. Оптимизация и рефакторинг кода.

Цель работы: изучить технологию оптимизации программного кода, изучить технологию рефакторинга программного кода

Ход работы

Задание:

Модифицируйте 2 программы, реализованные на C++:

1 программа

#include <iostream> #include <locale.h>using namespace std;

int main()

{

int number; setlocale(LC_CTYPE,"Russian"); cout << "Введите число: "; cin >> number;cin.ignore();

cout << "Вы ввели: "<< number <<"\n"; cin.get();

}

Описание: пользователю предлагаеться ввести цифру, но если он введет например: b6, тоему выдаст - "Вы ввели: 5 (Только номер). Необходимо добиться, чтобы программа различала отрицательные и положительные значения, цифры и буквы, а также автоматически выдавала значение введенного числа в квадрате.

2 программа

#include <stdio.h> /* Стандартный заголовочный файл ввода-вывода */ #include <iostream>

/* Библиотека (стандарт) */

#include <locale.h> /* Русификатор */ #include <windows.h> /* Русификатор */ usingnamespace std;

+

int main(int argc, char* argv[])

{

setlocale(LC_CTYPE,"Russian");

double plus, minus, pow, div; // объявление переменных через запятую double a1; //отдельное объявление переменной a1

double a2; // отдельное объявление переменной a2 cout << "Введите первое число: "; cin >> a1;

cout << "Введите второе число: "; cin >> a2;

plus = a1 + a2; // операция сложения minus = a1 - a2; // операция вычитания pow = a1 * a2;

// операция умножения div = a1 / a2; // операция деления

cout << a1 << "+" << a2 << "=" << plus << endl; cout << a1 << "-" << a2 << "=" << minus <<endl; cout << a1 << "*" << a2 << "=" << pow << endl; cout << a1 << "/" << a2 << "=" << div << endl;system("pause");

return 0;

}

Описание: простой калькулятор, который может: добавлять, вычитать, умножать и делить.

Необходимо реализовать функцию возведения числа в любую степень, а также умножениеотрицательных чисел.

Задание:

−Решите задачу оптимизации кода (файл с кодом возьмите у преподавателя), используя только элементарные конструкции (последовательность, ветвления, циклы). Программа должнабыть рабочей.

−Оптимизировать программу (можно использовать процедуры или функции).

−Оптимизированная программа должна содержать проверки всех переменных, которыевводятся с клавиатуры.

−Для созданных программ оценить метрические характеристики по Холстеду;

−Сравнить полученные результаты. Оформить результаты в таблицу. Сделатьсоответствующие выводы.

Расчет метрики Холстеда:

Метрика Холстеда относится к метрикам, вычисляемым на

основании анализа числа строк и синтаксических элементов исходного кода программы.

Метрика Холстеда позволяет оценить размер (в словах) и объем в битах программы на стадии анализа требований. Используя нормы выработки операторов в день можно оценить время на разработку.

Основу метрики Холстеда составляют четыре измеряемые характеристики программы: n1

— число уникальных операторов

+

Опреаторы	Число операторов	Операнды	Число операндов
::	7	a	2
=	8	b	2
==	1	c	2
.	6	mAparam	5
!=	1	mBparam	5
<	1	mCparam	3
-	3	result.x1	3
/	3	result.x2	3
*	2	result.status	2
- binary	3	det	3
return	1	4	3
{	12	2	2
}	12		
(	13		
)	13		

программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов); n2 — число уникальных операндов программы (словарь операндов); N1 — общее число операторов в программе; N2 — общее число операндов в программе.

+, *, /, - это операторы

x, y, z, 999, -25, number1 - это операнды

На основании этих характеристик рассчитываются оценки:

Словарь программы (Halstead Program Vocabulary, HPVoc): n = n1 + n2;Длина программы (Halstead Program Length, HPLen): N = N1 +

N2;

Объем программы (Halstead Program Volume, HPVol): V = Nlog2 n;

Сложность программы (Halstead Difficulty, HDiff): D = (n1/2) × (N2 / n2);

На основе показателя HDiff предлагается оценивать усилия программиста при разработкепри помощи показателя HEff (Halstead Effort): H = D × V.

Задание 2:

В рамках лабораторной работы приведите примеры реализации рефакторинга в следующих системах:

−Visual Studio.
−Visual Assist X.
−Refactor.
−JustCode
−ReSharper
−CodeIt

Ход выполнения работы

1. Выполнить анализ программного кода разрабатываемого ПО и модульных тестов с цельювыявления плохо организованного кода.
2. Используя шаблоны рефакторинга, выполнить реорганизацию программного кодаразрабатываемого ПО.
3. Выполнить описание произведенных операций рефакторинга (было-стало-шаблонрефакторинга).
4. В случае необходимости скорректировать проектную документацию (диаграммы классов,последовательностей).
5. Сделать выводы по результатам выполнения работы.Класс Main

Было:

```
package game;
import game.Characters.*;
import game.Characters.Character; import game.Energetics.Energetic; import game.Energetics.Lightning;import game.Levels.Block;
import game.Levels.Level; import game.Levels.Level_data;import game.Weapon.Bullet;
import game.Weapon.Weapon;
import javafx.animation.AnimationTimer;import javafx.application.Application;
import  javafx.scene.Scene; import javafx.scene.image.Image;
import javafx.scene.input.KeyCode;import javafx.scene.layout.Pane; import javafx.stage.Stage;
import java.io.DataInputStream;import java.io.FileInputStream; import java.io.IOException;
import java.util.ArrayList; import java.util.HashMap;
public class Main extends Application {
public static ArrayList<Block> blocks = new ArrayList<>();public static ArrayList<Bullet> bullets = new ArrayList<>();
public static ArrayList<Bullet> enemyBullets = new ArrayList<>();public static ArrayList<EnemyBase> enemies = new ArrayList<>();static HashMap<KeyCode, Boolean> keys = new HashMap<>(); public static Stage stage;
public static Scene scene;
public static Pane gameRoot = new Pane();public static Pane appRoot = new Pane(); public static Menu menu;
public static Character booker;public static HUD hud;
public static Weapon weapon; public static Elizabeth elizabeth;
```

```
static VendingMachine vendingMachine;static Tutorial tutorial;
private static CutScenes cutScene;public static Energetic energetic; public static Lightning
lightning; public static int levelNumber; static Level level;
public static AnimationTimer timer = new AnimationTimer() {@Override
public void handle(long now) {update();
}
};
private static void update() {
for (EnemyBase enemy : enemies) {enemy.update();
if (enemy.getDelete()) { enemies.remove(enemy);break;
}
}
Bullet.update(); Controller.update();booker.update();
if (!energetic.getName().equals(""))energetic.update();
if (levelNumber > 0)elizabeth.update();
if (lightning != null) {lightning.update();
if (lightning.getDelete())
lightning = null;
}
menu.update(); hud.update(); weapon.update();
if (booker.getTranslateX() > Level_data.BLOCK_SIZE * 295)cutScene = new
CutScenes(levelNumber);
}
@Override
public void start(Stage primaryStage) throws Exception {stage = primaryStage;
scene = new Scene(appRoot, 1280, 720);appRoot.getChildren().add(gameRoot); level = new
Level();
try (DataInputStream dataInputStream = new DataInputStream(new
FileInputStream("C:/DeadShock/saves/data.dat"))) {
levelNumber = dataInputStream.readInt(); level.createLevels(levelNumber);
level.changeImageView(levelNumber); vendingMachine = new VendingMachine(); booker =
new Character(); booker.setMoney(dataInputStream.readInt());
booker.setSalt(dataInputStream.readByte()); booker.setCountLives(2);
weapon = new Weapon(); weapon.setWeaponClip(dataInputStream.readInt());
weapon.setBullets(dataInputStream.readInt());
hud = new HUD(); elizabeth = new Elizabeth();energetic = new Energetic();
} catch (IOException e) { levelNumber = 0; level.createLevels(levelNumber);
vendingMachine = new VendingMachine();booker = new Character();
weapon = new Weapon(); hud = new HUD(); energetic = new Energetic();
tutorial = new Tutorial(levelNumber);
}
switch (levelNumber) {case 0:
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 117, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 127, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 148, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 161, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 171, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 185, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 204, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 215, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 228, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 233, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 243, 200));
```

```
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 252, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 262, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 277, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 280, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 286, 200));
break; case 1:
enemies.add(new EnemyRedEye(Level_data.BLOCK_SIZE * 57, 200));
enemies.add(new EnemyRedEye(Level_data.BLOCK_SIZE * 67, 200));
enemies.add(new EnemyRedEye(Level_data.BLOCK_SIZE * 74, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 87, 200));
enemies.add(new EnemyRedEye(Level_data.BLOCK_SIZE * 104, 150));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 117, 200));
enemies.add(new EnemyRedEye(Level_data.BLOCK_SIZE * 133, 200));
enemies.add(new EnemyRedEye(Level_data.BLOCK_SIZE * 156, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 177, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 193, 200));
enemies.add(new EnemyRedEye(Level_data.BLOCK_SIZE * 201, 200));
enemies.add(new EnemyRedEye(Level_data.BLOCK_SIZE * 216, 200));
enemies.add(new EnemyRedEye(Level_data.BLOCK_SIZE * 224, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 246, 200));
enemies.add(new EnemyRedEye(Level_data.BLOCK_SIZE * 260, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 277, 100));
Level_data.enemyBlocks.add(new Block("invisible",Level_data.BLOCK_SIZE*34,
Level_data.BLOCK_SIZE * 13));
Level_data.enemyBlocks.add(new Block("invisible",       Level_data.BLOCK_SIZE    *
        36, Level_data.BLOCK_SIZE * 13));
Level_data.enemyBlocks.add(new    Block("invisible",     Level_data.BLOCK_SIZE    *
        60, Level_data.BLOCK_SIZE * 9));
Level_data.enemyBlocks.add(new    Block("invisible",     Level_data.BLOCK_SIZE    *
        61, Level_data.BLOCK_SIZE * 9));
Level_data.enemyBlocks.add(new    Block("invisible",     Level_data.BLOCK_SIZE    *
        106, Level_data.BLOCK_SIZE * 7));
Level_data.enemyBlocks.add(new    Block("invisible",     Level_data.BLOCK_SIZE    *
        107, Level_data.BLOCK_SIZE * 7));
Level_data.enemyBlocks.add(new    Block("invisible",     Level_data.BLOCK_SIZE    *
        168, Level_data.BLOCK_SIZE * 13));
Level_data.enemyBlocks.add(new    Block("invisible",     Level_data.BLOCK_SIZE    *
        170, Level_data.BLOCK_SIZE * 13));
Level_data.enemyBlocks.add(new    Block("invisible",     Level_data.BLOCK_SIZE    *
        196, Level_data.BLOCK_SIZE * 13));
Level_data.enemyBlocks.add(new    Block("invisible",     Level_data.BLOCK_SIZE    *
        197, Level_data.BLOCK_SIZE * 13));
Level_data.enemyBlocks.add(new    Block("invisible",     Level_data.BLOCK_SIZE    *
        232, Level_data.BLOCK_SIZE * 8));
Level_data.enemyBlocks.add(new    Block("invisible",     Level_data.BLOCK_SIZE    *
        233, Level_data.BLOCK_SIZE * 8));
break;
}
menu = new Menu(); appRoot.getChildren().add(menu.menuBox);
booker.translateXProperty().addListener( ((observable, oldValue, newValue) -> {int offset =
newValue.intValue();
if (offset > 600 && offset < gameRoot.getWidth() - 680) {gameRoot.setLayoutX( - (offset -
600) ); level.getBackground().setLayoutX((offset - 600) / 1.5);
```

```
}
if (offset <= 100) level.getBackground().setLayoutX(0);
}));
vendingMachine.createButtons();
stage.getIcons().add(new Image("file:/C:/DeadShock/images/icon.jpg"));
stage.setTitle("DeadShock");
stage.setResizable(false); stage.setWidth(scene.getWidth());
stage.setHeight(scene.getHeight());stage.setScene(scene); stage.show();
timer.start();
}
public static void main(String[] args) {launch(args);
}
}
```

Стало:

```
package game;
import game.Characters.*;
import game.Characters.Character; import game.Energetics.Energetic; import
game.Energetics.Lightning;import game.Levels.Block;
import game.Levels.Level; import game.Levels.Level_data;import game.Weapon.Bullet;
import game.Weapon.Weapon;
import javafx.animation.AnimationTimer;import javafx.application.Application; import
javafx.scene.Scene;
import javafx.scene.image.Image; import javafx.scene.input.KeyCode;import
javafx.scene.layout.Pane; import javafx.stage.Stage;
import java.io.DataInputStream;import java.io.FileInputStream; import java.io.IOException;
import java.util.ArrayList; import java.util.HashMap;
public class Main extends Application {
public static ArrayList<Block> blocks = new ArrayList<>();public static ArrayList<Bullet>
bullets = new ArrayList<>();
public static ArrayList<Bullet> enemyBullets = new ArrayList<>();public static
ArrayList<EnemyBase> enemies = new ArrayList<>();static HashMap<KeyCode, Boolean>
keys = new HashMap<>(); public static Stage stage;
public static Scene scene;
public static Pane gameRoot = new Pane();public static Pane appRoot = new Pane(); public
static Menu menu;
public static Character booker;public static HUD hud;
public static Weapon weapon; public static Elizabeth elizabeth;
static VendingMachine vendingMachine;static Tutorial tutorial;
private static CutScenes cutScene;
public static Energetic energetic;public static Lightning lightning;
public static int levelNumber;static Level level;
public static AnimationTimer timer = new AnimationTimer() {@Override
public void handle(long now) {update();
}
};
private void initContent() { appRoot.getChildren().add(gameRoot);level = new Level();
        try                (DataInputStream dataInputStream = new
                           DataInputStream(new
FileInputStream("C:/DeadShock/saves/data.dat"))) {
levelNumber = dataInputStream.readInt(); level.createLevels(levelNumber);
level.changeImageView(levelNumber); vendingMachine = new VendingMachine(); booker =
new Character(); booker.setMoney(dataInputStream.readInt());
booker.setSalt(dataInputStream.readByte()); booker.setCountLives(2);
```

```
weapon = new Weapon(); weapon.setWeaponClip(dataInputStream.readInt());
weapon.setBullets(dataInputStream.readInt());
hud = new HUD(); elizabeth = new Elizabeth();energetic = new Energetic();
} catch (IOException e) { levelNumber = 0; level.createLevels(levelNumber);
vendingMachine = new VendingMachine();booker = new Character();
weapon = new Weapon(); hud = new HUD(); energetic = new Energetic();
tutorial = new Tutorial(levelNumber);
}
createEnemies(); menu = new Menu();
appRoot.getChildren().add(menu.menuBox); booker.translateXProperty().addListener(
((observable, oldValue, newValue) -> {int offset = newValue.intValue();
if (offset > 600 && offset < gameRoot.getWidth() - 680) {gameRoot.setLayoutX( - (offset -
600) ); level.getBackground().setLayoutX((offset - 600) / 1.5);
}
if (offset <= 100) level.getBackground().setLayoutX(0);
}));
vendingMachine.createButtons();
}
public static void createEnemies() {switch (levelNumber) {
case 0:
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 117, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 127, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 148, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 161, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 171, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 185, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 204, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 215, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 228, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 233, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 243, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 252, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 262, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 277, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 280, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 286, 200));break;
case 1:
enemies.add(new EnemyRedEye(Level_data.BLOCK_SIZE * 57, 200));
enemies.add(new EnemyRedEye(Level_data.BLOCK_SIZE * 67, 200));
enemies.add(new EnemyRedEye(Level_data.BLOCK_SIZE * 74, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 87, 200));
enemies.add(new EnemyRedEye(Level_data.BLOCK_SIZE * 104, 150));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 117, 200));
enemies.add(new EnemyRedEye(Level_data.BLOCK_SIZE * 133, 200));
enemies.add(new EnemyRedEye(Level_data.BLOCK_SIZE * 156, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 177, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 193, 200));
enemies.add(new EnemyRedEye(Level_data.BLOCK_SIZE * 201, 200));
enemies.add(new EnemyRedEye(Level_data.BLOCK_SIZE * 216, 200));
enemies.add(new EnemyRedEye(Level_data.BLOCK_SIZE * 224, 200));
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 246, 200));
enemies.add(new EnemyRedEye(Level_data.BLOCK_SIZE * 260, 200));
```

```
enemies.add(new EnemyComstock(Level_data.BLOCK_SIZE * 277, 100));
Level_data.enemyBlocks.add(new Block("invisible", Level_data.BLOCK_SIZE *
        34,
Level_data.BLOCK_SIZE * 13));
Level_data.enemyBlocks.add(new Block("invisible", Level_data.BLOCK_SIZE *
        36, Level_data.BLOCK_SIZE * 13));
Level_data.enemyBlocks.add(new Block("invisible", Level_data.BLOCK_SIZE *
        60, Level_data.BLOCK_SIZE * 9));
Level_data.enemyBlocks.add(new Block("invisible", Level_data.BLOCK_SIZE *
        61, Level_data.BLOCK_SIZE * 9));
Level_data.enemyBlocks.add(new Block("invisible", Level_data.BLOCK_SIZE *
        106, Level_data.BLOCK_SIZE * 7));
Level_data.enemyBlocks.add(new Block("invisible", Level_data.BLOCK_SIZE *
        107, Level_data.BLOCK_SIZE * 7));
Level_data.enemyBlocks.add(new Block("invisible", Level_data.BLOCK_SIZE *
        168, Level_data.BLOCK_SIZE * 13));
Level_data.enemyBlocks.add(new Block("invisible", Level_data.BLOCK_SIZE *
        170, Level_data.BLOCK_SIZE * 13));
Level_data.enemyBlocks.add(new Block("invisible", Level_data.BLOCK_SIZE *
        196, Level_data.BLOCK_SIZE * 13));
Level_data.enemyBlocks.add(new Block("invisible", Level_data.BLOCK_SIZE *
        197, Level_data.BLOCK_SIZE * 13));
Level_data.enemyBlocks.add(new Block("invisible", Level_data.BLOCK_SIZE *
        232, Level_data.BLOCK_SIZE * 8));
+Level_data.enemyBlocks.add(new B14lo1ck("invisible", Level_data.BLOCK_SIZE *
        233, Level_data.BLOCK_SIZE * 8));
break;
}
}
private static void update() {
for (EnemyBase enemy : enemies) {enemy.update();
```

Практическое занятие № 7. Разработка интерфейса пользователя.

Цель работы: изучение принципов проектирования пользовательского интерфейса.

Разработка пользовательского интерфейса десктопного приложения.

Теоретический материал

Общие сведения о десктопных приложениях

Десктопные приложения - это программы, требующие наличия оператора (человека, работающего с программой), содержащие в себе всю полную функциональность и способные работать отдельно на любой машине изолированно от других приложений. Microsoft Word, Excel,Блокнот, однопользовательские игры - все это примеры десктопных приложений. Для их работы необходимы лишь достаточные аппаратные ресурсы компьютера, само приложение и набор библиотек, содержащих функции для работы с приложением.

Десктопные приложения могут быть также и многопользовательскими. Например, редактор файлов, который в зависимости от логина и пароля, введенных при запуске, будет давать доступ к различным файлам. И программа, и файлы находятся на одном компьютере, просто производится локальное разграничение доступа для разных пользователей.

Пользовательский интерфейс

Пользовательский интерфейс (UI) - это способ, которым вы выполняете какую-либо задачу с помощью какого-либо продукта, т.е. совершаемые вами действия и то, что вы получаетев ответ.

Программный интерфейс не только решает нашу проблему взаимодействия с приложением, но и делает это взаимодействие максимально комфортным. Нам важно наличие интерфейса, позволяющего при меньшем количестве усилий ознакомиться с возможностями приложения и понять принципы работы в нем.

Чтобы не возникло проблем при использовании какого-либо приложения, можно визуализировать его функциональные возможности в виде понятных элементов, и за этой визуализацией кроется целая кухня UX/UI-дизайна.

Грань между UX (**User Experience**) и UI (**User Interface**) очень тонка, но если разобраться,то становится ясно, что UX помогает понять пользователя. В UX-дизайне больше психологического аспекта, нежели технологического. UX изучает пользователя: как пользователь живет, что он думает, как и что делает, что его окружает. Перед дизайнером ставится задача - помочь обычному человеку легко разобраться с вашим программным продуктом и получить при этом удовлетворение от работы с ним.

А понять пользователя очень важно. Никому не захочется заполнить двадцать полей формы для регистрации на сайте или перещелкать штук пятнадцать вкладок, прежде чем добраться до нужной функции. «Пользователя не следует заставлять взаимодействовать с программой дольше, чем абсолютно необходимо для решения той или иной задачи» (из книги Алана Купера «Психбольница в руках пациентов»).

Этапы разработки пользовательского интерфейса

Полный цикл разработки интерфейса представлен на рис. 2.1.

Рис. 2.1. Этапы разработки пользовательского интерфейса

Для сокращения общего времени разработки определение стилистики начинается после пользовательских сценариев.

Исследование. На этапе исследования проводится сбор информации о продукте, клиенте, его конкурентах или близких аналогах, сбор статистики использования текущего интерфейса (например, сайта или мобильного приложения), анализ устройств предполагаемой целевой аудитории.

Если уже известно, кто будет воплощать интерфейс в жизнь (разработчики), то знакомимся с ними и выясняем их возможности и ограничения.

Этот этап помогает понять, для кого разрабатывается интерфейс, с какими ограничениями следует его делать (размеры экранов, интерактивность), как не стоит делать (например, быть непохожими на конкурентов).

Пользовательские сценарии. На основе предоставленного описания работы интерфейса создается список задач (пользовательских сценариев), которые может выполнять пользователь врамках интерфейса. Например, обновить аватарку в профиле.

Все задачи расписываются по шагам, которые необходимо предпринять для решения задачи. Например:

- зайти на сайт;
- авторизоваться;
- перейти в профиль;
- нажать на аватарку;
- выбрать файл;
- подтвердить или изменить кадрирование изображения;
- сохранить.

Составленные списки шагов для каждой задачи помогают понять, где путь для решения слишком долог относительно остальных задач. Этап пользовательских сценариев больше всего подходит для сокращения пути решения задач пользователей в рамках интерфейса.

Пример выше можно сократить на несколько шагов. Например, сделать сохранениеавтоматическим, а обрезание изображения - опциональным.

Структура интерфейса. Полученный список шагов на предыдущем этапе ложится в основу структуры интерфейса. Становится известно количество экранов, их краткое содержаниеи положение в общей структуре. На данном этапе строится карта экранов ***(User Flow Diagram).***

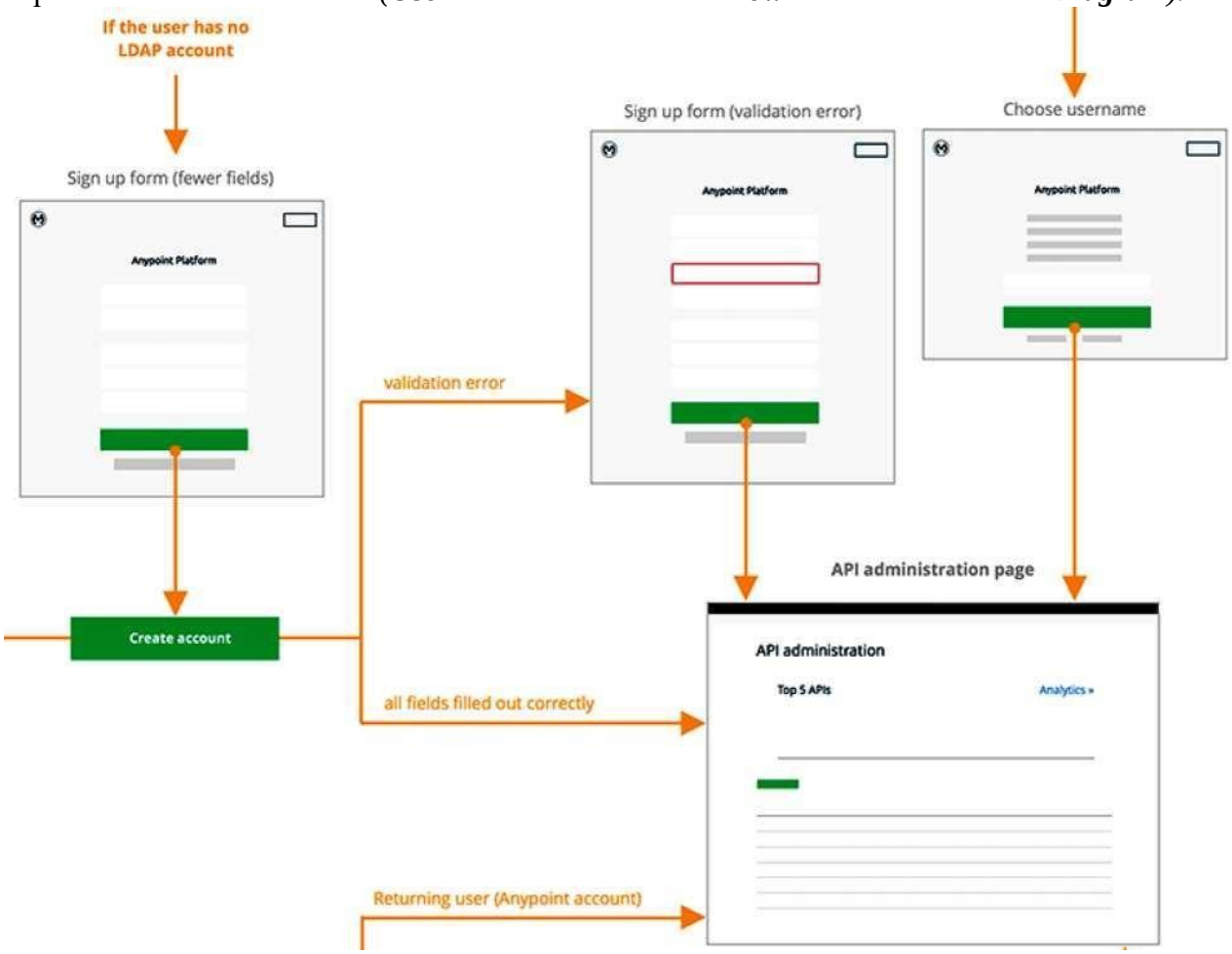


Рис. 2.2. Фрагмент карты экранов *(User Flow Diagram)*

Прототипирование интерфейса. В большинстве случаев реализуется два схематичных прототипа: черновой и финальный. Исключения составляют небольшие интерфейсы: простенькие мобильные приложения или маленькие сайты.

Черновой прототип представляет собой схематичные изображения экранов, связанные между собой. При черновом варианте на схемах изображены зоны и описания этих зон. Например, список новостей или шапка сайта. Все без деталей.

Черновой прототип помогает более наглядно понять, насколько объемным будет приложение, как много информации будет на каждом экране, как много нужно кликать, чтобы добраться до нужной страницы.

Следующим шагом идет финальный прототип, в котором схемы страниц все еще остаются связанными между собой, но на страницах уже видны все кнопки, тексты, чекбоксы, формы и прочие элементы. Пример чернового прототипа интернет-магазина приведен на рис. 2.3.

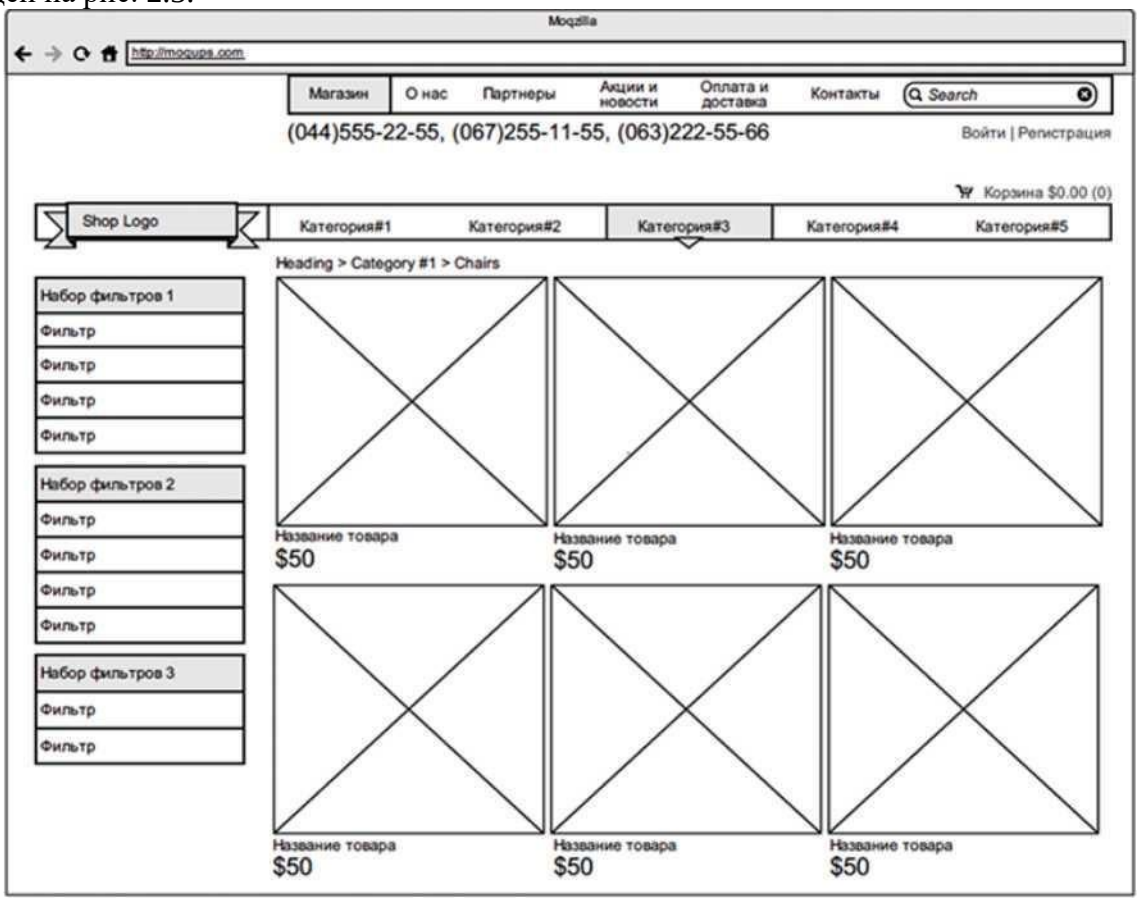


Рис. 2.3. Пример чернового прототипа интернет-магазина

В прототипах планируется функционал, расположение элементов страниц относительно друг друга, но никак не оформление. Цвета, изображения, иконки - это все этап оформления. На этапе проектирования невозможно сказать, как они будут взаимодействовать между собой, как будут смотреться вместе, будут ли перекрикивать друг друга.

Определение стилистики. После этапа исследования и параллельно с этапами проектирования идет определение будущей стилистики интерфейса.

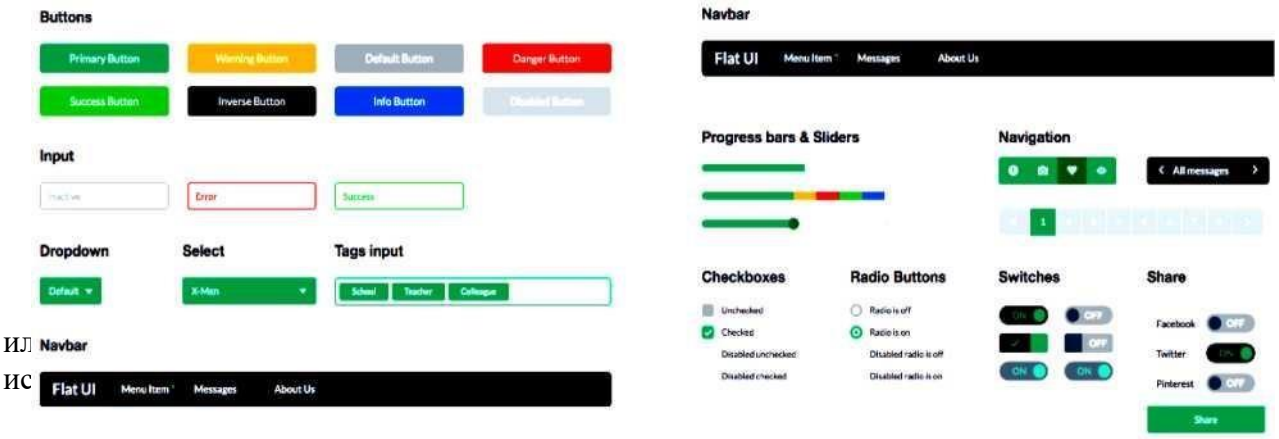
Для выбора стилистики готовятся несколько наборов изображений ***(moodboards)***. Эти наборы представлены страничками сайтов, иллюстрациями, кнопками, шрифтовыми композициями, связанными между собой стилистически.

Существует множество различных концепций, например: ***material design***, ***metro***, ***skeuomorphism*** и т.д. При выборе стиля интерфейса следует учесть текущие тенденции в

дизайне,адаптивность, время на разработку и внедрение дизайна, и много других не менее важных мо- ментов.

UI-кит - набор готовых решений пользовательского интерфейса. Это могут быть кнопки,поля ввода, «хлебные крошки», меню, переключатели, формы - все те элементы, что помогают пользователям взаимодействовать с сайтом или приложением.

Пример набора элементов из ***Flat UI-кита*** приведен на рис. 2.4.



ИЛ Navbar

ИС Flat UI Menu Item Messages About Us

вид всего приложения. Если предыдущий этап определения стилистики только дал направление, то дизайн- концепция призвана скрестить выбранное направление с имеющимся содержанием интерфейса. Дизайн-концепция может быть представлена любым объемом, но стараются его минимизировать для экономии времени. Обычно концепция представлена 1-3 экранами интерфейса. Если речь идет о сайте, то стараются показать вид одной и той же страницы для

нескольких устройств.

Для разработки дизайн-концепции используются online-инструменты:

- https://www.axure.com
- http://mockupbuilder.com
- https://www.fluidui.com
- и т.д.

Оформление всех экранов. После утверждения дизайн концепции настает время оформления всех остальных экранов интерфейса. Дизайн- концепция - это предположение о том,как может выглядеть весь интерфейс. Когда же очередь доходит до оформления всех экранов, тогда и происходит финализация внешнего вида: становится ясно, правильно ли подобраны кегльили интерлиньяж, хорошо ли сочетается толщина линий иконок с текстом, не конфликтует ли оформление форм (кнопок, полей ввода) с другими элементами экрана и многое другое.

Планом для оформления всех экранов являются структура и схематичный прототип интерфейса. Однако случаются отхождения от этого плана. Так при оформлении может выясниться, что всплывающее окно будет намного нагляднее и эффективнее, чем разъезжающийся блок информации посреди экрана.

Все оформленные экраны собираются в интерактивный прототип, который создаст максимально приближенный опыт использования интерфейса без прибегания к услугам разработчиков.

Анимация интерфейса. Часто этот этап начинается еще с момента дизайн-концепции и продолжается на протяжении оформления всех экранов.

Стараются показать только какие-либо нестандартные случаи анимации интерфейса, которые не предусмотрены операционной системой. Например, нету никакой надобности показывать, с какой скоростью будет выезжать следующий экран в интерфейсе приложения. Однако это тоже можно считать анимацией интерфейса.

Для ***Material design*** есть гайдлайны, которые наглядно объясняют, как надо анимироватьи как не надо.

Эти гайдлайны подходят для анимации интерфейсов любой платформы.

Подготовка материалов для разработчиков. На данном этапе уже присутствуют макетыинтерфейса во всех состояниях, прототип, связывающий весь интерфейс воедино и видеоролики,показывающие анимацию. Чтобы помочь разработчикам в реализации интерфейса, дизайнеры готовят все необходимые для этого материалы:

- спрайты;
- шрифт со всеми иконками;
- ***UI-Kit*** с повторяющимися элементами интерфейса и их состояниями.

Для иконок и прочей графики из интерфейса, для всех расстояний, отступов, размеров используют специальное программное обеспечение, например, Zeplin, которое самостоятельно готовит иконки и код.

Задание на лабораторную работу

Разработать пользовательский интерфейс для десктопного клиента, описанного в лабораторной работе № 1. Разработку пользовательского интерфейса производить согласно п. 2.настоящего пособия.

1. Произвести анализ аналогов, выявить их достоинства и недостатки, результаты анализотразить в отчете.
2. Разработать пользовательские сценарии и привести их часть в отчете.Разработать карту экранов в части описанных пользовательских сценариев.
3. Разработать черновой прототип экранов.
4. Подобрать подходящие стилистики для приложения не менее двух.
5. На основании одной из стилистик разработать дизайн концепцию приложения.

Вопросы для самопроверки

1. В чем отличие понятий UI и UX?
2. Какие этапы включает разработка пользовательского интерфейса?
3. Что такое UI-кит?
4. Для чего разрабатывают дизайн-концепцию?
5. В чем отличие чернового прототипа интерфейса от финального?

Практическое занятие №8. Создание приложения с БД. Создание запросов, хранимых процедур к БД

Цель работы: ознакомиться с основными конструкциями SQL, технологиями среды MS SQL Server Management, объектами SMO (среды MS Visual Studio).

Ход работы

Задание: Необходимо в среде SQL Server Management Studio:

Создать новую базу данных (далее БД); 2) создать в этой БД таблицы и связи для реализации всех функций предметной области МММ.

Для работы с базами данных MS SQL Server 2005 существует много возможностей - однаиз них — это инструмент SQL Server Management Studio (далее SSMS).

1. Запустите среду SQL Server Management Studio и подключитесь к локальному серверу MS SQL Server, используя технологию 1.
2. Если соединение прошло успешно, то откроется основное окно среды Management Studio. Рассмотрите подробно все компоненты среды.
3. Среда Management Studio представляет данные в виде окон, выделенных для отдельных типовданных. Сведения о базе данных отображаются в обозревателе объектов

и окнах документов.Найдите в открытом окне среды раздел «Обозреватель объектов», вид которого представленна рисунке 3.

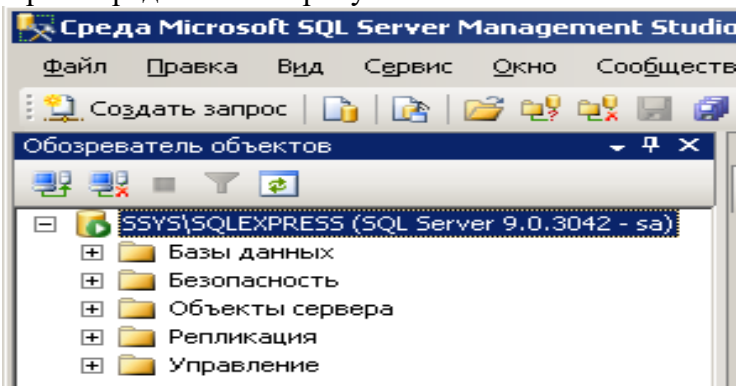


Рисунок 3 – Обозреватель объектов в среде SSMS

Обозреватель объектов является представлением в виде дерева, в котором отображаются все объекты базы данных на сервере. Обозреватель объектов включает сведения по всем серверам, к которым он подключен.

4. Далее найдите в открытом окне среды раздел «Окно документов». Окно документов представляет собой наиболее крупную часть среды Management Studio. В окнах документов могут размещаться редакторы запросов и окна обзора. По умолчанию отображается страница

«Подробности обозревателя объектов», подключенная к экземпляру компонента Database Engine на данном компьютере.

5. Найдите на панели инструментов среды кнопку «Создать запрос» и нажмите ее. Откроется окно создания запроса. С помощью данного окна можно создавать и выполнять запросы к БД. Наберите в основной области окна запроса select * from information_schema.tables и выполните запрос, нажав на кнопку «Выполнить». В нижней части окна запроса появится таблица с результатами, найдите ее. (Выполненный вами запрос извлекает информацию о существующих таблицах подключенной сейчас БД – это по умолчанию база данных Master итаблицы в ней системные).

6. Найдите в открытом окне среды раздел «Окно базы данных», вид которого представлен на рисунке 4.

Рисунок 4 – Окно базы данных в среде SSMS

Именно это поле со списком позволяет выбрать применяемую в данный момент базу данных для выполнения запросов. По умолчанию в данном окне выбрана БД для учетной записив которой зарегистрировался пользователь (для пользователя Sa это по умолчанию БД master)

Далее приступим к созданию своей бд, для этого:

7. В своей папке с проектом Лабы_ИСРКЭС создайте новую папку БД_МММ, в которой будетхраниться БД.

8. Самый простой способ создать базу данных — воспользоваться графическим интерфейсом SQL Server Management Studio. Создайте новую базу данных на Вашем локальном сервере MS SQL Server, используя технологию 2.

9. Убедитесь в обозревателе объектов в появившейся только что базе данных МММ.

10. Ознакомитесь с описанием предметной области в приложении 1.

11. Далее приступим к созданию таблиц. Существует несколько методов, рассмотрим один из них. В обозревателе объектов разверните узел БД МММ_ВашеФИО и выберите в списке ветвь Таблицы вызовите контекстное меню и выберите «Создать».

12. Создадим сначала таблицу Модель. Определим сначала все столбцы и

типы данных в окне создания таблицы. Вводите имена столбцов без пробелов, кириллицей, в нижнем регистре. Конструктор таблицы «Модель» изображен на рисунке 5.

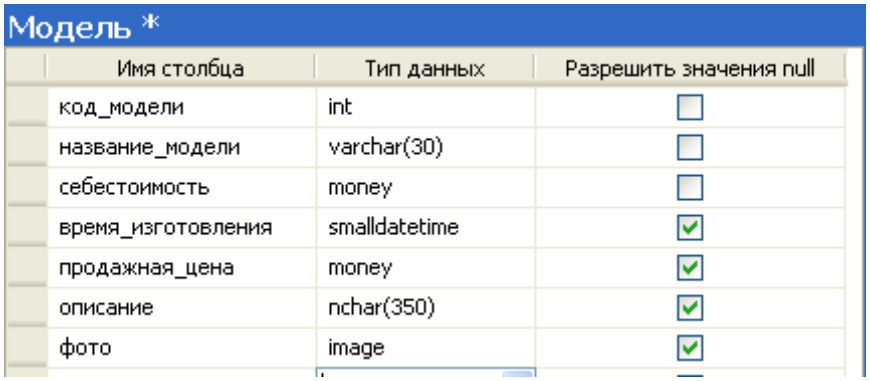


Рисунок 5 – Структура таблицы «Модель»

13. Сделайте столбец код_модели первичным ключом, используя технологию 3.

14. Закройте окно создания таблицы и сохраните, если еще этого не сделали, таблицу под именеммодель.

15. Сделайте столбец код_модели счетчиком, используя технологию 4.

16. Создайте аналогично остальные таблицы для раздела Заказ и Реализация продукции предметной области (таблицы Магазин, Заказ, Состав_заказа, Готовая_продукция). Внимательно определяйте типы данных.

17. Создайте связи между таблицами предметной области, используя технологию 5.

18. Заполните таблицу «Модель» данными, используя информацию из файла каталог (см. Сетевую папку. Местонахождение папки спросить у преподавателя.)

Задание: необходимо создать резервные копии базы данных «МММ» с использованием полного резервного копирования, разностного резервного копирования и резервного копирования журнала транзакций.

Указание: При выполнении работы используйте «Справочные материалы по SQL», которые расположены в дополнительном файле. (расположение файла спросить у преподавателя)

Ход работы:

1. Если необходимо, запустите SSMS, подключитесь к своему экземпляру SQL Server,используя технологию 1.

2. Откройте окно нового запроса. Измените контекст на базу данных МММ_вашеФИО,используя технологию 6.

3. Заполните таблицы Магазин, Заказ, Состав_заказа, Готовый продукт своими данными (неменее 5 строк в каждой таблице)

4. Наберите, исполните и сохраните тексты запросов для выполнения следующих функций ввашей БД (запросы создавайте с использованием языка SQL).

Извлечь все данные из таблицы Модель (запрос SELECT)

С помощью запроса добавить в таблицу Готовый_продукт одну запись с данными (запрос INSERT)

Серийный номер	Код модели	Дата производства
0076AA-Key	1	01.03.2009

С помощью запроса удалить из таблицы Модель запись о модели с кодом =2. (запрос DELETE)

Извлечь из таблицы Модель те названия моделей, чья цена >100 (запрос SELECT)

Посчитать с помощью запроса среднюю цену всех моделей (запрос SELECT)

Извлечь из таблиц Модель, Заказы, Магазины следующие данные – Заказанные названия моделей, количество моделей, названия магазинов (запрос SELECT – для

объединения таблиц)

Задания для самостоятельной работы:

Вывести названия магазинов, начинающихся с буквы 'M'(запрос SELECT, условие LIKE)

Подсчитать количество готовых продуктов для каждой модели (запрос SELECT с Group by, агрегатная функция COUNT)

Для каждого магазина посчитать среднюю стоимость всех заказов за все время сотрудничества. (запрос SELECT с Group by, агрегатная функция AVG)

МДК.01.02 Поддержка и тестирование программных модулей

Практическое занятие № 1. Тестирование «белым ящиком»

Цель работы: изучить метод тестирования «Белым ящиком»

Сегодня тестирование – это обязательная часть процесса разработки программного обеспечения (далее – ПО). Это связано с жесткими правилами конкуренции для компаний, производящих программные продукты (ПП).

Раньше таких компаний на рынке было мало и пользователи программных продуктов были продвинутыми и заменяли тестеров. Если в программе обнаруживались баги, то пользователь звонил или отправлял письмо в компанию, где ошибку исправляли и по почте отправляли дискетку со свежим релизом. Но начиная с 1990 года согласно статистики продажи персональных компьютеров с каждым годом удваивались. И появилась армия пользователей,которая не готова была что-то тестировать. Если что-то не устроило было проще обменять надругой софт, т.к. число компаний производящих ПО тоже увеличивалось с каждых готом. И упользователей появился выбор что покупать и чем пользоваться.

Таким образом, тестирование ушло внутрь компаний, и появилась профессия тестировщика.

Тестирование ПО – это проверка соответствия между реальным поведением программы и ее ожидаемым поведением на конечном наборе тестов, выбранном определенным образом. [IEEE Guide to Software Engineering Body of Knowledge, SWEBOK, 2004].

Все виды тестирования можно условно разделить на две большие группы: Статическое тестирование (static testing).

Динамическое тестирование (dynamic testing).

Статическое тестирование – это процесс анализа самой разработки программного обеспечения, т. е. тестирование без запуска программы.

К данной группе можно отнести анализ кода. Данный вид тестирования осуществляется в основном программистами. Проводят тестирование артефактов разработки программного обеспечения, таких как требования, дизайн или программный код, проводимое без исполненияэтих артефактов. Например, с помощью рецензирования или статического анализа.

Статический анализ кода (static code analysis) – это анализ исходного кода, производимый без его исполнения.

Динамическое тестирование – это тестовая деятельность, предусматривающая эксплуатацию (запуск) программного продукта.

Динамическое тестирование предполагает запуск программы, выполнение всех ее функциональных модулей и сравнение фактического ее поведения с ожидаемым.

Статическое тестирование позволяет обнаружить дефекты, которые являются результатом ошибки и привести к сбоям в программном обеспечении. Динамическое тестирование позволяет продемонстрировать непосредственно сбои в программном обеспечении.

Существует несколько признаков, по которым принято производить классификацию видов тестирования.

По знанию системы выделяют:

– тестирование «черного ящика» (black box testing);
– тестирование «белого ящика» (white box testing);
– тестирование «серого ящика» (grey box testing).

Метод белого ящика (white box testing, open box testing, clear box testing, glass box testing)

– у тестировщика есть доступ к внутренней структуре и коду приложения, а также есть достаточно знаний для понимания увиденного.

Разработка тестов методом белого ящика (white-box test design technique): Процедура разработки или выбора тестовых сценариев на основании анализа внутренней структуры компонента или системы.

Техники, основанные на структуре, или методе белого ящика

– тестирование операторов;

– тестирование альтернатив.

Альтернатива (decision): Точка программы, в которой управление имеет два или более альтернативных путей. Узел с двумя или более связями для разделения ветвей.

Тестирование условий альтернатив (decision condition testing): Разработка тестов методом белого ящика, при котором тестовые сценарии проектируются для исходов условий и результатов альтернатив.

Покрытие (coverage): Уровень, выражаемый в процентах, на который определенный элемент покрытия был проверен набором тестов.

Покрытие альтернатив (decision coverage): Процент результатов альтернативы, который был проверен набором тестов. Стопроцентное покрытие решений подразумеваетстопроцентное покрытие ветвей и стопроцентное покрытие операторов.

Покрытие кода (code coverage): Метод анализа, определяющий, какие части программного обеспечения были проверены (покрыты) набором тестов, а какие нет, например, покрытие операторов, покрытие альтернатив или покрытие условий. Еще выделяют серый ящик.

Задание 1. Разработать программу на Python.

Даны длины сторон треугольника, определить вид треугольника и его площадь.

Выполнить контроль вводимых чисел.

1. Разнасторонний треугольник
2. Равнобедренный треугольник
3. Равносторонний треугольникОграничения:

- три числа не могут быть определены как стороны треугольника;
- если хотя бы одно из них меньше или равно 0;
- сумма двух из них меньше третьего.

Задание 2. Подготовить набор тестовых вариантов для обнаружения ошибок впрограмме.

Результат оформить в следующем виде:

Таблица 1

A	B	C	Ожидаемый результат	Объект проверки
Значение	Значение	Значение	Что должн ополучится	Значения вводимых данных, либ оожидаемый результат
…	…	…	…	…

Задание 3. Разработать программу на Python.

Даны длины сторон треугольника, определить вид треугольника и его площадь.

Выполнить контроль вводимых чисел.

1. Остроугольный треугольник
2. Тупоугольный треугольник
3. Прямоугольный треугольникОграничения:

- три числа не могут быть определены как стороны треугольника;
- если хотя бы одно из них меньше или равно 0;

- сумма двух из них меньше третьего.

Подготовить набор тестовых вариантов для обнаружения ошибок в программе иоформить результат.

Задание 4. На основании проведенных тестов составьте рекомендации по исправлениюошибок, выявленных в ходе тестирования в виде отчета.

Пример:

1 тест. В ходе проведения первого теста было обнаружено, что при в ведении некорректных данных площадь все равно высчитывается.

Рекомендуется: в случае, если пользователь введет не корректные данные, следует выводить сообщение с просьбой исправить введенные значения. Добавить в программу проверку введенных значений на соответствие ограничения.

Практическое занятие № 2. Тестирование «черным ящиком»

Цель работы: изучить метод тестирования «Черным ящиком»

Сегодня тестирование – это обязательная часть процесса разработки программного обеспечения (далее – ПО). Это связано с жесткими правилами конкуренции для компаний, производящих программные продукты (ПП).

Раньше таких компаний на рынке было мало и пользователи программных продуктов были продвинутыми и заменяли тестеров. Если в программе обнаруживались баги, то пользователь звонил или отправлял письмо в компанию, где ошибку исправляли и по почте отправляли дискетку со свежим релизом. Но начиная с 1990 года согласно статистики продажи персональных компьютеров с каждым годом удваивались. И появилась армия пользователей,которая не готова была что-то тестировать. Если что-то не устроило было проще обменять надругой софт, т.к. число компаний производящих ПО тоже увеличивалось с каждых готом. И упользователей появился выбор что покупать и чем пользоваться.

Таким образом, тестирование ушло внутрь компаний, и появилась профессия тестировщика.

Рассмотрим определение, которое записано в SWEBOK.

Тестирование ПО – это проверка соответствия между реальным поведением программы и ее ожидаемым поведением на конечном наборе тестов, выбранном определенным образом. [IEEE Guide to Software Engineering Body of Knowledge, SWEBOK, 2004].

Все виды тестирования можно условно разделить на две большие группы: Статическое тестирование (static testing).

Динамическое тестирование (dynamic testing).

Статическое тестирование – это процесс анализа самой разработки программного обеспечения, т. е. тестирование без запуска программы.

К данной группе можно отнести анализ кода. Данный вид тестирования осуществляется в основном программистами. Проводят тестирование артефактов разработки программного обеспечения, таких как требования, дизайн или программный код, проводимое без исполненияэтих артефактов. Например, с помощью рецензирования или статического анализа.

Статический анализ кода (static code analysis) – это анализ исходного кода, производимый без его исполнения.

Динамическое тестирование – это тестовая деятельность, предусматривающая эксплуатацию (запуск) программного продукта.

Динамическое тестирование предполагает запуск программы, выполнение всех ее функциональных модулей и сравнение фактического ее поведения с ожидаемым.

Статическое тестирование позволяет обнаружить дефекты, которые являются результатом ошибки и привести к сбоям в программном обеспечении. Динамическое тестирование позволяет продемонстрировать непосредственно сбои в программном обеспечении.

Существует несколько признаков, по которым принято производить классификацию видов тестирования.

По знанию системы выделяют:

- тестирование «черного ящика» (black box testing);
- тестирование «белого ящика» (white box testing);
- тестирование «серого ящика» (grey box testing).

Метод чёрного ящика (black box testing, closed box testing) – у тестировщика либо нет доступа к внутренней структуре и коду приложения, либо недостаточно знаний для их понимания, либо он сознательно не обращается к ним в процессе тестирования.

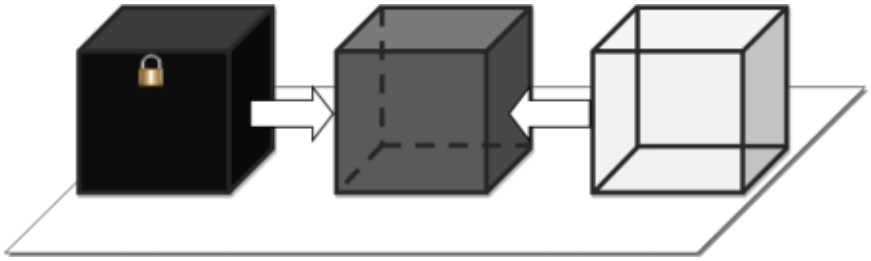


Рисунок 1

Разработка тестов методом черного ящика (black box test design technique)

Процедура создания и/или выбора тестовых сценариев, основанная на анализе функциональной или нефункциональной спецификации компонента или системы без знания внутренней структуры.

Техники разработки тестов на основе спецификаций, или методе черного ящика:

- эквивалентное разбиение;
- анализ граничных значений;
- тестирование таблицы решений;
- Эквивалентное разбиение (equivalence partitioning)

Разработка тестов методом черного ящика, в которой тестовые сценарии создаются для проверки элементов эквивалентной области. Как правило, тестовые сценарии разрабатываются для покрытия каждой области как минимум один раз.

Входные данные для программного обеспечения или системы разбиваются на группы, от которых ожидается сходное поведение, то есть они должны обрабатываться аналогичным образом. Эквивалентные области (или классы) могут быть определены как для валидных, таки для невалидных данных, то есть тех значений, которые должны отвергаться.

Эквивалентное разбиение применимо на всех уровнях тестирования. Эквивалентное разбиение может быть использовано с целью покрытия входных и выходных данных. Оно может применяться при ручном вводе данных, при передаче данных через интерфейсы в систему, или при проверке параметров интерфейсов в интеграционном тестировании.

Анализ граничных значений (boundary value analysis): Разработка тестов методом черного ящика, при котором тестовые сценарии проектируются на основании граничных значений.

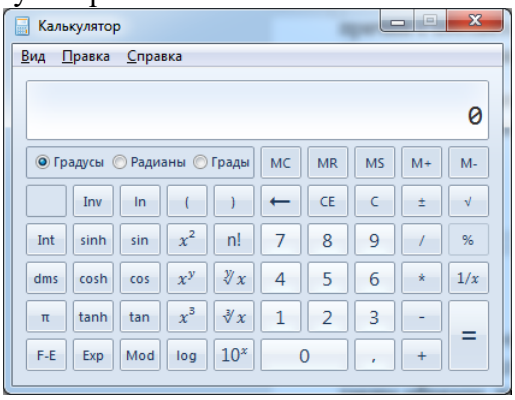
Граничное значение (boundary value): Входное значение или выходные данные, котороенаходится на грани эквивалентной области или на наименьшем расстоянии от обеих сторон грани, например, минимальное или максимальное значение области. Анализ граничных значений может применяться на всех уровнях тестирования

Таблица решений (decision table): Таблица, отражающая комбинации входных данных и/или причин с соответствующими выходными данными и/или действиям (следствиям), которая может быть использована для проектирования тестовых сценариев.

Таблицы решений – хороший метод для сбора системных требований, содержащих логические условия и документирования внутреннего дизайна системы. Они могут использоваться для записи сложных бизнес-правил, которые должна реализовывать система. Анализируются спецификации и определяются условия и действия системы. Входные условия и действия чаще всего формулируются таким образом, чтобы они могли принимать логические значения «истина» или «ложь».

Сильной стороной тестирования таблицы решений является то, что она создает комбинации условий, которые могли бы быть не проверены в ходе тестирования иным способом. Этот метод может быть применен ко всем ситуациям, в которых действие программного продукта зависит от нескольких логических альтернатив.

Задание 1. Написать калькулятор с небольшими багами.



Задание 2. Обменяться программой с другими студентами. Провести тестирование и написать отчет в тетради.

Таблица 2

Название теста	Описание сценария	Входные данные	Выходные данные	Удачное/неудачное тестирование	Предложения по исправлению найденных ошибок.	Пожелания пользователей
Функция суммы	Сложение двух положительных чисел; Проверка результата	Первая переменная=3 Вторая переменная=8	Результат=11	Неудачное	-	Поле для ввода значений и вывода, объединить

Практическое занятие № 3. Модульное тестирование

Цель работы: изучить возможность создания автоматических тестов, для модульноготестирования.

Сегодня тестирование – это обязательная часть процесса разработки программного обеспечения (далее – ПО). Это связано с жесткими правилами конкуренции для компаний, производящих программные продукты (ПП).

Раньше таких компаний на рынке было мало и пользователи программных продуктов были продвинутыми и заменяли тестеров. Если в программе обнаруживались баги, то пользователь звонил или отправлял письмо в компанию, где ошибку исправляли и по почте отправляли дискетку со свежим релизом. Но начиная с 1990 года согласно статистики продажи персональных компьютеров с каждым годом удваивались. И появилась армия пользователей,которая не готова была что-то тестировать. Если что-то не устроило было проще обменять надругой софт, т.к. число компаний производящих ПО

тоже увеличивалось с каждых готом. И упользователей появился выбор что покупать и чем пользоваться.

Таким образом, тестирование ушло внутрь компаний, и появилась профессия тестировщика.

Рассмотрим определение, которое записано в SWEBOK.

Тестирование ПО – это проверка соответствия между реальным поведением программы и ее ожидаемым поведением на конечном наборе тестов, выбранном определенным образом. [IEEE Guide to Software Engineering Body of Knowledge, SWEBOK, 2004].

Все виды тестирования можно условно разделить на две большие группы: Статическое тестирование (static testing).

Динамическое тестирование (dynamic testing).

Статическое тестирование – это процесс анализа самой разработки программного обеспечения, т. е. тестирование без запуска программы.

К данной группе можно отнести анализ кода. Данный вид тестирования осуществляется в основном программистами. Проводят тестирование артефактов разработки программного обеспечения, таких как требования, дизайн или программный код, проводимое без исполненияэтих артефактов. Например, с помощью рецензирования или статического анализа.

Статический анализ кода (static code analysis) – это анализ исходного кода, производимый без его исполнения.

Динамическое тестирование – это тестовая деятельность, предусматривающая эксплуатацию (запуск) программного продукта.

Динамическое тестирование предполагает запуск программы, выполнение всех ее функциональных модулей и сравнение фактического ее поведения с ожидаемым.

Статическое тестирование позволяет обнаружить дефекты, которые являются результатом ошибки и привести к сбоям в программном обеспечении. Динамическое тестирование позволяет продемонстрировать непосредственно сбои в программном обеспечении.

Существует несколько признаков, по которым принято производить классификацию видов тестирования.

По знанию системы выделяют:

– тестирование «черного ящика» (black box testing);
– тестирование «белого ящика» (white box testing);
– тестирование «серого ящика» (grey box testing).

Модульное тестирование, или юнит-тестирование (англ. unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода.Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Цель модульного тестирования — изолировать отдельные части программы и показать,что по отдельности эти части работоспособны.

Задание 1. Создание проекта программы, модули которого будут тестироваться.

Разработаем проект содержащий класс, который вычисляет площадь прямоугольника подлине двух его сторон.

Создадим в Visual Studio новый проект Visual C# -> Библиотека классов. Назовёмего MathTaskClassLibrary.

Class1 переименуем в Geometry.

В классе реализуем метод, вычисляющий площадь прямоугольника. Для демонстрацииостановимся на работе с целыми числами. Код программы приведён ниже.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MathTaskClassLibrary
{
    public class Geometry
    {
        public int RectangleArea(int a, int b)
        {
            return a * b;
        }
    }
}
```

Рисунок 2

Создание проекта для модульного тестирования в Visual Studio.

Чтобы выполнить unit-тестирование, необходимо в рамках того же самого решениясоздать ещё один проект соответствующего типа.

Правой кнопкой щёлкните по решению, выберите “Добавить” и затем “Создать проект…”.

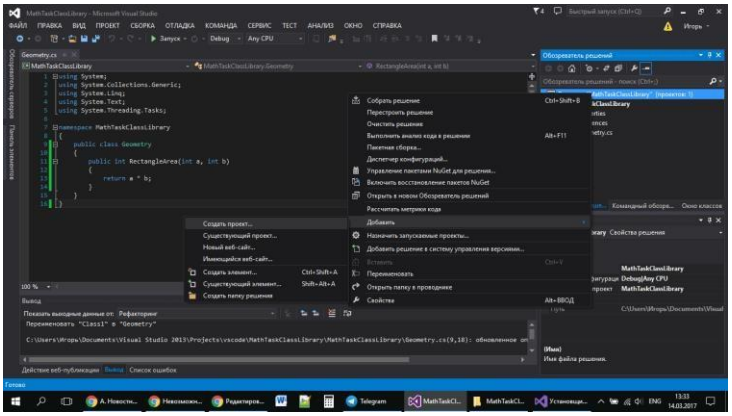


Рисунок 3

В открывшемся окне в группе Visual C# щёлкните “Тест”, а затем выберите “Проект модульного теста”. Введите имя проекта MathTaskClassLibraryTests и нажмите “ОК”. Таким образом проект будет создан.

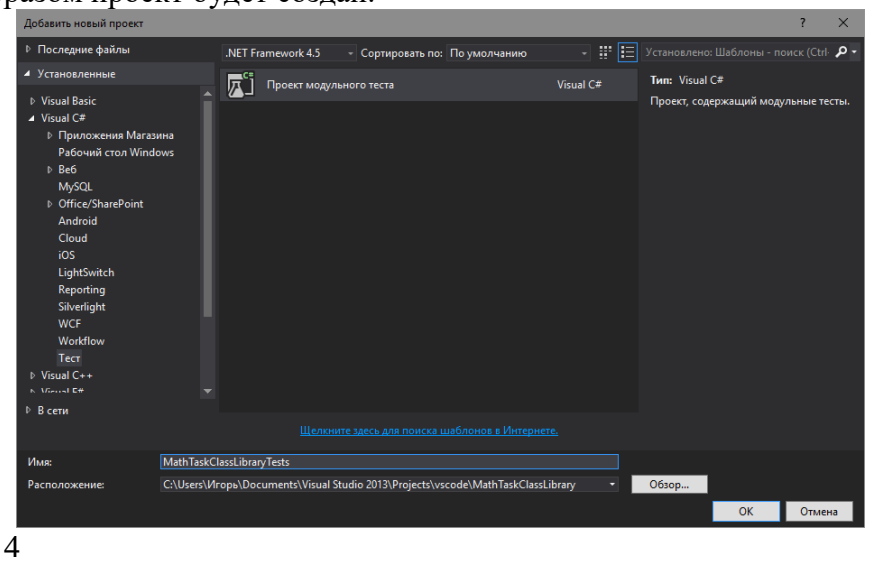


Рисунок 4

Перед Вами появится следующий код:

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace MathTaskClassLibraryTests
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void TestMethod1()
        {
        }
    }
}
```

Рисунок 5

Директива [TestMethod] обозначает, что далее идёт метод, содержащий модульный (unit)тест. А [TestClass] в свою очередь говорит о том, что далее идёт класс, содержащий методы, вкоторых присутствуют unit-тесты.

В соответствии с принятыми соглашениями переименуем класс UnitTest1в GeometryTests.

Затем в References проекта необходимо добавить ссылку на проект, код которого будемтестировать. Правой кнопкой щёлкаем на References, а затем выбираем “Добавить ссылку…”.

В появившемся окне раскрываем группу “Решение”, выбираем “Проекты” и ставим галочку напротив проекта MathTaskClassLibrary. Затем жмём “ОК”.

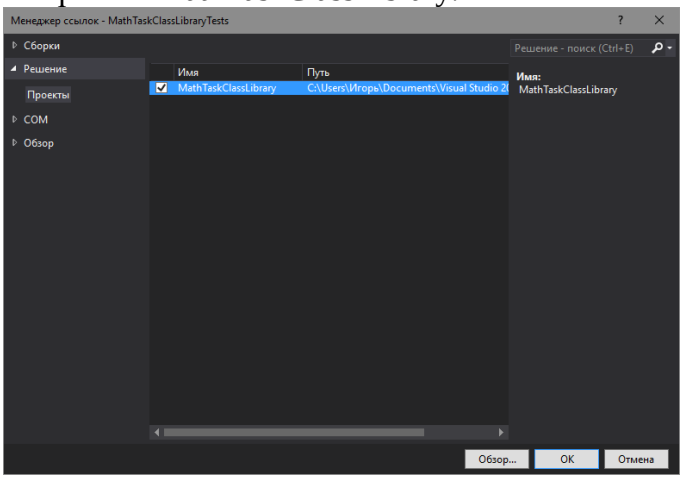


Рисунок 6

Также в коде необходимо подключить с помощью директивы using следующее пространство имён: using MathTaskClassLibrary;

Займёмся написание теста. Проверим правильно ли вычисляет программа площадь прямоугольника со сторонами 3 и 5. Ожидаемый результат (правильное решение) в данном случае это число 15.

Переименуем метод TestMethod1() в RectangleArea_3and5_15returned(). Новое название метода поясняет, что будет проверяться (RectangleArea – площадь прямоугольника) для какихзначений (3 и 5) и что ожидается в качестве правильного результата (15 returned).

Тестирующий метод обычно содержит три необходимых компонента:

1. исходные данные: входные значения и ожидаемый результат;
2. код, вычисляющий значение с помощью тестируемого метода;
3. код, сравнивающий ожидаемый результат с полученным.

Соответственно тестирующий код будет таким:

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using MathTaskClassLibrary;

namespace MathTaskClassLibraryTests
{
    [TestClass]
    public class GeometryTests
    {
        [TestMethod]
        public void RectangleArea_3and5_15returned()
        {
            // исходные данные
            int a = 3;
            int b = 5;
            int expected = 15;

            // получение значения с помощью тестируемого метода
            Geometry g = new Geometry();
            int actual = g.RectangleArea(a, b);

            // сравнение ожидаемого результата с полученным
            Assert.AreEqual(expected, actual);
        }
    }
}
```

Рисунок 7

Для сравнения ожидаемого результата с полученным используется метод AreEqual класса Assert. Данный класс всегда используется при написании unit тестов вVisual Studio.

Теперь, чтобы просмотреть все тесты, доступные для выполнения, необходимо открытьокно “Обозреватель тестов”. Для этого в меню Visual Studio щёлкните на кнопку “ТЕСТ”, выберите “Окна”, а затем нажмите на пункт “Обозреватель тестов”.

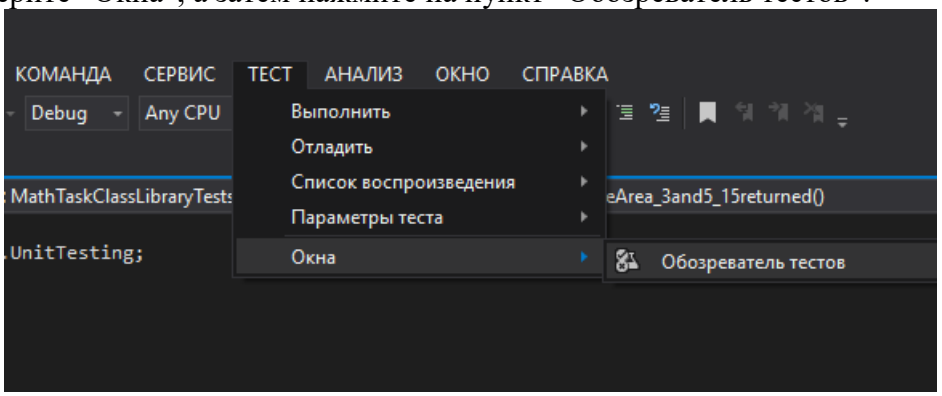


Рисунок 8

В студии появится следующее окно:

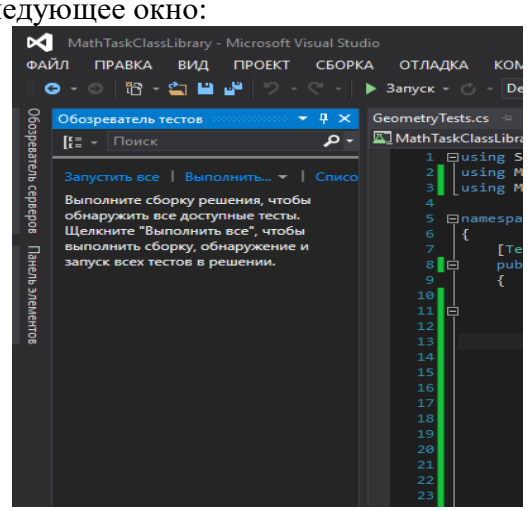


Рисунок 9

В данный момент список тестов пуст, поскольку решение ещё ни разу не было собрано.Выполним сборку нажатием клавиш Ctrl + Shift + B. После её завершения в “Обозревателе тестов” появится наш тест.

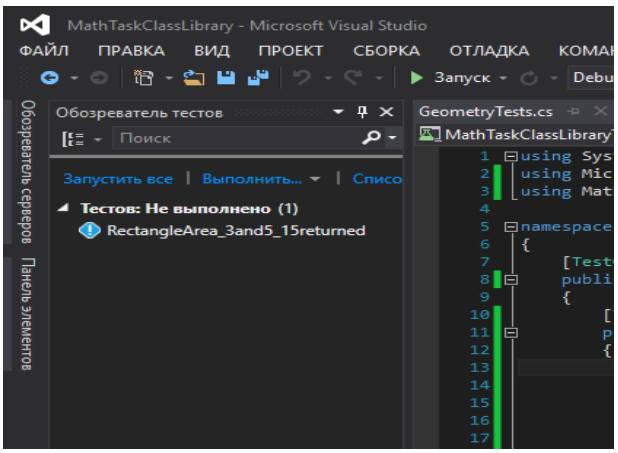


Рисунок 10

Синяя табличка с восклицательным знаком означает, что указанный тест никогда невыполнялся. Выполним его.

Для этого нажмём правой кнопкой мыши на его имени и выберем “Выполнить выбранные тесты”.

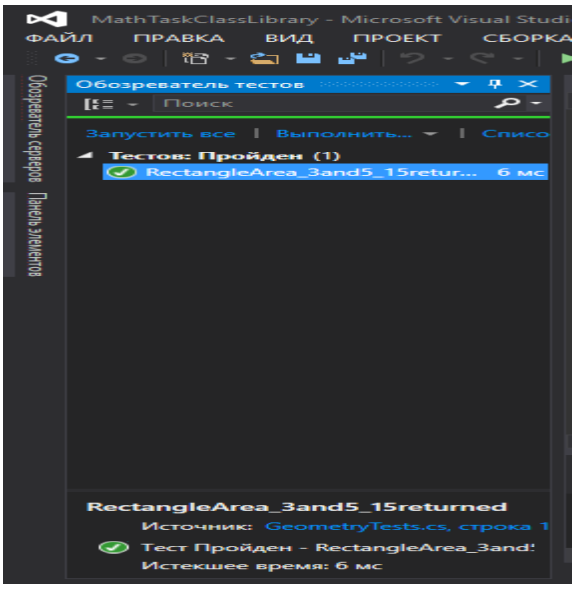


Рисунок 11

Зелёный кружок с галочкой означает, что модульный тест успешно пройден: ожидаемыйи полученный результаты равны.

Изменим код метода RectangleArea, вычисляющего площадь прямоугольника, чтобы сымитировать провал теста и посмотреть, как поведёт себя Visual Studio. Прибавим к возвращаемому значению 10.

Запустим unit-тест.

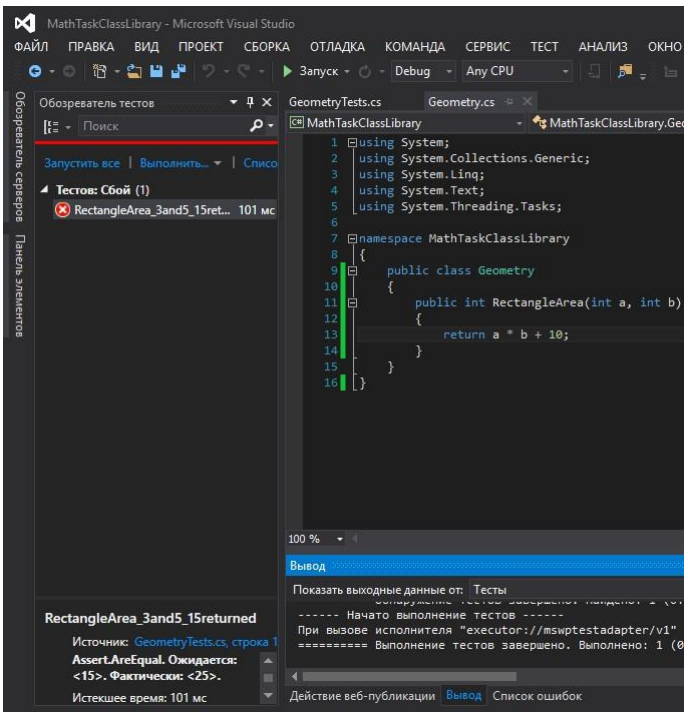


Рисунок 12

Как Вы видите, красный круг с крестиком показывает провал модульного теста, а нижеуказано, что при проверке ожидалось значение 15, а по факту оно равно 25.

Задание 2. Разработать программу для подсчета объема цилиндра и создать модульный

тест.

Практическое занятие № 4. Интеграционное тестирование

Цель работы: Овладение навыками интеграционного тестирования. Общие сведения

Интеграционное тестирование называют еще тестированием архитектуры системы. С одной стороны, это название обусловлено тем, что интеграционные тесты включают в себя проверки всех возможных видов взаимодействий между программными модулями и элементами, которые определяются в архитектуре системы – таким образом, интеграционныетесты проверяют полноту взаимодействий в тестируемой реализации системы. С другой стороны, результаты выполнения интеграционных тестов - один из основных источников информации для процесса улучшения и уточнения архитектуры системы, межмодульных и межкомпонентных интерфейсов. Т.е., с этой точки зрения, интеграционные тесты проверяют корректность взаимодействия компонент системы. В результате проведения интеграционноготестирования и устранения всех выявленных дефектов получается согласованная и целостнаяархитектура программной системы, т.е. можно считать, что интеграционное тестирование - это тестирование архитектуры и низкоуровневых функциональных требований. Интеграционное тестирование, как правило, представляет собой итеративный процесс, при котором проверяется совокупность модулей, возрастающая от итерации к итерации. В интеграционном тестирование выделяют три методы выполнения: восходящее тестирование;монолитное тестирование; нисходящее тестирование.

Задание: Согласно варианту провести один из методов интеграционного тестирования

Практическое занятие № 5. Оформление документации на программные средства с использованием инструментальных средств.

Цель работы: Разработать комплект документации на программное средство Программное документирование – это процесс записи информации, произведенной процессами жизненного цикла.

Процесс содержит набор действий, которые планируют, проектируют, разрабатывают, производят, редактируют, распространяют и сопровождают те документы, в которых нуждаются все заинтересованные лица, такие как менеджеры, инженеры и пользователи программного средства.

Общие требования к составу и содержанию документов, поддерживающих создание программных средств, представлены в ряде стандартов разного ранга. Состав документов широко варьируется в зависимости от класса и характеристик объекта разработки.

Существует несколько стандартов в области обеспечения документирования программных средств.

ISO 12207 - Информационные технологии. Процессы жизненного цикла программного обеспечения.

В этом стандарте документированию посвящен специальный раздел в группе вспомогательных процессов.

ISO 9000- 3 - Общее руководство качеством и стандарты по обеспечению качества.

Управлению качеством документации посвящен специальный раздел 6.2. Эти задачи отражены также в ряде разделов стандарта, непосредственно регламентирующих управлениекачеством сложных программных средств.

ISO 6592 – Обработка информации. Руководство по разработке документации для вычислительных систем

Главная цель этого стандарта состоит в установлении базисной структуры документации, на основе которой возможно для любого проекта обеспечить эффективное совершенствование и реализацию информационной системы, ПС или БД. В стандарте установлены руководящие принципы создания документов для информационных систем.

ISO 9294 - Информационные технологии. Руководящие положения по управлению документацией на программное обеспечение

Технический отчет этого стандарта представляет руководство по документированию ПС для руководителей, отвечающих за создание программной продукции. Руководство предназначено для помощи в управлении разработкой и эффективном документировании программных проектов.

IEEE 1063-1993 - Пользовательская документация на программное обеспечение.

+В нем представлены наиболее полно общие требования к пользовательской документации на программные средства широкого применения. Стандарт определяет минимальные требования к структуре и содержанию комплекта документов для пользователей программных продуктов.

ГОСТ 34.602—89 Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системыНастоящий стандарт распространяется на автоматизированные системы для автоматизации различных видов деятельности (управление, проектирование, исследование и т. п.), включая их сочетания, и устанавливает состав, содержание, правила оформления документа “Техническое задание на создание (развитие или модернизацию) системы”.

ГОСТ Р 51904-2002 — Программное обеспечение встроенных систем. Общие требования к разработке и документированию.

Стандарт распространяется на процессы разработки и документирования программного обеспечения встроенных систем реального времени. Стандарт распространяется на все действия, имеющие отношение к разработке программного обеспечения.

ГОСТ 19.101 – Виды программ и программных документов.

Настоящий стандарт устанавливает виды программ и программных документов для вычислительных машин, комплексов и систем независимо от их назначения и области применения. Виды программных документов и их содержание приведены в таблице 1:

Таблица 19

Вид программного документа	Содержание программного документа
1	2
Спецификация	Состав программы и документации на нее
Ведомость держателей подлинников	Перечень предприятий, на которых хранят подлинники программных документов
Текст программы	Запись программы с необходимыми комментариями
Описание программы	Сведения о логической структуре и функционировании программы
Программа и методика испытаний	Требования, подлежащие проверке при испытании программы, а также порядок и методы их контроля
Техническое задание	Назначение и область применения программы,технические, технико-экономические и специальные требования, предъявляемые к программе, необходимые стадии и сроки разработки, виды испытаний
Пояснительная записка	Схема алгоритма, общее описание алгоритма и (или) функционирования программы, а также обоснование принятых технических и технико-экономических решений
Эксплуатационные документы	Сведения для обеспечения функционирования и эксплуатации программы

Порядок выполнения работы:

Оформить пояснительную записку (ПЗ) на программный продукт. ПЗ на программное средство должна иметь следующую структуру:

1) Постановка задачи;
2) Входные и выходные данные;
3) Среда разработки и обоснование выбора языка программирования;
4) Описание алгоритма;
5) Описание используемых классов и методов;
6) Заключение.

Приложение А – Техническое задание;Приложение Б – Технический проект;

Приложение В – Иерархия функциональных диаграмм. Диаграмма сущность-связь.

Диаграмма потоков данных Приложение Г UML-диаграммы

Приложение Д Листинг программы

Содержание отчета: ПЗ на электронном и бумажном носителе.

МДК.01.03 Разработка мобильных приложений

Практическое занятие № 1. Установка инструментария и настройка среды для разработки мобильных приложений

Цель работы: выполнить установку среды разработки и настроить ее для работы

.

Установка и настройка компонентов среды разработки

Приложения для Android, как и большинство приложений для коммуникаторов, разрабатываются на стандартном ПК, где ресурсы не ограничены (по сравнению с мобильным устройством) и загружаются на целевой коммуникатор для отладки, тестирования и последующего использования. Приложения можно отлаживать и тестировать на реальном устройстве под управлением Android или на эмуляторе. Для первоначальной разработки и отладки удобнее использовать эмулятор, а затем выполнять окончательное тестирование на реальных устройствах.

Разработчика предоставляется возможность использовать средства разработки приложений для Android на ПК под управлением любой из распространенных операционных систем: Windows, Linux иMac OS X. Ниже будет детально описан процесс установки компонентов среды разработки под Windows XP, для прочих операционных систем отличия весьма незначительны.

Установка JDK

Для Android SDK требуется JDK версии не ниже 5 (на момент написания данного методическогоруководства была актуальна 7-я версия Sun JDK и 6-я версия Open-JDK). Для Windows традиционно используется Sun JDK, который можно бесплатно загрузить на сайте разработчика:

Страница загрузки Sun JDK

http://www.oracle.com/technetwork/java/javase/downloads/index.html

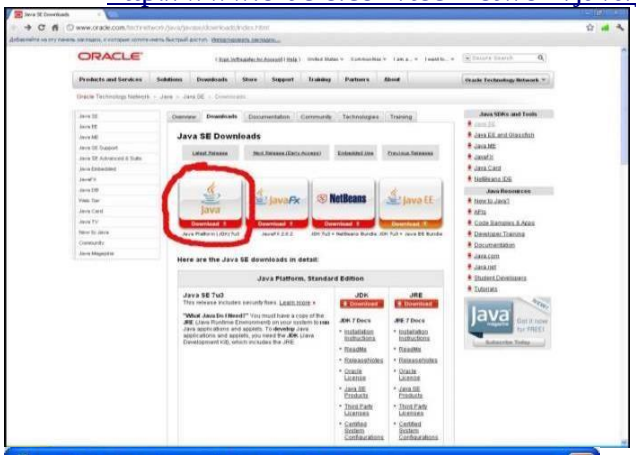


Рисунок 17, 18

Принятия условий лицензионного соглашения Oracle Binary Code License Agreement for Java SE

(1) можно выбрать подходящую для вашей платформы ссылку(2)Установка загруженного JDK достаточно проста:

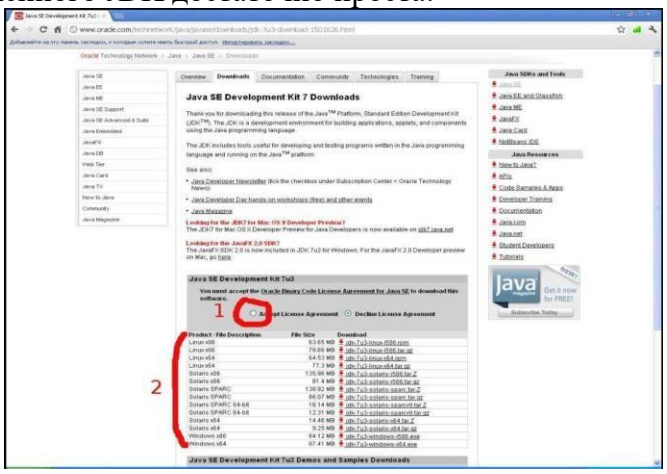


Рисунок 19

каталог для установки по умолчанию (1) не подходит, его можно изменить (2), а затемпродолжить установку:

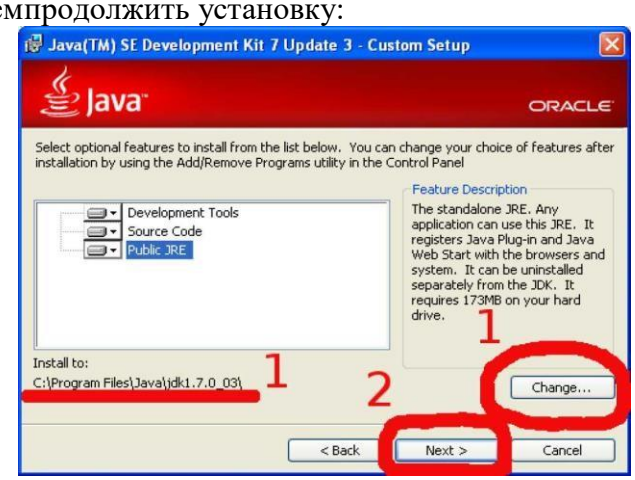


Рисунок 20

При установке Java Runtime так же можно изменить каталог для установки:

Рисунок 21

и продолжить установку:

Рисунок 22

Далее при желании можно зарегистрировать установленный продукт:

Рисунок 23

И установить JavaFX SDK, если не жалко 50 МБ дискового пространства:

Рисунок 24

Рисунок 25

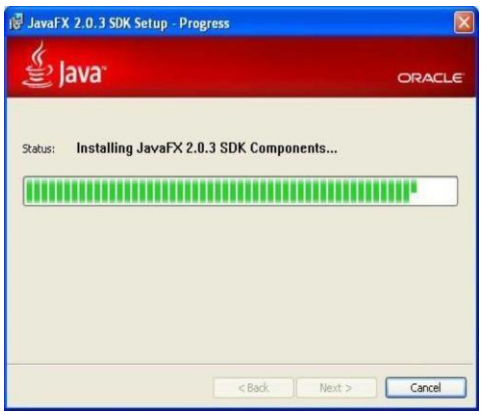


Рисунок 26

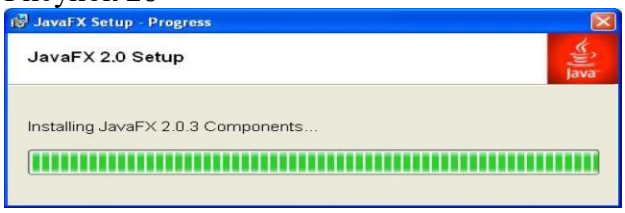


Рисунок 27

И, наконец, установка JDK закончена:

Рисунок 28

Установка Android SDK

На странице http://developer.android.com/sdk выбираем installer_r16-windows.exe

Здесь же имеются дополнительные инструкции по установке, а также ссылка на пошаговуюинструкцию по установке SDK на все поддерживаемые платформы: http://developer.android.com/sdk/installing.html

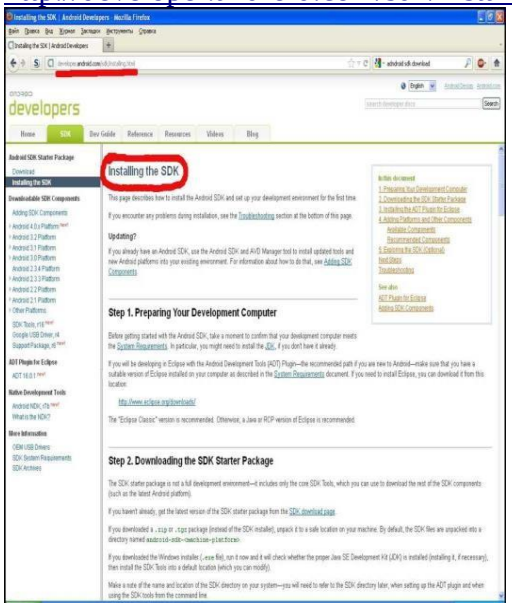


Рисунок 29

Ресурс http://developer.android.com является основным источником информации о платформе Android «из первых рук». В нем содержится огромное количество самых разнообразных сведений, необходимых разработчику, начиная с описания API и кончая такими вещами, как рекомендации по дизайну приложений и повышению производительности приложений. В данном методическом пособии в дальнейшем будут встречаться ссылки на конкретные страницы, посвященные изучаемым темам.

Установка загруженного Android SDK также не отличается особой сложностью:

Рисунок 30

Мастер установки обнаружит (если сможет) установленную версию JDK (1), после чегоустановку можно продолжить (2):

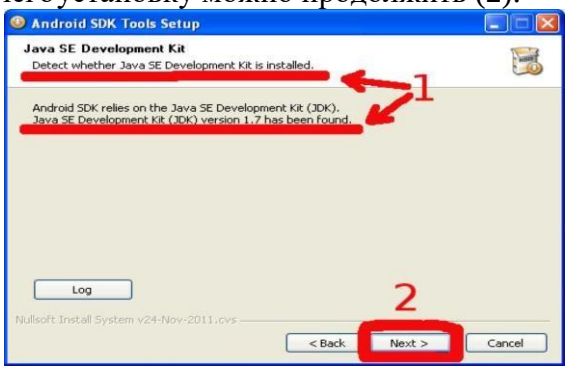


Рисунок 31

Рисунок 32

Установка начинается

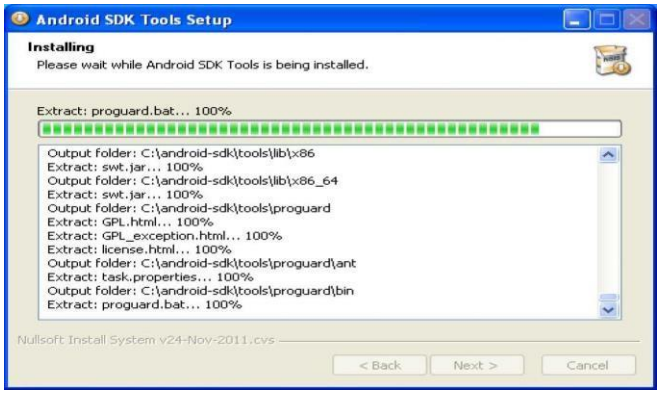


Рисунок 33

и заканчивается:

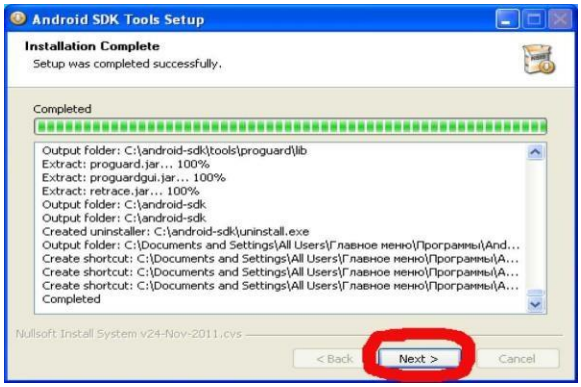


Рисунок 34

Можно сразу поставить галку(1) и запустить SDK Manager для окончательной настройки(2):

Рисунок 35

Требуется загрузить нужные версии API и другие полезные инструменты:

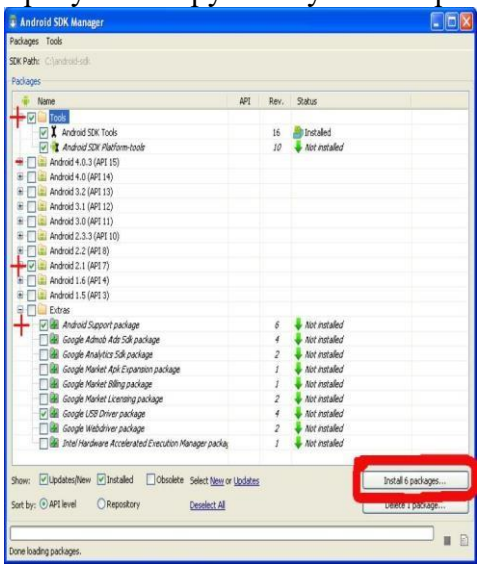


Рисунок 36

Далее мы запустим менеджер AVD (Android Virtual Devices) и сконфигурируем эмулятор на использование Android версии 2.1 с расширением Google API (GAPI). GAPI нужен, как правило, для приложений, использующих картографические возможности Android.

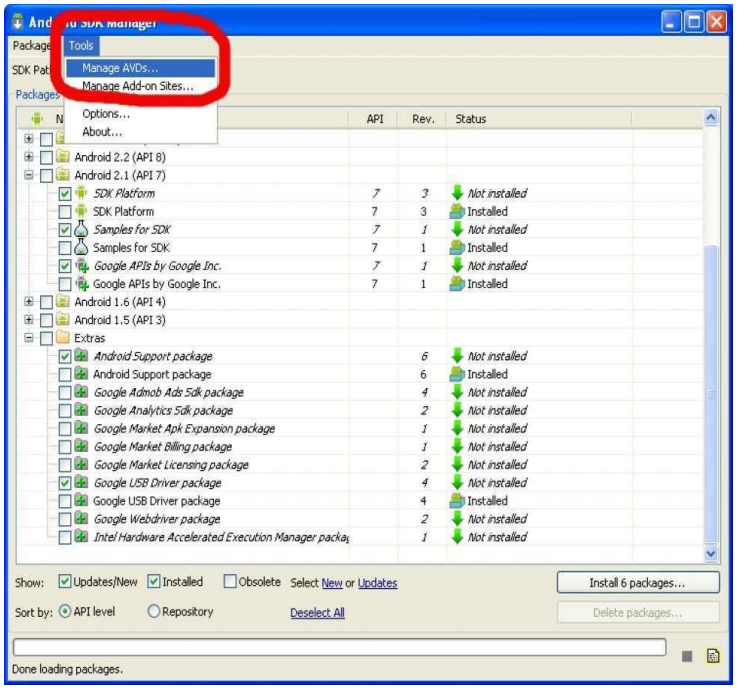


Рисунок 37Создадим новое виртуальное устройство:

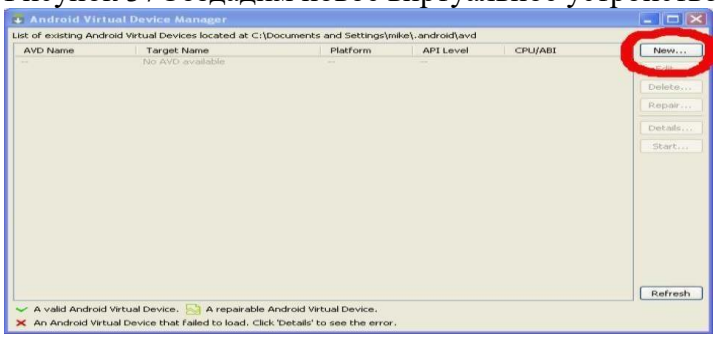


Рисунок 38

При создании нового виртуального устройства выбираем его название(1), целевой API (2), размер (или файл с образом) виртуальной SD-карты (3), одно из стандартных или собственное разрешение экрана (4) и, наконец, создаем устройство(5).

При необходимости можно указать плотность пикселей на экране, максимальный размер кучидля приложений внутри виртуальной машины, а также другие параметры:

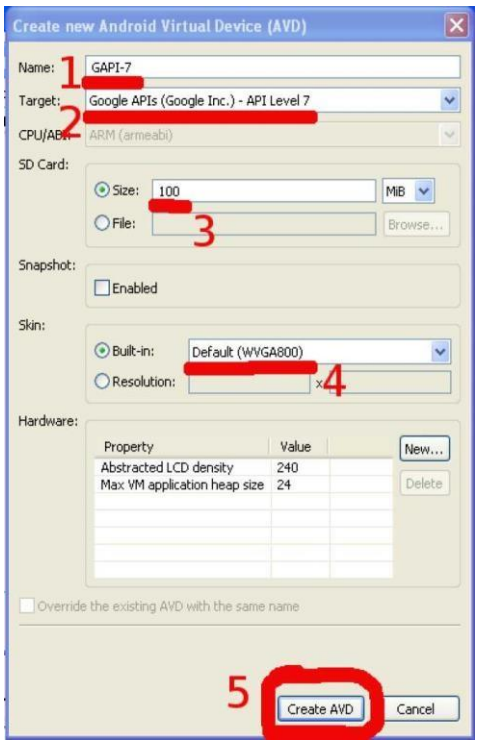


Рисунок 39

Убедимся, что все получилось, как хотели (1) и продолжим работу (2):

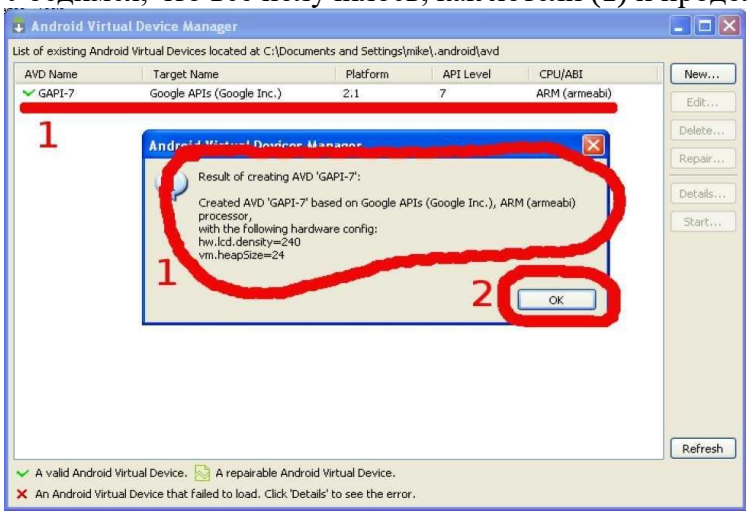


Рисунок 40

Пришло время запустить эмулятор:

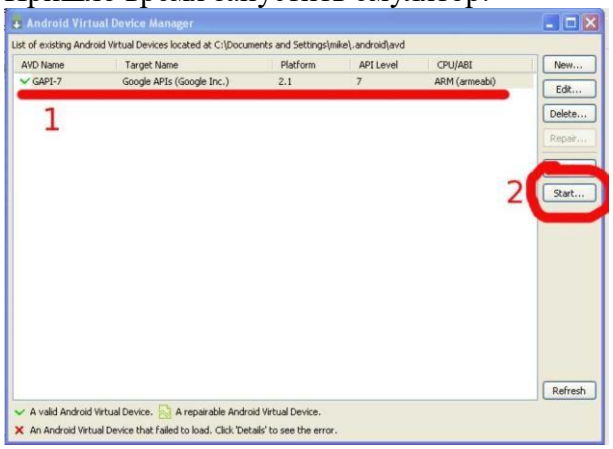


Рисунок 41

Важно: при использовании русскоязычных версий ОС Windows путь к рабочему каталогу AVD может включать русские буквы, что приведет к невозможности запуска

эмуляторов. В этом случае можно создать каталог для AVD в корневом каталоге диска (например, C:\AVD) и присвоить переменной окружения ANDROID_SDK_HOME значение этого пути.

При запуске эмулятора из менеджера AVD можно дополнительно указать некоторые параметры, особенно полезно управление размерами или плотностью пикселей экрана виртуального устройства: поставив галку (1), можно указать желаемый размер экрана (2) в дюймах, вычислить, при необходимости, плотность пикселей (3) на используемом мониторе ПК:

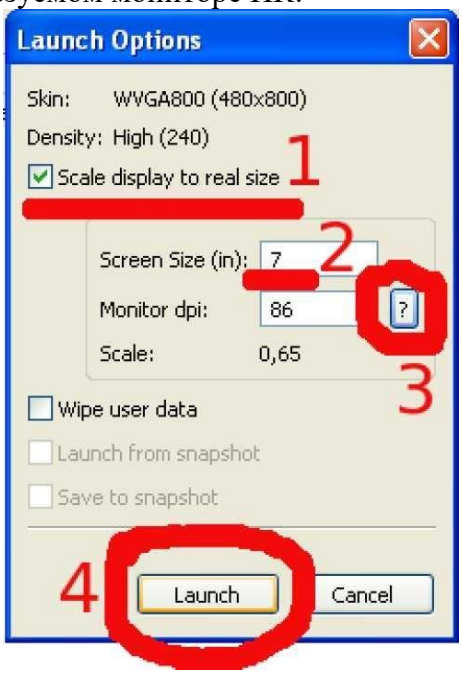


Рисунок 42

При первом запуске вы можете указать, следует ли отправлять анонимную статистику использования SDK в Google:

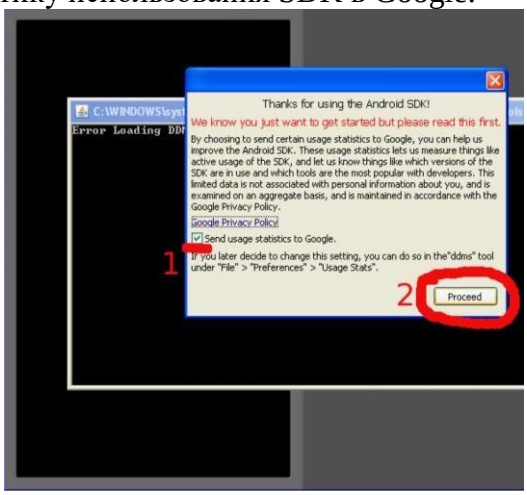


Рисунок 43

Более того, «пощупать» самые новые версии ОС Android и оценить их функциональные возможности можно безвозмездно, то есть даром просто выбрав нужную версию API при создании виртуального устройства.

Одна из неприятных особенностей платформы Android с точки зрения разработчика заключается в большом количестве выпущенных версий, значительно различающихся по возможностям. В настоящий момент Open Handset Alliance (т. е. Google) несколько затормозила этот процесс, и на данный момент флагманской версией является 4.0.x. Тем не менее, при выборе уровня API (API level) для нового ПО следует учитывать наличие у пользователей большого количество относительно старых

устройств, так что для охвата максимального сегмента рынка оптимальным выбором является Android 2.1 (поддерживается 97% процентов устройств с Android).

Так же, как и в реальном, в виртуальном устройстве существует опасность разрушитьфайловую систему при неправильном прекращении работы (сбое питания и т. п.).

Установка IDE Eclipse

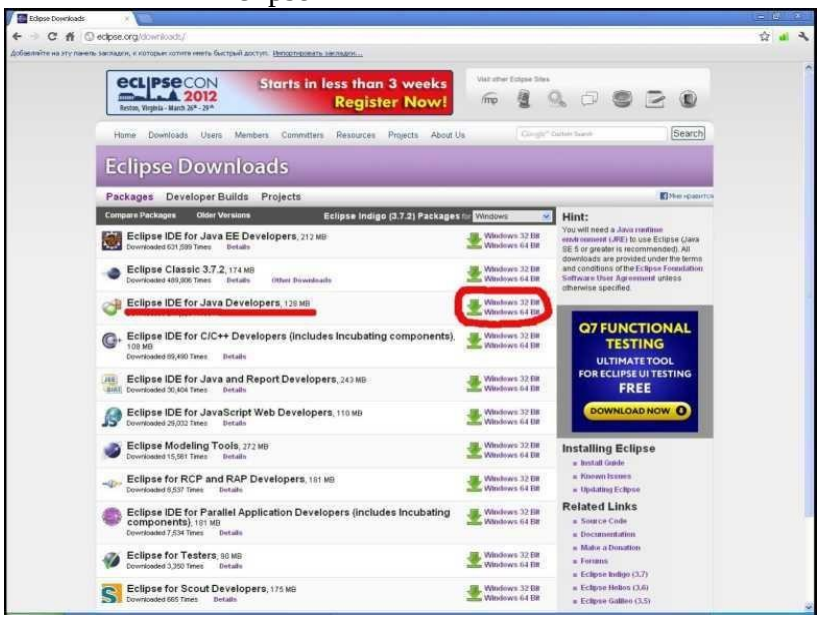


Рисунок 44

Интегрированная среда разработки Eclipse (довольно странное название, по-английски означает, в том числе помутнение рассудка) доступна для загрузки на официальном сайте по адресуhttp://eclipse.org/downloads/ и распространяется в виде ZIP-архива:

В нашем случае подходящим вариантом является Eclipse IDE for Java Developers.

После выбора подходящей для используемой на ПК операционной системы версии IDE, дляуменьшения времени загрузки имеет смысл выбрать европейское зеркало сайта:

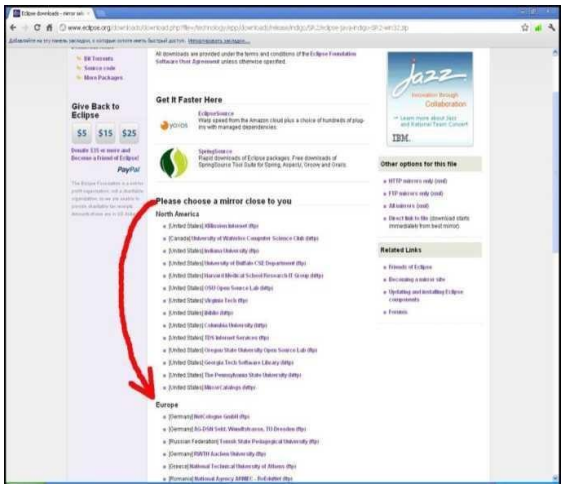


Рисунок 45

Получив нужный архив, выбираем подходящий путь для установки и распаковываем архив:

Рисунок 46

Установка плагина ADT

Для установки плагина ADT не требуется ручная загрузка, он будет установлен системой управления плагинами Eclipse. Тем не менее, мы рассмотрим установку подробно.

Запустим Eclipse

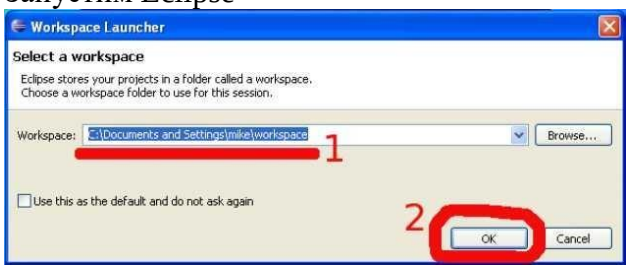


Рисунок 47

Выберем нужный пункт меню

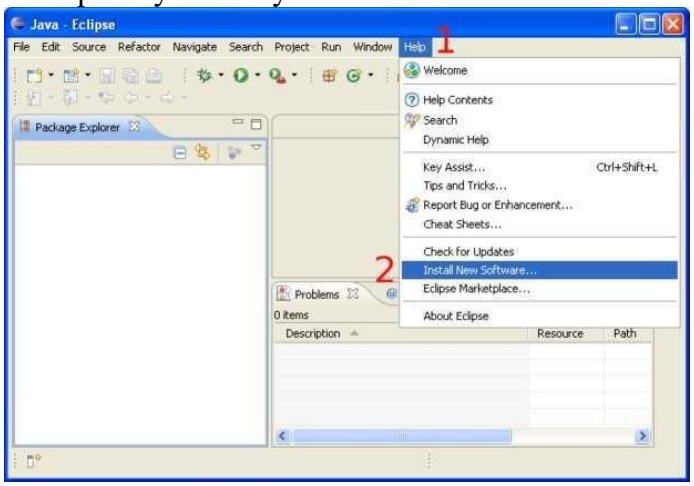


Рисунок 48

После установки выбранных компонентов перезапустим Eclipse

И настроим плагин ADT на использование уже установленного Android SDK.

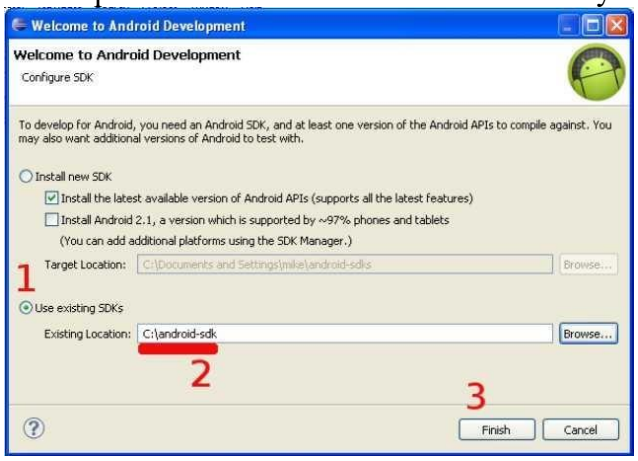


Рисунок 49

На снимке экрана (выше) можно увидеть, что ADT в состоянии сам загрузить и установить Android SDK. Мы использовали ручную установку для ускорения процесса (не тратили время на ожидание загрузки SDK).

Среда разработки установлена, пришло время создать первое приложение для платформыAndroid и запустить его в эмуляторе.

Создание первого приложения под Android

Для создания приложений в установленной нами среде разработки удобно использовать

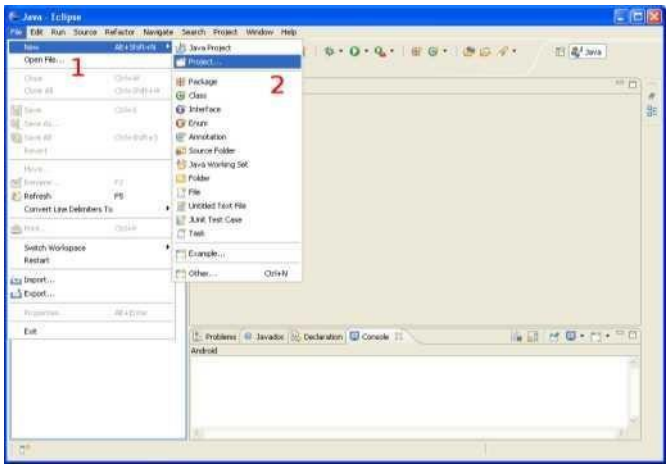


Рисунок 50

«Мастер создания нового проекта»:Выбираем «Android Project»

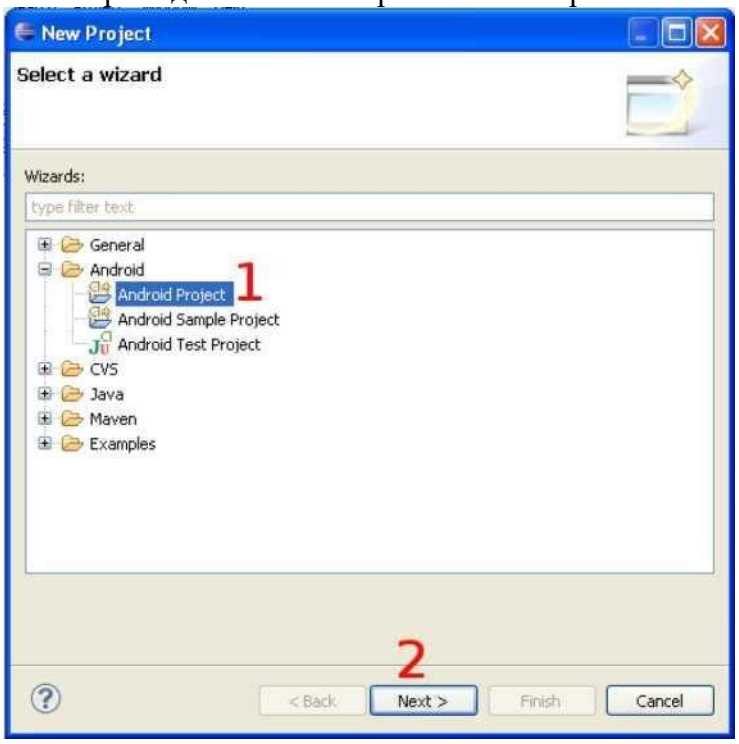


Рисунок 51

Вводим имя пакета (1) и имя класса Активности(2), который будет для нас автоматически создан мастером: Созданный проект сразу пока мы не внесли в него несовместимы с жизнью изменения является работоспособный приложением, так что мы его можем запустить:

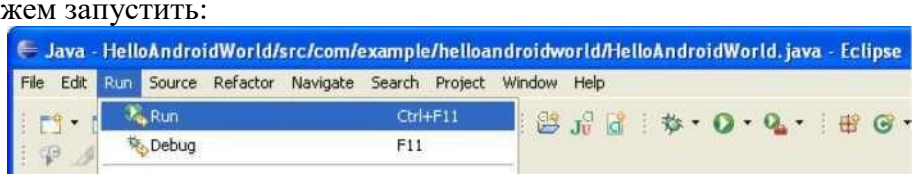


Рисунок 52

Даем понять Eclipse, как именно мы хотим запустить на выполнение наш проект:

Рисунок 53

Практическое занятие № 2. Установка среды разработки мобильных приложений с применением виртуальной машины

Цель работы: научиться устанавливать программное обеспечение на компьютер, получить практические навыки администрирования ОС.

Установка и настройка платформы виртуализации Oracle VM VirtualBox

VirtualBox - программа абсолютно бесплатная и полностью на русском языке, что делает её очень привлекательной для использования как на домашнем, так и на рабочем компьютере. Впервые система была предоставлена в 2007 г. компанией InnoTek в двух вариантах – с открытым и закрытым исходными кодами, причем обе были бесплатны при условии некоммерческого использования. В 2008 г. платформа была перекуплена компанией Sun Microsystems, которая и занимается её разработкой в настоящее время.

Платформа представляет собой систему виртуализации для host-систем Windows, Linux и MacOS и обеспечивает взаимодействие с гостевыми операционными системами Windows (2000/XP/2003/Vista/Seven), Linux (Ubuntu/Debian/ OpenSUSE/ Mandriva и пр.), OpenBSD, FreeBSD, OS/2 Warp.

Установка платформы Oracle VM VirtualBox

Скачать платформу, подходящую под Вашу систему, Вы можете по ссылке: http://www.virtualbox.org/wiki/Downloads

После того как установочный пакет оказался у Вас на жестком диске можно приступить к установке программы.

После запуска инсталлятора Вы увидите приветственное окно. Нажмите кнопку «Next» и в новом окне согласитесь с условиями лицензионного соглашения, поставив флажок «I accept the terms in the License Agreement». В следующем окне Вам будет предложено выбрать компоненты для установки и задать расположение исполняемых файлов. По умолчанию все компоненты устанавливаются на жесткий диск (а нам нужны все), а сама программа устанавливается в папку «Program Files» на системном диске. Если же вы хотите задать другое расположение, нажмите кнопку Browse и выберите новую папку для установки приложения.

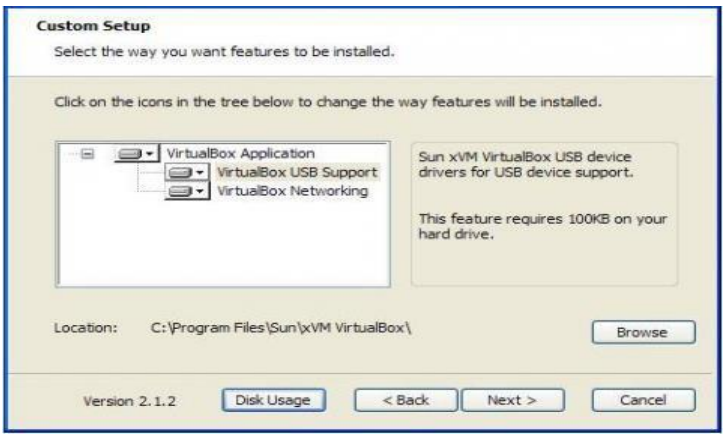


Рисунок 54Далее процесс установки не потребует от Вас никаких вмешательств, кроме разрешения на создания ярлыков на рабочем столе и в меню «Пуск». По окончанию установки программа запуститьсяавтоматически.

Создание и первичная настройка виртуальной машины

Запустим приложение Oracle VM VirtualBox (при установке платформы на рабочем столе создается ярлык, которым Вы можете воспользоваться). Для создания Вашей первой виртуальной машины щелкните кнопку «Создать»:

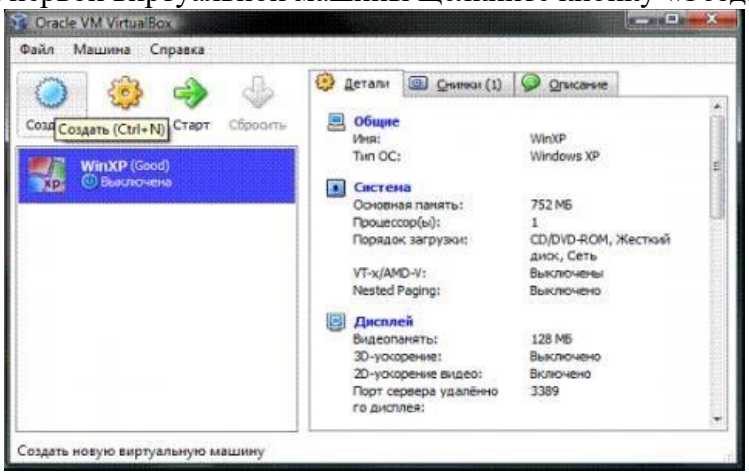


Рисунок 55

Примечание. В рассматриваемой платформе виртуализации уже существуют созданныевиртуальные машины, и поэтому при первом создании своей Вы не увидите списка виртуальных операционных систем.

Перед Вами откроется новое окно, в котором будет сообщение о запуске мастера создания виртуальной машины. Нажимаем кнопку «Next» и видим новое окно, предлагающее выбрать имя операционной системы, её семейство и версию. На рис. приводится выбор Windows XP, но Вы может выбрать любую из доступных систем по своему вкусу.

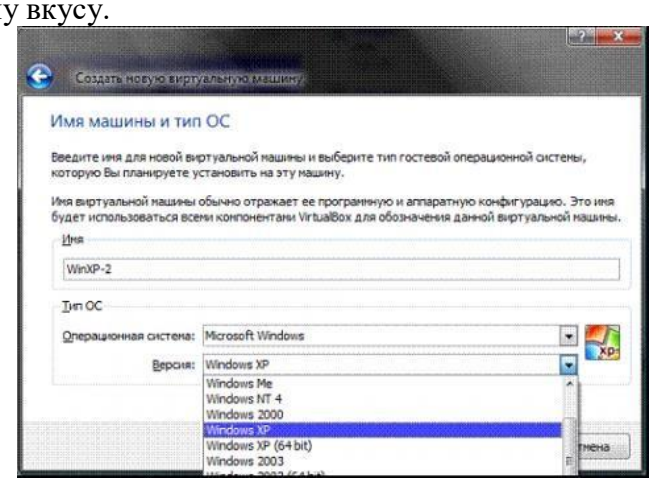


Рисунок 56

После нажатия кнопки «Next » Вам будет предложено определить размер оперативной памяти, выделяемой виртуальной машин. Здесь выбрано 1024мб, но для стабильной работы с виртуальной системой Windows XP достаточно будет и 512мб.

Далее потребуется создать виртуальный жесткий диск. Если Вы уже создавали виртуальные диски, можете использовать их, но мы рассмотрим именно процесс создания нового диска. Подтвердим, что создаваемый нами жесткий диск загрузочный, поставим флажок «Создать новый жесткий диск» и нажмем кнопку «Next».

Далее появится новое окно, которое сообщит Вам, что запущенный мастер поможет в создании виртуального диска, нажимаем кнопку «Next» для продолжения работы. В новом окне Вам будет предложено выбрать тип создаваемого диска – «динамически расширяющийся образ» или «образ фиксированного размера». Разница объясняется в справке данного окна, а от себя замечу, что непосредственно загрузочный диск удобнее создать фиксированного размера – это позволит Вам автоматически ограничить его размер, упростить и ускорить хранение, восстановление и создание резервных копий диска. К тому же, Вы можете создать для Вашей системы несколько жестких дискови вот уже те, которые не будут являться загрузочными, удобнее создавать динамически расширяющимися.

В следующем окне от Вас потребуется выбрать расположение создаваемого виртуальногожесткого диска и его размер. Для загрузочного жесткого диска с системой Windows XP достаточно размера установленного по умолчанию (10 Гб), а вот расположить его лучше вне Вашего системного раздела, т.к. не стоит не стоит перегружать Ваш реальный загрузочный диск и создавать на нем файлытакого размера.

После этого появится окно «Итог», в котором будет указан тип, расположение и размер создаваемого Вами жесткого диска. Если Вы согласны создать диск с такими параметрами, нажмите

«Финиш» и наблюдайте за процессом создания жесткого диска.

По завершения создания жесткого диска появится новое окно «Итог», в котором будут указаны параметры создаваемой Вами виртуальной машины. Если Вы не передумали ни по одному из описанных пунктов, нажимайте «Финиш» и переходите к настройке аппаратной части Вашей виртуальной машины.

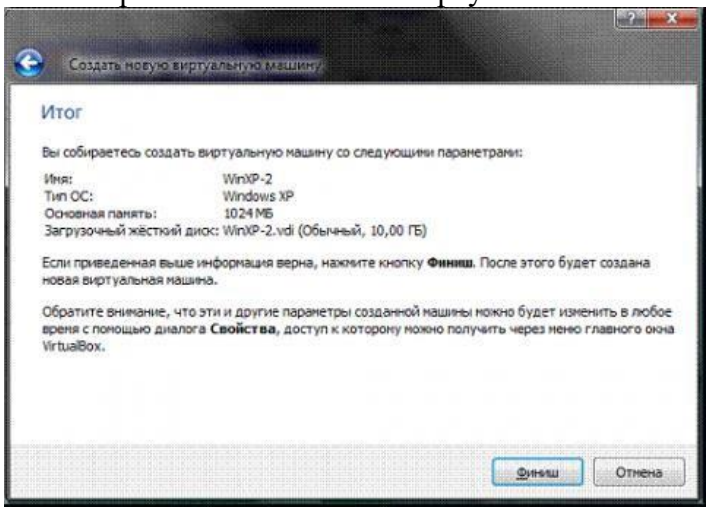


Рисунок 57Настройка аппаратной части виртуальной машины

Итак, Вы создали виртуальный жесткий диск, теперь настала очередь собрать наш виртуальный компьютер полностью. Для этого снова вернитесь к главному окну VirtualBox, в нем Вы уже можете увидеть только что созданную виртуальную машину WinXP-2, а в поле с правой стороны представленоеё описание, которое еще не похоже на описание полноценного ПК.

Рисунок 58

В колонке слева выберем нашу WinXP-2 и откроем её свойства, где колонка с левой стороны напоминает диспетчер устройств. На первой вкладке раздела «Общие» мы видим основные параметры нашей виртуальной машины:Перейдем на вкладку дополнительно и посмотрим, какие настройки системы мы можем произвести:

– «Папка для снимков». Если Вы разместили Ваш жесткий в собственном расположении, то лучше и эту папку перенести туда же, т.к. снимки имеют большой вес и, опять-таки, не стоит перегружать Ваш системный диск. Моя рекомендация – создавать снимки перед каждым значительным изменением, которые Вы хотите произвести в виртуальной системе, причем даже на одну виртуальную машину Вы можете создать несколько снимков, содержащих отличные друг от друга настройки и установленные приложения;

– «Общий буфер обмена» – определение того, как будет работать буфер обмена между Вашейhost-системой и виртуальной машиной. Вариантов работы буфера предоставлено четыре –

«выключено», «только из гостевой ОС в основную», «только из основной ОС в гостевую»,

«двунаправленный». Мы выберем последний вариант, т.к. это обеспечит нам максимальное удобство в работе;

– «Сменные носители информации запоминать изменения в процессе работы», тут мы ставимфлажок в знак согласия, т.к. данная опция позволит системе запомнить состояние CD\DVD-приводов;

– «Мини тулбар» – это небольшая консоль, содержащая элементы управления виртуальной машиной. Её лучше применять только в полноэкранном режиме, т.к. она полностью дублируется главным меню рабочего окна виртуальной машины. Располагать её действительно лучше сверху просто потому, что можно случайно нажать на какой-нибудь элемент управления, пытаясь, например, развернуть окно из панели задач виртуальноймашины.

Перейдем к разделу система и на первой вкладке материнская плата произведем следующие настройки:

– если нужно, откорректируем размер оперативной памяти Вашей виртуальной машины, хотяокончательно убедится в правильности выбранного объема Вы сможете только после запускавиртуальной машины. Выбирать размер Вы можете, исходя из объема доступной физическойпамяти, установленной на Вашем ПК. Например, при наличии 4ГБ ОЗУ оптимальным будетвыделение 1ГБ, т.е. одной четвертой части, что позволит Вашей виртуальной машине работать без малейших зависаний;

– откорректируем порядок загрузки - дисковод гибких дисков («дискета) можно вообще отключить, а первым обязательно поставьте CD/DVD-ROM, чтобы обеспечить возможностьустановки ОС с загрузочного диска. При этом в роли

загрузочного диска может выступать как и компакт-диск, так и образ ISO;

– все остальные настройки описаны в динамической справке снизу, и их применение зависит от аппаратной части вашего реального ПК, причем если Вы выставите настройки неприменимые к Вашему ПК система виртуальной машины просто не запуститься;

Перейдем к вкладке «Процессор», тут Вы можете выбрать количество процессоров, установленных на Вашу виртуальную материнскую плату. Обратите внимание, что это опция будет доступна только при условии поддержки аппаратной виртуализации AMD-V или VT-x, а такжевключенной опции OI APIC на предыдущей вкладке.

Здесь обратите Ваше внимание на настройки аппаратной визуализации AMD-V или VT-x. Перед включением этих настроек, нужно выяснить, поддерживает ли эти возможности Ваш процессор и включены ли они по умолчанию в BIOS (нередко они отключены).

Перейдем к разделу «Дисплей». В данном разделе на вкладке «Видео» Вы можете установить размер памяти виртуальной видео карты, а также включить 2D и 3D ускорение, причем включение 2Dускорения желательно, а 3D необязательно. На вкладке «Удаленный дисплей» Вы можете включить опцию, при которой Ваша виртуальная машина будет работать как сервер удаленного рабочего стола (RDP).

Переходим к разделу носители. Тут Вы можете увидеть созданной ранее виртуальный жесткий диск и позицию с надписью пусто. Выделяем эту позицию и осуществляем настройку.

Для настройки виртуального привода компакт-дисков можно пойти двумя путями:

– первый вариант - в раскрывающемся меню «Привод» выбираем Ваш реальный или виртуальный CD/DVD-ROM (существующие в реальной системе) и загружаем в него физический диск с дистрибутивом Windows XP или ISO-образ, если это эмулятор;

– второй вариант - щелкаем значок так, как показано на рисунке ниже и в отрывшемся окне добавляем ISO-образ загрузочного диска Windows XP, этим путем мы и пойдем.

Примечание. В данном пункте Вы уже не можете выбрать дистрибутив другой операционной системы, т.к. версия ОС уже была определена в самом начале процесса настройки виртуальной машины.

На рисунке ниже представлена процедура добавления ISO-образов в менеджер виртуальных носителей. В него Вы можете внести любое количество образов различного назначения, например, игры, дистрибутивы приложений, базы данных и пр., которые Вы сможете потом быстро переключатьчерез главное меню окна виртуализации VirtualBox.

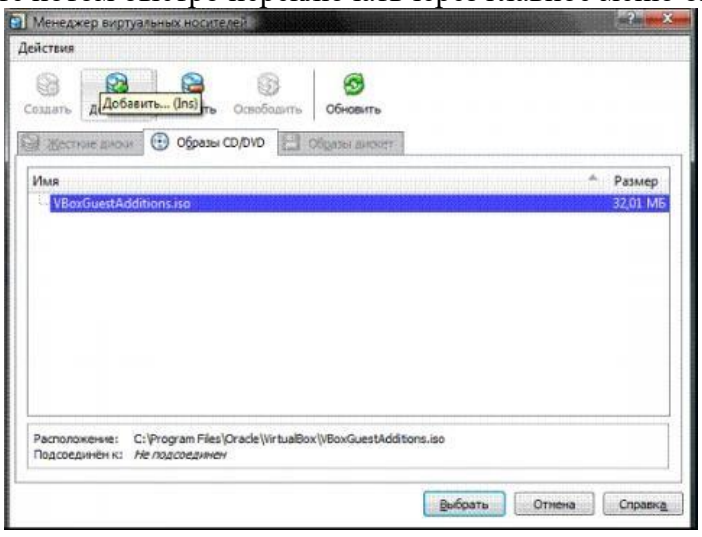


Рисунок 59

Далее Вы можете настроить слоты подключения накопителей, для упрощения описания привожу скриншоты, по которым Вы можете произвести действия по настройке. По привычке, я устанавливаюпривод компакт-дисков как «Первичный мастер IDE», жесткий диск, содержащий загрузочный раздел,как «Вторичный мастер IDE», а дополнительный виртуальный жесткий диск «Первичный слейв IDE».Настройка сети и сетевого взаимодействия не рассматривается в рамках данной статьи, поэтому замечу лишь то, что сетевой адаптер типа NAT включен по умолчанию, а этого уже достаточно дляпредоставления Вашей виртуальной машине доступа в Интернет. Тип выбираемого адаптера должен быть «Pcnet-Fast III (Am79C973)», т.к. только для этого адаптера присутствуют драйверы в ОС

Windows XP.

Раздел COM я подробно не описываю, т.к. подключать к портам данного типа уже нечего. В случае если Вам все же потребуется подключить устройство с интерфейсом RS-232C, то наиболее удобным будет включить COM-port виртуальной машины в режиме «хост-устройство», а в качестве

«пути к порту» использовать имя реально порта Вашего ПК, которое Вы можете посмотреть в диспетчере устройств.

Переходим к разделу USB, здесь ставим оба доступных флажка, а затем, используя кнопку сизображением «вилки» USB и «плюса», добавляем все доступные контроллеры.

Переходим к разделу «Общие папки» и выбираем папки, которые нужно сделать доступнымидля виртуальной машины.

Примечание. Обратите внимание на динамическую справку снизу – именно таким образом, черезокно командной строки, Вы сможете подключить общие папки к Вашей виртуальной машине.

На этом настройка аппаратной части Вашей виртуальной машины закончена, и можно перейтик установке операционной системы.

Настройка операционной системы виртуальной машины

Описание установки операционной системы в статье не описывается, т.к. на сайте представленодостаточно информации о методах и тонкостях данной операции, поэтому укажу первый шаг – возвращаемся к главному окну VirtualBox и нажимаем кнопку «Старт».

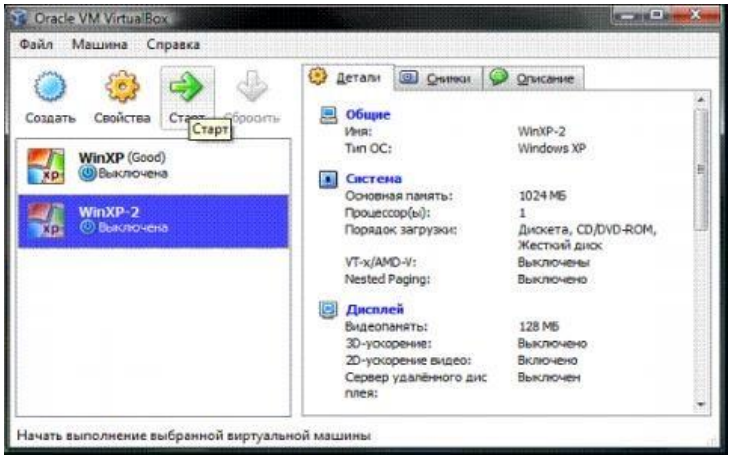


Рисунок 60

Начнется установка операционной системы Windws 7 Professional. После завершения установкиВы увидите следующее окно:

Для начала нам нужно установить драйверы для всех виртуальных аппаратных компонентов нашего виртуального ПК. Для этого в главном меню выбираем пункт «Устройства» - «Приводы оптических дисков» - «VboxGuestAdditions.iso». Впоследствии таким же образом Вы сможете подключить к своей виртуальной машине физический CD-ROM или загрузить ISO-образ.

После подключения образа «VboxGuestAdditions.iso» в папке Мой компьютер Вы увидите, что впривод компакт-дисков загружен данный виртуальный диск – остается его только запустить двойнымщелчком левой кнопки мыши.

Сам процесс установки происходит практически без Вашего участия и только в случае, если Вывключили ранее 3D-ускорение, то следует выбрать соответствующий компонент для дополнительнойустановки.

В последнем окне процесса инсталляции Вам будет предложено перезагрузить виртуальную машину - соглашайтесь, после перезагрузки Вы увидите, что изображение стало четче, размер окна виртуальной машины изменяется динамически, включена функция интеграции мыши и есть доступ в Интернет.

Давайте теперь подключим общие папки, чтобы получить возможность перенести в созданнуювиртуальную машину нужные для работы файлы и установить приложения. Это можно сделать с помощью командной строки, следуя справке VirtualBox, но я приведу способ с использованиемпроводника Windows. Для этого откройте папку Мой компьютер, в главном меню выберите «Сервис»

- «Подключить сетевой диск» и открывшемся окне в поле папка введите \\vboxsrv\имя_общей_папки, т.е. в нашем случае: \\vboxsrv\Share

После этих действий в папке «Мой компьютер» появится Ваша общая папка, доступная вкачестве сетевого диска.

Давайте теперь проверим, есть ли у Вас доступ в Интернет. Для этого откройте: меню «Пуск» -

«Программы» - «Стандартные» - «CMD-командная строка» и в открывшемся окне (рис. 31) введитеследующую команду: Ping ya.ru

Если в результате отработки команды Вы видите, что пакеты отправлены и получены, пусть даже частично, то у Вас все получилось, и доступ в Интернет виртуальной машине обеспечен.

Быстрый доступ и комфортная работа

Далее хотелось бы сказать несколько слов о двух возможностях VirtualBox, которые помогут ускорить Вашу работу с виртуальными машинами и сделать ее комфортнее.

Ярлык для быстрого запуска виртуальной машины

Для более быстрого и удобного запуска Вашей виртуальной машины создадим ярлык именнодля нее. Сделать это можно следующим образом:

a. Щелчком правой кнопки мыши на рабочем столе вызовем контекстное меню и выберемпункт «Создать ярлык»;

b. В открывшемся окне в поле «Укажите размещение объекта» введите "C:\ProgramFiles\Oracle\VirtualBox\VBoxManage.exe"

c. startvm Win7Pro;

d. В следующем окне введите имя ярлыка, например, «Win7Pro» и нажмите кнопку «Готово»;

e. На Вашем рабочем столе появился созданный ярлык «Win7Pro», щелкните на нем правой кнопкой;В открывшемся контекстном меню выберите «Свойства»;

f. В появившемся окне выберите сменить значок и поле выбора значков выберите любой значок, который посчитаете нужным.

Режим интеграции дисплеев

В режиме интеграции дисплеев Вы легко можете со своего рабочего стола организовать доступ к рабочему столу и элементам управления виртуальной машины, т.е. все окна, открываемые Вами в виртуальной машине, будут отображаться уже на Вашем рабочем столе, а не в отдельном окне VirtualBox. Данная функция значительно облегчает доступ к виртуальной машине, её элементам управления и, установленным на ней приложениям. Чтобы включить этот режим Вам нужно в главном меню окна

визуализации VirtualBox выбрать пункт с соответствующим названием или нажать сочетание клавиш «HOST + L», где «HOST» клавиша – левый «Ctrl» (по умолчанию).

Программа выполнения работы

1. Скачайте платформу, подходящую под Вашу систему по ссылке: http://www.virtualbox.org/wiki/Downloads
2. Создайте на диске D: папку DISTRIB. С указанного преподавателем сетевого ресурсаскачайте в эту папку ISO-образ операционной системы Windows 7 Pro (32 bit).
3. Запустите инсталлятор и выполните установку платформы Oracle VM VirtualBox.
4. Создайте новую виртуальную машину Win7Pro. Выберите для новой виртуальной машиныобъем оперативной памяти 1024 MB и создайте новый загрузочный жесткий диск фиксированного размера емкостью 25 GB на несистемном диске (диске D:) вашего компьютера.
5. Выполните настройку аппаратной части созданной виртуальной машины.
6. На несистемном диске (диске D:) вашего компьютера создайте папку для снимков.
7. Настройте буфер обмена между хостовой и виртуальной машинами как «двунаправленный».
8. Откорректируйте порядок загрузки. Первым обязательно поставьте CD/DVD-ROM, чтобыобеспечить возможность установки ОС с загрузочного диска. При этом в роли загрузочногодиска может выступать как и компакт-диск, так и образ ISO.
9. На вкладке «Процессор» выберите количество процессоров, установленных на Вашу виртуальную материнскую плату. Обратите внимание, что это опция будет доступна толькопри условии поддержки аппаратной виртуализации AMD-V или VT-x, а также включенной опции OI APIC на предыдущей вкладке. Здесь обратите Ваше внимание на настройки аппаратной визуализации AMD-V или VT-x. Перед включением этих настроек, нужно выяснить, поддерживает ли эти возможности Ваш процессор и включены ли они по умолчанию в BIOS (нередко они отключены).
10. В разделе «Дисплей» на вкладке «Видео» установите размер памяти виртуальной видеокарты, а также включите 2D и 3D ускорение, причем включение 2D ускорениежелательно, а 3D необязательно.
11. В разделе “Носители” выделите позицию с надписью “Пусто” и добавьте ISO-образ загрузочного диска Windows 7.
12. В разделе USB поставьте оба доступных флажка, а затем, используя кнопку с изображением «вилки» USB и «плюса», добавьте все доступные контроллеры.
13. Запустите установку и установите операционную систему Windows 7 Pro.
14. Установите драйверы для всех виртуальных аппаратных компонентов нашего виртуальногоПК. Для этого в главном меню выберите пункт «Устройства» - «Приводы оптических дисков» - «VboxGuestAdditions.iso». Впоследствии таким же образом Вы сможете подключить к своей виртуальной машине физический CD-ROM или загрузить ISO-образ. После подключения образа «VboxGuestAdditions.iso» в папке Мой компьютер Вы увидите,что в привод компакт-дисков загружен данный виртуальный диск – остается его только запустить двойным щелчком левой кнопки мыши.
15. Создайте ярлык для более быстрого и удобного запуска Вашей виртуальной машины.
16. Щелчком правой кнопки мыши на рабочем столе вызовите контекстное меню и выберитепункт «Создать ярлык»;
17. В открывшемся окне в поле «Укажите размещение объекта» введите "C:\ProgramFiles\Oracle\VirtualBox\VBoxManage.exe"
18. startvm Win7Pro;
19. В следующем окне введите имя ярлыка, например, «Win7Pro» и нажмите

кнопку «Готово»;На Вашем рабочем столе появился созданный ярлык «Win7Pro», щелкните на нем правойкнопкой;

20. В открывшемся контекстном меню выберите «Свойства»;
21. В появившемся окне выберите значок? Который посчитаете нужным.
22. Нажмите кнопку «ОК», а затем «Применить».
23. Включите режим интеграции дисплеев, выбрав в Главном меню пункт с соответствующимназванием.
24. Оформите отчет по практической работе.

Практическое занятие № 3. Создание эмуляторов и подключение устройств. Настройка режима терминала

Цель работы: Знакомство с интерфейсом среды программирования. Изучение структуры проекта. Рассмотрите основные понятия Android проекта.

Структура проекта

- зrc—«исходный код» приложения (java-классы)
- assets —пустая директория. Может использоваться для сохранения raw-файлов.
- gen—хранилище генерируемых системных файлов. В частности, здесь располагается файл R.java, в котором хранятся идентификаторы всех ресурсов, создаваемых в проекте (строковые ресурсы и т.п.).
- libs—различные библиотеки, используемые приложением
- res—ресурсы проекта.
- AndroidManifest.xml—файл описания проекта (поддерживаемые версии SDK, версия приложения и т.п.)
- project.properties—файл, включающий настройки проекта, такие как build target.Ресурсы проекта
- anim/ Содержит XML файлы, компилируемые в анимационные объекты.
- color/ Содержит XML файлы описывающие цвета.
- drawable/ Содержит растровые файлы (PNG, JPEG, orGIF), 9-Patch
- файлы, и XML файлы, описывающие Drawableshapes или Drawableobjectsвключающие
- множественные состояния(нормальное, нажатое, состояние фокуса).
- layout/ Содержит XML файлы описывающие макеты экрана
- menu/ Содержит XML файлы, определяющие меню приложения.
- raw/ Для хранения произвольных файлов.
- values/ Содержит XML файлы, компилируемые во множество видов ресурсов(strings.xmlи т.д).
- xml/ СодержитXML файлы конфигурирующие компоненты приложения. Например,XML файл определяющий экран настроек.

Пример простейшего файла AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.untitled"
android:versionCode="1 android:versionName="1.0">
<uses-sd kandroid:minSdkVersion="19"/>
<applicationandroid:label="@string/app_name"
android:icon="@drawable/ic_launcher">
<activity android:name="MyActivity"android:label="@string/app_name">
<intent-filter>
<action android:name="android.intent.action.MAIN"/>
<category android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
</activity>
```

</application>
</manifest>
Для начала работы требуется:
Java Development Kit
Android Software Development KitAndroid Virtual Device Manager

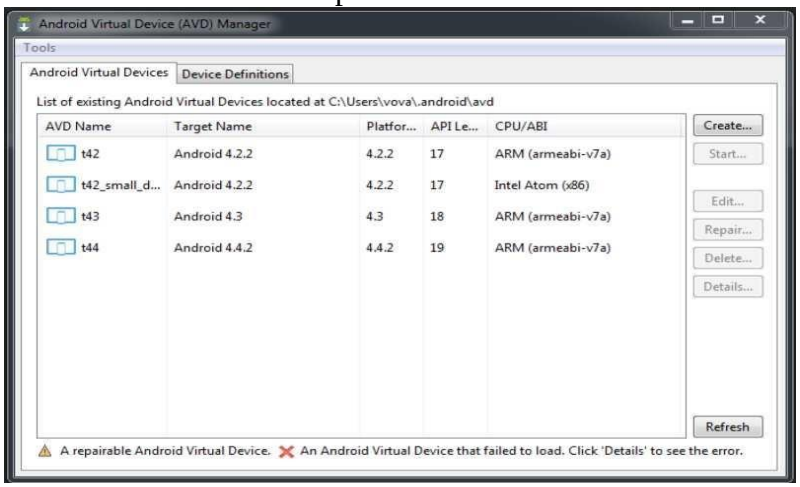


Рисунок 61

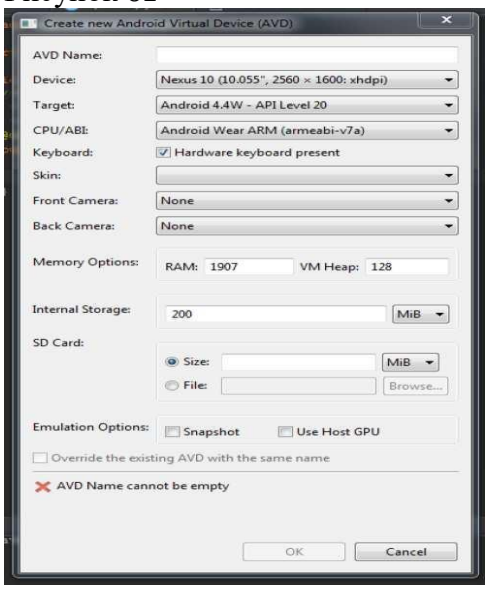


Рисунок 62
Задание 1. Создание проекта приложения
Запустите среду программирования в IDE InelliJ Idea

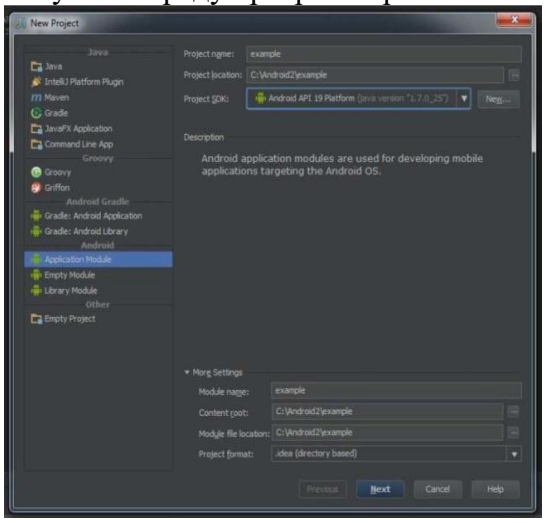


Рисунок 63

Задание 2. Создание приложений с одним экраном (Activity) Необходимо создать два activity и организовать переход между ними. Содержимое Activity 1 кнопка с именем btnl

Содержимое Activity 2: TextView с текстом «Переданный параметр: значение_параметра». значение_параметра - принятый параметр из Activity 1.

При запуске приложения пользователь должен попадать на экран с Activity 1. После нажатия накнопку btn1 необходимо осуществить переход к Activity 2 и передавать параметр из Activity 1. В качестве значения передаваемого параметра использовать свою фамилию.

Ключевые классы: Activity, Intent, Button, TextView.

Практическое занятие № 4. Создание нового проекта. Изучение и комментирование кода. Изменение элементов дизайна

Цель работы: Изучить работу Android приложения с базой данных.SQLite

SQLite—компактная встраиваемая реляционная база данных с открытым исходным кодом.Поддерживает динамическое типизированиеданных.

Возможные типы полей: INTEGER, REAL, TEXT, BLOB.SQLiteOpenHelper

Абстрактные методы

onCreate( ) -вызывается при первом создании базы данных;

onUpgrade( ) -вызывается при модификации базы данных Реализация метода onCreate

public void onCreate(SQLiteDatabasedb) { db.execSQL("CREATE TABLE"+ TABLE_NAME

+ "(_id INTEGER PRIMARY KEYAUTOINCREMENT ,"

+ COL NAME + " TEXT ,"+ COL PHONE + " TEXT) ; ") ;

Реализация метода onUpgrade@ Override

publicvoid onUpgrade(SQLiteDatabasedb, intoldVersion, intnewVersion)

{

db.execSQL("DROP TABLE IF EXISTS"+ TABLE_NAME) ;

onCreate(db) ;

}

Чтение из базы данных

Cursor query(String table, String [ ] columns, Stringselection, String [ ] selectionArgs, String groupBy, String having , String sortOrder)Позиционирование курсора

1. moveToFirst( ) -перемещает курсор в первую запись в выборке;
2. moveToLast( ) -перемешает курсор в последнюю запись в выборке;
3. moveToNext( ) -перемешает курсор в следующую запись и одновременноопределяет, существует ли эта запись. Метод moveToNext( ) возвращаетЦие, если курсор указывает на другую строку после перемещения, и false^enM текущая запись была последней в выборке;
4. moveToPrevious( ) -перемешает курсор в предыдущую запись;
5. moveToPosition( ) -перемешает курсор в указанную позицию;
6. getPosition( ) -возвращает текущий индекс позиции курсора.
7. isFirst( ) ;
8. isLast( ) ;
9. isBeforeFirst( ) ;
10. isAfterLast( ) . Обновление и удаление записей
11. intupdate(String table, ContentValuesvalues, String whereClause, String [ ] whereArgs)
12. intdelete (String table, String whereClause, String [ ] whereArgs)

Задание 1. Необходимо создать приложение, взаимодействующее с базой данных. Первое активити должно содержать три кнопки. При нажатии на первую кнопку должно

открываться новое активити, выводящее информацию из таблицы «Одногруппники» в удобном для восприятия формате.

При запуске приложения необходимо:

1. Создать БД, если ее не существует.
2. Создать таблицу «Одногруппники», содержащую поля:
3. ID;
4. ФИО;
5. Время добавления записи.
6. Удалять все записи из БД, а затем вносить 5 записей об одногруппниках. При нажатии на вторую кнопку необходимо внести еще одну запись в таблицу.

При нажатии на третью кнопку необходимо заменить ФИО в последней внесенной записи наИванов Иван Иванович.

Задание 2. Создать новое отдельное приложение на основе приложения, созданного в части 1. Переопределить функцию onUpgrade. При изменении версии БД необходимо удалить таблицу

«Одногруппники», создать таблицу «Одногруппники» содержащую следующие поля:

1. ID;
2. Фамилия;
3. Имя;
4. Отчество;
5. Время добавления записи.
6. Изменить версию базы данных.Задание 3.
1. Создайте новый проект MetroPicker.
2. Добавьте вспомогательную Активность ListViewActivity для отображения и выбора станций метро, в качестве заготовки используйте результаты лабораторной работы «ИспользованиеListView» .
3. Отредактируйте файл разметки res/layout/main.xml: добавьте кнопку выбора станции метро,присвоив идентификаторы виджетам TextView и Button для того, чтобы на них можно былоссылаться в коде.
4. Установите обработчик нажатия на кнопку в главной Активности для вызова списка станциии выбора нужной станции.
5. Напишите нужный обработчик для установки выбранной станции метро в виджет TextViewродительской Активности (метод setText виджета TextView позволяет установить отображаемый текст). Не забудьте обработать ситуацию, когда пользователь нажимает кнопку «Назад» (в этом случае «никакой станции не выбрано» и главная Активность должнаизвестить об этом пользователя).
6. Убедитесь в работоспособности созданного приложения, проверив реакцию различныедействия потенциальных пользователей.
7. Неявные намерения
8. Неявные намерения используются для запуска Активностей для выполнения заказанных действий в условиях, когда неизвестно (или безразлично), какая именно Активность (и из какого приложения) будет использоваться.
9. При создании Намерения, которое в дальнейшем будет передано методу startActivity, необходимо назначить действие (action), которое нужно выполнить, и, возможно, указать URI данных, которые нужно обработать. Также можно передать дополнительную информацию с помощью свойства extras Намерения. Android сам найдет подходящую Активность (основываясь на характеристиках Намерения) и запустит ее. Пример неявного вызова телефонной «звонилки»:
10. Intent intent = new Intent(Intent.ACTION DIAL, Uri.parse("tel:(495)502-99-11"));
11. startActivity(intent);
12. Для определения того, какой именно компонент должен быть запущен для

выполнения действий, указанных в Намерениях, Android использует Фильтры Намерений (Intent Filters').Используя Фильтры Намерений, приложения сообщают системе, что они могут выполнять определенные действия (action) с определенными данными (data) при определенных условиях (category) по заказу других компонентов системы.

13. Для регистрации компонента приложения (Активности или Сервиса) в качестве потенциального обработчика Намерений, требуется добавить элемент <intent-filter> в качестве дочернего элемента для нужного компонента в Манифесте приложения. У элемента <intent-filter> могут быть указаны следующие дочерние элементы (и соответствующие атрибуты у них):

14. <action>. Атрибут android:name данного элемента используется для указания названия действия, которое может обслуживаться. Каждый Фильтр Намерений должен содержать не менее одного вложенного элемента <action>. Если не указать действие, ни одно Намерение не будет «проходить» через этот Фильтр. У главной Активности приложения в Манифесте должен быть указан Фильтр Намерений с действием android.intent.action.MAIN

15. <category>. Сообщает системе, при каких обстоятельствах должно обслуживаться действие (с помощью атрибута android:name). Внутри <intent-filter> может быть указано несколько категорий. Категория android.intent.category.LAUNCHER требуется Активности, которая желает иметь «иконку» для запуска. Активности, запускаемые с помощью метода startActivity, обязаны иметь категорию android.intent.category.DEFAULT

16. <data>. Дает возможность указать тип данных, которые может обрабатывать компонент.

<intent-filter> может содержать несколько элементов <data>. В этом элементе могут использоваться следующие атрибуты:

17. android:host : имя хоста (например, www.specialist.ru)
18. android:mimetype : обрабатываемый тип данных (например, text/html)
19. android:path : «путь» внутри URI (например, /course/android)
20. android:port : порт сервера (например, 80)
21. android:scheme : схема URI (например, http) Пример указания Фильтра Намерений:

<activity android:name=".MyActivity" android:label="@string/app name" > <intent-filter>
<action android:name="android.intent.action.SEND" />
<category android:name="android.intent.category.DEFAULT" />
<data android:mimeType="text/plain" />
</intent-filter>
</activity>

22. При запуске Активности с помощью метода startActivity неявное Намерение обычно подходит только одной Активности. Если для данного Намерения подходят несколько Активностей, пользователю предлагается список вариантов.

23. Определение того, какие Активности подходят для Намерения, называется Intent Resolution. Его задача - определить наиболее подходящие Фильтры Намерений, принадлежащие компонентам установленных приложений. Для этого используются следующие проверки в указанном порядке:

24. Проверка действий. После этого шага остаются только компоненты приложений,у которых в Фильтрах Намерений указано действие Намерения. В случае, если действие в Намерении отсутствует, совпадение происходит для всех Фильтров Намерений, у которых указано хотябы одно действие.

25. Проверка категорий. Все категории, имеющиеся у Намерения, должны

26. присутствовать в Фильтре Намерений. Если у Намерения нет категорий, то на данном этапе ему соответствуют все Фильтры Намерений, за одним исключением, упоминавшимся выше: Активности, запускаемые с помощью метода startActivity,

обязаны иметь категорию android.intent.category.DEFAULT, так как Намерению, использованному в этом случае, по умолчанию присваивается данная категория, даже если разработчик не указал ничего явно. Из этого исключения, в свою очередь, есть исключение: если у Активности присутствуют действие

27. android.intent.action.MA.IN и категория android.intent.category.LAUNCHER, ему не требуетсяиметь категорию android.intent.category.DEFAULT.

28. Проверка данных. Здесь применяются следующие правила:

29. Намерение, не содержащее ни URI, ни типа данных, проходит через Фильтр,если он тоже ничего перечисленного не содержит.

30. Намерение, которое имеет URI, но не содержит тип данных (и тип данных невозможно определить по URI), проходит через Фильтр, если URI Намерения совпадает с URI Фильтра.Это справедливо только в случае таких URI, как mailto: или tel:, которые не ссылаются на реальные данные.

31. Намерение, содержащее тип данных, но не содержащее URI подходит толькодляаналогичных Фильтров Намерений.

32. Намерение, содержащее и тип данных, и URI (или если тип данных может быть вычислен из URI), проходит этот этап проверки, только если его тип данных присутствует в Фильтре. В этом случае URI должен совпадать, либо(!) у Намерения указан URI вида content: или file:,а у Фильтра URI не указан. То есть, предполагается, что если у компонента в Фильтре указантолько тип данных, то он поддерживает URI вида content: или file:.В случае, если после всех проверок остается несколько приложений, пользователю предлагается выбрать приложениесамому. Если подходящих приложений не найдено, в выпустившей Намерение Активностивозникает Исключение.

Заданеи 5.

Модифицируйте проект MetroPicker следующим образом:

1. Добавьте главное меню в Активность, отображающую список станций метро. В меню должен быть один пункт: «вернуться». Меню создайте динамически в коде,без использования строковых ресурсов.

2. Динамически создайте контекстное меню для Представления TextView, отображающего выбранную станцию метро главной Активности. Выбор пункта меню должен сбрасыватьвыбранную станцию.

3. Для главной Активности создайте основное меню из двух пунктов: «сбросить» и

«выйти». Реализуйте нужные функции при выборе этих пунктов.

4. Повторите реализацию п.п. 1, 2 и 3 с помощью ресурсов, описывающих меню.Задание 6.

1. Создайте новый Android проект ListViewSample.

2. В каталоге res/values создайте файл arrays.xml с данными полученными у преподавателя.

3. В каталоге res/layout создайте файл list_item.xml с данными полученными упреподавателя.

4. Модифицируйте метод onCreate вашей Активности: @Override public void onCreate(Bundle savedInstanceState) {super.onCreate(savedlnstanceState);

Resources r = getResources();

String[] stationsArray = r.getStringArray(R.array.stations);

ArrayAdapter<String> aa = new ArrayAdapter<String>(this, R.layout.list item, stationsArray);setListAdapter(aa); ListView

lv = getListView();

}

5. Измените базовый класс Активности с Activity на ListActivity.

6. Запустите приложение.

7. Для реакции на клики по элементам списка требуется добавить обработчик такогособытия, с помощью метода setOnItemClickListener. В качестве обработчика будет

использоваться анонимный объект класса OnItemClickListener. Добавьте следующий код в нужное место:

lv.setOnItemClickListener(new OnItemClickListener() { public void onItemClick(AdapterView<?> parent, View v,int position, long id) {

CharSequence text = ((TextView) v).getText();int duration = Toast.LENGTH_LONG; Context context = getApplicationContext();

Toast.makeText(context, text, duration).show();

}

});

8. Запустите приложение и «покликайте» по станциям метро.

Задание 7. «Использование управляющих элементов в пользовательском интерфейсе»Подготовка

1. Создайте новый проект ControlsSample.

2. Отредактируйте файл res/layout/main.xml так, чтобы остался только корневой элементLinearLayout. В него в дальнейшем будут добавляться необходимые дочерние элементы:

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"

android:layout width="fill parent" android:layout height="fill parent" android:orientation="vertical" >

</LinearLayout>

Использование графической кнопки

Для использования изображения вместо текста на кнопке потребуются три изображения для трехсостояний кнопки: обычное, выбранное («в фокусе») и нажатое. Все эти три изображения с соответствующими состояниями описываются в одном XML файле, который используется для создания такой кнопки.

1. Скопируйте нужные изображения кнопки в каталог res/drawable-mdpi, для обновлениясписка содержимого каталога в Eclipse можно использовать кнопку F5.

2. В этом же каталоге создайте файл smile_button.xml, описывающий, какие изображения в каких состояниях кнопки нужно использовать:

<?xml version="1.0" encoding="utf-8"?>

<selector xmlns:android="http://schemas.android.com/apk/res/android">

<item android:drawable="@drawable/smile_pressed" android:state_pressed="true"/>

<item android:drawable="@drawable/smile_focused" android:state_focused="true"/>

<item android:drawable="@drawable/smile_normal" />

</selector>

3. Добавьте элемент Button внутри LinearLayout в файле разметки res/layout/main.xml:

<Button android:id="@+id/button"

android:layout_width="wrap_content" android:layout_height="wrap_content" android:background="@drawable/smile_button" android:onClick="onButtonClicked" android:padding="10dp" />

4. Обратите внимание на атрибут android:onClick= "onButtonClicked", указывающий, какой методиз Активности будет использоваться как обработчик нажатия на данную кнопку. Добавьте этот методв Активность:

public void onButtonClicked(View v) {

Toast. makeText (this, "Кнопка нажата", Toast. LENGTH_SHORT).show();

}

5. Запустите приложение и посмотрите, как изменяется изображение кнопки в разныхсостояниях, а также как функционирует обработчик нажатия на кнопку.

Использование виджета CheckBox

1. Добавьте элемент CheckBox внутри LinearLayout в файле разметки res/layout/main.xml:

<CheckBox
android:id= "@+id/checkbox"
android:layout_width="wrap_content" android:layout_height="wrap_content" android:onClick="onCheckboxClicked"
android:text="Выбери меня" />

2. Атрибут android:onClick= "onCheckboxClicked" определяет, какой метод из Активностибудет использоваться как обработчик нажатия на виджет. Добавьте этот метод в Активность:

public void onCheckboxClicked(View v) { if (((CheckBox) v).isChecked()) { Toast.makeText(this, "Отмечено", Toast.LENGTH_SHORT).show();
} else {
Toast.makeText(this, "Не отмечено", Toast.LENGTH_SHORT).show();
}
}

3. Запустите приложение и посмотрите на поведение чекбокса в разных ситуациях.Использование виджета ToggleButton

Данный виджет хорошо подходит в качестве альтернативы радиокнопкам и чекбоксам, когда требуется переключаться между двумя взаимоисключающими состояниями, например, Включено/Выключено.

1. Добавьте элемент ToggleButton внутри LinearLayout в файле разметки res/layout/main.xml:

<ToggleButton android:id= "@+id/togglebutton" android:layout_width= "wrap_content" android:layout_height="wrap_content" android:textOn="Звонок включен" android:textOff="Звонок выключен" android:onClick= "onToggleClicked"/>

2. Атрибут android:onClick= "onToggleClicked" определяет, какой метод ИЗ Активности будет использоваться как обработчик нажатия на виджет. Добавьте этот метод в Активность:

public void onToggleClicked(View v) {if (((ToggleButton) v).isChecked()) {
Toast.makeText(this, "Включено", Toast.LENGTH_SHORT).show();
} else {
Toast.makeText(this, "Выключено", Toast.LENGTH_SHORT).show();
}
}

3. Запустите приложение и проверьте его функционирование.Использование виджета RadioButton

Радиокнопки используются для выбора между различными взаимоисключающими вариантами. Для создания группы радиокнопок используется элемент RadioGroup, внутри которого располагаютсяэлементы RadioButton.

1. Добавьте следующие элементы разметки внутри LinearLayout в файле res/layout/main.xml:

<RadioGroup android:layout_width="fill_parent" android:layout_height="wrap_content"android:orientation="vertical" >
<RadioButton
android:id= "@+id/radio_dog"
android:layout_width= "wrap_content" android:layout_height= "wrap_content" android:onClick="onRadioButtonClicked" android:text="Собачка" />
<RadioButton android:id="@+id/radio_cat"

android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:onClick="onRadioButtonClicked" android:text="Кошечка" />
<RadioButton
android:id= "@+id./radio_rabbit"
android:layout_width="wrap_content" android:layout
height="wrap content"
android:onClick="onRadioButtonClicked
android:text="Кролик" />
</RadioGroup>

2. Добавьте метод onRadioButtonClicked в Активность:

public void onRadioButtonClicked(View v) { RadioButton rb = (RadioButton) v;Toast. makeText(this, "Выбрано животное: " + rb.getText(), Toast.LENGTH_SHORT).show();
}

3. Проверьте работу приложения.Использование виджета EditText

Виджет EditText используется для ввода текста пользователем. Установленный для этоговиджета обработчик нажатий на кнопки будет показывать введенный текст с помощью Toast.

1. Добавьте элемент EditText внутри LinearLayout в файле разметки res/layout/main.xml:

<EditText
android:id= "@+id/user_name" android:layout_width="fill_parent" android:layout_height="wrap_content"android:hint="Введите имя"/>

2. Для обработки введенного пользователем текста добавьте следующий код в конце метода onCreate. Обратите внимание, этот обработчик, в отличие от предыдущих, использованных нами, возвращает значение true или false. Семантика этих значений традиционна: true означает, что событие(event) обработано и больше никаких действий не требуется, false означает, что событие не обработано этим обработчиком и будет передано следующим обработчикам в цепочке. В нашем случае реагирование происходит только на нажатие (ACTION_DOWN) кнопки Enter (KEYCODE_ENTER):

final EditText userName = (EditText) findViewById(R.id.user name); userName.setOnKeyListener(new View.OnKeyListener() { @Override
public boolean onKey(View v, int keyCode,
KeyEvent event) { if ((event.getAction() == KeyEvent.ACTION DOWN)
&& (keyCode == KeyEvent.KEYCODE_ENTER))
{ Toast.makeText(getApplicationContext(), userName.getText(), Toast.LENGTH_SHORT).show();
return true;
}
return false;
}
});

3. Запустите приложение и проверьте его работу.

4. Добавьте кнопку «Очистить» в разметку и напишите обработчик, очищающий текстовоеполе (используйте метод setText виджета EditText)

5. Проверьте работу приложения.

Практическое занятие № 5. Подготовка стандартных модулей. Передача данных между модулями. Тестирование и оптимизация мобильного приложения

Цель работы: Создание обработчиков событий. Изучить работу с потоками. Научиться работатьс мультимедиа файлами. Изучить работу с классом AsyncTask

Главный и обычный потоки Асинхронные потоки в Android

AsyncTask-это специальный абстрактный класс, предоставляющий набор методов дляреализации:

- onPreExecute-для размещения инициализирующего кода (UI поток)
- doInBackground-для размещения тяжелого кода, который будет выполняться в другом потоке
- onProgressUpdate-для информирования о прогрессе (UI поток)
- onPostExecute-для обработки результата, возвращенного doInBackground(UI поток) и вспомогательных методов
- isCancelled-для получения информации об отмене задачи
- publishProgress-для перевода сообщения о прогрессе в UI поток с последующим вызовомonProgressUpdate

Последовательность выполнения методов AsyncTask onPreExecute doInBackgroundonPostExecute publishProgress onProgressUpdate

Правила использования AsyncTask:

Объект AsyncTaskдолжен быть создан в UI-потоке

- Метод execute должен быть вызван в UI-потоке
- Метод executeможет быть запущен только один раз
- Не вызывайте методы onPreExecute, doInBackground, опРоз1Execи1еи onProgressUpdateПередача данных в AsyncTask

Объявляем класс

Class MyAsyncTaskextends AsyncTask<String, Integer, Long>{

}

- Первый параметр используется методом doInBackground protectedLongdoInBackground(String... urls)
- Второй параметр используется методом onProgressUpdateprotectedvoidonProgressUpdate(Integer... progress)
- Третий параметр используется методом onPostExecute protectedvoidonPostExecute(Longresult)

Промежуточные данные

Последовательность действий для передачи промежуточных данных в основной потокпрограммы:

- В методе doInBackgroundвызываем метод publishProgress
- В методе onProgressUpdate обрабатываем переданный в publishProgress параметр и выводимпрогресс

Метод get

- Возвращает результат выполнения метода doInBackground
- Вызывается из UI потока MyAsyncTaskat = newMyAsyncTask();result= at.get();

Задание 1. Разработать мобильное приложение, состоящее из четырех activity.

После запуска приложения пользователь должен попадать на экран с activityt. На данном экранедолжно быть представлено меню, состоящее из четырех кнопок. Высота кнопок должна составлять 20% от высоты экрана. Расстояние между кнопками - 2%. Первая и последняя кнопка должны быть наравном расстоянии от краев экрана. Ширина кнопок 75%, выравнивание посередине.

После нажатия на первую кнопку пользователь должен переходить к activity2, его внешний вид представлен на рисунке 64. Верстка должна осуществляться с

использованием LinearLayout, ширина кнопок должна задаваться в процентах от ширины экрана.

Рисунок 64

После нажатия на вторую кнопку в activity1 пользователь должен переходить к activity3, его внешний вид представлен на рисунке 65. Верстка должна осуществляться с использованием RelativeLayout (не использовать LinearLayout).

Рисунок 65

Третья кнопка в activity1 должна создавать activity3. Внешний вид activity3 представлен на рис.66

Рисунок 66

Кнопка должна быть выровнена по центру экрана. Цвет обводки кнопки #505050. Толщина обводки в соответствии с месяцем вашего рождения (от 1 до 12). Радиус

скругления 24dp. Цвет фона экрана #FFFFFF. При нажатии на кнопку ее цвет должен изменяться на светло-зеленый.

Нажатие на четвертую кнопку в activity1 должно приводить к закрытию приложения. Задание 2. Рассмотрите пример передачи данных. MyAsyncTaskat = newMyAsyncTask();at.execute("url1", "url2");

doInBackground(String... urls)

Задание 3. Рассмотрите пример вывода промежуточных данных.@Override

Protected void doInBackground(String... urls) { try{int cnt= 0;

for(Stringurl: urls) {

// обрабатываем первый параметр

// выводим промежуточные результатыcnt++;

publishProgress(cnt);

}

TimeUnit.SECONDS.sleep(l);

} catch(InterruptedExceptione) {e.printStackTrace();

}

return null;

}

@Override

protected void onProgressUpdate(Integer... values) {

super.onProgressUpdate(values); tv.setText("o6pa6oiaHO " + values[0] + " параметров");

}

Задание 4. Проверьте пример создания простой асинхронной задачиpublic class MainActivity extends Activity{ MyAsyncTask at; TextView tv;

public void onCreate(Bundle savedInstance State) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

tv= (TextView) findViewById(R.id.tv); MyAsyncTask at = new MyAsyncTask(); at.execute();

}

}

Задание 5. На основании изученных примеров разработать приложение сохраняющее статистику проигрываемых песен на радио Мегабайт. Для сохранения песни и названия необходимо создать базуданных, содержащую таблицу со следующими полями:

– ID
– Исполнитель
– Название трека
– Время внесения записи

При включении приложения необходимо производить проверку подключения к Интернету. В случае если подключение отсутствует - выводить всплывающее сообщение (Toast) с предупреждениемо запуске в автономном режиме (доступен только просмотр внесенных ранее записей).

После включения приложение должно производить асинхронный опрос сервера с интервалом 20секунд. Если название трека не совпадает с последней записью в таблице необходимо произвести запись в БД.

URL адрес, по которому можно получить информацию о текущем треке и исполнителе:http://media.ifmo.ru/api get current song.php

Формат возвращаемых данных - JSON.

В случае успешного выполнения запроса результат будет иметь вид:

{"result": "success", "info" : “Исполнитель - Название трека" }В случае ошибки API вернет следующую строку:

{"result": "error", "info" : "Информация об ошибке" }

Для успешного взаимодействия с API при обращении к странице необходимо передавать логин(login) и пароль(ра88’№огй) как POST- параметры.

login: 4707login password: 4707pass

Приложение должно содержать активити, позволяющее просматривать внесенные в базу данныхзаписи.

Задание 6. Работа с внешними файлами.

Разработать мобильное приложение, позволяющее пользователю асинхронно скачивать файлы журнала Научно-технический вестник. Файлы хранятся на сервере в формате PDF и расположены по адресу: http://ntv.ifmo.ru/flle/journal/идентификаторжурналафйГ

Не для всех ID имеются журналы, поэтому необходимо предусмотреть сообщение об отсутствиифайла. В случае если файл не найден, ответ от сервера будет содержать главную страницу сайта.

Определить существует ли файл можно по возвращаемому сервером заголовку (параметр content-type).

Примеры ссылок:

http://ntv.ifmo.ru/file/journal/1.pdf - возвращен PDF файл http://ntv.ifmo.ru/file/journal/2.pdf - файл не найден, возвращена главная страница сайта

Файлы должны храниться на устройстве в папке, создаваемой при первом запуске приложения(путь до папки и ее название определите самостоятельно).

После окончания загрузки файла должна становиться доступной кнопка «Смотреть» и кнопка

«Удалить».

При нажатии на кнопку «Смотреть» должно происходить открытие сохраненного на устройстве файла. Предусмотреть ошибку, если на устройстве не установлено приложение, открывающее PDF файлы. При нажатии на кнопку «Удалить» загруженный файл должен удаляться с устройства.

МДК 01.04 Системное программирование

Практическое занятие № 1. Использование потоков.

Цель работы: *ознакомление с общими принципами построения программы на языкеассемблер.*

Ход работы:

1. Изучить теоретическую часть (дополнительная информация в файле Структура COM иEXE файлов.docx)
2. Выполнить задание в соответствии с указаниями
3. Ответить на контрольные вопросы
4. Предъявить преподавателю результаты работы: проект и исходный код
5. Оформить отчет в соответствии с ходом работы

Теоретическая часть:

1.1 Структура ассемблерной программы

Каждый язык программирования имеет свои особенности. Язык ассемблера - не исключение. Традиционно первая программа выводит приветственное сообщение на экран ‘Hello World’.

В отличие от многих современных языков программирования в ассемблерной программе каждая команда располагается на ОТДЕЛЬНОЙ СТРОКЕ. Нельзя разместить несколько команд на одной строке. Не принято, также, разбивать одну команду на несколько строк. Язык ассемблера является РЕГИСТРОНЕЧУВСТВИТЕЛЬНЫМ.Т. е. в большинстве случаев нет разницы между большими и малыми буквами. Команда может быть ДИРЕКТИВОЙ - указанием транслятору. Они выполняются в процессе превращения программы в машинный код. Многие директивы начинаются с точки. Для удобства чтения программы они обычно пишутся БОЛЬШИМИ БУКВАМИ. Кроме

директив еще бывают ИНСТРУКЦИИ - команды процессору. Именно они и будут составлять машинный код программы.

1.2 Процесс обработки программы на языке ассемблера

Весь процесс технического создания ассемблерной программы можно разбить на 4 шага(исключены этапы создания алгоритма, выбора структур данных и т.д.).

Набор программы в текстовом редакторе и сохранение ее в отдельном файле. Каждый файл имеет имя и тип, называемый иногда расширением. Тип в основном используетсядля определения назначения файла. Например, программа на С имеет тип С, на Pascal -PAS, на языке ассемблера - ASM.

Обработка текста программы транслятором. На этом этапе текст превращается в машинный код, называемый объектным. Кроме того, есть возможность получить листингпрограммы, содержащий кроме текста программы различную дополнительную информацию и таблицы, созданные транслятором. Тип объектного файла - OBJ, файла листинга - LST. Этот этап называется ТРАНСЛЯЦИЕЙ.

Обработка полученного объектного кода компоновщиком. Тут программа "привязывается" к конкретным условиям выполнения на ЭВМ. Полученный машинный код называется выполняемым. Кроме того, обычно получается карта загрузки программы в ОЗУ. Выполняемый файл имеет тип EXE, карта загрузки - MAP. Этот этап называется КОМПОНОВКОЙ или ЛИНКОВКОЙ Запуск программы. Если программа работает не совсем корректно, перед этим может присутствовать этап ОТЛАДКИ программы при помощи специальной программы - отладчика. При нахождении ошибки приходится проводить коррекцию программы, возвращаясь к шагу 1. Таким образом, процесс создания ассемблерной программы можно изобразить в виде следующей схемы. Конечной целью, напомним, является работоспособный выполняемый файл HELLO. EXE.

1.3 Особенности создания ассемблерной программы в среде эмулятора EMU8086

Этот программный продукт содержит все необходимое для создания программы на языкеAssembler.

Пакет Emu8086 сочетает в себе продвинутый текстовый редактор, assembler, disassembler,эмулятор программного обеспечения (Виртуальную машину) с пошаговым отладчиком, примеры.

1.4 Правила оформления ассемблерных программ

При наборе программ на языке ассемблера придерживайтесь следующих правил: директивы набирайте большими буквами, инструкции - малыми;

пишите текст широко - не скупердяйничайте;

не выходите за край экрана, т.е. не делайте текст шире 80 знаков - его не удобно будетредактировать и печатать;

для отступов пользуйтесь табуляцией (клавиша TAB);

блоки комментариев задавайте с одинаковым отступом. Оптимальной считается такаястрока:

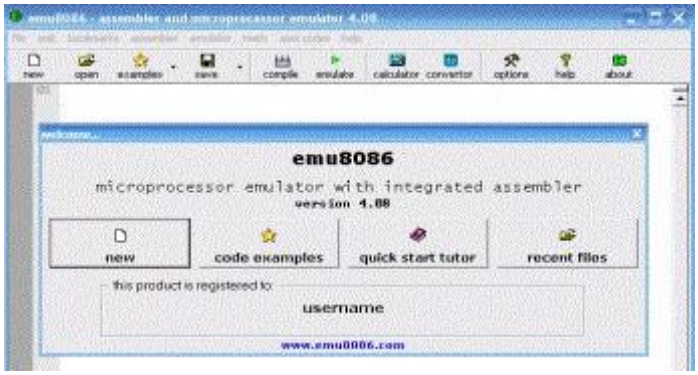
movax,<пробел>bx< (1-3) TAB>; <пробел>текст комментария

Количество табуляций перед комментарием определяется длиной аргументов команды и может быть от 1 до 3. По мере знакомства с синтаксисом языка будут приводиться дополнительные правила.

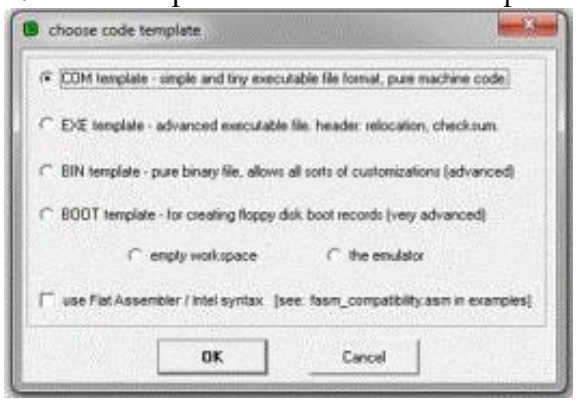
Задание 1. Работа с эмулятором Emu8086

Emu8086 сочетает в себе мощный редактор исходного кода, ассемблер, дизассемблер, программный эмулятор (виртуальный ПК) с отладчиком и поэтапное обучение. Эта программа компилирует исходный код и выполняет его с помощью эмулятора шаг за шагом.

1. Запустите эмулятор и щелкните на кнопку new



2. Выберите тип исполняемого файла:



Директивы, определяющие тип исполнимого файла:

#MAKE_COM# #MAKE_BIN# #MAKE_BOOT##MAKE_EXE#

Вы можете вставить эти директивы в исходный код для определения нужного вам типаисполнимого файла.

Описание типов исполнимых файлов:

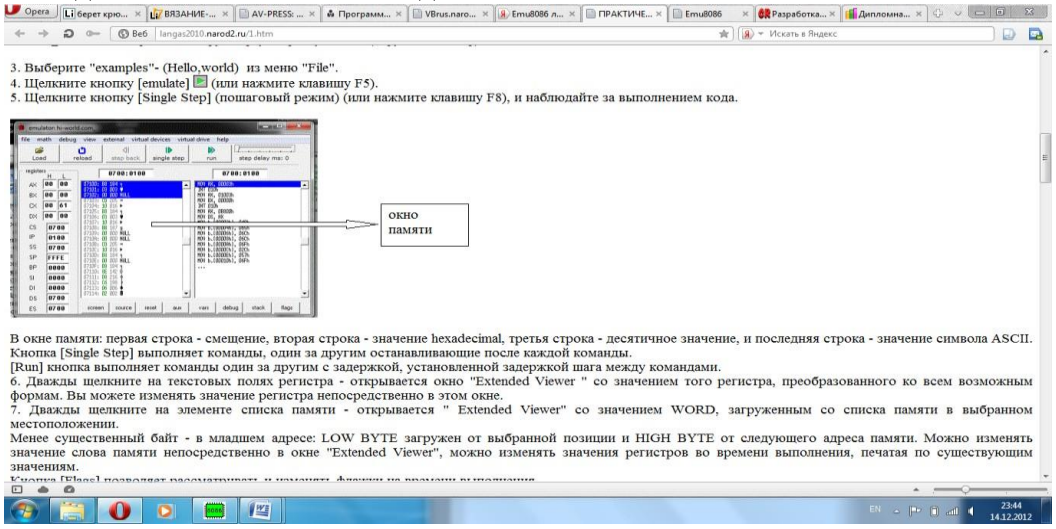
#MAKE_COM# - самый старый и самый простой формат исполнимого файла. Такиефайлы загружаются с префиксом 100h (256 байтов).

#MAKE_EXE# - более "продвинутый" формат исполнимого файла. Не ограничены размери количество сегментов.

#MAKE_BIN# - простой исполнимый файл.

#MAKE_BOOT# - эта директива копирует первую дорожку дискеты (загрузочный сектор).

3. Выберите "examples"- (Hello,world) из меню "File".
4. Щелкните кнопку [emulate] (или нажмите клавишу F5).
5. Щелкните кнопку [Single Step] (пошаговый режим) (или нажмите клавишу F8), инаблюдайте за выполнением кода.



В окне памяти: первая строка - смещение, вторая строка - значение hexadecimal, третьястрока - десятичное значение, и последняя строка - значение символа ASCII.

Кнопка [Single Step] выполняет команды, один за другим останавливающие после каждойкоманды.

[Run] кнопка выполняет команды один за другим с задержкой, установленной задержкойшага между командами.

6. Дважды щелкните на текстовых полях регистра - открывается окно "Extended Viewer "со значением того регистра, преобразованного ко всем возможным формам. Вы можете изменять значение регистра непосредственно в этом окне.

7. Дважды щелкните на элементе списка памяти - открывается " Extended Viewer" со значением WORD, загруженным со списка памяти в выбранном местоположении. Менее существенный байт - в младшем адресе: LOW BYTE загружен от выбранной позиции и HIGH BYTE от следующего адреса памяти. Можно изменять значение слова

памяти непосредственно в окне "Extended Viewer", можно изменять значения регистров вовремени выполнения, печатая по существующим значениям.

Кнопка [Flags] позволяет рассматривать и изменять флажки на времени выполнения.

В окне эмулятора вы можете запустить вашу программу на выполнение целиком (кнопкаRUN) либо в пошаговом режиме (кнопка SINGLE STEP). Пошаговый режим удобен для отладки. Ну а мы сейчас запустим программу на выполнение кнопкой RUN. После этого (если вы не сделали ошибок в тексте программы) вы увидите сообщение о завершении программы (рис. 1.3). Здесь вам сообщают о том, что программа передала управление операционной системе, то есть программа была успешно завершена. Нажмите кнопку ОКв этом окне и вы увидите, наконец, результат работы вашей первой программы на языке ассемблера (рис. 1.4).

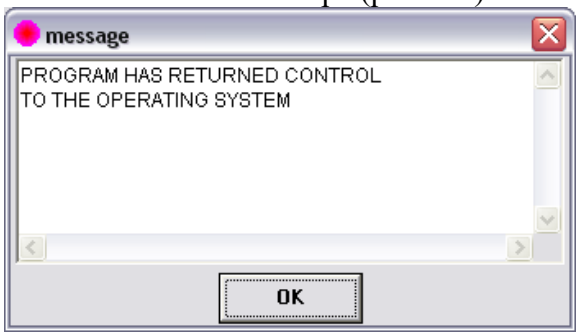


Рис. 1.3. Сообщение о завершении программы.

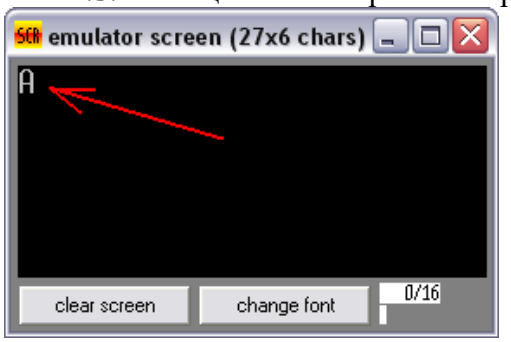
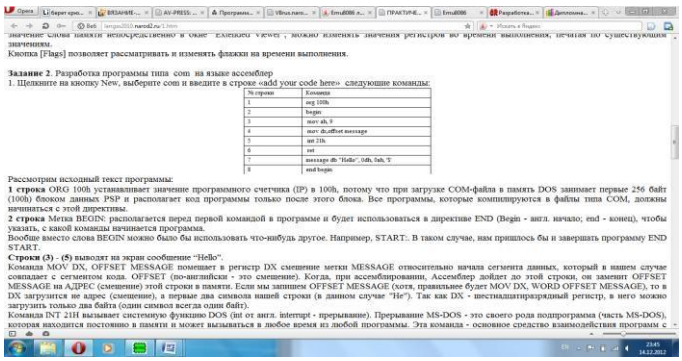


Рис. 1.4. Ваша первая программа выполнена.

Задание 2. Разработка программы типа com на языке ассемблер

1. Щелкните на кнопку New, выберите com и введите в строке «add your code here»следующие команды:



Рассмотрим исходный текст программы:

1 строка ORG 100h устанавливает значение программного счетчика (IP) в 100h, потомучто при загрузке COM-файла в память DOS занимает первые 256 байт (100h) блоком данных PSP и располагает код программы только после этого блока. Все программы, которые компилируются в файлы типа COM, должны начинаться с этой директивы.

2 строка Метка BEGIN: располагается перед первой командой в программе и будет использоваться в директиве END (Begin - англ. начало; end - конец), чтобы указать, скакой команды начинается программа.

Вообще вместо слова BEGIN можно было бы использовать что-нибудь другое. Например, START:. В таком случае, нам пришлось бы и завершать программу END START.

Строки (3) - (5) выводят на экран сообщение “Hello”.

Команда MOV DX, OFFSET MESSAGE помещает в регистр DX смещение метки MESSAGE относительно начала сегмента данных, который в нашем случае совпадает с сегментом кода. OFFSET (по-английски - это смещение). Когда, при ассемблировании, Ассемблер дойдет до этой строки, он заменит OFFSET MESSAGE на АДРЕС (смещение) этой строки в памяти. Если мы запишем OFFSET MESSAGE (хотя, правильнее будет MOV DX, WORD OFFSET MESSAGE), то в DX загрузится не адрес (смещение), а первые два символа нашей строки (в данном случае "Не"). Так как DX - шестнадцатиразрядный регистр, в него можно загрузить только два байта (один символ всегда один байт).

Команда INT 21H вызывает системную функцию DOS (int от англ. interrupt - прерывание). Прерывание MS-DOS - это своего рода подпрограмма (часть MS-DOS), которая находится постоянно в памяти и может вызываться в любое время из любой программы. Эта команда

- основное средство взаимодействия программ с операционной системой. В примере вызывается функция DOS (строка 7) - вывести строку на экран. Эта функция выводит строку от начала, адрес которого задается в регистрах DS: DX, до первого встречного символа $. При запуске COM-файла регистр DS автоматически загружается сегментнымадресом программы, а регистр DX был подготовлен предыдущей командой.

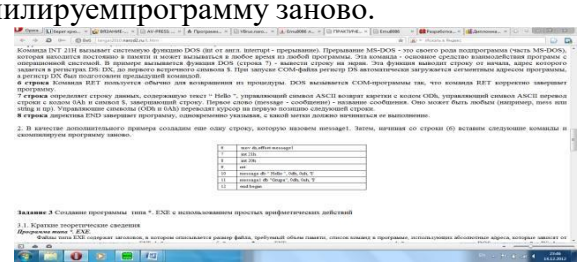
6 строка Команда RET пользуется обычно для возвращения из процедуры. DOS вызывается COM-программы так, что команда RET корректно завершает программу.

7 строка определяет строку данных, содержащую текст “ Hello ", управляющий символ ASCII возврат каретки с кодом ODh, управляющий символ ASCII перевод строки с кодом0Ah и символ $, завершающий строку. Первое слово (message - сообщение) - название сообщения. Оно может быть любым (например, mess или string и пр). Управляющие символы (ODh и 0Ah) переводят курсор на первую позицию следующей строки.

8 строка директива END завершает программу, одновременно указывая, с какой меткидолжно начинаться ее выполнение.

2. В качестве дополнительного примера создадим еще одну строку, которую назовем message1. Затем, начиная со строки (6) вставим следующие команды и

скомпилируемпрограмму заново.



Задание 3. Создание программы типа *. EXE с использованием простых арифметических действий

Простые арифметические операторы.

1. Сложение.

Команда ADD (Addition - сложение (гл. to add - сложить)) осуществляет сложение первого и второго операндов. Исходное значение первого операнда (приемника) теряется, замещаясь результатом сложения. Второй операнд не изменяется. В качестве первого операнда команды ADD можно указывать регистр (кроме сегментного) или ячейку памяти, в качестве второго - регистр (кроме сегментного), ячейку памяти или непосредственное значение, однако не допускается определять оба операнда одновременно как ячейки памяти. Операнды могут быть байтами или словами и представлять числа со знаком или без знака. Команду ADD можно использовать для

сложения как обычных целых чисел, так и двоично-десятичных (с использованием регистра AX для хранения результата). Команда воздействует на флаги OF, SF, ZF, AF, PFи CF.

Примеры:

mov al,10; загружаем в регистр AL число 10 add al,15; al = 25; al - приемник, 15 - источник

mov ax,25000; загружаем в регистр AX число 25000

add ax,10000; ax = 35000; ax - приемник, 10000 - источникmov cx, 200; загружаем в регистр CX число 200

mov bx,760;а в регистр BX - 760

add cx,bx; cx = 960, bx = 760 (bx не меняется); cx - приемник, bx - источник

2 Вычитание.

Команда SUB (Subtraction - вычитание) вычитает второй операнд (источник) из первого (приемника) и помещает результат на место первого операнда. Исходное значение первого операнда (уменьшаемое) теряется. Таким образом, если команду вычитания записать в общем виде

SUB операнд1, операнд2

то ее действие можно условно изобразить следующим образом: операнд1 - операнд2 - > операнд1

В качестве первого операнда можно указывать регистр (кроме сегментного) или ячейку памяти, в качестве второго - регистр (кроме сегментного), ячейку памяти или непосредственное значение, однако не допускается определять оба операнда одновременно как ячейки памяти. Операнды могут быть байтами или словами и представлять числа со знаком или без знака. Команда воздействует на флаги OF, SF, ZF, AF, PF и CF.

Примеры:

mov al,10

sub al,7 - --> al = 3; al - приемник, 7 - источникmov ax,25000

sub ax,10000 - --> ax = 15000; ax - приемник, 10000 - источникmov cx,100
mov bx,15
sub cx,bx - --> cx = 85, bx = 15 (bx не меняется); cx - приемник, bx - источник

3 Инкремент (увеличение на 1).

Команда INC (Increment - инкремент) прибавляет 1 к операнду, в качестве которого можноуказывать регистр (кроме сегментного) или ячейку памяти размером как в байт, так и в слово.

Не допускается использовать в качестве операнда непосредственное значение. Операндинтерпретируется как число без знака. Команда воздействует на флаги OF, SF, ZF, AF иPF. Команда не воздействует на флаг CF; если требуется воздействие на этот флаг, необходимо использовать команду Add Op,l.

Команда INC (Increment - инкремент) увеличивает на единицу регистр или значениеоперанда в памяти.

Она эквивалентна команде ADD источник, 1 только выполняется гораздо быстрее.

Примеры:

mov al,15
inc al - --> теперь AL = 16 (эквивалентна add al,1)mov dh,39h
inc dh - --> DH = 3Ah (эквивалентна add dh,1)mov cl,4Fh
inc cl - --> CL = 50h (эквивалентна add cl,1)

4 Декремент (уменьшение на 1).

Команда DEC (Decrement - декремент) вычитает 1 из операнда, в качестве которого можноуказывать регистр (кроме сегментного) или ячейку памяти размером как в байт, так и в слово. Не допускается использовать в качестве операнда непосредственное значение.

Операнд интерпретируется как число без знака. Команда воздействует на флаги OF, SF,ZF, AF и PF.

Она эквивалентна команде SUB источник, 1 только выполняется гораздо быстрее.

Примеры:

mov al,15
dec al - --> теперь AL = 14 (эквивалентна sub al,1)mov dh,39h
dec dh - --> DH = 38h (эквивалентна sub dh,1)mov cl,4Fh
dec cl - --> CL = 4Dh (эквивалентна sub cl,1)

3. 2. Создание программ производящих сложение и вычитание.

1. Щелкните на кнопку New, выберите Exe и введите в строке «add your code here» следующие команды для расчета выражения:

((5 + (AL – AH)) – BH) + BL где AL=9, BL=3, AH=4, BH=6.

: (№ + AL) – (BH + AH) – BL

```
mov al, 9
mov ah, 4
sub al, ah
mov dl, 5
add al, dl
mov bh, 6
sub al, bh
mov bl, 3
add al, bl
```

Для вывода результирующего значения в 10-ричной системе счисления добавьте следующий код:

```
mov bx, 0
mov bl, al
 mov ax, bx
push -1
mov cx,10
l:mov dx,0
div cx
push dx
cmp ax,0
jne l
mov ah,2h
l2:pop dx
cmp dx,-1
je ex
add dl,'0'
int 21h
jmp l2
ex:mov ax,4c00h
int 21h
end start
```

Контрольные задания:

Произвести расчет выражений:

1) (№ + AL) – (BH + AH) – BL

2) ((BH + (№ – AH)) – BL) + AL

где № - порядковый номер по журналу, AL=9, BL=3, AH=4, BH=6.

Контрольные вопросы:

1. Характеристика структуры файла типа *.com?
2. Какова структура ассемблерной программы?
3. С какой целью в код программы на ассемблере для DOS вводится строка ORG 100h?
4. Назначение команды MOV?
5. Прерывания 21h и 20h. Назначение?
6. Сущность и целесообразность использования команды RET вместо прерывания 20h?
7. Символ ‘$’ методика применения?
8. Связка “BEGIN: - END BEGIN". Правила применения?
9. Каковы этапы получения выполняемого файла?

Практическое занятие № 2. Обмен данными.

Цель работы: *ознакомление с общими принципами построения программы на языке ассемблер.*

Пример текста программы, которая на языке ассемблер в заданном массиве определяет элемент с максимальной величиной.

```
SGSTACK SEGMENT PARA STACK 'STACK'DB 32 DUP(?)
SGSTACK ENDS

DATA SEGMENT PARA PUBLIC 'DATA'MAX DW ?
ARRAY DW 10H, 20H, 30H, 0D0H,0A0HDATA ENDS

CODE SEGMENT PARA PUBLIC 'CODE' ASSUME CS:CODE, DS:DATA,
SS:SGSTACK
```

```
START: MOV AX, DATA ; загрузить в DS
MOV DS, AX ;селектор сегмента данных
LEA BX, ARRAY ; загрузить в BX начальный адрес массива MOV CX, 4 ; инициализировать счетчик
MOV AX, [BX] ; инициализировать начальное значение max

CYCLE: ADD BX, 2 ; перейти к следующему элементу массива CMP [BX], AX ; сравнить два значения
JBE BE ; переход если равен или ниже MOV AX, [BX] ; сохранить большее значение
BE: LOOP CYCLE ; проверка на выход из цикла (--CX при CX=0) MOV MAX, AX
; сохранение максимального значения

EXIT: XOR AL, AL ; выход в OS MOV AH, 4CH
INT 21H CODE ENDS END START
```

Порядок выполнения работ

Для выполнения лабораторной работы необходимо:

1. Получить вариант задание у преподавателя из таблицы
2. Составить программу согласно заданному варианту
3. Получить файл с исходным текстом программы в EXE и COM формате
4. Оттранслировать, отладить программу. Изучить листинг программы.
5. Скомпоновать выполняемый файл, изучить карту загрузки (порядок следованиясегментов, их размеры и относительные адреса)
6. Запустить программу под отладчиком.
7. Оформить отчет

Практическое занятие № 3. Сетевое программирование сокетов.

Цель работы

Изучить:

- возможности реализации архитектуры клиент-сервер на основе интерфейса сокетов Windows Sockets API;
- типы сокетов TCP/IP;
- основные методики и API-функции для разработки сетевых приложений с использованием Winsock.

Постановка задачи

1. Изучить методические указания к лабораторной работе, материалы лекций и рекомендуемую литературу.
2. Разработать консольное клиент-серверное приложение, демонстрирующеевзаимодействие на основе потоковых сокетов.
3. Разработать консольное клиент-серверное приложение, демонстрирующеевзаимодействие на основе дейтаграммных сокетов.

Методические указания

1. Понятие сокета

Сокет (Socket - гнездо, разъем) - абстрактное программное понятие, используемое для обозначения в прикладной программе конечной точки канала связи с коммуникационной средой, образованной вычислительной сетью.

Соединяя вместе два сокета, можно передавать данные между разными процессами (локальными и удаленными). Реализация сокетов обеспечивает инкапсуляцию протоколов сетевого и транспортного уровней.

Интерфейс, используемый приложением при взаимодействии с программным обеспечением транспортного протокола, называется ***интерфейсом прикладного программирования*** (Application Programming Interface - API). API интерфейс определяет набор операций, которые могут быть выполнены приложением при взаимодействии с программным обеспечением протокола.

Функции прикладного программного интерфейса сокетов (Sockets API) обеспечивают идентификацию конечных точек соединения, установку соединения, отправку сообщений, ожидание входящих сообщений, разрыв соединения и обработку ошибок.

2. Протоколы и семейства адресов

Важнейшим преимуществом сокетов Windows является предоставление единого независимого интерфейса сетевого программирования (Sockets API) для различных сетевых протоколов.

Платформы Win32 поддерживают разнообразные сетевые стеки протоколов : TCP/IP, IPX/SPX, NetBIOS/SMB, AppleTalk, ATM, Infrared Sockets. Каждому из них соответствует свое семейство адресов сокетов. Напрмер, TCP/IP соответствует семейство адресов AF_INET, IPX/SPX – AF_NS, ATM –AF_ATM и т.д.

Семейство адресов – важнейший параметр сокета. Он указывает используемый внастоящее время протокол и ограничивает применение других параметров сокета.

Мы рассмотрим адресацию сокетов только для стека протоколов TCP/IP, как самого распространенного на сегодняшний день.

Адрес сокета при использовании протоколов TCP/IP задает следующий набор значений:

- номер сети;
- номер конечного узла;
- номер порта прикладной службы.

3. Инициализация Winsock

Перед вызовом любой функции Winsock необходимо загрузить соответствующую версию библиотеки Winsock. Для использования в приложении Winsock 2 необходимо подключить библиотеку Ws2_32.lib и заголовочный файл Winsock2.h.

Имена новых или обновленных API-функций, добавленные в Winsock 2, начинаются с префикса WSA.

Инициализацию Winsock выполняет функция *WSAStartup:*

int WSAStartup(

WORD wVersionRequested,LPWSADATA IpWSAData);

Параметр *wVersionRequested* задает версию загружаемой библиотеки Winsock. На современных платформах Win32 используется версия 2.2. Для получения значения параметра *wVersionRequested* можно использовать макрос *MAKEWORD(2, 2)* либо значение 0x0202.

Параметр *IpWSAData* — указатель на структуру *LPWSADATA,* которая при вызове функции *WSAStartup* заполняется сведениями о версии загружаемой библиотеки.

По завершении работы с библиотекой Winsock необходимо вызвать функцию *WSACleanup* для выгрузки библиотеки и освобождения ресурсов:int WSACleanup (void);

4. Адресация сокетов для протокола IP

Для задания IP-адреса и номера порта службы используется структура

SOCKADDR_IN. Она определена в include-файле in.h следующим образом*:* struct sockaddr_in {

short sin_family;

u_short sin_port; struct in_addr sin_addr; char sin_zero[8];

};

Поле *sin_family* при использовании семейства адресов IP должно быть равно

AF_INET.

Поле *sin_port* задает, какой коммуникационный порт будет использован для идентификации службы.

Поле *sin_addr* структуры *SOCKADDR_IN* хранит IP-адрес в 4-байтном виде с типом unsigned long int. В зависимости от того, как это поле использовано, оно может представлять и локальный, и удаленный IP-адрес.

Поле *sin_zero* играет роль заполнителя, чтобы структура *SOCKADDR_IN* по размеру равнялась структуре *SOCKADDR.*

5. Специальные IP-адреса

Специальный адрес *INADDR_ANY (0.0.0.0)* позволяет приложению слушать клиента через любой сетевой интерфейс на несущем компьютере. Обычно приложения сервера используют этот адрес, чтобы привязать сокет к локальному интерфейсу для прослушивания соединений. Если на компьютере несколько сетевых адаптеров, то этот адрес позволит отдельному приложению получать отклики от нескольких интерфейсов.

Второй специальный адрес – *INADDR_BROADCAST (255.255.255.255),* позволяет широковещательно рассылать пакеты по IP – сети. Для его использования необходимо в приложении задать параметр сокета *SO_BROADCAST*.

6. Порядок байт

В памяти компьютера IP-адрес и номер порта представляются в системном порядке (host-byte-order). Для процессоров Intel Pentium это порядок от менее значимого к более значимому байту (обратный). Стандарты Internet требуют, чтобы многобайтные значения передавались от старшего байта к младшему, что называется сетевым порядком (network-byte order) или прямым порядком следования байтов. Поэтому существует необходимость преобразования чисел из одной формы в другую. Для этого предназначен целый ряд функций. Например, функции htonl (Host TO Network Long), WSAHtonl, htons (Host TO Network Short), WSAHtons преобразуют числа из системного порядка в сетевой. Функции ntohl, WSANtohl, ntohs, WSANtohs решают обратную задачу (из сетевого в системный).

Полезная вспомогательная функция *inet_addr* преобразует IP-адрес из точечно-десятичной нотации в 32-битное длинное целое без знака с сетевым порядком следования байт:unsigned long inet_addr(const char FAR *cp);

Поле *cp* - строка, заканчивающаяся нулевым символом, в которой задается IP-адрес вточечно-десятичной нотации.

7. Разрешение имен

В Winsock предусмотрено две функции для разрешения имени в IP-адрес.

Функции gethostbyname и WSAAsyncGetHostByName отыскивают в базе данных узла сведения об узле, соответствующие его имени. Обе функции возвращают структуру *HOSTENT*:

struct hostent

{

char FAR * h_name; char FAR * FAR * h_aliases; short h_addrtype;

short h_length;

char FAR * FAR * h_addr_list;

};

Поле *h_name* является официальным именем узла. Если в сети используется *доменная система имен* (Domain Name System, DNS), в качестве имени сервера будет

возвращено полное имя домена (Fully Qualified Domain Name, FQDN). Если в сети для разрешения имен применяется локальный файл (hosts, lmhosts) - это первая запись после IP-адреса.

Поле *h_aliases* – массив дополнительных имен узла, завершающийся нулем.Поле *h_addrtype* – возвращаемое семейство адресов.

Поле *h_length* определяет длину в байтах каждого адреса из поля *h_addr_list*.

Поле *h_addr_list* – массив, завершающийся 0 и содержащий IP-адреса узла. (Узел может иметь несколько IP-адресов). Каждый адрес в этом массиве представлен в сетевом порядке. Обычно приложение использует первый адрес из массива. Впрочем, при получении нескольких адресов, приложение должно выбирать адрес случайным образом из числа доступных, а не упорно использовать первый.

API–функция *gethostbyname* определена так:

struct hostent FAR *gethostbyname (const char FAR *name);

Параметр *name* представляет дружественное имя искомого узла. При успешном выполнении функции возвращается указатель на структуру HOSTENT, которая хранится в системной памяти. Приложение не должно полагать, что эти сведения непременно статичны. Поскольку эта память обслуживается системой, оно не должно освобождать возвращенную структуру.

WSAAsyncGetHostByName – асинхронная версия функции *gethostbyname*, оповещающая приложение о завершении своего выполнения с помощью сообщений Windows:

HANDLE WSAAsyncGetHostByName (HWND hWnd,
unsigned int wMsg, const char FAR *name,char FAR * buf,
int buflen);

Параметры *hWind* – дескриптор окна, которое получит сообщение по завершении выполнения асинхронного запроса. Параметр *wMsg* –Windows-сообщение, которое будет возвращено по завершении выполнения асинхронного запроса. Параметр *name* - дружественное имя искомого узла. Параметр *buf* – указатель на область данных, куда помещается *HOSTENT*. Этот буфер должен быть больше структуры *HOSTENT* и иметь размер, определенный в MAXGETHOSTSTRUCT.

Для преобразования IP-адреса в имя узла используются функции *gethostbyaddr* и

WSAAsyncGetHostByAddr. Функция *gethostbyaddr* определена так:struct HOSTENT FAR * gethostbyaddr(
const char FAR * addr,int len,
int type);

Параметр *addr* – указатель на IP–адрес в сетевом порядке.Параметр *len* задает длину параметра *addr* в байтах.

Параметр *type* должен иметь значение AF_INET (IP-адрес).

Функция *WSAAsyncGetHostByAddr* – асинхронная версия функции *gethostbyaddr*.

8. Номера портов

Приложения должны быть внимательны при выборе номера порта, поскольку некоторые доступные порты зарезервированы для использования популярными службами, такими как FTP, HTTP и т.д. Эти порты обслуживаются и распределяются центром Internet Assigned Numbers Authority (IANA), их описание содержится в RFC 1700.

Номера портов разделяются на 3 категории (стандартные, зарегистрированные и динамические и/или частные):

- Номера от 0 до 1023 зарезервированы для стандартных служб.
- Порты с номерами от 1024 до 49151 являются регистрируемыми. Они используются для различных целей.
- Порты с номерами от 49152 до 65535 представляют собой динамические и частные порты.

Во избежание накладок с портами, уже занятыми системой или другим приложением, ваша программа должна выбирать порты, начиная с 1024. Можно вместо конкретного номера порта задать 0, тогда система сама выберет произвольный неиспользуемый в данный момент номер.

Узнать номера портов, используемых стандартными службами, можно вызвав функцию *getservbyname* или *WSAAsyncGetServByName*. Эти функции извлекают статическую информацию из файла services, расположенного в папке

%WINDOWS%\System32\Drivers\Etc Функция *getservbyname* определена так:

struct servent FAR * getservbyname(const char FAR *name,

const char FAR *proto);

Параметр *name* представляет имя искомой службы.

Параметр *proto* ссылается на строку, указывающую протокол, под которым зарегистрирована служба из параметра *name*.

Структура servent имеет системный порядок следования байт.

Функция *WSAAsyncGetServByName* – асинхронная версия *getservbyname.*

9. Типы сокетов

Существуют три основных типа сокетов: потоковые, дейтаграммные и сырые.

Потоковые сокеты – это сокеты с установлением соединения, состоящие из потока байтов, который может быть двунаправленным. Т.е. через такую конечную точку приложение может и передавать, и получать данные. Потоковый сокет гарантирует обнаружение и исправление ошибок, обрабатывает доставку и сохраняет последовательность данных. Он подходит для передачи больших объемов данных, поскольку в этом случае накладные расходы, связанные с установлением соединения, незначительны по сравнению со временем передачи самого сообщения. Качествопередачи достигается за счет использования протокола TCP.

Дейтаграммные сокеты – это сокеты без установления соединения, не обеспечивающие надежность при передаче. Применяются для приложений, когда

неприемлемы затраты времени, связанные с установлением явного соединения. Для передачи данных используется протокол UDP.

Сырые сокеты (необрабатываемые, простые) – это сокеты, которые принимают пакеты сетевого уровня в обход протоколов транспортного уровня и отправляют их непосредственно приложению.

10. Коммуникационная модель клиент-сервер

Приложение, использующее сокеты, состоит из распределенной программы, исполняемой на обоих концах канала связи. Программу, инициирующую передачу, называют ***клиентом***. Приложение на другом конце соединения, называемое ***сервером***, представляет собой модуль, пассивно ожидающий входящих запросов на установку соединений от удаленных клиентов. Как правило, серверное приложение загружается при запуске системы и активно прослушивает свой порт, ожидая входящих соединений. Клиентские приложения пытаются установить соединение с сервером, после чего начинается обмен данными. По завершении сеанса связи клиент, как правило, разрывает соединение. На рисунке представлена базовая модель взаимодействия потоковых сокетов.

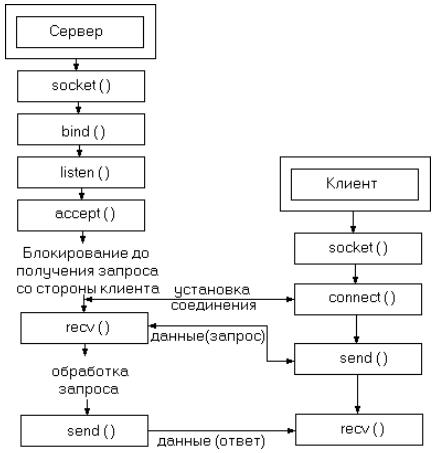


Рис. 1 Блок-схема взаимодействия потоковых сокетов

11. Создание сокета

Сокет идентифицирует пару, состоящую из IP-адреса и номера порта. Пара сокетов идентифицирует четырекомпонента: адреса и номера портов отправителя и получателя.

Обращение к сокетам осуществляется при помощи соответствующих дескрипторов сокетов. В Win32 сокет отличается от описателя файла и представлен отдельным типом – SOCKET.

Сокет создается одной из двух функций:

SOCKET WSASocket (int af,
int type,
int protocol,
LPWSAPROTOCOL_INFO IpProtocolInfo,GROUP g,
DWORD dwFlags);

или SOCKET socket (int af, int type, int protocol);

Первый параметр – *af*, определяет семейство адресов протокола и ограничивает применение второго и третьего параметров. Он может принимать значения *AF_UNIX, AF_INET, AF_OSI, AF_NS, AF_NETBIOS, AF_APPLETALK, AF_ATM* и т.д.. Значение

AF_INET позволяет взаимодействовать через объединенную сеть по IP-адресам.

Параметр *type* - это тип сокета для данного протокола. Для протокола TCP/IP он может принимать одно из следующих значений: *SOCK_STREAM* (транспортный протокол TCP, ориентированная на соединение надежная связь), *SOCK_DGRAM* (транспортный протокол UDP, ненадежная дейтаграммная связь), *SOCK_RAW* (простые сокеты).

Третий параметр — *protocol,* указывает конкретный протокол. Если этот параметр равен 0, это означает, что задействуется протокол по умолчанию для выбранных значений семейства адресов и типа. Для протокола TCP задается значение *IPPROTO_IP*, для протокола UDP – *IPPROTO_UDP*, для простых сокетов – *IPPROTO_RAW* или *IPPROTO_ICMP*.

Если в функции WSASocket указать константу *FROM_PROTOCOL_INFO* во всех трех параметрах (*af, type* и *protocol)* — для них будут использоваться значения из переданной структуры *WSAPROTOCOL_INFO*.

Теперь рассмотрим два последних флага из *WSASocket*. Параметр *GROUP* всегда равен 0, так как ни одна из версий Winsock не поддерживает группы сокетов. В параметре *dwFlags* указывают один или несколько следующих флагов:

WSA_FLAG_OVERLAPPED; WSA_FLAG_MULTIPOINT_C_ROOT;
WSA_FLAG_MULTIPOINT_C_LEAF; WSA_FLAG_MULTIPOINT_D_ROOT;
WSA_FLAG_MULTIPOINT_D_LEAF.

Первый флаг — *WSA_FLAG_OVERLAPPED,* указывает, что данный сокет допускает перекрытый ввод-вывод. Если сокет создается функцией *socket,* флаг *WSA_FLAG_OVERLAPPED* задан по умолчанию. В общем, рекомендуется всегда задавать этот флаг при использовании *WSASocket*. Последние четыре флага относятся к сокетам многоадресного вещания.

В случае успеха функция *socket()* возвращает дескриптор сокета, представляющий собой целое число, в случае неудачи возвращается –1.

12. Проверка и обработка ошибок

Большинство функций Winsock при успешном завершении возвращают 0, а в случае ошибки - значение SOCKET_ERROR, которое равно –1.

Для получения кода ошибки можно использовать функцию int WSAGetLastError (void);

Функция возвращает код последней ошибки. Коды ошибок описаны в Winsock.h или Winsock2.h (в зависимости от версии)

13. Серверные API-функции

Первый шаг после создания сокета — привязка сокета данного протокола к его стандартному адресу функцией *bind.* Второй — перевод сокета в режим прослушивания функцией *listen.* И наконец, сервер должен принять соединение клиента функцией *accept* или *WSAAccept.*

Фукция bind.

int bind (
SOCKET s,
const struct sockaddr FAR* name,
int namelen);

Параметр *s* задает дескриптор локального сокета, создаваемый функцией socket(), на котором ожидаются соединения клиентов.

Второй параметр – указатель на структуру, в которой хранится адрес сокета, соответствующий стандартам используемого протокола. Его нужно привести к типу *struct sockaddr*. В заголовочном файле Winsock определен тип *SOCKADDR,* соответствующий структуре *struct sockaddr.*Третий параметр задает размер переданной адресной структуры, зависящий отпротокола.

В случае успеха функция *bind()* возвращает 0. В случае ошибки - значение *SOCKET_ERROR* (т.е. -1). Самая распространенная ошибка при вызове *bind* — *WSAEADDRINUSE*. В случае использования TCP/IP это означает, что с локальным IP-интерфейсом и номером порта уже связан другой процесс, или они находятся в состоянии *TIME_WAIT*. При повторном вызове *bind* для уже связанного сокета возвращается ошибка *WSAEFAULT*.

Функция listen.

Для перевода сокета в состояние ожидания входящих соединений используется API-функция *listen().*

int listen(SOCKET s, int backlog); Первый параметр *s* — дескриптор сокета.

Параметр *backlog* – максимальное число входящих запросов на установление соединения, которые могут ждать в очереди на обработку пока сервер занят. Значение *backlog* зависит от поставщика протокола. Недопустимое значение заменяется ближайшим разрешенным.

Функции accept.

Прототип функции *accept:*

SOCKET accept(
SOCKET s,
struct sockaddr FAR* addr,int FAR* addrlen);

Параметр *s* — связанный сокет в состоянии прослушивания.

Второй параметр — адрес действительной структуры *SOCKADDR_IN.*

Параметр *addrlen* — ссылка на длину структуры *SOCKADDR_IN.*

Вызов *accept* обслуживает первый находящийся в очереди запрос на соединение. По его завершении структура *addr* будет содержать сведения об IP-адресе клиента, отправившего запрос, а параметр *addrlen* — размер структуры.

Кроме того, *accept* возвращает новый дескриптор сокета, соответствующий принятому клиентскому соединению. Для всех последующих операций с этим клиентом должен применяться новый сокет. Исходный прослушивающий сокет используется для приема других клиентских соединений и продолжает находиться в режиме прослушивания.

В Winsock 2 есть функция *WSAAccept,* способная устанавливать соединения в зависимости от результата вычисления условия.

14. Клиентские API-функции

Установление соединения осуществляется вызовом *connect* или *WSAConnect.*

Прототип функции connect:int connect (
SOCKET s,

const struct sockaddr FAR* name,int namelen);

Параметр *s* — действительный TCP-сокет для установления соединения, *name* — структура адреса сокета *(SOCKADDR_IN)* для TCP, описывающая сервер к которому подключаются, *namelen* — длина переменной *name.*

Если на компьютере, к которому вы подключаетесь, не запущен процесс, прослушивающий данный порт, функция *connect* вернет ошибку *WSAECONNREFUSED.* Другая ошибка — *WSAETIMEDOUT,* происходит, когда вызываемый адресат недоступен, например, из-за отказа коммуникационного оборудования на пути к узлу или отсутствия узла в сети.

15. Передача и прием данных по сокету (потоковые протоколы)

Для пересылки данных по сокету используются функции *send* и *WSASend.* Аналогично, для приема данных существуют функции *recv* и *WSARecv.*

Все буферы, используемые при отправке и приеме данных, состоят из элементов типа *char*.

Функция send.

API-функция *send* для отправки данных по сокету определена так:int send(
SOCKET s,
const char FAR * buf,int len,
int flags);

Параметр s определяет сокет для отправки данных.

Второй параметр — *buf,* указывает на символьный буфер, содержащий данные для отправки.

Третий — *len,* задает число отправляемых из буфера символов.

Параметр — *flags,* может принимать значения 0, *MSG_DONTROUTE, MSG_OOB,* или результат логического ИЛИ над любыми из этих параметров. При указании флага *MSG_DONTROUTE* не будут маршрутизироваться отправляемые пакеты. Флаг *MSG_OOB* указывает, что данные должны быть отправлены *вне полосы* (out of band), то есть срочно.

При успешном выполнении функция *send* вернет количество переданных байт, иначе — ошибку *SOCKET_ERROR.* Число, возвращаемое функцией *send* может быть меньше указанного размера буфера. Одна из типичных ошибок — *WSAE-CONNABORTED,* происходит при разрыве виртуального соединения из-за ошибки протокола или истечения времени ожидания. В этом случае сокет должен быть закрыт, так как он больше не может использоваться.

Число, возвращаемое функцией *send* может быть меньше указанного размера буфера.

В Winsock версии 2 определена функция *WSASend* — аналог *send.*

Функция recv.

Функция *recv* определена так:
int recv(
SOCKET s,
char FAR * buf,int len,
int flags);

Параметр s определяет сокет для приема данных.

Второй параметр — *buf,* является символьным буфером и предназначен для полученных данных.

Параметр *len* указывает число принимаемых байт или размер буфера *buf.*

Последний параметр — *flags,* может принимать значения 0, *MSG_PEEK, MSG_OOB* или результат логического ИЛИ над любыми из этих параметров. Разумеется, 0 означает отсутствие особых действий. Флаг *MSG_PEEK* указывает, что доступные данные должны копироваться в принимающий буфер и при этом оставаться в системном буфере. Его использовать не рекомендуется. Флаг *MSG_OOB* уже обсуждался при рассмотрении отправки данных.

Использование *recv* в сокетах, ориентированных на передачу сообщений или дейтаграмм, имеет несколько особенностей. Если при вызове *recv* размер ожидающих обработки данных больше предоставляемого буфера, то после его полного заполнения возникает ошибка *WSAEMSGSIZE*. Заметьте: ошибка превышения размера сообщения происходит только при использовании протоколов, ориентированных на передачу сообщений. Потоковые протоколы буферизируют поступающие данные и при запросе приложением предоставляют их в полном объеме, даже если количество ожидающих

обработки данных больше размера буфера. Таким образом, ошибка *WSAEMSGSIZE* неможет произойти при работе с потоковыми протоколами.

Функция *WSARecv* обладает дополнительными по сравнению с *recv* возможностями: поддерживает перекрытый ввод-вывод и фрагментарные дейтаграммные уведомления.

16. Завершение сеанса

По окончании работы с сокетом необходимо закрыть соединение и освободить все ресурсы, связанные с описателем сокета, вызвав функцию *closesocket.* Ее неправильное использование может привести к потере данных. Поэтому перед вызовом *closesocket* сеанс нужно корректно завершить функцией *shutdown.*

Функция shutdown.

Корректное завершение сеанса, при котором приложение уведомляет получателя об окончании отправки данных, осуществляется с помощью функции *shutdown.*

```
int shutdown(
SOCKET s,
int how );
```

Параметр *how* может принимать значения *SD_IRECEIVE, SD_SEND* или *SD_ BOTH.* Значение *SD_IRECEIVE* запрещает все последующие вызовы любых функций приема данных, на протоколы нижнего уровня это не действует. Если в очереди TCP-сокета есть данные, либо они поступают позже, соединение сбрасывается. UDP-сокеты в аналогиной ситуации продолжают принимать данные и ставить их в очередь.

SD_SEND запрещает все последующие вызовы функций отправки данных. В случае TCP-сокетов после подтверждения получателем приема всех отправленных данных передается пакет FIN.

SD_BOTH запрещает как прием, так и отправку.

Функция closesocket.

Эта функция закрывает сокет. Она определена так: int closesocket (SOCKET s);

Вызов *closesocket* освобождает дескриптор сокета, и все дальнейшие операции с сокетом закончатся ошибкой *WSAENOTSOCK*. Если не существует других ссылок на сокет, все связанные с дескриптором ресурсы будут освобождены, включая данные в очереди.

17. Протоколы без предварительного установления соединения

При использовании дейтаграммных сокетов сначала создают сокет функцией *socket* или *WSASocket.* Затем выполняют привязку сокета к интерфейсу, на котором будут принимать данные, функцией *bind* (как и в случае протоколов, ориентированных на сеансы). Разница в том, что вместо вызова *listen* или *accept* нужно просто ожидать приема входящих данных. Поскольку в этом случае соединения нет, принимающий сокет может получать дейтаграммы от любой машины в сети. Простейшая функция приема — *recvform:*

```
int recvfrom(
SOCKET s,
char FAR* buf,int len,
int flags,
struct sockaddr FAR* from,int FAR* fromlen );
```

Первые четыре параметра такие же, как и для функции *recv.* Параметр *from* — структура *SOCKADDR* для данного протокола слушающего сокета, на размер структуры

адреса ссылается *fromlen.* После возврата вызова структура *SOCKADDR* будет содержать адрес рабочей станции, которая отправляет данные.

В Winsock 2 применяется другая версия *recvform — WSARecvForm.*

Другой способ приема (отправки) данных в сокетах, не требующих соединения, — установление соединения (хоть это и звучит странно). После создания сокета можно вызвать *connect или WSAConnect,* присвоив параметру *SOCKADDR* адрес удаленного компьютера, с которым необходимо связаться. Фактически никакого соединения не происходит. Адрес сокета, переданный в функцию соединения, ассоциируется с сокетом, чтобы было можно использовать функции *recv* и *WSARecv* вместо *recvfrom* или *WSARecvFrom* (поскольку источник данных известен).

Есть два способа отправки данных через сокет, не требующий соединения. Первый исамый простой — создать сокет и вызвать функцию *sendto* или *WSASendTo.* Рассмотрим функцию *sendto:*

```
int sendto(
SOCKET s,
const char FAR * buf,int len,
int flags,
const struct sockaddr FAR * to,int tolen );
```

Параметры этой функции такие же, как и у *recvfrom,* за исключением *buf*— буфера данных для отправки, и *len* — показывающего сколько байт отправлять. Параметр *to* — указатель на структуру *SOCKADDR* с адресом принимающей рабочей станции.

Также можно использовать функцию *WSASendTo* из Winsock 2:

Как и при получении данных, сокет, не требующий соединения, можно подключать к адресу конечной точки и отправлять данные функциями *send* и *WSASend.* После создания этой привязки использовать для обмена данными функции *sendto* или *WSASendTo с* другим адресом нельзя — будет выдана ошибка. Отменить привязку сокета можно, лишь вызвав функцию *closesocket* с описателем этого сокета, после чего следует создать новый сокет.

Поскольку соединение не устанавливается, его формального разрыва или корректного закрытия не требуется. После прекращения отправки или получения данных отправителем или получателем просто вызывается функция *closesocket* с описателем требуемого сокета, в результате чего освобождаются все выделенные ему ресурсы.

Порядок выполнения работы

Задание 1.

Разработать TCP-сервер, создающий сокет, привязывающий его к локальному IP-адресу и порту и прослушивающий соединения клиентов. Номер порта и IP-адрес вводить с клавиатуры. IP-адрес задавать в десятично-точечной нотации.

Учесть, что функция accept возвращает новый дескриптор сокета, соответствующий принятому клиентскому соединению. Для всех последующих операций с данным клиентским соединением должен применяться новый сокет. Исходный прослушивающий сокет используется для приема других клиентских соединений и продолжает находиться в режиме прослушивания. При получении от клиента запроса на установление соединения вывести на экран IP-адрес и номер порта клиентского сокета.

Разработать приложение–клиент, соединяющееся с заданным TCP-сервером. Все отправленные и полученные по сокету данные вывести на экран.

При вызове API-функций выполнять проверку и обработку ошибок. Нарисовать блок-схему серверной и клиентской части программы. **Задание 2.**

Разработать серверное приложение, выполняющее получение данных через сокет безустановления соединения по протоколу UDP.

Разработать клиентское приложение, выполняющее отправку UDP-дейтаграмм. IP-адрес и порт удаленного получателя должен задаваться пользователем.

При вызове API-функций выполнять проверку и обработку ошибок.Адреса сокетов вывести на экран.

Нарисовать блок-схему серверной и клиентской части программы.

Варианты заданий:

1) Удаленный калькулятор (+,-,*,/);
2) Работа с массивом чисел (количество элементов в массиве, получение значения элемента массива по номеру, найти номер элемента в массиве по значению, увеличить значения элементов на заданное число);
3) Работа с массивом символов (преобразовать символы в верхний регистр, в нижний регистр, проверить, является ли символ цифрой, буквой, получить значение элемента массива);
4) Удаленный однострочный редактор (вставка символа в позицию, в конец строки, удаление символа из позиции);
5) Редактор для работы с двумя строками (конкатенация строк, проверка на равенство строк, проверка на равенство длин, вставка второй строки в заданную позицию первой);
6) Удаленный генератор псевдослучайных последовательностей (одно целое число вдиапазоне, массив чисел в диапазоне).

При разработке программ использовать функции Windows API или класс Socket пространства имен System.Net .NET Framework. В качестве языков программирования - C++, либо C#.

Контрольные вопросы

1. Что такое сокет? Какая версия Winsock используется на платформе Win32?
2. Сколько сокетов необходимо для взаимодействия клиента и сервера? Что представляет собой адрес сокета?
3. Назовите типы сокетов. В каком случае предпочтительнее использовать тот илиной тип сокетов?
4. Какой адрес можно использовать, чтобы прослушивать все сетевые интерфейсылокального узла?
5. Какой порядок байтов используется в Intel-совместимых процессорах? Какой порядок байтов применяется для работы с сокетами?
6. Какие номера портов могут задействовать прикладные программы при работе ссокетами?
7. Какая функция позволяет узнать номера портов, используемых стандартными службами? В каком файле хранится эта информация?
8. Чем отличается процесс получения и отправки данных на сокете, не требующемсоединения?
9. Как осуществляется корректное завершение сеанса работы с сокетом? В чем отличие завершения работы с потоковым и дейтаграммным сокетом?

Практическое занятие №4. Работы с буфером экрана.

Для выполнения практической работы потребуется отладочная плата 1986EvBrd_48_Rev3.В этой практической работе дан пример реализации драйвера дисплея и вывода текста на русском языке на экран, а так же организация вывода счётчика.

Проект состоит из восьми файлов. Один из них написан на языке ассемблер, остальные наязыке си.

Описание файлов

В файле font_ansi_5x8.c представлен шрифт в кодировке ANSI. Каждый из символов представлен в виде 5 байт (5 на 8 бит). Благодаря последовательности расположения описания символов (в том числе букв) в порядке следования определённом в кодировке ANSI имеется возможность в программе писать тексты на русском языке в удобочитаемомвиде.

В файле graphics.c размещены функции для вывода текста в буфер, очистки буфера ивыгрузки буфера на дисплей.

Файл startup_1986be9x.s

```
1           AREA    RESET, DATA, READONLY
2           IMPORT              main
3           DCD     0x20008000      ; Вершина стека
4           DCD     main            ; Выполнение программы main
5           END
```

Файл structs.h

```
1  #ifndef STRUCTS
2  #define STRUCTS
3
4  /* Определение структуры порта */
5  typedef struct {
6     unsigned RXTX;
7     unsigned OE;
8     unsigned FUNC;
9     unsigned DIGITAL;
10    unsigned PULL;
11    unsigned PD;
12    unsigned PWR;
13    unsigned GFEN;
14}struct_ports;
15

16#define A ((struct_ports *) 0x400A8000)
17#define D ((struct_ports *) 0x400C0000)
18#define F ((struct_ports *) 0x400E8000)
```

Файл driver_lcd.h

```
1  #ifndef DRIVER_LCD
2  #define DRIVER_LCD
3
4  #include "structs.h"
5
6  void LcdInit(void);
7  void LcdDispOn(void);
8 void LcdScreen(unsigned char screen[8][128]); 9

10#endif //DRIVER_LCD
```

Файл font_ansi_5x8.h

```
#ifndef FONT_ANSI_5X8
#define FONT_ANSI_5X8

extern const unsigned char ansi[256][5];
```

Файл graphics.h

```
#include "structs.h"
#include "driver_lcd.h"
#include "font_ansi_5x8.h"

void Print(unsigned row, unsigned col, char * byte);
void ClearScreen(void);
```

Файл main.c

```
1  /*.....*/
2  /* Практическая работа 02 (буфер дисплея) */
3  /*.....*/
4 #include "structs.h"
5  #include "driver_lcd.h"
6  #include "graphics.h"
8
9  void PortOut(struct_ports * port){
10   port->OE      = 0xFF;       /* Выход              *
                                                       /
11   port->FUNC      = 0x0000;    /* Порт                *
                                                        /
12   port->DIGITAL = 0xFF;        /* Цифровые            *
                                                        /
13   port->PULL     = 0x00000000; /* Подтяжка отключена   */
14   port->PD      = 0x00000000; /* Управляемый драйвер  */
15   port->PWR     = 0xFFFF;     /* Фронт ~10 нс        *
     port->GFEN    = 0x0000;                           /
16                               /* Фильтрация отключена
                                 */
17 }
18
19 static void Delay(unsigned count){
20  while(count--);
21 }
22
23 int main (void) {
24   /* Включение тактирования порта F
     */
25
     *(unsigned *)(0x4002001C) |= (1<<29);
26   PortOut(F);
27
28   /* Включение тактирования порта D
     */
29   *(unsigned *)(0x4002001C) |= (1<<24);
```

```
30  PortOut(D);
31  /* Включение тактирования порта A
    */
32  *(unsigned *)(0x4002001C) |= (1<<21);
33  PortOut(A);
34
35  Print(0,5,"Привет, программист!");
36  Print(2,0,"Это пример вывода    ");
    Print(3,0,"текста на LCD-дисплей");
37  ScreenUpdate();
38
39  Print(7,0,"Счет:");
40  char str[2] = "00";
41  for(int i = 35; i >= 0; --i){
42      str[0] = '0' + i/10;
43      str[1] = '0' + i%10;
44      Print(7,36,str);
        ScreenUpdate();
45      Delay(2000000);
46  }
```

Файл driver_lcd.c

```
1   #include "driver_lcd.h"
2
3   /* Обозначения в соответствии с описанием дисплея */
4   /* MT-12864Jv.1 МЭЛТ */
5   #define LCD_SET_COL 0x40
6   #define LCD_SET_ROW 0xB8
7
8   #define Pause /*Delay(5)*/
9   #define Pause2ms Delay(100)
10
11  #define LCD_MT_DB A->RXTX
12  #define LCD_MT_RW_off F->RXTX &= ~(1<<3)
13  #define LCD_MT_RW_on F->RXTX |= (1<<3)
14  #define LCD_MT_E
15  #define LCD_MT_RES
16  #define LCD_MT_A0_on F->RXTX |= (1<<4)
17  #define LCD_MT_A0_off F->RXTX &= ~(1<<4)
```

```
18 #define LCD_MT_E1 F->RXTX |= (F->RXTX & ~0x3) | 0x1
19 #define LCD_MT_E2 F->RXTX |= (F->RXTX & ~0x3) | 0x2
20 #define LCD_MT_E3 F->RXTX |= 0x3
21
22 /* Реализация функции задержки времени */
23 static void Delay(unsigned count){
24 while(count--);
25 }
26
27 /* Строб для записи байта в дисплей */
28 static void Strob(void){
29    Pause; /* Необходимая пауза */
30    D->RXTX |=  (1<<3); /* Включаем  3 бит порта D */ // Set_Stb_Pin
31    Pause; /* Необходимая пауза */
32    D->RXTX &= ~(1<<3); /* Выключаем 3 бит порта D */ // Set_Stb_Pin
33    Pause; /* Необходимая пауза */
34 }
35
36 /* Инициализация дисплея */
37 void LcdInit(void){
38    F->RXTX &= ~(1<<2); /* Выключаем второй бит порта F */ // Clr_Res
39    Pause2ms; /* Необходимая пауза */
40    F->RXTX |= (1<<2); /* Включаем 2 бит порта F */ // Set_Res
41    LCD_MT_E3; /* Включаем 0 и 1 бит порта F */ // Set_E1_Pin
42    LCD_MT_A0_off;
43    LCD_MT_RW_on;
44    Strob();
45    LCD_MT_A0_off;
46    LCD_MT_RW_off;
47 }
48
49 /* Включение дисплея */
50 void LcdDispOn(void){
51     LCD_MT_A0_off;
52    LCD_MT_RW_off;
53    Pause; /* Необходимая пауза */
54    A->RXTX |= 0x3F; /* Младшие 6 бит в 1 */
55    Strob();
56 }
57
58 /* Выбор чипа */
59 static void SetChip(unsigned num_chip){
60    F->RXTX &= ~(3<<0);
61    F->RXTX |=  (1<<num_chip);
62 }
63
64 /* Выбор ряда */
65 static void SetRow(unsigned char row){
66    LCD_MT_A0_off;
67    LCD_MT_RW_off;
68    A->RXTX = (LCD_SET_ROW | row);
69    Strob();
```

```
70 }
71
72 /* Выбор колонки */
73 static void SetCol(unsigned char col){
74     LCD_MT_A0_off;
75     LCD_MT_RW_off;
76     A->RXTX = (LCD_SET_COL | col);
77     Strob();
78 }
79
80 /* Вывод байта */
81 static void LcdWriteData(unsigned char data){
82     LCD_MT_A0_on;
83     LCD_MT_RW_off;
84     A->RXTX = data;
85     Strob();
86     LCD_MT_A0_off;
87     LCD_MT_RW_off;
88 }
89
90 /* Вывод изображения из буфера на экран */
91 void LcdScreen(unsigned char screen[8][128]){
92     SetChip(0);
93     for(unsigned row = 0; row < 8; ++row){
94         SetRow(row);
95         SetCol(0);
96         for(unsigned col = 0; col < 64; ++col){
97             unsigned char byte = screen[row][col];
98         LcdWriteData(byte);99
          }
100     }
101     SetChip(1);
102     for(unsigned row = 0; row < 8; ++row){
103         SetRow(row);
104         SetCol(0);
```

Файл font_ansi_5x8.c

```
1  #include "font_ansi_5x8.h"
2
3  const unsigned char ansi[256][5] = {
4      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x00
5      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x01
6      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x02
7      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x03
8      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x04
9      {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x05
```

```
10    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x06
11    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x07
12    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x08
13    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x09
14    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x0A
15    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x0B
16    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x0C
17    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x0D
18    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x0E
19    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x0F
20
21    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x10
22    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x11
23    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x12
24    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x13
25    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x14
26    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x15
27    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x16
28    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x17
29    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x18
30    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x19
31    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x1A
32    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x1B
33    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x1C
34    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x1D
35    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x1E
36    {0x00, 0x00, 0x00, 0x00, 0x00}, // spec 0x1F
37
38    {0x00, 0x00, 0x00, 0x00, 0x00}, // sp 0x20
39    {0x00, 0x00, 0x5f, 0x00, 0x00}, // ! 0x21
40    {0x00, 0x07, 0x00, 0x07, 0x00}, // " 0x22
41    {0x14, 0x7f, 0x14, 0x7f, 0x14}, // # 0x23
42    {0x24, 0x2a, 0x7f, 0x2a, 0x12}, // $ 0x24
43    {0x23, 0x13, 0x08, 0x64, 0x62}, // % 0x25
44    {0x36, 0x49, 0x55, 0x22, 0x50}, // & 0x26
45    {0x00, 0x05, 0x03, 0x00, 0x00}, // ' 0x27
46    {0x00, 0x1c, 0x22, 0x41, 0x00}, // ( 0x28
47    {0x00, 0x41, 0x22, 0x1c, 0x00}, // ) 0x29
48    {0x14, 0x08, 0x3E, 0x08, 0x14}, // * 0x2A
49    {0x08, 0x08, 0x3E, 0x08, 0x08}, // + 0x2B
50    {0x00, 0x00, 0x50, 0x30, 0x00}, // , 0x2C
51    {0x10, 0x10, 0x10, 0x10, 0x10}, // - 0x2D
52    {0x00, 0x60, 0x60, 0x00, 0x00}, // . 0x2E
53    {0x20, 0x10, 0x08, 0x04, 0x02}, // / 0x2F
54    {0x3E, 0x51, 0x49, 0x45, 0x3E}, // 0 0x30
55    {0x00, 0x42, 0x7F, 0x40, 0x00}, // 1 0x31
56    {0x42, 0x61, 0x51, 0x49, 0x46}, // 2 0x32
57    {0x21, 0x41, 0x45, 0x4B, 0x31}, // 3 0x33
58    {0x18, 0x14, 0x12, 0x7F, 0x10}, // 4 0x34
59    {0x27, 0x45, 0x45, 0x45, 0x39}, // 5 0x35
60    {0x3C, 0x4A, 0x49, 0x49, 0x30}, // 6 0x36
61    {0x01, 0x71, 0x09, 0x05, 0x03}, // 7 0x37
```

```
62    {0x36, 0x49, 0x49, 0x49, 0x36}, // 8 0x38
63    {0x06, 0x49, 0x49, 0x29, 0x1E}, // 9 0x39
64    {0x00, 0x36, 0x36, 0x00, 0x00}, // : 0x3A
65    {0x00, 0x56, 0x36, 0x00, 0x00}, // ; 0x3B
66    {0x08, 0x14, 0x22, 0x41, 0x00}, // < 0x3C
67    {0x14, 0x14, 0x14, 0x14, 0x14}, // = 0x3D
68    {0x00, 0x41, 0x22, 0x14, 0x08}, // > 0x3E
69    {0x02, 0x01, 0x51, 0x09, 0x06}, // ? 0x3F
70    {0x32, 0x49, 0x59, 0x51, 0x3E}, // @ 0x40
71    {0x7E, 0x11, 0x11, 0x11, 0x7E}, // A 0x41
72    {0x7F, 0x49, 0x49, 0x49, 0x36}, // B 0x42
73    {0x3E, 0x41, 0x41, 0x41, 0x22}, // C 0x43
74    {0x7F, 0x41, 0x41, 0x22, 0x1C}, // D 0x44
75    {0x7F, 0x49, 0x49, 0x49, 0x41}, // E 0x45
76    {0x7F, 0x09, 0x09, 0x09, 0x01}, // F 0x46
77    {0x3E, 0x41, 0x49, 0x49, 0x7A}, // G 0x47
78    {0x7F, 0x08, 0x08, 0x08, 0x7F}, // H 0x48
79    {0x00, 0x41, 0x7F, 0x41, 0x00}, // I 0x49
80    {0x20, 0x40, 0x41, 0x3F, 0x01}, // J 0x4A
81    {0x7F, 0x08, 0x14, 0x22, 0x41}, // K 0x4B
82    {0x7F, 0x40, 0x40, 0x40, 0x40}, // L 0x4C
83    {0x7F, 0x02, 0x0C, 0x02, 0x7F}, // M 0x4D
84    {0x7F, 0x04, 0x08, 0x10, 0x7F}, // N 0x4E
85    {0x3E, 0x41, 0x41, 0x41, 0x3E}, // O 0x4F
86    {0x7F, 0x09, 0x09, 0x09, 0x06}, // P 0x50
87    {0x3E, 0x41, 0x51, 0x21, 0x5E}, // Q 0x51
88    {0x7F, 0x09, 0x19, 0x29, 0x46}, // R 0x52
89    {0x46, 0x49, 0x49, 0x49, 0x31}, // S 0x53
90    {0x01, 0x01, 0x7F, 0x01, 0x01}, // T 0x54
91    {0x3F, 0x40, 0x40, 0x40, 0x3F}, // U 0x55
92    {0x1F, 0x20, 0x40, 0x20, 0x1F}, // V 0x56
93    {0x3F, 0x40, 0x38, 0x40, 0x3F}, // W 0x57
94    {0x63, 0x14, 0x08, 0x14, 0x63}, // X 0x58
95    {0x07, 0x08, 0x70, 0x08, 0x07}, // Y 0x59
96    {0x61, 0x51, 0x49, 0x45, 0x43}, // Z 0x5A
97    {0x00, 0x7F, 0x41, 0x41, 0x00}, // [ 0x5B
98    {0x55, 0x2A, 0x55, 0x2A, 0x55}, // 0x5C
99    {0x00, 0x41, 0x41, 0x7F, 0x00}, // ] 0x5D
100   {0x04, 0x02, 0x01, 0x02, 0x04}, // ^ 0x5E
101   {0x40, 0x40, 0x40, 0x40, 0x40}, // _ 0x5F
102   {0x00, 0x01, 0x02, 0x04, 0x00}, // ' 0x60
103   {0x20, 0x54, 0x54, 0x54, 0x78}, // a 0x61
104   {0x7F, 0x48, 0x44, 0x44, 0x38}, // b 0x62
105   {0x38, 0x44, 0x44, 0x44, 0x20}, // c 0x63
106   {0x38, 0x44, 0x44, 0x48, 0x7F}, // d 0x64
107   {0x38, 0x54, 0x54, 0x54, 0x18}, // e 0x65
108   {0x08, 0x7E, 0x09, 0x01, 0x02}, // f 0x66
109   {0x0C, 0x52, 0x52, 0x52, 0x3E}, // g 0x67
110   {0x7F, 0x08, 0x04, 0x04, 0x78}, // h 0x68
111   {0x00, 0x44, 0x7D, 0x40, 0x00}, // i 0x69
112   {0x20, 0x40, 0x44, 0x3D, 0x00}, // j 0x6A
113   {0x7F, 0x10, 0x28, 0x44, 0x00}, // k 0x6B
```

```
114   {0x00, 0x41, 0x7F, 0x40, 0x00}, // l 0x6C
115   {0x7C, 0x04, 0x18, 0x04, 0x78}, // m 0x6D
116   {0x7C, 0x08, 0x04, 0x04, 0x78}, // n 0x6E
117   {0x38, 0x44, 0x44, 0x44, 0x38}, // o 0x6F
118   {0x7C, 0x14, 0x14, 0x14, 0x08}, // p 0x70
119   {0x08, 0x14, 0x14, 0x18, 0x7C}, // q 0x71
120   {0x7C, 0x08, 0x04, 0x04, 0x08}, // r 0x72
121   {0x48, 0x54, 0x54, 0x54, 0x20}, // s 0x73
122   {0x04, 0x3F, 0x44, 0x40, 0x20}, // t 0x74
123   {0x3C, 0x40, 0x40, 0x20, 0x7C}, // u 0x75
124   {0x1C, 0x20, 0x40, 0x20, 0x1C}, // v 0x76
125   {0x3C, 0x40, 0x30, 0x40, 0x3C}, // w 0x77
126   {0x44, 0x28, 0x10, 0x28, 0x44}, // x 0x78
127   {0x0C, 0x50, 0x50, 0x50, 0x3C}, // y 0x79
128   {0x44, 0x64, 0x54, 0x4C, 0x44}, // z 0x7A
129   {0x08, 0x08, 0x36, 0x41, 0x41}, // { 0x7B
130   {0x00, 0x00, 0x7F, 0x00, 0x00}, // | 0x7C
131   {0x41, 0x41, 0x36, 0x08, 0x08}, // } 0x7D
132   {0x02, 0x01, 0x02, 0x02, 0x01}, // ~ 0x7E
133   {0x00, 0x00, 0x00, 0x00, 0x00}, // DE 0x7F
134
135   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x80
136   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x81
137   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x82
138   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x83
139   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x84
140   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x85
141   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x86
142   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x87
143   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x88
144   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x89
145   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x8A
146   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x8B
147   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x8C
148   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x8D
149   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x8E
150   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x8F
151
152   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x90
153   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x91
154   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x92
155   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x93
156   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x94
157   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x95
158   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x96
159   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x97
160   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x98
161   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x99
162   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x9A
163   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x9B
164   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x9C
165   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x9D
```

```
166   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x9E
167   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0x9F
168
169   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xA0
170   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xA1
171   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xA2
172   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xA3
173   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xA4
174   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xA5
175   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xA6
176   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xA7
177   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xA8
178   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xA9
179   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xAA
180   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xAB
181   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xAC
182   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xAD
183   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xAE
184   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xAF
185
186   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xB0
187   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xB1
188   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xB2
189   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xB3
190   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xB4
191   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xB5
192   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xB6
193   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xB7
194   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xB8
195   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xB9
196   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xBA
197   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xBB
198   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xBC
199   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xBD
200   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xBE
201   {0x00, 0x00, 0x00, 0x00, 0x00}, // 0xBF
202
203   {0x7E, 0x11, 0x11, 0x11, 0x7E}, // А 0xC0
204   {0x7F, 0x49, 0x49, 0x49, 0x33}, // Б 0xC1
205   {0x7F, 0x49, 0x49, 0x49, 0x36}, // В 0xC2
206   {0x7F, 0x01, 0x01, 0x01, 0x03}, // Г 0xC3
207   {0xE0, 0x51, 0x4F, 0x41, 0xFF}, // Д 0xC4
208   {0x7F, 0x49, 0x49, 0x49, 0x49}, // Е 0xC5
209   {0x77, 0x08, 0x7F, 0x08, 0x77}, // Ж 0xC6
210   {0x49, 0x49, 0x49, 0x49, 0x36}, // З 0xC7
211   {0x7F, 0x10, 0x08, 0x04, 0x7F}, // И 0xC8
212   {0x7C, 0x21, 0x12, 0x09, 0x7C}, // Й 0xC9
213   {0x7F, 0x08, 0x14, 0x22, 0x41}, // К 0xCA
214   {0x20, 0x41, 0x3F, 0x01, 0x7F}, // Л 0xCB
215   {0x7F, 0x02, 0x0C, 0x02, 0x7F}, // М 0xCC
216   {0x7F, 0x08, 0x08, 0x08, 0x7F}, // Н 0xCD
217   {0x3E, 0x41, 0x41, 0x41, 0x3E}, // О 0xCE
```

```
218   {0x7F, 0x01, 0x01, 0x01, 0x7F}, // П 0xCF
219   {0x7F, 0x09, 0x09, 0x09, 0x06}, // Р 0xD0
220   {0x3E, 0x41, 0x41, 0x41, 0x22}, // С 0xD1
221   {0x01, 0x01, 0x7F, 0x01, 0x01}, // Т 0xD2
222   {0x27, 0x48, 0x48, 0x48, 0x3F}, // У 0xD3
223   {0x1C, 0x22, 0x7F, 0x22, 0x1C}, // Ф 0xD4
224   {0x63, 0x14, 0x08, 0x14, 0x63}, // Х 0xD5
225   {0x7F, 0x40, 0x40, 0x40, 0xFF}, // Ц 0xD6
226   {0x07, 0x08, 0x08, 0x08, 0x7F}, // Ч 0xD7
227   {0x7F, 0x40, 0x7F, 0x40, 0x7F}, // Ш 0xD8
228   {0x7F, 0x40, 0x7F, 0x40, 0xFF}, // Щ 0xD9
229   {0x01, 0x7F, 0x48, 0x48, 0x30}, // Ъ 0xDA
230   {0x7F, 0x48, 0x30, 0x00, 0x7F}, // Ы 0xDB
231   {0x7F, 0x48, 0x48, 0x30, 0x00}, // Ь 0xDC
232   {0x22, 0x41, 0x49, 0x49, 0x3E}, // Э 0xDD
233   {0x7F, 0x08, 0x3E, 0x41, 0x3E}, // Ю 0xDE
234   {0x46, 0x29, 0x19, 0x09, 0x7F}, // Я 0xDF
235   {0x20, 0x54, 0x54, 0x54, 0x78}, // а 0xE0
236   {0x3C, 0x4A, 0x4A, 0x49, 0x31}, // б 0xE1
237   {0x7C, 0x54, 0x54, 0x28, 0x00}, // в 0xE2
238   {0x7C, 0x04, 0x04, 0x04, 0x0C}, // г 0xE3
239   {0xE0, 0x54, 0x4C, 0x44, 0xFC}, // д 0xE4
240   {0x38, 0x54, 0x54, 0x54, 0x08}, // е 0xE5
241   {0x6C, 0x10, 0x7C, 0x10, 0x6C}, // ж 0xE6
242   {0x44, 0x44, 0x54, 0x54, 0x28}, // з 0xE7
243   {0x7C, 0x20, 0x10, 0x08, 0x7C}, // и 0xE8
244   {0x78, 0x42, 0x24, 0x12, 0x78}, // й 0xE9
245   {0x7C, 0x10, 0x28, 0x44, 0x00}, // к 0xEA
246   {0x20, 0x44, 0x3C, 0x04, 0x7C}, // л 0xEB
247   {0x7C, 0x08, 0x10, 0x08, 0x7C}, // м 0xEC
248   {0x7C, 0x10, 0x10, 0x10, 0x7C}, // н 0xED
249   {0x38, 0x44, 0x44, 0x44, 0x38}, // о 0xEE
250   {0x7C, 0x04, 0x04, 0x04, 0x7C}, // п 0xEF
251   {0x7C, 0x14, 0x14, 0x14, 0x08}, // р 0xE0
252   {0x38, 0x44, 0x44, 0x44, 0x44}, // с 0xF1
253   {0x04, 0x04, 0x7C, 0x04, 0x04}, // т 0xF2
254   {0x0C, 0x50, 0x50, 0x50, 0x3C}, // у 0xF3
255   {0x18, 0x24, 0x7E, 0x24, 0x18}, // ф 0xF4
256   {0x44, 0x28, 0x10, 0x28, 0x44}, // х 0xF5
257   {0x7C, 0x40, 0x40, 0x40, 0xFC}, // ц 0xF6
258   {0x0C, 0x10, 0x10, 0x10, 0x7C}, // ч 0xF7
259   {0x7C, 0x40, 0x7C, 0x40, 0x7C}, // ш 0xF8
260   {0x7C, 0x40, 0x7C, 0x40, 0xFC}, // щ 0xF9
261   {0x04, 0x7C, 0x50, 0x50, 0x20}, // ъ 0xFA
262   {0x7C, 0x50, 0x20, 0x00, 0x7C}, // ы 0xFB
263   {0x7C, 0x50, 0x50, 0x20, 0x00}, // ь 0xFC
264   {0x28, 0x44, 0x54, 0x54, 0x38}, // э 0xFD
265   {0x7C, 0x10, 0x38, 0x44, 0x38}, // ю 0xFE
266   {0x08, 0x54, 0x34, 0x14, 0x7C}  // я 0xFF
267};
```

Файл graphics.c

```
1  #include "graphics.h"
2
3  /* Флаг, есть ли что выводить на экран */
4  static unsigned char
   grafic_out = 1;5
   6 /* Буфер для хранения графической
   информации */7  /* Доступ к памяти
   дисплея намного медленнее */ 8  static
   unsigned char screen[8][128];
9
   10/* Очистка буфера экрана */
   11void ClearScreen(void){
12
13
14
15}
16
grafic_out = 1;
for(unsigned i = 0; i < sizeof(screen);
  ++i)screen[0][i] = 0x00;
   17/* Печать нуль терминированной строки в позицию (row, col) */
   18void Print(unsigned row, unsigned col, char * byte){
19
20
21
22
```

```
grafic_out = 1;
if (row > 7)
return;
while(*byte){
  for(unsigned i = 0; i < 5 && col < 128;
    ++i)screen[row][col++] =
    ansi[*byte][i];
  if (col < 128)
    screen[row][col++] =
    0x00;
```

```
    ++byte;
 }
30/* Вывод изображения, если были изменения */
31void ScreenUpdate(void){
32
33
34
35
36}
 if (grafic_out){
    grafic_out = 0;
    LcdScreen(scr
    een);
 }
```

Задания для самостоятельного выполнения

1. Реализуйте горизонтальную бегущую строку.
2. Реализуйте вертикальную бегущую строку.
3. Реализуйте строку, бегущую по периметру дисплея.
4. Реализуйте счёт в двоичной позиционной системе счисления.
5. Реализуйте счёт в шестнадцатеричной позиционной системе счисления.
6. Добавьте в файл graphics.c функцию вывода переменной типа unsigned int, добавьтеобъявление в graphics.h.

Перечень учебных изданий, Интернет ресурсов, дополнительной литературы

Основная литература

1. Белугина, С. В. Разработка программных модулей программного обеспечения для компьютерных систем. Прикладное программирование : учебное пособие для спо / С. В. Белугина. - 3-е изд., стер. - Санкт-Петербург : Лань, 2022. - 312 с. - URL: https://e.lanbook.com/book/200390 (дата обращения: 28.03.2022). - Режим доступа: ЭБС Лань, для зарегистрир. пользователей. - ISBN 978-5-8114-9817-8. – Текст : электронный.
2. Федорова, Г. Н. Разработка, внедрение и адаптация программного обеспечения отраслевой направленности : учебное пособие / Г. Н. Федорова. - Москва : КУРС : ИНФРА-М, 2022 - 336 с. - (Среднее Профессиональное Образование). - URL: https://znanium.com/catalog/product/1858587 (дата обращения: 28.03.2022). - Режим доступа: ЭБС Znanium.com, для зарегистрир. пользователей. - ISBN 978-5-906818-41-6. - Текст : электронный.
3. Гагарина, Л. Г. Технология разработки программного обеспечения : учебное пособие / Л. Г. Гагарина, Е. В. Кокорева, Б. Д. Сидорова-Виснадул ; под ред. Л. Г. Гагариной. - Москва : ФОРУМ : ИНФРА-М, 2021. - 400 с. - (Среднее профессиональное образование). - URL: https://znanium.com/catalog/product/1189951 (дата обращения: 29.03.2022). - Режим доступа: ЭБС Znanium.com, для зарегистрир. пользователей. - ISBN 978-5-8199-0812-9. - Текст : электронный.
4. Зубкова, Т. М. Технология разработки программного обеспечения : учебное пособие / Т. М. Зубкова. - Санкт-Петербург : Лань, 2022. - 324 с. - URL: https://ezpro.fa.ru:3178/book/206882 (дата обращения: 09.03.2022). - Режим доступа: ЭБС Лань, для зарегистрир. пользователей. - ISBN 978-5-8114-3842-6. - Текст : электронный.
5. Волк, В. К. Практическое введение в программную инженерию : учебное пособие / В. К. Волк. - Санкт-Петербург : Лань, 2022. - 100 с. - URL: https://ezpro.fa.ru:3178/book/206669 (дата обращения: 09.03.2022). - Режим доступа: ЭБС Лань, для зарегистрир. пользователей. - ISBN 978-5-8114-3656-9. - Текст : электронный.
6. Белугина, С. В. Разработка программных модулей программного обеспечения для компьютерных систем. Прикладное программирование : учебное пособие для СПО / С. В. Белугина. - 3-е изд., стер. - Санкт-Петербург : Лань, 2022. - 312 с. - URL: https://e.lanbook.com/book/200390 (дата обращения: 28.03.2022). - Режим доступа: ЭБС Лань, для зарегистрир. пользователей. - ISBN 978-5-8114-9817-8. - Текст : электронный.
7. Коржинский С. Н. Разработка мобильных приложений : учебник / Коржинский С. Н. - Москва : КноРус, 2022. - 421 с. - URL: https://book.ru/book/944559 (дата обращения: 18.08.2022). - Режим доступа: ЭБС Book.ru, для зарегистрир. пользователей. - ISBN 978-5-406-09570-6. - Текст : электронный.
8. Соколова, В. В. Разработка мобильных приложений : учебное пособие для среднего профессионального образования / В. В. Соколова. - Москва : Юрайт, 2022. - 175 с. - (Профессиональное образование). - URL: https://ezpro.fa.ru:3217/bcode/495527 (дата обращения: 09.03.2022). - Режим доступа: ЭБС Юрайт, для зарегистрир. пользователей. - ISBN 978-5-534-10680-0. - Текст : электронный.
9. Заяц, А. М. Введение в гибридные технологии разработки мобильных приложений : учебное пособие для спо / А. М. Заяц, Н. П. Васильев. - 2-е изд., стер. - Санкт-Петербург : Лань, 2022. - 160 с. - URL: https://e.lanbook.com/book/200459 (дата обращения: 09.03.2022). - Режим доступа: ЭБС Лань, для зарегистрир. пользователей. - ISBN 978-5-8114-9555-9. - Текст : электронный.
10. Пирская, Л. В. Разработка мобильных приложений в среде Android Studio : учебное пособие / Л. В. Пирская ; Южный федеральный университет. - Ростов-на-Дону ; Таганрог : Южный федеральный университет, 2019. - 125 с. : ил., табл. - URL: https://biblioclub.ru/index.php?page=book&id=598634 (дата обращения: 09.03.2022). - Режим

доступа: ЭБС Университетская библиотека онлайн, для зарегистрир. пользователей. - Библиогр. в кн. - ISBN 978-5-9275-3346-6. - Текст : электронный.

11. Кузнецов, А. С. Системное программирование : учебное пособие / А. С. Кузнецов, И. А. Якимов, П. В. Пересунько. - Красноярск : СФУ, 2018. - 170 с. - URL: https://e.lanbook.com/book/157574 (дата обращения: 09.03.2022). - Режим доступа: ЭБС Лань, для зарегистрир. пользователей. - ISBN 978-5-7638-3885-5. - Текст : электронный.

12. Васильева, И. И. Системное и прикладное программирование : учебное пособие / И. И. Васильева. - Елец : ЕГУ им. И.А. Бунина, 2019. - 130 с. - URL: https://e.lanbook.com/book/195791 (дата обращения: 09.03.2022). - Режим доступа: ЭБС Лань, для зарегистрир. пользователей. - ISBN 978-5-00151-039-0. - Текст : электронный.

Электронные издания (электронные ресурсы)

1. Учебники по программированию http://programm.ws/index.php
2. Eckel B. Thinking in C++ (2nd Edition) Free Electronic Book.
3. http://www.mindview.net/Books/TICPP/ThinkingInCPP2e.html

Дополнительная литература

1. Подбельский, В. В. Программирование. Базовый курс C# : учебник для среднего профессионального образования / В. В. Подбельский. - Москва : Юрайт, 2022. - 369 с. - (Профессиональное образование). - URL: https://ezpro.fa.ru:3217/bcode/475775 (дата обращения: 28.03.2022). - Режим доступа: ЭБС Юрайт, для зарегистрир. пользователей. - ISBN 978-5-534-11467-6. - Текст : электронный.

2. Гниденко, И. Г. Технология разработки программного обеспечения : учебное пособие для среднего профессионального образования / И. Г. Гниденко, Ф. Ф. Павлов, Д. Ю. Федоров. - Москва : Юрайт, 2022. - 235 с. - (Профессиональное образование). - URL: https://ezpro.fa.ru:3217/bcode/492496 (дата обращения: 28.03.2022). - Режим доступа: ЭБС Юрайт, для зарегистрир. пользователей. - ISBN 978-5-534-05047-9. - Текст : электронный.

3. Исаченко, О. В. Программное обеспечение компьютерных сетей : учебное пособие / О. В. Исаченко. - 2-е изд., испр. и доп. - Москва : ИНФРА-М, 2021. - 158 с. - (Среднее профессиональное образование). - URL: https://znanium.com/catalog/product/1189344 (дата обращения: 29.03.2022). - Режим доступа: ЭБС Znanium.com, для зарегистрир. пользователей. - ISBN 978-5-16-015447-3. - Текст : электронный.

4. Черткова, Е. А. Программная инженерия. Визуальное моделирование программных систем : учебник для среднего профессионального образования / Е. А. Черткова. - 2-е изд., испр. и доп. - Москва : Юрайт, 2022. - 147 с. - (Профессиональное образование). - URL: https://ezpro.fa.ru:3217/bcode/493226 (дата обращения: 28.03.2022). - Режим доступа: ЭБС Юрайт, для зарегистрир. пользователей. - ISBN 978-5-534-09823-5. - Текст : электронный.

5. Немцова, Т. И. Программирование на языке высокого уровня. Программирование на языке C++ : учебное пособие / Т. И. Немцова, С. Ю. Голова, А. И. Терентьев ; под ред. Л. Г. Гагариной. - Москва : ФОРУМ : ИНФРА-М, 2021. - 512 с. + Доп. Материалы. - (Среднее профессиональное образование). - URL: https://znanium.com/catalog/product/1172261 (дата обращения: 29.03.2022). - Режим доступа: ЭБС Znanium.com, для зарегистрир. пользователей. - ISBN 978-5-8199-0699-6. - Текст : электронный.

6. Мартишин, С. А. Основы теории надежности информационных систем : учебное пособие / С. А. Мартишин, В. Л. Симонов, М. В. Храпченко. - Москва : ФОРУМ : ИНФРА-М, 2020. - 255 с. - (Высшее образование: Бакалавриат). - URL:

https://znanium.com/catalog/product/1062374 (дата обращения: 04.07.2022). - Режим доступа: ЭБС Znanium.com, для зарегистрир. пользователей. - ISBN 978-5-8199-0757-3. - Текст : электронный

7. Советов, Б. Я. Информационные технологии: теоретические основы : учебное пособие / Б. Я. Советов, В. В. Цехановский. - 2-е изд., стер. - Санкт-Петербург : Лань, 2021. - 444 с. - URL: https://ezpro.fa.ru:3178/book/167404 (дата обращения: 28.03.2022). - Режим доступа: ЭБС Лань, для зарегистрир. пользователей. - ISBN 978-5-8114-1912-8. – Текст : электронный.

8. Ехлаков, Ю. П. Управление программными проектами. Стандарты, модели : учебное пособие для вузов / Ю. П. Ехлаков. - 3-е изд., стер. - Санкт-Петербург : Лань, 2021. - 244 с. - URL: https://ezpro.fa.ru:3178/book/175498 (дата обращения: 09.03.2022). - Режим доступа: ЭБС Лань, для зарегистрир. пользователей. - ISBN 978-5-8114-8362-4. - Текст : электронный.

9. Беспалов, Д. А. Операционные системы реального времени и технологии разработки кроссплатформенного программного обеспечения : учебное пособие : в 3 частях / Д. А. Беспалов, С. М. Гушанский, Н. М. Коробейникова ; Южный федеральный университет. - Ростов-на-Дону ; Таганрог : Южный федеральный университет, 2021. - Часть 3. - 214 с. : ил. - URL: https://biblioclub.ru/index.php?page=book&id=683905 (дата обращения: 09.03.2022). - Режим доступа: ЭБС Университетская библиотека онлайн, для зарегистрир. пользователей. - Библиогр.: с. 187-188. - ISBN 978-5-9275-3628-3 (Ч. 3). - ISBN 978-5-9275-3366-4. - Текст : электронный.

10. Тузовский, А. Ф. Проектирование и разработка web-приложений : учебное пособие для среднего профессионального образования / А. Ф. Тузовский. - Москва : Юрайт, 2022. - 218 с. - (Профессиональное образование). - URL: https://ezpro.fa.ru:3217/bcode/495109 (дата обращения: 29.03.2022). - Режим доступа: ЭБС Юрайт, для зарегистрир. пользователей. - Библиогр. в кн. - ISBN 978-5-534-10017-4. - Текст : электронный.

11. Соколова, В. В. Вычислительная техника и информационные технологии. Разработка мобильных приложений : учебное пособие для вузов / В. В. Соколова. - Москва : Юрайт, 2022. - 175 с. - (Высшее образование). - URL: https://ezpro.fa.ru:3217/bcode/490305 (дата обращения: 09.03.2022). - Режим доступа: ЭБС Юрайт, для зарегистрир. пользователей. - ISBN 978-5-9916-6525-4. - Текст : электронный.

12. Жулабова, Ф. Т. Системное программирование. Лабораторные работы : учебное пособие для спо / Ф. Т. Жулабова. - 2-е изд., стер. - Санкт-Петербург : Лань, 2021. - 208 с. - URL: https://e.lanbook.com/book/164955 (дата обращения: 09.03.2022). - Режим доступа: ЭБС Лань, для зарегистрир. пользователей. - ISBN 978-5-8114-7721-0. - Текст : электронный.

13. Гунько, А. В. Системное программирование в среде Linux : учебное пособие / А. В. Гунько. - Новосибирск : НГТУ, 2020. - 235 с. - URL: https://e.lanbook.com/book/152228 (дата обращения: 09.03.2022). - Режим доступа: ЭБС Лань, для зарегистрир. пользователей. - ISBN 978-5-7782-4160-2. - Текст : электронный.