Московский государственный технический университет
имени Н. Э. Баумана
Факультет Информатика и системы управления
Кафедра Компьютерные системы и сети

«УТВЕРЖДАЮ»
Заведующий кафедрой ИУ-6
________ Сюзев В.В.

Г. С. Иванова, Т.Н. Ничушкина

Программирование на языке С++ в среде Visual Studio 2008

Методические указания по выполнению лабораторных работ № 2 - 8
по дисциплине Объектно-ориентированное программирование

Москва 2013

Содержание

Содержание

Лабораторная работа № 2 3
Лабораторная работа № 3 9
Лабораторная работа № 4 17
Лабораторная работа № 5 22
Лабораторная работа № 6 26
Лабораторная работа № 7 32
Лабораторная работа № 8 38

Лабораторная работа № 2

Программирование выражений и ввода-вывода

Цель работы: изучение средств языка С++ для программирования выражений и ввода-вывода

Объем работы: 2 часа

Теоретическая часть

1 Выражения

Обычно последовательно выполняемые операции в программе компонуются в *выражения*. Различают простые и сложные выражения.

Простые выражения представляют собой строковую запись формул, которая может включать литералы, поименованные константы, переменные, обращения к стандартным функциям и знаки операций, такие как «+», «-» и т.д.

Порядок выполнения операций в выражении определяется приоритетами операций и их ассоциативностью (для последовательно записанных одинаковых операций). Для изменения порядка выполнения операций используются круглые скобки. Например:

```
int  a=10, b=3; float ret; ret=a/b;  // результат: ret=3
c=1;   b=c++;         // результат: b=1, c=2
c=1;   sum=++c;       // результат: c=2 , sum=2
c=a<<4;               // эквивалентно c=a*16;
a+=b;                 // эквивалентно a=a+b;
a=b=5;                // эквивалентно b=5; a=b;
a=(b=s/k)+n;          // эквивалентно b=s/k; a=b+n;
c=(a>b)?a:b;          // если a>b, то c=a, иначе c=b
```

Сложные выражения представляют собой последовательность простых, записанных через запятую «,»:

<Выражение1>,<Выражение2>,...<Выражение n>

По таблице приоритетов операций запятая имеет низший ранг, поэтому простые выражения, разделенные запятой, выполняются последовательно слева направо, а в качестве результата выражения берется тип и значение *самого правого выражения***.**

Например:

```
int m=5,z;
z=(m=m*5,m*3);   // результат:  m=25, z=75
```

```
int d,k;
k=(d=4,d*8);        // результат:  d=4, k=32 .
                    c=(a=5, b=a*a); // эквивалентно a=5; b=a*a; c=b;
```

2 Элементарный ввод вывод

Для ввода/вывода данных скалярных типов и строк обычно используют стандартные функции ввода/вывода, описанные в библиотеке stdio. Для того чтобы применять эти функции необходимо, чтобы программе был доступен файл `stdio.h`, содержащий прототипы функций из этой библиотеки. Для этого необходимо подключить этот файл с помощью директивы препроцессора `include`:

```
#include <stdio.h>
```

В библиотеке существуют три вида функций, организующих элементарный ввод с клавиатуры и вывод на экран:

- форматный ввод/вывод – для выполнения операций ввода/вывода над скалярными значениями, символами и строками;
- ввод/вывод строк;
- ввод/вывод символов.

2.1 Форматный ввод /вывод

Ввод чисел, символов и строк с клавиатуры:

int scanf(<Форматная строка>, <Список адресов переменных>); // возвращает

//количество введенных значений или `EOF`(-1)

Вывод чисел, символов и строк на экран:

int printf(<Форматная строка>, <Список выражений>); // возвращает

// количество выведенных байтов

Форматная строка – это строка, которая помимо символов содержит управляющие спецификации вида:

%[-] [<Целое 1>] [.<Целое 2>] <Формат>

где «-» – выравнивание по левой границе,

<Целое 1> – ширина поля вывода;

<Целое 2> – количество цифр дробной части числа;

<Формат > – формат для ввода/вывода значения.

Основные форматы для ввода и вывода:

d – целое десятичное число;

u – целое десятичное число без знака;

o – целое число в восьмеричной системе счисления;

x – целое число в шестнадцатеричной системе счисления (% 4x – без гашения незначащих нулей);

f – вещественное число;

e – вещественное число в экспоненциальной форме;

c – символ;

p – ближний указатель (адрес);

s – символьная строка, вводит строку до первого пробела.

Кроме этого, форматная строка может содержать Esc-последовательности:

\n – переход на следующую строку;

\t – переход на следующую позицию табуляции;

\r – перевод каретки;

\f – перевод страницы;

\n hhh – вставка символа с кодом ANSI hhh (код задается в шестнадцатеричной системе счисления);

%% – печать знака %.

Примеры форматного ввода/вывода:

а) `i=26;`

```
printf ("%-6d∪∪∪%%∪ %o∪ %x\n", i, i, i);
```

Выведенная строка: `26∪∪∪∪∪∪∪%∪32∪1A ↵`

б) `scanf("%d  %d",&a,&b);`

Вводимые значения: 1) `24 28` 2) `24↵`
`28`

в) `scanf("%d,%d",&a,&b);`

Вводимые значения: `24,28`

г) `scanf("%s",name);`

Вводимые значения: `Иванов  Иван`

Результат ввода: `Иванов`

Последний пример демонстрирует, что по формату s строка вводится только до пробела. Чтобы ввести всю строку используют специальную функцию ввода строк или повторяют ввод в другие переменные из той же строки.

Примечание. Начиная с версии Visual C++ 2008 разработчики языка рекомендуют вместо стандартной функции `scanf` использовать функцию `scanf_s`, которая контролирует длину вводимой пользователем строки, позволяя исключить ошибку

переполнения буфера. При вызове функции `scanf_s` максимально возможную длину вводимой строки указывают после соответствующего формата.

Пример.

```
#include <stdio.h>
#include <conio.h>
int main()
{
   int i,result;      float fp;     char  c,s[81];
   result = scanf_s( "%d %f %c %s", &i, &fp, &c, 1, s, 80 );
   printf( "The number of fields input is %d\n", result );
   printf( "The contents are: %d %f %c %s \n", i, fp, c, s);
   _getch();
}
```

2.1 Ввод/вывод строк

Ввод строк с клавиатуры:

`char* gets`(<Строковая переменная>); // возвращает копию строки или NULL

Вывод строк на экран с переходом на следующую строку:

`int puts` (<Строковая константа или переменная>);

Примеры:

а) `puts("Это строка");`

Результат: `Это строка↵`

б) `gets(st);`

Вводимые значения: `Иванов Иван↵`

Результат: `Иванов  Иван`

2.3 Ввод/вывод символов

Ввод символов с клавиатуры:

`int getchar();` // возвращает символ или EOF

Вывод символов на экран:

`int putchar`(<Символьная переменная или константа>);

Например:

`ch=getchar(); putchar('t');`

Пример. Программа определения корней квадратного уравнения $Ax^2+Bx+C=0$ при условии, что дискриминант неотрицателен.

```
#include <locale>
#include <stdio.h>
```

```
#include <conio.h>
#include <math.h>
// основная программа
int main(int argc, char* argv[])
{
    setlocale(0,"russian");
    float A,B,C,E,D,X1,X2;
    puts("Введите A,B,C:");
    scanf_s("%f  %f  %f",&A,&B,&C);
    printf("A=%5.2f  B=%5.2f   C=%5.2f\n",A,B,C);
    E=2*A;     D=sqrt(B*B-4*A*C);
    X1=(-B+D)/E;     X2=(-B-D)/E;
    printf("X1= %7.3f   X2=%7.3f\n",X1,X2);
    puts("Нажмите любую клавишу для завершения...");
    _getch();
    return 0;
}
```

Порядок выполнения работы

1. Прочитать и проанализировать задание в соответствии со своим вариантом.
2. Разработать схему алгоритма решения задачи.
3. Написать программу.
4. Вызвать среду программирования Visual Studio, создать новый проект консольного приложения и ввести текст программы в среду программирования.
5. Подобрать тестовые данные (не менее 3-х вариантов).
6. Отладить программу на выбранных тестовых данных.
7. Продемонстрировать работу программы преподавателю.
8. Составить отчет по лабораторной работе.
9. Защитить лабораторную работу преподавателю.

Требования к отчету

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не

допускается). Схемы также должны быть напечатаны при помощи компьютера или нарисованы с использованием чертежных инструментов, в том числе карандаша.

Каждый отчет должен иметь титульный лист, на котором указывается:

а) наименование факультета и кафедры;

б) название дисциплины;

в) номер и тема лабораторной работы;

г) фамилия преподавателя, ведущего занятия;

д) фамилия, имя и номер группы студента;

е) номер варианта задания.

Кроме того отчет по лабораторной работе должен содержать**:**

1) схему алгоритма, выполненную вручную или в соответствующем пакете;
2) текст программы;
3) результаты тестирования, которые должны быть оформлены в виде таблицы вида:

Исходные данные	Ожидаемый результат	Полученный результат

4) выводы.

Контрольные вопросы

1. Что такое «простые и сложные выражения»? Чем конструкции выражений С++, отличаются от аналогичных конструкций в Паскале?

2. В каких случаях используется символ «;» в С++?

3. Какие процедуры реализуют операции ввода-вывода в С++?

4. Почему в С++ используют разные процедуры ввода-вывода при работе со сроками, числами и символами?

Лабораторная работа № 3

Программирование ветвлений и циклов

Цель работы: изучение средств языка C++ для программирования ветвлений и циклов

Объем работы: 2 часа

Теоретическая часть

1 Оператор условной передачи управления

Оператором условной передачи управления называют конструкцию, позволяющую выбрать одно из возможных продолжений вычислительного процесса в зависимости от результата условия. Синтаксис оператора:

`if` (<Выражение>) <Оператор;> [else <Оператор;>]

Квадратные скобки показывают, что описание ветви `else` – необязательно.

Выражение – любое выражение, записанное по правилам Си или C++. Если результат выражения не равен нулю, то выполняется оператор, следующий за выражением**.** Если результат выражения равен нулю, то выполняется оператор альтернативной ветви. При отсутствии описания альтернативной ветви управление передается следующему за `if` оператору.

Пример. Написать программу решения системы уравнений:

$$\begin{cases} ax=b \\ x+cy=1 \end{cases}$$

Схема алгоритма решения системы уравнений приведена на рисунке 1.

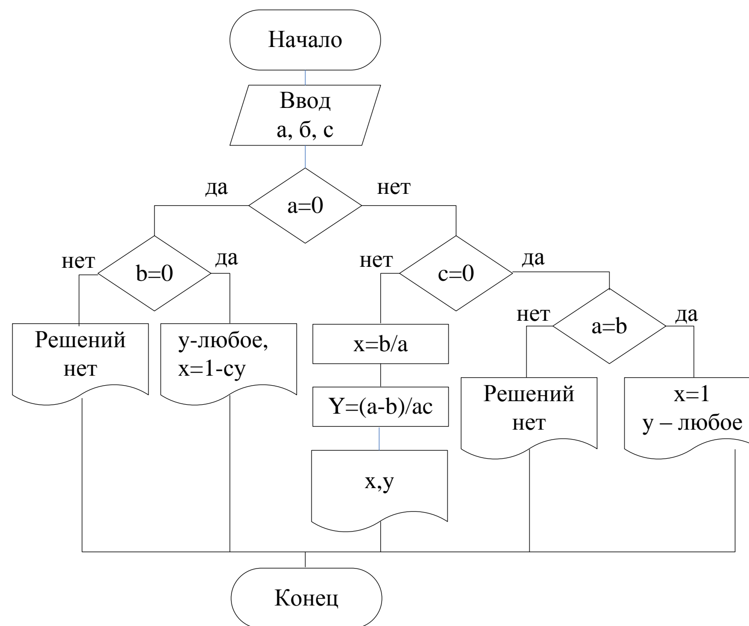

Рисунок 1 – Схема алгоритма решения системы уравнений

Текст программы реализует приведенную схему алгоритма.

```
#include <locale>
#include <stdio.h>
#include <conio.h>
int main(int argc, char* argv[])
{
      setlocale(0,"russian");
      float y,x,a,b,c;
      puts("Введите a,b,c:");
      scanf("%f %f %f",&a,&b,&c);
      printf("a=%5.2f   b=%5.2f  c=%5.2f\n",a,b,c);
     if (a==0)
        if (b==0) puts("Решение не существует.");
        else printf("y – любое,  x=1-c*y");
     else
        if (c==0)
                  if (a=b) puts("Решение не существует.");
                  else puts("x=1, y - любое");
     else
      {
            x=b/a;     y=(a-b)/a/c;
          printf("x= %7.3f   y=%7.3f\n",x,y);
      }
      puts("Нажмите любую клавишу для завершения...");
     _getch();
```

```
    return 0;
}
```

2 Операторы организации циклических процессов

Операторы цикла задают многократное повторение некоторой последовательности действий. В программировании выделяют три разновидности циклов, отличающихся способом определения количества повторений.

Итерационными называются циклы, в которых количество повторений заранее неизвестно. Оно определяется некоторой целевой функцией. В частности, это может быть точность вычислений. Такие циклы, например, реализуются при решении задачи нахождения суммы бесконечного ряда, определения значения интеграла на отрезке, длины кривой и т.д. Для организации итерационного цикла в С++ существуют два оператора: реализующие итерационные циклы: цикл с предусловием `while` и цикл с постусловием `do … while`.

Счетными называются циклы, в которых количество повторений заранее известно или легко определяется. Примерами подобных циклов являются вычисление степени числа, факториала числа, суммы определенного числа членов некоторой последовательности и т.д. В С++ для реализации счетного цикла используется оператор `for`.

Оператор Цикл-пока имеет следующий формат:

while (<Выражение>) <Оператор>;

где <Выражение> – выражение возможно сложное, состоящее из нескольких простых, определяет условие выполнения цикла;

<Оператор> – любой оператор С++, в том числе блок операторов.

Цикл выполняется до тех пор, пока результат выражения отличен от нуля, т.е. «Истина». Если условие нарушено сразу при входе в цикл, то тело цикла не выполнится ни разу.

Пример. Вычислить сумму ряда при $x > 1$ с заданной точностью ε:

$$S = 1 - 1/x + 1/x^2 - 1/x^3 + \dots$$

Рекуррентная формула определения очередного n+1-го члена ряда: $R_{n+1} = - R_n / x$.

На каждом шаге итерации сумма ряда определяется по формуле: $S = S + R_n$.

Условие выхода из цикла: $| R_n | < \varepsilon$.

Схема алгоритма приведена на рисунке 2.

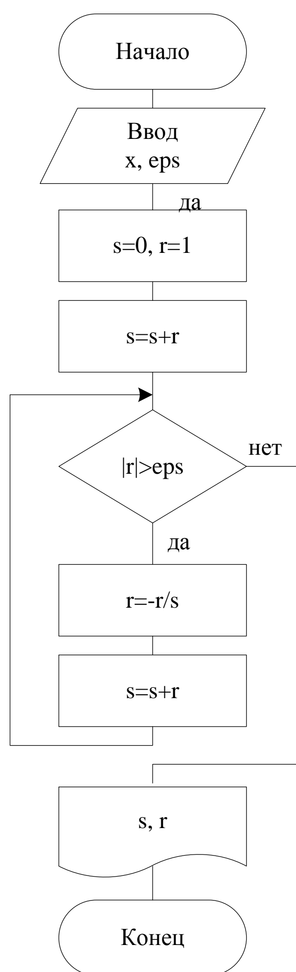

Рисунок 2 – Схема алгоритма программы

Текст программы:

```
#include <locale.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>
int main(int argc, char* argv[])
{
      setlocale(0,"russian");
     float s, r,x,eps;
      puts("Введите x, eps:"); scanf_s("%f  %f", &x, &eps);
      s=0; r=1; s+=r;
      while (fabs(r)>eps)
      {
            r=-r/x;     s+=r;
      }
      printf("Результат= %10.7f  r=%10.8f\n", s,r);
      puts("Нажмите любую клавишу для завершения...");
     _getch();
```

```
    return 0;
}
```

Оператор цикла с постусловием отличается от оператора цикла с предусловием тем, что условие проверяется после выполнения оператора тела цикла. Цикл выполняется до тех пор, пока результат выражения «Истина», т.е. отличен от нуля. Поскольку проверка происходит после выполнения тела цикла, даже если условие сразу нарушено, тело цикла выполняется один раз. Формат оператора:

do <Оператор > while (<Выражение>) ; .

Пример. Разработать фрагмент программы, которая вводит только значения, входящие в заданный интервал.

```
do
{
   printf("Введите значение от %d до %d:",low,high);
   scanf_s("%d",&a);
}
while (a<low || a>high);
```

Оператор for реализует цикл, для которого известен диапазон и шаг некоторого параметра цикла:

for ([<Выражение1>];[<Выражение2>];[<Выражение3>])[<Оператор>;]

где <Выражение1> – инициализирующее выражение – выполняется один раз перед циклом и задает начальные значения переменным цикла. Может отсутствовать, но при этом точка с запятой остаются.

<Выражение2> – выражение условия – определяет предельное значение параметра цикла. Может отсутствовать, при этом точка с запятой остается.

<Выражение3> – выражение модификации – выполняется на каждой итерации после тела цикла, но до следующей проверки условия и обычно используется для изменения параметра(ов) цикла. Может отсутствовать;

<Оператор> – тело цикла – может быть любым оператором C++, блоком операторов или может отсутствовать.

Нетрудно заметить, что счетный цикл легко реализуется через цикл с предусловием (рисунок 3).

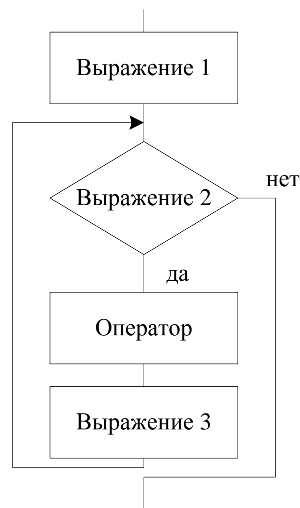

Рисунок 3 – Схема алгоритма оператора счетного цикла С++

Пример. Найти сумму N натуральных чисел.

На рисунке 4 приведена схема алгоритма программы.

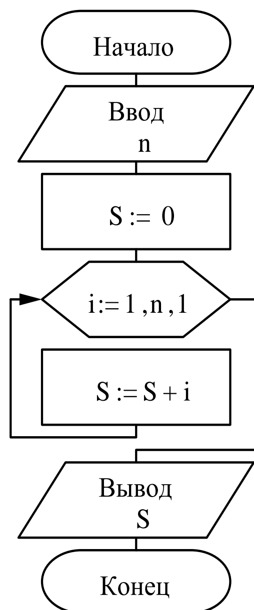

Рисунок 4 – Схема алгоритма программы

Текст программы:

```
#include <locale.h>
#include <stdio.h>
#include <conio.h>
int main(int argc, char* argv[])
{
     setlocale(0,"russian");
     int i,n,s;
```

```
    puts("Введите количество членов последовательности:");
    scanf_s("%d",&n);
    for (i=1,s=0;i<=n;i++) s+=i;
    printf("Сумма=%5d  при n=%3d\n",s,n);
    puts("Нажмите любую клавишу для завершения...");
   _getch();
    return 0;
}
```

Порядок выполнения работы

1. Прочитать и проанализировать задание в соответствии со своим вариантом.
2. Разработать схему алгоритма решения задачи.
3. Написать программу.
4. Вызвать среду программирования Visual C++, создать новый проект консольного приложения и ввести текст программы в среду программирования.
5. Подобрать тестовые данные (не менее 3-х вариантов).
6. Отладить программу на выбранных тестовых данных.
7. Продемонстрировать работу программы преподавателю.
8. Составить отчет по лабораторной работе.
9. Защитить лабораторную работу преподавателю.

Требования к отчету

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы также должны быть напечатаны при помощи компьютера или нарисованы с использованием чертежных инструментов, в том числе карандаша.

Каждый отчет должен иметь титульный лист, на котором указывается:

а) наименование факультета и кафедры;

б) название дисциплины;

в) номер и тема лабораторной работы;

г) фамилия преподавателя, ведущего занятия;

д) фамилия, имя и номер группы студента;

е) номер варианта задания.

Кроме того отчет по лабораторной работе должен содержать**:**

1) схему алгоритма, выполненную вручную или в соответствующем пакете;
2) текст программы;
3) результаты тестирования, которые должны быть оформлены в виде таблицы вида:

Исходные данные	Ожидаемый результат	Полученный результат

4) выводы.

Контрольные вопросы

1. В каких случаях используется конструкция «ветвление», конструкции «циклов»?
2. Какие операторы С++ реализуют ветвление и циклы в программе?
3. Какой тип цикла вы используете в своей программе и почему?
4. Какой синтаксис имеют этот операторы?
5. В чем заключаются основные отличия синтаксиса операторов С++ и Паскаля?

Лабораторная работа № 4

Программирование обработки массивов

Цель работы: изучение средств языка С++ для программирования описаний и обработки массивов

Объем работы: 2 часа

Теоретическая часть

1 Одномерные массивы

По правилам Си и С++ одномерный массив можно объявить:

- статически – с использованием абстракции массив и указанием его размера, например:

`int a[10];` // массив на 10 целых чисел, индекс меняется от 0 до 9

`unsigned int koord[10];` // массив целых беззнаковых чисел

- динамически – объявив только указатель на будущий массив и выделив память под массив во время выполнения программы, например:

```
int *dinmas;     // объявление указателя на целое число
dinmas=new int [100]; // выделение памяти под массив на 100 элементов
...
delete [] dibmas;
```

Пример. Программа определения максимального элемента массива и его номера:

```
#include <locale.h>
#include <stdio.h>
#include <conio.h>
int main(int argc, char* argv[])
{
      setlocale(0,"russian");
      float a[5],amax;  int i,imax;
      puts("Введите 5 значений:");
      for(i=0;i<5;i++) scanf("%f",&a[i]);
     amax=a[0];   imax=0;
     for(i=1;i<5;i++)
            if(a[i]>amax)
            {
                  amax=a[i]; imax=i;
            }
      puts("Значения:");
      for(i=0;i<5;i++) printf("%7.2f ",a[i]);
```

```
    printf("\n");
    printf("Максимум = %7.2f  номер = %5d\n",amax, imax);
    puts("Нажмите любую клавишу для завершения...");
    _getch();
    return 0;
}
```

Следует помнить, что по правилам Си и С++ независимо от способа объявления массива его имя – это имя переменной-указателя, содержащего адрес первого элемента массива. Поэтому для адресации элементов массива независимо от способа объявления можно использовать адресную арифметику. При этом следующие формы обращения эквивалентны:

`(list+i)` ⇔ `&(list[i])` // адреса элементов

`*(list+i)` ⇔ `list[i]` // значения элементов

2 Многомерные массивы

Так же, как и одномерные массивы, двух- и более мерные массивы можно объявить:

- статически, например:

  `int a[4][5];` // матрица элементов целого типа из 4 строк и 5 столбцов,
  // индексы меняются: первый от 0 до 3, второй от 0 до 4

  `float b[10][20][2];` // трехмерный массив вещественных чисел из
  // 10 строк, 20 столбцов и 2 слоев

- динамически, с помощью указателей, например:

  `short **matr;` .

При статическом объявлении помять будет предоставлена одним куском по количеству определенных элементов. Элементы в памяти будут расположены построчно: элементы нулевой строки, элементы первой строки и т.д. (см. рисунок 1).

Рисунок 1 – Размещение элементов статической матрицы в памяти

При динамическом описании обычно реализуют более удобные структуры. Например матрицу с указателями, хранящими адреса строк матрицы в явном виде – *массив динамических векторов* (см. рисунок 2):

`float **D2;` // объявлен указатель на матрицу

`D2=new float *[3];` // выделение памяти под массив указателей на строки
// матрицы

```
for(int i=0;i<3; i++)
        D2[i]=new float [4]; // выделение памяти под элементы строк
```

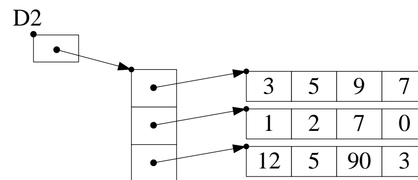

Рисунок 2 – Массив динамических векторов

Обращение к элементам этой структуры может выполняться также как и к элементам матрицы, например:

```
D2[1][2]=3;
```

Пример. Написать программу, которая сортирует строки матрицы по возрастанию элементов с использованием указателей.

```
#include <locale.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int **mas,*ptr;
int b,n,m,i,j,k;
int main()
{
      setlocale(0,"russian");
// Выделение памяти под матрицу
      printf("Введите n=");
      scanf_s("%d",&n);
      printf("Введите m=");
      scanf_s("%d",&m);
      mas=new  int *[n];         // выделение памяти под массив указателей
      for(i=0;i<n;i++)
            mas[i]=new int[m];   // выделение памяти под строки матрицы
//Заполнение матрицы данными
      for(i=0;i<n;i++)
      {
            printf("Введите %d элемента %d-й строки\n",m,i);
            for (j=0;j<m;j++)
                  scanf_s("%d",&mas[i][j]);
      }
// Вывод исходной матрицы на экран
      puts("Введенная матрица:");
      for(i=0;i<n;i++)
      {
```

```
        for (j=0;j<m;j++)
            printf("%3d",mas[i][j]);
        printf("\n");
    }
```

// Сортировка строк матрицы - реализована через указатели

```
    for(i=0;i<n;i++)
    {
        k=1;
        while(k!=0)
        {
            ptr=mas[i];
            for(k=0,j=0;j<m-1;ptr++,j++)
                if (*ptr>*(ptr+1))
                {
                    b=*ptr;     *ptr=*(ptr+1);   *(ptr+1)=b;
                    k++;
                }
        }
    }
```

// Вывод результата

```
    puts("Сортированная матрица:");
    for(i=0;i<n;i++)
    {
        for (j=0;j<m;j++)
            printf("%3d",mas[i][j]);
        printf("\n");
    }
```

// Удаление динамической матрицы

```
    for(i=0;i<n;i++)        // удаление строк динамической матрицы
        delete[] mas[i];
    delete[] mas;           // удаление массива указжателей на строки
    puts("Нажмите любую клавишу для завершения...");
    _getch();
    return 0;
}
```

Порядок выполнения работы

1. Прочитать и проанализировать задание в соответствии со своим вариантом.
2. Разработать схему алгоритма решения задачи.
3. Написать программу.

4. Вызвать среду программирования Visual C++, создать новый проект консольного приложения и ввести текст программы в среду программирования.

5. Подобрать тестовые данные (не менее 3-х вариантов).

6. Отладить программу на выбранных тестовых данных.

7. Продемонстрировать работу программы преподавателю.

8. Составить отчет по лабораторной работе.

9. Защитить лабораторную работу преподавателю.

Требования к отчету

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы также должны быть напечатаны при помощи компьютера или нарисованы с использованием чертежных инструментов, в том числе карандаша.

Каждый отчет должен иметь титульный лист, на котором указывается:

а) наименование факультета и кафедры;

б) название дисциплины;

в) номер и тема лабораторной работы;

г) фамилия преподавателя, ведущего занятия;

д) фамилия, имя и номер группы студента;

е) номер варианта задания.

Кроме того отчет по лабораторной работе должен содержать**:**

1) схему алгоритма, выполненную вручную или в соответствующем пакете;
2) текст программы;
3) результаты тестирования, которые должны быть оформлены в виде таблицы вида:

Исходные данные	Ожидаемый результат	Полученный результат

4) выводы.

Контрольные вопросы

1. Чем массив С++ отличается от массива Паскаля? Что такое «адресная арифметика»?
2. Как объявить массив в программе?
3. Как выполнить доступ к элементам массива?
4. Как осуществляется ввод-вывод элементов массива?

Лабораторная работа № 5

Программирование обработки текстовой информации

Цель работы: изучение средств языка C++ для программирования обработки текстовой информации

Объем работы: 2 часа

Теоретическая часть

В C++ символьная строка определяется как массив символов, который заканчивается *нуль-символом* – маркером конца строки «\0» (см. рисунок 1). Если маркер конца строки в символьном массиве отсутствует, то такой массив строкой не является. Маркер конца строки позволяет отслеживать реальную длину строки, которая может быть меньше размера массива.

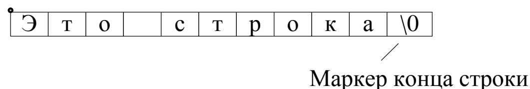

Рисунок 1 – Внутреннее представление строки

Обращение к элементам строки осуществляется по индексу, так же как и к элементу массива. Соответственно и нумерация символов начинается с 0.

Строку можно объявить теми же способами, что и одномерный массив символов. Единственно при объявлении надо учитывать, что любая строка должна заканчиваться маркером конца строки (значение 0), под который должен быть выделен один байт:

- объявление со статическим выделением памяти на этапе компиляции

`char` <Имя строки>[<Размер>] [= <Строковая константа>];

- объявление указателя на строку

`char` *<Имя указателя>[= < Строковая константа>];

Второй вариант предполагает, что либо строка инициализируется при объявлении, либо память под строку выделяют отдельно из области динамической памяти.

Примеры:

а) `char str[6];` // под строку выделено 6 байт, т.е. она может иметь длину до 5

б) `char *stroka = "Пример";` // под строку выделено 7 байт

в) `char *ptrstr;`

```
  ptrstr=new char[6];  // динамически выделили 6 байт
     …
  delete[] ptrstr;     // освободили память
```

Между способами а и б-в, так же, как и для массивов, существует существенное различие. В первом случае `str` – неизменяемый указатель, значение которого устанавливается один раз, когда под строку распределяется память (см. рисунок 2). К этому указателю нельзя применять адресную арифметику.

Во втором случае `ptrstr` и `stroka` – обычные указатели, которые можно изменять. Причем если указатель `ptrstr` утратит свое исходное значение, то станет невозможным корректное освобождение выделенной под строку памяти.

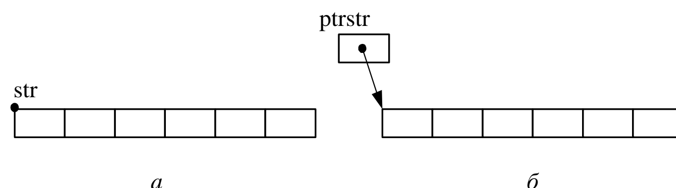

Рисунок 2 – Различие между способами описания строки:

а – неизменяемый указатель; *б* – изменяемый указатель

Пример. Разработать программу, которая выделяет слова их исходной строки с использованием функции `strtok_s`().

```
#include <locale.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main( void )
{
      // исходная строка
      char string[]="A string\tof ,,tokens\nand some more tokens";
      // строка разделителей
      char seps[] = " ,\t\n", *token, *context;
      setlocale(0,"russian");
      token = strtok_s( string,seps,&context);
      while(token!=NULL)
    {
            printf("%s ",token);
            token=strtok_s(NULL,seps,&context);
    }
      puts("\nНажмите любую клавишу для завершения...");
      _getch();
      return 0;
}
```

Порядок выполнения работы

1. Прочитать и проанализировать задание в соответствии со своим вариантом.
2. Разработать схему алгоритма решения задачи.
3. Написать программу.
4. Вызвать среду программирования, создать новый проект консольного приложения и ввести текст программы в среду программирования.
5. Подобрать тестовые данные (не менее 3-х вариантов).
6. Отладить программу на выбранных тестовых данных.
7. Продемонстрировать работу программы преподавателю.
8. Составить отчет по лабораторной работе.
9. Защитить лабораторную работу преподавателю.

Требования к отчету

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы также должны быть напечатаны при помощи компьютера или нарисованы с использованием чертежных инструментов, в том числе карандаша.

Каждый отчет должен иметь титульный лист, на котором указывается:

а) наименование факультета и кафедры;

б) название дисциплины;

в) номер и тема лабораторной работы;

г) фамилия преподавателя, ведущего занятия;

д) фамилия, имя и номер группы студента;

е) номер варианта задания.

Кроме того отчет по лабораторной работе должен содержать**:**

1) схему алгоритма, выполненную вручную или в соответствующем пакете;
2) текст программы;
3) результаты тестирования, которые должны быть оформлены в виде таблицы вида:

Исходные данные	Ожидаемый результат	Полученный результат

4) выводы.

Контрольные вопросы

1. Как организована «строка» в C++? Чем ее организация отличается от строки Паскаля?
2. Как используется адресная арифметика при работе со строками?
3. Чем принципиально отличаются функции и процедуры строк от аналогичных подпрограмм Паскаля?
4. Какой синтаксис имеет описание строк?
5. Чем различаются статические и динамические строки?

Лабораторная работа № 6

Программирование с использованием динамических структур данных

Цель работы: изучение средств языка С++, используемых при работе с динамической памятью

Объем работы: 6 часов.

Теоретическая часть

Динамическими принято называть структуры данных, которые создаются в процессе выполнения программы. Понятие «создаются» можно интерпретировать по-разному:

- создаются – в смысле «получают память и начинают реально существовать»;
- создаются – в смысле «организуются, строятся из некоторых элементов».

К первому типу относятся динамические массивы, строки и структуры, память под которые выделяется во время выполнения программы. К структурам второго типа относятся списки.

Список – структура, при организации которой использованы указатели, содержащие адреса следующих элементов. Элемент списка состоит из двух частей: информационной и адресной. *Информационная* часть содержит поля данных. *Адресная* – включает от одного до n указателей, содержащих адреса других элементов. Количество связей, между соседними элементами списка определяет его связность: односвязные, двусвязные, n-связные.

По структуре списки бывают линейными, древовидными и сетевыми. На линейных списках обычно реализуют разные дисциплины обслуживания:

- очередь – дисциплина обработки элементов данных, в которой добавление элементов выполняется в конец, а удаление – из начала;
- стек – дисциплина обработки элементов данных, в которой добавление и удаление элементов осуществляется с одной стороны;
- дек – дисциплина обработки элементов данных, в которой добавление и удаление элементов может выполняться с двух сторон.

Для описания элемента списка используют структуры, одно или несколько полей которых – указатели на саму эту структуру. Ниже приведены примеры описания элементов односвязного и двусвязного списков.

Элемент односвязного списка:

```
struct element // тип элемента
```

```
{
    char name[16];      // информационное поле 1
    char telefon[7];    // информационное поле 2
    element *p;         // адрес следующего элемента
};
```

Элемент двусвязного списка:

```
struct element          // тип элемента
{
    char name[16];      // информационное поле 1
    char telefon[7];    // информационное поле 2
    element *prev;      // адресное поле «предыдущий»
    element *next;      // адресное поле «следующий»
};
```

Создание списковой структуры предполагает:

- описание элемента списка;
- объявление указателей для работы со списком;
- создание пустого списка;
- добавление элементов к списковой структуре и удаление их из нее в процессе работы.

Рассмотрим эти операции на конкретном примере.

1. Описание элемента списка:

```
struct element  // тип  на элемента
{
     int num;        // целое число
     element *p;    // указатель на следующий элемент
};
```

2. Описание переменной – указателя списка и нескольких переменных-указателей в статической памяти:

```
element * first,     // адрес первого элемента
          *n,*f,*q;   // вспомогательные указатели
```

3. Исходное состояние – «список пуст»:

```
first=NULL;
```

Добавление элементов в список. Возможно несколько вариантов добавления элементов к списку:

- добавление элемента к пустому списку;
- перед первым, например, при построении списка по типу стека;
- после последнего, например, при построении списка по типу очереди;
- после/перед заданным элементом, например при построении сортированного списка.

1 Добавление элемента к пустому списку:

```
first=new element;    // запросили память под элемент
first->num=5;         // занесли данные в информационное поле
first->p=NULL;        // записали признак конца списка NULL
```

2 Добавление элемента перед первым (по типу стека):

```
q=new element;        // запросили память под элемент
q->num=4;             // занесли данные в информационное поле
q->p=first;           // записали в новый элемент адрес первого
first=q;              // записали в качестве первого адрес нового элемента
```

3 Добавление элемента после первого (по типу очереди):

```
q=new element;        // запросили память под элемент
q->num=4;             // занесли данные в информационное поле
q->p=NULL;            // записали признак конца списка NULL
first->p=q;           // записали признак конца списка NULL
```

Удаление элемента из списка. При удалении элемента из списка также возможны варианты:

- удаление первого элемента;
- удаление элемента с адресом q;
- удаление последнего элемента.

1. Удаление первого элемента

```
q=first;              // скопировали адрес первого элемента
firs=first->p;        // запомнили адрес нового первого элемента
delete q;             // удалили бывший первый элемент
```

2. Удаление элемента с адресом q

```
f=first;              // скопировали адрес первого элемента
while(f->p!=q)  f=f->p;  // нашли предыдущий элемент
q=q->p;               // перешли к следующему элементу
delete f->p;          // удалили элемент с адресом q
f->p=q;               // сменили адрес в предыдущем элементе
```

3. Удаление последнего элемента

```
f=q=first;              // скопировали адрес первого элемента
while(q->p!=NULL)       // нашли последний и предыдущий элементы
{
   f=q;
   q=q->p;
}
f->p=NULL;              // объявили предыдущий элемент последним
delete q;               // удалили последний элемент
```

Пример. Написать программу, которая формирует список деталей, содержащий наименование и диаметр детали. Удалить из списка все детали с диаметром, меньшим 1.

```
#include <locale.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
struct zap{       // тип элемента
      char det[10];
      float diam;
      zap *p;
};
int main(int argc, char* argv[])
{
      setlocale(0,"russian");
     zap a,*r,*q,*f;
     r=new zap;
     r->p=NULL;
      puts("Вводите названия деталей и их диаметр:");
     scanf("%s %f\n",r->det,&r->diam);
      while( (scanf_s("\n%s",a.det,sizeof(a.det)-1 )),
                  strcmp(a.det,"end")!=0 ){
            scanf("%f",&a.diam);
          q=r;
          r=new zap;  strcpy(r->det,a.det);      r->diam=a.diam;
          r->p=q;
      }
      // удаление записей
      q=r;
      do {
            if (q->diam<1){
                  if( q==r)   {
```

```
                        r=r->p;     delete(q);   q=r;
                } else  {
                        q=q->p;     delete(f->p);     f->p=q;
                }
        } else {
                f=q; q=q->p;
        }
    } while(q!=NULL);
    q=r;
    puts("Результаты:");
    if(q==NULL) puts("Данные отсутствуют.");
    else
        do {
                printf("%s %5.1f\n",q->det,q->diam);
                q=q->p;
        } while (q!=NULL);
    puts("Нажмите любую клавишу для завершения...");
    _getch();
    return 0;
}
```

Порядок выполнения работы

1. Прочитать и проанализировать задание в соответствии со своим вариантом.
2. Разработать схему алгоритма решения задачи.
3. Написать программу.
4. Вызвать среду программирования, создать новый проект консольного приложения и ввести текст программы в среду программирования.
5. Подобрать тестовые данные (не менее 3-х вариантов).
6. Отладить программу на выбранных тестовых данных.
7. Продемонстрировать работу программы преподавателю.
8. Составить отчет по лабораторной работе.
9. Защитить лабораторную работу преподавателю.

Требования к отчету

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы также должны быть напечатаны при помощи

компьютера или нарисованы с использованием чертежных инструментов, в том числе карандаша.

Каждый отчет должен иметь титульный лист, на котором указывается:

а) наименование факультета и кафедры;

б) название дисциплины;

в) номер и тема лабораторной работы;

г) фамилия преподавателя, ведущего занятия;

д) фамилия, имя и номер группы студента;

е) номер варианта задания.

Кроме того отчет по лабораторной работе должен содержать**:**

1) схему алгоритма, выполненную вручную или в соответствующем пакете;
2) текст программы;
3) результаты тестирования, которые должны быть оформлены в виде таблицы вида:

Исходные данные	Ожидаемый результат	Полученный результат

4) выводы.

Контрольные вопросы

1. Что такое «динамическая память»? В каких случаях она используется?
2. Как организуются данные в динамической памяти?
3. Как описать ресурсы, необходимые для создания списка?
4. Как выполняется добавление элемента к списку?
5. Как выполняется удаление элемента списка?

Лабораторная работа № 7

Создание простых классов

Цель работы: изучение средств языка С++ используемых при объявлении простых классов

Объем работы: 4 часа

Теоретическая часть

В С++ так же, как и в других языках программирования, класс – создаваемый программистом структурный тип данных, который используется для описания множества объектов предметной области, имеющих общие свойства и поведение.

Класс объявляется следующим образом:

```
class <Имя класса>
{
 private:      <Внутренние (недоступные) компоненты класса>
 protected:    <Защищенные компоненты класса>
 public:       <Общие (доступные) компоненты класса>
};
```

Описание предусматривает три секции. Компоненты класса, объявленные в секции `private`, называются *внутренними*. Они доступны только компонентным функциям того же класса и функциям, объявленным *дружественными* описываемому классу.

Компоненты класса, объявленные в секции `protected`, называются *защищенными*. Они доступны компонентным функциям не только данного класса, но и его потомков. При отсутствии наследования – интерпретируются как внутренние.

Компоненты класса, объявленные в секции `public`, называются *общими*. Они доступны за пределами класса в любом месте программы. Именно в этой секции осуществляется объявление полей и методов *интерфейсной части* класса.

Если при описании класса тип доступа к компонентам не указан, то по умолчанию принимается тип `private`.

В качестве компонентов в описании класса фигурируют *поля*, применяемые для хранения параметров объектов, и *функции*, описывающие правила взаимодействия с этими полями. В соответствии со стандартной терминологией ООП функции – компоненты класса или *компонентные функции* можно называть *методами*.

Пример 1.1. Описание класса.

А. Описание компонентных функций внутри класса.

```
#include <stdio.h>
class First
{
public:
 char c;
 int x,y;
 /* компонентные функции, определенные внутри класса */
```

```
 void print()
 {
      printf ("%c %d %d ",c,x,y);
 }
 void set(char ach,int ax,int ay)
 {
      c=ach;     x=ax;     y=ay;
 }
};
```

Б. Описание компонентных функций вне класса.

```
#include <stdio.h>
class First
{
  public:   char c;
            int x,y;
   void print();
   void set(char ach,int ax,int ay);
};
   /* компонентные функции, описанные вне класса */
void First::print()
{   printf ("%c %d %d ",c,x,y);    }
void First::set (char ach,int ax,int ay)
{   c=ach;     x=ax;     y=ay;   }
```

В программе, использующей классы, по мере необходимости объявляют объекты этих классов.

Объекты – переменные программы, соответственно на них распространяются общие правила длительности существования и области действия переменных, а именно:

- внешние, статические и внешние статические объекты создаются до вызова функции `main()` и уничтожаются по завершении программы;
- автоматические объекты создаются каждый раз при вызове функции, в которой они объявлены, и уничтожаются при выходе из нее;
- объекты, память под которые выделяется динамически, создаются оператором `new` и уничтожаются оператором `delete`.

При объявлении полей в описании класса не допускается их инициализация, поскольку в момент описания класса память для размещения его полей еще не выделена. Выделение памяти осуществляется не для класса, а для объектов этого класса, поэтому возможность инициализации полей появляется только во время или после объявления объекта конкретного класса.

Объявление объектов и способы инициализации их полей зависят от наличия или отсутствия в классе специального инициализирующего метода – *конструктора*, а также от того, в какой секции класса описано инициализируемое поле. Конструктор, являясь методом класса, может инициализировать любое поле объекта при его создании. Если в классе отсутствует конструктор, но описаны защищенные `protected` или скрытые `private` поля, то возможно создание только неинициализированных объектов. Для этого

используется стандартная конструкция объявления переменных или указателей на них. Например:

```
First   a,    // объект класса First
        *b,   // указатель на объект класса First
      c[4];   // массив с из четырех объектов класса First
```

При объявлении указателя, как и для обычных переменных, память под объект не выделяется. Это необходимо сделать отдельно, используя операцию `new`, после работы с динамическим объектом память необходимо освободить:

```
b=new First; ... delete b;
```

Объект, созданный таким способом, называют *динамическим*.

Значения полей неинициализированных статических и динамических объектов или массивов объектов задают в процессе дальнейшей работы с объектами: защищенных и скрытых – только в методах класса, а общедоступных – в методах класса или непосредственным присваиванием в программе.

При отсутствии в классе конструктора и защищенных `protected` или скрытых `private` полей для объявления инициализированных объектов используют оператор инициализации, применяемый при создании инициализированных структур, например:

```
First a = {'A',3,4},
    c[4] = {{'A',1,4},{'B',3,5},{'C',2,6},{'D',1,3}};
```

Инициализирующие значения при этом должны перечисляться в порядке следования полей в описании класса.

Пример. Различные способы инициализации общедоступных полей объекта.

```
#include <locale.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>
class sstro
{
public:
 char str1[80];      int x,y;
 void set_str(char *vs)  // инициализация полей
 {
      strcpy(str1,vs);   x=0;  y=0;
 }
 void print(void)        // вывод содержимого полей
 {
      printf("x=%5d   y=%5d  str: ",x,y);
      puts(str1);
 }
};
void main()
{
 setlocale(0,"russian");
 sstro aa = {"Строка",200,400}; // инициализированный объект
 sstro bb,cc;     // неинициализированные объекты
```

```
 bb.x=200;     // инициализация посредством прямого обращения
 bb.y=150;
 strcpy(bb.str1,"Строка");
 cc.set_str("Строка"); // вызов инициализирующего метода
 aa.print();  bb.print(); cc.print();
 _getch();
}
```

Конструктор – метод класса, который *а в т о м а т и ч е с к и вызывается при выделении памяти под объект*.

По правилам С++ *конструктор* имеет то же имя, что и класс, не наследуется в производных классах, может иметь аргументы, но не возвращает значения, может быть параметрически перегружен.

Конструктор определяет операции, которые необходимо выполнить при создании объекта. Традиционно такими операциями являются инициализация полей класса и выделение памяти под динамические поля, если такие в классе объявлены. Явный вызов конструктора не возможен, что в некоторых случаях усложняет создание инициализированных объектов.

При освобождении объектом памяти автоматически вызывается другой специальный метод класса – *деструктор*. Имя деструктора по аналогии с именем конструктора, совпадает с именем класса, но перед ним стоит символ «~» («тильда»). Деструктор определяет операции, которые необходимо выполнить при уничтожении объекта. Обычно он используется для освобождения памяти, выделенной под динамические поля объекта данного класса конструктором, и при необходимости может быть объявлен виртуальным. Деструктор не возвращает значения, не имеет параметров и не наследуется производными классами. Класс может иметь только один деструктор или не иметь ни одного. В отличие от конструктора деструктор может вызываться явно.

Момент уничтожения объекта, а, следовательно, и автоматического вызова деструктора определяется типом памяти, выбранным для размещения объекта: локальная, глобальная, внешняя и т. д. Если программа завершается с использованием функции `exit()`, то вызываются деструкторы только глобальных объектов. При аврийном завершении программы, использующей объекты некоторого класса, функцией `abort()` деструкторы объектов не вызываются.

Пример. Создание, инициализация и уничтожение объекта при наличии в классе конструктора и деструктора.

```
#include <locale.h>
#include <iostream>
using namespace std;
class Num
{
    int n;
 public:
    Num(int an){   cout<<"Конструктор"<<endl; n=an;   }
    ~Num()      {   cout<<"Деструктор"<<endl;   }
};
void main(int argc, char* argv[])
```

```
{
    setlocale(0,"russian");
    Num N(56);
    system("pause");
}
```

Порядок выполнения работы

1. Прочитать и проанализировать задание в соответствии со своим вариантом.

2. Разработать диаграмму объектов и диаграмму классов для указанных в задании объектов.

3. Написать программу.

4. Вызвать среду программирования, создать новый проект консольного приложения и ввести текст программы в среду программирования.

5. Подобрать тестовые данные (не менее 3-х вариантов).

6. Отладить программу на выбранных тестовых данных.

7. Продемонстрировать работу программы преподавателю.

8. Составить отчет по лабораторной работе.

9. Защитить лабораторную работу преподавателю.

Требования к отчету

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы также должны быть напечатаны при помощи компьютера или нарисованы с использованием чертежных инструментов, в том числе карандаша.

Каждый отчет должен иметь титульный лист, на котором указывается:

а) наименование факультета и кафедры;

б) название дисциплины;

в) номер и тема лабораторной работы;

г) фамилия преподавателя, ведущего занятия;

д) фамилия, имя и номер группы студента;

е) номер варианта задания.

Кроме того отчет по лабораторной работе должен содержать**:**

1) диаграмму объектов и диаграмму классов, выполненные вручную или в соответствующем пакете;

2) текст программы;

3) результаты тестирования, которые должны быть оформлены в виде таблицы вида:

Исходные данные	Ожидаемый результат	Полученный результат

4) выводы.

Контрольные вопросы

1. Что такое «класс»? В каких случаях используется эта конструкция?
2. Что такое «объект»?
3. Какие диаграммы используют для описания взаимодействия классов и объектов?
4. Какой синтаксис имеет объявление класса и объектов этого класса?
5. Объясните различие между описанием класса с конструктором и без конструктора?
6. Как можно инициализировать объекты класса, описанного без конструктора?

Лабораторная работа № 8

Создание классов с использованием наследования

Цель работы: изучение средств языка С++, используемых при наследовании классов

Объем работы: 6 часов

Теоретическая часть

Наследованием называют конструирование новых более сложных *производных* классов (классов-потомков) из уже имеющихся *базовых* классов (классов-родителей) посредством добавления полей и методов. Это – эффективное средство расширения функциональных возможностей существующих классов без их перепрограммирования и повторной компиляции существующих программ.

По определению компонентами производного класса являются:

- компоненты базового класса, за исключением конструктора, деструктора и компонентной функции, переопределяющей операцию «присваивания» (=) (см. раздел 5.4);
- компоненты, добавляемые в теле производного класса.

В функциональном смысле производные классы являются более мощными по отношению к базовым классам, так как, включая поля и методы базового класса, они обладают еще и своими компонентами.

Ограничение доступа к полям и функциям базового класса при наследовании осуществляется с помощью специальных описателей, определяющих вид наследования:

`class` <Имя производного класса >:

<Вид наследования><Имя базового класса>{<Тело класса>};

где вид наследования определяется ключевыми словами: `private`, `protected`, `public`.

Видимость полей и функций базового класса из производного определяется секцией, в которой находится объявление компонента и видом наследования (см. таблицу 1).

Таблица 1. Видимость компонентов базового класса в производном

Вид наследования	Объявление компонентов в базовом классе	Видимость компонентов в производном классе
`private`	`private`	не доступны
	`protected`	`private`
	`public`	`private`
`protected`	`private`	не доступны
	`protected`	`protected`
	`public`	`protected`
`public`	`private`	не доступны
	`protected`	`protected`
	`public`	`public`

Если вид наследования явно не указан, то по умолчанию принимается `private`. Однако хороший стиль программирования требует, чтобы в любом случае вид наследования был задан явно.

В языке С++ *конструкторы и деструкторы базового класса в производных классах не наследуются*. Однако если базовый класс содержит хотя бы один конструктор и деструктор, то производный класс также должен включать собственные конструктор и деструктор. При этом С++ поддерживает определенные правила взаимодействия между этими компонентами базовых и производных классов.

При создании объектов производного класса предусмотрен *автоматический вызов конструктора базового класса* для инициализации его полей. Однако следует помнить, что по умолчанию осуществляется вызов конструктора базового класса *без параметров*. Если такой конструктор в базовом классе отсутствует, то компилятор выдает сообщение об ошибке `error C2512`.

Поскольку явный вызов конструктора базового класса в программе невозможен, чтобы передать этому конструктору аргументы для инициализации полей базового класса, следует:

- добавить соответствующие параметры к собственным параметрам конструктора производного класса;
- вызвать конструктор базового класса *в списке инициализации* конструктора производного класса, передав ему соответствующие аргументы.

```
class A
{
     int x;
  public:
     A(int ax):x(ax){} // конструктор базового класса
};
class B
{
```

```
        int y;
      public:
        B(int ax,int ay):A(ax),y(ay){} // конструктор производного класса
    };
```

При этом соблюдается строгий порядок конструирования полей базового и производного классов: не зависимо от порядка указания в списке инициализации сначала вызывается конструктор базового класса, а затем – конструкторы полей, объявленных в производном классе.

Пример 2.2. Проектирование классов с использованием наследования (классы Целое число и Вещественное число).

Пусть требуется разработать классы для реализации объектов Целое число и Вещественное число. Объект Целое число должен хранить длинное целое в десятичной записи и уметь выводить его значение. Объект Вещественное число должен хранить вещественное число, задаваемое в виде `ccccc.dddddd`, и его символьное представление. Он также должен уметь выводить свое значение на экран.

Для обоих объектов необходимо предусмотреть возможность инициализации как в момент объявления переменной, так и в процессе функционирования. Поскольку вещественное число включает длинное целое как целую часть, класс для его реализации можно наследовать от класса, реализующего длинное целое число (рис. 1).

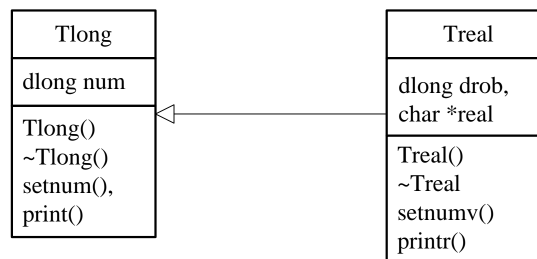

Рис. 1. Диаграмма классов при наследовании

```
#include <locale.h>
#include <stdlib.h>
#include <iostream>
#include <string.h>
using namespace std;
typedef  unsigned  long  dlong;
class Tlong                         // Класс Целое число
{
 public:
     dlong num;            // числовое поле класса
     Tlong(){}             // неинициализирующий конструктор
     Tlong(dlong an):num(an){} // конструктор
     ~Tlong(){}              // деструктор
     void print(void)       // вывод значения поля
    { cout<<" Целое число : "<<num<<endl; }
     void setnum(dlong an)  // инициализации поля
     { num=an; }
```

```
};
class  Treal: public Tlong     // Класс Вещественное число
{
 public:
     dlong drob;        // дробная часть числа
     char *real;        //  запись вещественного числа
     Treal(){real=NULL;}      // конструктор без параметров
     Treal(char *st) :Tlong() // инициализирующий конструктор
    {   setnumv(st); }
     ~Treal()// деструктор
     {   if (real!=NULL) delete [] real;  }
     void printr();   // вывод вещественного числа
     void setnumv(char * st);   // инициализация полей класса
  };
void Treal::setnumv(char * st)
{
    int l=strlen(st);  char *ptr;
    real=new char[l+1]; strcpy(real,st);
    ptr=strchr(real,'.'); *ptr='\0';
    drob=dlong(atol(ptr+1));
    num=dlong(atol(real));
    *ptr='.';
}
void Treal::printr()
{
    cout<<"Вещественное число: "<<real<<endl;
    cout<<"Целая часть: "; print();
    cout<<"Дробная часть: "<<drob<<endl;
}
void main ()
{
    setlocale(0,"russian");
    Treal a("456789.1234321"), // объект производного класса
          *pa=new Treal("456789.1234321"), // указатель
          mask[3]= // инициализированный массив объектов
         {
           Treal("1748.5932"),
           Treal("4567.34321"),
           Treal("18689.9421")
         };
    a.printr();
    pa->printr();     delete pa;
    for(int i=0;i<3;i++)
    {
        cout<<"Элемент массива "<<(i+1)<<": "<<endl;
```

```
            mask[i].printr();
        }
        system("pause");
    }
```

Порядок выполнения работы

1. Прочитать и проанализировать задание в соответствии со своим вариантом.
2. Разработать диаграмму объектов и диаграмму классов предметной области.
3. Написать программу.
4. Вызвать среду программирования, создать новый проект консольного приложения и ввести текст программы в среду программирования.
5. Подобрать тестовые данные (не менее 3-х вариантов).
6. Отладить программу на выбранных тестовых данных.
7. Продемонстрировать работу программы преподавателю.
8. Составить отчет по лабораторной работе.
9. Защитить лабораторную работу преподавателю.

Требования к отчету

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы также должны быть напечатаны при помощи компьютера или нарисованы с использованием чертежных инструментов, в том числе карандаша.

Каждый отчет должен иметь титульный лист, на котором указывается:

а) наименование факультета и кафедры;
б) название дисциплины;
в) номер и тема лабораторной работы;
г) фамилия преподавателя, ведущего занятия;
д) фамилия, имя и номер группы студента;
е) номер варианта задания.

Кроме того отчет по лабораторной работе должен содержать**:**

1) схему алгоритма, выполненную вручную или в соответствующем пакете;
2) текст программы;

3) результаты тестирования, которые должны быть оформлены в виде таблицы вида:

Исходные данные	Ожидаемый результат	Полученный результат

4) выводы.

Контрольные вопросы

1. Что такое «наследование»? В каких случаях используется этот механизм?
2. Как показать наследование на диаграмме объектом и на диаграмме классов?
3. Как описывается наследование классов в программе?
4. В чем заключаются особенности работы с конструкторами базового и производного классов?
5. В каких случаях фиксируются ошибки при выполнении конструкторов производных классов?