Федеральное государственное бюджетное образовательное учреждение высшего образования
Казанский государственный энергетический университет
Кафедра информационных технологий и интеллектуальных систем

Массивы

Методические указания для выполнения лабораторных работ

Казань, 2024

Лабораторная работа. Массивы

Цель работы: приобрести навыки разбиения массивов задач на отдельные функциональные блоки и выделение их в функции, научиться описывать функции, правильно сопоставлять формальные и фактические параметры, использовать указатели и ссылки, формировать случайные числа в заданном диапазоне.

План лабораторной работы

1. Индексированные массивы

2. Ассоциативные массивы

3. Многомерные массивы

1. Краткие теоретические сведения о модульном программировании. Правила программирования функций.

В предыдущих лабораторных работах программы были написаны единым, функционально неделимым, кодом. Алгоритм программы находился в главной функции, причём других функций, кроме стандартных, в программе не было. С увеличением объёма программы становится невозможно удерживать в памяти все детали. Поэтому единственным возможным способом *решения сложных задач* является их разбиение на части. При программировании на C++ подобные задачи могут быть разделены на более простые и обозримые с помощью *функций.* Этот приём позволяет рассматривать программу в более укрупнённом виде - на уровне взаимодействия функций. Применение функций является первым шагом к повышению степени абстрактизации программы и ведёт к *упрощению* её структуры.

В случае разбиения задачи на отдельные модули программа будет состоять из *отдельных фрагментов кода*, которые выделены в функции. Работа отдельной функции не зависит от работы какой-нибудь другой. То есть алгоритм в каждой функции функционально достаточен и не зависим от других алгоритмов программы. Однажды написав функцию, её можно будет с лёгкостью переносить в другие программы.

Функция (в программировании) — это именованная последовательность описаний и операторов, выполняющих какое-либо законченное действие. Функция может принимать параметры и возвращать значение или выполнять определенные, как правило, часто повторяющиеся действия. Итак, функции

позволяют сделать программу модульной, то есть разделить программу на несколько маленьких подпрограмм (функций), которые в совокупности выполняют поставленную задачу. Еще один огромнейший плюс функций в том, что их можно многократно использовать. Данная возможность позволяет многократно использовать один раз написанный код, что в свою очередь, намного сокращает объем кода программы!

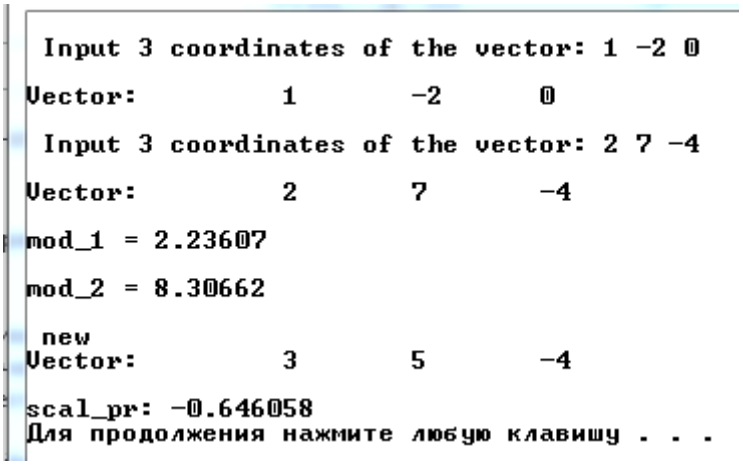
Проиллюстрируем всё сказанное выше примерами.

Пример 1. Даны два вектора с координатами {1,-2,0}, {2, 7,-4}. Найти модуль каждого вектора, сумму векторов и их скалярное произведение.

В рамках процедурного программирования (только одна, главная, функция) решение задачи будет выглядеть следующим образом:

```
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
     int  a,b,c,  x,y,z,  u,v,w;//переменные  для  координат3-х
векторов
     float mod_1, mod_2, scal_pr;// модули векторов и их скалярное
произведение
     cout<<" Input 3 coordinates of the vector:";cin >>a>>b>>c;
    cout<< "Vector: "<<'\t'<<a<<'\t'<<b<<'\t'<<c<<endl;
    cout<<" Input 3 coordinates of the vector:";cin >>x>>y>>z;
    cout<< "Vector: "<<'\t'<<x<<'\t'<<y<<'\t'<<z<<endl;

     mod_1= sqrt(float(a*a+b*b+c*c));// модуль первого вектора
     mod_2= sqrt(float(x*x+y*y+z*z));// модуль второго вектора
     cout<<"\nmod_1 = "<< mod_1<<endl;
     cout<<"\nmod_2 = "<< mod_2<<endl;

     u=a+x; v=b+y;w= c+z;// определение суммы векторов
    cout<<"\n new";
    cout<< "Vector: "<<'\t'<<u<<'\t'<<v<<'\t'<<w<<endl;


     scal_pr=(a*x+b*y+c*z)/mod_1/mod_2;
    cout<< "\nscal_pr: "<<scal_pr<<endl;
system("pause");return 0;
}
```

Решение задачи имеет следующий вид:



А вот аналогичный пример с функциями:

```
#include <iostream>
#include <cmath>
using namespace std;
 void inp_vect(int & a, int & b, int & c)// функция для ввода вектора с консоли
 {cout<<"\n Input 3 coordinates of the vector:";cin >>a>>b>>c;}
 void print(int a, int b, int c)// функция для вывода вектора на консоль
 {cout<< "\nVector: "<<'\t'<<a<<'\t'<<b<<'\t'<<c<<endl;}
 float modul(int a, int b, int c)// функция     вычисления модуля вектора
 {return sqrt(float(a*a+b*b+c*c));}
 int main()
{
     int  a,b,c,  x,y,z,  u,v,w;//переменные  для  координат3-х векторов
     float mod_1, mod_2, scal_pr;// модули векторов и их скалярное произведение

     inp_vect(a,b,c); print(a,b,c);// ввод и выод векторов
     inp_vect(x,y,z); print(x,y,z);

     mod_1= modul(a,b,c);// модуль первого вектора
     mod_2= modul(x,y,z);// модуль первого вектора

     cout<<"\nmod_1 = "<< mod_1<<endl;
     cout<<"\nmod_2 = "<< mod_2<<endl;

     u=a+x; v=b+y;w= c+z;// определение суммы векторов
    cout<<"\n new "; print(u,v,w);

     scal_pr=(a*x+b*y+c*z)/(modul(a,b,c)*modul(x,y,z));// обращение к функции внутри вычисляющегооператора
```

```
      cout<< "\nscal_pr: "<<scal_pr<<endl;
system("pause");
      return 0;
}
```

По сути, после компиляции не будет *никакой разницы для процессора*, как для первого кода, так и для второго. Однако программисту после создания отдельных функций - для ввода/вывода вектора, вычисления его модуля - код главной программы стал,

во-первых, более читабельным,

во-вторых, при необходимости изменить формат вывода на консоль достаточно сделать изменения лишь в функции, и эти изменения распространятся на вывод любого количества векторов,

в-третьих, при вычислении скалярного произведения нет необходимости заранее вычислять модуль вектора и записывать его значение в отдельную переменную: при наличии функции `modul`, её можно включить непосредственно в математическое выражение так, как это делается обычно со стандартными функциями.

Таким образом, любая программа на C++ состоит из функций, одна из которых должна иметь имя `main` (с нее начинается выполнение программы). Функция начинает выполняться в момент вызова.

Свойства функций можно описать следующим образом:

1. Любая функция имеет тип, также, как и любая переменная.
2. Функция может возвращать значение, тип которого в большинстве случаев аналогичен типу самой функции.
3. Если функция не возвращает никакого значения, то она должна иметь тип `void` (пусто) - такие функции иногда называют процедурами.
4. При объявлении функции, после ее типа должно находиться имя функции и две круглые скобки — открывающая и закрывающая, внутри которых могут находиться один или несколько аргументов (параметров) функции, которых также может не быть вообще:
5. После списка аргументов функции ставится открывающая фигурная скобка, после которой находится само тело функции.
6. В конце тела функции обязательно ставится закрывающая фигурная скобка.

Сказанное в пунктах 4 - 6 означает, что формат описания функции следующий:

```
Тип_функции Имя_функции ( [Список_параметров ])
{
Операторы;      // Тело_функции
      return [значение];}
```

Здесь *Тип_функции* определяет тип величины, возвращаемого функцией. Функция может возвращать любой тип за исключением массива. Если функция ничего не возвращает, то тип возврата должен быть `void` – пустой.

Имя_функции – любой допустимый идентификатор.

Список параметров представляет собой *последовательность пар типов* и *идентификаторов*, разделяемых запятыми.

Параметры – это переменные, которые получают значения аргументов, передаваемых функции при ее вызове. Если функция не требует параметров, то список параметров будет пуст.

Фигурные скобки окружают *тело функции*. Тело функции состоит из операторов, определяющих, что именно эта функция делает.

При написании функции очень важно понять различие между ***формальными*** *и* ***фактическими*** параметрами**.**

Пример 2. Допустим определена простейшая функция для сложения трёх целочисленных значений и реализован её вызов в главной программе

Программа реализующая описанный алгоритм:

```
1  #include<iostream>
2  using namespace std;
3  int sum(int a, int b, int c)
4  {
5   int result;
6       result = a + b + c;
7       return result;
8  }
9  int main()
10 {
11      int k=1, l=-5, m=-2, s;
12      s=sum(k,l,m);
13      cout<<"s = "<<s<< endl;
14      cout<<"s = "<<sum(2,-5,0)<<endl;
15      system("pause");
16      return 0;
```

Аргументы ***a***, ***b*** и ***c*** называются *формальными* параметрами. Это переменные, которые определены в теле функции (т.е. к ним можно обращаться только *внутри фигурных скобок*).

При написании определения функции программа не знает их значения. При вызове функции вместо них подставляются фактические параметры – значения, с которыми функция вызывается.

Фактическими параметрами (или фактическими аргументами) являются значения переменных ***k, l*** и ***m***, которые описаны и инициализированы в главной программе

Формальные параметры ***a***, ***b*** и ***c*** принимают значения фактических аргументов, заданных при вызове, и функция выполняется.

Аналогично, в *Примере 1* в трёх описанных функциях используются формальные параметры ***a***, ***b*** и ***c***, а вызов этих функций реализуется несколько раз с разными фактическими значениями, это и ***a***, ***b*** и ***c***, описанные уже в `main`, и ***x, y, z,*** и ***u, v, w.***

Таким образом, ***передача значений*** в функцию реализуется через сопоставление формальных и фактических параметров. Передавать можно, как видно из предыдущих примеров, один или несколько параметров.

Возврат значений из функции может быть реализован по-разному. Самым простым вариантом является ***возврат одного значения***. Именно этот вариант и реализуют приведенные выше примеры: в операторе `return` указывается возвращаемое значение (имя вычисленной переменной или вычисляющее выражение).

В вызывающей программе функции с одним возвращаемым значениям, как правило, ставятся справа от знака присваивания (оператор 12, *Примера* 2), а также могут быть непосредственно включены в оператор вывода (оператор 14) или в арифметическое выражение, как это сделано при вычислении скалярного произведения *Примера 1*.

Прототипы функций. Любая функция должна быть объявлена и определена. Как и для других величин, объявлений может быть несколько, а определение только одно. Объявление функции должно находиться в тексте раньше ее вызова для того, чтобы компилятор мог осуществить проверку правильности вызова.

Правилом хорошего тона, считается до главной программы сделать объявление функции (***прототип***), которое задает ее имя, тип возвращаемого значения и список передаваемых параметров. А определение функции, которое содержит, кроме объявления, тело функции, записанное в фигурных скобках, можно вынести либо после главной программы, либо вообще в другой файл. Прототип включает в себя только заголовок функции с её параметрами и закачивается точкой с запятой после круглой скобки:

```
Тип_функции Имя_функции ( [Список_параметров ]);
```

В определении, в прототипе и при вызове одной и той же функции типы и порядок следования параметров должны совпадать. На имена параметров *ограничений по соответствию не накладывается*, поскольку функцию можно вызывать с различными аргументами, а в прототипах имена компилятором игнорируются (они служат только для улучшения читаемости программы).

Пример 3. Составить программу, которая по выбору номера варианта

1: вычисляет периметр треугольника по трем сторонам

2: находит площадь треугольника по трём сторонам

3: находит высоту треугольника на сторону с номером i

Программа должна работать, если треугольник с заданными сторонами *a, b, c* существует.

Словесный алгоритм этой задачи состоит из следующих шагов:

1. Определяем математическое условие для проверки существования треугольника: треугольник со сторонами *a, b, c* существует, если сумма любых двух сторон больше третьей стороны: $a+b>c$; $a+c>b$; $c+b>a$. Если хотя бы одно условие не выполняется, программа должна выдать соответствующее сообщение и прекратить работу, в противном случае треугольник существует и возможно продолжение решения задачи.
2. Вводим номер *Nvar* необходимого варианта.
3. В случае, если *Nvar = 1*, то идём к пункту 4, если *Nvar = 2*, то идём к пункту 5, если *Nvar = 3*, то идём к пункту 6, во всех остальных случаях выдаём сообщение, что номер варианта задан неверно.
4. Вычисляем периметр треугольника $p=a+b+c$. Выводим результат на печать. Завершаем работу программы.
5. Вычисляем площадь треугольника по теореме Герона: $S=\sqrt{\bar{p}\cdot(\bar{p}-a)\cdot(\bar{p}-b)\cdot(\bar{p}-c)}$. Здесь $\bar{p}=p/2$. Выводим результат на печать. Завершаем работу программы.
6. Вводим номер стороны *i*, на которую требуется найти высоту, договариваясь, что $i=1=a$, $i=2=b$, $i=3=c$. Можно вводить и символьные значения, но пока мы не рассматривали правила работы с символами, поэтому ограничимся работой с вариантами чисел. Выбираем соответствующую номеру формулу для высоты:

$$h_i=\frac{2\cdot S}{i}$$

Выводим результат. Завершаем работу программы.

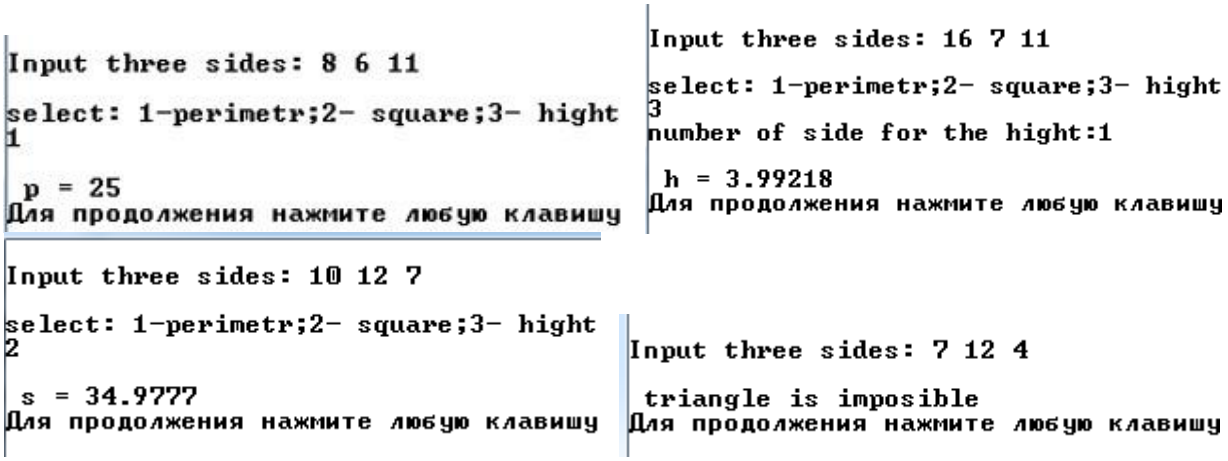
7. Завершаем работу программы, если не один из вариантов не был выбран.

Текст программы, реализующий данный алгоритм, приведён ниже.

```
#include<iostream>
#include<cmath>
using namespace std;
// прототипы функций
bool exist_triangle(float a,float b,float c);// проверка на
существование тр-ка
float perimetr(float a,float b,float c);// расчет периметра
float square(float a,float b,float c);// расчёи площади
float hight(float a,float b,float c);//рассчёт высоты
```

```
int main()
{float a,b,c;
cout<< "\nInput three sides: "; cin>> a>>b>>c;
bool test=1;
float p, s,h;
int var, i;
test=exist_triangle(a,b,c);
if (!test)
     { cout<<"\n triangle is imposible\n";  system
("pause");return 1;}
cout<< "\nselect: 1-perimetr;2- square;3- hight\n";
cin>> var;
// выбор варианта решения
switch (var)
{case 1: {p=perimetr(a,b,c); cout<<" \n p = "<<p<<endl;};break;
case 2: {s=square(a,b,c); cout<<" \n s = "<<s<<endl;};break;
case 3: {h=hight(a,b,c); cout<<" \n h = "<<h<<endl;};break;
default:{cout<<"variant of calculation is not
selected\n";break;}
}
system("pause");
return 0;
}
//Определение функций
bool exist_triangle(float a,float b,float c)
{if (a>(b+c) || b>(a+c) || c>(a+b))return false; else return
true;
}
float perimetr(float a,float b,float c)
{return a+b+c;}
float square(float a,float b,float c)
{
float p=perimetr(a,b,c)/2;
return sqrt(p*(p-a)*(p-b)*(p-c));
}
float hight(float a,float b,float c)//высота по номеру стороны i
{float h;int i;
cout<<"number of side for the hight:"; cin>> i;
switch (i)
{    case 1: h=2*square(a,b,c)/a;break;
     case 2: h=2*square(a,b,c)/b;break;
     case 3: h=2*square(a,b,c)/c; break;
     default:{cout<<"incorrect number of a side\n"; h=0;};break;
}
return h;
}
```

Решение задачи с проверкой полноты всех вариантов выбора представлено ниже:



Анализ программы. В представленном варианте решения описание четырёх функций выделено в прототипы. Все функции возвращают одно значение, в том числе и логического типа `bool`. Поставленная задача хорошо решается с применением оператор выбора `switch`.

Если из функции необходимо ***вернуть несколько значений***, как например, в `inp_vect()`, в теле которой предусмотрен ввод значений на три формальных параметра, то для этого необходимо реализовывать передачу значений в/из функции с помощью новых элементов языка C^{++} - *указателей и ссылок*.

Указатели. Когда компилятор обрабатывает оператор определения переменной, например `int i = 10`; он выделяет память в соответствии с типом (`int`) и инициализирует её указанным значением (10). Все обращения к переменной по её имени (`i`) *заменяются компилятором на адрес области памяти*, в которой хранится значение переменной. Программист может определить собственные переменные для хранения адресов областей памяти. Такие переменные называются *указателями.*

Указатели, следовательно, *предназначены для хранения адресов областей памяти.* В C^{++} различают три вида указателей – указатели на объект, на функцию и на `void`, отличающиеся свойствами и набором допустимых значений.

Рассмотрим вначале простейшее объявление указателя *на объект:*

```
Тип *Имя;
```

где `Тип` может быть любым (кроме ссылки и битового поля), причём тип может быть к этому моменту объявлен, но не определён.

Звёздочка относится непосредственно к имени, поэтому для того, чтобы объявить несколько указателей, требуется ставить её перед именем каждого из них:

```
int *a, *b, *c;
```

Размер указателя зависит от модели памяти, можно определять указатель на указатель и т.д.

Указатель может быть константой или переменной, а также указывать на константу или переменную. Например:

```
int 1;                          // целая переменная
const int ci = 1;               // целая константа
int * p i ;                     // указатель на целую переменную
const int * pci;                // указатель на целую константу
int * const cp = &i;            // указатель-константа на целую переменную
const int * const cpc = &ci;    // указатель-константа на целую константу
```

С указателями можно выполнять *следующие операции*:

- *разыменование* (*) – получение значения величины, адрес которой хранится в указателе;
- взятие адреса (&);
- присваивание;
- арифметические операции
- сложение указателя только с константой,
- вычитание: допускается разность указателей и разность указателя и константы,
- инкремент (++) увеличивает значение указателя на величину `sizeof(тип)`;
- декремент (--) уменьшает значение указателя на величину `sizeof(тип)`;
- сравнение;
- приведение типов.

Краткие итоги

· Для *экономии памяти и времени*, затрачиваемого на обращение к данным, в программах используют указатели на объекты.

· Указатель не является самостоятельным типом, он всегда связан с другим типом.

· Указатель может быть константой или переменной, а также указывать на константу или переменную.

· Указатель типа void указывает на область памяти любого размера. Разыменование такого указателя необходимо проводить с операцией приведения типов.

· До первого использования в программе объявленный указатель необходимо проинициализировать.

· Над указателями определены операции: разыменование, взятие адреса, декремент, инкремент, увеличение (уменьшение) на целую константу, разность, определение размера.

· Над указателями определены операции сравнения.

Ссылки представляют собой *синоним имени*, указанного при инициализации ссылки. Ссылку можно рассматривать как указатель, который всегда *разыменовывается*. Формат объявления ссылки:

```
Тип & Имя;
```

где **Тип** - это тип величины, на которую указывает ссылка, & — оператор ссылки, означающий, что следующее за ним ***Имя*** является именем переменной ссылочного типа, например:

```
int kol:
int & pal = kol;            // ссылка pal - альтернативное имя
                            для kol
const char& CR = ' \ n ';   // ссылка на константу
```

Запомните следующие правила.

• Переменная-ссылка должна явно инициализироваться при ее описании, кроме некоторых случаев, когда она является, например, параметром функции.

• После инициализации ссылке не может быть присвоена другая переменная.

• Тип ссылки должен совпадать с типом величины, на которую она ссылается.

• Не разрешается определять указатели на ссылки, создавать массивы ссылок и ссылки на ссылки.

Ссылки применяются чаще всего в качестве параметров функций и типов возвращаемых функциями значений. Ссылки позволяют использовать в функциях переменные, передаваемые по адресу, без операции разадресации, что улучшает читаемость программы.

Ссылка, в отличие от указателя, не занимает дополнительного пространства в памяти и является просто другим именем величины. Операция над ссылкой приводит к изменению величины, на которую она ссылается.

Примеры задач на применение ссылок и указателей

Пример 4. Рассмотрим пример на применение *операции разыменования,* что означает, что по адресу переменной в памяти мы переходим к действиям над значением, хранимом по данному адресу. Это операция и называется разыменованием.

```
#include <iostream>
using namespace std;
```

```
int main()
{
    int  i_val = 7;
    int* i_ptr = &i_val;//Используя унарную операцию
                        //взятия  адреса  &,  мы  извлекаем  адрес
переменной
                        //i_val и присваиваем ее указателю.

    // выведем на экран значение переменной i_val
    cout <<  i_val << endl; //используем саму переменную
    cout << *i_ptr << endl; //обращаемся к значению переменной
                        // i_val через указатель: здесь используется
                        //операция разыменования:
                        //она позволяет перейти от адреса к значению.
       system("pause");
  return 0;
}
```

Таким образом, к любой переменной мы можем обратиться как по её имени, так и по её адресу, применив к нему операцию разыменования.

Справедливо тождество: выражение a == *(&a), где a любого типа — всегда истинно.

Пример 5. Перестроим решение задачи про треугольник (*Пример 3)* таким образом, чтобы все необходимые параметры треугольника – периметр, площадь и высоту - получать сразу в одной функции. Условие проверки существования треугольника и вариативные действия в зависимости от выполнения этого условия так же внесём в новую функцию.

Текст этой программы

```
#include<iostream>
#include<cmath>
using namespace std;
// прототипы функций
void triangle(float af,float bf,float cf, float & pr, float &sq,
int in, float &hi);//функция определения параметров треугольника
по его трём сторонам
int main()
{
float a,b,c;
cout<< "\nInput three sides: "; cin>> a>>b>>c;
bool test=1;
float p, s,h;
int number;// номер  стороны для вычислени высоты
cout<<"number of side for the hight:"; cin>> number;
triangle(a,b,c,p,s,number,h);
cout<<" \n p = "<<p<<endl;
cout<<" \n s = "<<s<<endl;
cout<<" \n h = "<<h<<endl;
system("pause");
```

```
return 0;
}
//Определение функции
void triangle(float a,float b,float c, float & p, float &s, int
i, float &h)
{
     if (a>(b+c) || b>(a+c) || c>(a+b))
{
     cout <<"triangle is impossible!"; system("pause"); exit(1);
}
p= a+b+c;

float p1=p/2;
s= sqrt(p1*(p1-a)*(p1-b)*(p1-c));

switch (i)
{    case 1: h=2*s/a;break;
     case 2: h=2*s/b;break;
     case 3: h=2*s/c; break;
     default:{cout<<"incorrect number of a side\n"; h=0;};break;
}
}
```

Решение задачи с проверкой полноты работоспособности всех вариантов:

```
Input three sides: 12 17 18
number of side for the hight:3
 p = 47
 s = 98.2926
 h = 10.9214
```

```
Input three sides: 12 37 15
number of side for the hight: 3
triangle is impossible!Для продолж
```

```
Input three sides: 12 18 17
number of side for the hight:4
incorrect number of a side
 p = 47
 s = 98.2926
 h = 0
```

Анализ программы. Для запуска функции `triangle` описаны и инициализированы переменные *a*, *b*, *c*, *number*, значения которых передаются в функцию. Для этого в оперативной памяти области функции выделяется место, куда копируются передаваемые значения. Такой способ передачи параметров называется ***передача по значению.*** Даже, если с этими переменными производить какие-то операции в теле `triangle`, изменяющие их значения, то измененные значения будут не доступны главной программе.

Для остальных параметров из списка (*p*, *s*, *h*) достаточно лишь описать их в вызывающей программе. ***Передача по ссылке*** (знак &) означает, что эти переменные не копируются в область памяти функции, а функция, в которую

их имена пересылается, работает с той областью памяти, которая выделена для них в главной программе. Поэтому любые изменения с этими переменными в теле `triangle` автоматически отражаются на переменных, описанных в `main`.

Имена формальных и фактических параметров, а также имена параметров в прототипе и при определении функции могут отличаться, а могут быть одинаковыми. Транслятор сам определяет, как работать с этими параметрами, главное, чтобы их количество во всех случаях (описание, определение, вызов) было одинаковым.

Примеры задач на формирование случайных чисел в заданном диапазон значений.

Для формирования чисел значением от 1 до *N*, достаточно взять *целочисленный остаток* от деления случайного целочисленного числа, формируемого функцией `rand()`, на число *N*, добавив 1. Нередко возникают ситуации, когда нужно задать не только верхний, но и нижний предел для формирования случайных чисел.

Если диапазон формируемых *целых чисел* лежит в интервале от заданных целых чисел *M* и *N* (*M* < *N*), то формула для определения случайного числа в диапазон будет следующая:

```
Сл.число = rand()%(N-M+1) + M;
```

Если формируется *вещественное число* в заданном диапазоне [-*M*; +*N*], то можно использовать следующую формулу:

```
Сл.число = N*rand()/RANDMAX + M*rand()/RANDMAX;
```

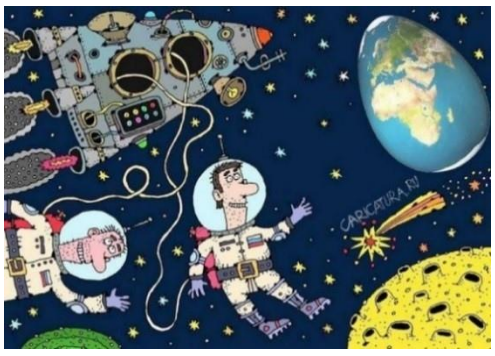
Пример 6.

Астронавты исследуют гравитационное поле $U(m,x,y)$ космического объекта сложной формы, задаваемого формулой:

$$U(m,x,y)=\varphi(m,x,y)\,\frac{\varphi\left(m,\sqrt{|x+y|},x\right)+\varphi(m^2,x+y,x-y)}{\varphi(m^2,x-y,x+y)};$$

где *функция массы*:

$$\varphi(m,x,y)=\begin{cases}\lg m\sqrt{\dfrac{x+y}{\sqrt[3]{x-y}}}, & \text{для } x>0,\ y>0,\ x>y;\\ \ln m\sqrt{\dfrac{x}{y}}, & \text{для } x>0,\ y>0;\\ \log_2 m, & \text{во всех остальных случаях.}\end{cases}$$



Они направляют к объекту зонд массой $m\in[10;15]$ в точку с координатами $x\in[-1;1]$, $y\in[-1;1]$. Определить значение потенциала и функции

массы $\varphi(m,x,y)$ в этой точке. Второй зонд массой $m_1 \in [10;15]$ запускают в другую точку $x_1 \in [-2;2], y_1 \in [-2;2]$. Вычислить функцию массы и в этой точке.

Словесный алгоритм главной программы:

1. Используя генератор случайных чисел, инициализируем в указанных диапазонах массу зонда $m \in [10;15]$ и координаты точки $x \in [-1;1], y \in [-1;1]$, куда зонд запускают.
2. Вычисляем $U(m,x,y)$ и $\varphi(m,x,y)$.
3. Выводим результаты на консоль.
4. Инициализируем массу $m_1 \in [10;15]$ другого зонда и координаты другой точки $x_1 \in [-2;2], y_1 \in [-2;2]$.
5. Вычисляем в ней функцию массы $\varphi(m_1,x_1,y_1)$.
6. Конец программы.

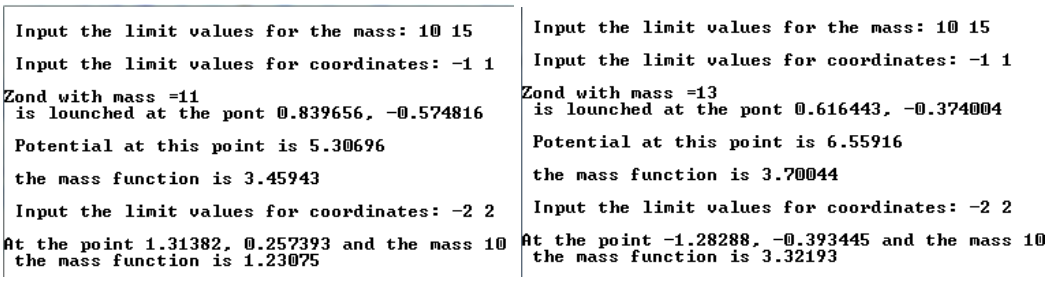
Для вычисления значений случайных чисел, потенциала $U(m,x,y)$ и функции массы $\varphi(m,x,y)$ строим отдельные функции.

Текст программы, реализующей решение поставленной задачи, следующий:

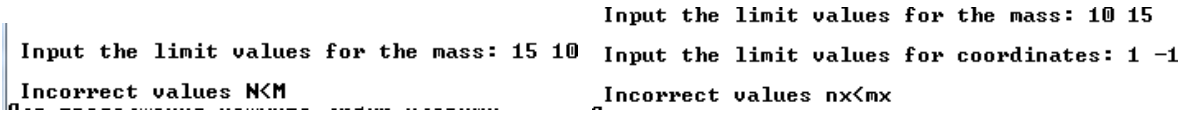
```
#include<iostream>
#include<cmath>
#include<ctime>
using namespace std;
// прототипы функций
float U(float m,float x,float y, float & f);//функция потенциала
float phi(float m,float x,float y);// функция массы
int rand_int(int M, int N);// формирование целых чисел в
диапазоне M<N
float rand_float(float M, float N);//формирование вещественных
чисел в диапазоне M<N
int main()
{
     srand(time(0));
     float m,x,y,fi;
     int N, M, mx,nx;
     cout<<"\n Input the limit values for the mass: "; cin
>>M>>N;
     m=float(rand_int(M,N)); //формирование массы
     cout<<"\n Input the limit values for coordinates: "; cin
>>mx>>nx;
     x=rand_float(mx,nx);//формирование  координат
     y=rand_float(mx,nx);//
     cout<<"\nZond with mass ="<<m<<"\n is lounched at the pont "
          <<x<<", "<<y<<endl;
cout<<"\n Potential at this point is "<<U(m,x,y,fi)<<endl;
cout<<"\n the mass function is "<<fi<<endl;
```

```
// переходим в другую точку
    cout<<"\n Input the limit values for coordinates: "; cin
>>mx>>nx;
     m=float(rand_int(M,N));
     x=rand_float(mx,nx);
     y=rand_float(mx,nx);
cout<<"\nAt the point "<<x<<", "<<y<<" and the mass "<<m<<"\n
the mass function is "<<phi(m,x,y)<<endl;
cout<<endl;
system("pause");
return 0;
}
float U(float m,float x,float y, float & f)
{
     f=phi(m,x,y);
     return f*(phi(m,sqrt(abs(x+y)),x)+phi(m*m,x+y,x-y))/
          phi(m*m,x-y,x+y);
}
float phi(float m,float x,float y)
{
     float f,p=1./3;
     if (x>0 && y>0)
          if(x>y) f=log10(m)*sqrt(x+y)/pow((x-y),p);
          else f=log(m)*sqrt(x/y);
     else f=log(m)/log(2.);
     return f;
}
int rand_int(int M, int N)
{
     if(M>=N)
     {cout<< "\n Incorrect values N<M\n";
system("pause");exit(1);}
     else
          return rand()%(N-M+1)+M;
}
float rand_float(float M, float N)
{
     if(M>=N)
     {cout<< "\n Incorrect values nx<mx\n";
system("pause");exit(1);}
     else
     return N*rand()/RAND_MAX+M*rand()/RAND_MAX;
}
```

Решение задачи (для двух запусков программы) имеет следующий вид:



В случае неправильного ввода данных предусмотрено преждевременное завершение программы



Анализ программы.

Очевидно, что применение функций, в которые выневено выичление громоздких выражений, существенно упростило написание главной программы. Позволяя видеть стратегию решения задачи. Функция `phi может быть вызвана и вычислена как в теле функции U` (её значение передаётся в `main` через ссылку), так и в теле главной программы .

Результаты работы программы показывают, что вызов функции `srand` с аргументом 0 позволяет работать с разыми случайными числами.

Полнота задачи: проверка получения разных случайных чисел, принадлежности их заданным диапазонам значений, преждевременный выход в случае неправильного ввода знчений - продемонстрирована четырьмя результатами работы программы.

Примеры решения задач по обратке статических массивов

Пример 7: Дан массив a[8] типа int, заполнить его по формуле a[i] = старшая восьмерка бит = 3*i, младшая = i+1, остальные = 0xFF, напечатать его в HEX форме. Посчитать и вывести на монитор количество бит установленных в 1 для каждого четного элемента массива.

Решение этой задачи выполним в рамках структурного подхода: разобьём задачу на функции так, чтобы главная программа имела линейный алгоритм.

Первая функция `init_arr` будет инициализировать и выводить на консоль статический массив, вторая `bits` – считать в нём и выводить количество бит установленных в 1 для каждого четного элемента массива.

```
void init_arr(int la[], int DIM); // инициализация la[DIM]
void bits(int la[], int DIM); // кол-во единичных бит в
                              //каждом элементе массива la[DIM]
int main()
```

```
{
	const int DIM = 8; // определяем размер массива

	int a[DIM]; // описание массива

	setlocale(LC_ALL, "rus"); // для вывода русского шрифта в консоль
		init_arr(a, DIM);
	    bits(a, DIM);
	printf("\n");
	system("pause");
	return 0;
}

void init_arr(int la[], int DIM)
{
	for (int i = 0; i < DIM; i++)   // заполняем и печатаем массив
	{
		la[i] = (((3 * (long)i) << 24) + i + 1) | 0x00fff00;
		printf("\n la[%d] = %lx", i, la[i]);
	}

}
void bits(int la[], int DIM)
{
	int j, kbit, i; // описание типов переменных
	for (i = 0; i < DIM; i++) // цикл по четным элементам
{

if (la[i] % 2 == 0)
	{
		kbit = 0; // количество бит установленных в 1=0
		for (j = 0; j < 8 * sizeof(int); j++) // цикл по битам
			if ((la[i] & (1L << j)) != 0)
				kbit++; // считаем количество бит, установленных
					// в 1  выводим значения
		printf("\n В la[%d] бит установленных в 1 - %d", i, kbit);
	}
}}
```

Решение задачи:

```
la[0] = ffff01
la[1] = 3ffff02
la[2] = 6ffff03
la[3] = 9ffff04
la[4] = cffff05
la[5] = fffff06
la[6] = 12ffff07
la[7] = 15ffff08
В la[1] бит установленных в 1 - 19
В la[3] бит установленных в 1 - 19
В la[5] бит установленных в 1 - 22
В la[7] бит установленных в 1 - 20
Для продолжения нажмите любую клавишу . . .
```

Пример 8. Посчитать и вывести на монитор количество бит установленных в 1 в коде ASCII для четырех символов 'R', 'r', 'Y', 'y'. Найти и напечатать символ, в котором максимальное количество бит установлено в 1.

При выполнении арифметических и битовых операций с символьными данными используется их числовые коды. В коде программы обратите внимание на особенности вывода чисел в 10-м и 16-м форматах в операторе `cout`.

Назначение модулей программы, реализующих поставленную задачу, описано в комментариях к прототипам функций.

Решение задачи:

```
void init_arr(char s[], int a[], int DIM);//инициализация массива a[] числом бит, установленных в 1, в элементах массива s[].
int bits(char s);//вычисление в параметре s количества бит, установленных в 1
int max_bit(int la[], int DIM);// поиск индекса максимального элемента массива la[]
void print_codes(char s[], int DIM);//печать числовых кодов символьного массива в Dec и HEX-форматах
void prnt(char s[], int a[], int DIM);// печать массива символов s[] и соответствующего массива a[], содержащего значения количества единичных бит в символах
int main()
{
	const int DIM = 4; // определяем размер массива
	char a[DIM] = {'R','r','Y','y'}; // описание и инициализация массива символов
	print_codes(a, DIM);
	int b[DIM], k; // описание массива для хранения количества единичных бит в символах
```

```
	init_arr(a, b, DIM);
	prnt(a, b,  DIM);

	k=max_bit(b, DIM); // определение индекса элемента с макимальным
количество единичных бит
	cout << "Max number of bits = " << b[k] << "  in a symbol = " <<
a[k] << endl;
	system("pause");
	return 0;
}
void init_arr(char s[], int a[], int DIM)
{
	for (int i = 0; i < DIM; i++)   // заполняем и печатаем массив
	{
		a[i] = bits(s[i]);
	}
	return;
}
int bits(char s)
{
	int j, kbit, i; // описание типов переменных
		kbit = 0; // количество бит установленных в 1=0
		for (j = 0; j < 8 * sizeof(char); j++) // цикл по битам
		if ((s & (1L << j)) != 0)
			kbit++; // считаем количество бит //установленных в 1
выводим значения
	return kbit;
}
int max_bit(int la[], int DIM)
{
	int i, imax;
	i = imax = 0;
	for (i = 1; i < DIM; i++)
		if (la[i] > la[imax])imax = i;
	return imax;
}
void print_codes(char s[], int DIM)
{
	cout << endl;
	for (int i = 0; i < DIM; i++)
		cout << s[i] << '\t' <<dec<<int(s[i])<<'\t'<< hex<<int(s[i])
<< endl;
	cout << endl;
}
void prnt(char s[], int a[], int DIM)
{
	cout << endl;
	for (int i = 0; i < DIM; i++)
		cout << s[i] << '\t' << a[i] << endl;
	cout << endl;
```

```
}
```

Решение задачи:

```
R       82      52
r       114     72
Y       89      59
y       121     79


R       3
r       4
Y       4
y       5

Max number of bits = 5  in a symbol = y
Для продолжения нажмите любую клавишу . . .
```

Примеры задач на анализ кода программ

Пример 9. Изучить текст программы и показать, что будет выведено на консоль, показав и объяснив все промежуточные результаты.

```
#include <iostream>
using namespace std;
int expr(int x, int y);
int main()
{
	int a[] = { 3,-4,0 }, b[] = { 9,0,-3 };
	int n = sizeof(a) / sizeof(int);
	for (int i = 0; i < n; i++)
	{
		int z = expr(a[i], b[i]);
		cout << "\n i = " << i << " z = " << z << endl;
	}
	system("pause");
	return 0;
}
int expr(int x, int y)
{
	int z = 3;
	if (x == 0 && y != 0) z -= 5 / y;
	else if (y == 0 && x != 0) z += 5 * x;
	else if (x != 0 && y != 0)z = x / y + y / x;
	return z;
}
```

Решение задачи:

1. Инициализируем значения массивов:

index	0	1	2
a	3	-4	0

index	0	1	2
b	9	0	-3

```
2. n=3*4/4=3
3. цикл: i=0<3? Да!
i=0:переход в expr: сопоставляем формальн. параметры с фактич:
          x=a(0)=3, y=b(0)=9
         (x=0? И y!=0)?0 нет!
         (y=0? И x!=0)?0 нет!
         (y=!0? И x!=0)?0 да!, тогда z=3/9+9/3=3
          возврат z=3 в main

 Печать с новой строки: i = 0  z =3
      i=0+1=1<3? Да!
i=1: переход в expr:  x=a(1)=-4, y=b(1)=0
         (x=0? И y!=0)?0 нет!
         (y=0? И x!=0)?0 да!, тогда z=3+5*(-4)=-17
        возврат z=-17 в main

Печать с новой строки: i = 1  z = -17

    i=1+1=2<3? Да!
    i=2: переход в expr: x=a(2)=0, y=b(2)=-3
            (x=0? И y!=0)?0 да!, тогда z=3-5/(-3)= 4
           возврат z=4 в main
         Печать с новой строки: i = 2  z = 4
    i=2+1=3<3? Нет! Выходим из цикла
    Завершаем программу

    В результате, консоль будет выглядеть следующим образом:

    i = 0  z =3
    i = 1  z = -17
    i = 2  z = 4
```

Пример 10. Изучить текст программы и показать, что будет выведено на консоль, показав и объяснив все промежуточные результаты.

```
void f(char a, char& d)
{
   a = 'в'; d = 'д';
}
int main()
{
   char a[]="голос";
   f(a[0],a[4]);
   cout << a;
```

```
    return 0;
}.
```

Решение задачи:

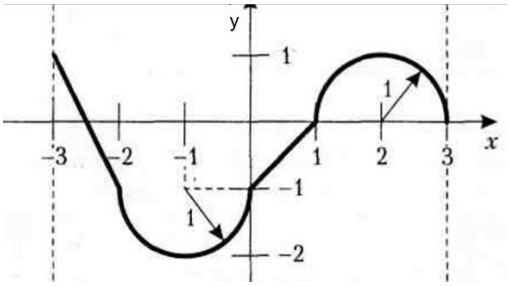
1. Распишем элементы массива с индексами

инд	0	1	2	3	4
символ	г	о	л	о	с

2. Передача управление в f(a[0],a[4])
   1.Сопоставляем фактические параметры с формальными:
   в ячейку a копируется ‘г’ – реализована передача по значению,
   d принимает адрес элемента a[4].
3. Присвоение a = 'в' реализуется на копию ячейки a[0], поэтому значение a[0] не изменится.
   Присвоение d = 'д' реализуется по адресу a[4], а это значит, что старое значение ‘с’ будет заменено на 'д'.
   Возврат в main
4. На консоль выводится символьный массив со значениями “голод”

Пример решения задач на описание геометрической фигуры

Пример 11 Составить программу, которая по введенному значению аргумента x вычисляет значение функции y, заданной в виде графика на интервале [-3;3].



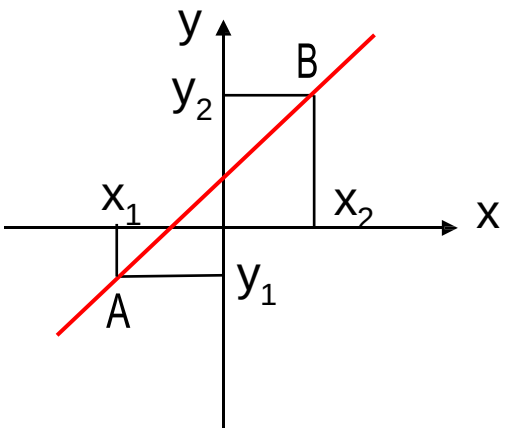
Решение этой задачи выполним в рамках структурного подхода: разобьём задачу на функции так, чтобы главная программа имела линейный алгоритм.

Уравнение прямой,
проходящей через две данные точки
A (x_1, y_1) и B (x_2, y_2)

$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1}$$

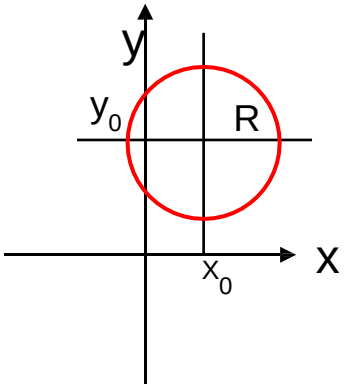
отсюда следует:

$$y = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1$$



Уравнение окружности радиуса R
с центром в точке (x_0, y_0):

$$(x - x_0)^2 + (y - y_0)^2 = R^2$$



отсюда следует:

$$y = \pm\sqrt{R^2 - (x - x_0)^2} + y_0$$

Это уравнения верхней (+) и нижней (-) полуокружности
с центром в точке (x_0, y_0).

Используя фрмулы определения уравнения прямых по двум точкам и уравнения окружности с заданным радиусом и координами центра, получим уравнения для всех точек заданного интервала:

$y = -2x - 5$	если $-3 \le x < -2$
$y = -\sqrt{1 - (x+1)^2} - 1$	если $-2 \le x < 0$
$y = x - 1$	если $0 \le x < 1$
$y = \sqrt{1 - (x-2)^2}$	если $1 \le x \le 3$

Решение:

```
float y1(float x);
float y2(float x);
float y3(float x);
float y4(float x);
void plot(float x);
int main()
{

	double x; char S;
	cout << "Input any value or except Ctrl+z, if you want to finish the program: ";
	while(cin >> S)
	{
		cout << "x: "; cin >> x;
		plot(x);
		cout << "Input any value or except Ctrl+z, if you want to finish the
program: ";
	}
	system("pause");
	return 0;
}
float y1(float x) {return -2 * x - 5;}
float y2(float x) { return -sqrt(1 - pow(x + 1, 2)) - 1;}
float y3(float x) { return x - 1; }
float y4(float x) { return sqrt(1 - pow(x - 2, 2)); }
void plot(float x)
{
	double y;
	if (x >= -3 && x <= 3)
	{
```

```
                if (x < -2)
                        y = y1(x);
                else
                        if (x >= -2 && x < 0)
                                y = y2(x);
                        else
                                if (x >= 0 && x < 1)
                                        y = y3(x);
                                else
                                        y = y4(x);
                cout << "y = " << y << endl;
        }
        else
                cout << "enter number     -3 <= x <= 3 ";
        return;
}
```

Решение

```
Input any value or except Ctrl+z, if you want to finish the program: 2
x: 2
y = 1
Input any value or except Ctrl+z, if you want to finish the program: 5
x: 5
enter number    -3 <= x <= 3 Input any value or except Ctrl+z, if you want to finish the program:
```

2. Методика выполнения самостоятельной работы

1. Ознакомиться с условием задачи и примерами решения аналогичных задач из раздела 1 данной лабораторной работы. Особое внимание уделить примерам с применением указателей и ссылок. Из теоретической части обратить внимание на свойства и правила применения указателей и ссылок.
2. Для задач Задания 1 достаточно разобрать аналогичные примеры из раздела 1. Для задач Задания 2 контроль решения осуществить после отладки программы, сравнивая её результаты с расчетами на калькуляторе. Для задач повышенной сложности необходимо вначале составить контрольные примеры, на основании которых построить модель процесса.
3. Проверить полноту задачи: рассмотреть все возможные исходы решения в зависимости от исходных данных, предусмотреть случаи возможного зависания, зацикливания программы и запрограммировать корректную реакцию программы на эти ситуации. Изучить примеры проверки полноты из анализа программы *Примера 6.*
4. Записать словесный алгоритм или составить блок-схему алгоритма.
5. Записать код программы на C^{++}.
6. Запустить программу, провести синтаксическую отладку.
7. Проверить работоспособность программы путём сравнения результатов с контрольным примером на все возможные случаи исходных данных.

8. Завершить работу составлением Отчёта, где будут описаны все этапы выполнения самостоятельного задания и приведены распечатки консольного вывода. Образец отчета приведен в Приложении.

3. Задания для самостоятельной работы

Задание 1. Составить функции для задач своего варианта. Для ряда случайно заданных значений аргумента реализовать расчёт с применением построенной функции и вывод соответствующих результатов. Для повторяющихся операций в главной программе предлагается использовать оператор `for`.

1. Написать процедуру *Mean(X,Y,Amean,Gmean),* вычисляющую *средне арифметическое Amean =(X+Y)/2 и среднее геометрическое Gmean =* $\sqrt{X \cdot Y}$ двух положительных чисел *X, Y*. С помощью функции найти среднее арифметическое и среднее геометрическое для пар (*A,B*), (*A,C*), (*A,D*), если заданы *A,B,C,D*.

2. Написать функцию RectPS(x_1,y_1,x_2,y_2,*P,S*), вычисляющую периметр *P* и площадь *S* прямоугольника со сторонами, параллельными осям координат, по координатам (x_1,y_1), (x_2,y_2) его противоположных вершин. С помощью этой процедуры вычислить периметры и площади трёх прямоугольников со сторонами, параллельными осям. Для вычисления длины стороны создать отдельную функцию.

3. Написать функцию GetCountsOfDigits(*N*), вычисляющую количество цифр в числе *N* на основе либо метода цикла, если число $N < 100000$, либо метода логарифма в противном случае. (см. *Пример 4* Л/Р №4). Проверить функцию на 5 случайных числах. При задании больших чисел правильно учесть тип аргумента.

4. Написать функцию InvDigits(K), меняющую порядок следования цифр целого положительного числа *K* на обратный. Применить процедуру к пяти случайным числам, содержащим не менее 3 и не более 7 цифр. Для определения числа цифр в числе создать соответствующую функцию.

5. Написать функцию DigitCountSum(K, S), находящую количество цифр целого положительного числа *K,* а также их сумму *S*. Применить процедуру к пяти случайным числам *K*, содержащим не менее 5 и не более 9 цифр. Для определения числа цифр в числе создать соответствующую функцию.

6. Написать функцию AddRightDigit (D, K), добавляющую к целому положительному числу *K* справа цифру *D*. Для *D* организовать проверку на принадлежность диапазону [0;9]. С помощью этой функции последовательно добавить к числу *K* справа заданные цифры D_1 и D_2.

7. Написать функцию AddLeftDigit (D, K), добавляющую к целому положительному числу *K* слева цифру *D*. Для *D* организовать проверку на

принадлежность диапазону [0;9]. С помощью этой функции последовательно добавить к числу *K* слева заданные цифры D_1 и D_2.

8. Написать функцию SortInc3(*A, B, C*), меняющую содержимое переменных *A, B, C* таким образом, чтобы их значения оказались упорядоченными по возрастанию. Для эффективной сортировки создать функцию Swap(*X, Y*) для обмена значений *X, Y*. Переменные *A, B, C* – вещественные параметры, являющиеся одновременно входными и выходными. С помощью SortInc3 упорядочить три набора случайных чисел.

9. Написать функцию SortDec3(*A, B, C*), меняющую содержимое переменных *A, B, C* таким образом, чтобы их значения оказались упорядоченными по убыванию. Для эффективной сортировки создать функцию Swap(*X, Y*) для обмена значений *X, Y*. Переменные *A, B, C* – вещественные параметры, являющиеся одновременно входными и выходными. С помощью SortDec3 упорядочить три набора случайных чисел.

10. Написать функцию ShiftRight3(*A, B, C*), выполняющую *правый циклический сдвиг*: значение *A* переходит в *B*, значение *B* – в *C*, значение *C* – в *A*. Переменные *A, B, C* – вещественные параметры, являющиеся одновременно входными и выходными. С помощью этой функции выполнить правый циклический сдвиг для трёх наборов случайных чисел.

11. Написать функцию ShiftLeftt3(*A, B, C*), выполняющую *левый циклический сдвиг*: значение *A* переходит в *C*, значение *C* – в *B*, значение *B* – в *A*. Переменные *A, B, C* – вещественные параметры, являющиеся одновременно входными и выходными. С помощью этой функции выполнить левый циклический сдвиг для трёх наборов случайных чисел.

Задание 2. Составить программу, разбивая задачу на несколько функций и, при необходимости, используя случайные числа в указанных диапазонах.

Для проверки работоспособности программы и её полноты обязательно составить контрольный пример. Номер варианта выбираем по формуле: $N_{вар}=N_{ст.билета}\%6+1$

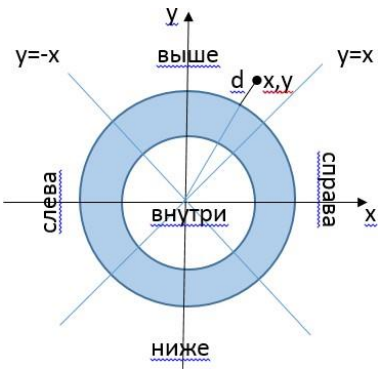
1. Температура а резервуаре растёт со временем t по закону $T = T_0 e^{kt}$. Смоделировать эксперимент (в виде отдельной функции), в котором нагрев температуры осуществляется до заданного значения $T_{пред}$ и определяется время достижения этой температуры. Экспериментатор имеет возможность изменять параметры $T_0 (t = 0)$ и k^4. Провести эксперимент для трёх разных наборов ($T_0 \in [10; 20]$, $k \in [1 \cdot 10^{-4}; 8 \cdot 10^{-4}]$) и оценить, при каких параметрах необходимая температура достигается быстрее.

2. Треугольник задан координатами своих вершин. Определить а) длину каждой стороны, b) является ли треугольник прямоугольным: можно использовать свойства скалярного произведения или теорему Пифагора (с

точностью до ε<<1). Если треугольник прямоугольный, то c) указать размер гипотенузы, в противном случае d) размер самой короткой стороны (равносторонний треугольник – частный случай непрямоугольных треугольников). Для каждого пункта заданий (a ÷ d) составить отдельные функции.

3. Согласно закону Мура, количество транзисторов, размещаемых на кристалле интегральной схемы, удваивается каждые 24 месяца. Ядро процессора Core2 Duo содержит порядка 300 миллионов транзисторов. Оценить время, за которое число транзисторов на процессоре возрастёт до $N = 3 \cdot 10^{14}$. Полученное значение выразить в годах, месяцах и днях, полагая, что месяц, в среднем, содержит 30 дней. Закон Мура и перевод времени из месяцев в года, месяцы и дни реализовать с помощью отдельных функций.

4. Смоделировать игру, в которой необходимо угадать число со значением в диапазоне от 1 до 10, «задуманное» компьютером. В распоряжении игрока $N \in [10; 20]$ фишек. Он называет любое число. Если оно совпадает с «загаданным» числом компьютера, игрок получает 1 «жизнь», добавляющую к его фишкам $m \in [2; 5]$ фишек. И, наоборот, если не угадывет число, то лишается m фишек. Игра продолжается до тех пор, пока не будут израсходованы все фишки, или пока игрок сам не захочет прекратить игру. По завершении игры программа должна сообщить общее количество произведённых угадываний, количество успешных угадываний и количество оставшихся у игрока фишек.



5. Смоделировать игру «Мишень», суть которой в следующем. Компьютер задаёт невидимую игроку мишень в виде кольца, образованного окружностями с центром в начале координат и радиусами R_1 и R_2. Игрок вводит координаты точки x, y. Игра завершается, когда произошло попадание в мишень и компьютер сообщает о победе игрока, или когда игрок ошибочно вводит хотя бы один нецифровой символ. Если попадания нет, то компьютер сообщает игроку, куда он попал: «внутрь мишени», «ниже | выше мишени», «левее | правее мишени» - и, в случае внешнего положения точки, сообщает её минимальное расстояние d до кольца. Для каждого случая попадания в указанные области и для определения расстояния d разработать отдельные функции.

6. Фирма вяжет веники. Запасы хворостин исчисляются числом $N > 1000$. На один веник уходит $m \in [20; 30]$ хворостин. Примерно, каждая k-я хворостина из N оказывается бракованной и выкидывается. Определить сколько веников будет произведено и какое количество не бракованных хворостин останется не использованными. Для моделирования процесса производства и процесса отбраковки составить отдельные функции, параметры для них задать в главной программе.

Задание 3. Написать код программы, реализующей операции с одномерными массивами, разбивая код на функции и стремясь к линейной структуре главной программы**. *Разработка контрольного примера является обязательной.*** (см. Примеры 7 – 8 данной методички)

Вариант 1. Написать программу, которая позволит ввести с клавиатуры число x типа unsigned int, создать и вывести на дисплей массив a, в котором a[номер бита числа x] = значению бита числа x.

Вариант 2. Дан целый массив a[10], заполнить его по формуле a[i] = старшая восьмерка бит = i, младшая = i+1, напечатать. Посчитать и вывести на монитор количество бит установленных в 1 для каждого элемента массива.

Вариант 3. Дан массив a[8] типа char, заполнить его по формуле a[i] = 2^n +5 и вывести на монитор элементы массива, в которых есть 3 бита установленных в 0 и следующих подряд.

Вариант 4. Дан целый массив a[15], заполнить его по формуле a[i] = 3*i напечатать. Посчитать и вывести на монитор максимальное количество бит установленных в 0 и следующих подряд для каждого элемента массива.

Вариант 5. Дан целый массив a[18], заполнить его по формуле a[i] = старшая восьмерка бит = 2*i, младшая = i, напечатать. Посчитать и вывести на монитор количество бит установленных в 0 для каждого элемента массива.

Вариант 6. Дан целый массив a[20], заполнить его по формуле a[i] = 5*i+2 , напечатать. Вывести на монитор номер и значение элемента массива, в котором максимальное число бит установлены в 1.

Вариант 7. Дан целый массив a[8], заполнить его по формуле a[i] = 3*i+4, напечатать. Вывести на монитор номер и значение элемента массива, в котором максимальное число бит установлены в 1 и следующих подряд.

Вариант 8. Дан массив a[25] типа unsigned int, заполнить его по формуле a[i] = старшие 16 = i<<4, младшие 16 = i*3, напечатать. Посчитать и вывести

на монитор максимальное количество бит установленных в 1 следующих подряд для каждого элемента массива.

Вариант 9. Посчитать и вывести на монитор количество бит установленных в 0 в коде ASCII для английских букв. Создать и напечатать массив этих букв по возрастанию в них количества бит установленных в 1.

Вариант 10. Посчитать и вывести на монитор количество бит установленных в 0 в коде ASCII для символов '0', '1', ..., '9'. Создать и напечатать массив этих символов по возрастанию в них количества бит установленных в 1.

Вариант 11. Дан целый массив a[19], заполнить его по формуле a[i] = 7*i+11 , напечатать его в порядке возрастания количества бит установленных для a[i] в 0.

Вариант 12. Дан целый массив a[25], заполнить его по формуле a[i] = 4*i+23 . Найти и вывести на монитор все пары элементов данного массива, у которых количество бит установленных в 1 равно.

Задание 4. Изучить код представленной программы и оформить анализ его работы в соответствии с представленным в методических указаниях образцом (пример 9). Номер варианта выбираем по формуле: $N_{вар}=N_{ст.билета}\%6+1$

Вариант 1. Изучить текст программы и показать промежуточные вычисления, определив, что будет выведено на экран с точностью до 1 знака после точки (Выполнять без компьютера!)

```
#include <iostream>
#include<cmath>
using namespace std;
int main()
{
     float pi=atan(1.)*4;
     float c[]={pi/6.,pi/4,pi/3,pi/2,pi*2/3,pi};
int n=sizeof(c)/4;
float s=0.001e3;
for (int i=0;i<n;i+=2)
s+=cos(c[i+1])+sin(c[i]);
cout<<"\ns = "<<s<<endl;
system("pause");
      return 0;
```

Вариант 2. Изучить текст программы и показать, что будет выведено на консоль, показав и объяснив все промежуточные результаты. (Выполнять без компьютера!)

```
#include<iostream>
#include<cmath>
using namespace std;
float prog(float a, int n);
int main()
{
	float tn=-3e0, tk=0.3e1, dt=2;
	int n=2;
	cout<<"\n\tt"<<"\ty\n\n";
		for(float t=tn;t<tk+dt/2;t+=dt)
		cout<<"\t"<<t<<"\t"<<prog(t,n)<<endl;
	system("pause");
	return 0;
}
float prog(float a, int nn)
{
	float fi=2.e1, u=1.;
	for(int i=1;i<=nn;i++)
	{
		u*=a*a;
		fi-=u/i;
	}
	return fi;
}
```

Вариант 3. Изучить текст программы и показать, что будет выведено на консоль, показав и объяснив все промежуточные результаты (Выполнять без компьютера!)

```
#include<iostream>
#include<cmath>
using namespace std;
float bond(float t);
int main()
{
	float zn=-1.0e1, zk=11e0, dz=0.5e1;

	cout<<"\n\tz"<<"\ty\n\n";
		for(float z=zn;z<zk;z+=dz)
		cout<<"\t"<<z<<"\t"<<bond(z)<<endl;
	system("pause");
	return 0;
}
float bond(float t)
{
```

```
    float a=3.127e2;
    if(t<0) return (int)t%3;
    else if (t>0) return fmod(a,t);
    else return cos(5.1e1*t);}
```

Вариант4. Изучить текст программы и показать, что будет выведено на консоль, показав и объяснив промежуточные результаты. (Выполнять без компьютера!)

```
#include <iostream>
using namespace std;
void constr(int b,int e, int s);
float res(int x, int st);
int main()
{
int a=-11,b=7,c=6;

cout<<"\n Construction table of selected function\n";
constr(a,b,c);
cout<<"\n\n"<<sizeof(a+b+c)<<endl;
system("pause");
return 0;
}
void constr(int b, int e, int s)
{
    for(int k=b;k<=e;k+=s)
    {
        cout <<"\t" << k << "\t" << res(k,5)<< endl;
    }
return;
}
float res(int x, int st)
{
    float z;
    if (x<0) z=x/st;
    else if(x>0) z= (float)x/st;
    else z=st;
    return z;
}
```

Вариант 5. Изучить текст программы и показать, что будет выведено на консоль, если на запрос с консоли ввести x=0.1 и eps=0.05. Показать и объяснить промежуточные результаты.

```
#include<iostream>
#include<cmath>
using namespace std;
float d(float x, float e);
```

```
int main()
{
	float x,eps,z;
	cout<<"\nInput argument << 1:"; cin >>x;
	 cout<<"\nInput accuracy:"; cin >>eps;
	z= d(x, eps);
	cout<<"\nFor x = "<<x<<" z(x) = "<< z <<endl;
	system("pause");
	return 0;
}
float d(float x, float e)
{
	float sum=1., u=2., cf=-1;
	for(int i=1;abs(u)>e;i++)
	{
		u*=x*x;
		sum+=cf*u;
		cf=-cf;
		cout<<"\n i = "<<i<<endl;
	}
	return sum;}
```

Вариант 6. Изучить текст программы и показать, что будет выведено на консоль, показав и объяснив промежуточные результаты. (Выполнять без компьютера!)

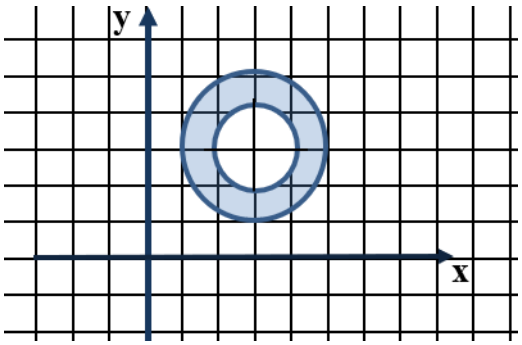
```
#include<iostream>
#include<cmath>
using namespace std;
float z(float t);
int main()
{
	float tn=-9, tk=9, dt=3;

	cout<<"\n\tt"<<"\ty\n\n";
		for(float t=tn;t<tk+dt/2;t+=dt)
		cout<<"\t"<<t<<"\t"<<z(t)<<endl;
	system("pause");
	return 0;
}
float z(float t)
{
	if(t<0) return t*t;
	else if (t>0) return fmod(10,t);
	else return pow(2,t);
}
```

Задание 5. По «мишени», изображенной на рисунке (закрашенная фигура), происходит обстрел шариками, которые случайным образом попадают в поле по x∈[a1; a2] и по y∈[b1; b2]. Ддля построения уравнений кривых воспользуйтесь примером 11. Все неравенства, закладываемые в программу должны быть записаны математически перед кодом решаемой задачи.

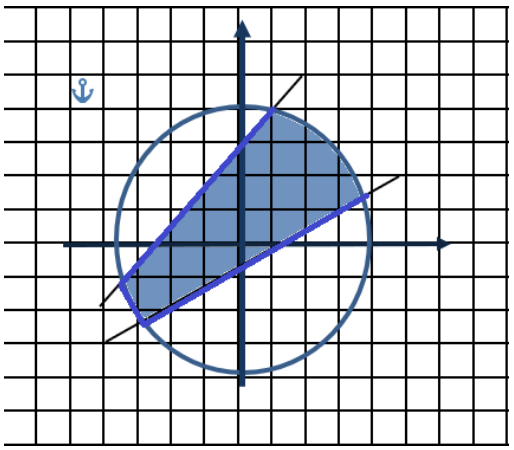
Составить две функции, одна генерирует координаты шариков в заданном диапазоне, а вторая определяет, попал шарик в мишень или не попал. В главной программе проводить «обстрел», т.е. генерацию координат шариков, до тех пор, пока с консоли не будет введено число 0. После проверки попадания/непопадания точки в «мишень» в главной программе напечатать соответствующее сообщение.

На рисунках показано начало координат. Каждая клеточка на разметке поля – одна единица длины**.**

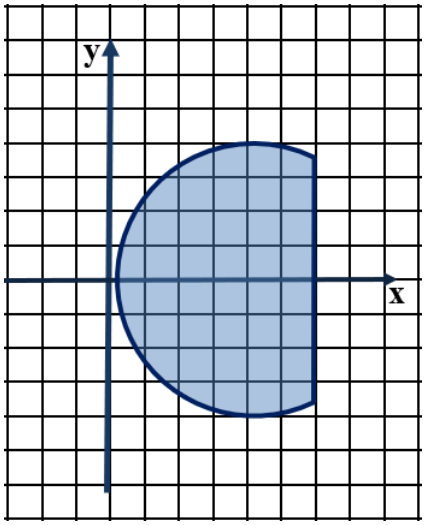
Вариант 1. a1=0; a2=6, b1=0; b2=6;



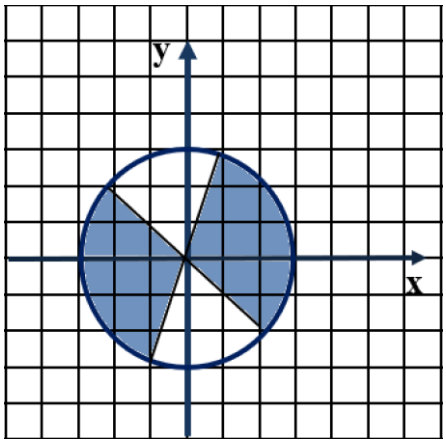
Вариант 2. a1=-2; a2=2, b1=-2; b2=2;



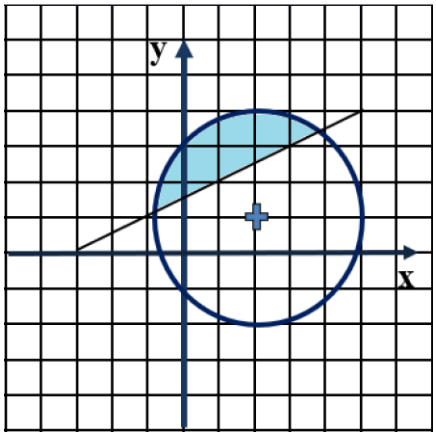
Вариант 3. a1=-3; a2=8, b1=-5; b2=5;



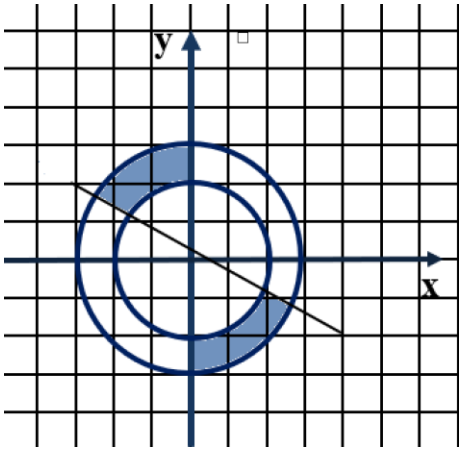
Вариант 4. a1=-5; a2=7, b1=-5; b2=8;



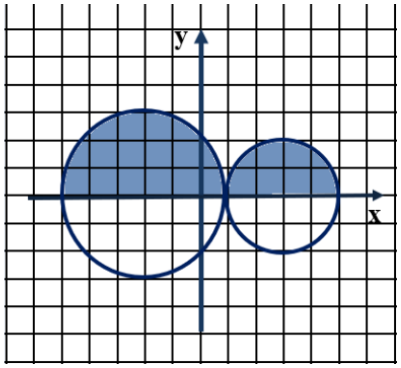
Вариант 5. a1=-5; a2=7, b1=-5; b2=8;



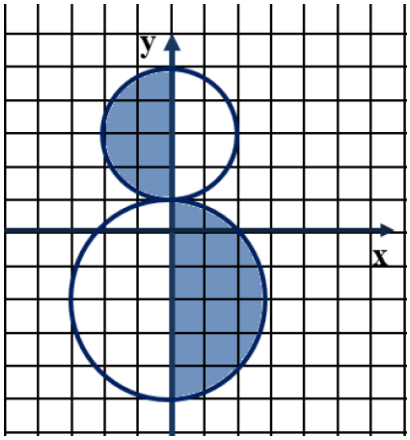
Вариант 6. a1=-4; a2=7, b1=-5; b2=7;



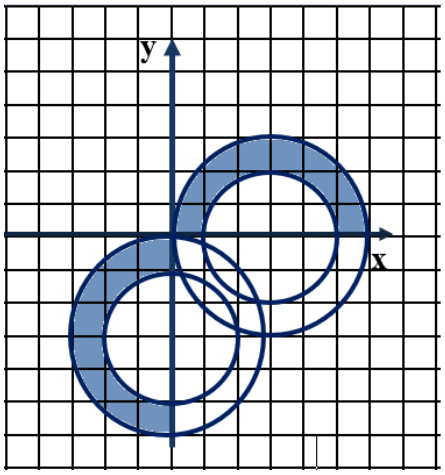
Вариант 7. a1=-7; a2=7, b1=-6; b2=7;



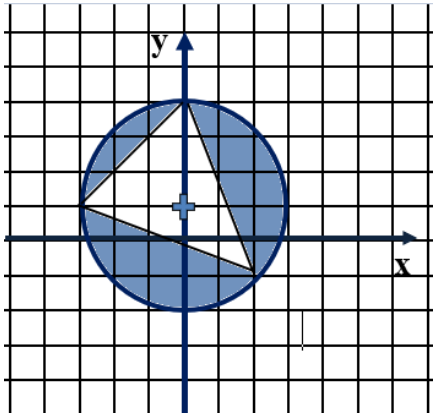
Вариант 8. a1=-5; a2=7, b1=-6; b2=7;



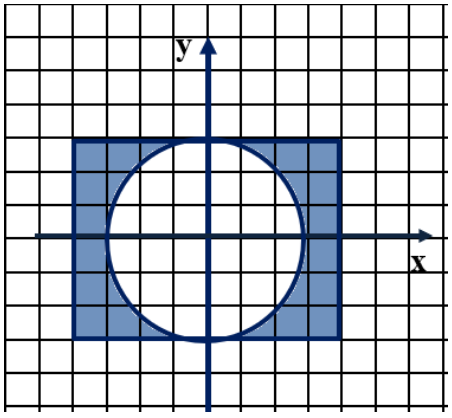
Вариант 9. a1=-5; a2=7, b1=-7; b2=7;



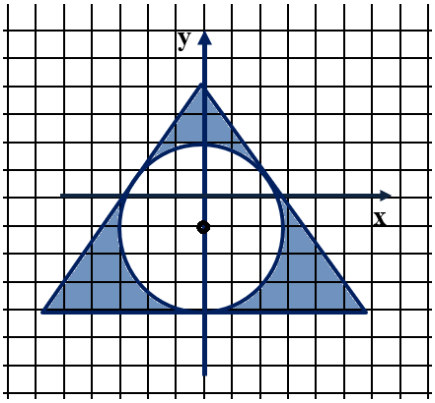
Вариант 10. a1=-5; a2=7, b1=-4; b2=7;



Вариант 11. a1=-5; a2=5, b1=-7; b2=6;



Вариант 12. a1=-7; a2=7, b1=-7; b2=7;



Задание 6. Изучить код представленной программы и оформить анализ его работы в соответствии с представленным в методических указаниях образцом в Примере 10.

1. Что будет выведено на экран в результате работы программы?

```
void f(char* a, char* b)
{
   a = b; b = a;
}
int main()
{
   char a[]="abcde";
   char b[]= "opr";
   f(a,b);
   cout<<a;
   return 0;
}
```

2. Каков результат выполнения программы

```
void f(int a, int& d)
{
   a = 7; d = -1;
}
int main()
{
int a[]={6,-1,0,3,2};
f(a[0],a[4]);
cout<< a[0] <<" "<<a[2]<<" "<<a[4]<<endl;
return 0;
}
```

3. Каков результат выполнения программы

```
void f(char &a, char& d)
{
   a = 'в'; d = 'с';
}
int main()
{
   char a[]="голос";
```

```
        f(a[0],a[4]);
        cout << a;;
        return 0;
    }.
```

4. Какой результат появится на консоли после выполнения программы

```
    void f(int a, int& c, int& d)
    {
    int b;
        a = 5; c = 7; b = 9;
    }
    int main()
    {
    int a, b, c, d;
        a = 1; c = 0; b = 3;
    f(b,a,c);
    cout<<a <<c << b;
    return 0;
}
```

5. Что будет выведено на экран в результате работы программы?

```
void f(int&a, int&b)
{
   a = b; b = a;
}
int main()
{
     int a[]={2,-1,3,0};
int b[]= {4,-1,0};
f(a[2], b[1]);
cout<<a[1]<<" "<<b[1] <<" "<<a[2]<<" "<<b[2]<<endl;
system("pause");
return 0;
}
```

6. Программа содержит объявления:

```
int f(char* s, char c);
char str1[81], str2[81], sim; int k,k1,k2;
```

Отметить правильные операторы вызова функции f и объяснить, почему отвергнуты остальные.

```
k1=f(str1,sim);k2=f(str2,sim);k=k1+k2;
k=f(str1,'a')+f(str2,'a');
k=f(str1,sim)+f(str2,sim);
k=f(*str1,sim)+f(*str2,sim);
k=f(str1,&sim)+f(str2,&sim);
```

7. Программа содержит объявления:

```
float a,b; int k; void f(float *, float *, int);
```

Отметить правильные операторы вызова функции **f** и объяснить, почему отвергнуты остальные.

```
f(&a, &b, 2);
f(&a, b, &k);
f(a, &b, k);
f(&b, &a, k);
f(&a, &b, 2);
```

8. Операцию изменения значения переменных X, Y так, чтобы X получила минимальное, а Y – максимальное из их начальных значений, можно реализовать следующими фрагментами программы на языке С:

R = X + Y; X = min(X, Y); Y = R – X;

if (X > Y) { X = X + Y; Y = X – Y; X = X – Y; }

R = (X < Y)? X: Y; Y = X + Y – R; X = R;

if (X > Y) R = X: else R = Y; X = X + Y – R; Y = R;

R = X * Y; if (X > Y) X = Y; Y = R / X;

Процедура P определена на языке С следующим образом:

```
char z;
void P ()
{  switch (z)
       case '+': z = '*'; break;
       case '–': z = '/'; break;
       case '*': z = '–'; break;
       default: z = '#'; break;
       case '/':  z = '+';
}
```

Результатом последовательности вызовов процедуры P() могут быть следующие последовательности значений переменной z …

Выбрать *не менее* двух вариантов

1) '+', '*', '–', '/', '+'
2) '*', '#', '#'
3) '/', '+', '–', '*', '+'
4) '<', '#', '#'
5) '/', '+', '*', '–'
6) '–', '/', '+', '*', '–', '/'

9. _____

10. Заголовок функции имеет вид: **`void f(int& a, float b, char c);`**

Переменные в вызывающей функции описаны так: **`int a; char b; float d,x;`**

Выбрать правильные варианты и обосновать, почему отвергаются оставшиеся варианты

Вариант 1 **`f(a,d-x/2,b);`** Вариант 3 **`f(a,x,'b');`**
Вариант 2 **`f(a,a,b);`** Вариант 4 **`f(2,d,b);`**

11. Заголовок функции имеет вид: **`void P(int& a, char* b, float c);`**

Переменные в вызывающей функции описаны так:

`int a; char* b, *c; float d,x;`

Выбрать правильные варианты и обосновать, почему отвергаются оставшиеся

варианты

Вариант 1 `P(a,b,c);`
Вариант 2 `P(3,b,x);`
Вариант 3 `P(a,b+5,d-a);`
Вариант 4 `P(a,c,d-x/2);`

12. Каков результат следующего выражения?

```
int *a; int b[2]; a = b;
b[0] = 7; b[1] = 10; *a++; cout << *a;
```

Обосновать выбор правильного варианта

Вариант 1 7
Вариант 2 10
Вариант 3 8
Вариант 4 11

* * *

4. Контрольные вопросы

1. Для чего необходимо разбивать задачу на отдельные модули?
2. Что такое функция в C^{++}? Что такое прототип функции?
3. Перечислить основные свойства функции. Когда функция имеет тип `void`?
4. Что такое параметры функции? Каково различие между фактическими и формальными параметрами?
5. Каким образом реализуется возврат одного и нескольких значений из функции? Что такое ссылка?
6. Что такое операция разыменования и в чём ее преимущество перед обращением к переменной по имени?
7. Записать формулы формирования целого и вещественного случайных чисел.
8. Как можно описать и инициализировать одномерный статический массив?
9. Как передаётся одномерный массив в функции?
10. Какой тип будет иметь функция, инициализирующая одномерный массив, для которого память выделена в вызывающей функции?
11. Напишите прототип функция, из которой можно получить сумму элементов массива, количество чётных элементов в нём и индекс максимального элемента.
12. Как определить уравнение прямой, заданной двумя точками?
13. Как определить, попадает ли точка в окружность?

14. Какой тип будет иметь функция, определяющая попала ли точка в заданную область?

15. Как реализуется передача параметров по значению?

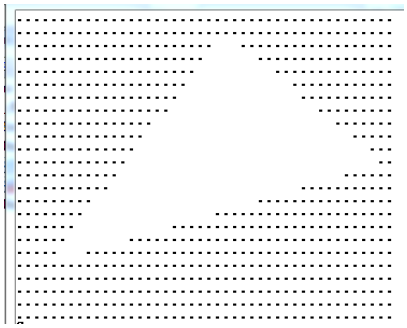
16. Как реализуется передача параметров по ссылке?

17. Для чего используется указатель?

5. Домашнее задание

1. Треугольник задан тремя сторонами *a, b, c*. Составить функции для определения по ним радиусов вписанной и описанной окружностей. Найти сумму и разность полученных радиусов для трёх треугольников, стороны которых задаются случайным образом (использовать или создать собственную функцию для проверки существования треугольника).

2. Разработать программу, которая выводила бы на экран следующую картинку:



Размер области построения 44 точки по горизонтали, 24 – по вертикали.

Разработать функции:

1 – задать случайным образом координату точки *x* в заданном диапазоне;

2 - задать координату точки *y* в заданном диапазоне;

3 – рассчитать в уравнении линии $y = a \cdot x + b$ коэффициенты *a* и *b* по двум точкам.

	•1	
		•3
•2		

Координата точек вершин треугольника: точка 1 лежит в верхней 1/3 плоскости экрана, точка 2 – в левой 1/3 экрана, точка 3 – в центральной части правой 1/3 экрана.

3. Изучить текст программы и показать, что будет выведено на консоль, показав и объяснив промежуточные результаты. (Выполнять без компьютера!)

```
#include <iostream>
using namespace std;
//Объявление (прототипы) функций:
void tab(int b,int e, int s);
float res1(int x, int st);
float res2(int x, int st);
int main()
{
int xn=-8,xk=8,dx=4;

cout<<"\n Construction table of selected function\n";
tab(xn, xk, dx);
system("pause");
return 0; }
```

```
void tab(int b,int e, int s)
{
     for(int i=b;i<=e;i+=s)
     {
          float result;
          if(i<0) result=res1(i,2);
          else if(i>0) result=res2(i,3);
          else result=0;
          cout <<"\t" << i << "\t" << result<< endl;
     }
return;
}
float res1(int x, int st)
{
     return x/st;
}
float res2(int x, int st)
{
     return (float)x/st;
}
```