ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Р.Е. АЛЕКСЕЕВА

Е.А. СУХАНОВА

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ВЫСОКОГО УРОВНЯ

КОМПЛЕКС УЧЕБНО-МЕТОДИЧЕСКИХ МАТЕРИАЛОВ

Часть 2

Рекомендовано Ученым советом Нижегородского государственного технического университета им. Р.Е. Алексеева в качестве учебно-методического пособия для студентов всех форм обучения, включающих элементы дистанционных технологий по специальности 230201 «Информационные системы и технологии»

Нижний Новгород 2008

УДК 681.3.06

Суханова Е.А. Программирование на языке высокого уровня: комплекс учебно-методических материалов: ч.2 / Е.А. Суханова; НГТУ им. Р.Е. Алексеева. Н.Новгород, 2008. – 69 с.

Содержит описание лабораторных работ, указания к курсовому проектированию, глоссарий и список рекомендуемой учебной литературы

Предназначен для студентов всех форм обучения, включающих элементы дистанционных технологий.

Редактор Е.В. Комарова

Подписано в печать 09.07.2008. Формат 60 x 84 $^{1}/_{16}$. Бумага офсетная. Печать офсетная. Усл. печ. л. 4,4. Уч.-изд. л. 4,0. Тираж 200 экз. Заказ .

Нижегородский государственный технический университет им. Р.Е. Алексеева. Типография НГТУ. 603950, ГСП-41, г. Нижний Новгород, ул. Минина, 24.

СОДЕРЖАНИЕ

1. ОПИСАНИЕ ПРАКТИЧЕСКИХ ЗАНЯТИЙ 4

1.1. ОПИСАНИЕ СРЕДЫ РАЗРАБОТКИ 4

1.2. РАБОТА С УКАЗАТЕЛЯМИ 12

1.3. РАБОТА С ФУНКЦИЯМИ. РЕКУРСИЯ 15

1.4. РАБОТА СО СТРОКАМИ 17

1.5. РАБОТА С ФАЙЛАМИ 19

1.6. РАБОТА СО СТРУКТУРАМИ 21

1.7. РАБОТА С КЛАССАМИ 23

1.8. РЕКОМЕНДАЦИИ К КУРСОВОМУ ПРОЕКТУ 25

2. КОНТРОЛЬ ЗНАНИЙ 31

ГЛОССАРИЙ 33

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ 37

ПРИЛОЖЕНИЯ 39

1. ОПИСАНИЕ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

1.1. ОПИСАНИЕ СРЕДЫ РАЗРАБОТКИ

Лабораторные работы проходят в среде разработки Microsoft Visual Studio 2005.

Для выполнения лабораторных работ необходимо запустить Microsoft Visual Studio 2005.

В открывшемся окне выбрать меню File->New->Project (рис. 1.).

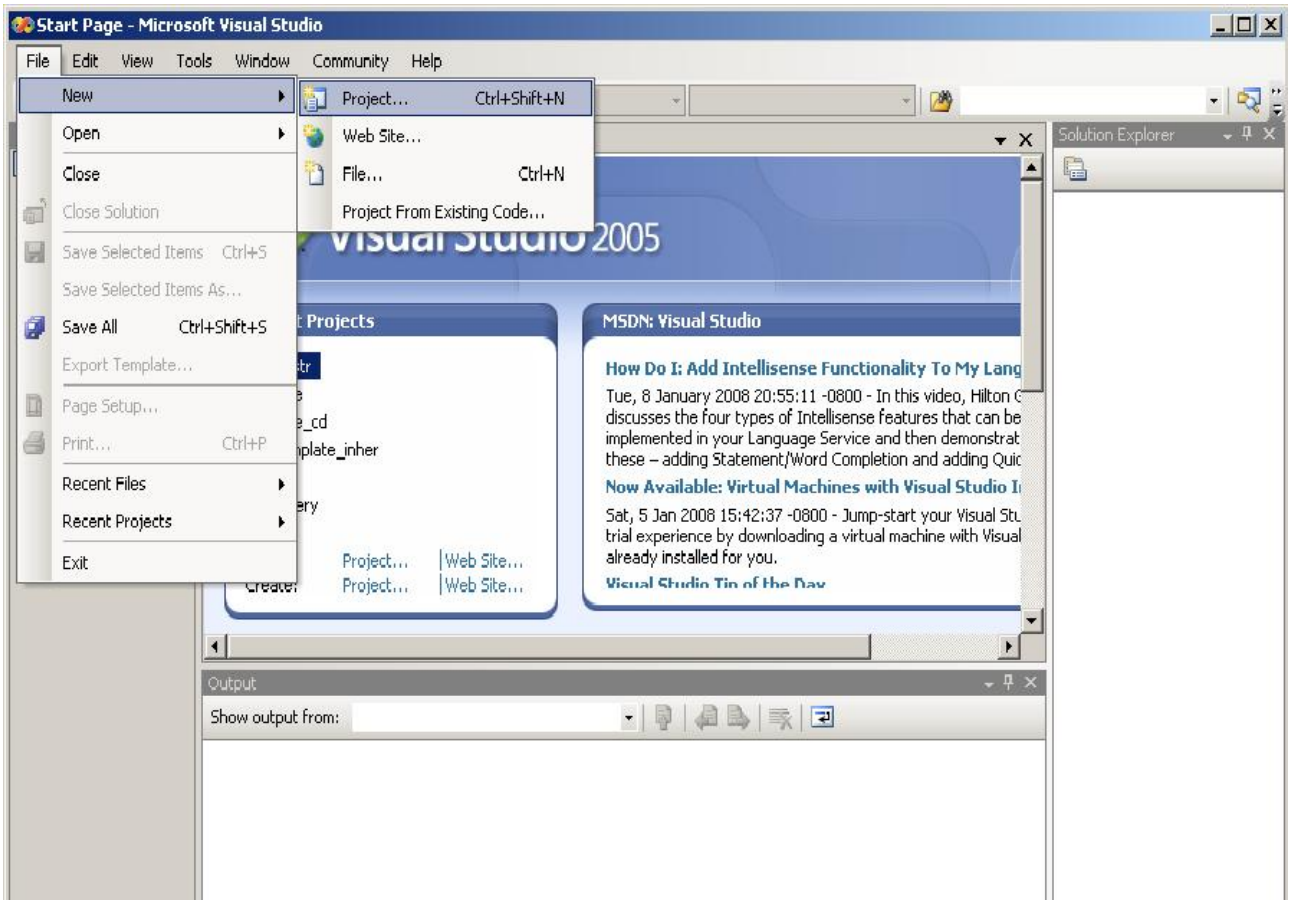

Рис. 1. Создание нового проекта

Далее откроется окно, в котором необходимо выбрать тип проекта, имя проекта, путь к папке проекта (см. рис. 2.).

В левом окне следует выбрать тип проекта Visual C++, в шаблонах в правом окне выбираем шаблон Win32 Console Application.

Затем указываем имя проекта. Имя нужно было выбирать осмысленное, чтобы затем было проще искать выполненную работу.

После задания имени нужно выбрать папку, где будет храниться проект. Проекты сохраняются в папке «Студент», где необходимо указать группу и фамилию студента, выполняющего работу.

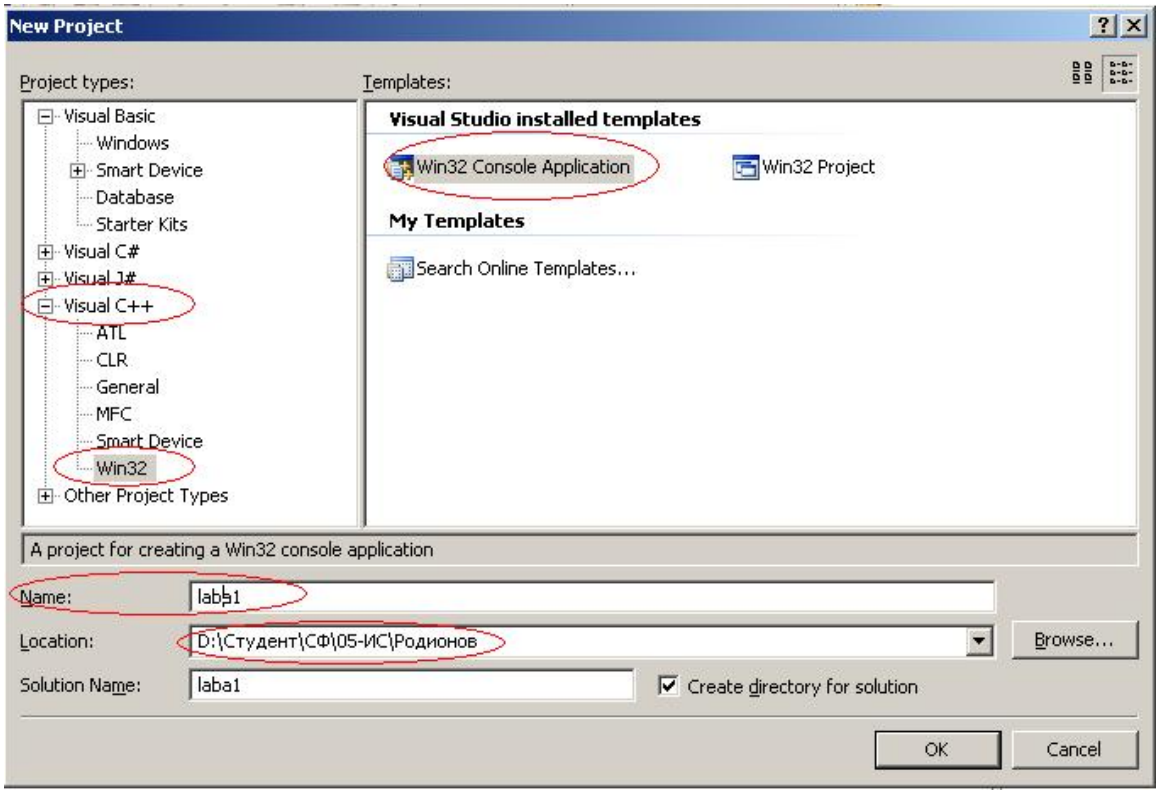

Рис. 2. Задание параметров нового проекта

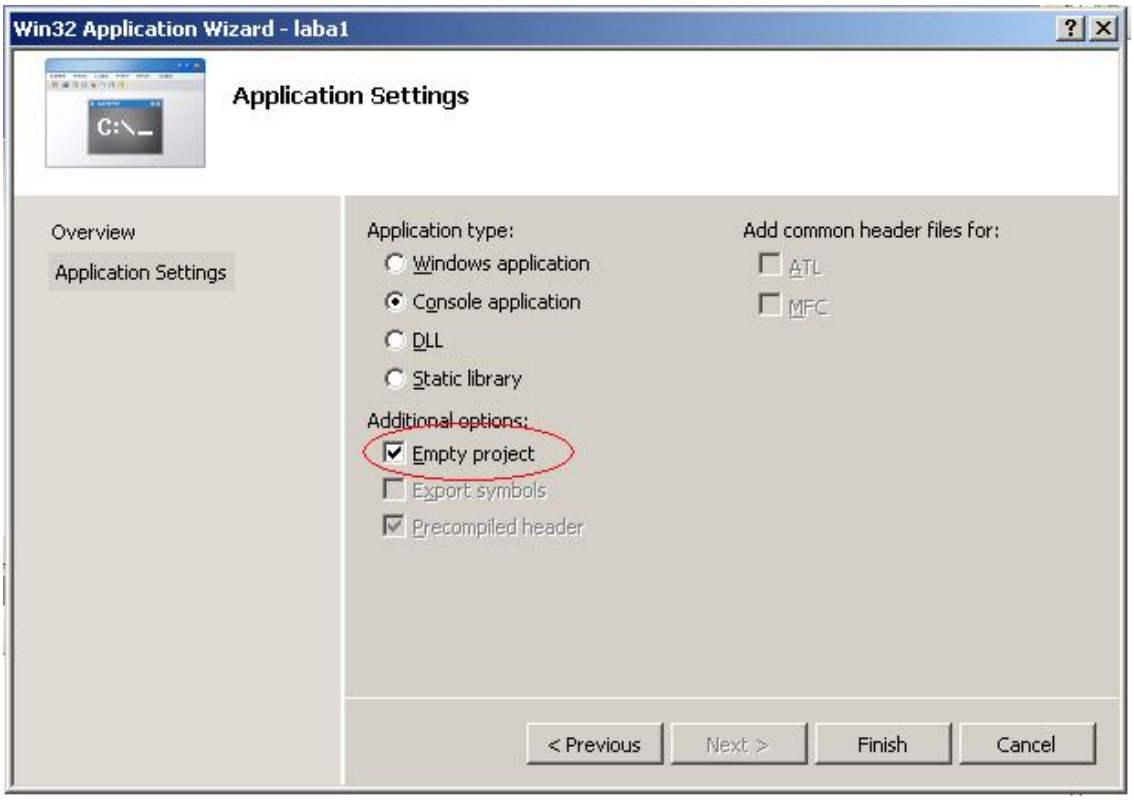

Рис. 3. Выбираем тип проекта Empty

После указания всех параметров проекта нужно нажать кнопку «ОК», в следующем окне необходимо выбрать кнопку «Next», и в открывшемся диалоговом окне выбрать дополнительную опцию «Пустой проект» (см. рис. 3.).

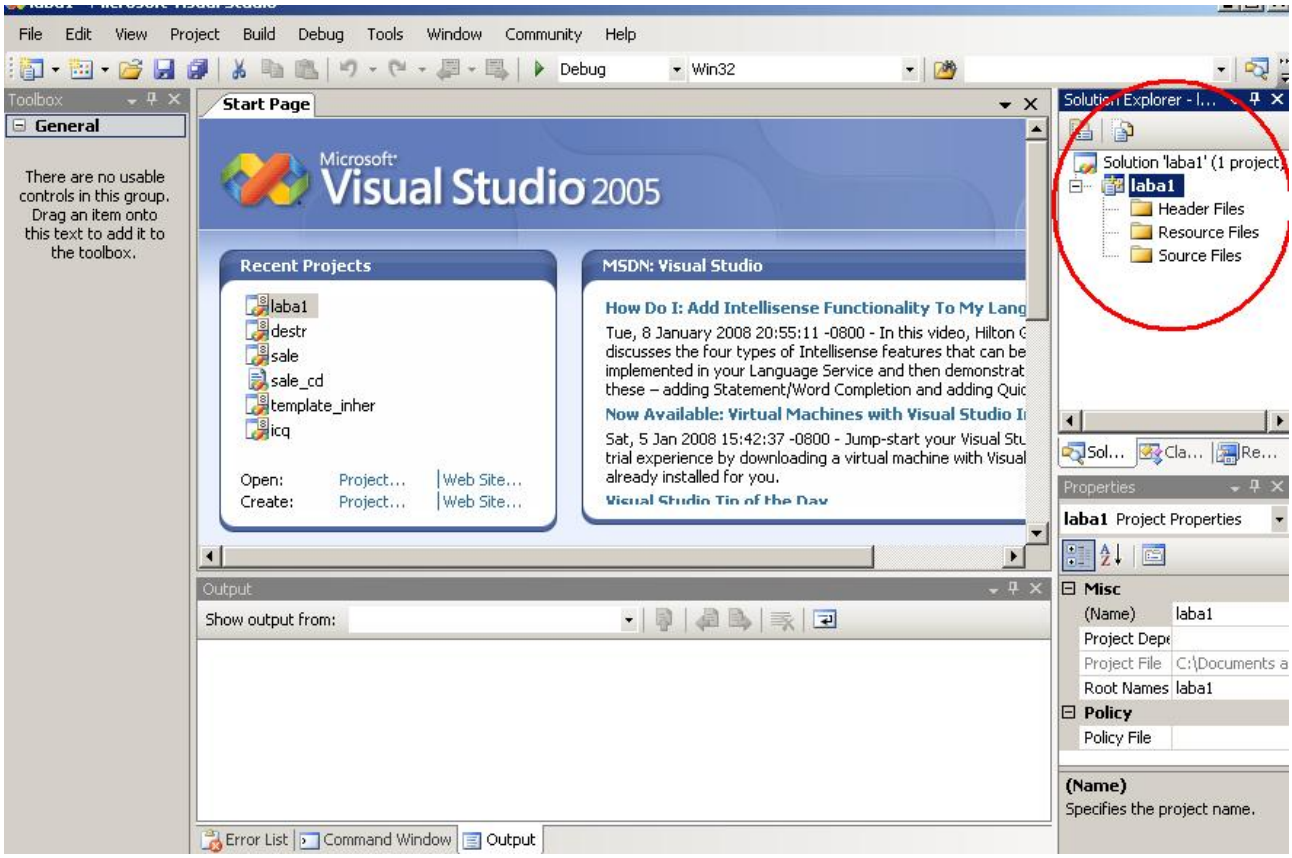

Рис. 4. Вид пустого проекта

После этого будет создан пустой проект, вид которого показан на рис. 4. Справа видно окно проекта Solution Explorer. Если по каким-то причинам оно не отображается, то нужно зайти в меню View->Solution Explorer.

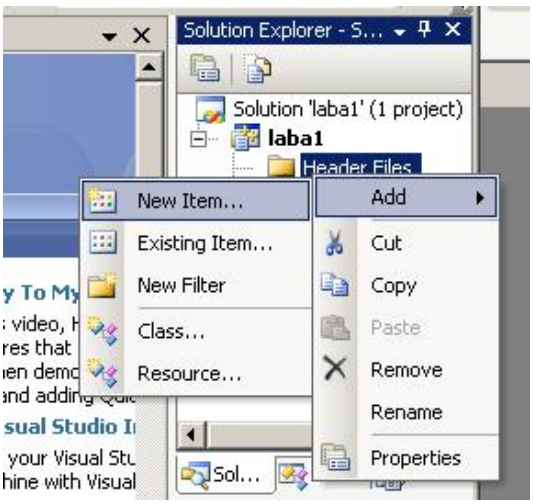

Рис. 5. Добавление новых файлов в проект

Теперь в пустой проект необходимо добавить файлы. Добавлять будем два вида файлов – заголовочные и файлы с кодом.

Для добавления заголовочных файлов нужно правой кнопкой мыши нажать на строчку Header Files в окне Solution Explorer, выбрать пункт Add->New Item (см. рис. 5.).

После данной операции откроется окно, показанное на рис. 6. В нем необходимо выбрать категорию Code, шаблон файла – Header Files(.h), указать имя файла. Путь будет прописан по умолчанию – файл будет добавлен в папку проекта.

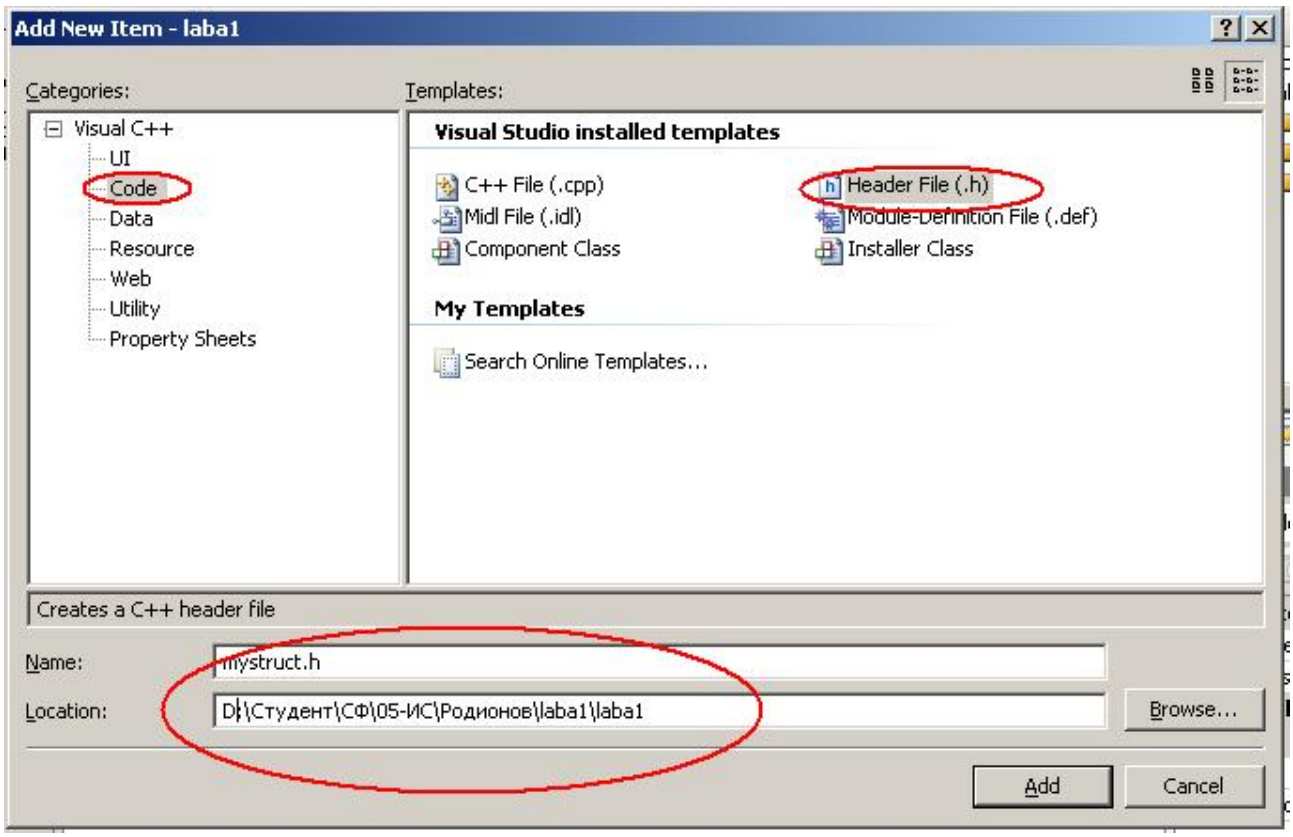

Рис. 6. Выбор типа и имени добавляемого файла

В тех лабораторных работах, где требуется создание структур или классов, необходимо добавлять три файла – один заголовочный и два файла с кодом. Имя одного из файлов с кодом должно совпадать с именем заголовочного файла (см. рис. 7.).

В тех лабораторных работах, где не требуется создание пользовательского типа данных, достаточно добавить в проект один файл с кодом, где и будет набираться код.

Если все сделано правильно, в окне Solution Explorer появятся файлы, которые были добавлены в проект, а рядом с вкладкой Start Page появятся вкладки с именами этих файлов.

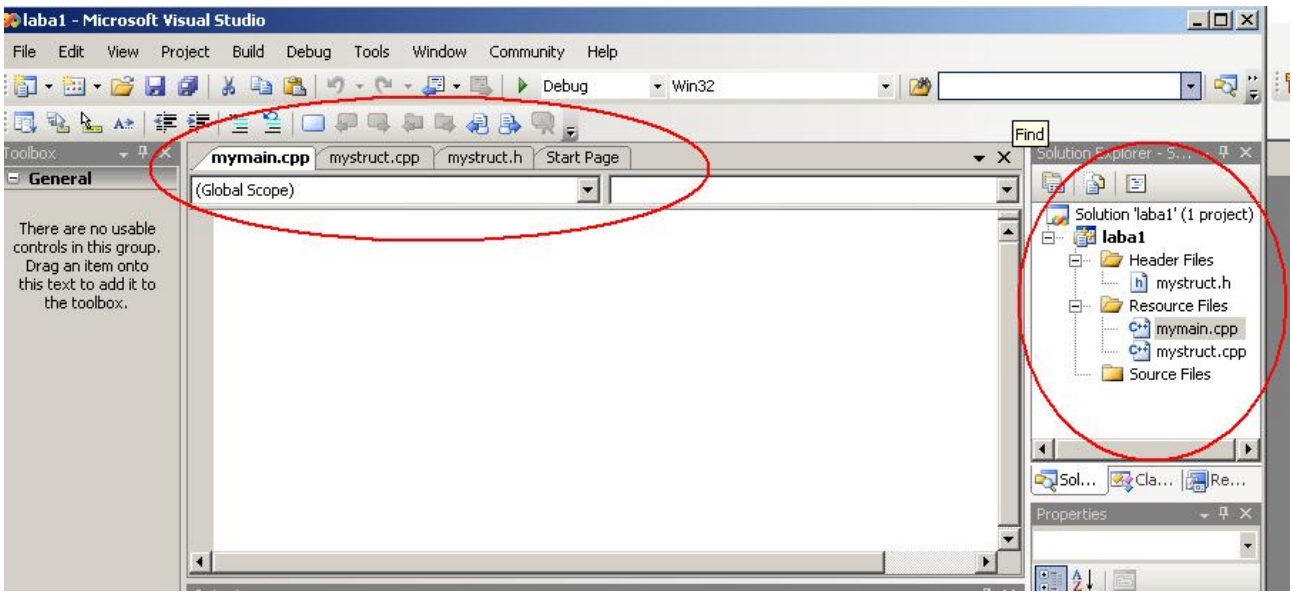

Рис. 7. Вид проекта после добавления файлов

Затем необходимо набрать программу.

mymain.cpp* | mystruct.cpp* | **mystruct.h*** | Start Page

(Global Scope)

```
#ifndef _MYSTRUCT_H_
#define _MYSTRUCT_H_

struct TNode {
    int value;
    TNode* pnext;
};

void add2list(TNode **pphead, int val);
void print(TNode *phead);
void deleteList(TNode *phead);

#endif
```

mymain.cpp* | **mystruct.cpp*** | mystruct.h* | Start Page

(Global Scope)

```
#include "mystruct.h"
#include <iostream>
using namespace std;

void add2list(TNode **pphead, int val)
{
    TNode **pp = pphead, *pnew;
    while(*pp)
    {
        if(val < (*pp)->value)
            break;
        else
            pp = &((*pp)->pnext);
    }
    pnew = new TNode;
    pnew->value = val;
    pnew->pnext = *pp;
    *pp = pnew;
}

void print(TNode *phead)
{
    TNode* p = phead;
    while(p)
    {
```

mymain.cpp* | mystruct.cpp* | mystruct.h* | Start Page

(Global Scope)

```
#include "mystruct.h"
#include <iostream>
using namespace std;

int main()
{
    TNode *phead = 0;
    srand(time(0));
    for(int i = 0; i < 10; ++i)
        add2list(&phead, rand() % 100);
    cout << "Elements of the list:" << endl;
    print(phead);
    deleteList(phead);
    return 0;
}
```

Рис. 8. Файлы с программой

Все файлы, в которые были внесены изменения, теперь помечены звездочкой (см. рис. 8.).

Теперь программу можно компилировать. Для этого необходимо выбрать пункт меню Build->Build Solution либо нажать комбинацию «горячих» клавиш <Ctrl>+<Shift>+<B> (рис. 9.).

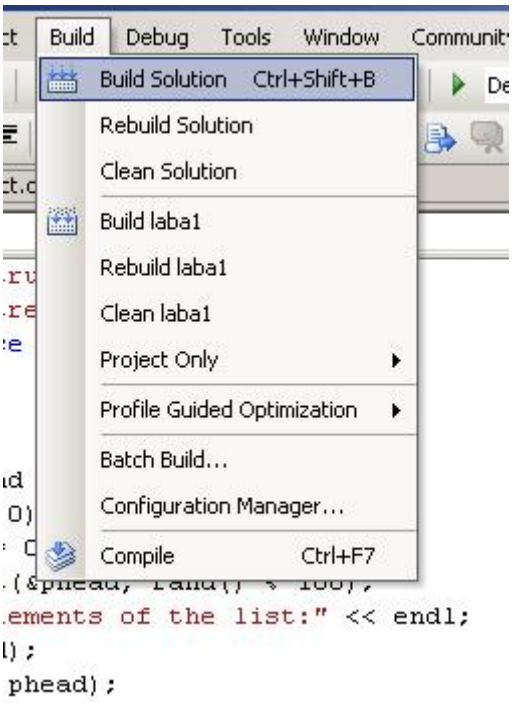

Рис. 9. Компиляция программы

Если код набран верно, то ошибок не будет, и внизу в окне вывода будет выведена информация об успешной компиляции.

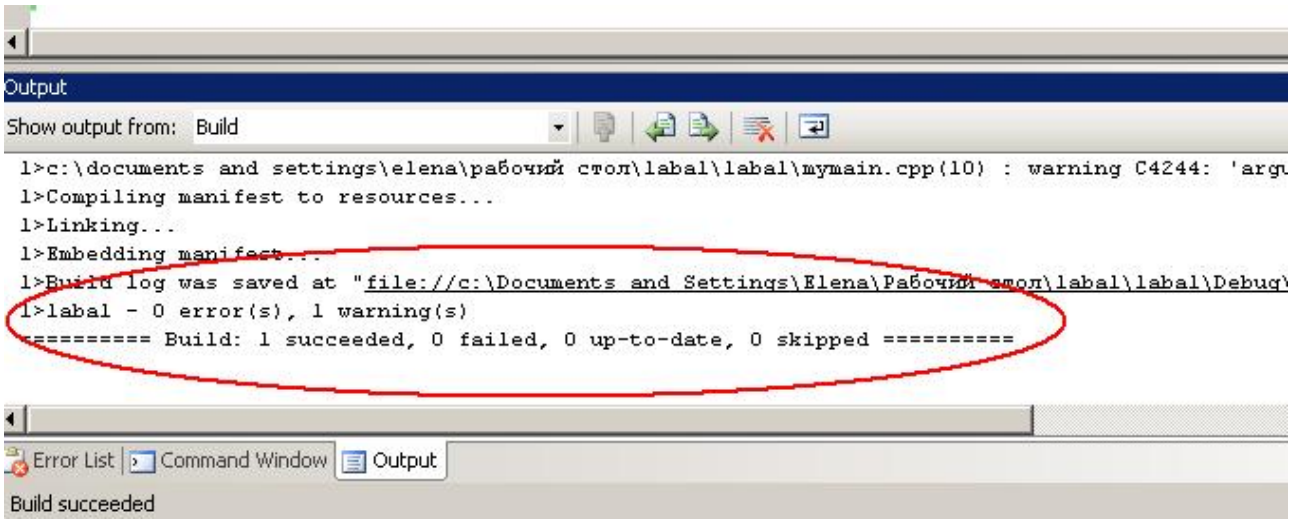

Рис. 10. Информация об успешной компиляции

Такое бывает редко, чаще всего появится информация об ошибках. В этом случае нужно дважды нажать на сообщение об ошибке, будет подсвечена строчка, где компилятор нашел ошибку (рис. 11.). Ее нужно будет исправить, после чего скомпилировать программу повторно. До тех пор, пока в программе есть ошибки, ее невозможно будет запустить на выполнение.

Программа автоматически сохраняется после компиляции.

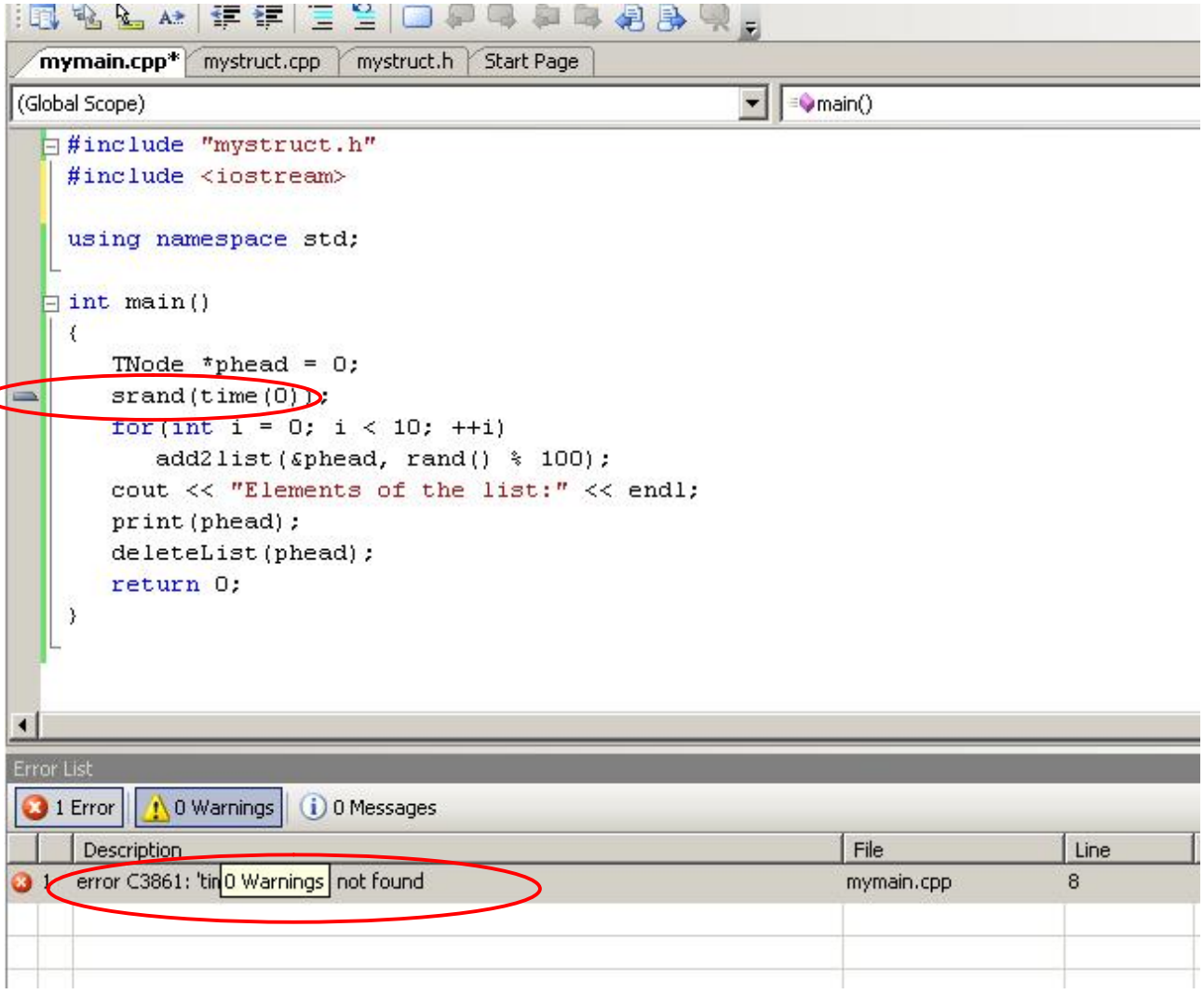

Рис. 11. Информация об ошибках в программе

Для запуска программы на выполнение необходимо выбрать пункт меню Debug->Start Without Debugging (см. рис. 12.).

Программа будет запущена на выполнение в консольном окне (см. рис. 13.).

Перед сдачей преподавателю программу нужно обязательно протестировать на корректность работы. Для этого необходимо задавать программе различные входные данные, проверять правильность полученных результатов.

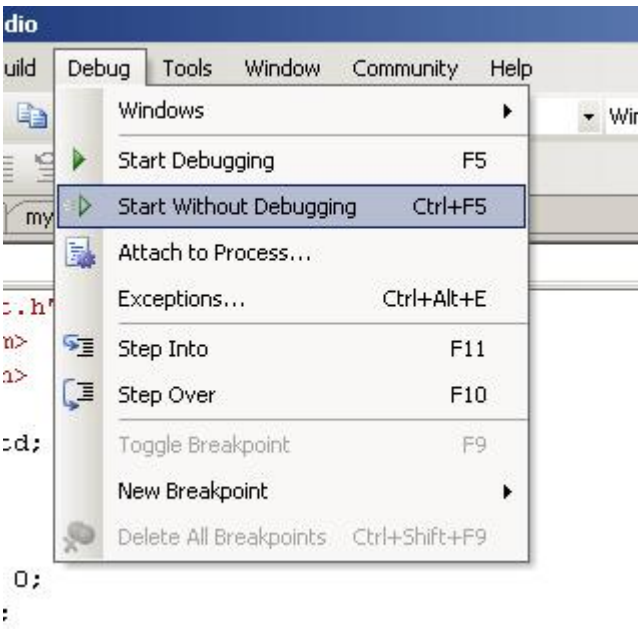

Рис. 12. Запуск программы на выполнение

Если программа запускается, но не работает (выдает неправильные результаты, работает не так, как предполагалось, выдает ошибку во время выполнения), то ее нужно отлаживать, для этого служит меню Debug. Программу нужно запустить в режиме Debug->Start Debugging и с помощью установки контрольных точек и пошагового выполнения найти логическую ошибку в коде.

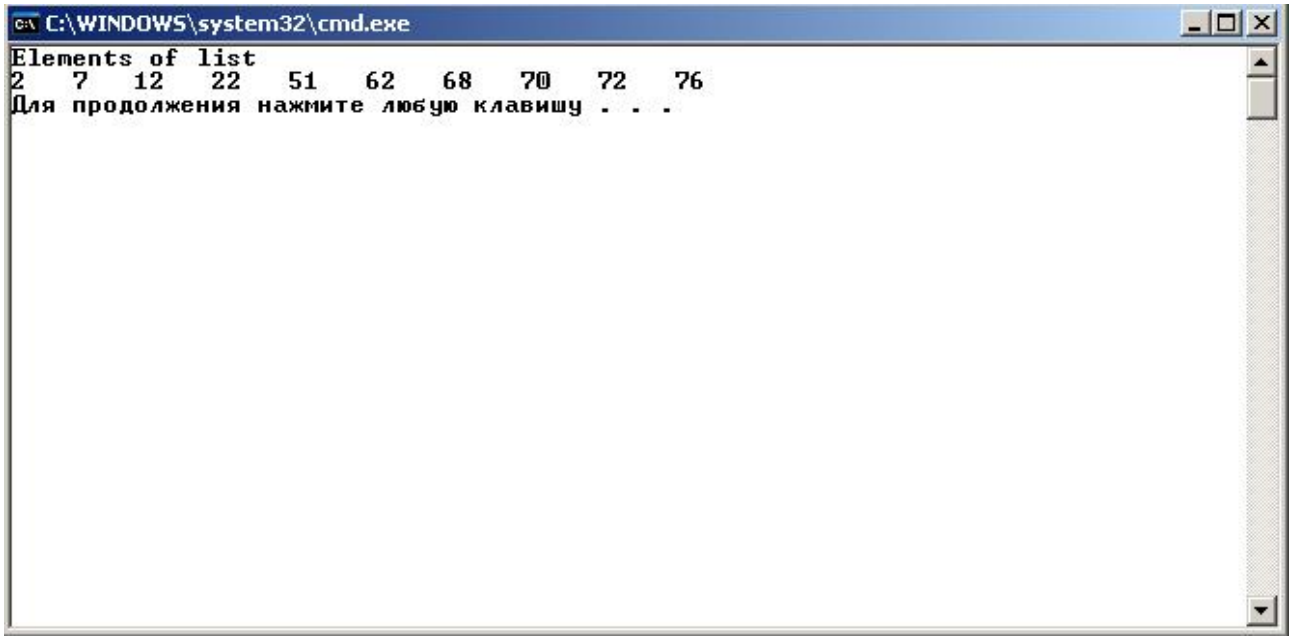

Рис. 13. Выполнение программы

1.2. РАБОТА С УКАЗАТЕЛЯМИ

Лабораторная работа 1

Цель работы

Изучить работу с указателями, взаимосвязи указателей и массивов, арифметику указателей.

Задание

Лабораторная работа может выполняться каждым студентом отдельно или парами студентов. Задание у каждого студента или у каждой пары студентов свое. Задание на лабораторную работу совпадает с номером компьютера, за которым сидит студент. Если студенты выполняют лабораторную работу вдвоем, то отчет оформляется также один на двоих.

При выполнении заданий необходимо память под массивы выделять динамически с помощью оператора new и освобождать в конце работы программы с помощью оператора delete.

1. Известны данные наблюдений за погодой за неделю. Каждый день показания погоды снимаются трижды. Определить самый теплый и самый холодный день за неделю. Определить самый теплый полдень и самый холодный полдень.

2. Написать программу, трансформирующую квадратную матрицу. Матрица – это таблица. В новой таблице столбцы должны стать строками, а строки – столбцами, т.е. матрица «поворачивается» вокруг главной диагонали.

3. Написать программу, которая определяет, является ли введенный пользователем квадрат – магическим. Магический квадрат – это квадрат, у которого сумма чисел в каждой строке, в каждом столбце и по каждой диагонали одинаковая.

4. Есть двумерный массив, в котором хранится следующая информация: номер дня недели, порядковый номер продавца, порядковый номер продукта, сумма, на которую данный продавец продал данного товара за день. Всего в фирме работает четыре продавца и продается пять видов продукции. Данные о продажах хранятся в течение недели. Определить, какой продавец продал больше всего третьего товара в понедельник.

5. Есть двумерный массив, в котором хранится следующая информация: номер дня недели, порядковый номер продавца, порядковый номер продукта, сумма, на которую данный продавец продал данного товара за день. Всего в фирме работает четыре продавца и продается пять видов продукции. Дан-

ные о продажах хранятся в течение недели. Определить, на какую сумму продал товара продавец с номером три.

6. Группу респондентов из 50 человек попросили сказать, за какую партию на выборах они будут голосовать. Всего в выборах участвует пять партий. Если респондент не определился, то его ответ засчитывался как «0». Если «Против всех», то «-1». Результат опроса поместили в одномерный массив. Нужно определить, какая партия победила в опросе, а также сосчитать, сколько голосов наберет каждая партия и вариант «Против всех».

7. Есть двумерный массив, в котором хранится следующая информация: номер дня недели, порядковый номер продавца, порядковый номер продукта, сумма, на которую данный продавец продал данного товара за день. Всего в фирме работает четыре продавца и продается пять видов продукции. Данные о продажах хранятся в течение недели. Определить, на какую сумму продали второго товара за эту неделю.

8. Группу студентов из 40 человек попросили оценить качество продуктов в студенческой столовой по десятибалльной шкале, где 1- отвратительно, а 10 – отлично. Результат опроса поместили в одномерный массив. Нужно определить среднее арифметическое ответов, наиболее часто встречающееся значение, а также, сколько раз встретилось каждое значение.

9. Есть двумерный массив, в котором хранится следующая информация: номер дня недели, порядковый номер продавца, порядковый номер продукта, сумма, на которую данный продавец продал данного товара за день. Всего в фирме работает четыре продавца и продается пять видов продукции. Данные о продажах хранятся в течение недели. Определить самого успешного продавца недели.

10. Есть двумерный массив, в котором хранится следующая информация: номер дня недели, порядковый номер продавца, порядковый номер продукта, сумма, на которую данный продавец продал данного товара за день. Всего в фирме работает четыре продавца и продается пять видов продукции. Данные о продажах хранятся в течение недели. Определить самый популярный товар недели.

11. Есть двумерный массив, в котором хранится следующая информация: номер дня недели, порядковый номер продавца, порядковый номер продукта, сумма, на которую данный продавец продал данного товара за день. Всего в фирме работает четыре продавца и продается пять видов продукции. Данные о продажах хранятся в течение недели. Определить удачный день недели.

12. Написать программу, которая получает от пользователя квадрат и определяет, все ли числа в нем совершенные. Совершенное число – это число, которое равно сумме всех своих делителей, включая единицу. Например, 6 = 1+2+3.

13. Написать программу, которая получает от пользователя квадрат и определяет, все ли числа в нем простые.

14. Написать программу, которая получает от пользователя квадрат и составляет новый квадрат, состоящий из факториалов чисел, содержащихся в первом квадрате.

15. Написать программу, которая получает от пользователя квадрат и составляет новый квадрат, состоящий из сумм цифр чисел, содержащихся в первом квадрате.

Отчет по лабораторной работе должен содержать

1. Титульный лист.
2. Цель работы.
3. Листинг программы с комментариями
4. Результат работы.
5. Выводы

Листинг программы должен быть набран шрифтом `Courier New`. Размер шрифта – 12.

На титульном листе отчета обязательно должен стоять номер лабораторной работы и номер выбранного варианта.

Вопросы

1. Операторы условия в С.
2. Операторы цикла в С.
3. Понятие массива.
4. Выделение памяти под массивы.
5. Сортировка и поиск в массиве.
6. Многомерные массивы.
7. Понятие указателя.
8. Взаимосвязь указателей и массивов.
9. Арифметика указателей.
10. Использование спецификатора const с указателями.
11. Динамическое выделение памяти под массивы.
12. Массивы указателей.

1.3. РАБОТА С ФУНКЦИЯМИ. РЕКУРСИЯ

Лабораторная работа 2

Цель работы

Изучить работу с функциями, передачу данных в функции, а также принцип рекурсивного решения задач.

Задание

Лабораторная работа может выполняться каждым студентом отдельно или парами студентов. Задание у каждого студента или у каждой пары студентов свое. Задание на лабораторную работу должно совпадать с номером компьютера, за которым сидит студент. Если студенты выполняют лабораторную работу вдвоем, то отчет оформляется также один на двоих.

1. Напишите рекурсивную функцию, которая печатает массив. Функция принимает массив и размер массива и ничего не возвращает. Функция прекращает свою работу и возвращается, если принимаемый массив имеет нулевой размер.
2. Напишите рекурсивную функцию, реализующую линейный поиск в массиве.
3. Напишите рекурсивную функцию, которая принимает массив и размер массива как аргументы и возвращает наименьший элемент массива. Функция должна прекращать свою работу и возвращаться, если принимаемый массив имеет один элемент.
4. Напишите рекурсивную функцию, которая принимает массив и размер массива как аргументы и возвращает наибольший элемент массива. Функция должна прекращать свою работу и возвращаться, если принимаемый массив имеет один элемент.
5. Напишите рекурсивную функцию, которая печатает массив в обратном порядке. Функция принимает массив и размер массива и ничего не возвращает. Функция прекращает свою работу и возвращается, если принимаемый массив имеет нулевой размер.
6. Напишите рекурсивную функцию, реализующую механизм двоичного поиска в отсортированном массиве.
7. Напишите рекурсивную функцию, которая переводит число из двоичной системы счисления в десятичную.
8. Напишите рекурсивную функцию, которая переводит число из шестнадцатеричной системы счисления в десятичную.

9. Напишите рекурсивную функцию, которая переводит число из восьмеричной системы счисления в десятичную.

10. Напишите рекурсивную функцию, которая заполняет массив целых чисел значениями, вводимыми пользователем с клавиатуры, в обратном порядке.

11. Напишите рекурсивную функцию, определяющую сумму элементов массива

12. Напишите рекурсивную функцию, определяющую максимальный элемент, лежащий на главной диагонали квадратной матрицы.

13. Напишите рекурсивную функцию, определяющую сумму элементов, лежащих на главной диагонали квадратной матрицы.

14. Напишите рекурсивную функцию, определяющую среднее арифметическое элементов, лежащих на главной диагонали квадратной матрицы.

15. Напишите рекурсивную функцию, определяющую минимальный элемент, лежащий на главной диагонали квадратной матрицы.

Отчет по лабораторной работе должен содержать

1. Титульный лист.
2. Цель работы.
3. Листинг программы с комментариями
4. Результат работы.
5. Выводы

Листинг программы должен быть набран шрифтом `Courier New`. Размер шрифта – 12.

На титульном листе отчета обязательно должен стоять номер лабораторной работы и номер выбранного варианта.

Вопросы

1. Понятие функции в С.
2. Передача массивов в функции.
3. Организация рекурсии в программе.
4. Классы памяти и область действия.
5. Указатели на функции.

1.4. РАБОТА СО СТРОКАМИ

Лабораторная работа 3

Цель работы

Понять принцип работы со строками в С.

Задание

Лабораторная работа может выполняться каждым студентом отдельно или парами студентов. Задание у каждого студента или у каждой пары студентов свое. Задание на лабораторную работу должно совпадать с номером компьютера, за которым сидит студент. Если студенты выполняют лабораторную работу вдвоем, то отчет оформляется также один на двоих.

Все задания должны быть выполнены без использования стандартных функций для работы со строками библиотеки string.h и других библиотек.

1. Напишите программу, которая получает строки от пользователя и удаляет лишние пробелы

2. Напишите программу, которая находит в строке числа и выводит их в одну строку. Строки вводятся по одной в зависимости от желания пользователя (признак конца ввода «-1»)

3. Напишите программу, которая меняет регистр вводимых букв.

4. Напишите программу, которая выводит на консоль непечатные символы.

5. Напишите программу, которая находит и выводит все числа, находящиеся во введенных строках, преобразуя их к целочисленному типу.

6. Напишите программу, которая выводит на консоль введенные символы в обратном порядке.

7. Напишите программу, которая читает строки и выводит их на экран таким образом, что строки располагаются в порядке убывания количества символов.

8. Напишите программу, которая читает строки и выводит их на экран отдельными лексемами (по одному слову на строке).

9. Напишите программу, которая выводит на консоль строки в обратном порядке.

10. Напишите программу, которая проверяет введенные строчки на повтор последнего символа.

11. Напишите программу, которая выводит сумму всех чисел, находящихся во введенных строках.

12. Напишите программу, которая выводит самое длинное слово во введенных строчках.

13. Напишите программу, которая проверяет введенные строчки на повтор последнего слова.

14. Напишите программу, которая выводит самое короткое слово во введенных строчках.

15. Напишите программу, которая находит в строке числа и выводит на экран их сумму. Строки вводятся по одной в зависимости от желания пользователя (признак конца ввода «-1»)

Отчет по лабораторной работе должен содержать

1. Титульный лист.
2. Цель работы.
3. Листинг программы с комментариями
4. Результат работы.
5. Выводы

Листинг программы должен быть набран шрифтом `Courier New`. Размер шрифта – 12.

На титульном листе отчета обязательно должен стоять номер лабораторной работы и номер выбранного варианта.

Вопросы

1. Понятие символа в С.
2. Понятие строки в С.
3. Основы обработки строк в С.
4. Библиотечные функции для работы со строками.

1.5. РАБОТА С ФАЙЛАМИ

Лабораторная работа 4

Цель работы

Понять принцип работы с файлами в С.

Задание

Лабораторная работа может выполняться каждым студентом отдельно или парами студентов. Задание у каждого студента или у каждой пары студентов свое. Задание на лабораторную работу должно совпадать с номером компьютера, за которым сидит студент. Если студенты выполняют лабораторную работу вдвоем, то отчет оформляется также один на двоих.

Для выполнения лабораторной работы нужно пользоваться функциями стандартной библиотеки `stdio.h`

1. Напишите программу, которая читает текстовый файл и записывает новый файл, в котором строки располагаются в порядке убывания количества символов.

2. Напишите программу, которая читает текстовый файл и записывает новый файл, в котором предложения располагаются в порядке убывания количества символов.

3. Напишите программу, которая читает файл в формате html и записывает новый файл, в котором удалены все теги.

4. Напишите программу, которая читает текстовый файл и удаляет все лишние пробелы. Результат записывается в новый файл.

5. Напишите программу, которая читает строку текста из файла и записывает в новый текстовый файл предложение с обратным порядком слов.

6. Напишите программу, которая построчно читает текстовый файл и записывает в новый файл обратный порядок символов в строках.

7. Напишите программу, которая читает текстовый файл, являющийся программой на C++, и записывает новый файл, содержащий ту же программу без комментариев.

8. Напишите программу, которая читает текстовый файл и записывает новый файл, в котором строки располагаются в алфавитном порядке.

9. Напишите программу, которая читает текстовый файл и записывает новый файл, в котором строки располагаются в обратном алфавитном порядке.

10. Напишите программу, которая читает текстовый файл и печатает таблицу, показывающую, сколько раз в тексте встречается каждая буква алфавита.

11. Напишите программу, которая читает текстовый файл и печатает таблицу, показывающую, сколько раз в тексте встречаются гласные буквы русского алфавита без учета регистра символов.

12. Напишите программу, которая читает текстовый файл и записывает в новый файл самое короткое слово первого файла, а также сколько раз это слово встретилось в файле.

13. Напишите программу, которая читает текстовый файл, являющийся программой на C++, и записывает новый файл, количество использованных циклов в программе. Если цикл закомментирован, то он считаться не должен.

14. Напишите программу, которая читает текстовый файл, являющийся программой на C++, и записывает новый файл, количество использованных операторов условия в программе. Если условие закомментировано, то оно считаться не должно.

15. Напишите программу, которая читает текстовый файл и записывает в новый файл самое длинное слово первого файла, а также сколько раз это слово встретилось в файле.

Отчет по лабораторной работе должен содержать

1. Титульный лист.
2. Цель работы.
3. Листинг программы с комментариями
4. Результат работы.
5. Выводы

Листинг программы должен быть набран шрифтом `Courier New`. Размер шрифта – 12.

На титульном листе отчета обязательно должен стоять номер лабораторной работы и номер выбранного варианта.

Вопросы

1. Понятие файла в С.
2. Функции для работы с файлами стандартной библиотеки.

1.6. РАБОТА СО СТРУКТУРАМИ

Лабораторная работа 5

Цель работы

Понять принцип организации структур, работы со структурами, передачи структур в функции, выделения памяти под структуры. Понять принцип модульного программирования.

Задание

Лабораторная работа может выполняться каждым студентом отдельно или парами студентов. Задание у каждого студента или у каждой пары студентов свое. Задание на лабораторную работу должно совпадать с номером компьютера, за которым сидит студент. Если студенты выполняют лабораторную работу вдвоем, то отчет оформляется также один на двоих.

Лабораторная работа должна быть выполнена в трех файлах. Один файл заголовочный (например, `My.h`), в нем будет содержаться описание структуры, а также прототипы функций, работающих со структурой. Один файл с кодом (например, `My.cpp`). В нем будет описание функций, прототипы которых указаны в заголовочном файле. Имена файлов должны совпадать. Третий файл также должен быть с кодом (например, `Test.cpp`). В нем будет всего одна функция – `main`. В этой функции будет объявление объектов типа описанной структуры и работа с созданными пользовательскими функциями.

Заголовочный файл должен быть включен в оба файла с кодом.

1. Разработать структуру и функции для работы с календарем.
2. Разработать структуру и функции для работы с таблицей.
3. Разработать структуру и функции для работы с комплексными числами.
4. Разработать структуру и функции для работы с библиотекой.
5. Разработать структуру и функции для работы с коллекцией картин.
6. Разработать структуру и функции для работы с телефонной книжкой.
7. Разработать структуру и функции для расчета траектории движения тела, брошенного под углом к горизонту.
8. Разработать структуру и функции для расчета основных параметров электрической цепи.
9. Разработать структуру и функции для работы с очередью.

10. Разработать структуру и функции для работы с бинарными деревьями.
11. Разработать структуру и функции для работы в циклической арифметике.
12. Разработать структуру и функции для работы с шестнадцатеричными числами.
13. Разработать структуру и функции для работы в арифметике с насыщением.
14. Разработать структуру и функции для работы с двоичными числами (предусмотреть возможность работы с большими числами – до 100 разрядов).
15. Разработать структуру и функции для работы с двунаправленным списком.

Отчет по лабораторной работе должен содержать

1. Титульный лист.
2. Цель работы.
3. Листинг программы с комментариями
4. Результат работы.
5. Выводы

Листинг программы должен быть набран шрифтом `Courier New`. Размер шрифта – 12.

На титульном листе отчета обязательно должен стоять номер лабораторной работы и номер выбранного варианта.

Вопросы

1. Определение структур.
2. Доступ к элементам структур.
3. Передача структур в функции.
4. Объединения.
5. Битовые поля.
6. Структуры с самоадресацией.
7. Понятие стека.
8. Понятие очереди.
9. Понятие дерева.

1.7. РАБОТА С КЛАССАМИ

Лабораторная работа 6

Цель работы

Понять принцип организации классов, работу с классами. Научиться проводить объектную декомпозицию задачи. Понять отличие объектно-ориентированного программирования от структурного подхода.

Задание

Лабораторная работа может выполняться каждым студентом отдельно или парами студентов. Задание у каждого студента или у каждой пары студентов свое. Задание на лабораторную работу должно совпадать с номером компьютера, за которым сидит студент. Если студенты выполняют лабораторную работу вдвоем, то отчет оформляется также один на двоих.

Лабораторная работа должна быть выполнена как минимум в трех файлах. Один файл заголовочный (например, `My.h`), в нем будет содержаться описание класса. Один файл с кодом (например, `My.cpp`). В нем будет описание функций класса. Имена файлов должны совпадать. Третий файл также должен быть с кодом (например, `Test.cpp`). В нем будет всего одна функция – main. В этой функции будет объявление объектов типа описанного класса и работа с созданными объектами. На каждый класс необходимо предусмотреть свой модуль.

В каждом классе необходимо перегрузить операции +, =, ++, <<, >>, <, >, !=, [], и т.д.

Также в классе нужно определить конструктор по умолчанию, конструктор инициализации, конструктор копирования, и деструктор.

Разработать классы:

1) для работы со строками. Предусмотреть такие функции как сложение строк, копирование, поиск элемента, вырезание элемента из строки, смену регистров и т.д.;
2) для работы со связным двунаправленным списком;
3) для работы с вектором произвольной длины;
4) для работы с телефонной книжкой;
5) для работы с абстрактной таблицей;
6) для работы с графом;
7) для моделирования полета самолета;

8) для моделирования работы лифта;
9) для представления календаря;
10) для работы с комплексным числом;
11) для работы с множеством;
12) для работы с бинарным деревом;
13) для программы «Ипподром»;
14) для работы с дробями;
15) для программы расчета кредитных выплат банку.

Отчет по лабораторной работе должен содержать

1. Титульный лист.
2. Цель работы.
3. Листинг программы с комментариями
4. Результат работы.
5. Выводы

Листинг программы должен быть набран шрифтом `Courier New`. Размер шрифта – 12.

На титульном листе отчета обязательно должен стоять номер лабораторной работы и номер выбранного варианта.

Вопросы

1. Понятие класса и объекта в C++.
2. Отделение интерфейса от реализации. Инкапсуляция данных.
3. Специальные члены класса.
4. Отношения дружественности.
5. Константные функции. Константные объекты.
6. Статические элементы класса.
7. Наследование.
8. Перегрузка операций.
9. Полиморфизм. Позднее связывание.
10. Шаблоны классов.
11. Обработка ошибок в программе на C++.

1.8. РЕКОМЕНДАЦИИ К КУРСОВОМУ ПРОЕКТУ

Общие сведения по курсовому проекту

Курсовой проект представляет собой результат решения инженерных задач по разработке программного обеспечения различного назначения.

Задачи курсового проекта и требования к проекту

Курсовое проектирование имеет своей целью:

- расширение, углубление и систематизацию теоретических знаний и практических навыков, полученных в курсе «Программирование на языке высокого уровня» и предшествующих курсов;
- закрепление навыков самостоятельной работы, совершенствование в овладении методами принятия технических решений;
- развитие умения разрабатывать и читать технические документы, составлять и технически грамотно оформлять результаты проделанной работы.

Курсовой проект представляет собой разработку, в которой решается задача проектирования и реализации программного обеспечения различного назначения от создания игр до моделирования жизненных процессов.

Тематика курсовых проектов

Темы курсовых проектов должны соответствовать учебному плану.

Ценность курсовых проектов во многом определяется глубиной разработки темы, поисков различных вариантов решения проблемы, оригинальности разработки, четкостью выводов.

Примеры тем для курсового проектирования приведены в прил. 2.

Комплектность готового проекта

Указания по комплектованию курсового проекта

Курсовой проект представляет собой разработанное программное обеспечение, а также совокупность разработанных текстовых и графических документов. Программное обеспечение реализуется в виде исполняемого файла и набора необходимых библиотек, собранных в одну папку. Папка должна быть представлена на дискете или компакт-диске.

Текстовые документы выполняются машинописным способом и комплектуются в отдельную папку.

Первым листом в папке является титульный лист курсового проекта, который выполняется по установленному образцу. Далее в папку помещаются текстовые документы в следующем порядке:

- задание на курсовое проектирование,
- пояснительная записка (ПЗ).

Текстовые документы курсового проекта оформляются в соответствии с ГОСТ 2.105-95, ГОСТ 2.106-95 и СТП1-У-НГТУ-93. Физические величины обозначаются в них согласно ГОСТ 8.417-81 и ОСТ 4.000.001.

Задание на курсовое проектирование

Задание на курсовое проектирование содержит указание предметной области, функциональные и количественные требования к системе, а также другие требования, определяющие индивидуальность системы.

К функциональным требованиям относятся основные функции, выполняемые системой, конфигурация и распределенные свойства системы, наличие интерфейсов связи с другими информационными системами.

К количественным параметрам системы могут быть отнесены количество пользовательских рабочих мест, объем хранимых данных (количество записей в наиболее критических таблицах), требования к производительности (время выполнения запросов, время реакции на команды пользователя и др., скорость обработки потоков данных), дополнительные технические требования

Количественные параметры могут уточняться после предварительного проектирования системы.

Пояснительная записка

В пояснительной записке студент в краткой и четкой форме должен изложить творческий замысел курсового проекта, примененные методы и технические решения, результаты расчетов и исследований.

Решение каждого из вопросов курсового проекта следует начинать с изучения литературных источников. В результате этого должны быть предложены технические решения (обычно 2..3), удовлетворяющие поставленным требованиям. Далее, на основе их сравнения, выбирается наиболее удачное решение. Не следует рассматривать заведомо неприемлемые варианты решения задачи.

Все принятые разработчиком технические решения должны быть обоснованы. В качестве обоснования могут использоваться сравнение предложенных вариантов, рекомендации литературных источников (с указанием конкретных условий их применимости), результаты расчетов.

Текстовый и иллюстративный материал, взятый из литературы и др. источников, допускается приводить лишь в исключительных случаях, когда без этого невозможно выполнить расчет, сделать выводы и т.д., т.е. когда просто ссылка на соответствующий источник недостаточна.

Содержание должно быть конкретным и относиться только непосредственно к теме разработки. Не допускается переписывание из литературы в пояснительную записку определений, выводов соотношений, общих положений и пр. Достаточно сослаться на конечный результат.

Рекомендуемое содержание пояснительной записки может выглядеть следующим образом:

Содержание
Введение
1 Разработка и анализ технического задания
1.1 Описание предметной области
1.2 Разработка технического задания
1.2.1 Назначение разработки
1.2.2 Область применения
1.2.3 Функциональные требования к системе
1.2.4 Количественные требования к системе
1.2.5 Требования к техническим средствам
1.2.6 Требования к безопасности и целостности информации
1.2.7 Требования к информационной и программной совместимости
1.2.8 Требования к графическому интерфейсу
1.3 Анализ технического задания
1.4 Выбор методов и средств решения технического задания
2 Разработка программного обеспечения
2.1 Разработка алгоритмов
2.2 Разработка классов
2.3 Программная реализация
2.3.1 Правила именования переменных
2.3.2 Структура программы
2.3.3 Организация входных и выходных данных
3 Расчеты и оценки
4 Руководство пользователя
4.1 Инсталляция программы
4.2 Описание интерфейса
4.3 Работа с программой
Заключение
Список используемой литературы
Приложения

Пояснительная записка начинается с титульного листа. Далее следует содержание пояснительной записки, первый лист которого оформляется как заглавный лист пояснительной записки по форме 5 ГОСТ 2.104-68.

Во **введении** кратко рассматривается современное состояние инженерной или научной задачи, решению которой способствует выполнение курсового проекта. Указывается степень новизны (новая разработка или модернизация существующей). Раздел занимает 1-2 листа.

В разделе **Разработка и анализ технического задания** описывается предметная область, разрабатываемого программного обеспечения и четко

формулируются задачи, которые решаются в курсовом проекте. Раздел занимает 6-7 листов. Этот раздел может состоять из следующих частей.

1. ***Исследование (описание) предметной области***, в которой формулируются основные требования и особенности предметной области, влияющие на разработку курсового проекта.

2. ***Разработка технического задания***. В техническом задании приводится конкретная формулировка требований, которым должна удовлетворять проектируемая система. Среди них:

- функциональные требования к системе (основные функции, выполняемые системой);
- количественные требования к системе (количество рабочих мест, количество и объем записей, объем файлов данных, требуемое время реакции системы, время выполнения запросов, скорость обработки потоков данных и др.);
- требования по безопасности и целостности информации (категории доступа пользователей к той или иной информации);
- требования по совместимости (наличие интерфейсов связи с другими информационными системами, совместимость с прежними форматами данных).

3. ***Анализ задания***. В этом пункте приводится содержательная постановка задачи, в которой анализируются возможные способы реализации функциональных, количественных и других требований к программному обеспечению.

4. ***Выбор способов и средств***, в которой осуществляется выбор методологии (структурная, объектно-ориентированная, другая) и инструментальных средств (сервер БД, язык программирования, методы защиты информации и т.д.) решения поставленных задач.

В разделе **Разработка программного обеспечения** описываются особенности реализации программного обеспечения, разрабатываемого в ходе выполнения курсового проекта: особенности алгоритмов функционирования программы, полученная иерархия классов в программе, детали, связанные с написанием кода. Раздел занимает 6-7 страниц. Этот раздел может состоять из следующих частей:

1. ***Разработка алгоритмов***. Здесь необходимо рассмотреть общий алгоритм работы программы, а также описать оригинальные алгоритмы наиболее важных частей программного обеспечения курсового проекта.

2. ***Разработка классов***. При разработке курсового проекта необходимо создать иерархическую древовидную структуру классов. Для этого необходимо провести объектную декомпозицию задачи и выделить главные сущности. В этом пункте нужно описать пользовательские классы, их свойства и методы, а также отношения, в которых находятся классы между собой.

3. ***Программная реализация***. В этом пункте необходимо рассмотреть особенности реализации кода программного обеспечения: соглашения по именованию переменных, структуру программы, а также реализацию входных и выходных данных.

В разделе **Расчеты и оценки** проводятся следующие расчеты, отражающие специфику разрабатываемого программного обеспечения:

- расчет (оценка) требуемых ресурсов вычислительных средств. Служит для количественного обоснования выбора технических средств;

- оценка загрузки вычислительных средств, оценка производительности. Служит для обоснования выполнения требований технического задания по времени реакции системы на запросы и другие действия пользователей, оценки доли времени или доли использования ресурсов технических средств.

Раздел занимает 1-2 листа.

В разделе **Руководство пользователя** содержится информация по работе конечного пользователя с программным обеспечением. Раздел занимает 5-6 страниц. Может состоять из следующих пунктов:

1. ***Инсталляция программы***. Содержит сведения, необходимые для установки программного продукта. Необходимые библиотеки, пути, программное обеспечение сторонних разработчиков.

2. ***Описание интерфейса***. В этом пункте приводятся соглашения по пользовательскому интерфейсу на примере одной из типовых форм программы.

3. ***Работа с программой***. Здесь необходимо объяснить конечному пользователю особенности работы с программным продуктом.

В разделе **Заключение** приводятся выводы о степени соответствия выполненного проекта техническому заданию, а также возможные способы дальнейшего улучшения программного продукта. Этот пункт занимает 1-2 листа.

После заключения дается **Список литературы**, на которую делаются ссылки в пояснительной записке. В списке литературы могут быть указаны только опубликованные источники и страницы Интернета. Не допускается наличие в списке литературы пунктов типа

- Курс лекций по ПЯВУ

- www.citforum.ru.

В списке литературы могут указываться только источники, на которые имеется ссылка в тексте пояснительной записки. Список литературы должен состоять не менее чем из 3 источников.

В **Приложении** приводится листинг программы (полный или частичный – на усмотрение студента), а также скрин-шоты всех форм и сообщений системы. Нумерация страниц в приложениях отдельная - в каждом из них с 1-й страницы.

Общий объем пояснительной записки с приложениями - 25-40 стр.

Порядок выполнения и защиты курсовых проектов

Курсовой проект выполняется каждым студентом индивидуально в установленный преподавателем срок. Вся ответственность за выполнение курсового проекта и принятые в процессе решения лежит на студенте.

Во время выполнения курсового проекта студент должен посещать консультации для предоставления руководителю промежуточных результатов и обсуждения предложенных технических решений. График консультаций пре-

доставляется руководителем на первой консультации по курсовому проектированию

После получения темы студент должен разработать техническое задание на курсовое проектирование и согласовать его с руководителем курсового проекта. Затем студент анализирует поставленное техническое задание, выбирает способы и средства его решения. На следующем этапе разрабатываются алгоритмы, объекты, модули для решения задачи. Результат проектирования предоставляется на обсуждение руководителю. После согласования студент приступает к написанию кода программного продукта: реализации объектов, алгоритмов, разработке интерфейса пользователя. Затем первая (альфа) версия программы согласовывается с руководителем проекта, фиксируются все ошибки и недочеты, а также пожелания руководителя. После этого проект дорабатывается. После окончательной доработки проекта оформляется проектная документация (см. п.4)

Защита проекта происходит в установленный руководителем или деканатом день. На защите студент обязан иметь работоспособную версию разработанного программного обеспечения и полный комплект технической документации. Во время защиты студент кратко рассказывает о своем проекте, особенностях и новизне разработки, после чего отвечает на вопросы. Руководитель смотрит работу программы и проверяет документацию на проект. В случае неполной комплектации проект не может считаться защищенным. Проект не может считаться защищенным, если студент не знает основных принципов работы программы, содержания пояснительной записки. Если на защиту представлены два одинаковых проекта, то оба будут сняты с защиты и отправлены на переделку.

К защите не допускаются проекты с несогласованным техническим заданием или с несогласованной альфа-версией.

2. КОНТРОЛЬ ЗНАНИЙ

Экзаменационные вопросы

1. Классификация языков программирования. Свойства языков программирования
2. История и назначение языка C/C++
3. Основные парадигмы программирования
4. Понятие переменной. Встроенные типы данных и объявление переменных. Константные переменные. Перечисления
5. Операции и выражения
6. Понятие оператора. Пустой оператор. Оператор return
7. Оператор if (ЕСЛИ)
8. Оператор while (ПОКА)
9. Оператор do/while (ВЫПОЛНЯТЬ/ПОКА)
10. Оператор for (ЦИКЛ)
11. Оператор множественного выбора switch
12. Определение массива. Объявления и инициализация массивов в программе
13. Сортировка массивов
14. Поиск в массивах
15. Многомерные массивы
16. Объявления и инициализация переменных указателей
17. Операции над указателями
18. Выражения и арифметические действия с указателями
19. Использование спецификатора const с указателями
20. Взаимосвязи между указателями и массивами
21. Массивы указателей
22. Динамическое выделение памяти под массивы.
23. Определения функций
24. Классы памяти. Область действия
25. Рекурсия

26. Ссылки и ссылочные параметры. Вызов функций по ссылке с аргументами указателями
27. Перегрузка функций
28. Передача массивов в функции
29. Указатель на функцию
30. Введение в обработку строк
31. Работа с файлами
32. Компоновка программ и препроцессор
33. Определение структур. Доступ к элементам структур
34. Использование структур
35. Битовые поля. Объединения
36. Построение связных списков на основе структур с самоадресацией
37. Определения классов. Отделение интерфейса от реализации
38. Область действия класс и доступ к элементам класса. Управление доступом к элементам. Функции доступа
39. Специальные члены класса
40. Классы как элементы других классов. Дружественность
41. Константные объекты и константные функции. Статические элементы класса
42. Одиночное наследование. Типы наследования
43. Множественное наследование. Виртуальные базовые классы
44. Перегрузка операций
45. Виртуальные функции
46. Абстрактные базовые классы и конкретные классы
47. Полиморфизм. Новые классы и динамическое связывание
48. Шаблоны функций
49. Шаблоны классов
50. Понятие особой ситуации или исключения. Особые ситуации и традиционная обработка ошибок
51. Динамическая идентификация типов
52. Операция reinterpret_cast. Операция const_cast
53. Операция static_cast. Операция dynamic_cast

ГЛОССАРИЙ

Абстрактный класс	– это класс, объекты которого в программе создать нельзя
Адрес переменной	– это адрес области памяти, с которой связана данная переменная
Время жизни	– это время, в течение которого переменная связана с определенной областью памяти
Деструктор	– функция, вызываемая в момент разрушения объекта. Содержит освобождение динамической памяти, используемой объектом. Имя совпадает с именем класса, но в начале ставится знак ~
Дружественность	– отношения между классами или между классом и внешней функцией, позволяющие получить доступ к закрытым и защищенным элементам класса
Заголовочный файл	– файл, содержащий прототипы функций и описание пользовательских типов данных. Имеет расширение .h
Значение переменной	– это содержимое ячейки или совокупности ячеек памяти, связанных с данной переменной
Имя переменной	– это идентификатор, используемый в программах для ссылки на значение переменной
Индекс	– номер позиции элемента массива, заключенный в квадратные скобки

Инкапсуляция	– один из основных принципов объектно-ориентированного программирования. Позволяет скрывать реализацию класса от конечного пользователя класса
Исключение	– событие, которого не ждали и которое нужно каким-то образом обрабатывать
Класс	– это абстрактный тип данных, состоящий из элементов-данных и элементов-функций
Компилятор	– программа, обрабатывающая пользовательскую программу, проверяющая синтаксические ошибки, подключающая библиотечные файлы и формирующая исполняемый файл
Композиция	– включение в класс объектов других типов в качестве элементов
Конкретный класс	– класс, объекты которого в программе создать можно
Конструктор	– функция класса, используемая для инициализации элементов-данных класса. Имя конструктора совпадает с именем класса
Массив	– последовательная группа ячеек памяти, имеющих одинаковое имя и одинаковый тип
Наследование	– один из способов повторного использования программного обеспечения, при котором новые классы создаются из уже существующих классов путем заимствования их функций и атрибутов и обогащения этими возможностями новых классов

Область видимости	– это блок программы, из которого можно обратиться к этой переменной
Перегрузка функций	– определение в программе нескольких функций с одним именем, но разным списком параметров
Перегрузка операций	– переопределение встроенных операций для пользовательских типов данных
Переменная	– это объект данных, который явным образом определен и именован в программе
Полиморфизм	– возможность для объектов разных классов, связанных с помощью наследования, реагировать различным образом при обращении к одной и той же функции-элементу
Программа	– алгоритм + структура данных
Прототип функции	– это заголовок функции, после которого ставится точка с запятой. Необходим для использования функции до ее определения
Рекурсивная функция	– это функция, которая вызывает сама себя либо непосредственно, либо через другие функции
Связный список	– разновидность линейных структур данных, представляющая собой последовательность элементов, обычно отсортированную в соответствии с заданным правилом
Спецификатор доступа	– один из private, protected, public. Определяет доступ к элементу пользовательского класса из внешних функций или объектов других типов

Ссылка, ссылочный параметр	– это скрытый указатель, псевдоним параметра
Строка	– это массив символов, заканчивающийся специальным символом конца строки
Структура	– это абстрактный тип данных, состоящий из элементов-данных
Тип переменной	- связывает переменную с множеством значений, которые она может принимать
Указатель	– переменная, которая содержат в качестве своих значений адреса памяти.
Файл программы (кода)	– файл, содержащий определение пользовательских функций. Имеет расширение .cpp
Функция	– именованный блок программы, созданный для решения одной небольшой задачи
Шаблон класса	– параметризованный тип. Задает способ получения семейства сходных классов
Шаблон функции	– позволяет определять функции для любого типа данных. В отличие от перегруженных функций работа по созданию функций для конкретного типа ложится на компилятор

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

Основная литература

1. **Дейтел, Пол Дж**. Как программировать на C++ / Пол Дж. Дейтел, Харви Дейтел. – М.: ЗАО Издательство Бином, 1999. – 1024 с.

2. **Каррано, Ф.М.** Абстракция данных и решение задач на C++. Стены и зеркала / Ф.М. Каррано, Дж.Дж. Причард. – 3-е изд. – М.: Издательский дом «Вильямс», 2003. – 848 с.

3. **Страуструп, Б**. Язык программирования C++. Специальное издание / Б. Страуструп. – М.: Бином-Пресс, 2006. – 1104 с.

4. **Коплиен, Дж.** Программирование на C++. Классика CS / Дж. Коплиен. – СПб.: Питер, 2005. – 479 с.

5. **Элджер, Дж.** C++: библиотека программиста / Дж. Элджер — СПб.: Питер, 1999. – 259 с.

6. **Культин, Н.** C/C++ в задачах и примерах / Н. Культин. – СПб.: БХВ-Петербург, 2005. – 288 с.

7. **Керниган, Б.** Язык программирования C / Б. Керниган, Д. Ритчи. – 2-е изд. – М.: Издательский дом «Вильямс», 2006. – 304 с.

8. **Давыдов, В.Г.** Программирование и основы алгоритмизации: учебное пособие / В.Г. Давыдов. – М.: Высшая школа, 2003. – 447 с.

9. **Александреску, А.** Современное проектирование на C++ / А. Александреску. – М.: Издательский дом «Вильямс», 2002. – 336 с.

10. **Сатер, Г.** Новые сложные задачи на C++ / Г. Сатер. – М.: Издательский дом «Вильямс», 2005. – 272 с.

Дополнительная литература

11. **Мэйерс, С**. Эффективное использование C++. 55 верных способов улучшить структуру и код ваших программ / Скотт Мэйерс. – М.: ДМК Пресс, 2006. – 300 с.

12. **Мэйерс, С**. Наиболее эффективное использование C++. 35 новых рекомендаций по улучшению ваших программ и проектов / Скотт Мэйерс. – М.: ДМК Пресс, 2000. – 304 с.

13. **Хэзфилд, Р.** Искусство программирования на С. Фундаментальные алгоритмы, структуры данных и примеры приложений. Энциклопедия программиста / Ричард Хэзфилд [и др.]. – Киев: Изд-во «ДиаСофт», 2001. – 736 с.

14. **Дэвис, С.Р.** C++ для «чайников» / Стефан Р. Дэвис. – М.: Издательский дом «Вильямс», 2003. – 336 с.

15. **Лафоре, Р.** Объектно-ориентированное программирование в C++ / Р. Лафоре. – 4-е изд. – СПб.: Питер, 2004. – 923 с.

16. **Якушев, Д.М.** «Философия» программирования на C++ / Д.М. Якушев. – 2-е изд. – М.: Бук-пресс, 2006. – 320 с.

17. **Джосьютис, Н.** C++ стандартная библиотека. Для профессионалов / Н.М. Джосьютис. – СПб.: Питер, 2004. – 730 с.

18. **Щупак, Ю.А.** Win32 API. Эффективная разработка приложений / Ю.А. Щупак. – СПб.: Питер, 2007. – 572 с.

19. **Фленов, М.Е.** Программирование на C++ глазами хакера / М.Е. Фленов. – СПб.: БХВ–Петербург, 2004. – 336 с.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1

Разработать программу, моделирующую функционирование экологически замкнутой системы.

Введение

Компьютеры давно стали неотъемлемой частью нашей жизни. С их помощью решаются самые разнообразные задачи – сложные, трудоемкие научные вычисления, задачи различного моделирования – от сложных механизмов до простейших физических законов, экономические расчеты, развивающие игры и т.д.

Моделирование экологических процессов – задача очень важная и актуальная. В связи с активной деятельностью человека во всем мире происходят необратимые изменения, эти процессы не могут не вызывать беспокойства человека. В реальности моделирование поведения экологически замкнутой биологической системы практически невозможно, единственная реальная возможность – это создание аквариума.

Компьютер позволяет решить эту проблему с помощью компьютерного моделирования. Получаемая модель может быть сколь угодно сложной, все зависит от поставленной задачи, имеющихся технических средств и т.д.

Подобное моделирование позволит выявить определяющие факторы, влияющие на равновесие биосистемы, что, в конечном итоге, поможет экологам сохранить равновесие в реальных биосистемах.

В рамках данного курсового проекта решается задача моделирования замкнутой биосистемы на примере леса. При разработке модели были выявлены три основных участника экосистемы – волки, зайцы и трава. Все они существуют на замкнутом пространстве, необходимо подобрать такие начальные параметры, при которых система будет продолжительное время находиться в равновесии.

1 Разработка и анализ технического задания

1.1 Исследование предметной области

Любое моделирование подразумевает под собой создание модели исследуемого объекта, соответственно, отказ от некоторых свойств объекта, неважных при решении поставленной задачи.

В моделируемой системе будут присутствовать объекты трех типов – волки, зайцы и трава.

Трава живет, размножается и умирает. Трава не может размножиться в ту точку леса, где уже кто-то находится. Трава в реальной жизни получает пита-

тельные вещества из почвы, в нашем проекте эта особенность никак не учитывается.

Зайцы живут, размножаются, питаются и умирают. Умереть заяц может по двум причинам: от старости и от голода. Заяц питается травой. Если травы нет и заяц в течение определенного времени не может ничего съесть, он умирает от голода. Размножаются зайцы через определенный промежуток времени. Для упрощения моделирования у зайца нет понятия пола, новый заяц появляется путем простого клонирования на соседнюю свободную точку леса. Если свободных точек нет, но рядом есть трава, то новый заяц вполне может появиться. А если свободных точек нет, но рядом только зайцы или волки, то новый заяц появиться не может. Таким образом, будет происходить регуляция численности популяции зайцев.

Волки живут, размножаются, питаются и умирают. Умереть волк, так же, как и заяц, может от голода и от старости. Питается волк только зайцами. Через определенные промежутки времени волк размножается. Так же для упрощения модели у волка нет понятия пола, то есть волк, как и заяц, размножается клонированием. Если есть свободные соседние клетки, то на какой-либо из них может появиться новый волк.

Система умирает, когда в ней не останется ни одного объекта. Цель моделирования – подобрать такие начальные параметры, чтобы система как можно дольше оставалась в равновесии.

1.2 Техническое задание на курсовое проектирование

1.2.1 Назначение разработки

Данная система предназначена для моделирования процессов, проходящих в реальных экологических системах. С помощью разработанного программного обеспечения можно будет анализировать степень влияния того или иного параметра на развитие системы.

1.2.2 Область применения

Данная разработка может использоваться экологами и биологами при изучении проблем экологического равновесия замкнутых систем.

1.2.3 Требования к функциональным характеристикам

1. Пользователь должен иметь возможность изменять начальное количество объектов системы.

2. Пользователь может изменять основные характеристики объектов системы, такие как время жизни, период размножения, время без еды.

3. Результаты должны быть максимально наглядными.

4. После гибели последнего объекта системы, она должна останавливаться.

5. Должна быть наглядная информация о количестве «прожитых» тактов.

1.2.4Требования к количественным характеристикам

1. Система должна пересчитывать свое состояние не реже чем 1 раз в секунду.
2. Количество точек в системе должно быть не менее 2000.
3. Количество рабочих мест – 1.

1.2.5 Требования к техническим средствам

1. Процессор – Intel Pentium IV или Intel Core Duo/
2. Оперативная память – 512 Мб и выше.
3. Жесткий диск – 5 Мб свободного места на диске для программы.
4. Монитор – SVGA, видеоадаптер с поддержкой разрешения 1280x1024
5. Клавиатура, мышь

Задача моделирования очень ресурсоемкая, поэтому к техническим средствам представляются такие жесткие требования.

1.2.6 Требования к информационной и программной совместимости

Программное обеспечение должно функционировать на операционных системах семейства MS Windows 2000/XP/Vista. Функционирование программы под операционной системой MS Windows 98/Me невозможно, так как она не поддерживает технические требования, предъявляемые к проекту.

Программа должна функционировать независимо от наличия в системе установленных средств разработки.

1.2.7 Требования к интерфейсу пользователя

Программа должна иметь графический интуитивно понятный Windows-совместимый пользовательский интерфейс. Отображение промежуточных и итоговых результатов моделирования должно быть максимально наглядно.

1.3 Анализ технического задания

Проектируемая система состоит из двух независимых частей – части, отвечающей за моделирование, и части, отвечающей за вывод информации на экран.

Решить задачу можно несколькими способами:

1. Расчетная часть реализуется в виде динамически подключаемой библиотеки DLL на C++. А графическая часть выполняется в виде exe-файла, к которому подключается разработанная DLL.

Достоинством такого подхода является быстрота написания программного кода, простота отладки графической части. Среди недостатков можно отметить сложность отладки расчетной части и возможные проблемы при подключении библиотеки.

2. И расчетная, и графическая части выполняются в виде одного exe-файла на C++. Достоинством такого подхода является простота сборки проекта. Недостатки – сложность написания графической части на C++.

Для решения поставленной задачи мы выберем второй способ, потому что во-первых, проще отладить расчетную часть, а во-вторых, не возникнет проблем со сборкой проекта.

1.4 Выбор методов и средств решения задачи

В связи с тем, что сегодня уровень сложности программного обеспечения очень высок, разработка приложений MS Windows с использованием только какого-либо языка программирования (например, языка C++) значительно затрудняется. Программист должен затратить массу времени на решение стандартных задач по созданию многооконного интерфейса. Реализация технологии связывания и встраивания объектов - OLE - потребует от программиста еще более сложной работы.

Чтобы облегчить работу программиста, практически все современные компиляторы с языка C++ содержат специальные библиотеки классов. Такие библиотеки включают в себя практически весь программный интерфейс MS Windows и позволяют пользоваться при программировании средствами более высокого уровня, чем обычные вызовы функций. За счет этого значительно упрощается разработка приложений, имеющих сложный интерфейс пользователя, облегчается поддержка технологии OLE и взаимодействие с базами данных.

Современные интегрированные средства разработки приложений MS Windows позволяют автоматизировать процесс создания приложения. Для этого используются генераторы приложений. Программист отвечает на вопросы генератора приложений и определяет свойства приложения - поддерживает ли оно многооконный режим, технологию OLE, трехмерные органы управления, справочную систему. Генератор приложений, создаст приложение, отвечающее требованиям, и предоставит исходные тексты. Пользуясь им как шаблоном, программист сможет быстро разрабатывать свои приложения.

В настоящее время существует несколько сред разработки на языке C++, наиболее популярны для разработки приложений под MS Windows две – Borland Builder C++ и MS Visual C++.

Интегрированная среда Borland Builder C++ обеспечивает скорость визуальной разработки, продуктивность повторно используемых компонент в сочетании с мощью языковых средств C++, усовершенствованными инструментами и разномасштабными средствами доступа к базам данных.

Borland Builder C++ может быть использован везде, где требуется дополнить существующие приложения расширенным стандартом языка C++, повысить быстродействие и придать пользовательскому интерфейсу качества профессионального уровня.

Студия разработчика фирмы Microsoft (Microsoft Developer Studio) - это интегрированная среда для разработки, позволяющая функционировать различным средам разработки, одна из которых Visual C++.

В студии разработчика можно строить обычные программы на C и C++, создавать статические и динамические библиотеки, но основным режимом ра-

боты является создание Windows-приложений с помощью инструмента MFC AppWizard (Application Wizard - мастер приложений) и библиотеки базовых классов MFC (Microsoft Foundation Class Library). Такие приложения называются MFC-приложениями. Главная особенность этих Windows-приложений состоит в том, что они работают как совокупность взаимодействующих объектов, классы которых определены библиотекой MFC.

MFC AppWizard реализует средства автоматизированного создания приложений. Заполнив несколько диалоговых панелей, можно указать характеристики приложения и получить его тексты, снабженные обширными комментариями. MFC AppWizard позволяет создавать однооконные и многооконные приложения, а также приложения, не имеющие главного окна, - вместо него используется диалоговая панель. Можно также включить поддержку технологии OLE, баз данных, справочной системы.

Поэтому для разработки своей программы мы выберем среду разработки MS Visual C++ 2005.

2 Разработка программного обеспечения

2.1 Разработка алгоритмов

2.1.1 Общий алгоритм работы системы

Все объекты системы живут в замкнутом пространстве – в лесу. Соответственно жизнь сразу после старта и заканчивается в двух случаях:

1) когда все объекты леса умерли;
2) когда пользователь выходит из системы.

После старта начинается «жизнь» леса. Изначально есть некоторое количество травы, зайцев и волков. Появляются они в случайном месте.

Трава сразу начинает расти и размножаться. Через некоторое время трава умирает.

Зайцы сначала должны вырасти до определенного возраста, при этом они должны питаться травой. Они сразу же могут двигаться, если в течение некоторого времени они не поедят, то они погибнут. Если заяц достиг возраста размножения, то в лесу появится еще один заяц. Новый заяц появляется на одной и соседних точек леса. Размножаются зайцы с некоторой периодичностью. Достигнув возраста смерти, заяц умирает.

Алгоритм жизни волков такой же, как и у зайцев.

Все объекты отображаются на экране в виде картинок. Положение картинки означает положение объекта в лесу. Картинка обновляется после каждого такта жизни леса. После расчета одного такта устанавливается небольшая задержка, чтобы картинка на экране не мелькала слишком часто.

2.1.2 Алгоритм поиска травы зайцем

Заяц может передвигаться на определенное расстояние. Также он способен видеть на определенное количество точек леса. Сначала заяц начинает поиск травы в соседних клетках. Если травы там нет, то он сканирует пространство вокруг себя на большее расстояние. Если он обнаруживает траву, то он переходит на точку, где в данный момент находится трава. Если трава находится вне зоны одного прыжка зайца, то заяц начинает движение в сторону найденной травы. В следующий раз заяц опять начнет с поиска травы, потому что за один такт может появиться трава в другом месте, которое будет ближе к текущему положению зайца. Может так случиться, что заяц травы не найдет. Тогда он случайным образом выбирает направление своего движения, а также длину прыжка. При этом точка, куда прыгает заяц, должна быть свободна и от других зайцев, и от волков. Если так случилось, что вокруг нет совсем свободного места (когда популяция зайцев достигает критического максимума), тогда у зайца наступает состояние «паники» и он не двигается.

2.1.3 Алгоритм размножения волка

Волк размножается, достигнув возраста размножения. Новый волк появляется в свободной соседней точке пространства. Если такой точки нет, но везде вокруг волка трава, то новый волк появится там, где до этого была трава. Трава при этом погибнет. Если вокруг волка только зайцы, то новый волк появится в точке, где до этого находился заяц. Заяц, так же, как и трава, погибнет.

2.1.4 Алгоритм поиска зайца волком

Алгоритм питания волка отличается от алгоритма питания зайца прежде всего тем, что заяц питается неподвижной травой, а волк кушает передвигающихся зайцев.

Волк начинает искать зайца в соседних клетках на расстоянии прыжка. Если заяц обнаружен, то он будет съеден. Если заяц находится в зоне видимости, но вне досягаемости, то волк начинает движение в сторону зайца. На следующем такте волк опять анализирует положение зайца, потому что заяц может упрыгать в поисках травы. Тогда волк опять начинает сканирование пространства в поисках зайца.

Если в пределах видимости нет зайцев, то волк случайным образом выбирает направление движения и величину прыжка. Если в точке, куда прыгнул волк, была трава, то после того, как волк убежит с этой точки, там снова появится трава.

Если волк не может никуда прыгнуть (когда популяция волков достигнет критического максимума), тогда волк просто стоит на месте.

2.2 Разработка классов

В моделировании участвуют объекты трех разных видов – трава, зайцы и волки. У них совершенно разные алгоритмы поведения, но есть у них общие

качества, такие как положение в пространстве, возраст, время размножения, время смерти. Также все они живут, размножаются, а волк с зайцем еще двигаются и питаются.

В результате проведенной объектной декомпозиции задачи была получена иерархия пользовательских классов, показанная на рис. П1

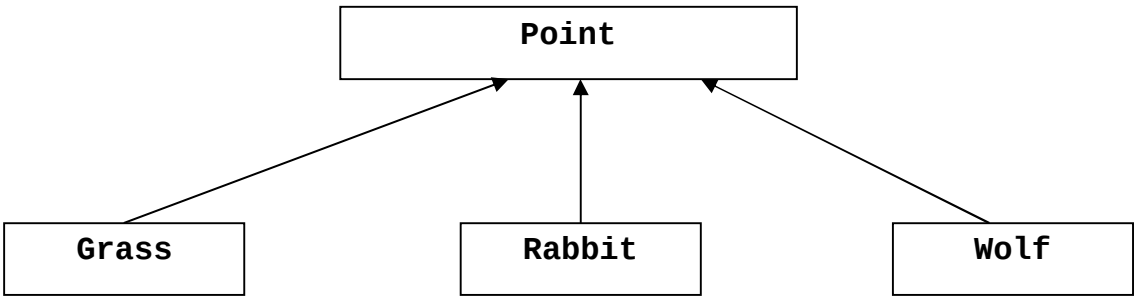

Рисунок П1 – Иерархия классов системы

2.2.1 Класс Point

Этот класс является абстрактным базовым классом иерархии. Он содержит общие для всех классов свойства и методы.

К атрибутам класса `Point` относятся:

```
struct points
    {
        int x,y;
    }p;
```
- структура Точка, в ней хранятся координаты объекта в лесу

`int Age;` - возраст объекта

`int MaxAge;` - возраст смерти объекта

`int ChildTime;` - время размножения

К методам класса относятся функции доступа к атрибутам класса.

Также в классе определена функция `Live`. В классе `Point` она не может быть реализована, так как каждый конкретный объект живет по своему собственному алгоритму, поэтому эта функция является чистой виртуальной.

2.2.2 Класс Grass

Этот класс не имеет никаких отличных от класса `Point` атрибутов, но в этом классе переопределена функция `Live`, потому что трава уже имеет свой алгоритм жизни.

```
int Grass::Live(Point** p)
{
    //время умирать**************
    if(GrowUp()==-1) return -1;
    //***************************
```

```
        //новая трава***************
        if(!(Age%ChildTime))
        {
            rand();
            int k=0;
            do
            {
                int i,j;
                i=rand()%3-1;
                j=rand()%3-1;
                if((Point::p.x+i<0)||
                    (Point::p.x+i>=ROW)||
                    (Point::p.y+j<0)||
                    (Point::p.y+j>=COL))
                {
                    k--;
                    continue;
                }
                if(!p[CoordInNum(Point::p.x+i,
                    Point::p.y+j)])
                {
                   p[CoordInNum(Point::p.x+i,
                   Point::p.y+j)]          =          new
                   Grass(Point::p.x+i,     Point::p.y+j,
                   MaxAge, ChildTime);
                    return 1;
                }
            }
            while(++k<8);
            return 0;
        }
        //**************************
    }
```

Трава растет, достигает возраста размножения, порождает новую траву через заданный промежуток времени и, достигнув времени смерти, трава умирает.

2.2.3 Класс Rabbit

У класса `Rabbit` появляются новые атрибуты – это максимальное время, которое заяц может провести без пищи, а также счетчик голодного времени зайца.

Функция `Live` в этом классе также переопределена. В ней реализован алгоритм, описанный в пункте 2.1.2, и алгоритм размножения зайцев.

Алгоритм поиска травы зайцем:

```
//надо покушать**************
    int k=0;
    do
    {
        int i,j;
        i=rand()%5-2;
        j=rand()%5-2;

        if((Point::p.x+i<0)||(Point::p.x+i>=ROW)||(P
      oint::p.y+j<0)|| (Point::p.y+j>=COL))
        {
            k--;
            continue;
        }
        if(p[CoordInNum(Point::p.x+i,  Point::p.y+j)]
     &&  (p[CoordInNum(Point::p.x+i,  Point::p.y+j)]-
     >Type()==1))
        {
            p[CoordInNum(Point::p.x,Point::p.y)]= 0;
            Point::p.x +=i;
            Point::p.y +=j;
            IsEat=0;
        delete p[CoordInNum(Point::p.x,Point::p.y)];
        p[CoordInNum(Point::p.x,Point::p.y)]=this;
            return 1;
        }
    }
    while(++k<48);
    IsEat++;
    int i,j;
    do
    {
        i=rand()%5-2;
        j=rand()%5-2;
    }
    while((Point::p.x+i<0)||(Point::p.x+i>=ROW)||(Po
    int::p.y+j<0)|| (Point::p.y+j>=COL));
    //*********************
```

Размножение зайцев реализовано следующим образом:

```
//новый кролик***************
    if(!(Age%ChildTime))
    {
```

```
            rand();
            int k=0;
            do
            {
                int i,j;
                i=rand()%3-1;
                j=rand()%3-1;

        if((Point::p.x+i<0)||(Point::p.x+i>=ROW)||(Point
       ::p.y+j<0)|| (Point::p.y+j>=COL))
                {
                    k--;
                    continue;
                }
        if(!p[CoordInNum(Point::p.x+i,Point::p.y+j)])
                {
                    p[CoordInNum(Point::p.x+i,
        Point::p.y+j)]   =   new   Rabbit(Point::p.x+i,
        Point::p.y+j, MaxAge, ChildTime);
                    return 1;
                }
    if(p[CoordInNum(Point::p.x+i,Point::p.y+j)]->Type()
         ==1)
                {
    delete p[CoordInNum(Point::p.x+i, Point::p.y+j)];
                    p[CoordInNum(Point::p.x+i,
        Point::p.y+j)]   =   new   Rabbit(Point::p.x+i,
        Point::p.y+j, MaxAge, ChildTime);
                    return 1;
                }
            }
            while(++k<8);
            return 0;
        }
        //**************************
```

2.2.4 Класс Wolf

У класса `Wolf` также есть новые атрибуты, они такие же, как у класса `Rabbit`.

Функция `Live` переопределена, в ней реализованы алгоритмы, описанные в пп. 2.1.3 и 2.1.4. Реализация этой функции очень похожа на реализацию функции `Live` в классе `Rabbit`, поэтому здесь ее приводить не будем.

2.2.5 Класс Forest

Все объекты вышеперечисленных классов живут не сами по себе, а в общем замкнутом пространстве, поэтому необходимо иметь класс-коллекцию объектов. В роли такого класса выступает класс Forest. В нем содержится массив указателей на все объекты леса. В этом классе реализована функция жизни леса, в которой с определенной периодичностью вызываются функции жизни для каждого существующего объекта в лесу.

```
void forest::forest_live()
{
    //задаем начальные значения и выводим на экран
    start();
    //*****************************************
    //а теперь собственно плодимся и размножаемся
    for(int i=0;i<COUNTER; i++)
        {
            if(frst[i])
            {

                int s = frst[i]->Live(frst);
                if(s==-1)
                {
                    delete frst[i];
                    frst[i] = 0;
                }
            }
        }
    //*****************************************
}
```

Функция `start` населяет лес жителями, т.е. после ее работы создается нужное количество объектов разных типов в случайных точках леса.

2.3 Программная реализация

2.3.1 Структура программы

Каждый класс программы реализован в своем модуле, файлы интерфейса были созданы с помощью MFC AppWizard.

Расчетная часть программы состоит из следующих модулей:

1. `Point` – описание и определение функций абстрактного базового класса точки.
2. `Grass` – описание и определение функций класса травы
3. `Rabbit` – описание и определение функций класса зайцев
4. `Wolf` – описание и определение функций класса волков

5. `Forest` – описание и определение функций класса-коллекции леса.

Графическая часть программы реализована в файлах, созданных мастером приложений. В эти файлы вносились изменения, связанные с отображением объектов леса на экране, созданием и установкой таймера обновления картинки, а также изменялась панель состояния, на которую выводится информация о текущем состоянии леса – количество прожитых тактов и количество обитателей.

Кроме этого в программу были добавлены картинки, схематично изображающие обитателей леса.

Текст всех файлов программы приведен в приложении А.

2.3.2 Соглашения по именованию переменных

В ходе работы над курсовым проектом были выработаны следующие соглашения по именованию переменных и написанию кода.

1. Все переменные в программе имеют осмысленное название.
2. Имена всех пользовательских классов начинаются с заглавной буквы, имена функций классов также начинаются с заглавной буквы.
3. Имена всех глобальных констант прописаны заглавными буквами
4. Имена объектов классов и переменных, связанных с пользовательскими классами, начинаются с буквы, совпадающей с начальной буквой в имени класса.
5. Текст программы снабжен комментариями, важные блоки функций отделены друг от друга с помощью комментариев.
6. Текст программы структурирован, вложенные операторы смещены вправо относительно заголовков.

2.3.3 Входные и выходные данные

Входными данными программы являются:

1) количество травы;
2) количество зайцев;
3) количество волков;
4) время жизни травы;
5) время жизни зайцев;
6) время жизни волков;
7) возраст размножения травы;
8) возраст размножения зайцев;
9) возраст размножения волков.

Выходные данные – это состояние системы на каждом такте жизни леса. В качестве выходных данных можно рассматривать и количество тактов, которое система прожила. Чем дольше тактов проживает система, тем лучше.

3 Расчеты и оценки

Программа тестировалась на системе следующей конфигурации:

Процессор – Intel Core2 Duo 6320

Оперативная память – 2Гб

Видеоадаптер – NVIDIA GeForce 7600 GT

Операционная система – MS Windows XP SP2

На тестовой конфигурации не было никаких задержек, обновление состояния на экране происходило строго по установленному таймеру. Максимально возможное количество объектов в лесу – 7000.

Как показали эксперименты, из всех параметров на длительность жизни системы оказывает самое большое влияние количество объектов в лесу. При количестве 2000 самое длительное время жизни системы составило 132 такта, а минимальное – 37.

При количестве объектов 4000 это время увеличилось до 188 тактов, а при количестве 7000 – до 267.

Так же важным параметром является время голодания зайцев и волков.

И соотношение времени голодания и времени размножения. Если новый объект может появиться до того, как текущий объект первый раз должен поесть, то система начинает жить как минимум в два раза дольше, чем при обратном соотношении. Параметры и показатели жизни системы сведены в табл. П1.

Таблица П1

Показатели устойчивости системы

Количество объектов	2000	2000	7000
Количество травы	100	100	1000
Количество зайцев	50	50	150
Количество волков	20	20	60
Время жизни травы	3	3	5
Время жизни зайцев	6	6	10
Время жизни волков	15	15	13
Возраст размножения травы	1	1	1
Возраст размножения зайцев	3	2	3
Возраст размножения волков	5	7	6
Минимальное время жизни системы	28	38	56
Максимальное время жизни системы	132	129	267

Возможно несколько окончаний жизни системы.

1. Волки съедают всех зайцев, потом умирают сами, после чего остается в лесу одна трава. В этом состоянии система может находиться бесконечно долго.

2. Умирают все волки. После этого возможно два сценария развития системы. Зайцы поедают всю траву, затем сами умирают. Это смерть системы. Во

втором сценарии зайцы все умирают (например, они находились слишком далеко от травы), после чего остается одна трава.

3. Зайцы съедают всю траву. После этого в зависимости от соотношений времени жизни и местоположения умирают сначала все зайцы, потом все волки, или наоборот. Система умирает.

Если в результате выживает только трава, это так же считаем гибелью системы, потому что два вида объектов из трех перестали существовать.

4 Руководство пользователя

Для установки программы достаточно скопировать файл программы в любую доступную директорию компьютера.

4.1 Описание интерфейса

Программа представляет собой однооконное приложение, в окно которого выводится состояние системы (рис. П2). Вверху окна есть панель состояния, где отображается цифровая информация о состоянии системы в режиме реального времени.

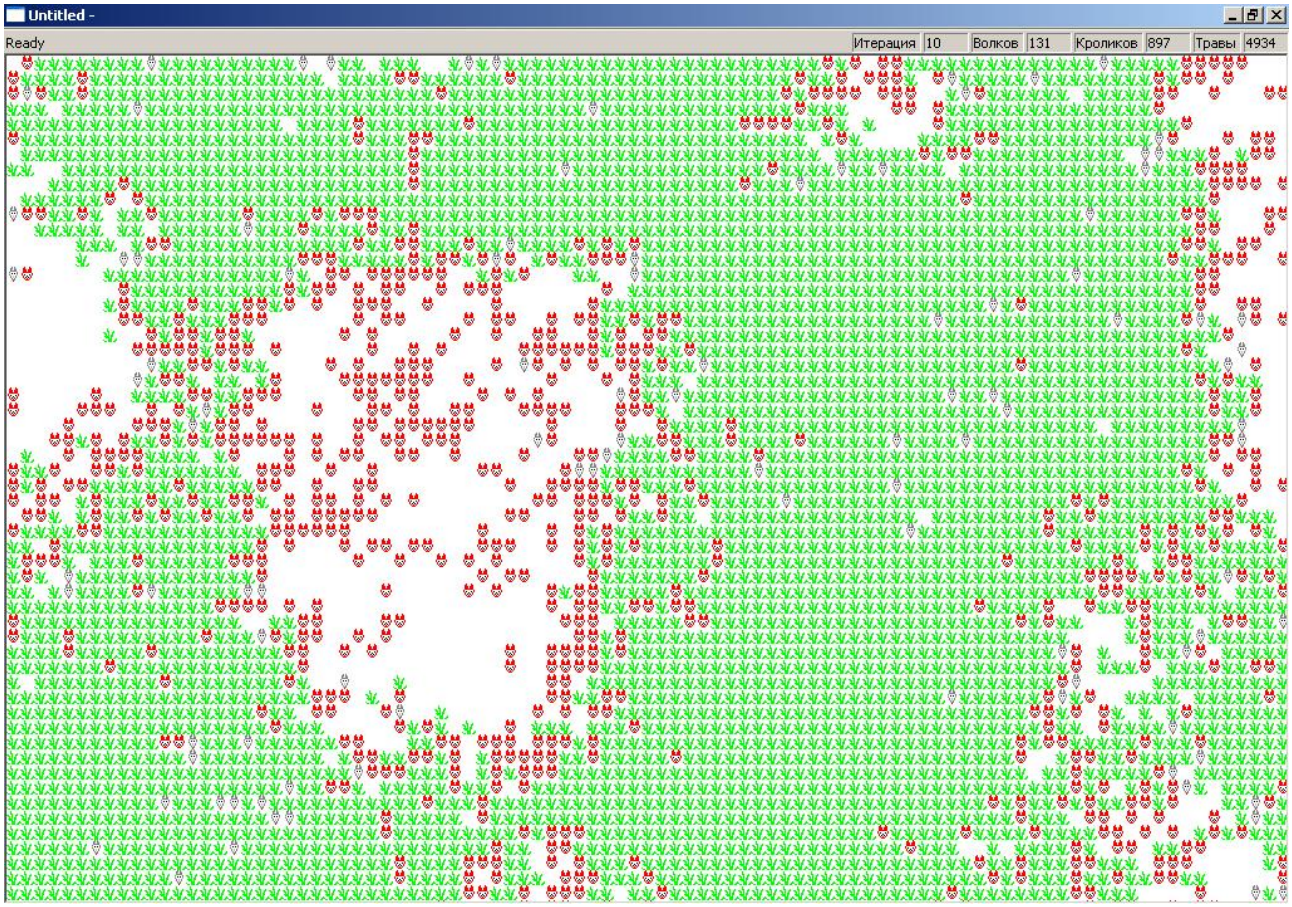

Рисунок П2 – Основное окно программы

Все объекты системы отображаются в виде значков. Зеленый значок – трава, значок с красными ушками – зайцы, значок с серыми ушами – волки.

Разноцветность значков позволяет легко отслеживать тип представленных объектов.

Количество объектов в системе зависит от версии системы. В представленной версии количество объектов составляет 7000. Для их корректного отображения разрешение экрана должно быть не ниже 1280x1024 точек. Если разрешение экрана меньше, то будет невозможно отобразить все объекты на экране. Это не страшно, но пользователь не сможет видеть полную картину, хотя по строке состояния он сможет правильно оценить ситуацию.

4.2 Работа с системой

Работа с системой начинается с запуска исполняемого файла на выполнение.

После этого возникает диалоговое окно, в котором пользователь может задать начальные параметры системы (см. п. 2.3.3).

После этого открывается главная форма системы (рис. П2), где с периодичностью в полсекунды происходит обновление картинки. Картинка показывает лес, в котором живут волки, зайцы и трава. Точно оценивать складывающуюся ситуацию помогает панель состояния. В данной программе панель состояния находится вверху, а не внизу окна как обычно бывает. Это помогает отслеживать количество объектов в лесу, а также номер текущей итерации. Когда система умирает, обновление ее состояния прекращается, и счетчик итераций останавливается. Это позволяет оценить качество модели.

Заключение

В ходе курсового проектирования была разработана программа, моделирующая функционирование экологически замкнутой системы. В качестве объекта моделирования был выбран лес и лесные обитатели – трава, зайцы и волки.

К достоинствам системы можно отнести наглядность результатов, красочность картинки, простоту работы с программой.

К недостаткам – малое количество параметров модели, а также ограниченное количество объектов в модели.

Данный проект может дальше развиваться в сторону усложнения поведения существующих объектов, а также внедрения в систему новых объектов леса.

Задача, поставленная на курсовое проектирование, выполнена в полном объеме. Проект полностью соответствует техническому заданию.

Приложение А[*]

Листинг программы

```
//Point.h*****************************************
#pragma once

#define ROW 70
#define COL 100

class Point
{
protected:
     struct points
     {
          int x,y;
     }p;
     int Age;
     int MaxAge;
     int ChildTime;
     int GetPos();
     void GetPos(int &x1, int &y1);
     void SetPos(int x1, int y1);
     void SetPos(int num_p);
     void NumInCoord(int num, int &x1, int &y1);
     int CoordInNum(int x1, int y1);
     int GrowUp(void);
public:
     class PointException {};
     class Overflow:public PointException
     {
     public:
          int index;
          Overflow(int i):index(i){}
     };
     Point(void);
     Point(int x1, int y1);
     Point(int num_p);
     virtual ~Point(void);
     virtual int Live(Point** p)=0;
     virtual int Type()=0;

};
//*****************************************************

//Point.cpp***************************************************
#include "Point.h"
```

[*] Автор комплекса приводит только приложение А

```
Point::Point(void)
{
     p.x=p.y=Age=0;
}

Point::~Point(void)
{
}
Point::Point(int x1, int y1)
{
     SetPos(x1, y1);
     Age=0;
}

Point::Point(int num_p)
{
     SetPos(num_p);
     Age=0;
}

int Point::GetPos()
{
     return p.x*ROW+p.y;
}

void Point::GetPos(int &x1, int &y1)
{
     x1 = p.x;
     y1 = p.y;
}
void Point::SetPos(int num_p)
{
     if(ROW*COL<num_p) throw Overflow(num_p);
     p.x=num_p/COL;
     p.y=num_p%COL;
}

void Point::SetPos(int x1, int y1)
{
     if(x1>ROW) throw Overflow(x1);
     p.x=x1;
     if(y1>COL) throw Overflow(y1);
     p.y=y1;
}


int Point::GrowUp()
{
     if(Age==MaxAge) return -1;
     return ++Age;
}
```

```
void Point::NumInCoord(int num, int &x1, int &y1)
{
     x1 = num/COL;
     y1 = num%COL;
}
int Point::CoordInNum(int x1, int y1)
{
     return x1*COL + y1;
}
//*********************************************************


//Grass.h*****************************************************
#pragma once
#include "point.h"

class Grass : public Point
{
private:
     void SetGrass(int max, int ch);
public:
     Grass(void);
     Grass(int x, int y, int max, int ch);
     Grass(int pos, int max, int ch);
     virtual ~Grass(void);
     virtual int Type();
     virtual int Live(Point** p);
};
//*********************************************************

//Grass.cpp***************************************************
#include "Grass.h"
#include <stdlib.h>
#include <time.h>

Grass::Grass(void)
{
     SetGrass(3,1);
}

Grass::~Grass(void)
{}
Grass::Grass(int x, int y, int max, int ch):Point(x,y)
{
     SetGrass(max,ch);
}

Grass::Grass(int pos, int max, int ch):Point(pos)
{
     SetGrass(max,ch);
}
```

```
void Grass::SetGrass(int max, int ch)
{
     MaxAge = max;
     ChildTime = ch;
}

int Grass::Live(Point** p)
{
     //время умирать**************
     if(GrowUp()==-1) return -1;
     //***************************
     //новая трава****************
     if(!(Age%ChildTime))
     {
          rand();
          int k=0;
          do
          {
               int i,j;
               i=rand()%3-1;
               j=rand()%3-1;
     if((Point::p.x+i<0)||(Point::p.x+i>=ROW)||(Point::p.y+j<0)||
(Point::p.y+j>=COL))
               {
                    k--;
                    continue;
               }
               if(!p[CoordInNum(Point::p.x+i, Point::p.y+j)])
               {
                    p[CoordInNum(Point::p.x+i,  Point::p.y+j)]
= new Grass(Point::p.x+i, Point::p.y+j, MaxAge, ChildTime);
                    return 1;
               }
          }
          while(++k<8);
          return 0;
     }
     //***************************
}
int Grass::Type()
{
     return 1;
}
//************************************************

//Rabbit.h*****************************************
#pragma once
#include "point.h"

class Rabbit:public Point
{
     int EatTime;
```

```
     int IsEat;
     int GrowUp();
     void SetRabbit(int max, int ch);
public:
     Rabbit(void);
     virtual ~Rabbit(void);
     Rabbit(int x, int y, int max, int ch);
     Rabbit(int pos, int max, int ch);
     virtual int Type();
     virtual int Live(Point** p);
};
//***********************************************

//Rabbit.cpp*****************************************
#include "Rabbit.h"
#include <stdlib.h>

Rabbit::Rabbit(void)
{
     SetRabbit(6, 3);
}

Rabbit::~Rabbit(void)
{
}
Rabbit::Rabbit(int x, int y, int max, int ch):Point(x,y)
{
     SetRabbit(max, ch);
}
Rabbit::Rabbit(int pos, int max, int ch):Point(pos)
{
     SetRabbit(max, ch);
}
void Rabbit::SetRabbit(int max, int ch)
{
     MaxAge = max;
     ChildTime = ch;
     EatTime = ChildTime-1;
     if(EatTime<1)
          EatTime = 1;
     IsEat = 0;
}
int Rabbit::GrowUp()
{
     if(Point::GrowUp()==-1)
          return -1;
     if(IsEat==EatTime)
          return -1;
     return Age;
}
int Rabbit::Type()
{    return 2;}
```

```
     int Rabbit::Live(Point** p)
     {
          //время умирать**************
          if(GrowUp()==-1) return -1;
          //***************************
          //новый кролик****************
          if(!(Age%ChildTime))
          {
               rand();
               int k=0;
               do
               {
                    int i,j;
                    i=rand()%3-1;
                    j=rand()%3-1;

     if((Point::p.x+i<0)||(Point::p.x+i>=ROW)||(Point::p.y+j<0)||
(Point::p.y+j>=COL))
                    {
                         k--;
                         continue;
                    }
                    if(!p[CoordInNum(Point::p.x+i, Point::p.y+j)])
                    {
                         p[CoordInNum(Point::p.x+i,  Point::p.y+j)]
= new Rabbit(Point::p.x+i, Point::p.y+j, MaxAge, ChildTime);
                         return 1;
                    }
               if(p[CoordInNum(Point::p.x+i, Point::p.y+j)]->Type()
==1)
                    {
                         delete
               p[CoordInNum(Point::p.x+i, Point::p.y+j)];
                         p[CoordInNum(Point::p.x+i,  Point::p.y+j)]
= new Rabbit(Point::p.x+i, Point::p.y+j, MaxAge, ChildTime);
                         return 1;
                    }
               }
               while(++k<8);
               return 0;
          }
          //***************************
          //надо покушать**************
          int k=0;
          do
          {
               int i,j;
               i=rand()%5-2;
               j=rand()%5-2;

     if((Point::p.x+i<0)||(Point::p.x+i>=ROW)||(Point::p.y+j<0)||(P
oint::p.y+j>=COL))
```

```
                {
                     k--;
                     continue;
                }
                if(p[CoordInNum(Point::p.x+i,
Point::p.y+j)]&&(p[CoordInNum(Point::p.x+i,  Point::p.y+j)]->Type()
==1))
                {
                     p[CoordInNum(Point::p.x,Point::p.y)] = 0;
                     Point::p.x +=i;
                     Point::p.y +=j;
                     IsEat=0;
                     delete p[CoordInNum(Point::p.x,Point::p.y)];
                     p[CoordInNum(Point::p.x,Point::p.y)] = this;
                     return 1;
                }
           }
           while(++k<48);
           IsEat++;
           int i,j;
           do
           {
                i=rand()%5-2;
                j=rand()%5-2;
           }
           while((Point::p.x+i<0)||(Point::p.x+i>=ROW)||
     (Point::p.y+j<0)||(Point::p.y+j>=COL));
           //*********************
           if(p[CoordInNum(Point::p.x+i,
Point::p.y+j)]&&(p[CoordInNum(Point::p.x+i,  Point::p.y+j)]->Type()
==1))
                {
                     p[CoordInNum(Point::p.x,Point::p.y)] = 0;
                     Point::p.x +=i;
                     Point::p.y +=j;
                     IsEat=0;
                     delete p[CoordInNum(Point::p.x,Point::p.y)];
                     p[CoordInNum(Point::p.x,Point::p.y)] = this;
                     return 1;
                }
           //************************
           if(!p[CoordInNum(Point::p.x+i, Point::p.y+j)])
           {
                p[CoordInNum(Point::p.x,Point::p.y)] = 0;
                Point::p.x +=i;
                Point::p.y +=j;
                p[CoordInNum(Point::p.x,Point::p.y)] = this;
                return 1;
           }
           return 0;
           //***************************
     }//*************************************************
```

```
//Wolf.h*********************************************
#pragma once
#include "point.h"

class wolf:public Point
{
    int EatTime;
     int IsEat;
     int GrowUp();
     void SetWolf(int max, int ch);
public:
    wolf(void);
    virtual ~wolf(void);
    wolf(int x, int y, int max, int ch);
     wolf(int pos, int max, int ch);
     virtual int Type();
     virtual int Live(Point** p);
};
//*********************************************************

//Wolf.cpp*******************************************
#include "wolf.h"
#include <stdlib.h>

wolf::wolf(void)
{
    SetWolf(15, 5);
}

wolf::~wolf(void)
{}

wolf::wolf(int x, int y, int max, int ch):Point(x,y)
{
     SetWolf(max, ch);
}

wolf::wolf(int pos, int max, int ch):Point(pos)
{
     SetWolf(max, ch);
}

void wolf::SetWolf(int max, int ch)
{
     MaxAge = max;
     ChildTime = ch;
     EatTime = ChildTime-1;
     if(EatTime<1)
          EatTime = 1;
     IsEat = 0;
}
```

```
int wolf::GrowUp()
{
     if(Point::GrowUp()==-1)
          return -1;
     if(IsEat==EatTime)
          return -1;
     return Age;
}
int wolf::Type()
{
     return 3;
}
int wolf::Live(Point** p)
{
     //время умирать**************
     if(GrowUp()==-1) return -1;
     //***************************
     //новый волк****************
     if(!(Age%ChildTime))
     {
          rand();
          int k=0;
          do
          {
               int i,j;
               i=rand()%3-1;
               j=rand()%3-1;

     if((Point::p.x+i<0)||(Point::p.x+i>=ROW)||(Point::p.y+j<0)||
(Point::p.y+j>=COL))
               {
                    k--;
                    continue;
               }
               if(!p[CoordInNum(Point::p.x+i, Point::p.y+j)])
               {
                    p[CoordInNum(Point::p.x+i,  Point::p.y+j)]
= new wolf(Point::p.x+i, Point::p.y+j, MaxAge, ChildTime);
                    return 1;
               }
          if(p[CoordInNum(Point::p.x+i, Point::p.y+j)]->Type()
==1)
               {
                    delete          p[CoordInNum(Point::p.x+i,
Point::p.y+j)];
                    p[CoordInNum(Point::p.x+i,  Point::p.y+j)]
= new wolf(Point::p.x+i, Point::p.y+j, MaxAge, ChildTime);
                    return 1;
               }
          }
          while(++k<8);
          return 0; }
```

```
          //**************************
          //надо покушать**************
          int k=0;
          do
          {
               int i,j;
               i=rand()%5-2;
               j=rand()%5-2;

     if((Point::p.x+i<0)||(Point::p.x+i>=ROW)||(Point::p.y+j<0)||
(Point::p.y+j>=COL))
               {
                    k--;
                    continue;
               }
               if(p[CoordInNum(Point::p.x+i,
Point::p.y+j)]&&(p[CoordInNum(Point::p.x+i,  Point::p.y+j)]->Type()
==2))
               {
                    p[CoordInNum(Point::p.x,Point::p.y)] = 0;
                    Point::p.x +=i;
                    Point::p.y +=j;
                    IsEat=0;
                    delete p[CoordInNum(Point::p.x,Point::p.y)];
                    p[CoordInNum(Point::p.x,Point::p.y)] = this;
                    return 1;
               }
          }
          while(++k<96);
          //******************************************
         //просто побежать куда-нибудь****************
         IsEat++;
         int i,j;
          do
          {
               i=rand()%7-3;
               j=rand()%7-3;
          }
          while((Point::p.x+i<0)||(Point::p.x+i>=ROW)||
(Point::p.y+j<0) ||(Point::p.y+j>=COL));
          if(p[CoordInNum(Point::p.x+i,
Point::p.y+j)]&&(p[CoordInNum(Point::p.x+i,  Point::p.y+j)]->Type()
==2))
          {
               p[CoordInNum(Point::p.x,Point::p.y)] = 0;
               Point::p.x +=i;
               Point::p.y +=j;
               IsEat=0;
               delete p[CoordInNum(Point::p.x,Point::p.y)];
               p[CoordInNum(Point::p.x,Point::p.y)] = this;
               return 1;
          }
```

```
          Point *pp=NULL;
          if(p[CoordInNum(Point::p.x+i,      Point::p.y+j)]      &&
(p[CoordInNum(Point::p.x+i, Point::p.y+j)]->Type()==1))
         {
               pp = p[CoordInNum(Point::p.x+i, Point::p.y+j)];
               p[CoordInNum(Point::p.x+i, Point::p.y+j)] = 0;

          }
          if(!(p[CoordInNum(Point::p.x+i, Point::p.y+j)]))
          {
               if(pp)
                    p[CoordInNum(Point::p.x,Point::p.y)]=pp;
               else
                    p[CoordInNum(Point::p.x,Point::p.y)]=0;
               Point::p.x +=i;
               Point::p.y +=j;
               p[CoordInNum(Point::p.x,Point::p.y)] = this;
               return 1;
          }
          return 0;
          //***************************
     }
     //*************************************************

     //forest.h*********************************************
     #pragma once
     #include "point.h"
     #include "grass.h"
     #include "rabbit.h"
     #include "wolf.h"
     #define COUNTER ROW*COL
     class forest
     {
     private:
          Point *frst[COUNTER];
          int Grass_Max, Grass_Child;
          int Rabbit_Max, Rabbit_Child;
          int Wolf_Max, Wolf_Child;
          int Grass_Col, Rabbit_Col, Wolf_Col;
     public:
          forest(void);
          virtual ~forest(void);
          forest(int  gr,int  gr_max,  int  gr_ch,  int  rab,  int
rab_max, int rab_ch, int wl, int wl_max, int wl_ch);
          void forest_live();
          int getType(int);
          void start();
          void print();

     };
     //*************************************************
```

```
//forest.cpp*********************************************
#include "forest.h"
#include <stdlib.h>
#include <time.h>
#include <windows.h>
#include <iostream>

using namespace std;

forest::forest(void)
{
     for(int i=0; i<COUNTER; i++)
          frst[i]= 0;
     Grass_Max=Grass_Child=0;
     Rabbit_Max= Rabbit_Child=0;
     Wolf_Max=Wolf_Child=0;
     Grass_Col=Rabbit_Col=Wolf_Col=0;
}

forest::~forest(void)
{
     for(int i=0; i<2000; i++)
          if(frst[i])
               delete frst[i];
}

     forest::forest(int  gr,int  gr_max,  int  gr_ch,  int  rab,  int
rab_max, int rab_ch, int wl, int wl_max, int wl_ch)
     {
          for(int i=0; i<COUNTER; i++)
               frst[i]= 0;
          Grass_Max=gr_max;
          Grass_Child=gr_ch;
          Rabbit_Max= rab_max;
          Rabbit_Child=rab_ch;
          Wolf_Max=wl_max;
          Wolf_Child=wl_ch;
          Grass_Col=gr;
          Rabbit_Col=rab;
          Wolf_Col=wl;
     }

     void forest::forest_live()
     {
          //задаем начальные значения и выводим на экран
          start();
          print();
               //*******************************************
          //а теперь собственно плодимся и размножаемся
          for(int i=0;i<COUNTER; i++)
          {
               if(frst[i])
```

```
          {
               int s = frst[i]->Live(frst);
               if(s==-1)
               {
                    delete frst[i];
                    frst[i] = 0;
               }
          }
     }
     //****************************************
}

void forest::start()
{
     Sleep(1000);
     srand(time(0));
     int i=0;
     //травку создаем*******************************
     for(i=0;i<Grass_Col;i++)
     {
          int pos = rand()%7000;
          if(!frst[pos])
               frst[pos]=new Grass(pos,Grass_Max,
               Grass_Child);
          else
               i--;
     }
     //****************************************
     //а теперь волков******************************
     for(int i=0;i<Wolf_Col;i++)
     {
          int pos = rand()%7000;
          if(!frst[pos])
               frst[pos]=new wolf(pos,Wolf_Max, Wolf_Child);
          else
               i--;
     }
    //****************************************
     //кроликов создаем*****************************
     for(int i=0;i<Rabbit_Col;i++)
     {
          int pos = rand()%7000;
          if(!frst[pos])
          frst[pos]=new Rabbit(pos,Rabbit_Max, Rabbit_Child);
          else
               i--;
     }
     //****************************************
}
```

```
void forest::print()
{
     system("cls");
     for(int i=0;i<COUNTER; i++)
     {
          if(!(i%COL))
               cout<<"\n";
          if(!frst[i])
               cout<<" ";
          else if(frst[i]->Type()==1)
               cout<<".";
          else if(frst[i]->Type()==2)
               cout<<(char)2;
        else if(frst[i]->Type()==3)
               cout<<(char)5;
     }
     Sleep(1000);
}
int forest::getType(int i)
{
     if(!(frst[i])) return 0;
     return frst[i]->Type();

}
//*************************************************************
```

ПРИЛОЖЕНИЕ 2

Задание для курсового проектирования

1. Программа для событийного моделирования очереди посетителей банка. Требуется рассчитать среднее время, которое проводит в очереди клиент банка (известны все возможные банковские операции, время обеда, время выполнения одной операции, время обслуживания в разных кассах, количество касс и т.д.)

2. Программа, моделирующая работу покерного автомата (пять карт за один раз).

3. Программа, моделирующая работу игрового автомата (в трех строчках по три картинки).

4. Программа, играющая с пользователем в морской бой. Предусмотреть вариант, при котором компьютер всегда выигрывает.

5. Программа, расставляющая шахматные фигуры таким образом, чтобы ни одна из них не была под боем. Количество и вид фигур задает пользователь. Предусмотреть возможность пользователю самому расставить часть фигур.

6. Программа, читающая с шахматной доски позицию фигур и определяющая ситуацию для белого короля: шах, мат, пат или обычная игровая ситуация.

7. Программа, генерирующая квадрослов. Слова выбираются из словаря, прилагающегося к программе.

8. Программа, решающая японские кроссворды.

9. Программа, реализующая записную книжку. Должен быть класс «Записная книжка», предусматривающий описание бинарного дерева поиска, хранящего информацию о знакомых и родственниках.

10. Программа, отслеживающая поступление больных в приемный покой больницы и график дежурств медперсонала.

11. Программа, моделирующая работу лифта. Лампочки на этажах включаются по прибытии лифта или при появлении на этаже человека.

12. Программа, играющая с пользователем в крестики-нолики на поле бесконечного размера.

13. Программа, генерирующая чайнворд. Слова выбираются из словаря, прилагающегося к программе.

14. Разработать программу в помощь отделу кадров предприятия. Программа должна выдавать информацию о работающих сотрудниках, об имеющихся на предприятии вакансиях.

15. Программа, генерирующая магический квадрат со стороной n.

16. Программа, позволяющая собрать оптимальную конфигурацию компьютера по заданным параметрам из прайс-листа.

17. Программа, определяющую максимальную вложенность циклов в файле «.cpp». Необходимо получить статистику по циклам: какой цикл чаще используется, какие циклы чаще всего оказываются вложенными и т.д.

18. Программа, позволяющая найти выход из лабиринта размером nxn.

19. Программа, реализующая управление поездами (отправление с вокзала, путь следования, прибытие на вокзал).

20. Программа-справочник «Желтые страницы».

21. Программа для учета успеваемости студента в сессию.

22. Программа в помощь агенту по недвижимости.

23. Программа в гейм-клуб по учету игрового времени клиента.

24. Программа в помощь фотографу (хранение и быстрый поиск нужной фотографии)

25. Программа, «разговаривающая» с пользователем на выбранную им тему. Слова программы отображаются на экране.

26. Программа, переводящая SQL-запрос на естественный язык.

27. Программа, переводящая математическое выражение на естественный язык.

28. Программа, изменяющая в программе на C++ все операторы множественного выбора switch на операторы if…else if.

29. Программа, реализующая электронную карту больного.

30. Программа в помощь менеджеру по организации его рабочего времени.

31. Программа, генерирующая предложения. К программе должен прилагаться словарь.