МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение
высшего образования
**«Сибирский государственный университет науки и технологий
имени академика М.Ф. Решетнева»**

О. А. Филимонова

ПРОГРАММИРУЕМ НА C++

Указатели. Массивы. Функции

Лабораторный практикум в 3 частях
Часть 2

Красноярск 2019

УДК 004.42
ББК 32.972(я7-5)
Ф53

Рецензенты:
кандидат технических наук, доцент К. В. Богданов
(Сибирский государственный университет науки и технологий
имени академика М.Ф. Решетнева);
кандидат технических наук, доцент А. В. Пятаева
(Сибирский федеральный университет)

Печатается по решению методической комиссии ИИТК

Филимонова, О.А.
Ф53 **Программируем на C++**: Указатели. Массивы. Функции. Лабораторный практикум в 3 ч. Ч. 2. / О.А. Филимонова ; Сиб. гос. ун-т науки и технологий. – Красноярск, 2019. – 90 с.

Издание является второй частью лабораторного практикума по дисциплине «Программирование». Рассматриваются работа с динамической памятью, массивами, построение функций. Предназначено для бакалавров направления 09.03.01 «Информатика и вычислительная техника», 09.03.02 «Информационные системы и технологии» всех форм обучения и иных студентов, специализирующихся в области информатики.

УДК 004.42
ББК 32.972(я7-5)

Оглавление

Предисловие

Дисциплина «Программирование» является одной из важных для подготовки бакалавров направления 09.03.01 «Информатика и вычислительная техника» и 09.03.02 «Информационные системы и технологии».

В данном издании рассматривается построение программ на языке программирования C++ в соответствии с парадигмой структурного и модульного программирования – в виде набора функций, каждая из которых является совокупностью или одной из трех типов базовых конструкций: последовательное исполнение, ветвление и цикл.

Издание является второй частью к лабораторному практикуму по дисциплине «Программирование» [1] и предназначено для бакалавров направления 09.03.01 «Информатика и вычислительная техника», 09.03.02 «Информационные системы и технологии» всех форм обучения и иных студентов, специализирующихся в области информатики.

Лабораторный практикум состоит из одиннадцати лабораторных работ, посвященных изучению указателей, массивов, функций, которые построены по однотипной структуре и включают в себя следующее.

1. Цель работы.

2. О чем необходимо помнить? (Моменты теории, на которых требуется акцентировать внимание.)

3. Порядок написания программы.

4. Примеры написания программ (полный листинг программ и разъяснения по построению алгоритма решения задач, а так же рекомендации по развитию проекта).

5. Порядок выполнения задания.

6. Контрольные вопросы и задания (помощь в проверке и закреплении знаний; задачи, решение которых необходимо для наилучшего усвоения тем).

7. Задания (15 вариантов задач различной сложности по каждой лабораторной работе).

Теоретический материал не претендует на полноту и подробность изложения, необходимо также руководствоваться материалами лекций по дисциплине, методическими пособиями [2] и книгами [3] – [7].

Тексты программ написаны на языке программирования C++ в интегрированной среде разработки приложений Microsoft Visual Studio 2010. Их необходимо не только рассматривать, но и вводить, выполнять, изменять и развивать для получения опыта и знаний.

1. УКАЗАТЕЛИ

Указатели – это переменные, которые содержат адрес области в оперативной памяти. Этой областью могут быть простые переменные какого-либо типа, массивы, исполняемый код функции и пр.

Указатели являются мощным средством косвенного доступа к данным – статическим и динамическим.

Статические данные создаются и сохраняют свое значение в соответствии с классом памяти. Например, данное, объявленное в блоке, будет уничтожено при выходе из него.

Динамические данные создаются операцией new и сохраняют свое значение, пока память не будет освобождена операцией delete. Кроме того, появляется возможность в процессе выполнения программы решать, какое количество однотипных данных требуется создать. Указатели важны и при работе с функциями.

Лабораторная работа 1

РАБОТА С ДИНАМИЧЕСКОЙ ПАМЯТЬЮ

Цель работы

Научиться работать с данными, используя их *адрес*, а именно:

– объявлять указатель в программе;

– помещать «правильный» адрес в переменную-указатель;

– выполнять операции над указателем;

– работать с динамической памятью;

– знать и уметь применять стандартные алгоритмы обработки данных.

О чем необходимо помнить?

Изучить основные положения теории.

Сначала **указатель необходимо объявить**. Хорошим стилем программирования является его обнуление. Например:

```
int * p1=0;
```

Объявлена переменная p1 – указатель на тип int. Это значит, что в нее может быть помещен адрес любой переменной типа int.

```
double * p2=0;
```

Объявлена переменная p2 – указатель на тип double. Это значит, что в нее может быть помещен адрес любой переменной типа double.

```
void * p3=0;
```

Особый указатель p3, в который можно поместить адрес *любой* переменной. При этом операции над таким указателем ограничены.

Затем **указатель необходимо инициализировать «правильным» адресом**:

- ✓ *Адресом существующей статической переменной:*

```
int a, *p;
p=&a;  //применяем & – операцию определения адреса переменной
```

- ✓ *Адресом первого байта выделенной динамической памяти:*

```
int *p1=new int;      //одна переменная типа int
int *p2=new int(5);  //одна переменная типа int со значением 5
int *p3=new int[6];  //шесть переменных типа int, расположенных в памяти подряд. Вместо константы 6 можно записать переменную, содержащую количество чисел.
```

После выполнения операции new в p1, p2, p3 *адрес первого числа*.

Не забывайте освобождать динамическую память, когда работа с ней завершена:

```
delete p1;
delete p2;
delete[]p3;
```

- ✓ *Адресом другого аналогичного указателя, уже правильно инициализированного:*

```
int a, *p, *p1;
p=&a;  //указатель правильно инициализирован
p1=p;
```

Выполнить необходимые операции над указателем:

- ✓ * - унарная операция косвенного доступа (операция разадресации) – определяет значение по адресу переменной;
- ✓ К указателю можно прибавить целое число n, при этом мы смещаемся в памяти вперед на n объектов (применяется для массивов).
- ✓ От указателя можно отнять целое число n, при этом мы смещаемся в памяти назад на n объектов (применяется для массивов).
- ✓ Два однотипных адреса можно сравнить с помощью операций отношения.
- ✓ Найти разность между двумя однотипными указателями и определить, на каком расстоянии находятся два объекта (имеет смысл для массивов).

*Назначение символа * может быть различным и определяется из контекста:*

- ✓ *Если * в операторе объявления типа – значит, объявлена переменная-указатель p (int *p;)*
- ✓ *Если * в другом месте, то это либо унарная операция косвенного доступа (*p=5;), либо бинарная операция умножения (c=a*b;), не имеющая прямого отношения к работе с указателями.*

При работе с числами необходимо знать и уметь применять **некоторые стандартные алгоритмы:**

✓ *Для вычисления суммы ряда чисел необходимо*:

– объявить переменную соответствующего типа под сумму и обнулить ее (double summa=0;)

– в процессе перебора чисел, обнаружив подходящее число, увеличить сумму на это число (summa+=ОчередноеЧисло;)

✓ *Для вычисления произведения ряда чисел необходимо:*

– объявить переменную соответствующего типа под произведение и инициализировать ее. При выборе типа помните, что произведение быстро растет! (double pr=1;)

– в процессе перебора чисел, обнаружив подходящее число, домножить произведение на это число (pr*=ОчередноеЧисло;)

✓ *Для вычисления количества чего-либо необходимо:*

– объявить переменную под количество и обнулить ее (int kol=0;)

– обнаружив подходящее число, увеличить количество, как правило, на 1 (kol++;)

В дальнейшем круг стандартных алгоритмов будет расширен.

Порядок написания программы

1. Выделить память динамически для нескольких переменных одного типа, расположенных подряд.
2. Ввести переменные с клавиатуры.
3. Провести указанную обработку данных.
4. Вывести на экран исходные данные.
5. Вывести результат обработки.
6. Освободить динамическую память.

Примеры написания программ

Задача. Выделить память динамически под 5 переменных целого типа, расположенных подряд, ввести их с клавиатуры. Вычислить произведение ненулевых чисел. Вывести исходные числа и результат.

Анализ решения задачи. После выделения динамической памяти под несколько переменных, расположенных подряд:

```
int *p=new int[5];
```

необходимо ввести и обработать каждое число. Как к ним обратиться? Используем известные операции над указателями: p – адрес первого числа (по определению операции new), значит, *p – значение первого числа (операция 1). Адрес второго числа вычисляем как p+1 (операция 2). Значение второго числа – *(p+1). Очень важны круглые скобки, которые устанавливают правильный порядок действий: унарная операция * имеет наивысший приоритет, но операция + должна быть выполнена раньше.

Далее вычисляем адрес третьего числа – p+2 и его значение – *(p+2) и т.д.

Замечаем, что действия однотипны – значит, они могут быть выполнены с помощью цикла:

```
for(i=0; i<5; i++)   // для перебора пяти чисел
      //обработка значения очередного числа  *(p+i)
```

В тексте программы обращайте внимание, где работа *с адресом (указателем)*, а где – со *значением по адресу*.

Программа будет строиться в виде проекта с одним файлом исходного кода.

Текст функции main() будет выглядеть так:

```
#include<iostream>          //подключение системных средств для
using namespace std;        //возможности использовать потоки ввода-вывода
#include <windows.h>        //прототип функций русификации
int main(void)        //имя функции main()
{
        SetConsoleCP(1251);       //вызов функций русификации
        SetConsoleOutputCP(1251);
        const int SIZE=5;  //количество чисел
        int *p=0;       //объявляем указатель p
        long product=1;     //переменная под произведение
        int i;   //переменная цикла
//выделяем память динамически
        p=new int[SIZE];  //инициализируем  указатель p
//заполнение чисел с клавиатуры
        cout<<"Введите числа: ";
        for(i=0; i<SIZE; i++)       //для перебора чисел
                cin>>*(p+i);    //ввод с клавиатуры значения i-го числа
//вычисление произведения
        for(i=0; i<SIZE; i++)       //для перебора чисел
                if(*(p+i)!=0)          // если значение i-го числа не 0
                        product *= *(p+i); //накапливаем произведение
//вывод чисел на экран
        cout<<"\nВаши числа: \n";
        for(i=0; i<SIZE; i++)       //для перебора чисел
                cout<< *(p+i)<<endl;      //вывод  значения i-го числа
        cout<< "\nПроизведение равно "<< product<<endl;
        delete [] p;   //освободили память
        return 0;
}
```

Порядок выполнения задания

1. Получить задание у преподавателя.
2. По вышеуказанному плану написать программу.
3. Ввести программу в компьютер.

4. Компилируя программу, найти и исправить ошибки синтаксиса.

5. Разработать достаточное количество контрольных примеров для обнаружения и исправления логических ошибок в программе.

6. Выполнить программу с исходными данными из контрольных примеров, убедиться в правильности решения задачи.

7. Предъявить правильно работающую программу и контрольные примеры для проверки преподавателю.

Контрольные вопросы и задания

1. Что такое указатель? Как объявить указатель в программе?

2. Какими способами можно поместить «правильный» адрес в указатель? Что такое «правильный» адрес?

3. Как обратиться к значению переменной, зная только ее адрес?

4. Какие операции можно выполнить над переменной-указателем?

5. Что получим, прибавив к адресу 1? Когда это имеет смысл делать?

6. Выделите память под одну вещественную переменную.

7. Выделите память под несколько вещественных переменных, расположенных в памяти подряд.

8. Выделите память под 3 вещественных переменных, расположенных в памяти **не** подряд. Поместите по адресам конкретные значения.

9. Объявите простую переменную целого типа. Определите ее адрес и поместите адрес в соответствующую переменную, предварительно ее объявив. Измените значение переменной по ее адресу.

10. Для наилучшего освоения темы выполните варианты 10, 8, 14.

Задание

Произвести действия с данными, выделив память под них **подряд,** динамически.

Вариант 1. Выделить память динамически под 7 переменных вещественного типа, ввести их с клавиатуры. Вычислить сумму чисел. Вывести исходные числа и результат.

Вариант 2. Выделить память динамически под 9 переменных целого типа, ввести их с клавиатуры. Вычислить количество ненулевых чисел. Вывести исходные числа и результат.

Вариант 3. Выделить память динамически под 5 переменных целого типа, ввести их с клавиатуры. Вычислить количество положительных чисел. Вывести исходные числа и результат.

Вариант 4. Выделить память динамически под n переменных вещественного типа, ввести их с клавиатуры. Увеличить каждое число на к. Вывести исходные числа и результат.

Вариант 5. Выделить память динамически под n переменных целого типа, ввести их с клавиатуры. Увеличить каждое число в к раз. Вывести исходные числа и результат.

Вариант 6. Выделить память динамически под n переменных вещественного типа, ввести их с клавиатуры. Выделить целые части каждого числа. Вывести исходные числа и результат.

Вариант 7. Выделить память динамически под n переменных вещественного типа, ввести их с клавиатуры. Выделить дробные части каждого числа. Например, если число равно 23,5673, то дробная часть равна 0,5673. Вывести исходные числа и результат.

Вариант 8. Выделить память динамически под n переменных вещественного типа, ввести их с клавиатуры. Поменять местами значения первого и последнего числа. Вывести исходные числа и результат.

Вариант 9. Выделить память динамически под n переменных вещественного типа, ввести их с клавиатуры. Определить номер первого отрицательного числа. Вывести исходные числа и результат.

Вариант 10. Выделить память динамически под n переменных целого типа, ввести их с клавиатуры. Вычислить сумму и количество отрицательных чисел. Вывести исходные числа и результат.

Вариант 11. Выделить память динамически под n переменных вещественного типа, ввести их с клавиатуры. Вычислить произведение положительных чисел. Вывести исходные числа и результат.

Вариант 12. Выделить память динамически под n переменных целого типа, ввести их с клавиатуры. Вычислить среднее арифметическое чисел. Вывести исходные числа и результат.

Вариант 13. Выделить память динамически под 7 переменных вещественного типа, ввести их с клавиатуры. Определить количество перемен знака. Вывести исходные числа и результат.

Вариант 14. Выделить память динамически под n ($n \geq 3$) переменных целого типа, ввести их с клавиатуры. Вычислить среднее геометрическое последних 3 чисел. Вывести исходные числа и результат.

Вариант 15. Выделить память динамически под n переменных вещественного типа, ввести их с клавиатуры. Определить *первый номер* числа,

большего некоторого, наперед заданного числа. Вывести исходные числа и результат.

2. МАССИВЫ

Массивы позволяют удобным способом размещать в памяти и обрабатывать большое количество однотипных данных.

Массив – это набор данных одного типа, расположенных в памяти подряд, имеющих одно имя, и отличающихся друг от друга по порядку следования.

Данные называются *элементами массива*. Пользователи программы различают элементы массива по *номеру* (первый, второй, третий и т.д.). Программисты же – по *индексу* (нулевой, первый, второй и т.д.).

Зная номер элемента массива легко вычислить индекс и наоборот. Например, первый элемент массива имеет индекс ноль. Элемент массива с индексом пять имеет номер шесть и т.д. Это не сложно, но помнить об этой разнице необходимо, так как, используя при написании программы вместо индекса номер, можно выйти за границы массива, а это уже серьезная ошибка.

Лабораторная работа 2

ОБРАБОТКА СТРОК

Цель работы

Научиться обрабатывать строки в программе, а именно:
– объявлять строки в программе статически и динамически;
– работать со строкой как с единым данным и посимвольно;
– вводить строки с пробелами внутри и без пробелов;
– писать стандартный цикл обработки строки;
– определять адрес любого символа строки;
– обращаться к символам строки по имени и через адрес;
– производить различные действия над символами.

О чем необходимо помнить?

Изучить основные положения теории.

Строка – это набор символов, завершающийся особым символом конца строки.

Особенностью работы со строкой является то, что *размер выделенной памяти* чаще не совпадает с *реальной длиной строки*. Например, хотим выделить память под фамилию студента. Сколько необходимо символов?

Мысленно перебираем возможные варианты (Ким, Иванов, Мережковский, Новиков-Прибой и пр.) Необходимо выделить столько памяти, чтобы самая длинная фамилия поместилась, например 30 или 50.
Но как обрабатывать такую строку, как узнать, где она заканчивается? Для этого после последнего символа строки ставится особый символ – символ конца строки, ноль-символ, который имеет специальное обозначение '\0'. Его код равен 0.

Память под строку можно выделить статически или динамически.

При статическом выделении памяти необходимо заранее определить и указать с помощью *константы* ее размер (количество символов):

```
const int SIZE=51;
```

или

```
#define SIZE 51
```

А затем объявить строку:

```
char str[SIZE];        //без инициализации
```

или

```
char str1[SIZE]= "это значение строки";       //с инициализацией
```

Имя строки должно быть «говорящим» о назначении данного.

Часто со строкой работаем как с единым данным благодаря стандартным функция, например:

```
cin>>str;        //ввод строки без пробелов внутри
cin.getline(str,SIZE);        //ввод строки с пробелами внутри
cout<<str<<endl;     //вывод любой строки
strcpy_s(str, SIZE,"Новое значение строки"); //копирование в строку str
```

Но не стоит забывать, что объявленная нами строка может состоять из не более 50 символов, каждый из которых мы можем анализировать, изменять, подсчитывать и т. д., то есть подвергать определенной обработке. При этом мы должны уметь работать с отдельными символами.

Для этого необходимо помнить, что:

str – имя строки или ее константный (не изменяемый) *адрес;*

str+i – *адрес* i-го символа строки;

*(str+i) – *значение* i-го символа строки, вычисляемого через ее адрес;

str[i] – *значение* i-го символа строки, вычисляемого через ее имя,

где i – *индекс* символа, начинающийся с нуля.

При **динамическом выделении памяти** можно заранее определить и указать с помощью *константы* ее размер (количество символов):

```
const int SIZE=51;
```

или

```
#define SIZE 51
```

или предварительно вычислить или ввести этот размер, например:

```
int size=strlen(str)+1;                //размер, как у строки str
```

или

```
int size;
cin>>size;      //ввод размера с клавиатуры
```

А затем объявить строку динамически:

```
char *str=new char[SIZE];
```

или

```
char *str=new char[size];
```

Объявление указателя и выделение памяти можно выполнить порознь:

```
char *str;
str=new char[size];
```

Следует помнить, что в нужном месте программы выделенная память должна быть освобождена:

```
delete []str;
```

Для работы с отдельными символами помним, что:

str – *адрес* строки*;*

str+i – *адрес* i-го символа строки;

*(str+i) или str[i] – *значение* i-го символа строки, вычисляемого через ее адрес,

где i – *индекс* символа, начинающийся с нуля.

Обработка строки не зависит от способа выделения памяти, поэтому все приведенное ранее верно:

```
cin>>str;        //ввод строки без пробелов внутри
cin.getline(str,size)  ;        //ввод строки с пробелами внутри
cout<<str<<endl;     //вывод любой строки
strcpy_s(str, size,"Новое значение строки"); //копирование в строку str
```

Обратим внимание на то, что массивы создаются для *одинаковой обработки каждого элемента*. Чтобы организовать одинаковую обработку каждого символа строки, необходимо написать **стандартный цикл обработки строки.**

В качестве переменной цикла можно использовать *индекс текущего символа строки*:

```
for (int i=0; str[i]!=0; i++)          //пока не обнаружим ноль-символ
{
  //алгоритм обработки значения i-го символа строки
}
```

или

```
for (int i=0;*(str+i)!=0; i++)        //пока не обнаружим ноль-символ
{
  //алгоритм обработки значения i-го символа строки
}
```

i – индекс текущего символа строки. Увеличивая индекс на 1 при каждой итерации, перемещаемся к следующему символу строки. При обнаружении ноль-символа (код равен 0) цикл завершается.

Можно написать цикл, используя в качестве переменной цикла *адрес текущего символа строки*:

```
for(char *p=str; *p!=0; p++)
{
  //алгоритм обработки значения текущего символа строки (*p)
}
```

По сути, все циклы ведут к одному результату, разная лишь форма обращения к символу строки. Все циклы могут быть применены и к статической, и к динамической строке.

Порядок написания программы

1. Выделить память под строку статически или динамически.
2. Ввести строку с клавиатуры.
3. Провести указанную обработку строки.
4. Вывести на экран исходную строку.
5. Вывести результат обработки.
6. Освободить динамическую память (если она выделялась).

Примеры написания программ

Задача 1. Написать программу, которая вычисляет *номер первого* символа строки, совпавшего с указанным символом. Если символа в строке нет, результат равен 0.

Анализ решения задачи. Нельзя путать два понятия – *индекс* символа строки и *номер* символа строки. Понятия схожи, но номер начинается с 1, а индекс с 0, с помощью индекса организуется перебор символов в строке. Зная индекс i, легко определить номер – i+1. Объявим переменную n и инициализируем ее нулем, который результатом быть не может (номер начинается с 1). Если символ в строке найдется, ноль заменится на реальный результат, иначе – ноль будет играть роль сигнала о том, что символ в строке не найден.

Программа будет строиться в виде проекта с одним файлом исходного кода.

1 вариант решения – обращение к элементу массива идет через *имя* элемента массива, память выделяется *статически*, строка вводится *без пробелов внутри*.

Текст функции main() будет выглядеть так:

```
#include<iostream>          //подключение системных средств для
using namespace std;        //возможности использовать потоки ввода-вывода
#include <windows.h>      //прототип функций русификации
int main(void)              //имя функции main()
{
```

```
	SetConsoleCP(1251);	//вызов функций русификации
	SetConsoleOutputCP(1251);
	const int SIZE=41;		//задали размер строки
	char str[SIZE], simvol;	//объявили данные
	int n=0,i;	//номер искомого элемента массива, индекс
// ввод исходных данных
	cout<<"\nВведите строку ";
	cin>>str;
	cout<<"Введите символ ";
	cin>>simvol;
//организуем перебор символов строки для их обработки
	for(i=0; str[i]!=0; i++)
		if(str[i]== simvol)   //анализ текущего символа строки
		{
			n= i+1;		//вычислен номер первого символа
			break;		//прекратили выполнение цикла
		}
	if(n!=0)	//если символ в строке найден
		cout<<"\nСимвол "<<n<<" по счету\n";
	else
		cout<<"Символ в строке не найден\n";
	return 0;
}
```

2 вариант решения – обращение к элементу массива идет через *адрес* элемента массива, память выделяется *динамически,* строка вводится с *пробелами внутри*.

Текст функции main() будет выглядеть так:

```
#include<iostream>		//подключение системных средств для
using namespace std;	//возможности использовать потоки ввода-вывода
#include <windows.h>	//прототип функций русификации
int main(void)		//имя функции main()
{
	SetConsoleCP(1251);	//вызов функций русификации
	SetConsoleOutputCP(1251);
	const int SIZE=41;		//задали размер строки
	char *str= new char[SIZE], simvol;	//объявили данные
	int n=0,i;	//номер символа и индекс
// ввод исходных данных
	cout<<"\nВведите строку ";
	cin.getline(str,SIZE);
	cout<<"Введите символ ";
	cin>>simvol;
//организуем перебор символов строки для их обработки
```

```
    for(i=0;*(str+i)!=0; i++)
        if(*(str+i)== simvol)    //анализ текущего символа строки
        {
            n= i+1;        //вычислен номер первого символа
            break;         //прекратить выполнение цикла
        }
    if(n!=0)       //если символ в строке найден
        cout<<"Символ "<<n<<" по счету\n";
    else
        cout<<"Символ в строке не найден\n";
    return 0;
}
```

Задача 2. Даны две строки. Написать программу, в которой вторая строка дописывается в конец первой строки (произвести слияние двух строк).

Анализ решения задачи. Рассмотрим решение задачи, когда в качестве переменной цикла при обработке строк используем *адрес текущего символа* строки. Указатели p1 и p2 инициализируются адресами первых символов строк (pBegin1, pBegin2). Для перехода к следующим символам значения указателей увеличиваются на 1 – получаем адреса следующих символов (p1++, p2++). Зная адреса, получаем доступ к значениям символов (*p1, *p2). Решение разбивается на три этапа:

1. Поиск адреса ноль-символа первой строки – ведь мы не знаем, сколько символов в первой строке;
2. Начиная с этой позиции каждый символ второй строки, записываем в текущую позицию первой строки.
3. Завершаем первую строку ноль-символом.

Программа будет строиться в виде проекта с одним файлом исходного кода.

Текст функции main() будет выглядеть так:

```
#include<iostream>        //подключение системных средств для
using namespace std;      //возможности использовать потоки ввода-вывода
#include <windows.h>      //прототип функций русификации
int main(void)             //имя функции main()
{
    SetConsoleCP(1251);      //вызов функций русификации
    SetConsoleOutputCP(1251);
    const int N=81;      //задаем максимальный размер первой строки
    char *pBegin1, *pBegin2, *p1, *p2;
    pBegin1=new char[N+1];          //выделение памяти
    pBegin2=new char[N/2+1];              //под строки
    cout<<"Введите две строки:"<<endl;
    cin.getline(pBegin1, N/2);   //используем не более половины строки
```

```
        cin.getline(pBegin2, N/2);
//поиск адреса ноль-символа первой строки
        for(p1=pBegin1; *p1!=0; p1++)
                ;       //пустое тело цикла
//каждый символ второй строки записываем в текущую позицию первой
        for(p2=pBegin2; *p2!=0; p1++,p2++)          //перебор  второй строки
                *p1=*p2;
        *p1=0;          //замыкаем первую строку ноль-символом
        cout<<"Результат:"<<endl;
        cout<<pBegin1<<endl;    //вывод измененной первой строки
        delete[]pBegin1;
        delete[]pBegin2;
        return 0;
}
```

Задание. В качестве упражнения перепишите текст программы, используя в циклах индекс. Сравните тексты. Какой текст вам кажется более понятным, той формой и пользуйтесь.

Порядок выполнения задания

1. Получить задание у преподавателя.
2. По вышеуказанному плану написать программу.
3. Ввести программу в компьютер.
4. Компилируя программу, найти и исправить ошибки синтаксиса.
5. Разработать контрольные примеры.
6. Выполнить программу с исходными данными из контрольных примеров, убедиться в правильности решения задачи.
7. Предъявить правильно работающую программу и контрольные примеры для проверки преподавателю.

Контрольные вопросы и задания

1. Дать определение строки.
2. Как объявить строку статически?
3. Как объявить строку динамически?
4. В чем особенность цикла обработки строки?
5. Какие операции можно произвести над символами?
6. Как обратиться к символу строки по имени?
7. Как обратиться к символу строки по адресу?
8. Напишите цикл подсчета количества символов строки.
9. Объявив строку статически и динамически, обратитесь к символам строки по имени, по адресу при каждом выделении памяти.
10. Для наилучшего освоения темы выполните варианты 4, 9, 11.

Задание

Решить задачу, написав два варианта программы, выделяя память под строку сначала статически, а затем динамически.

Вариант 1. В заданной строке подсчитать количество латинских букв.

Вариант 2. В заданной строке подсчитать количество не цифр.

Вариант 3. В заданной строке подсчитать количество не латинских букв.

Вариант 4. В заданной строке для каждого символа строки вывести следующую информацию: номер, адрес, сам символ и его код.

Вариант 5. Дана строка. Заменить все цифры в строке на пробелы и подсчитать количество маленьких латинских букв.

Вариант 6. Дана строка. Изменить заданную строку, заменив большие латинские буквы маленькими, а маленькие - большими. Посторонние символы не менять.

Вариант 7. Дана строка. Заменить в исходной строке первые n символов на символ, введенный с клавиатуры. Программа должна работать корректно при любом соотношении n с реальной длиной строки.

Вариант 8. Дана строка. Заменить в ней указанный символ другим. Символы ввести с клавиатуры.

Вариант 9. Дана строка. Инвертировать ее, то есть произвести обмен значениями между первым и последним символами строки, вторым и предпоследним и так далее. Дополнительных строк не создавать.

Вариант 10. Дана строка. Заменить n последних символов на один и тот же каждый. Символ ввести с клавиатуры. Программа должна работать корректно при любом соотношении n с реальной длиной строки.

Вариант 11. Дана строка. Вычислить *адрес* символа строки, совпавшего с указанным символом. Если ни один символ из строки не совпал, то вывести сообщение. *Подсказка*: если при выводе адреса на экране появляется часть строки, начиная с найденного символа – это может считаться правильным решением.

Вариант 12. Дана строка. Проверить, является ли она палиндромом. Фраза называется палиндромом, если она читается от конца к началу так же, как от начала к концу при игнорировании всех знаков, кроме букв.

Считать, что в строке все буквы строчные (маленькие) и нет посторонних символов. Например, фраза "аргентинаманитнегра" – палиндром.

Вариант 13. Написать программу, которая анализирует строку, начиная с позиции pos, и вычисляет номер первого найденного пробела. Если пробела нет или значение pos неподходящее – результат равен 0.

Вариант 14. Даны две строки. Создать третью строку, в которой сначала идут символы первой строки, а затем через пробел символы второй строки. Память под строку-результат выделить динамически, сколько необходимо.

Вариант 15. Даны две строки. Определить, равны ли строки (совпадение должно быть буквальным).

Лабораторная работа 3

ОБРАБОТКА ОДНОМЕРНЫХ ЧИСЛОВЫХ МАССИВОВ

Цель работы

Научиться обрабатывать одномерные числовые массивы, в том числе:

– объявлять числовой массив в программе статически и динамически;

– вводить числовой массив с клавиатуры, объявлять с инициализацией или заполнять случайными числами;

– выводить массив на экран монитора;

– писать стандартный цикл обработки числового массива;

– обращаться к числам массива по имени или через адрес;

– знать и уметь применять стандартные алгоритмы обработки массива.

О чем необходимо помнить?

Изучить основные положения теории.

Числовой массив – это набор чисел, идущий в памяти подряд. Как правило, *размер выделенной памяти* совпадает с *реальной длиной массива*. В отличие от строки, с числовым массивом *нельзя* работать как с единым данным. *Любая обработка числового массива – это цикл*!

Память под числовой массив можно выделить статически или динамически.

При статическом выделении памяти необходимо заранее определить и указать с помощью *константы* размер массива:

```
const int SIZE=5;
```

или

```
#define SIZE 5
```

А затем объявить числовой массив:

int mas[SIZE]; //без инициализации

int mas1[SIZE]={2,3,4,-7,0}; //с инициализацией

int mas2[100]={2,3,4}; //остальные числа равны 0

Имя числового массива должно быть «говорящим» о назначении данного.

Не стоит забывать, что объявленный нами числовой массив состоит из SIZE (5) чисел, каждый из которых мы можем анализировать, изменять, подсчитывать и т.д., то есть подвергать определенной обработке. При этом мы должны уметь работать с отдельными числами (элементами массива).

Для этого необходимо помнить, что:

mas – имя числового массива или его *константный* (не изменяемый) *адрес;*

mas+i – *адрес* i-го элемента массива;

*(mas+i) – *значение* i-го элемента массива, вычисляемого через его адрес;

mas[i] – *значение* i-го элемента массива, вычисляемого через его имя,

где 0≤ i< SIZE – *индекс* элемента массива.

При динамическом выделении памяти можно по-разному определить размер массива:

с помощью *константы*

const int SIZE=5;

или

#define SIZE 5

или определяя его в процессе выполнения программы

int size;

cin>> size; //ввод размера с клавиатуры

А затем объявить числовой массив динамически:

int * mas=new int[SIZE];

или объявление указателя и выделение памяти выполнить порознь:

int * mas;

mas=new int[size];

Следует помнить, что в нужном месте программы выделенная память должна быть освобождена:

delete []mas;

Для работы с отдельными элементами массива помним, что:

mas – *адрес* числового массива;

mas+i – *адрес* i-го элемента массива;

*(mas+i) или mas[i] – *значение* i-го элемента массива, вычисляемого через его адрес,

где 0≤ i< SIZE – *индекс* элемента массива.

Обратим внимание на то, что массивы создаются для *одинаковой обработки каждого элемента*. Чтобы организовать одинаковую обработку каждого числа массива, необходимо написать **стандартный цикл обработки числового массива,** который не зависит от способа выделения памяти**:**

```
for (int i=0; i<SIZE; i++)          //SIZE – размер массива
{
  //алгоритм обработки значения i-го элемента массива
}
```

i – индекс текущего элемента массива. Увеличивая индекс на 1 при каждой итерации, перемещаемся к следующему элементу массива.

Цикл может быть применен и к статическому, и к динамическому числовому массиву.

При работе с числами необходимо знать и уметь применять **некоторые стандартные алгоритмы:**

- ✓ *Для вычисления максимального (минимального) значения из ряда чисел необходимо*:

  – объявить переменную соответствующего типа под максимум (минимум) и инициализировать ее первым из чисел (Тип max=ПервоеЧисло;)

  – в процессе перебора чисел, если очередное число окажется больше (меньше) предполагаемого максимума (минимума) заменить предполагаемый максимум (минимум) на очередное число, например:

  if (ОчередноеЧисло>max) max=ОчередноеЧисло;

- ✓ *Для формирования в программе случайного числа необходимо:*

  – подключить заголовочные файлы stdlib.h и time.h

  – *один раз в программе* инициализировать генератор случайных чисел: srand((unsigned)time(NULL));

  – формировать подходящие числа по формулам:

  chislo= rand()%(d +1); //целое число из диапазона [0; d]

  chislo= rand()%(2* d +1) – d; // целое число из диапазона [-d; d]

  chislo= **(**rand()%(2* d +1) – d)/0.9873427; // вещественное число из диапазона [-d; d].

  Задание. Попробуйте сами изобрести формулы для получения случайных чисел из желаемого диапазона [min, max].

Порядок написания программы

1. Выделить память под числовой массив статически или динамически.

2. Ввести числовой массив с клавиатуры или заполнить случайными числами.

3. Провести указанную обработку массива.

4. Вывести на экран исходный массив.
5. Вывести результат обработки.
6. Освободить динамическую память (если она выделялась).

Примеры написания программ

Задача. Написать программу, вычисляющую сумму элементов целочисленного одномерного массива.

При решении задачи выделяем следующие этапы:

1. объявление массива
2. заполнение массива данными
3. вывод массива на экран монитора
4. вычисление суммы элементов целочисленного одномерного массива.
5. вывод результата на экран.

Пункты 2-4 выполняются с помощью цикла.

Программа будет строиться в виде проекта с одним файлом исходного кода.

1 вариант решения – обращение к элементу массива идет через *имя* элемента массива, память выделяется *статически*, массив заполняется *случайными числами*.

Текст функции main() будет выглядеть так:

```
#include<iostream>        //подключение системных средств для
using namespace std;      //возможности использовать потоки ввода-вывода
#include<time.h>          //для формирования случайных
#include<stdlib.h>        //чисел
#include <windows.h>      //прототип функций русификации
int main(void)        //имя функции main()
{
        SetConsoleCP(1251);      //вызов функций русификации
        SetConsoleOutputCP(1251);
        const int SIZE=5;  //задаем размерность массива
        int mas[SIZE];       //объявляем массив статически
        int summa=0;         //переменная под сумму
        int i;
         int k=100;    //диапазон случайных чисел
//инициализируем генератор случайных чисел
        srand((unsigned)time(NULL));
//заполнение массива случайными числами
        for(i=0; i<SIZE; i++)              //перебор элементов массива
                mas[i]=rand()%(2*k+1)-k;  //формируем случайное число [-k;+k]
//вывод элементов массива на экран
        cout<<"\nИсходный массив:\n";
        for(i=0; i<SIZE; i++)              //перебор элементов массива
```

```
            cout<< mas[i]<<endl;      //выводим  i-ый элемент  массива
//вычисление суммы
        for(i=0; i<SIZE; i++)              //перебор элементов массива
            summa+=mas[i];   //увеличиваем сумму на i-ый элемент массива
        cout<< "\nСумма равна "<<summa<<endl;
        return 0;
}
```

2 вариант решения – обращение к элементу массива идет через *адрес* элемента массива, память выделяется *динамически*, заполняется массив *с клавиатуры*.

Текст функции main() будет выглядеть так:

```
#include<iostream>         //подключение системных средств для
using namespace std;        //возможности использовать потоки ввода-вывода
#include <windows.h>     //прототип функций русификации
int main(void)               //имя функции main
{
        SetConsoleCP(1251);      //вызов функций русификации
        SetConsoleOutputCP(1251);
        const int SIZE=5;  //задаем размерность массива
        int *mas= new int[SIZE];//выделяем память динамически под массив
        int summa=0;        //переменная под сумму
        int i;
//заполнение массива с клавиатуры
        cout<<"Введите массив:\n";
        for(i=0; i<SIZE; i++)      //перебор элементов массива
            cin>>*(mas+i);
//вывод элементов массива на экран
        cout<<"\nИсходный массив:\n";
        for(i=0; i<SIZE; i++)      //перебор элементов массива
            cout<< *(mas+i)<<endl;  //выводим  i-ый элемент массива
//вычисление суммы
        for(i=0; i<SIZE; i++)      // перебор элементов массива
            summa+=mas[i];  //другая форма обращения к элементу
        cout<< "\nСумма равна "<<summa<<endl;
        delete[]mas;          //освобождаем динамическую память
        return 0;
}
```

Порядок выполнения задания

1. Получить задание у преподавателя.
2. По вышеуказанному плану написать программу.
3. Ввести программу в компьютер.
4. Компилируя программу, найти и исправить ошибки синтаксиса.

5. Разработать достаточное количество контрольных примеров.

6. Выполнить программу с исходными данными из контрольных примеров, убедиться в правильности решения задачи.

7. Предъявить правильно работающую программу и контрольные примеры для проверки преподавателю.

Контрольные вопросы и задания

1. Дать определение одномерного числового массива.

2. Как объявить одномерный числовой массив статически с инициализацией?

3. Как объявить одномерный числовой массив динамически?

4. Напишите стандартный цикл обработки одномерного числового массива.

5. Чем он отличается от перебора строки и почему?

6. Как обратиться к элементу статического массива по имени?

7. Как обратиться к элементу динамического массива по адресу?

8. Напишите цикл перебора элементов массива от конца к началу.

9. Объявите одномерный числовой массив статически, динамически. Обратитесь к элементам массива по имени, по адресу при каждом выделении памяти.

10. Для наилучшего освоения темы выполните варианты 4, 11, 15.

Задание

Решить задачу, написав два варианта программы:

1. выделить память под массив статически и заполнить его с клавиатуры;
2. выделить память динамически (размер введите с клавиатуры) и заполнить его случайными числами.

Вариант 1. Дан одномерный целочисленный массив. Вычислить сумму всех элементов массива, кратных некоторому числу.

Вариант 2. Дан одномерный числовой массив. Вычислить произведение ненулевых элементов массива с *номера* n до *номера* k.

Вариант 3. Дан одномерный числовой массив. Вычислить сумму и количество всех отрицательных элементов массива.

Вариант 4. Дан одномерный числовой массив. Вычислить минимум и номер минимального элемента в массиве.

Вариант 5. Дан одномерный целочисленный массив. Вычислить количество элементов массива, кратных некоторому числу.

Вариант 6. Дан одномерный числовой массив. Вычислить сумму элементов массива с четными *номерами*.

Вариант 7. Дан одномерный числовой массив. Вычислить количество элементов массива, *значение* которых попадает в интервал [a,b].

Вариант 8. Дан одномерный числовой массив. Вычислить произведение ненулевых элементов массива с нечетными *номерами*.

Вариант 9. Дан одномерный числовой массив. Вычислить в какой позиции массива *впервые* встретилось число со значением a.

Вариант 10. Дан одномерный числовой массив. Вычислить среднее арифметическое отрицательных элементов массива.

Вариант 11. Дан одномерный числовой массив. Вычислить произведение, в состав которого войдет ненулевое значение каждого *третьего* элемента массива.

Вариант 12. Дан одномерный числовой массив. Вычислить максимум и *номер* максимального элемента в массиве.

Вариант 13. Дан одномерный числовой массив. Определить, является ли массив арифметической прогрессией.

Вариант 14. Дан одномерный числовой массив. Увеличить значение каждого элемента массива на k и вывести измененный массив на экран монитора.

Вариант 15. Дан одномерный целочисленный массив. Все значения массива, кратные 3 заменить значением, кратным 5 с той же степенью кратности и вывести измененный массив на экран монитора.

Лабораторная работа 4

ОБРАБОТКА ДВУМЕРНЫХ ЧИСЛОВЫХ МАССИВОВ (НА ПРИМЕРЕ МАТРИЦ)

Цель работы

Научиться обрабатывать матицу, в том числе:
– объявлять матрицу в программе статически и динамически;

– вводить матрицу с клавиатуры, объявлять с инициализацией или заполнять случайными числами;

– выводить матрицу на экран монитора ровными столбцами и строками;

– писать стандартные циклы обработки матрицы;

– обращаться к элементам матрицы по имени и через адрес;

– знать и уметь применять стандартные алгоритмы обработки матрицы.

О чем необходимо помнить?

Изучить основные положения теории.

Матрица – это набор чисел, идущий в памяти подряд, *построчно*. Память под матрицу можно выделить статически или динамически.

При статическом выделении памяти необходимо заранее определить и указать с помощью *констант* количество строк и столбцов матрицы:

```
const int STR=2, STLB=3;
```

или

```
#define STR 2
#define STLB 3
```

А затем объявить матрицу:

```
int A[STR][STLB];          //без инициализации
int B[STR][STLB]={2,3,4,-7,0,-5};      //с инициализацией построчно
int C[STR][STLB]={2,3};          //с инициализацией построчно, остальные – 0.
```

Имя матрицы должно быть «говорящим» о назначении данного.

Не стоит забывать, что объявленная нами матрица состоит из набора чисел, каждый из которых мы можем анализировать, изменять, подсчитывать и т.д., то есть подвергать определенной обработке. При этом мы должны уметь работать с отдельными числами (элементами) матрицы.

Для этого необходимо помнить, что:

A[i] –*константный* (не изменяемый) *адрес* i-ой строки матрицы*;*

A[i]+j – *адрес* элемента матрицы из i-ой строки и j-го столбца;

*(A[i]+j) – *значение* элемента матрицы из i-ой строки и j-го столбца, вычисляемого через *адрес* элемента матрицы;

A[i][j] – *значение* элемента матрицы из i-ой строки и j-го столбца, вычисляемого через *имя* матрицы,

где $0 \le i < STR$, $0 \le j < STLB$ – *индексы* строк и столбцов.

Память динамически выделим как под одномерный массив. Помним, что элементы матрицы лежат в памяти построчно, друг за другом, как в одномерном массиве. Количество строк и столбцов матрицы укажем с помощью *констант*:

```
const int STR=2, STLB=3;
```
или
```
#define STR 2
#define STLB 3
```
или с помощью переменных:
```
int str, stlb;
cin>>str>>stlb;                 //ввод размерности матрицы с клавиатуры
```
А затем объявим матрицу динамически:
```
int * matr=new int[STR*STLB];
```
или объявление указателя и выделение памяти выполняем порознь:
```
int * matr;
matr=new int[str*stlb];
```
Следует помнить, что в нужном месте программы выделенная память должна быть освобождена:
```
delete []matr;
```
Для работы с отдельными числами матрицы помним, что:

matr –*адрес* матрицы*;*

matr+i*STLB+j– *адрес* элемента матрицы из i-ой строки и j-го столбца;

*(matr+i*STLB+j) или matr[i*STLB+j]– *значение* элемента матрицы из i-ой строки и j-го столбца, вычисляемое через адрес,

где 0≤i<STR, 0≤j<STLB – *индексы* строк и столбцов.

Обратим внимание на то, что матрицы создаются для *одинаковой обработки каждого элемента*. Чтобы организовать одинаковую обработку каждого числа матрицы, необходимо написать один из **циклов обработки матрицы:**
```
int i, j;
```
Для обработки одной строки с номером N:
```
i=N-1;          //вычисляем индекс обрабатываемой строки
for(j=0; j<STLB; j++)       //цикл перебора одной строки
{
// алгоритм обработки элемента матрицы из i-ой строки и j-го столбца
}
```
Для обработки всей матрицы построчно:
```
for(i=0; i<STR; i++)
{
  //возможные действия в начале каждой строки
        for(j=0; j<STLB; j++)
        {
        //алгоритм обработки элемента матрицы из i-ой строки и j-го
        //столбца
        }
```

```
  //возможные действия в конце каждой строки
}
```

Для обработки одного столбца с номером M:

```
j=M-1;          //вычисляем индекс обрабатываемого столбца
for(i=0; i<STR; i++)                //цикл перебора одного столбца
{
// алгоритм обработки элемента матрицы из i-ой строки и j-го столбца
}
```

Для обработки всей матрицы по столбцам:

```
for(j=0; j<STLB; j++)
{
  //возможные действия в начале каждого столбца
        for(i=0; i<STR; i++)
        {
        //алгоритм обработки элемента матрицы из i-ой строки и j-го
        //столбца
        }
  //возможные действия в конце каждого столбца
}
```

Циклы могут быть применены и к статической, и к динамической матрице.

Порядок написания программы

1. Выделить память под матрицу статически или динамически.
2. Ввести матрицу с клавиатуры или заполнить случайными числами.
3. Провести указанную обработку матрицы.
4. Вывести на экран матрицу в матричном виде.
5. Вывести результат обработки.
6. Освободить динамическую память (если она выделялась).

Примеры написания программ

Задача. Вычислить количество положительных элементов произвольного столбца целочисленной матрицы.

Этапы решения задачи:

1. объявить матрицу;
2. заполнить матрицу числами;
3. вывести матрицу в матричном виде. Для этого задаем ширину вывода каждого числа (setw(8)) – например, восемь символов. Более короткие числа по умолчанию прижимаются вправо, при этом столбец выравнивается по правому краю;

4. вычислить количество положительных элементов произвольного столбца матрицы;

5. вывести результат.

Пункты 2-4 выполняются с помощью соответствующих циклов.

Программа будет строиться в виде проекта с одним файлом исходного кода.

1 вариант решения – обращение к элементу массива идет через *имя* элемента массива, память выделяется *статически*, массив заполняется с *клавиатуры*.

Текст функции main() будет выглядеть так:

```
#include<iostream>        //подключение системных средств для
using namespace std;      //возможности использовать потоки ввода-вывода
#include<iomanip>         //для форматированного вывода матрицы
#include <windows.h>      //прототип функций русификации
int main(void)            //имя функции main()
{
      SetConsoleCP(1251);      //вызов функций русификации
      SetConsoleOutputCP(1251);
      const int STR=3, STLB=4;       //задаем размерность матрицы
      int matr[STR][STLB];     //объявляем матрицу статически
      int kol=0;               //переменная под количество
      int i, j, nstlb;
//заполнение матрицы числами с клавиатуры
      cout<<"\nВведите матрицу:\n";
      for(i=0; i<STR; i++)       //перебор элементов матрицы построчно
            for(j=0; j<STLB; j++)
                  cin>>matr[i][j];
//вывод матрицы
      cout<<"\nИсходная матрица:\n";
      for(i=0; i<STR; i++)       //перебор элементов матрицы построчно
      {
         for(j=0; j<STLB; j++)
            cout<<setw(8)<< matr[i][j];      //выводим элемент матрицы
         cout<<endl;             //завершим вывод очередной строки
      }
 //вычисление количества положительных элементов произвольного
 //столбца матрицы
      cout<<"Введите номер обрабатываемого столбца: ";
      cin>>nstlb;
      j=nstlb-1;              //определяем индекс обрабатываемого столбца
      for(i=0; i<STR; i++)       // перебор строк одного столбца с индексом j
            if(matr[i][j]>0)     // если элемент матрицы положительный
                  kol++;         //количество увеличиваем на 1
```

```
        cout<< "\nКоличество положительных элементов в столбце "
        <<nstlb<<" равно  " << kol<<endl;
        return 0;
}
```

2 вариант решения – обращение к элементу массива идет через *адрес* элемента массива, память выделяется *динамически*, заполняется массив *с клавиатуры*.

Текст функции main() будет выглядеть так:

```
#include<iostream>        //подключение системных средств для
using namespace std;      //возможности использовать потоки ввода-вывода
#include<iomanip>         //для форматированного вывода матрицы
#include <windows.h>      //прототип функций русификации
int main(void)            //имя функции main()
{
        SetConsoleCP(1251);      //вызов функций русификации
        SetConsoleOutputCP(1251);
        int str,stlb,nstlb;
        cout<<"Введите количество строк и столбцов в матрице: ";
        cin>>str>>stlb;      //задаем количество строк и столбцов в матрице
        int *matr=new int[str*stlb];       //объявляем матрицу динамически
        int kol=0;     //переменная под количество
        int i, j;
        int *p=matr;  //адрес первого элемента в матрице
//заполнения матрицы  числами с клавиатуры
        cout<<"Введите матрицу\n";
        for(i=0; i<str; i++)          //перебор элементов матрицы  построчно
                for(j=0; j<stlb; j++)
                        cin>>*( p+i*stlb+j);        //или  p[i*stlb+j]
//вывод матрицы
        cout<<"\nИсходная матрица:\n";
        for(i=0; i<str; i++)          //перебор элементов матрицы построчно
        {
                for(j=0; j<stlb; j++)
                        cout<<setw(8)<<*(p+i*stlb+j);          //или  p[i*stlb+j]
                cout<<endl;          //завершаем вывод строки матрицы
        }
//вычисления количества положительных элементов произвольного
//столбца матрицы
        cout<<"Введите номер анализируемого столбца: ";
        cin>>nstlb;  //задаем номер анализируемого столбца
        j=nstlb-1;     //определяем индекс столбца
        for(i=0; i<str; i++)            //перебор строк одного столбца с индексом j
                if(*(p+i*stlb+j)>0)  // если элемент массива положительный
```

```
                kol++;          //количество увеличиваем на 1
  cout<<"Количество положительных элементов в столбце "<<nstlb<<
  " равно "<<kol<<endl;
  delete[]matr;
  return 0;
  }
```

Задание. Перепишите тексты программ, заполняя матрицу случайными числами.

Порядок выполнения задания

1. Получить задание у преподавателя.
2. По вышеуказанному плану написать программу.
3. Ввести программу в компьютер.
4. Компилируя программу, найти и исправить ошибки синтаксиса.
5. Разработать достаточное количество контрольных примеров.
6. Выполнить программу с исходными данными из контрольных примеров, убедиться в правильности решения задачи.
7. Предъявить правильно работающую программу и контрольные примеры для проверки преподавателю.

Контрольные вопросы и задания

1. Дать определение матрицы.
2. Как объявить матрицу статически?
3. Как объявить матрицу динамически?
4. Напишите цикл обработки матрицы построчно.
5. Напишите цикл обработки матрицы по столбцам.
6. Напишите циклы обработки одной строки, одного столбца матрицы.
7. Как обратиться к статическому элементу матрицы по имени?
8. Как обратиться к динамическому элементу матрицы по адресу?
9. Объявите матрицу статически. Определите адрес каждой строки матрицы, адрес и значение каждого элемента матрицы.
10. Для наилучшего освоения темы выполните варианты 1, 7, 12.

Задание

Решить задачу, написав два варианта программы:

1. выделить память для матрицы динамически и заполнить ее случайными числами;
2. выделить память для матрицы статически и заполнить ее с клавиатуры;

Вариант 1. Дана матрица. Вывести ее в матричной форме. Вычислить количество элементов матрицы, попадающих в интервал [a,b].

Вариант 2. Дана матрица. Вывести ее в матричной форме. Вычислить произведение ненулевых элементов матрицы в указанном столбце.

Вариант 3. Дана матрица. Вывести ее в матричной форме. Уменьшить значение каждого элемента матрицы на a.

Вариант 4. Дана матрица. Вывести ее в матричной форме. Вычислить количество элементов, кратных числу k в столбце с номером M.

Вариант 5. Дана квадратная матрица. Вывести ее в матричной форме. Заменить элементы главной диагонали числом k.

Вариант 6. Дана матрица. Вывести ее в матричной форме. Вычислить максимум в указанной строке матрицы.

Вариант 7. Дана матрица. Вывести ее в матричной форме. Определить, содержится ли число 5 в каждом столбце матрицы в отдельности.

Подсказка: вы получите тот же результат, если просто подсчитаете количество чисел, равных 5, в каждом столбце матрицы.

Вариант 8. Дана матрица. Вывести ее в матричной форме. Определить, равен ли первый и последний элемент в каждой строке матрицы в отдельности.

Вариант 9. Дана матрица. Вывести ее в матричной форме. Вычислить минимум в столбце с номером M.

Вариант 10. Дана матрица. Вывести ее в матричной форме. Определить, равны ли между собой все числа в строке матрицы с номером N.

Подсказка: вы получите тот же результат, если просто подсчитаете количество чисел, равных первому элементу строки N.

Вариант 11. Дана матрица. Вывести ее в матричной форме. Вычислить количество положительных чисел в каждом столбце матрицы. Вы получите дополнительный балл, если результаты поместите в одномерный массив (но это необязательно).

Вариант 12. Дана матрица. Вывести ее в матричной форме. Вычислить произведение чисел в каждой строке матрицы. Вы получите дополнительный балл, если результаты поместите в одномерный массив (но это необязательно).

Вариант 13. Дана матрица. Вывести ее в матричной форме. Вычислить среднее арифметическое в каждой строке матрицы. Вы получите до-

полнительный балл, если результаты поместите в одномерный массив (но это необязательно).

Вариант 14. Дана матрица. Заполнить две строки случайными числами, а каждую последующую как сумму соответствующих элементов двух предыдущих. Вывести матрицу на экран.

Вариант 15. Дана квадратная матрица. Вывести ее в матричной форме. Вычислить сумму элементов матрицы, расположенных выше главной диагонали матрицы.

Подсказка: проанализируйте соотношения индексов строки и столбца у элементов, расположенных выше главной диагонали.

3. ФУНКЦИИ ЯЗЫКА С++

Любая нетривиальная задача требует разделения ее на несколько логически завершенных частей, каждая из которых реализуется определенной функцией.

Наличие функций избавляет нас от повторного программирования. Небольшую функцию легче понять, отладить и модифицировать.

Функция main() при этом строится как взаимодействие этих (*написанных нами, пользовательских*) функций. Программа с функциями упрощается за счет сокрытия деталей реализации, имеет более четкую структуру.

Функция – это независимая и самодостаточная единица программы, спроектированная для решения *одной* подзадачи. Функция содержит *алгоритм*, который указывает, как из набора исходных данных получить требуемые результаты.

Функция должна выполняться для множества задач из класса однотипных. Это достигается наличием *параметров* и *возвращаемыми в точку вызова результатами*.

Связанные между собой функции могут быть помещены в отдельный файл (модуль), компилируемый независимо от остального кода. Это помогает отлаживать программу по частям или разными программистами. Готовый модуль может использоваться неоднократно в разных проектах, без каких-либо изменений.

Любая функция должна быть *определена* и *объявлена* до ее *вызова*.

Определение функции состоит из заголовка и тела функции.

Заголовок включает в себя необходимые для функции директивы препроцессора и имя функции.

Имя функции состоит из *типа результата функции (тип функции)*, передаваемого из функции с помощью оператора return, *названия функции* и *списка параметров функции*, заключенные в круглые скобки.

Если функция ничего не вычисляет и не передает, тип у функции – void.

Параметры функции – это переменные, ожидающие *исходные данные* извне (из вызывающей функции). Имя *каждого* параметра пишется вместе со своим типом, через запятую.

Тело функции заключается в фигурные скобки и пишется аналогично функции main():

- объявляется переменная под результат, передаваемый стандартным образом (с помощью оператора return);
- объявляются вспомогательные (локальные) переменные;
- пишутся операторы, с помощью которых получаем результат. Для этого используем параметры функции как известные величины;
- завершается функция оператором return, который *прекращает выполнение функции и передает результат* в точку вызова функции. Оператор return может быть написан и в другом месте функции в составе, например, условного оператора.

Чтобы инициировать работу функции необходимо написать **оператор вызова функции**. Он может появиться внутри функции main(), внутри другой функции или внутри вызываемой функции (если функция рекурсивная).

Оператор вызова функции может писаться как самостоятельный или внутри выражения (если функция передает соответствующий результат).

Схематично *оператор вызова* можно изобразить так:

ИмяПеременнойРезультата = НазваниеФункции (ИмяАргумента1, ИмяАргумента2, …);

Если тип результата функции void, то так:

НазваниеФункции (ИмяАргумента1, ИмяАргумента2, …);

Аргументы функции – это те конкретные значения, которые инициализируют *параметры* для дальнейшего выполнения функции и получения ее результатов. Аргументы могут быть выражены константами, переменными или выражениями.

Между параметрами и аргументами теснейшая связь. Аргументы должны соответствовать параметрам:

- по типу;
- по количеству;
- по порядку следования;
- по смыслу.

Помним, что компилятор проверяет правильность написания каждого оператора. Чтобы проверить правильность написания оператора вызова функции, компилятору необходимо знать *характеристики функции*. Если определение функции находится в том же файле выше точки вызова – то

все в порядке, но, как правило, это не так. Поэтому *до вызова* должно находиться **объявление функции (или прототип функции)**.

Объявление в *точности повторяет имя функции* и завершается точкой с запятой. Обычно объявления функций находятся в заголовочных файлах, соответствующим образом оформленных: стандартные функции – в стандартных заголовочных файлах, пользовательские функции – в пользовательских заголовочных файлах. Это полностью соответствует принципам структурной и модульной парадигмы программирования.

Важно! ***Функции пишутся один раз для многократного использования***. Поэтому *строго* соблюдайте правила написания функции:

- функция решает *одну* подзадачу;
- все *исходные данные* в функцию поступают через *параметры*;
- все *результаты передаются* в точку вызова.

Игнорирование этих правил превращает функцию в кусок текста, снабженный заголовком, и нарушает принципы структурной и модульной парадигмы программирования.

Лабораторная работа 5

ФУНКЦИИ С ОДНИМ РЕЗУЛЬТАТОМ

Цель работы

Научиться писать функции с одним результатом на основе простых исходных данных, в том числе:

– писать программу, состоящую из нескольких функций;

– уметь определить, объявить и вызвать функцию указанного типа;

– уметь размещать текст программы в файлах проекта в соответствии с парадигмой модульного программирования.

О чем необходимо помнить?

Изучить основные положения теории.

Функции могут быть самые разные – принимать различные исходные данные и вычислять необходимое количество результатов.

Рассмотрим построение простейшей функции, вычисляющей *один* результат, возвращаемый в точку вызова *стандартным путем* с помощью оператора return, на основе простых исходных данных.

Переменная-аргумент имеет *две важные характеристики*: текущее значение и адрес.

Передача в функцию ***текущего значения аргумента*** *используется, если внутри функции требуется анализировать текущее значение аргумента (с помощью операторов if, switch и циклов) или использовать его при вычислениях по формулам.*

Порядок написания программы

1. Внимательно прочитать условие задачи и формулировку функции.
2. Провести анализ характеристик функции:

Дать название функции;

Выделить список исходных данных функции, определить их типы;

Определить тип результата функции, передаваемого с помощью оператора return.

3. Написать текст функции пользователя, используя параметры как известные величины и передав результат с помощью оператора return.
4. Написать прототип функции.
5. Написать функцию main() с вызовом функции пользователя:

Объявить аргументы и переменную для результата функции;

Определить (ввести с клавиатуры) аргументы;

Написать оператор вызова функции и принять результат функции в переменную;

Вывести результат функции на экран монитора.

6. Создать проект из двух файлов исходного кода с расширением cpp (для функции пользователя и main()), и одного заголовочного файла с расширением h с прототипом функции пользователя.
7. Выполнить проект.

Примеры написания программ

Задача 1. Написать программу с функцией, вычисляющую площадь произвольного треугольника.

Задача демонстрирует передачу в функцию текущих значений аргумента для вычисления по формуле.

Анализ решения задачи. Определим основные характеристики функции, отвечая на следующие вопросы:

- ✓ Придумайте имя функции: AreaOfTriangle
- ✓ Определите список параметров: придумайте имена параметров, определите их типы и порядок их следования. Следует помнить, что параметры функции – это ее исходные данные. В нашей задаче: для определения площади треугольника следует знать его стороны – значит, три параметра. Стороны – это длины, длина может выражаться и дробным, и целым числом, выбираем типы параметров – double. Имена параметров и порядок следования – a, b, c.
- ✓ Определить тип функции, который выражается типом результата: результат площадь, она выражается дробным числом, значит тип функции – double.

Помним, что все имена (функции, параметров, аргументов) должны быть «говорящими». Это способствует лучшему пониманию программы.

После этого мы можем написать определение функции, ее объявление и функцию main() с вызовом функции AreaOfTriangle().

Программа будет строиться в виде проекта из трех файлов:

- ✓ AreaOfTriangle.cpp – определение функции пользователя;
- ✓ AreaOfTriangle.h – объявление функции пользователя;
- ✓ Main.cpp – текст функции main().

Текст файла AreaOfTriangle.cpp будет выглядеть так:

```
#include<math.h>          //математические функции
double AreaOfTriangle (double a, double b, double c)
{
      double area,p;      //результат и вспомогательная переменная
      p=(a+b+c)/2;
      area=sqrt(p*(p-a)*(p-b)*(p-c)); //параметры считаем известными
                                      //величинами
      return area;  //завершаем функцию и передаем результат из area
}
```

Текст файла AreaOfTriangle.h будет выглядеть так:

```
#ifndef  AREAOFTRIANGLE_H      //оформление заголовочного файла
#define  AREAOFTRIANGLE_H
double AreaOfTriangle (double a, double b, double c);     //прототип функции
#endif
```

Текст файла Main.cpp будет выглядеть так:

```
#include<iostream>        //подключение системных средств для
using namespace std;      //возможности использовать потоки ввода-вывода
#include <windows.h>    //прототип функций русификации
#include "AreaOfTriangle.h"    //прототип пользовательской функции
                                //ставьте свой #include в конец списка
int main(void)             //имя функции main()
{
      SetConsoleCP(1251);     //вызов функций русификации
      SetConsoleOutputCP(1251);
      double a,b,c,S;      //аргументы и результат функции, могут быть
                           //названы произвольно (не как параметры)
      cout<<"\nВведите стороны треугольника: ";
      cin>>a>>b>>c;      //определяем аргументы
      S=AreaOfTriangle(a,b,c);          //вызов функции
      cout<< "Площадь треугольника равна  "<< S<<endl;
      return 0;
}
```

Задача 2. Написать программу с функцией, вычисляющую количество цифр в целом числе.

Задача демонстрирует передачу в функцию текущего значения аргумента для его анализа (с помощью операторов if, switch и циклов).

Анализ решения задачи. Определим основные характеристики функции, отвечая на следующие вопросы:

- ✓ Придумайте имя функции: QuantityOfNumeral
- ✓ Определите список параметров: цель функции – вычислить количество цифр в целом числе, значит, необходимо знать это целое число. Параметр функции один, целого типа: int n.
- ✓ Определить тип функции, который выражается типом результата: результат количество – целый тип int.

Помним, что все имена (функции, параметров, аргументов) должны быть «говорящими». Это способствует лучшему пониманию программы.

После этого мы можем написать определение функции, ее объявление и функцию main() с вызовом функции QuantityOfNumeral().

Программа будет строиться в виде проекта из трех файлов:

- ✓ QuantityOfNumeral.cpp – определение функции пользователя;
- ✓ QuantityOfNumeral.h – объявление функции пользователя;
- ✓ Main.cpp – текст функции main().

Текст файла QuantityOfNumeral.cpp будет выглядеть так:

```
int QuantityOfNumeral (int n)
{
        int kol=0;     // объявили переменную под результат
        if(n==0) return 1;   //особое решение
        else if(n<0) n*=-1;          //особое решение
        while(n>0)
        {
                n/=10;
                kol++;
        }
        return kol;    //завершаем функцию и передаем результат из kol
}
```

Текст файла QuantityOfNumeral.h будет выглядеть так:

```
#ifndef  QUANTITYOFNUMERAL _H       //оформление заголовочного
#define  QUANTITYOFNUMERAL _H       // файла
int QuantityOfNumeral (int n);           //прототип функции
#endif
```

Текст файла Main.cpp будет выглядеть так:

```
#include<iostream>         //подключение системных средств для
using namespace std;       //возможности использовать потоки ввода-вывода
#include <windows.h>      //прототип функций русификации
#include "QuantityOfNumeral.h "       //прототип пользовательской функции
```

```
int main(void)              //имя функции main()
{
        SetConsoleCP(1251);      //вызов функций русификации
        SetConsoleOutputCP(1251);
        int kol,chislo;          //аргумент и результат функции, могут быть
                                 //названы произвольно (не как параметры)
        cout<<"\nВведите целое число: ";
        cin>>chislo;             //определяем аргумент
        kol= QuantityOfNumeral (chislo);        //вызов функции
        cout<< "Количество цифр равно: "<<kol<<endl;
        return 0;
}
```

Порядок выполнения задания

1. Получить задание у преподавателя.
2. По вышеуказанному плану написать программу.
3. Ввести программу в компьютер.
4. Компилируя программу, найти и исправить ошибки синтаксиса.
5. Выполнить программу с исходными данными из контрольного примера, убедиться в правильности решения задачи.
6. Предъявить правильно работающую программу и контрольный пример для проверки преподавателю.

Контрольные вопросы и задания

1. По каким правилам пишется текст функции пользователя?
2. Какую роль для функции играют параметры?
3. Как и когда конкретные значения попадают в параметры?
4. Можно ли внутри функции пользователя объявлять переменные? Для каких целей?
5. Как определить, какие данные объявлять в параметрах, а какие в теле функции?
6. Как пишется прототип функции? Где в программе его следует размещать, и для чего он нужен?
7. Как и где пишется оператор вызова функции? Что такое аргументы функции, их назначение, как и где они пишутся?
8. Должны ли иметь одинаковые имена соответствующие параметр и аргумент? Как связаны между собой эти понятия?
9. Напишите прототипы и операторы вызова функций для нескольких вариантов из задания.
10. Для наилучшего освоения темы выполните варианты 2, 4, 15.

Задание

Написать программу с функцией пользователя с передачей в нее текущих значений аргументов. Вызвать функцию дважды, используя аргументы-переменные и аргументы-константы.

Вариант 1. Написать программу с функцией, вычисляющей расстояние между двумя точками на плоскости.

Вариант 2. Написать программу с функцией, определяющей, является ли символ латинской буквой.

Вариант 3. Написать программу с функцией, формирующей одно случайное число из диапазона [min, max]. В main() вызвать эту функцию заданное количество раз.

Вариант 4. Известны координаты четырех точек на плоскости, представляющие собой вершины выпуклого четырехугольника. Определить, можно ли вписать в него окружность (суммы длин противоположных сторон для этого должны быть равны). Программа должна содержать одну функцию, вычисляющую расстояние между двумя точками на плоскости.

Вариант 5. Написать программу с функцией, округляющей дробное число до n знаков после запятой.

Вариант 6. Написать программу с функцией, определяющей, является ли год високосным. Год високосный, если делится без остатка на 4. Если год делится без остатка на 100, то високосный год лишь тот, что делится и на 400. Примеры: високосные года – 1600, 2000, 2016. Не високосные года – 2100, 2200, 2015.

Вариант 7. Написать функцию, вычисляющую сумму делителей натурального числа, начиная с 1 и исключая само число. В main(), вызывая эту функцию, доказать, что 220 и 284 – дружественные числа. Два числа называются дружественными, если сумма делителей одного числа равна другому числу и наоборот.

Вариант 8. Написать программу с функцией, вычисляющей корень линейного уравнения.

Вариант 9. Написать программу с функцией, вычисляющей число Фибоначчи с номером n.Числа вычисляются по формуле $F_{n+2}=F_{n+1}+F_n$, где $n\geq 0$ и $F_0=0$, $F_1=1$.

Вариант 10. Написать программу с функцией, определяющей максимальную цифру в целом числе.

Вариант 11. Написать программу с функцией, вычисляющей наибольший общий делитель двух натуральных чисел.

Вариант 12. Написать функцию, вычисляющую сумму делителей натурального числа, начиная с 1 и исключая само число. В main(), вызывая эту функцию, найти совершенные числа в первой сотне. Число называется совершенным, если оно равно сумме своих делителей.

Вариант 13. Для создания финансовой пирамиды ее организатор привлекает N участников, каждый из которых в свою очередь также привлекает по N участников.

Написать программу с функцией, определяющей число уровней пирамиды, если в нее вовлечено все население города из M человек.

Вариант 14. Написать программу с функцией, вычисляющей *целую степень* дробного числа. Учесть, что степень может быть положительной, отрицательной, нулевой.

Вариант 15. Дано натуральное число N. Написать программу с функцией, которая меняет порядок цифр в числе на обратный. Например: N=265, результат – 562.

Лабораторная работа 6

ФУНКЦИИ С НЕСКОЛЬКИМИ РЕЗУЛЬТАТАМИ

Цель работы

Научиться писать функции с несколькими результатами на основе простых исходных данных, в том числе:

– писать программу, состоящую из нескольких функций;

– уметь определить, объявить и вызвать функцию указанного типа;

– уметь размещать текст программы в файлах проекта в соответствии с парадигмой модульного программирования.

О чем необходимо помнить?

Изучить основные положения теории. Повторить тему «Указатели».

Рассмотрим построение функции, вычисляющей на основе простых исходных данных *несколько* результатов, один из которых возвращается в точку вызова *стандартным путем* с помощью оператора return, а остальные – через *параметры-указатели или через параметры-ссылки*.

Переменная-аргумент имеет две важные характеристики: текущее значение и адрес.

Передача в функцию ***адреса аргумента*** *используется, если внутри функции пользователя требуется* ***изменить*** *текущее значение аргумента.*

Этот механизм используется, в частности, для передачи из функции дополнительных результатов.

*Адрес аргумента можно передавать **явно**, используя параметры-указатели и **неявно**, используя параметры-ссылки.*

Напишем **схематично функцию, в которой адрес аргумента-результата передается явно через параметр-указатель (схема 1)**:

```
…Func(…, ТипРезультата*Rezult)     //параметр-указатель
{
        …
        *Rezult=…;          //вычисление и передача результата
        …
        return …;
}
int main(void)
{
        ТипРезультата  MyRezult; //объявим переменную под результат
        …
        …Func(…, &MyRezult); //передаем адрес аргумента-результата
        cout<<MyRezult<<endl;
        return 0;
}
```

Напишем **схематично функцию, в которой адрес аргумента-результата передается неявно через параметр-ссылку (схема 2)**. Параметр-*ссылка* инициализируется аргументом (при вызове функции) и становится *псевдонимом аргумента*:

```
…Func(…, ТипРезультата&Rezult)    //параметр-ссылка
{
        …
        Rezult=…;          //вычисление и передача результата
        …
        return …;
}
int main(void)
{
        ТипРезультата  MyRezult; //объявим переменную под результат
        …
        …Func(…, MyRezult);   //передаем имя аргумента-результата
        cout<<MyRezult<<endl;
        return 0;
}
```

Обе схемы ведут к одному результату, но использование ссылки проще (спорное, субъективное мнение).

Порядок написания программы

1. Внимательно прочитать условие задачи и формулировку функции.
2. Провести анализ характеристик функции:

Дать название функции;

Выделить список исходных данных функции, определить их типы;

Определить тип результата функции, передаваемого с помощью оператора return. *Если функция не всегда выполняется успешно, через return передается сигнал типа bool о результате ее выполнения (true, false).*

Определить тип результатов функции, передаваемых с помощью параметров-указателей или параметров-ссылок.

3. Написать текст функции пользователя.
4. Написать прототип функции.
5. Написать функцию main() с вызовом функции пользователя.
6. Создать проект из двух файлов исходного кода с расширением cpp (для функции пользователя и main()), и одного заголовочного файла с расширением h с прототипом функции пользователя.
7. Выполнить проект.

Примеры написания программ

Задача. Написать программу с функцией, вычисляющей корни квадратного уравнения.

Анализ решения задачи. Функция, вычисляющая корни квадратного уравнения, не всегда может быть завершена успешно: известно, что при отрицательном дискриминанте корни не могут быть вычислены.

Поэтому функция имеет три результата: сигнал об успешном (неуспешном) ее завершении и значения двух корней.

Обычно *сигнал передаем с помощью оператора return (запомните это, как правило)*, а значения корней – через параметры.

Определим основные характеристики функции, отвечая на следующие вопросы:

1. Название функции – Kvadr
2. Исходные данные – коэффициенты a,b,c, их тип double
3. Результаты:

через return передаем результат типа bool (успешна ли функция?)

через параметры передаем корни типа double.

Помним, что все имена (функции, параметров, аргументов) должны быть «говорящими». Это способствует лучшему пониманию программы.

После этого мы можем написать определение функции, ее объявление и функцию main() с вызовом функции Kvadr().

Программа будет строиться в виде проекта из трех файлов:

- ✓ Kvadr.cpp – определение функции пользователя;
- ✓ Kvadr.h – объявление функции пользователя;

- ✓ Main.cpp – текст функции main().

1 вариант решения – дополнительные результаты передаются через параметры-указатели (схема 1).

Текст файла Kvadr.cpp будет выглядеть так:

```
#include <math.h>
bool Kvadr(double a, double b, double c, double *x1, double *x2)
 {
        double d;     //вспомогательная переменная
        d=b*b-4*a*c;
        if(d>=0)
        {
                *x1=(-b+sqrt(d))/(2*a);   //вычисление корней и их передача
                *x2=(-b-sqrt(d))/(2*a);    //как значений по адресу аргументов
                return true;
        }
        else
                return false;
}
```

Текст файла Kvadr.h будет выглядеть так:

```
#ifndef  KVADR_H
#define  KVADR_H
bool  Kvadr(double a, double b, double c, double *x1, double *x2);
#endif
```

Текст файла Main.cpp будет выглядеть так:

```
#include<iostream>         //подключение системных средств для
using namespace std;       //возможности использовать потоки ввода-вывода
#include <windows.h>     //прототип функций русификации
#include "kvadr.h"       //прототип пользовательской функции в конце списка
int main(void)              //имя функции main()
{
        SetConsoleCP(1251);      //вызов функций русификации
        SetConsoleOutputCP(1251);
        double a, b, c, x1, x2;      //аргументы: исходные и результаты
        bool result;            //результат, принятый от return
        cout << "\nВведите коэффициенты" << endl;
        cin >> a >> b >> c;          //ввод исходных аргументов
        result = Kvadr (a,b,c,&x1,&x2);          //вызов функции
        if (result == true)
                cout << "\nx1=" << x1 << "\nx2=" << x2 << endl;
         else
                 cout << "\nНет действительных корней" << endl;
        return 0;
}
```

2 вариант решения – дополнительные результаты передаются через параметры-ссылки (схема 2).

Текст файла Kvadr.cpp будет выглядеть так:

```
#include <math.h>
bool Kvadr(double a, double b, double c, double &x1, double &x2)
 {
        double d;     //вспомогательная переменная
        d = b*b-4*a*c;
        if (d>=0)
        {
                x1=(-b+sqrt(d))/(2*a);     //вычисление корней и их передача
                x2=(-b-sqrt(d))/(2*a);     //как ссылок на  аргументы
                return true;
        }
        else
                return false;
}
```

Текст файла Kvadr.h будет выглядеть так:

```
#ifndef  KVADR_H
#define  KVADR_H
bool Kvadr(double a, double b, double c, double &x1, double &x2);
#endif
```

Текст файла Main.cpp будет выглядеть так:

```
#include<iostream>         //подключение системных средств для
using namespace std;       //возможности использовать потоки ввода-вывода
#include <windows.h>     //прототип функций русификации
#include "kvadr.h"      //прототип пользовательской функции в конце списка
int main(void)              //имя функции main()
{
        SetConsoleCP(1251);      //вызов функций русификации
        SetConsoleOutputCP(1251);
        double a, b, c, x1, x2;      //аргументы: исходные и результаты
        bool result;            //результат, принятый от return
        cout << "\nВведите коэффициенты" << endl;
        cin >> a >> b >> c;         //ввод исходных аргументов
        result = Kvadr(a,b,c, x1, x2);     //вызов функции
        if (result == true)
                cout << "\nx1=" << x1 << "\nx2=" << x2 << endl;
         else
                 cout << "\nНет корней" << endl;
        return 0;
}
```

Порядок выполнения задания

1. Получить задание у преподавателя.
2. По вышеуказанному плану написать программу.
3. Ввести программу в компьютер.
4. Компилируя программу, найти и исправить ошибки синтаксиса.
5. Выполнить программу с исходными данными из контрольного примера, убедиться в правильности решения задачи.
6. Предъявить правильно работающую программу и контрольный пример для проверки преподавателю.

Контрольные вопросы и задания

1. Что такое ссылка (указатель)?
2. Сколько раз инициализируется ссылка? Чем?
3. Можно ли переназначить ссылку в программе?
4. Когда в функции используется параметр-ссылка (параметр-указатель)?
5. Как объявить параметр-ссылку (параметр-указатель)?
6. Как написать аргумент, соответствующий параметру-ссылке (параметру-указателю)?
7. Охарактеризуйте написанные операторы:

```
int a,b; a=func(b);
```

Напишите прототип функции func().

8. Сравните передачу данных по значению и ссылке (по форме и содержанию).
9. Напишите прототипы функций и их вызовы к нескольким вариантам задания.
10. Для наилучшего освоения темы выполните варианты 1, 9, 13.

Задание

Написать программу с функцией пользователя в двух вариантах:

1. Один результат возвратить с помощью оператора return, остальные – с помощью параметров-ссылок.
2. Один результат возвратить с помощью оператора return, остальные – с помощью параметров-указателей.

Важно! Оцените, всегда ли успешно будет выполнена ваша функция: если нет, то передавайте с помощью оператора return сигнал об ее успешности; если да, то передавайте с помощью оператора return один из результатов.

Вариант 1. Написать программу с функцией, в которой два числа обмениваются своими значениями.

Вариант 2. Написать программу с функцией, в которой по радиусу вычисляется площадь круга и длина окружности.

Вариант 3. Написать программу с функцией, в которой для произвольного прямоугольника вычисляется периметр и площадь.

Вариант 4. Написать программу с функцией, вычисляющей n-ый член арифметической прогрессии по формуле $a_n=a_1+d(n-1)$ и сумму n членов арифметической прогрессии по формуле $S_n=n(2a_1+d(n-1))/2$, где a_1 – первый член прогрессии, d – разность прогрессии.

Вариант 5. Написать программу с функцией, в которой для трех чисел вычисляется среднее арифметическое и среднее геометрическое.

Вариант 6. Написать программу с функцией, в которой вычисляются координаты (x, y) точки, делящей отрезок в отношении m1: m2.

Даны концы отрезка (x1, y1) и (x2, y2).

x= (x1+k*x2)/(1+k), y=(y1+k*y2)/(1+k), где k=m1/m2.

Вариант 7. Написать программу с функцией, вычисляющей n-ый член геометрической прогрессии по формуле $a_n=a_1q^{n-1}$ и сумму n членов геометрической прогрессии по формуле $S_n=a_1(q^n-1)/(q-1)$, где a_1 – первый член прогрессии, q – разность прогрессии.

Вариант 8. Написать программу с функцией, в которой для произвольного целого числа вычисляется количество и сумма цифр.

Вариант 9. Написать программу с функцией, в которой для произвольного натурального числа вычисляется количество и сумма делителей.

Вариант 10. Написать программу с функцией, в которой для трех чисел вычисляются минимальное и максимальное числа.

Вариант 11. Написать программу с функцией, в которой для произвольного цилиндра вычисляются объем и площадь его полной поверхности.

Вариант 12. Доказать, что любую целочисленную денежную сумму, большую 7 рублей, можно выплатить без сдачи трешками и пятерками.

Написать программу с функцией, в которой для N>7 найти такие целые неотрицательные a и b, что 3a+5b=N.

Вариант 13. Написать программу с функцией, в которой по номеру квартиры вычислить номер подъезда и этаж. Считать известными количество квартир на одном этаже, количество этажей в доме.

Вариант 14. Написать программу с функцией, в которой определяются полярные координаты точки (R, φ) по ее прямоугольным декартовым координатам, если известно, что $R^2= x^2 +y^2$; φ=arctg(y/x).

Вариант 15. Написать программу с функцией, в которой из произвольного количества секунд выделяется количество целых часов **и** из остатка - количество полных минут. Например, 3725сек это 1 час и 2 минуты.

Лабораторная работа 7

ФУНКЦИИ И СТРОКИ

Цель работы

Научиться писать функции с участием строк, в том числе:

– писать программу, состоящую из нескольких функций;

– уметь определить, объявить и вызвать функцию указанного типа;

– уметь размещать текст программы в файлах проекта в соответствии с парадигмой модульного программирования.

О чем необходимо помнить?

Изучить основные положения теории.

Рассмотрим построение функции с участием строк. Строка может играть роль *исходного данного* или *результата* работы функции.

Строка определяется *ее адресом* (см. Лабораторные работы 1, 2).

*Передача в функцию **адреса строки** используется, если внутри функции пользователя требуется **анализировать или изменять** символы строки-аргумента. При этом строка играет роль или исходного данного или результата работы функции.*

Чтобы акцентировать внимание на назначении строки, используем спецификатор *const* при объявлении параметров функции:

- ✓ const char* p – если строка не изменяется внутри функции;
- ✓ char* p – если строка изменяется внутри функции.

Напишем **схематично функцию, в которой строка будет играть роль исходного данного (схема 1)**:

```
…Func(const char*p, …)  //параметр-указатель на неизменяемую строку
    {
            …
            int i;
            for(i=0; p[i]!=0; i++)       //перебор символов строки
                    //анализ p[i]
            return …;
    }
```

```
int main(void)
{
        char str[81];           //объявили исходную строку-аргумент
        cin.getline(str,81);          //ввели строку-аргумент
        …
        …Func(str, …);      //передаем адрес строки-аргумента
        //вывод результата
        return 0;
}
```

Напишем **схематично функцию, в которой строка будет играть роль результата функции**. **Память под строку-результат выделим в функции main() (схема 2).** Обычно строка-результат получается на основе исходной строки (она тоже будет присутствовать).

```
char* Func(const char*p1, char*p2, …)         //p1-исходная, p2-результат
{
        …
        int i, j;
        for(i=0,j=0; p1[i]!=0; i++)   //перебор символов исходной строки
                if(…)          //условие выборки символов
                p2[j++]=p1[i];
        p2[j]=0;       //замкнули строку-результат ноль-символом
        return p2;     передали адрес строки-результата
}
int main(void)
{
        char str1[81];                  //объявили исходную строку
        char str2[81];                 //строка-результат
        char *p;       //переменная под результат функции
        cin.getline(str1,81);                  //ввели исходную строку
        …
        p=Func(str1,str2, …);       //передаем адреса строк
        cout<<p<<endl;     //вывод результата
        return 0;
}
```

Замечание. Строку-результат можно вывести другими способами:

```
cout<<str2<<endl;
```

или

```
cout<< Func(str1,str2, …)<<endl;
```

Такую свободу действия дает нам хорошо оформленная функция, а именно переданный результат, который, строго говоря, необязателен.

Напишем **схематично функцию, в которой строка будет играть роль результата функции**. **Память под строку-результат выделим в функции пользователя (только динамически) (схема 3).** Обычно строка-

результат получается на основе исходной строки (она тоже будет присутствовать).

```
char* Func(const char*p1, …)            //p1-исходная, p2-результат
{
        …
        char*p2=new char[strlen(p1)+1]; //память строки-результата
        int i, j;
        for(i=0,j=0;p1[i]!=0;i++)    //перебор символов исходной строки
            if(…)          //условие выборки символов
            p2[j++]=p1[i];
        p2[j]=0;       //замкнули строку-результат ноль-символом
        return p2;     //передали адрес строки-результата
}
int main(void)
{
        char str1[81];                  //объявили исходную строку
        char *p;       //переменная под результат функции
        cin.getline(str1,81);                  //ввели исходную строку
        …
        p=Func(str1, …);    //передаем адрес исходной строки
        cout<<p<<endl;     //вывод результата
        delete[]p; //освободили динамическую память
        return 0;
}
```

Замечание. Вывод строки-результата можно совместить с вызовом функции:

```
cout<< Func(str1, …)<<endl;
```

Замечание. Анализируя характеристики стандартных функций обработки строк, видим, что случаев выделения памяти внутри функции немного. Вероятно, это связано с тем, что возникает проблема освобождения динамической памяти. Поэтому, если стоит проблема выбора, склоняемся к варианту выделения памяти в main().

Порядок написания программы

1. Внимательно прочитать условие задачи и формулировку функции.
2. Провести анализ характеристик функции:

Дать название функции;

Выделить список исходных данных функции, определить их типы;

Определить тип результата функции, передаваемого с помощью оператора return.

Определить типы дополнительных результатов функции (если они есть), передаваемых с помощью параметров-ссылок или параметров-указателей.

3. Написать текст функции пользователя.

4. Написать прототип функции.

5. Написать функцию main() с вызовом функции пользователя.

6. Создать проект из двух файлов с расширением cpp (для функции пользователя и main()), и одного заголовочного файла с расширением h. Выполнить проект.

Примеры написания программ

Задача 1. Написать программу с функцией, которая определяет, содержится ли указанный символ в строке.

Задача демонстрирует передачу в функцию исходной строки (схема 1).

Анализ решения задачи. Чтобы ответить на вопрос функции можно просто подсчитать количество указанного символа в строке: если количество 0 – значит, символ не содержится, иначе – содержится. Но мы поступим немного поизящнее – обратите внимание на построение алгоритма функции.

Для определения характеристик функции ответим на три вопроса:

- ✓ Имя функции – SimvolToStr
- ✓ Список параметров функции (исходные данные). Функции необходимо передать неизменяемую строку – const char*p, и символ – char simv;
- ✓ Тип функции (тип возвращаемого результата). Функция определяет содержится ли символ в строке. Ответ – «да» или «нет». Значит, тип результата – bool.

Помним, что все имена (функции, параметров, аргументов) должны быть «говорящими». Это способствует лучшему пониманию программы.

После этого мы можем написать определение функции, ее объявление и функцию main() с вызовом функции SimvolToStr().

Программа будет строиться в виде проекта из трех файлов:

- ✓ SimvolToStr.cpp – определение функции пользователя;
- ✓ SimvolToStr.h – объявление функции пользователя;
- ✓ Main.cpp – текст функции main().

Текст файла SimvolToStr.cpp будет выглядеть так:

```
bool SimvolToStr(const  char*p, char simv)
{
        int i;   //вспомогательная переменная
//организуем перебор символов строки для их обработки
        for(i=0; p[i]!=0; i++)
                if(p[i]==simv)       //если символ строки совпал с simv
                    return true;       //завершаем функцию
//если мы в этой точке программы, то символ не найден
```

```
            return false;
        }
```

Текст файла SimvolToStr.h будет выглядеть так:

```
#ifndef  SIMVOLTOSTR_H
#define  SIMVOLTOSTR _H
bool SimvolToStr(const char*p, char simv);          //прототип функции
#endif
```

Текст файла Main.cpp будет выглядеть так:

```
#include<iostream>         //подключение системных средств для
using namespace std;       //возможности использовать потоки ввода-вывода
#include <windows.h>     //прототип функций русификации
#include " SimvolToStr.h"        //прототип пользовательской функции
int main(void)              //имя функции main()
{
        SetConsoleCP(1251);      //вызов функций русификации
        SetConsoleOutputCP(1251);
        char str[41], simvol;        //объявили данные
        bool n;         //переменная для результата функции
// ввод исходных данных
        cout<<"\nВведите строку:";
        cin.getline(str,41);
        cout<<"Введите символ: ";
        cin>>simvol;
        n= SimvolToStr(str, simvol);     //вызов функции
        if(n==true)   //анализ результата функции
                cout<<"Символ в строке найден\n";
        else
                cout<<"Символ в строке не найден\n";
        return 0;
}
```

Задача 2. Написать программу в которой формируется строка-результат из тех символов первой строки, которые содержатся и во второй.

Задача демонстрирует формирование в функции строки-результата на основе исходных строк (схема 2).

Анализ решения задачи. В задаче фигурируют три строки: две исходные и одна строка-результат. Строка-результат формируется из некоторых символов первой строки, а именно тех, которые содержатся во второй. Значит, для каждого символа первой строки необходимо определить, содержится ли он во второй строке. Вспоминаем, что эта часть алгоритма у нас уже написана в виде функции SimvolToStr() в задаче 1. Используем эту функцию при решении задачи 2.

Необходимо написать еще одну функцию, в которой на основании двух строк формируется третья.

Для определения характеристик функции ответим на три вопроса:

- ✓ Имя функции – Func
- ✓ Список параметров функции (исходные данные). Функции необходимо передать две неизменяемых строки – const char*p1,const char*p2, и изменяемую строку – char*p3;
- ✓ Тип функции (тип возвращаемого результата). Функция формирует строку-результат, строка определяется адресом, значит результат – адрес строки-результата. Адрес строки соответствует типу char*.

Помним, что все имена (функции, параметров, аргументов) должны быть «говорящими». Это способствует лучшему пониманию программы.

После этого мы можем написать определение функции, ее объявление и функцию main() с вызовом функции Func().

Программа будет строиться в виде проекта из пяти файлов:

- ✓ SimvolToStr.cpp – уже готов (см. задача 1);
- ✓ SimvolToStr.h – уже готов (см. задача 1);
- ✓ Func.cpp – определение функции пользователя;
- ✓ Func.h – объявление функции пользователя;
- ✓ Main.cpp – текст функции main().

Текст файла Func.cpp будет выглядеть так:

```
#include"SimvolToStr.h"
 char* Func(const char*p1,const char*p2,char*p3)
 {
        int i, j;
        for(i=0,j=0; p1[i]!=0; i++)
              if(SimvolToStr(p2,p1[i])==true)
                    p3[j++]=p1[i];
        p3[j]=0;
        return p3;
 }
```

Текст файла Func.h будет выглядеть так:

```
#ifndef  FUNC_H
#define  FUNC_H
char* Func(const char*p1,const char*p2,char*p3);
#endif
```

Текст файла Main.cpp будет выглядеть так:

```
#include<iostream>        //подключение системных средств для
using namespace std;      //возможности использовать потоки ввода-вывода
#include<windows.h>     //прототип функций русификации
#include"Func.h"          //прототип функции пользователя
int main(void)            //имя функции main()
{
```

```
    SetConsoleCP(1251);      //вызов функций русификации
    SetConsoleOutputCP(1251);
    const int SIZE=81;
    char*p;        //переменная для результата функции
    char str1[SIZE],str2[SIZE],str3[SIZE];         //объявили аргументы
// ввод исходных данных
    cout<<"\nВведите первую строку:";
    cin.getline(str1,SIZE);
    cout<<"\nВведите вторую строку:";
    cin.getline(str2,SIZE);
    p=Func(str1,str2,str3);     //вызов функции
    if(p[0]==0)           //анализ первого символа строки-результата
        cout<<"Строка пустая"<<endl;
    else
        cout<<"Итоговая строка: "<<p<<endl;
    return 0;
    }
```

Замечание. Понимаем, что p содержит адрес строки str3. Результат из функции Func() можно не возвращать (ее тип может быть void). При этом вариантов вызова функции становится меньше (пользователь функции ограничен, а это плохо).

Варианты вызова нашей функции:

```
    p=Func(str1,str2,str3);
    cout<< Func(str1,str2,str3)<<endl;
    strcpy_s(SomeStr, SIZE, Func(str1,str2,str3));        //и т. д.
```

Задание. Обратите внимание на достаточно простой и легко читаемый алгоритм функции Func() за счет сокрытия части алгоритма в функции SimvolToStr(). Правда, чтобы оценить это, необходимо написать Func() без использования SimvolToStr(). Если не поленитесь – получите полезный опыт.

Задание. Перепишите функцию Func(), выделяя память под строку-результат внутри функции Func() по схеме 3.

Порядок выполнения задания

1. Получить задание у преподавателя.
2. По вышеуказанному плану написать программу.
3. Ввести программу в компьютер.
4. Компилируя программу, найти и исправить ошибки синтаксиса.
5. Выполнить программу с исходными данными из контрольного примера, убедиться в правильности решения задачи.
6. Предъявить правильно работающую программу и контрольный пример для проверки преподавателю.

Контрольные вопросы и задания

1. Что такое строка? Особенности обработки строки.
2. Докажите, что строка определяется ее адресом.
3. Какой параметр необходимо объявить в функции для передачи ей строки из main() в качестве исходного данного?
4. Какой параметр необходимо объявить в функции для передачи ей строки из main() в качестве результата функции?
5. Как внутри функции пользователя создать строку-результат?
6. Если внутри функции пользователя создается строка-результат, как ее возвратить в вызывающую функцию?
7. Как в функции пользователя обработать строку?
8. Как вызывается функция пользователя с передачей ей статической строки?
9. Как вызывается функция пользователя с передачей ей динамической строки?
10. Для наилучшего освоения темы выполните варианты 13, 9, 15.

Задание

Написать программу с функцией пользователя, обрабатывающей строки. Если функция формирует строку-результат, напишите ее в двух вариантах:

1. с выделением памяти под строку-результат в main();
2. с выделением памяти под строку-результат в функции пользователя.

Вариант 1. Написать программу с функцией, вычисляющей количество какого-либо символа в строке.

Вариант 2. Написать программу с функцией, вычисляющей *адрес* первого символа строки, совпадающего с указанным символом. Если символ не найдется – результат NULL.

Вариант 3. Написать программу с функцией, приводящей все латинские буквы в строке к нижнему регистру (маленькие буквы). Посторонние символы не менять.

Вариант 4. Написать программу с функцией, устанавливающей длину исходной строки n символов: если строка длиннее – уменьшите ее, если строка короче – добавьте в конец необходимое количество символов *.

Вариант 5. Написать программу с функцией, сравнивающую на идентичность исходную строку со строкой-шаблоном. Строка-шаблон не содержит пробелов. Идентичная строка может содержать какое угодно

число пробелов после любого символа. Например: Строка-шаблон – «A>B&&B>C», идентичная строка – «A > B& & B > C ».

Вариант 6. Написать программу с функцией, определяющей, равны ли две строки (строки должны совпадать и по длине и по качеству символов).

Вариант 7. Написать программу с функцией, формирующей строку-результат путем вставки после каждого символа исходной строки указанного символа. Например, дана исходная строка fdch и символ 6. Строка-результат должна быть f6d6c6h6.

Вариант 8. Написать программу с функцией, формирующей строку-результат путем переноса из исходной строки всех не пробельных символов.

Вариант 9. Написать программу с функцией, формирующей строку-результат путем переноса из исходной строки только цифр.

Вариант 10. Написать программу с функцией, формирующей строку-результат путем переноса из исходной строки только латинских букв.

Вариант 11. Даны две строки. Написать программу с функцией, формирующей строку-результат из таких символов второй строки, которые не входят в первую.

Вариант 12. Даны две строки. Написать программу с функцией, формирующей строку-результат путем переноса в нее б*о*льших символов из каждой пары символов, составленных из символов первой и второй строки с одинаковыми номерами.

Вариант 13. Написать программу с функцией, формирующей строку-копию исходной строки.

Вариант 14. Дана строка, в которой символы могут повторяться многократно. Написать программу с функцией, формирующей строку-результат так, что каждый символ исходной строки в ней присутствует только один раз.

Вариант 15. Дана строка, содержащая как цифры, так и посторонние символы. Написать программу с функцией, формирующей из цифровых символов строки целое число. Например, дана строка 12gh5i, тогда результатом функции будет целое число 125.

Лабораторная работа 8

ФУНКЦИИ И ОДНОМЕРНЫЕ ЧИСЛОВЫЕ МАССИВЫ

Цель работы

Научиться писать функции с участием одномерных числовых массивов, в том числе:

– писать программу, состоящую из нескольких функций;

– уметь определить, объявить и вызвать функцию указанного типа;

– уметь размещать текст программы в файлах проекта в соответствии с парадигмой модульного программирования.

О чем необходимо помнить?

Изучить основные положения теории.

Рассмотрим построение функции с участием одномерных числовых массивов. Одномерный числовой массив может играть роль *исходного данного* или *результата* работы функции.

Одномерный числовой массив (далее просто Массив) определяется *его адресом и количеством элементов* (см. Лабораторные работы 1, 3).

*Передача в функцию **Массива** используется, если внутри функции пользователя требуется **анализировать или изменять** элементы Массива-аргумента. При этом Массив играет роль или исходного данного или результата работы функции.*

Чтобы акцентировать внимание на назначении Массива, используем спецификатор *const* при объявлении параметров функции:

- ✓ const Тип *p, int n – если Массив не изменяется внутри функции;
- ✓ Тип *p,int n – если Массив изменяется внутри функции.

  Тип – любой стандартный числовой тип (int, long, double и пр.)

Приведем в качестве примеров написания функций две основные схемы. *Не следует понимать, что любая задача должна быть уложена в предлагаемые схемы (во всяком случае, этих схем может быть в разы больше, чтобы отобразить различные нюансы написания функций). Применять схемы надо творчески, усвоив лишь механизм их написания.*

Напишем **схематично функцию, в которой Массив будет играть роль исходного данного (схема 1)**:

```
…Func(const Тип*p, int n, …)   //передача неизменяемого Массива
{
        …
        int i;
        for(i=0; i<n; i++)   //перебор элементов Массива
              //анализ p[i]
        return …;
}
```

```
int main(void)
{
        const int SIZE=5;
        Тип mas[SIZE]={1,5,-3,0,-6};  //объявили исходный Массив
        …
        …Func(mas,SIZE, …);    //вызов функции
        //вывод результата
        return 0;
}
```

Замечание. Для простоты Массив объявлен с инициализацией. Обычно массив заполняется (с клавиатуры, из файла, случайными числами).

Напишем **схематично функцию, в которой Массив будет играть роль результата функции**. **Память под Массив-результат выделим в функции main() (схема 2).** Часто Массив-результат получается на основе исходного Массива (он тоже будет присутствовать).

Запись элементов исходного Массива в Массив-результат происходит выборочно, по какому-либо критерию. Память под оба Массива выделяем первоначально одинакового размера. В процессе заполнения Массива размер его определяется и передается в main() стандартным путем.

```
int  Func(const Тип*p1,Тип*p2,int n, …)//p1-исходный, p2-результат
{
        …
        int i, j;
        for(i=0,j=0; i<n; i++)   //перебор элементов исходного Массива
                if(…)          //условие выборки элементов
                p2[j++]=p1[i];       //запись элементов в Массив-результат
        return j;       передали размер Массива-результата
}
int main(void)
{
        const int SIZE=5;
        Тип mas1[SIZE]={4,7,-3,0,3}; //объявили исходный Массив
        Тип mas2[SIZE];           //Массив-результат
        int N;          //переменная под результат функции
        …
        N=Func( mas1,mas2,SIZE,…);  //передаем Массивы
        //выводим Массив-результат mas2 размером N
        return 0;
}
```

Замечание. В программах ввод, вывод массива будет производиться с помощью соответствующих функций.

Порядок написания программы

1. Внимательно прочитать условие задачи и формулировку функции.

2. Выделить функции пользователя и провести анализ характеристик каждой функции в отдельности:

Дать название функции;

Выделить список исходных данных функции, определить их типы;

Определить тип результата функции, передаваемого с помощью оператора return.

Определить тип дополнительных (если они будут) результатов функции, передаваемых с помощью параметров-ссылок или параметров-указателей.

Написать текст функции пользователя.

Написать прототип функции.

3. Написать функцию main() с вызовом функций пользователя.

4. Создать проект из двух файлов с расширением cpp (для функций пользователя и main()), и одного заголовочного файла с расширением h.

5. Выполнить проект.

Примеры написания программ

Задача 1. Написать программу, вычисляющую значение максимального элемента целочисленного одномерного массива.

Задача демонстрирует применение обеих схем написания функций и их модификаций.

Выделим функции:

1. Заполнение массива случайными числами в диапазоне от –к до +к.
2. Вывод одномерного массива.
3. Вычисление значения максимального элемента целочисленного одномерного массива.

Анализ решения задачи. Необходимо написать три функции.

Для определения характеристик каждой функции ответим на три вопроса.

Функция заполнения случайными числами:

- ✓ Имя функции – InMas
- ✓ Список параметров функции (исходные данные). Функции необходимо передать изменяемый одномерный массив для его заполнения – int *p, int n; и переменную, задающую диапазон изменения чисел - int k;
- ✓ Тип функции (тип возвращаемого результата). Функция ничего не будет вычислять, поэтому и передавать нечего. Тип функции – void

Функция вывода массива на экран монитора:

- ✓ Имя функции – OutMas
- ✓ Список параметров функции (исходные данные). Функции необходимо передать неизменяемый одномерный массив для его вывода – const int *p, int n;
- ✓ Тип функции (тип возвращаемого результата). Функция ничего не будет вычислять, поэтому и передавать нечего. Тип функции – void

Функция вычисления значения максимального элемента:

- ✓ Имя функции – MaxMas
- ✓ Список параметров функции (исходные данные). Функции необходимо передать неизменяемый одномерный массив – const int *p, int n;
- ✓ Тип функции (тип возвращаемого результата). Максимальный элемент того же типа, что и сам массив, поэтому тип результата - int.

Помним, что все имена (функции, параметров, аргументов) должны быть «говорящими». Это способствует лучшему пониманию программы.

После этого мы можем написать определения функций, их объявления и функцию main() с вызовом функций пользователя. Поскольку функции тесно связаны между собой, имеет смысл поместить их в один файл. Файл с объявлениями функций тоже будет один.

Программа будет строиться в виде проекта из трех файлов:

- ✓ Mas.cpp – определения функций пользователя;
- ✓ Mas.h – объявления функций пользователя;
- ✓ Main.cpp – текст функции main().

Текст файла Mas.cpp будет выглядеть так:

```
#include<iostream>        //подключение системных средств для
using namespace std;      //возможности использовать потоки ввода-вывода
#include<stdlib.h>        //для формирования случайных чисел
void InMas(int*p,int n,int k)
    {
        int i;
        for(i=0; i<n; i++)   //перебор элементов массива
            p[i]=rand()%(2*k+1)-k; //формируем случайное число
        return;
    }
void OutMas(const int*p, int n)
    {
        int i;
        for(i=0; i<n; i++)
            cout<< p[i]<<endl;
        cout<< endl;
```

```
        return;
    }
int MaxMas(const int*p, int n)
    {
        int i;
//описание стандартного алгоритма см. Лабораторная работа 3
        int max=p[0];
        for(i=0; i<n; i++)
          if(p[i]>max)
            max=p[i];
        return max;
    }
```

Текст файла Mas.h будет выглядеть так:

```
#ifndef  MAS_H
#define  MAS_H
void InMas(int*p,int n,int k);
void OutMas(const int*p,int n);
int MaxMas(const int*p, int n);
#endif
```

Текст файла Main.cpp будет выглядеть так:

```
#include<iostream>        //подключение системных средств для
using namespace std;      //возможности использовать потоки ввода-вывода
#include<stdlib.h>        //для функции srand()
#include<time.h>          //для функции time ()
#include <windows.h>     //прототип функций русификации
#include "Mas.h"          //прототипы функций пользователя
int main(void)           //имя функции main()
{
    SetConsoleCP(1251);      //вызов функций русификации
    SetConsoleOutputCP(1251);
    const int SIZE=5;
    int mas[SIZE];      //объявляем массив
    int max;       //переменная под результат функции
    srand((unsigned)time(NULL));   //инициализируем генератор
                                   //случайных чисел
    InMas(mas,SIZE,10);
    cout<<"Исходный массив:  "<<endl;
    OutMas(mas,SIZE);
    max=MaxMas(mas,SIZE);
    cout<< "\nМаксимальный элемент равен "<<max<<endl;
    return 0;
}
```

Задача 2. Написать программу, формирующую массив-результат из положительных элементов исходного массива.

Задача демонстрирует применение обеих схем написания функций и их модификаций.

Выделим функции:

1. Заполнение массива случайными числами в диапазоне от –к до +к.
2. Вывод одномерного массива.
3. Формирование массива-результата из положительных элементов исходного массива.

Анализ решения задачи. Необходимо написать три функции, но две из них уже написаны в Задаче 1.

Для определения характеристик третьей функции ответим на три вопроса.

Функция формирования массива-результата из положительных элементов исходного массива (схема 2):

- ✓ Имя функции – RezMas
- ✓ Список параметров функции (исходные данные). Функции необходимо передать неизменяемый одномерный массив – const int *p1, int n, и изменяемый массив – int *p2, int n
- ✓ Тип функции (тип возвращаемого результата). Внутри функции вычисляем реальный размер массива-результата - int.

Помним, что все имена (функции, параметров, аргументов) должны быть «говорящими». Это способствует лучшему пониманию программы.

После этого мы можем написать определения функций, их объявления и функцию main() с вызовом функций пользователя. Поскольку функции тесно связаны между собой, добавим в уже созданные файлы Mas.cpp и Mas.h объявление и определение функции RezMas().

Программа будет строиться в виде проекта из трех файлов:

- ✓ Mas.cpp – определения функций пользователя;
- ✓ Mas.h – объявления функций пользователя;
- ✓ Main.cpp – текст функции main().

Текст файла Mas.cpp будет выглядеть так:

```
#include<iostream>
using namespace std;
#include<stdlib.h>
void InMas(int *p,int n,int k)
	{
		int i;
		for(i=0; i<n; i++)
			p[i]=rand()%(2*k+1)-k;
		return;
```

```
}
void OutMas(const int *p, int n)
{
        int i;
        for(i=0; i<n; i++)
                cout<< p[i]<<endl;
        cout<<endl;
        return;
}
int RezMas(const int*p1,int*p2,int n)
{
        int i,j;
        for(i=0,j=0; i<n; i++)
          if(p1[i]>0)
            p2[j++]=p1[i];
        return j;
}
```

Текст файла Mas.h будет выглядеть так:

```
#ifndef  MAS_H
#define  MAS_H
void InMas(int *p, int n, int k);
void OutMas(const int *p, int n);
int RezMas(const int*p1,int*p2,int n);
#endif
```

Текст файла Main.cpp будет выглядеть так:

```
#include<iostream>
using namespace std;
#include<stdlib.h>
#include<time.h>
#include <windows.h>
#include "Mas.h"
int main(void)
{
        SetConsoleCP(1251);
        SetConsoleOutputCP(1251);
        const int SIZE=5;
        int mas1[SIZE],mas2[SIZE];
        int N;
        srand((unsigned)time(NULL));
        InMas(mas1,SIZE,10);
        cout<<"Исходный массив: "<<endl;
        OutMas(mas1,SIZE);
        N=RezMas(mas1,mas2,SIZE);
```

```
    cout<< "Массив с положительными элементами:"<<endl;
    OutMas(mas2,N);
    return 0;
}
```

Порядок выполнения задания

1. Получить задание у преподавателя.
2. По вышеуказанному плану написать программу.
3. Ввести программу в компьютер.
4. Компилируя программу, найти и исправить ошибки синтаксиса.
5. Выполнить программу с исходными данными из контрольного примера, убедиться в правильности решения задачи.
6. Предъявить правильно работающую программу и контрольный пример для проверки преподавателю.

Контрольные вопросы и задания

1. Что такое одномерный числовой массив?
2. Как объявить массив статически и динамически?
3. Докажите, что любой массив определяется адресом и количеством элементов.
4. Какой параметр необходимо объявить в функции для передачи ей одномерного числового массива из main() в качестве исходного данного?
5. Какой параметр необходимо объявить в функции для передачи ей одномерного числового массива из main() в качестве результата функции?
6. Как в функции пользователя обработать одномерный числовой массив?
7. Как вызывается функция пользователя с передачей ей статического одномерного числового массива?
8. Как вызывается функция пользователя с передачей ей динамического одномерного числового массива?
9. Напишите прототипы и вызовы функций к нескольким вариантам задания.
10. Для наилучшего освоения темы выполните варианты 12, 10, 11.

Задание

Написать программу с функциями пользователя, обрабатывающими одномерные числовые массивы. Память под все массивы выделите в main() двумя способами:

1. статически;
2. динамически.

Вариант 1. Написать программу, выполняющую инвертирование одномерного числового массива.

Выделим функции:

1. Заполнение массива с клавиатуры**.**
2. Вывод одномерного массива.
3. Инвертирование одномерного массива (первый элемент массива меняется местом с последним, второй – с предпоследним и т.д.).

Вариант 2. Написать программу, вычисляющую сумму и количество положительных элементов одномерного числового массива.

Выделим функции:

1. Заполнение массива с клавиатуры**.**
2. Вывод одномерного массива.
3. Вычисление суммы и количество положительных элементов массива (функция с двумя результатами).

Вариант 3. Написать программу циклического сдвига элементов в массиве на одну позицию вправо. Например, если в массиве содержится пять значений (1,2,3,4,5), то вызов функции изменит массив на (5,1,2,3,4).

Выделим функции:

1. Заполнение массива с клавиатуры.
2. Вывод одномерного массива.
3. Циклический сдвиг элементов в исходном массиве.

Вариант 4. Написать программу, формирующую массив-результат на базе исходного массива.

Выделим функции:

1. Заполнение вещественного массива с клавиатуры.
2. Вывод вещественного одномерного массива.
3. Вывод целого одномерного массива
4. Формирование массива-результата путем округления каждого элемента исходного массива до целого.

Вариант 5. Написать программу, формирующую массив-результат на базе исходного массива.

Выделим функции:

1. Заполнение массива с клавиатуры.
2. Вывод одномерного массива.
3. Формирование массива-результата путем исключения из исходного массива к-го элемента, где к – номер элемента исходного массива.

Вариант 6. Написать программу, формирующую массив-результат на базе исходного массива.

Выделим функции:

1. Заполнение массива числами с клавиатуры.
2. Вывод одномерного массива.

3. Формирование массива-результата, состоящего из четных элементов исходного массива.

Вариант 7. Написать программу, формирующую массив-результат на базе исходного массива.

Выделим функции:

1. Заполнение массива числами с клавиатуры.
2. Вывод одномерного массива.
3. Формирование массива-результата путем вставки в исходный массив после N-ого по счету элемента значения, равного к.

Вариант 8. Написать программу, формирующую массив-результат на базе исходного массива.

Выделим функции:

1. Заполнение массива с клавиатуры.
2. Вывод одномерного массива.
3. Формирование массива-результата, состоящего из тех элементов исходного массива, которые больше a, но меньше b.

Вариант 9. Написать программу, формирующую массив-результат на базе исходного массива.

Выделим функции:

1. Заполнение массива с клавиатуры.
2. Вывод одномерного массива.
3. Формирование массива -результата, состоящего из тех элементов исходного массива, которые кратны числу a.

Вариант 10. Написать программу, формирующую массив-результат на базе исходного массива.

Выделим функции:

1. Заполнение массива числами с клавиатуры.
2. Вывод одномерного массива.
3. Формирование массива-результата, состоящего из *индексов* отрицательных элементов исходного массива.

Вариант 11. Написать программу, формирующую массив-результат на базе исходного массива.

Выделим функции:

1. Заполнение целочисленного массива с клавиатуры.
2. Вывод одномерного массива.
3. Вычисление суммы цифр целого числа.
4. Формирование массива-результата, каждый элемент которого вычисляется как сумма цифр соответствующего элемента исходного массива.

Вариант 12. Написать программу, формирующую копию исходного массива.

Выделим функции:

1. Заполнение массива числами с клавиатуры.
2. Вывод одномерного массива.
3. Формирование копии исходного массива.

Вариант 13. Написать программу, формирующую массив-результат на базе исходного массива.

Выделим функции:

1. Заполнение массива с клавиатуры.
2. Вывод одномерного массива.
3. Формирование массива-результата, состоящего из n последних элементов исходного массива.

Вариант 14. Написать программу, формирующую массив-результат на базе вещественного исходного массива.

Выделим функции:

1. Заполнение массива с клавиатуры.
2. Вывод одномерного массива.
3. Формирование массива-результата путем записи элементов исходного массива через одного.

Вариант 15. Дан одномерный массив из вещественных чисел: a_1,a_2, a_3, …,a_{2n}. Сформировать массив-результат, в котором элементы исходного массива идут в следующем порядке: a_1,a_{n+1},a_2,a_{n+2}, …,a_n,a_{2n}.

Выделим функции:

1. Заполнение массива числами с клавиатуры.
2. Вывод одномерного массива.
3. Формирование массива-результата по указанному правилу.

Лабораторная работа 9

ФУНКЦИИ И МАТРИЦЫ

Цель работы

Научиться писать функции с участием матриц, в том числе:

– писать программу, состоящую из нескольких функций;

– уметь определить, объявить и вызвать функцию указанного типа;

– уметь размещать текст программы в файлах проекта в соответствии с парадигмой модульного программирования.

О чем необходимо помнить?

Изучить основные положения теории.

Рассмотрим построение функции с участием матриц. Матрицу будем передавать в функцию как одномерный массив.

Матрица определяется *адресом первого элемента, количеством строк и количеством столбцов* (см. Лабораторные работы 1, 4).

*Передача в функцию **матрицы** используется, если внутри функции пользователя требуется **анализировать или изменять** элементы матрицы-аргумента. При этом матрица играет роль или исходного данного или результата работы функции.*

Чтобы акцентировать внимание на назначении матрицы, используем спецификатор *const* при объявлении параметров функции:

- ✓ const Тип *p,int n,int m – если матрица не изменяется внутри функции;
- ✓ Тип *p,int n,int m – если матрица изменяется внутри функции.

Тип – любой стандартный числовой тип (int, long, double и пр.)

Приведем в качестве примеров написания функций две основные схемы. *Не следует понимать, что любая задача должна быть уложена в предлагаемые схемы (во всяком случае, этих схем может быть в разы больше, чтобы отобразить различные нюансы написания функций). Применять схемы надо творчески, усвоив лишь механизм их написания.*

Напишем **схематично функцию, в которой матрица будет играть роль исходного данного (схема 1)**:

```
…Func(const Тип*p,int n,int m, …)   //передача неизменяемой матрицы
{
        …
        int i, j;
        for(i=0; i<n; i++)   //перебор матрицы построчно
                for(j=0; j<m; j++)
                //анализ p[i*m+j]
        …
        return …;
}
int main(void)
{
        const int N=2,M=3;
        Тип mas[N][M]={1,5,-3,0,-6,7};
        …
        …Func(mas[0],N,M, …);          //вызов функции
        //вывод результата
        return 0;
}
```

Замечание. Для простоты матрица объявлена с инициализацией. Обычно матрица заполняется (с клавиатуры, из файла, случайными числами).

Напишем **схематично функцию, в которой Массив будет играть роль результата функции**. **Память под Массив-результат выделим в функции main() (схема 2).**

```
…Func(Тип*p,int n,int m, …)
{
        …
        int i, j;
        for(i=0; i<n; i++)   //перебор матрицы построчно
                for(j=0; j<m; j++)
                        p[i*m+j]=…;
        return …;
}
int main(void)
{
        const int N=2,M=3;
        Тип mas[N][M];
        …
        …Func(mas[0],N,M, …);          //вызов функции
        //вывод измененной матрицы mas
        return 0;
}
```

Замечание. В программах ввод, вывод массива будет производиться с помощью соответствующих функций.

Важное замечание. Если матрица обрабатывается *построчно* (одна строка, несколько строк или вся), то можно использовать функции обработки *одномерного массива*, передавая *адрес* первого обрабатываемого элемента матрицы и *количество* обрабатываемых элементов. И наоборот. Функция обработки матрицы построчно может быть использована для обработки одномерного массива, так как одномерный массив из M элементов, это матрица из одной строки и M столбцов.

Порядок написания программы

1. Внимательно прочитать условие задачи.

2. Выделить функции пользователя и провести анализ характеристик каждой функции в отдельности:

Дать название функции;

Выделить список исходных данных функции, определить их типы;

Определить тип результата функции, передаваемого с помощью оператора return.

Определить тип дополнительных (если они будут) результатов функции, передаваемых с помощью параметров-ссылок или параметров-указателей.

Написать текст функции пользователя.

Написать прототип функции.

3. Написать функцию main() с вызовом функций пользователя.

4. Создать проект из двух файлов с расширением cpp (для функций пользователя и main()), и одного заголовочного файла с расширением h.

5. Выполнить проект.

Примеры написания программ

Задача 1. Определить, является ли указанная строка матрицы убывающей последовательностью.

Выделим функции:

1. заполнение матрицы числами с клавиатуры;
2. вывод матрицы в матричном виде;
3. определение, является ли указанная строка матрицы убывающей последовательностью.

Анализ решения задачи. Необходимо написать три функции.

Для определения характеристик каждой функции ответим на три вопроса.

Функция заполнения матрицы числами с клавиатуры:

- ✓ Имя функции – InMatr
- ✓ Список параметров функции (исходные данные). Функции необходимо передать матрицу для ее изменения (ввода) – int *p, int n, int m;
- ✓ Тип функции (тип возвращаемого результата). Функция ничего не будет вычислять, поэтому и передавать нечего. Тип функции – void

Функция вывода матрицы на экран монитора:

- ✓ Имя функции – OutMatr
- ✓ Список параметров функции (исходные данные). Функции необходимо передать неизменяемую матрицу – const int *p, int n, int m;
- ✓ Тип функции (тип возвращаемого результата). Функция ничего не будет вычислять, поэтому и передавать нечего. Тип функции – void

Функция определения, является ли указанная строка матрицы убывающей последовательностью:

- ✓ Имя функции – Ubyv
- ✓ Список параметров функции (исходные данные). Функции необходимо передать неизменяемую матрицу – const int *p, int n, int m; и *номер* обрабатываемой строки int N;
- ✓ Тип функции (тип возвращаемого результата). Функция вычисляет ответ типа «да» или «нет» – тип функции bool

Помним, что все имена (функции, параметров, аргументов) должны быть «говорящими». Это способствует лучшему пониманию программы.

После этого мы можем написать определения функций, их объявления и функцию main() с вызовом функций пользователя. Поскольку функции тесно связаны между собой, имеет смысл поместить их в один файл. Файл с объявлениями функций тоже будет один.

Программа будет строиться в виде проекта из трех файлов:

- ✓ Matr.cpp – определения функций пользователя;
- ✓ Matr.h – объявления функций пользователя;
- ✓ Main.cpp – текст функции main().

Текст файла Matr.cpp будет выглядеть так:

```
#include<iostream>        //подключение системных средств для
using namespace std;      //возможности использовать потоки ввода-вывода
#include<iomanip>         //для форматированного вывода матрицы
void InMatr(int*p,int n,int m)
{
        int i, j;
        for(i=0; i<n; i++)
                for(j=0; j<m; j++)
                        cin>>p[i*m+j];
        return;
}
void OutMatr(const int*p,int n,int m)
{
        int i, j;
        cout<<endl;
        for(i=0; i<n; i++)
        {
                for(j=0; j<m; j++)
                        cout<<setw(5)<<p[i*m+j];
                cout<<endl;
        }
        return;
}
bool Ubyv(const int*p,int n,int m,int N)
{
        int i,j;
        i=N-1;          //определяем индекс обрабатываемой строки
        for(j=0; j<m-1; j++)        //перебор столбцов одной строки
                if(p[i*m+j]<=p[i*m+j+1])    //если соседние элементы «плохие»
                        return false;           //завершаем функцию
```

```
        return true;   //в остальных случаях - ответ «да»
}
```

Текст файла Matr.h будет выглядеть так:

```
#ifndef  MATR_H
#define  MATR_H
void InMatr(int*p,int n,int m);
void OutMatr(const int*p,int n,int m);
bool Ubyv(const int*p,int n,int m,int N);
#endif
```

Текст файла Main.cpp будет выглядеть так:

```
#include<iostream>        //подключение системных средств для
using namespace std;      //возможности использовать потоки ввода-вывода
#include <windows.h>     //прототип функций русификации
#include "Matr.h"         //прототипы функций пользователя
int main(void)
{
        SetConsoleCP(1251);
        SetConsoleOutputCP(1251);
        const int STR=3,STLB=4;        //задаем размерность матрицы
        int matr[STR][STLB];     //объявляем матрицу
        int N;          //переменная под номер обрабатываемой строки
        cout<<"Введите массив"<<endl;
        InMatr(matr[0],STR,STLB);
        cout<<"Исходный массив: ";
        OutMatr(matr[0],STR,STLB);
        cout<<"Введите номер обрабатываемой строки: ";
        cin>>N;
        if(Ubyv(matr[0],STR,STLB,N)==true)   //анализ результата функции
                cout<<N<<" строка убывает"<<endl;
        else
                cout<<N<<" строка не убывает"<<endl;
        return 0;
}
```

Задача 2. Написать программу умножения двух матриц с получением матрицы-произведения.

Выделим функции:

1. заполнение матрицы числами с клавиатуры;
2. вывод матрицы в матричном виде;
3. вычисление произведения двух матриц.

Анализ решения задачи. Необходимо написать три функции, но две из них уже написаны в Задаче 1. Для построения алгоритма решения третьей функции необходимо основательно вспомнить линейную алгебру. Определиться с размерами матриц: если размер левой матрицы n*m, пра-

вой матрицы m*q, то размер матрицы-результата n*q. Для получения одного элемента матрицы-результата с индексами i,k необходимо i-ую строку левой матрицы умножить на k-ый столбец правой матрицы.

Для определения характеристик третьей функции ответим на три вопроса.

Функция вычисляющая произведение двух матриц:

- ✓ Имя функции – Multiply
- ✓ Список параметров функции (исходные данные). Функции необходимо передать две неизменяемые матрицы – const int *p1, const int *p2, изменяемую матрицу-результат int *p3, и размеры матриц: int n, int m, int q;
- ✓ Тип функции (тип возвращаемого результата). Функция вычисляет матрицу-результат. Возможный тип функции int*, но мы определим тип как void

Помним, что все имена (функции, параметров, аргументов) должны быть «говорящими». Это способствует лучшему пониманию программы.

После этого мы можем написать определения функций, их объявления и функцию main() с вызовом функций пользователя. Поскольку функции тесно связаны между собой, добавим определение и объявление функции Multiply() в ранее написанные файлы Matr.cpp и Matr.h.

Программа будет строиться в виде проекта из трех файлов:

- ✓ Matr.cpp – определения функций пользователя;
- ✓ Matr.h – объявления функций пользователя;
- ✓ Main.cpp – текст функции main().

Текст файла Matr.cpp будет выглядеть так:

```
#include<iostream>        //подключение системных средств для
using namespace std;      //возможности использовать потоки ввода-вывода
#include<iomanip>         //для форматированного вывода матрицы
void InMatr(int*p,int n,int m)
{
        int i,j;
        for(i=0; i<n; i++)
                for(j=0; j<m; j++)
                        cin>>p[i*m+j];
        return;
}
void OutMatr(const int*p,int n,int m)
{
        int i,j;
        cout<<endl;
        for(i=0; i<n; i++)
        {
```

```
            for(j=0; j<m; j++)
                cout<<setw(5)<<p[i*m+j];
            cout<<endl;
        }
        return;
}
void Multiply(const int*p1,const int*p2,int*p3,int n,int m,int q)
{
        int i,j,k;
        for(i=0; i<n; i++)   //для каждого элемента
            for(k=0; k<q; k++)//матрицы-результата
            {
                p3[i*q+k]=0;
                for(j=0; j<m; j++)  //i-ая строка p1 умножается на
                                    // k-ый столбец p2
                    p3[i*q+k]+=p1[i*m+j]*p2[j*q+k];
            }
        return;
}
```

Текст файла Matr.h будет выглядеть так:

```
#ifndef  MATR_H
#define  MATR_H
void InMatr(int*p,int n,int m);
void OutMatr(const int*p,int n,int m);
void Multiply(const int*p1,const int*p2,int*p3,int n,int m,int q);
#endif
```

Текст файла Main.cpp будет выглядеть так:

```
#include<iostream>        //подключение системных средств для
using namespace std;       //возможности использовать потоки ввода-вывода
#include <windows.h>    //прототип функций русификации
#include "Matr.h"          //прототипы функций пользователя
int main(void)
{
        SetConsoleCP(1251);
        SetConsoleOutputCP(1251);
        const int N=2,M=3,Q=2;          //задаем размерности матриц
        int A[N][M],B[M][Q],C[N][Q];         //объявляем матрицы
        cout<<"Введите первую матрицу"<<endl;
        InMatr(A[0],N,M);
        cout<<"Введите вторую матрицу"<<endl;
        InMatr(B[0],M,Q);
        cout<<"Исходные матрицы:\n";
        OutMatr(A[0],N,M);       //выводим исходные
```

```
    OutMatr(B[0],M,Q);        //матрицы на экран
    Multiply(A[0],B[0],C[0],N,M,Q);
    cout<<"Произведение матриц равно:\n";
    OutMatr(C[0],N,Q);        //выводим матрицу-результат на экран
    return 0;
}
```

Порядок выполнения задания

1. Получить задание у преподавателя.
2. По вышеуказанному плану написать программу.
3. Ввести программу в компьютер.
4. Компилируя программу, найти и исправить ошибки синтаксиса.
5. Выполнить программу с исходными данными из контрольного примера, убедиться в правильности решения задачи.
6. Предъявить правильно работающую программу и контрольный пример для проверки преподавателю.

Контрольные вопросы и задания

1. Что такое матрица?
2. Как объявить матрицу статически и динамически?
3. Докажите, что любая матрица определяется адресом первого элемента, количеством строк и столбцов матрицы.
4. Какой параметр необходимо объявить в функции для передачи ей матрицы из main() в качестве исходного данного?
5. Какой параметр необходимо объявить в функции для передачи ей матрицы из main() в качестве результата функции?
6. Как в функции пользователя обработать матрицу?
7. Как вызывается функция пользователя с передачей ей статической матрицы?
8. Как вызывается функция пользователя с передачей ей динамической матрицы?
9. Напишите прототипы и вызовы функций к нескольким вариантам задания.
10. Для наилучшего освоения темы выполните варианты 8, 12, 15.

Задание

Написать программу с функциями пользователя, обрабатывающими матрицы. Память для всех матриц выделите в main() двумя способами:

1. статически;
2. динамически.

Вариант 1. Написать программу умножения матрицы на число соответствующего типа.

Выделим функции:

1. Заполнение матрицы случайными числами в диапазоне [-k,k].
2. Вывод матрицы в матричном виде.
3. Умножение элементов матрицы на число соответствующего типа.

Вариант 2. Написать программу формирования копии исходной матрицы.

Выделим функции:

1. Заполнение матрицы случайными числами в диапазоне [-k,k].
2. Вывод матрицы в матричном виде.
3. Формирование матрицы-результата как копии исходной матрицы.

Вариант 3. Написать программу формирования матрицы, противоположной исходной матрицы.

Выделим функции:

1. Заполнение матрицы случайными числами в диапазоне [-k,k].
2. Вывод матрицы в матричном виде.
3. Формирование матрицы-результата путем умножения каждого элемента исходной матрицы на -1.

Вариант 4. Написать программу вычисления суммы двух матриц.

Выделим функции:

1. Заполнение матрицы случайными числами в диапазоне [-k,k].
2. Вывод матрицы в матричном виде.
3. Формирование матрицы-результата как поэлементной суммы двух исходных матриц.

Вариант 5. Написать программу транспонирования матрицы.

Выделим функции:

1. Заполнение матрицы случайными числами в диапазоне [-k,k].
2. Вывод матрицы в матричном виде.
3. Формирование матрицы-результата путем транспонирования исходной матрицы.

Вариант 6. Написать программу умножения элементов строки матрицы с номером N на число соответствующего типа.

Выделим функции:

1. Заполнение матрицы случайными числами в диапазоне [-k,k].
2. Вывод матрицы в матричном виде.
3. Изменение элементов строки матрицы с номером N по указанному правилу.

Вариант 7. Написать программу сравнения двух матриц одинакового размера на равенство.

Выделим функции:

1. Заполнение матрицы случайными числами в диапазоне [-k,k].
2. Вывод матрицы в матричном виде.
3. Сравнение двух матриц на равенство. Результат – номер первой строки, где обнаружились не равные элементы.

Вариант 8. Написать программу умножения элементов столбца матрицы с номером M на число соответствующего типа.

Выделим функции:

1. Заполнение матрицы случайными числами в диапазоне [-k,k].
2. Вывод матрицы в матричном виде.
3. Изменение элементов столбца матрицы с номером M по указанному правилу.

Вариант 9. Написать программу вычисления первого максимума матрицы.

Выделим функции:

1. Заполнение матрицы случайными числами в диапазоне [-k,k].
2. Вывод матрицы в матричном виде.
3. Вычисление значения максимума, и индексов строки и столбца, где зафиксирован *первый* максимум.

Вариант 10. Написать программу обмена значениями столбцов с номерами M1 и M2 в матрице.

Выделим функции:

1. Заполнение матрицы случайными числами в диапазоне [-k,k].
2. Вывод матрицы в матричном виде.
3. Изменение исходной матрицы по указанному правилу.

Вариант 11. Написать программу обмена значениями строк с номерами N1 и N2 в матрице.

Выделим функции:

1. Заполнение матрицы случайными числами в диапазоне [-k,k].
2. Вывод матрицы в матричном виде.
3. Изменение исходной матрицы по указанному правилу.

Вариант 12. Написать программу изменения k-той строки матрицы по формуле A_{kj}+=koef*A_{nj}; koef подобрать так, чтобы первый элемент k-той строки стал равен 0.

Выделим функции:

1. Заполнение матрицы случайными числами в диапазоне [-k,k].
2. Вывод матрицы в матричном виде.
3. Изменение исходной матрицы по указанному правилу. Матрица, k и n – исходные данные функции.

Вариант 13. Написать программу формирования матрицы-результата путем удаления из исходной матрицы строки с номером N и столбца с номером M.

Выделим функции:

1. Заполнение матрицы случайными числами в диапазоне [-k,k].
2. Вывод матрицы в матричном виде.
3. Формирование матрицы-результата по указанному правилу.

Вариант 14. Написать программу выполнения следующего действия над матрицами A*=B. Матрицы квадратные, одного размера. Воспользуйтесь приведенным примером 2, как основой для построения алгоритма.

Выделим функции:

1. Заполнение матрицы случайными числами в диапазоне [-k,k].
2. Вывод матрицы в матричном виде.
3. Изменение матрицы A по указанному правилу.

Вариант 15. Написать программу определения номера строки и столбца первой седловой точки матрицы.

Выделим функции:

1. Заполнение матрицы случайными числами в диапазоне [-k,k].
2. Вывод матрицы в матричном виде.
3. Определение седловой точки. Матрица A имеет седловую точку A_{ij}, если A_{ij} является минимальным элементом в i-ой строке и максимальным в j-ом столбце.

Лабораторная работа 10

ШАБЛОНЫ ФУНКЦИЙ

Цель работы

Научиться писать шаблоны функций, в том числе:

– писать программу, состоящую из нескольких функций;

– уметь определить и вызвать шаблонную функцию;

– знать механизм обработка функций-шаблонов компилятором;

– знать особенности размещения текста программы в файлах проекта при использовании шаблонов функций.

О чем необходимо помнить?

Изучить основные положения теории.

Многие алгоритмы *не зависят* от типа данных, с которыми они работают.

Шаблоны функций определяют алгоритм, который будет применяться к данным различного типа. Конкретный тип данных будет передаваться при вызове функции.

Порядок работы для разработки шаблона функции:

- ✓ Формулируем функцию, шаблон которой хотим написать;
- ✓ Выделяем данные (одно или несколько), типы которых хотим обобщить. Если данных насколько, решаем, связаны их типы или нет. В зависимости от этого выделяем один или несколько параметризованных (обобщенных, формальных) типов.
- ✓ Придумываем имена параметризованных типов.
- ✓ Пишем текст шаблона функции, по известным правилам написания функции, используя наряду со стандартными типами параметризованные при объявлении параметров, типа функции и вспомогательных (локальных) переменных.
- ✓ При вызове шаблона функции типы передаваемых аргументов уже известны. Используем их для явной передачи в функцию-шаблон.

При компиляции программы с шаблонами функций компилятор, «глядя» на оператор вызова функции, *заменяет* соответствующий параметризованный тип на стандартный – в результате получается полноценный экземпляр функции, который и компилируется.

Если функция вызывается *несколько раз* с разными типами, то формируются *несколько экземпляров* функций с *одинаковыми именами, но с разным списком параметров* – в языке C++ это возможно и называется *перегрузкой* функций. Компилятор по списку аргументов в операторе вызова функции однозначно определяет, какая из перегруженных функций вызвана.

Схематично построение шаблонов функций можно изобразить так:

```
//перечень параметризованных типов T1,T2
template<class T1,class T2>
bool Func(T1 a,T2 b)                //T1 и T2 используем как имена типов
{
        T1 c;          //вспомогательная переменная типа T1
        bool d;        //переменная для результата стандартного типа
        …
        return d;
}
int main(void)
{
        int A;
        bool Result;
        double B;
        cin>>A>>B;
        Result=Func<int,double>(A,B); //int соответствует T1,double -T2
        cout<<Result<<endl;
}
```

Замечание. Очередность типов при вызове функции (<**int,double**>) соответствует очередности параметризованных типов в template (<**class T1,class T2**>).

Замечание. Не ищите закономерности или правил в чередовании стандартных и параметризованных типов в схеме. Все зависит от алгоритма функции.

Особенность построения проекта с шаблонами функций:

- ✓ Объявление (прототип) функции-шаблона не пишем;
- ✓ Определение (текст) функции-шаблона помещаем в *заголовочный файл*;
- ✓ Заголовочный файл подключаем к вызывающей функции (обычно main()) директивой #include.

Порядок написания программы

Выполняем задание с использованием готового текста проекта по теме «Функции и матрицы» (Лабораторная работа 9). Превратим каждую функцию в шаблон. Ставим цель: хотим, чтобы каждая функция выполнялась для *матрицы любого (числового) типа*.

1. Внимательно прочитать условие задачи и формулировку функций.
2. Для каждой функции:

Понять, какое данное мы хотим обобщить (очевидно, это тип матрицы);

Заменить конкретный тип матрицы параметризованным типом;

Определить, какие данные в функции связаны с матрицей по типу, и у них также заменить тип параметризованным.

Замечание. В функции можно, при необходимости, вводить несколько параметризованных типов.

3. Правильно оформить тексты шаблонов функций. Помните, что алгоритмы функций *не меняются*.
4. Написать функцию main() с вызовом шаблонов функций для матриц целого и вещественного типа (убедиться в правильности работы шаблона).
6. Создать проект, учитывая его особенности с использованием шаблонов функций.
7. Выполнить проект.

Примеры написания программ

Задача. Написать программу с шаблоном функции вывода матрицы на экран монитора. Протестируем шаблон путем вывода на экран монитора матриц целого и вещественного типа.

Анализ решения задачи. Функция была нами написана ранее (см. Лабораторная работа 9, Задача 1).

Ставим цель: хотим, чтобы функция выводила матрицу *любого* типа. Значит, *вводим параметризованный тип T*, который будет означать тип матрицы.

Вспомним характеристики функции:

- ✓ Параметры: указатель на тип матрицы, количество строк и количество столбцов. Количества от типа матрицы не зависят – они целого типа при любом типе матрицы. Значит, параметры шаблона будут следующими: const T*p, int n, int m;
- ✓ Тип функции void, так и останется. Ничего не вычисляем.
- ✓ В теле функции есть вспомогательные (локальные) переменные i и j – индексы строк и столбцов матрицы. Они всегда целые, не зависят от типа матрицы.

После этого, проанализировав *все переменные* функции, мы можем написать определение шаблона функции и функцию main() с ее вызовом.

Программа будет строиться в виде проекта из двух файлов:

- ✓ Shablon.h – определение шаблона функции
- ✓ Main.cpp – текст функции main()

Текст файла Shablon.h будет выглядеть так:

```
#ifndef  SHABLON_H
#define  SHABLON_H
template <class T>
void OutMatr(const T*p,int n,int m)
{
	int i, j;
	cout<<endl;
	for(i=0; i<n; i++)
	{
		for(j=0; j<m; j++)
			cout<<setw(8)<<p[i*m+j];
		cout<<endl;
	}
	return;
}
#endif
```

Текст файла Main.cpp будет выглядеть так:

```
#include<iostream>
using namespace std;
#include <windows.h>
#include "shablon.h"
int main(void)
{
	SetConsoleCP(1251);
```

```
    SetConsoleOutputCP(1251);
    int A[2][3]= {1,2,3,4,5,6};
    double B[3][3] = {1.5,2.6,3.7,4.8};
    cout << "\n Целый массив:\n";
    OutMatr<int>(A[0],2,3);
    cout << "\n Дробный массив:\n";
    OutMatr<double>(B[0],3,3);
    return 0;
}
```

Порядок выполнения задания

1. Получить задание у преподавателя.
2. По вышеуказанному плану написать программу.
3. Ввести программу в компьютер.
4. Компилируя программу, найти и исправить ошибки синтаксиса.
5. Выполнить программу с исходными данными из контрольного примера, убедиться в правильности решения задачи.
6. Предъявить правильно работающую программу и контрольный пример для проверки преподавателю.

Контрольные вопросы и задания

1. Когда вместо обычных функций можно писать шаблоны?
2. Что такое параметризованный тип?
3. Могут ли быть параметризованными тип функции-шаблона, параметры, локальные переменные?
4. Как определить, какое данное должно выражаться параметризованным типом?
5. Сколько параметризованных типов может быть у шаблона функции?
6. Могут ли присутствовать стандартные типы в шаблонах функции?
7. Как компилятор обрабатывает шаблон функции?
8. Как размещаются шаблоны функций в файлах проекта?
9. Что такое перегрузка функций? Как она используется в теме «Шаблоны функций»?
10. Для наилучшего освоения темы замените обычные функции шаблонами в ранее выполненных работах (там, где это имеет смысл).

Задание

Выполнить задание с использованием готового текста проекта по теме «Функции и матрицы» (Лабораторная работа 9). Превратить каждую функцию проекта в шаблон. Цель: хотим, чтобы каждая функция выполнялась для матрицы любого (числового) типа.

Лабораторная работа 11

ПАРАМЕТР ФУНКЦИИ – УКАЗАТЕЛЬ НА ФУНКЦИЮ

Цель работы

Научиться писать функции с использованием параметра – указателя на функцию, в том числе:

– писать программу, состоящую из нескольких функций;

– уметь определить, объявить и вызвать функцию указанного типа;

– уметь размещать текст программы в файлах проекта в соответствии с парадигмой модульного программирования.

О чем необходимо помнить?

Изучить основные положения теории.

Рассмотрим построение функции с параметром – указателем на другую функцию.

Пусть мы хотим построить функцию, вычисляющую интеграл от некой подынтегральной функции с помощью простейшего численного метода – методом левых прямоугольников. Алгоритм строится достаточно просто, но подынтегральная функция может быть разной. Нельзя ли передать в качестве исходного данного (параметра) функцию? Оказывается можно. *Функция*, как и одномерный массив, *передается с помощью ее адреса,* следовательно, параметр должен быть указателем на функцию.

Как объявить указатель на функцию? Важны характеристики функции, а именно: тип функции, количество, типы и порядок следования параметров. Следовательно, объявляем указатель на функцию определенного вида. Например:

int **(*F1)**(int,double);

F1 – указатель на функцию целого типа с параметрами int и double.

double **(*F)**(double);

F – указатель на функцию вещественного типа с параметром double.

К слову сказать, такой указатель подойдет для передачи в функцию вычисления интеграла подынтегральной функции.

Как передать в качестве аргумента адрес функции? Все просто – как и для одномерного массива, *имя функции – это ее адрес*. Поэтому

F=sin; //стандартная функция sin() имеет те же характеристики, что и F

Если желаемой стандартной функции нет – можно написать ее самим, важно, чтобы она обладала заданными характеристиками.

Как вызвать функцию с использованием указателя, инициализированным адресом конкретной функции?

double result=(*F)(3.1415926);

или result=F(a); //a – аргумент функции

Можно приступать к написанию примеров.

Порядок написания программы

1. Внимательно прочитать условие задачи и формулировку функции.

2. Выделить функции пользователя и провести анализ характеристик каждой функции в отдельности:

Дать название функции;

Выделить список исходных данных функции, определить их типы;

Определить тип результата функции, передаваемого с помощью оператора return.

Определить тип дополнительных (если они будут) результатов функции, передаваемых с помощью параметров-ссылок или параметров-указателей.

Написать текст функции пользователя.

Написать прототип функции.

3. Написать функцию main() с вызовом функций пользователя.

4. Создать проект из двух файлов с расширением cpp (для функций пользователя и main()), и одного заголовочного файла с расширением h.

5. Выполнить проект.

Примеры написания программ

Задача 1. Написать программу с функцией, вычисляющей интеграл от подынтегральной функции заданного вида (функция вычисляет вещественное значение от вещественного аргумента).

Выделим функции:

1. Вычисление интеграла.
2. Подынтегральная функция, например x^2+1.

Анализ решения задачи. Необходимо написать две функции.

Для определения характеристик каждой функции ответим на три вопроса.

Функция вычисления интеграла:

- ✓ Имя функции – Integral
- ✓ Список параметров функции (исходные данные):

  Указатель на подынтегральную функцию – double **(*F)**(double);

  Нижний предел интегрирования – double a;

  Верхний предел интегрирования – double b;

  Количество разбиений отрезка [a;b]– int n (чем больше, тем точнее результат).
- ✓ Тип функции (тип возвращаемого результата) - double

Подынтегральная функция (x^2+1):

- ✓ Имя функции – MyFunc
- ✓ Список параметров функции (исходные данные) – double x;
- ✓ Тип функции (тип возвращаемого результата) – double.

Помним, что все имена (функции, параметров, аргументов) должны быть «говорящими». Это способствует лучшему пониманию программы.

После этого мы можем написать определения функций, их объявления и функцию main() с вызовом функций пользователя. Поскольку функции тесно связаны между собой, имеет смысл поместить их в один файл. Файл с объявлениями функций тоже будет один.

Программа будет строиться в виде проекта из трех файлов:

- ✓ Integral.cpp – определения функций пользователя;
- ✓ Integral.h – объявления функций пользователя;
- ✓ Main.cpp – текст функции main().

Текст файла Integral.cpp будет выглядеть так:

```
double Integral(double (*F)(double), double a,double b,int n)
	{
		int i;
		double h=(b-a)/n, result=0;
		for(i=0; i<n; i++)
			result+=(*F)(a+i*h);
		return h*result;
	}
double MyFunc(double x)
	{
		return x*x+1;
	}
```

Текст файла Integral.h будет выглядеть так:

```
#ifndef  INTEGRAL_H
#define  INTEGRAL_H
double Integral(double (*F)(double), double a, double b,int n);
double MyFunc(double x);
#endif
```

Текст файла Main.cpp будет выглядеть так:

```
#include<iostream>		//подключение системных средств для
using namespace std;		//возможности использовать потоки ввода-вывода
#include <windows.h>		//прототип функций русификации
#include "Integral.h"		//прототипы функций пользователя
int main(void)			//имя функции main()
{
	SetConsoleCP(1251);		//вызов функций русификации
	SetConsoleOutputCP(1251);
	double a, b, res;
	int n ;
	cout<< "\nВведите a, b, n: ";
	cin>>a>>b>>n;
```

```
    res=Integral(MyFunc,a, b, n);
    cout<< "\nИнтеграл равен "<<res<<endl;
    return 0;
}
```

Задача 2. Написать программу сортировки по возрастанию одномерного целочисленного массива с использованием стандартной функции qsort().

Анализ решения задачи. Функция qsort() объявляется в stdlib.h так:

```
void qsort(void*p,int N,int width,int (*fcmp)(const void*,const void*));
```

Прекрасный финальный пример!

Функция сортирует N элементов одномерного массива *любого* типа, начиная с адреса p.

Поскольку тип массива заранее неизвестен, адрес массива p - указатель на тип void, width – размер одного элемента в байтах.

Последний параметр fcmp – указатель на функцию заданного вида – наша тема!

Функция сравнивает два элемента массива, опираясь на их адреса (параметр – const void* – тип элементов неизвестен). Результат функции – равен 1, если первый элемент>второго;

-1, если первый элемент<второго;

0 – при равенстве.

Массив будет отсортирован по возрастанию. Для сортировки по убыванию поменяйте местами 1 и -1.

Функцию сравнения пишем сами, поскольку только мы знаем, какого типа массив.

Для написания программы нам потребуются ранее написанные функции (Лабораторная работа 8) InMas() и OutMas().

Программа будет строиться в виде проекта из трех файлов:

- ✓ Mas.cpp – готов;
- ✓ Mas.h – готов;
- ✓ Main.cpp – текст функции main().

Текст файла Main.cpp будет выглядеть так:

```
#include<iostream>
using namespace std;
#include<stdlib.h>
#include<time.h>
#include <windows.h>
#include "Mas.h"
int MyFcmp(const void*p1,const void*p2)
{
    int *q1=(int*)p1,*q2=(int*)p2;  //указатели приводятся к типу int
    if(*q1>*q2) return 1;           //сравнение двух элементов массива
```

```
        else if(*q1<*q2) return -1;
        else return 0;
}
int main(void)
{
        SetConsoleCP(1251);
        SetConsoleOutputCP(1251);
        const int SIZE=10;
        int mas[SIZE];
        srand((unsigned)time(NULL));
        InMas(mas,SIZE,100);
        cout<<"Исходный массив: "<<endl;
        OutMas(mas,SIZE);
        qsort( mas,SIZE,sizeof(int),MyFcmp);
        cout<< "Отсортированный массив:"<<endl;
        OutMas(mas, SIZE);
        return 0;
}
```

Задание. Превратите функцию MyFcmp() в шаблон. Перепишите программу, отсортируйте массивы разного типа.

Порядок выполнения задания

1. Получить задание у преподавателя.
2. По вышеуказанному плану написать программу.
3. Ввести программу в компьютер.
4. Компилируя программу, найти и исправить ошибки синтаксиса.
5. Выполнить программу с исходными данными из контрольного примера, убедиться в правильности решения задачи.
6. Предъявить правильно работающую программу и контрольный пример для проверки преподавателю.

Контрольные вопросы и задания

3. Когда в функции используется параметр – указатель на функцию?
4. Как объявить указатель на функцию?
3. Как поместить в указатель на функцию конкретный адрес?
4. Как определить адрес у конкретной функции?
5. Определите адрес у функции InMas()?
6. Объявите указатель и поместите в него адрес InMas().
7. Как вызывается функция с использованием указателя на функцию?
8. Для наилучшего освоения темы выполните варианты 1, 10, 15.

Задание

Написать программу сортировки указанного массива функцией qsort(), используя при необходимости, ранее написанные проекты Лабораторных работ 8, 9.

Вариант 1. Написать программу сортировки статической строки по убыванию.

Вариант 2. Написать программу сортировки динамической строки по убыванию.

Вариант 3. Написать программу сортировки статической строки по возрастанию.

Вариант 4. Написать программу сортировки динамической строки по возрастанию.

Вариант 5. Написать программу сортировки последних N элементов динамического целого одномерного массива по убыванию.

Вариант 6. Написать программу сортировки K первых элементов статического вещественного одномерного массива по убыванию.

Вариант 7. Написать программу сортировки статической вещественной матрицы по возрастанию.

Вариант 8. Написать программу сортировки динамической целой матрицы по убыванию.

Вариант 9. Написать программу сортировки N-ой строки статической вещественной матрицы по убыванию.

Вариант 10. Написать программу сортировки N-ой строки динамической целой матрицы по возрастанию.

Вариант 11. Написать программу сортировки K подряд идущих строк, начиная с N-ой строки статической вещественной матрицы по убыванию.

Вариант 12. Написать программу сортировки K подряд идущих строк, начиная с N-ой строки динамической целой матрицы по возрастанию.

Вариант 13. Написать программу сортировки двух последних строк статической вещественной матрицы по убыванию.

Вариант 14. Написать программу сортировки двух последних строк динамической целой матрицы по возрастанию

Вариант 15. Написать программу сортировки с одного элемента до другого в статической целой матрице по убыванию. Элементы задаются индексами.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Программируем на С++ : лабораторный практикум : в 3 ч. Ч. 1. Операторы / авт.-сост. : О. А. Филимонова ; Сиб. гос. аэрокосмич. ун-т. – Красноярск, 2011. – 44 с.

2. Гуменникова, А.В. Программирование на языке С++ в среде Microsoft Visual Studio : учеб. пособие. В 2 ч. Ч.1 / А.В. Гуменникова, О.А. Орешкина, А.Г.Зотин ; Сиб. гос. аэрокосмич. ун-т. – Красноярск, 2014. – 192 с.

3. Павловская, Т. А. Программирование на языке высокого уровня / Т. А. Павловская. – СПб. : Питер, 2009. – 464 с.

4. Щупак, Ю. А. С/С++. Структурное и объектно-ориентированное программирование : практикум / Ю. А. Щупак, Т. А. Павловская. – СПб. : Питер, 2011. – 352 с.

5. Васильев, А. Самоучитель С++ с примерами и задачами / А. Васильев. – СПб. : Наука и техника, 2010. – 480 с.

6. Либерти, Дж. Освой самостоятельно С++ за 21 день / Дж. Либерти, Б. Джонс. – 5-е изд. ; пер. с англ. – М. : Вильямс, 2007. – 784 с.

7. Мейерс, С. Эффективное использование С++. 50 рекомендаций по улучшению ваших программ и проектов : пер с англ. / С. Мейерс – М. : ДНК Пресс ; СПб. : Питер, 2006. – 240 с.

Учебно-практическое издание

ФИЛИМОНОВА Ольга Александровна

ПРОГРАММИРУЕМ НА С++

УКАЗАТЕЛИ. МАССИВЫ. ФУНКЦИИ

Лабораторный практикум в 3 частях
Часть 2