БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ РАДИОФИЗИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ
КАФЕДРА ИНФОРМАТИКИ И КОМПЬЮТЕРНЫХ СИСТЕМ

Н. В. Серикова

ПРАКТИЧЕСКОЕ РУКОВОДСТВО
к лабораторному практикуму

«ФУНКЦИИ»

по дисциплине

«ПРОГРАММИРОВАНИЕ НА C++»

2024
МИНСК

Практическое руководство к лабораторному практикуму «ФУНКЦИИ» по дисциплине «ПРОГРАММИРОВАНИЕ НА С++» предназначено для студентов, изучающих базовый курс программирования на языке С++, специальностей «Компьютерная безопасность», «Прикладная информатика», «Радиофизика».

Руководство содержит некоторый справочный материал, примеры решения типовых задач с комментариями.

Автор будет признателен всем, кто поделится своими соображениями по совершенствованию данного пособия.

Возможные предложения и замечания можно присылать по адресу:

E-mail: Serikova@bsu.by

ОГЛАВЛЕНИЕ

ОБЛАСТИ ВИДИМОСТИ И КЛАССЫ ПАМЯТИ ПЕРЕМЕННЫХ

Область видимости определяет, из каких частей программы возможен доступ к переменной

Класс памяти определяет время, в течение которого переменная существует в памяти компьютера

В C++ существуют 3 типа области видимости переменных:

- локальная область видимости
- область видимости файла
- область видимости класса (будет рассмотрена позднее)

Переменные, имеющие локальную область видимости доступны только внутри того блока, в котором они определены (блоком обычно считается код, заключенный в фигурные скобки).

Переменные, имеющие область видимости файла, доступны из любого места файла, в котором они определены

В C++ существует 3 класса памяти:

- auto (автоматический)
- static (статический)
- динамический (будет рассмотрен позднее)

- Автоматическая переменная «рождается» в момент ее объявления и прекращает свое существование в момент завершения выполнения блока, где она определена. Автоматическая переменная не инициализируется автоматически. Если она инициализируется при объявлении, инициализация будет выполняться каждый раз при входе в блок и «рождении» переменной.
- У переменных, имеющих класс памяти static, время жизни равно времени жизни всей программы. Статическая переменная по умолчанию инициализируется нулем. Статическая переменная создается и инициализируется один раз – при первом выполнении блока.

- **Глобальные переменные** объявляются вне всех блоков и классов (о последних речь пойдет позже) и имеют область видимости файла. Они доступны всем функциям и блокам, начиная с той точки файла программы, где они объявлены.
- Глобальные переменные можно сделать доступными и из других файлов, если программа состоит из нескольких файлов.
- По умолчанию глобальные переменные имеют статический класс памяти.
- Глобальные переменные живут все время выполнения программы. Если они не инициализируются явно, по умолчанию, глобальные переменные инициализируются нулевым значением.

По возможности, использования глобальных переменных следует избегать

СИНТАКСИС ОПИСАНИЯ ФУНКЦИЙ

Функция – изолированный именованный блок кода, имеющий определенное назначение

Информация в функцию передается с помощью аргументов (фактических параметров), задаваемых при ее вызове. Эти аргументы должны соответствовать формальным параметрам, указанным в описании функции.

Синтаксис описания функций

```
Тип_Возврата   Имя_функции (список_параметров)          // заголовок
{
   операторы;                                            // тело функции
   return выражение;
}
```

- Область видимости параметров функции, объявленных в ее заголовке и переменных, объявленных в ее теле, ограничивается блоком тела функции.
- Эти переменные, если они не объявлены с атрибутом static уничтожаются после завершения выполнения функции, а хранимые ими значения безвозвратно теряются.
- Если функция не возвращает значения, в качестве типа возврата указывается void, а оператор return не содержит выражения, значение которого должно быть возвращено в вызываемую функцию.

ПРОТОТИПЫ ФУНКЦИЙ

- К моменту вызова функции компилятор должен иметь информацию о ней, чтобы скомпилировать правильно ее вызов.

- Если текст функции размещен в файле с исходным текстом после её вызова или вообще размещён в другом файле, необходимо объявить функцию с помощью оператора, называемого прототипом функции.

- Все прототипы функций обычно помещаются в начале исходного файла программы. Заголовочные файлы, включаемые для использования стандартных библиотечных функций, помимо прочего включают в себя прототипы этих функций.

- Описание прототипа ничем не отличается от описания заголовка функции:

Тип_Возврата Имя_функции **(***список_параметров***);**

- Описание прототипа, в отличие от заголовка, заканчивается точкой с запятой.

СПОСОБЫ ПЕРЕДАЧИ ДАННЫХ

Способы передачи данных в вызываемую функцию

- Путем передачи аргументов функции
- С использованием глобальных переменных
- Через файлы на внешних запоминающих устройствах

Способы передачи данных из вызываемой функции

- Через возвращаемое значение
- Через формальные параметры, вызываемые по ссылке
- Путем изменения значений глобальных переменных
- Через файлы на внешних запоминающих устройствах
- Кроме того, функция может выполнять какое-либо действие, не требующее передачи из нее данных в вызывающую функцию

В C++ используются два механизма передачи аргументов в функцию: по значению и по ссылке.

ПЕРЕДАЧА АРГУМЕНТОВ В ФУНКЦИЮ ПО ЗНАЧЕНИЮ

- При передаче по значению в функции создаются копии аргументов с именами формальных параметров.
- Значения аргументов копируются в созданные переменные.
- **По завершении работы функции обратное копирование из формальных параметров в переменные - аргументы не производится.**
- Если в теле функции значения формальных параметров были изменены, эти изменения по завершении работы функции будут потеряны и никак не отразятся на значениях фактических параметров.
- По умолчанию все аргументы, кроме массивов, передаются в функции по значению

ПЕРЕДАЧА АРГУМЕНТОВ В ФУНКЦИЮ ПО ССЫЛКЕ

- При передаче по ссылке в вызываемую функцию передаются ссылки на переменные – аргументы.
- Копии аргументов с именами формальных параметров не создаются.
- Вместо этого, по ссылке обеспечивается доступ к участкам памяти, занимаемым аргументами.
- Говоря другими словами, обработка аргументов ведется «на месте».
- Чтобы передать значения аргументов по ссылке, в качестве формальных параметров указывают переменные – ссылки.
- Все изменения формальных параметров, сделанные в функции, происходят с аргументами.
- Если в качестве аргумента по ссылке передается константа, то формальный параметр-ссылка должен быть объявлен с модификатором const.

ПЕРЕДАЧА УКАЗАТЕЛЕЙ

- Указатели обычно передаются по значению. Доступ к объектам, на которые они указывают при этом происходит «на месте».
- Применение указателя в качестве параметра позволяет функции получить аргумент, а не его копию.

ПЕРЕДАЧА МАССИВОВ

Массивы передаются в функции по ссылке.

- При этом записывать перед формальным параметров знак ссылки & не следует:
  int example (short a[3]);

- При описании в качестве формального параметра одномерного массива, его размер указывать необязательно:
  int example(short a[ ]);

- При описании в качестве формального параметра многомерного массива, его размер по левому измерению указывать необязательно:
  int example(short y[ ][4][3]);

Массив не может быть непосредственно возвращаемым значением

ПЕРЕДАЧА УКАЗАТЕЛЕЙ НА ФУНКЦИИ

- Указатель может хранить адрес функции. Это позволяет присваивать ему адрес точки вызова функции и вызывать ее через указатель.
- Указатель на функцию должен не только содержать адрес памяти, где находится функция, которую необходимо вызвать. Такой указатель должен поддерживать информацию о количестве и типах аргументов и типе возвращаемого значения.

Объявление указателей на функцию:

Тип_возврата **(*** *Имя_указателя***) (***список_типов_параметров***);**

Скобки, в которые взято *Имя_указателя позволяет отличить описание указателя на функцию
double (*pfun)(char*, int);

от описания прототипа функции, возвращающей указатель на double:
double *pfun (char*, int);

ИНИЦИАЛИЗАЦИЯ ПАРАМЕТРОВ

- Параметры функции, передаваемые по значению, можно инициализировать в ее прототипе
- Если при вызове функции аргумент, соответствующий инициализированному формальному параметру, будет опущен, формальному параметру будет присвоено инициализирующее значение. Если аргумент при вызове задан, инициализирующее значение игнорируется.
- Инициализировать можно произвольное число параметров функции
- Так как аргументы сопоставляются формальным параметрам по порядку следования, то, чтобы опустить при вызове какой-либо аргумент придётся опустить и все следующие за ним.

ВОЗВРАЩАЕМЫЕ ЗНАЧЕНИЯ

- В случае, если функция возвращает значение, его тип должен быть определен в описании функции. Он указывается перед идентификатором функции в описании прототипа и в заголовке функции.
- Количество аргументов у функции может быть произвольным, но возвращаемое значение только одно (или ни одного).
- Если функция не возвращает значения, в качестве типа возвращаемого значения следует указать void (по умолчанию считается, что функция возвращает целочисленное значение).
- Возвращаемое значение является операндом оператора return. В качестве возвращаемого значения может быть указано выражение, вырабатывающее значение соответствующего типа.

ВОЗВРАТ ССЫЛОК: ФУНКЦИЯ В ЛЕВОЙ ЧАСТИ ОПЕРАТОРА ПРИСВАИВАНИЯ

- **Функция может возвращать ссылку на объект**
- В этом случае записанное по адресу, на который указывает ссылка, значение может быть модифицировано.
- Для этого вызов функции должен осуществляться из левой части оператора присваивания.

ПЕРЕГРУЗКА ФУНКЦИЙ

- Перегруженная функция выполняет различные действия, зависящие от количества аргументов и типов данных, передаваемых ей в качестве аргументов
- Для того, чтобы создать перегруженную функцию необходимо описать требуемое число одноименных функций с требуемыми наборами формальных параметров.
- Сопоставив количество и типы аргументов при вызове, компилятор сгенерирует обращение к требуемой перегруженной функции.

ШАБЛОН ФУНКЦИЙ

Подготовку перегружаемых функций помогают автоматизировать шаблоны. Шаблон семейства функций определяется один раз, но это определение параметризуется. Для этого используется список параметров шаблона (в качестве параметра на этапе компиляции передается конкретный тип данных).

Шаблон семейства функций состоит из двух частей:

- заголовок шаблона (template <список параметров шаблона>)
- обычного определения функции (заголовок и тело функции), в котором тип возвращаемого значения и/или типы параметров обозначаются именами параметров шаблона, введенных в его заголовке.

Имена параметров шаблона могут использоваться и в теле определения функции для обозначения типов локальных объектов.

Формат простейшей функции-шаблона:

```
template <typename type>
заголовок функции
{ тело функции
}
```

где вместо слова type может использоваться произвольное имя.

РЕКУРСИЯ

Рекурсивным называется объект, который частично определяется через самого себя.

Вызов рекурсивной процедуры должен выполняться по условию, которое на каком-то уровне рекурсии станет ложным.

Если условие истинно, то цепочка рекурсивных вызовов продолжается. Когда оно становится ложным, то рекурсивный спуск заканчивается и начинается поочередный рекурсивный возврат из всех вызванных на данный момент копий рекурсивной процедуры.

Максимальное число рекурсивных вызовов процедуры без возвратов, которое происходит во время выполнения программы, называется **глубиной рекурсии.**

Число рекурсивных вызовов в каждый конкретный момент времени, называется **текущим уровнем рекурсии.**

Структура рекурсивной процедуры может принимать три разных формы:

1. Форма с выполнением действий до рекурсивного вызова *(с выполнением действий на рекурсивном спуске).*

```
void Rec()
{
    … S … ;
    if (условие) Rec ();
    return;
 }
```

2. Форма с выполнением действий после рекурсивного вызова *(с выполнением действий на рекурсивном возврате).*

```
void Rec()
{
    if условие Rec ();
    … S … ;
    return;
}
```

3. Форма с выполнением действий как до, так и после рекурсивного вызова *(с выполнением действий как на рекурсивном спуске, так и на рекурсивном возврате).*

```
void Rec ()
{
    … S1 …;
    if (условие) Rec();
    … S2 … ;
    return;
};
```

ПРИМЕР 1. Глобальные и локальные переменные

Пример вывода на экран значения переменной a через функции. Функция f1 - без формального параметра, функция f2 выводит на экран значение переданного параметра.

```
#include <iostream>  // for cin cout

using namespace std;

int a = 11;                    // глобальная переменная

void f1(); // прототип функции f1 без параметров
void f2(int); // прототип функции f2 с одним параметром

void main ()
{
    cout<<"  1  "<<endl;
    f1();            // вызов функции f1            a = 11
    f2(a);           // вызов функции f2            a = 11

    cout<<endl<<"  2 "<<endl;
    a = 22;
    f1();            // вызов функции f1            a = 22
    f2(a);           // вызов функции f2            a = 22

    cout<<endl<<"  3 "<<endl;
    f1();            // вызов функции f1            a = 22
    f2(33);          // вызов функции f2            a = 33
}

                                    // определение функции  f1
void f1()
{
    cout<<" 1 a= "<<a<<endl;
}

                                    // определение функции  f2
void f2(int a)
{
    cout<<" 2 a= "<<a<<endl;
}
```

ПРИМЕР 2. Глобальные и локальные переменные

Пример вывода на экран значения переменной a через функции. Определение функций до вызова. Функция *f*1 - без параметра имеет локальную переменную a, функция *f*2 выводит на экран значение переданного параметра и значение глобальной переменной a.

```
#include <iostream>  // for cin cout

using namespace std;

int a = 11;                          // глобальная переменная

                                    // определение функции  f1
void f1()
{
    int a = 55;
    cout << " 1 a = " << a << endl;
}
                                    // определение функции  f2
void f2(int b)
{
    cout << " 2 a = " << a << "  b = " << b << endl;
}


void main ()
{
    cout << "  1  " << endl;
    f1();                  // вызов функции f1  a = 55
    f2(a);                 // вызов функции f2  a = 11  b = 11

    cout << endl << "  2 " << endl;
    a = 22;
    f1();              // вызов функции f1        a = 55
    f2(a);             // вызов функции f2        a = 22  b = 22

    cout << endl << "  3 " << endl;
    f1();            // вызов функции f1          a = 55
    f2(33);          // вызов функции f2          a = 22  b = 33
}
```

ПРИМЕР 3. Глобальные и локальные переменные

Пример вывода на экран значения переменной a через функции. Функция *f*1 - без параметра, функция *f*2 выводит на экран значение переданного параметра.

```
#include <iostream>     // for cin cout
using namespace std;

int a = 11;             // глобальная переменная

void f1();     // объявление функции f1 без параметров
void f2(int); //объявление функции f2 с одним параметром

void main ()
{   cout << "  1  " << endl;
    f1();               // вызов функции f1      a = 11
    f2(a);              // вызов функции f2      a = 11

    cout << endl << "  2 " << endl;
    {   int a = 22;
        f1();                   // вызов функции f1      a = 11
        f2(a);                  // вызов функции f2      a = 22
        cout << endl << "  3  " << endl;
        {   int a = 33;
            f1();               // вызов функции f1      a = 11
            f2(a);              // вызов функции f2      a = 33
        }
        cout << endl << "  4  " << endl;
        f1();                   // вызов функции f1      a = 11
        f2(a);                  // вызов функции f2      a = 22
    }
    cout << endl << "  5  " << endl;
    f1();                       // вызов функции f1      a = 11
    f2(a);                      // вызов функции f2      a = 11
    cout << endl;
}
                          // определение функции  f1
void f1()
{
    cout << " 1 a = " << a << endl;
}
                         // определение функции  f2
void f2(int a)
{
    cout << " 2 a = " << a << endl;
}
```

ПРИМЕР 4. Передача параметров по значению. Возврат результата.

Вычислить минимальное из расстояний между заданными точками плоскости A(x1; y1), B(x2; y2) и C(1;2).

```
#include <math.h>
#include <iostream>  // for cin cout

using namespace std;

      // объявление функции (прототип)
      // идентификатор формального параметра можно опустить
double dd(double, double, double, double);

void main ()
{
    double xl, yl, x2, y2, d1, d2, d3, m;

    cout <<"Input koordinaty: ";
    cin >> xl >> yl >> x2 >> y2;

    d1 = dd(xl, yl, x2, y2) ;  // d1 - расстояние от A до B
    cout<<" d1= "<<d1<<endl;

    d2 = dd(xl, yl, 1, 2);     // d2 - расстояние от A до C
    cout<<" d2= "<<d2<<endl;

    d3 = dd(x2, y2, 1, 2);     // d3 - расстояние от B до C
    cout<<" d3= "<<d3<<endl;

    m =  (d1< d2) ?d1 : d2;
    if (d3 < m)
        m = d3;           //  минимальное расстояние
    cout <<  "min: " << m <<endl;
}


                              //определение функции
double dd(double a1, double  b1, double  a2, double  b2)
{                             // возвращаем результат
    return sqrt(
                (a1 - a2) * (a1 - a2) + (b1 - b2) * (b1 - b2)
               );
}
```

ПРИМЕР 5. Передача параметров по значению. Возврат результата.

Вычислить факториал N!

```
#include <iostream>  // for cin cout
using namespace std;

long  Iter_Fact (int n);  //объявление функции (прототип)
//  идентификатор формального параметра можно опустить
void main()
{
    int i;
    long Fact1, Fact2, Fact3;

    Fact1= Iter_Fact (10);         //вызов функции  можно так
    i = 10;
    Fact2 = Iter_Fact (i);        //вызов функции  или так
    Fact3 = Iter_Fact (5 + 5);    //вызов функции  или так

    cout << Fact1<<"  "<< Fact2 <<"  "<< Fact3 << endl;
}

                              //определение функции
long  Iter_Fact (int n)
{
    long  f = 1;
    for (int i = 2; i <= n;  i++)
        f = f * i;
    return (f);                 // возвращаем результат
}
```

ПРИМЕР 6. Передача параметров по значению. Инициализация параметров по умолчанию

Вычислить целую степень числа x^n (через умножение).

```
#include <iostream>  // for cin cout
using namespace std;
        //объявление функции  с инициализацией параметров
double stepen (double x=1, unsigned  n=1);

void main ()
{
    double  a = 7.1, b;
    unsigned  k = 5;

    b = stepen(2.7, k) +1 / stepen(a + 1, k);
                                        //  2.7 k  + 1/ (a+1)k
    cout << b << endl;

    b = stepen(2., k) + stepen(3, k);       //  2k  + 3k
    cout << b << endl;

    b = stepen();                           //   11
    cout << b << endl;

    b = stepen(5);                          //   51
    cout << b << endl;
}

double stepen (double x, unsigned  n)  //определение функции
                                    //  x - число  n - степень
{
    double  y;
    y = 1;
    for (unsigned  i = 1; i <= n;  i++)
        y = y * x;
    return  y;                      // возвращаем результат
}
```

ПРИМЕР 7. Передача параметров по значению, по ссылке.

Обмен значениями двух переменных

```
#include <iostream>  // for cin cout
using namespace std;
  // определение функции без прототипа
  //  передача параметров по значению
void change (double x, double y)
{
     double  z = x;
     x = y;
     y = z;
}
                  //определение функции без прототипа
                 //  передача параметров по ссылке
void changeRef (double &x, double &y)
{
     double z = x;
     x = y;
     y = z;
}
                //определение функции без прототипа
               //  передача параметров через указатель
void changePtr (double* x, double* y)
{
     double z = *x;
     *x = *y;
     *y = z;
}

void main()
{
     double d = 1.23;
     double e = 4.56;

     change(d, e) ;
     cout << " d=   " << d <<"  e = " << e <<endl;
                                  // значения не изменились
     changePtr(&d, &e) ;
     cout << " d=   " << d <<"  e = " << e <<endl;
                                  // значения изменились
     changeRef(d, e) ;
     cout << " d=   " << d <<"  e = " << e <<endl;
                                   // значения изменились
}
```

ПРИМЕР 8. Передача параметров по ссылке

Перераспределение значений вещественных переменных a, b, c так, чтобы стало a >= b >= c. Функция maxmin(x,y) присваивает параметру x большее из двух вещественных чисел, а параметру y – меньшее.

```
#include <iostream>  // for cin cout
using namespace std;

void maxmin (double  &, double &);       //объявление функции
void max_min (double *, double *);       //объявление функции

void  main ()
{
     double  a, b, c;
     cin >> a >> b >> c;
     maxmin (a ,b);
     cout << a  <<"  "<< b << "  "<< c << endl;
     maxmin (a, c);        //a  получит максимальное значение
     cout << a  <<"  "<< b << "  "<< c << endl;
     maxmin (b, c);       //c получит минимальное значение
     cout << a  <<"  "<< b << "  "<< c << endl;
     cin >> a >> b >> c;
     max_min (&a , &b);
     cout << a  <<"  " << b << "  "<< c << endl;
     max_min (&a, &c);      //a  получит максимальное значение
     cout << a  <<"  " << b << "  "  << c << endl;
     max_min (&b, &c);     //c получит минимальное значение
     cout << a  <<"  "<< b << "  " << c << endl;
}

void maxmin (double & x, double &y) // определение функции
{    if (x < y)
      {  double r = x;
                 x = y;
                 y = r ;
      }
}
void max_min (double  *x, double *y) //определение функции
{
     if (*x < *y)
       { double r =*x;
                *x  = *y;
                *y  = r ;
        }
}
```

ПРИМЕР 9. Передача параметров по ссылке.

Выделение целой и дробной частей вещественного числа.

```
#include <iostream>  // for cin cout
using namespace std;


void intfrac(float, float&, float&); //прототип функции

void main()
{
     float number, intpart, fracpart;

     cout << "Enter a real number: \n ";
     cin >> number;
     do
     {
       //вызов функции, неинициализированные переменные
      // intpart, fracpart передаются для возврата результата
          intfrac(number, intpart, fracpart);

          cout  << " integer part is " << intpart << endl;
          cout  << " fraction part is " << fracpart << endl;
          cout << "\nEnter a real number: \n ";
          cin >> number;
     }
     while(number != 0);   //завершение работы, если введен 0
}

                                 // определение функции
void intfrac(float n,  float&  intp,  float&  fracp)
{
      //конвертирование в long int для выделения целой части
   long temp = static_cast<long>(n);
        intp = static_cast<float>(temp);
                    //конвертирование назад в тип  float
     fracp = n - intp;        //нахождение дробной части
}
```

ПРИМЕР 10. Передача одномерного массива по ссылке

Вычисление суммы элементов массива.

```
#include <iostream>  // for cin cout
using namespace std;


  //размерность массива – глобальная константа, видима везде
const int n = 7;
                //прототип функции
int adder  (int iarray[n]);

void main()
{
     int  iarray1[n]={5, 1, 6, 20, 15, 0, 12};
                    //описание и инициализация массива
     int  iarray2[n]={ 1, 2, 3, 4, 5, 6, 7};
                    //описание и инициализация массива
     int isum;

    //вызов функции adder, ей передается только адрес массива
     isum = adder(iarray1);
     cout << "isum=" << isum <<  endl;

    //вызов функции adder, ей передается только адрес массива
     isum = adder(iarray2);
     cout << "isum=" << isum <<  endl;
}

                                    //определение функции
int adder  (int iarray[n])
{
     int i, ipartial = 0;
                //n – глобальная константа, видима здесь
     for  (i = 0; i < n;  i++)
          //накопление суммы элементов в переменной ipartial
         ipartial  +=  iarray[i];
     return (ipartial);
     //полученная сумма в качестве возвращаемого значения
}
```

ПРИМЕР 11. Передача одномерного массива и количества обрабатываемых значений массива

Вычисление суммы элементов массива.

```
#include <iostream>  // for cin cout
using namespace std;

    //прототип функции, появился второй параметр
   // идентификатор формального параметра можно опустить
int adder  (int [], int);
    //размер массива - глобальная константа, видима везде

void main()
{
    const int n = 7;
    int isum;
        // описание и инициализация массива
    int  iarray[n] = {5, 1, 6, 20, 15, 0, 12};
          //  вызов функции, ей передается адрес массива и
          //  количество элементов для обработки
          //  2-ой параметр должен быть меньше
          //  количества элементов массива n
    isum = adder(iarray, 3);
    cout << "isum=" << isum <<  endl;

}


int adder  (int array[], int k)   //определение функции
{
    int ipartial = 0;
   // k - формальный параметр функции, его значение равно 3
    for (int i = 0; i < k;  i++)
        ipartial += array[i];     //накопление суммы
    return (ipartial);
             //накопленная сумма возвращается в функцию
}
```

ПРИМЕР 12. Передача одномерного массива и количества обрабатываемых значений массива

Вычисление суммы элементов массива.

```
#include <iostream>  // for cin cout
using namespace std;

int adder  (int [], int);

void main()
{
     const int n = 7;
     int  iarray[n] = {5, 1, 6, 20, 15, 0, 12};
     int isum;
     int i = 3;

                    // вычисление суммы i элементов
     isum= adder(iarray, i);
     cout << "isum=" << isum <<  endl;

// функции adder передается адрес 4-го элемента и
// предполагается   вычисление суммы n-i элементов,
// начиная с 4-го  и до конца массива
     isum= adder(iarray + i, n - i);
     cout << "isum=" << isum <<  endl;
}

             // другое идентичное описание заголовка функции
int adder  (int *piarray, int k)
{
     int i , ipartial = 0;
       // k – формальный параметр, его значение равно
       // значению переданному из main
     for ( i = 0;  i < k;  i++)
         ipartial += piarray[i];
                  // piarray[i] есть *(piarray + i);
     return (ipartial);
}
```

ПРИМЕР 13. Передача в функцию строки символов.

```
#include <ctype.h>
#include <iostream>  // for cin cout

using namespace std;

void convertToUppercase (char *);    //прототип функции

void main()
{
     char string [ ] = "aaaaaaa";

     cout<<string<<endl;

     convertToUppercase(string);
     cout<<string<<endl;

          //  ошибка выполнения: строка-константа
     /*
     char *s1= "pointer_of_simvol";
     convertToUppercase(s1);
     cout << s1 <<endl;      */

}

                              //определение функции
void convertToUppercase (char *Ptr)
{
     while (*Ptr !=  '\0')
     {
          *Ptr = toupper(*Ptr);
                    // функция преобразования символов
          ++Ptr;
     }
}
```

ПРИМЕР 14. Выделение подстроки из строки

```
#include <string.h>
#include <iostream>  // for cin cout
using namespace std;

        //  Выделение подстроки st из строки  str
        //   с позиции номер poz количество символов kol
char *strsubstr
        ( char *str,char *st,unsigned poz, unsigned kol);

void  main()
{
    const int MAX = 80;
    char s[MAX], subs[MAX];
    char* str;
    unsigned poz,kol;

    cout << " s= \n";
    cin.getline(s, MAX);            // ввод  строки

    cout<<" N pozicii =  "<<endl;
    cin>>poz;

    cout<<" Number char =  "<<endl;
    cin>>kol;

             //  Выделение подстроки subs из строки  s
            //   с позиции номер poz количество символов kol
    str = strsubstr(s, subs, poz, kol);
    cout<<" str = "<<str<<endl;         // результат
    cout<<" subs = "<<subs<<endl;       // результат
}

           //  Выделение подстроки st из строки  str
           //   с позиции номер poz количество символов kol
char *strsubstr
     ( char *str, char *st,unsigned poz, unsigned kol)
{   unsigned i,j;
    for ( i = poz, j = 0;
         i < (poz + kol) && i < strlen (str);  i++, j++)
        st[j] = str[i];
    st[j] = '\0';
    return st;
}
```

ПРИМЕР 15. Удаление подстроки из строки

```
#include <string.h>
#include <iostream>  // for cin cout
using namespace std;
                  //  Удаление подстроки st из строки  str
char * strdelsub( char *str,char *st);

void  main()
{
     const int MAX = 80;
     char s[MAX], subs[MAX];
     char* str;

     cout << " s= \n";
     cin.getline(s, MAX);          // ввод  строки
     cout << " subs = \n";
     cin.getline(subs, MAX);       // ввод  подстроки
                    //  Удаление подстроки subs из строки  s
     str = strdelsub( s,subs);

     cout<<" str = "<<str<<endl;     // результат
     cout<<" s = "<<s<<endl;         // результат
}

              //      Удаление подстроки  st из строки str
char * strdelsub(char *str, char *st)
{
// Используем функцию для определения
// вхождения подстроки в строку
// функция
char *strstr( const char *string, const char *strCharSet );
//    = NULL, если подстрока не найдена
     char *s = strstr(str,st);
     if (s != NULL)
     {         // номер позиции вхождения подстроки в строку
          unsigned n = s - str;
          unsigned j = strlen(st);
          unsigned k = strlen(str) - j;
              // удаление j символов строки str
          for (unsigned i = n; i <= k; i++)
               str[i] = str[i + j];
     }
     return str;
}
```

ПРИМЕР 16. Вставка подстроки в строку

```
#include <string.h>
#include <iostream>  // for cin cout
using namespace std;
        //  Вставка подстроки st в строку  str с позиции poz
        //  MAX_ - максимальное количество символов строки
char * strinssub
       ( char *str, char *st, unsigned poz, unsigned MAX_);
void  main()
{
    const unsigned MAX = 80;
    char s[MAX], subs[MAX];
    unsigned poz;

    cout << " s= \n";
    cin.getline(s, MAX);                  // ввод  строки
    cout << " subs = \n";
    cin.getline(subs, MAX);               // ввод  подстроки
    cout << "c pozicii nomer " << endl;
    cin >> poz;
                  //  Вставка подстроки subs в строку  s
    str = strinssub(s, subs, poz, MAX);
    cout << " str = " << str << endl;         // результат
    cout << " s = " << s << endl;          // результат
}

       //  Вставка подстроки  st в строку str с позиции poz
char * strinssub
        ( char *str, char *st, unsigned poz, unsigned MAX_)
{
    unsigned st_len = strlen(st), str_len = strlen(str);
    if (poz > str_len ||
        st_len + str_len >= MAX_) // не забываем и про место
                                  // для концевого нуля
        return str;
    // сдвиг на l символов вправо
    for (i = str_len - 1; i >= poz && i >= 0;  i--)
        str[i + st_len] = str[i];
    str[st_len + str_len] = '\0';
    // вставка символов с позиции номер poz
    for (i = poz, j = 0; j < st_len; i++, j++)
        str[i] = st[j];
    return str;
}
```

ПРИМЕР 17. Замена подстроки на подстроку

```
#include <string.h>
#include <iostream>  // for cin cout
using namespace std;
    //  Замена подстроки st1 на подстроку st2 в строке  str
    //  MAX_ - максимальное количество символов строки
char * strchangsub
           (char *str, char *st1, char *st2, unsigned MAX_);
         //  Удаление подстроки st из строки  str
char * strdelsub( char *str,char *st);
        //  Вставка подстроки st в строку  str с позиции poz
char * strinssub
         (char *str, char *st, unsigned poz, unsigned MAX_);
void  main()
{   const unsigned MAX = 80;
    char s[MAX], s1[MAX], s2[MAX];
    char* str;
    cout << " s= \n";
    cin.getline(s, MAX);           // ввод  строки
    cout << " s1 = \n";
    cin.getline(s1, MAX);          // ввод  подстроки   1
    cout << " s2 = \n";
    cin.getline(s2, MAX);          // ввод  подстроки   2
            //  Замена подстроки s1 на s2 в строке  s
    str = strchangsub( s,s1,s2,MAX);
    cout<<" s = "<<str<<endl;      // результат
    cout<<" s = "<<s<<endl;        // результат
}
   //  Замена подстроки st1 на подстроку st2 в строке  str
char * strchangsub
            (char *str, char *st1, char *st2, unsigned MAX_)
{   if (strlen(st2) + strlen(str) <= MAX_)
    {        //  определение вхождения подстроки в строку
        char *s = strstr(str, st1);
        if (s != NULL)
        {   // номер позиции вхождения подстроки в строку
            int n = s - str;
            strdelsub(str, st1);   // удаление подстроки st1
           // вставка подстроки s2 в строку str  с номера n
            strinssub(str, st2, n, MAX_);
        }
    }
    return str;
}
```

ПРИМЕР 18. Передача массива строк

```
#include <iostream>  // for cin cout

using namespace std;

void print (char *m[ ]);        // прототип функции
void main ()
{
     char *s[] = {"odin", "dwa", "try", 0};
     print (s);
}

                                    //определение функции
void print (char *m[ ])
{
     for (int i = 0; m[i];  ++i)
          cout << m[i] <<endl;
}
```

ПРИМЕР 19. Передача двумерного массива

Вычисление суммы элементов матрицы.

```
#include <iostream>  // for cin cout

using namespace std;

const int n = 2;   //глобальные переменные – размеры матрицы
const int m = 3;

int adder  (int [n][m]);     //прототип функции

void main()
{
     int isum;
     int  iarray[n][m] = {5, 1, 6, 20, 15, 0};
     int  iarray1[n][m]= {1, 2, 3, 4, 5, 6};

     isum= adder(iarray);
     cout << "isum=" << isum <<  endl;

     cout << "isum=" << adder(iarray1) <<  endl;
}


            //определение функции   сумма элементов матрицы
int adder  (int iarray[n][m])
{
     int ipartial = 0;
     for (int i = 0; i < n;  i++)
          for (int j = 0; j < m; j++)
               ipartial += iarray[i][j];
     return (ipartial);
}
```

ПРИМЕР 20. Передача двумерного массива. Можно опустить первую размерность матрицы

Вычисление суммы элементов матрицы.

```
#include <iostream>  // for cin cout

using namespace std;

const int n = 2;     //глобальные параметры – размеры матрицы
const int m = 3;

                // можно опустить первую размерность массива
int adder  (int [][m]);

int main()
{
     int isum;
     int  iarray[n][m] = {5, 1, 6, 20, 15, 0};
     int  iarray1[n][m] = { 1, 2, 3, 4, 5, 6};

     isum= adder(iarray);
     cout << "isum=" << isum <<  endl;

     cout << "isum=" << adder(iarray1) <<  endl;
     return 0;
}

             //определение функции сумма элементов матрицы
int adder  (int iarray[][3])
{
     int ipartial = 0;

     for (int i = 0; i < n;  i++)
          for (int j = 0;  j < m;  j++)
               ipartial += iarray[i][j];
     return (ipartial);
}
```

ПРИМЕР 21. Передача двумерного массива. Другой способ передачи адреса матрицы в функцию

Вычисление суммы элементов матрицы.

```
#include <iostream>  // for cin cout

using namespace std;

const int n = 2;  //глобальные переменные – размеры матрицы
const int m = 3;
                              //функция не возвращает значения,
void adder  (int [ ][m], int *);
                    // для возврата значения- второй параметр
void  main()
{
     int isum;
     int  iarray[n][m] = {5, 1, 6, 20, 15, 0};

             //вызов функции,
             // передача второго параметра для возврата суммы
     adder(iarray, &isum);
     cout << "isum=" << isum <<  endl;
}

       //определение функции
       // другая запись для передачи адреса матрицы в функцию
void adder  (int (*piarray)[m], int *pipartial)
{
     *pipartial = 0;
     for (int i = 0; i < n;  i++)
         for (int j = 0; j < m;  j++)
             *pipartial += piarray[i][j];
}
              //  return не нужен - результат в pipartial
```

ПРИМЕР 22. Передача двумерного массива. Передача первой размерности матрицы через параметр

Вычисление суммы элементов матрицы.

```
#include <iostream>  // for cin cout

using namespace std;

const int n = 2;  // глобальные переменные – размеры массива
const int m = 3;

void adder  (int (*)[m],  int *, int);

int main()
{
     int  iarray[n][m] = {5, 1, 6, 20, 15, 0};
     int isum, i = 1;

     adder(iarray, &isum, i);
     cout << "isum=" << isum <<  endl;
     return 0;
}

         //определение функции
void adder  (int (*piarray)[m], int *pipartial, int k)
{
     *pipartial = 0;
        // параметр k получил значение, равное значению
        // переменной i из main()
     for (int i = 0; i < k;  i++)
        for (int j = 0;  j < m;  j++)
                  *pipartial += piarray[i][j];
}
```

ПРИМЕР 23. Передача двумерного массива. Передача размерностей матрицы через параметры

Вычисление суммы элементов матрицы.

```
#include <iostream>  // for cin cout

using namespace std;

void main()
{
    int isum;
                //прототип функции с параметром
                // «указатель на указатель на целое»
    int adder  (int **, int, int);

   int  iarray[2][3]={  5,   1,  6,   //элементы матрицы
                      20, 15,   0};

 // определение и инициализация  вспомогательного массива
 // указателей имя которого –
//  «указатель на указатель на целое»
   int *par[ ]={&iarray[0][0],  &iarray[1][0]};

  //вызов функции и передача ей адреса массива указателей
 // т.е. адреса нулевой строки
    isum = adder(par, 2, 3);

    cout << "isum=" << isum <<  endl;
}

    //заголовок функции - полное соответствие прототипу
int adder  (int **mas, int n, int m)
{
    int ipartial = 0;
    for (int i = 0; i < n;  i++)
        for (int j = 0; j < m;  j++)
            ipartial += mas[i][j];
    return (ipartial);
}
```

ПРИМЕР 24. Передача двумерного массива. Передача размерностей матрицы через параметры

Вычисление суммы элементов матрицы.

```
#include <iostream>  // for cin cout

using namespace std;

void main()
{
    int isum;
                    //прототип функции с параметром
                    // «указатель на указатель на целое»
   int adder  (int **, int, int);

   int  iarray[2][3]={5,1,6,20,15,0};  //элементы матрицы

      //определение и инициализация  вспомогательного
      // массива указателей
      // имя которого -  «указатель на указатель на целое»
    int *par[ ]={iarray[0], iarray[1]};

   // вызов функции и передача ей адреса массива указателей
   // т.е. адреса нулевой строки
    isum = adder(par, 2, 3);

    cout << "isum=" << isum <<  endl;
}

  //заголовок функции - полное соответствие прототипу
int adder  (int **mas, int n, int m)
{
    int ipartial = 0;
    for (int i = 0;  i < n;  i++)
        for (int j = 0;  j < m;  j++)
            ipartial += mas[i][j];
    return (ipartial);
}
```

ПРИМЕР 25. Передача двумерного массива. Передача размерностей матрицы через параметры

Вычисление суммы элементов матрицы.

```
#include <iostream>  // for cin cout

using namespace std;

//прототип функции с параметром «массив указателей на целое»
int adder  (int*  [ ], int, int);

void main()
{
     int isum;
     int  iarray[2][3]={5,1,6,20,15,0};

                    // определение и инициализация
                    // вспомогательного массива указателей
     int *par[]={&iarray[0][0], &iarray[1][0]};

     //вызов функции и передача ей адреса нулевой строки
     isum = adder(par, 2, 3);

     cout << "isum=" << isum <<  endl;
 // или  cout << "isum=" << adder(par, 2, 3) <<  endl;

}

                //заголовок функции  соответствует прототипу
int adder  (int *mas[ ], int n, int m)
{
     int ipartial = 0;
     for (int i = 0; i < n;  i++)
          for (int j = 0;  j < m;  j++)
               ipartial += mas[i][j];
     return (ipartial);       //возврат значения в функцию
}
```

ПРИМЕР 26. Передача двумерного массива. Передача размерностей матрицы через параметры

Вычисление суммы элементов матрицы.

```
#include <iostream>  // for cin cout

using namespace std;

//прототип функции с параметром «массив указателей на целое»
int adder  (int*  [ ], int, int);

void main()
{
     int isum;
     int  iarray[2][3]={5,1,6,20,15,0};

               //определение и инициализация
               // вспомогательного массива указателей
     int *par[]={iarray[0], iarray[1]};

      //вызов функции и передача ей адреса нулевой строки
     isum = adder(par, 2, 3);
     cout << "isum=" << isum <<  endl;
 // или  cout << "isum=" << adder(par, 2, 3) <<  endl;

}
                  //заголовок функции  соответствует прототипу
int adder  (int *mas[ ], int n, int m)
{
     int ipartial = 0;
     for (int i = 0; i < n;  i++)
          for (int j = 0;  j < m;  j++)
               ipartial += mas[i][j];
     return (ipartial);         //возврат значения в функцию
}
```

ПРИМЕР 27. Инициализация параметра-ссылки

Функция с инициализацией параметра-ссылки

```
#include <iostream>  // for cin cout

using namespace std;

int n = -55;        // глобальная переменная

  //определение функции с инициализацией параметра-ссылки
void invert(int& k=n)
{
    cout << " ---  k= "<< k<<endl;
    k = -k;
}

void main()
{
    int a = 22, b = 66;
    double d = 3.33;
                                // вызов без параметра
    invert();
    cout << " n= " << n<<endl;

                                //  вызов с параметром
    invert(a);
    cout << <<"a = " << a<<endl;
    cout << " n = " << n<<endl;

    //   invert(a+b);
    //   ошибка компиляции: выражение не имеет адреса
    //   invert (d);
    //   ошибка компиляции разный тип
}
```

ПРИМЕР 28. Возврат ссылки

```
#include <iostream>  // for cin cout

using namespace std;

int x=0;                          // глобальная переменная

int& setx();                     // прототип функции

int main()
{
// функция вызывается из левой части оператора присваивания
        setx() = 92;

    // вывод нового значения глобальной переменной
    cout << "x=" << x << endl;
    return 0;
}


int&  setx()
{
    return x;           // возвращает ссылку на x
}
```

ПРИМЕР 29. Возврат ссылки на элемент массива

Функция определяет ссылку на элемент массива с максимальным значением.

```
#include <iostream>  // for cin cout

using namespace std;

    //  прототип функции возвращающей ссылку на элемент,
   //   имеющий максимальное значение
int& rmax(int n, int d[])
{
    int im = 0;
    for (int i = 1; i < n;  i++)
        im = d[im] > d[i] ? im : i;
    return d[im];
}

void main ()
{
    int n = 4;
    int x[] = {10, 20, 30, 14};

                  //определение максимального элемента
    cout << " max = " << rmax(n,x);
    cout << endl;

// «левосторонний »вызов функции позволяет занести 0 в x[2],
//  являющийся максимальным элементом массива
    rmax(n, x) = 0;

    for (int i = 0; i < n;  i++)
        cout << "   "<<x[i];
    cout << endl;
}
```

ПРИМЕР 30. Классы памяти

Пример демонстрирует использование спецификаторов класса памяти при объявлении переменных.

```
#include <iostream>  // for cin cout
using namespace std;

int a1 = 111;                        // глобальная переменная
void fun();//  объявление функции fun без параметров

void main ()
{    extern int   a1;
     static int   a2;           // по умолчанию =0
     register int ra = 0;
     int          a3 = 0;       // автоматическая переменная
     cout<<" a1= "<<a1<<endl;              // 111
     cout<<" ra= "<<ra<<endl;              // 0
     cout<<" a2= "<<a2<<endl;              // 0
     cout<<" a3= "<<a3<<endl<<endl;        // 0

     fun();                              // вызов функции f1
     fun();                              // вызов функции f1

     cout<<" a1= "<<a1<<endl;              // 111
     cout<<" ra= "<<ra<<endl;              // 0
     cout<<" a2= "<<a2<<endl;              // 0
     cout<<" a3= "<<a3<<endl<<endl;        // 0
}
                                 // определение функции  f1
void fun()
{
     int  a1= 11;      // создание новой локальной переменной
     register int ra = 1;
               // создание новой статической переменной
               // (сохраняется значение)
     static int a2 = 22;

     a2 += 2;
     a1 += 2;
     cout<<" a1= "<<a1<<"  "<<::a1<<endl;
     cout<<" a2= "<<a2<<endl;
     cout<<" ra= "<<ra<<endl<<endl;
}
```

ПРИМЕР 31. Перегрузка функций (равное количество аргументов)

Пример программы, в которой перегружаются две функции с равным числом аргументов и имеющие одинаковые имена. Первая функция int func (int a, int b) определяет сумму двух целых чисел a и b, вторая - float func (float a, float b) – разность двух чисел с плавающей точкой.

```
#include <iostream>  // for cin cout
using namespace std;
         //прототип функции вычисления суммы целых чисел
int  func  (int a, int b);
        //прототип функции вычисления разности вещ. чисел
float func (float a, float b);

void main ()
{
    int a = 25, b = 40;
    float c = 2.45f,  d = 6.15f;

          //вызов функции с аргументами целого типа
    cout <<  "\n summa=" <<  func (a,b)<<endl;

        //вызов функции с аргументами типа float
    cout <<  "\n raznost=" <<  func (c,d)<<endl;
}

         //определение функции вычисления суммы целых чисел
int func(int a, int b)
{
    return (a + b);
}

        //определение функции вычисления разности вещ. чисел
float func(float a, float b)
{
   return (a - b);
}
```

ПРИМЕР 32. Перегрузка функций (разное количество аргументов)

Пример перегрузки функций, имеющих разное количество параметров. Функции выполняют вывод на экран символа (заданного в виде аргумента или определенного в функции) некоторое количество раз (заданное в качестве аргумента или определенное в функции).

```
#include <iostream>  // for cin cout
using namespace std;
  // прототип функции для вывода символа '*' 45 раз
void repchar();
  // прототип функции для вывода символа-аргумента 45 раз
void repchar(char);
  // прототип функции для вывода символа-аргумента заданное
  // через второй аргумент количество  раз
void repchar(char, int);

void main()
{            // вызов функции для вывода символа '*' 45 раз
    repchar();
             // вызов функции для вывода символа '=' 45 раз
    repchar('=');
             // вызов функции для вывода символа '+' 30 раз
    repchar('+', 30);
}
           // определение функции вывода символа '*' 45 раз
void repchar()
{   for (int j = 0; j < 45; j++)
       cout << '*';
    cout << endl;
}
     // определение функции вывода символа-аргумента 45 раз
void repchar(char ch)
{   for (int j = 0; j < 45; j++)
               cout << ch;
    cout << endl;
}
  // определение функции вывода символа-аргумента заданное
  // через второй аргумент количество  раз
void repchar(char ch, int n)
{
    for(int j = 0;  j < n; j++)
        cout << ch;
    cout << endl;
}
```

ПРИМЕР 33. Перегрузка функций (переменное количество аргументов)

Рассмотрим функцию, которая принимает переменное число аргументов и выводит на экран их количество и принятые значения. Число действительно переданных значений задаёт первый аргумент. Передаваемые аргументы располагаются в стеке подряд после первого аргумента, поэтому их можно извлечь с помощью арифметики указателей. Для 32 битной платформы размер каждого из аргументов округляется в большую сторону кратно 4 байтам, для 64 битной платформы – кратно 8 байтам.

```
#include <iostream>   // cin cout
#include <iomanip>   // cin cout
using namespace std;

void example (int,...);

int main(void)
{
    int var1 = 5, var2 = 6, var3 = 7,
        var4 = 8, var5 = 9;
    example(1, var1);         //вызов функции с
                              // одним аргументом
    example(2, var1, var2); //вызов функции с
                              // двумя аргументами
    example(5, var1, var2, var3, var4, var5);
                              // с пятью аргументами
    return 0;
}
void example(int arg1, ...)
{
    int *ptr = &arg1;  // указатель на количество аргументов
    // смещаемся на первый аргумент
    ptr += sizeof(void*) / sizeof(int);
          // в зависимости от битности платформы (32 или 64)
          // сдвигаемся либо на 4 байта, либо на 8,
          // то есть на один int или на два.
    cout << " peredano argumentov: " << arg1 << endl;
                            // (не включая счётчик аргументов)
    for ( ; arg1 > 0; arg1--)
    {
        cout << *ptr << "  ";
        // смещаемся по стеку на следующий аргумент
        ptr += sizeof(void*) / sizeof(int);
    }
    cout << endl;
}
```

ПРИМЕР 34. Перегрузка функций (переменное количество аргументов)

Рассмотрим функцию, которая принимает переменное число аргументов и выводит на экран их количество и принятые значения. Функция использует 0 как индикатор завершения списка переданных аргументов:

```
#include <iostream>   // cin cout
#include <iomanip>   // cin cout

using namespace std;

void example (int, ...);

int main(void)
{
    int var1 = 5, var2 = 6, var3 = 7,
        var4 = 8, var5 = 9;
    example(var1, 0);         // вызов функции с
                              // одним аргументом
    example(var1, var2, 0); // вызов функции с
                              // двумя аргументами
    example(var1, var2, var3, var4, var5, 0);
                              // с пятью аргументами
    return 0;
}

void example(int arg1, ...)
{
    int number = 1;
    int *ptr = &arg1; // установка указателя на
                      // первый аргумент

    while (*ptr)
    {
        number++;
        cout << *ptr;
        // смещаемся по стеку на следующий аргумент
        ptr += sizeof(void*) / sizeof(int);
    }
    cout << endl
         << " peredano argumentov" << number - 1 << endl;
}
```

ПРИМЕР 35. Шаблон функций

Определим шаблон семейства функций, вычисляющих абсолютное значение числовых величин разных типов:

```
#include <iostream>  // for cin cout

using namespace std;
                          //шаблон семейства функций T abs(T n)
template <typename T>
T abs (T n)
{
    return (n < 0) ? -n : n;
}

void main()
{          //иниализация переменных разных типов и знаков
    int    i1 =  5;
    int    i2 = -6;
    long   l1 =  70000L;
    long   l2 = -80000L;
    double d1 =   9.95;
    double d2 = -10.15;

                                        //вызовы функций
    cout << "abs(" << i1 << ")=" << abs(i1) << endl;
                                                //abs(int)
    cout << "abs(" << i2 << ")=" << abs(i2) << endl;
                                                //abs(int)
    cout << "abs(" << l1 << ")=" << abs(l1) << endl;
                                                //abs(long)
    cout << "abs(" << l2 << ")=" << abs(l2) << endl;
                                                //abs(long)
    cout << "abs(" << d1 << ")=" << abs(d1) << endl;
                                                //abs(double)
    cout << "abs(" << d2 << ")=" << abs(d2) << endl;
                                                //abs(double)
    cout << endl;
}
```

ПРИМЕР 36. Шаблон функций с несколькими типами данных

Определим шаблон семейства функций, выводящих на экран значения переменных разных типов

```
#include <iostream>     // for cin cout

using namespace std;
                        //шаблон семейства функций
template <typename type1, typename type2>

void myfunc(type1 x, type2 y)
{
     cout << x << "  " << y << endl;
}

int main()
{
    myfunc(10, "hi");

    myfunc(0.23, 10L);

    return 0;
}
```

ПРИМЕР 37. Рекурсивные функции. Вычисление факториала

Рекурсивное определение факториала (сравните с примером 8 вычисление факториала через цикл).

0! = 1
для любых n > 0 n ! = n * (n - 1)!

```
#include <iostream>              // for cin cout
using namespace std;

double  Rec_Fact_Up (int);     // прототип функции
double  Rec_Fact_Dn (double, int, int); // прототип функции

void main()
{    int i = 5;
     double Fact;
     Fact = Rec_Fact_Up (i);       // вычисление факториала
     cout << i << " != " <<  Fact << endl;
     int n = i;
                                  // вычисление факториала
     Fact = Rec_Fact_Dn (1.0, 1, n);
     cout <<i << " ! = " <<  Fact << endl;
}

double  Rec_Fact_Up (int n)
{    if (n <= 1)
        return 1.0;
     else
        return  Rec_Fact_Up(n-1) * n;
     //вычисление на рекурсивном возврате
     // Mult=Rec_Fact_Up(n-1) -  рекурсивный вызов;
}    // Mult *n - оператор накопления факториала

double  Rec_Fact_Dn(double Mult, int i, int m)
{
     Mult = Mult * i;
     if (i == m)
       return Mult;
     else
       return  Rec_Fact_Dn (Mult, i+1, m);
     //вычисление на рекурсивном спуске
     // Rec_Fact_Dn() -  рекурсивный вызов;
     // Mult=Mult *i - оператор накопления факториала
}
```

ПРИМЕР 38. Рекурсивная функция вывода на экран строки символов в обратном порядке

```
#include "stdafx.h"
#include <iomanip>
#include <iostream>              // cin cout

using namespace std;

void  Reverse ();                // прототип функции

int main()
{
    cout << "Input string:";
    Reverse();
    return 0;
}

void  Reverse ()         // рекурсивная функция
{
    char ch;
    cin.get(ch);
    if (ch != '\n')
    {
        Reverse();         // рекурсивный вызов функции
             //действия выполняются на рекурсивном возврате
        cout << ch;
    }
}
```

ПРИМЕР 39. Рекурсивная функция возведения вещественного числа X в целую степень N >= 0

Программа реализует алгоритм возведения вещественного числа X в целую степень N >= 0 за минимальное число операций умножения.

```
#include "stdafx.h"
#include <iomanip>
#include <iostream>  // cin cout
using namespace std;

double rec_degree(double, int );

int main ()
{
    double x, y;
    int n;
    cout << " X= ";
    cin >> x;
    cout <<" N = ";
    cin >> n ;
    if (n > 0)
    {
        y = rec_degree(x, n);
        cout << x << " v stepeni  " << n << " = " << y
                                                  << endl;
    }
    return 0;
}
                              // рекурсивная функция
double rec_degree(double x,  int n)
{
    if (!n)
        return 1;
    if ( !( n% 2))
    {                                    //n - четное
        double r = rec_degree(x, n/2);
                           // рекурсивный вызов функции
          //действия выполняются на рекурсивном возврате
        return r*r;
    }
    else                                 //n - нечетное
        return x *rec_degree(x, n-1);
                           // рекурсивный вызов функции
         //действия выполняются на рекурсивном возврате
}
```

ПРИМЕР 40. Рекурсивная функция вывода на экран число в виде строки символов

```
#include <iostream>   // cin cout
using namespace std;
void printd (int);

int main ()
{
    int a;
    cin >> a;
    if (a < 0)
    {
        cout<<'-';
        a = -a;
    }
    printd(a);
    cout <<  endl;
    return 0;
}
                                // рекурсивная функция
void printd (int n)
{
    if (n)
    {
        printd(n/10);         // рекурсивный вызов функции
           //действия выполняются на рекурсивном возврате
        cout << (char(n % 10 + '0'));
    }
}
```

ПРИМЕР 41. Вычисление НОД двух чисел (по алгоритму Евклида)

```
#include <iostream>
using namespace std;
int NODV(int a, int b)     // Алгоритм Евклида с вычитанием
{
    while (a != b)
    {
        if (a > b)
            a -= b;
        else
            b -= a;
    }
    return a;
}
//-----------------------------------------------
int NODV_rec(int a, int b)  // Алгоритм Евклида с вычитанием
{                           // рекурсивный
    if (a == b)
        return a;
    if (a > b)
        return NODV_rec(a - b, b);
    return NODV_rec(a, b - a);
}
//-----------------------------------------------
int NODD1(int a, int b)    // Алгоритм Евклида с делением
{
    while (a && b)
    {
        if (a > b)
            a %= b;
        else
            b %= a;
    }
    return a + b;
}
//-----------------------------------------------
int NODD2(int a, int b)    // Алгоритм Евклида с делением
{                          // оптимизированный
    while (b)
    {
        int c = a % b;
        a = b;
        b = c;
    }
    return a;
}
```

```
//-------------------------------------------
int NODD_rec1(int a, int b)  // Алгоритм Евклида с делением
{                            // рекурсивный
    if (b == 0)
        return a;
    return NODD_rec1(b, a % b);
}

//-------------------------------------------
int NODD_rec2(int a, int b)  // Алгоритм Евклида с делением
{                            // рекурсивный
    return b? NODD_rec2(b, a % b) : a;
}

//-------------------------------------------
int NOD(int a, int b)        // Алгоритм Евклида с делением
{
    while(b)
        b ^= a^ = b^ = a %= b;
    return a;
}

void main()
{
    int a, b;
    cin >> a;
    cin >> b;
    if (a <= 0 || b <= 0)
        cout << "Error" << endl;
    else
    {
        cout << NODV(a, b) << endl;
        cout << NODV_rec(a, b) << endl;
        cout << NODD1(a, b) << endl;
        cout << NODD2(a, b) << endl;
        cout << NODD_rec1(a, b) << endl;
        cout << NODD_rec2(a, b) << endl;
        cout << NOD(a, b) << endl;
    }
}
```

ПРИМЕР 42. Рекурсивная функция вычисления чисел Фибоначчи

```
#include <iostream>   // cin cout
using namespace std;

unsigned long fib (unsigned long);

int main ()
{ unsigned long number;
  cout << " Input number";
  cin >> number;
  cout << number<<" chislo Fib = "<< fib(number) << endl;
 return 0 ;
}

                        // рекурсивная функция
unsigned long fib (unsigned long n)
{
    if (n == 0 || n == 1)
       return n;
    else
       return  fib(n - 1) + fib(n - 2);
          // рекурсивный вызов функции
         //действия выполняются на рекурсивном возврате
}
```

ПРИМЕР 43. Рекурсивная функция вычисления суммы элементов числовой последовательности

```
#include <iostream>   // cin cout
#include <iomanip>   // cin cout

using namespace std;

double sum (int [],  int);

int main ()
{
  const int nn =100;
  int array[nn], n;
  cout<<" n= ";
  cin>>n;

  for (int i = 0; i < n; i++)
  {   array[i] = rand() % 20;
      cout << setw(3) << array[i];
  }
cout <<endl;
cout << sum (array, n) << endl ;
return 0 ;
}
                                  // рекурсивная функция
double sum (int s[], int n)
{
    if (n == 1)
      return s[0];
    else
      return sum(s, n-1) + s[n-1];
           // рекурсивный вызов функции
          //действия выполняются на рекурсивном возврате
}
```

ПРИМЕР 44. Передача указателей на функции

```
#include <iostream>  // for cin cout
#include <math.h>
using namespace std;

// прототип функции:  первый параметр – указатель на функцию
double calc_fun (double (*p_f)(double), double x);

void main()
{    double (*p_func)(double) = NULL;
     double  x;

     cout<<" x= "<<endl;
     cin>>x;
     p_func = sin;              // фактический параметр  sin
     cout << "sin("<<x<<")= "<<p_func(x)<<endl;

     cout<<" x= "<<endl;
     cin>>x;
     p_func = &sin;     //  или так фактический параметр  sin
     cout << "sin("<<x<<")= "<< p_func(x) << endl;

     cout<<" x= "<<endl;
     cin>>x;
               //  вычисление через функцию calc_fun :
               // фактический параметр  cos
     cout << "cos("<<x<<")= "<<calc_fun(cos, x) << endl;

     cout<<" x= "<<endl;
     cin>>x;
              //  или так вычисление через функцию calc_fun :
              //  фактический параметр  &cos
     cout<<"cos("<<x<<")= "<<calc_fun(&cos,x)<<endl;
}

     //  определение  функции
     //  первый  параметр -  указатель на функцию
      //  второй  параметр – аргумент функции
double calc_fun (double (*p_f)(double), double x)
{
     return p_f(x);
      //  возвращаем результат : значение функции p_f от x
}
```

ПРИМЕР 45. Решение нелинейного уравнения x=f(x) методом простых итераций

Постановка задачи
Методом последовательных приближений найти решение уравнения $x=f(x)$ с заданной точностью r.

Математическая модель .
Выберем начальное приближение $x0$
Положим $x1 = f(x0);$
$x2 = f(x1);$ ... $xn = f(xn);$
Погрешность $d = |xn-1 - xn|$
Условие сходимости: $|f'(x)| < 1$

Параметры:

Заданная точность	double r
Начальное приближение	double x
Указатель на функцию	double (*p_func)(double);
Максимальное число итераций	integer max_i
Достигнутая точность	double d

Возвращаемое значение: double x
Внутренние переменные:

Текущее значение $xn-1$	double x
Текущее значение $f(xn-1)$	double y

Алгоритм.

1.Задать x = x0, r
2 Если r < 1e-12, положить r = 1e-12
3. Инициализировать d = 1
4. Пока d > r и число итераций не больше максимального выполнять
    4.1. Вычислить y = f(x)
    4.2. Положить d = |x-y|
    4.3. Положить x = y

5. Вывести значение x

```
#include <math.h>
#include <iomanip>
#include <iostream>  // for cin cout
using namespace std;

double solveqn (double r, double x,
            double(*p_func)(double), int max_i, double &d);
double myfun(double x);

void main()
{
    double d = 0;

cout << setprecision(12)<< solveqn(1e-5,1.0,cos,10000, d);
cout << " d = " << setprecision(12) << d << endl<<endl;

cout << setprecision(12)<<solveqn (1e-5,10.0,myfun,10000,d);
cout << " d = " << setprecision(12) << d << endl<<endl;
}
                          // определение функции
double myfun(double x)
{
    return sin(x);
}
                                  // Решение уравнения f(x)=0
double solveqn  (double r, double x,
              double (*p_func)(double), int max_i, double&d)
            // r     -  требуемая точность (<=1e-12),
            // x     -  начальное приближение
            // p_func -  указатель на функцию f(x)
            // max_i  -  максимальное количество итераций
            // d      -  достигнутая точность
{
    double y;
    if (r < 1e-12)
      r = 1e-12;
    d =1;
    for (int i = 1; (i <= max_i) && (d > r);  i++)
    {
        y = p_func(x);
        d = fabs(x - y);
        x = y;
    }
    return y;
}
```

ПРИМЕР 46. Вычисление интегралов

```
#include <math.h>
#include <iostream>  // for cin cout
using namespace std;

                  //прототип функции вычисления интеграла
                  // по критерию двойного пересчета
void Vych_Int(double a,double b,double eps,
              double(*pf)(double),double &I,int &k);
                  //прототип функции вычисления интеграла
                  // по методу левых прямоугольников
void Sum(double a,double b,double h,
         double (*pf)(double),double &S);
                 //прототипы подынтегральных функций
double  f1  (double);
double  f2  (double);

void main ()
{    double a, b, eps;  // отрезок интегрирования,  точность
     double Int;        // значение интеграла
     int K_iter;        // количество итераций

     cout << "Input a, b, eps"<<endl;
     cin >> a >> b >> eps;
                    //вычисление интеграла для f1
     Vych_Int(a, b, eps, &f1, Int, K_iter);
     cout << "Integral    for f1  =" << Int <<
             " K_iter= "<< K_iter << endl;

                    //вычисление интеграла для f2
     Vych_Int(a, b, eps, &f2, Int, K_iter);
     cout << "Integral    for f2  =" << Int <<
             "  K_iter = " << K_iter << endl;
}

                                  // подынтегральные функции
double  f1 (double x)
{
          return cos (x) / exp(0.3 * log(x));
}

double  f2 (double x)
{
       return  cos (x * x * x) / sqrt(sqrt(x));
}
```

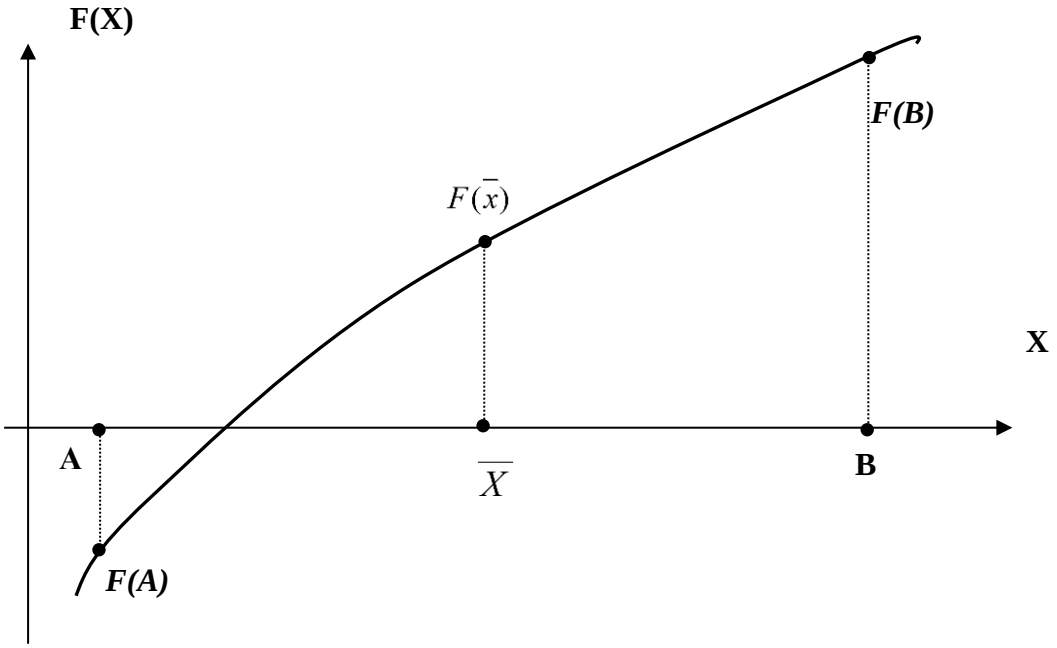
```
                // функция вычисления интеграла
                // по критерию двойного пересчета
void Vych_Int (double a,double b,double eps,
                double (*pf)(double),double &I,int &k)
{
     int n = 4;//инициализация  количества разбиений
                //определение шага интегрирования
     double h = (b - a) / n;
      // переменные для значений сумм с шагом h и с шагом h/2
     double S1 = 0, S2 = 0;

      /вызов функции Sum с шагом h: в S1 возвращается сумма
     Sum(a, b, h, pf, S1);
     k = 0;
                // запуск процесса двойного пересчета
     do
     {       // сохраняем значение интеграла предыдущего шага
          S2 = S1;
          n *= 2; //увеличение количества отрезков разбиения и
                    //уменьшение шага интегрирования в 2 раза
          h = (b - a) / n;
                    //вызов функции Sum с шагом h=h/2
          Sum (a, b, h, pf, S1);
          k++;
     } while (fabs(S1 - S2) > eps);
     I = S1;
}

   // функция выч. интеграла по методу левых прямоугольников
void Sum(double a, double b, double h,
          double (*pf) (double), double &S)
{
     double x, sum;
     x = a;
     sum = 0;
     while (x < b)
     {
          sum = sum + (*pf)(x); //накопление суммы высот
          x = x + h;
     }
                                    //вычисление площади
     S = h * sum;
}
```

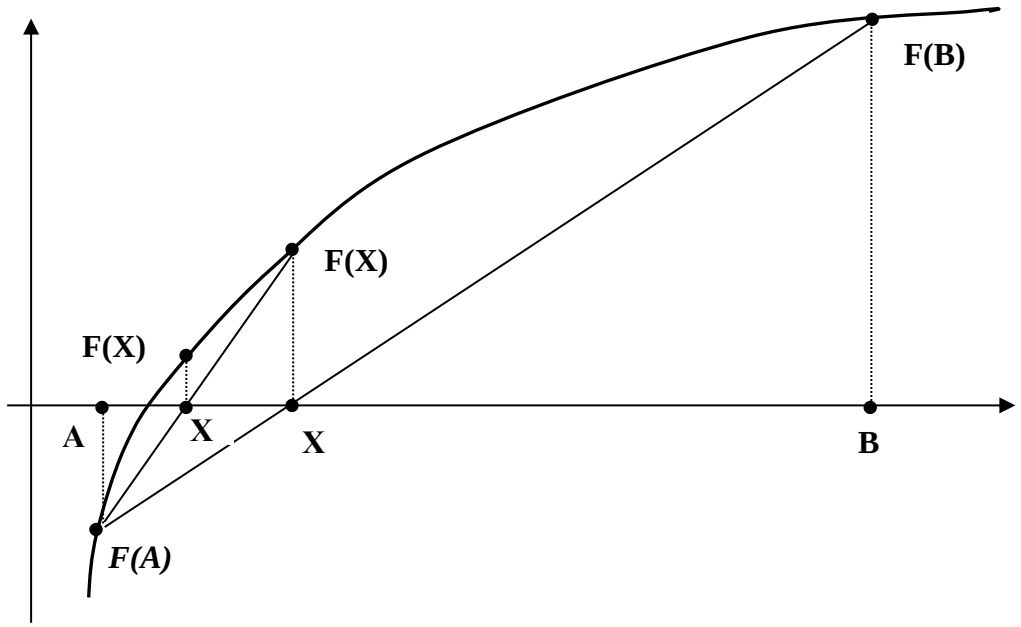
РЕШЕНИЕ НЕЛИНЕЙНЫХ УРАВНЕНИЙ

Метод деления отрезка пополам



1. Определяем середину отрезка [A, B] X = (A+B)/2. Вычисляем значение функции F(X).
2. Выбираем одну из двух частей отрезка для дальнейшего уточнения корня. Корень будет находиться в том отрезке, на концах которого функция меняет знак.
3. Если F(A)*F(X) < 0, то B = X , иначе A = X. Продолжаем процесс деления как с первоначальным отрезком [A, B].
4. Итерационный процесс продолжаем до тех пор, пока интервал [A, B] не станет меньше заданной погрешности, либо пока значение заданной функции в точке X не станет меньше заданной погрешности.
   | A - B | < eps либо | F(X) | < eps

Метод хорд

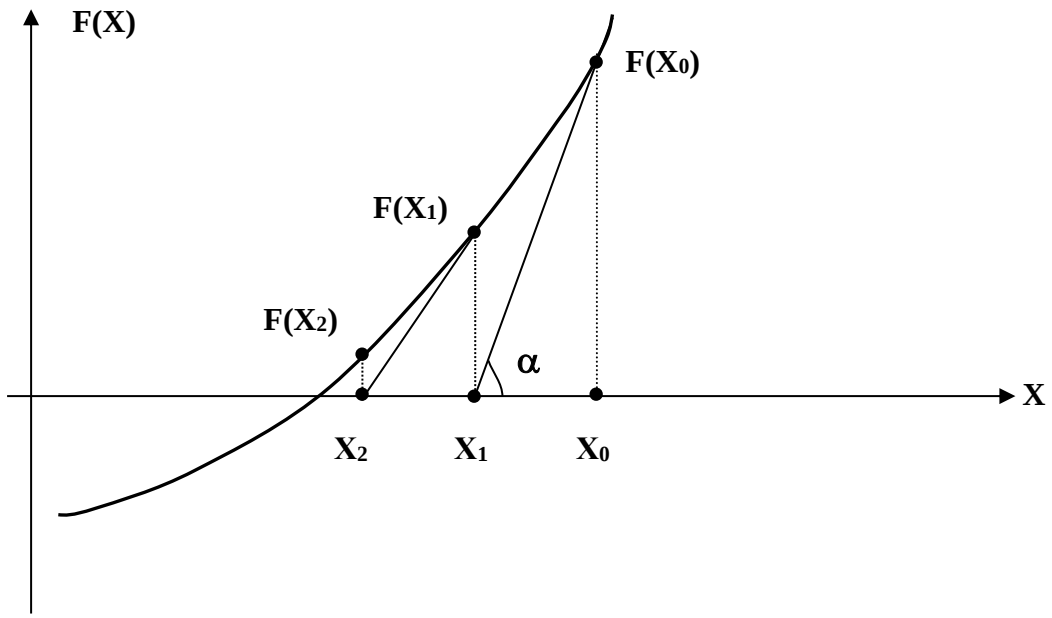


1. Интервал определяется графическим методом, очередное приближение берётся в точке X, где пресекает ось абсцисс прямая линия, проведённая через точки F(A) и F(B).

$$X = A - \frac{B-A}{F2-F1} \cdot F1 \quad \text{или} \quad X = B - \frac{B-A}{F2-F1} \cdot F2$$

2. В качестве нового интервала для продолжения итерационного процесса выбираем тот из отрезков [A, X] или [X, B], на концах которого функция принимает значения с разными знаками.
3. Итерационный процесс заканчивается по условию:
   $|A - B| < eps$ либо $|F(X)| < eps$

Метод Ньютона (касательных)



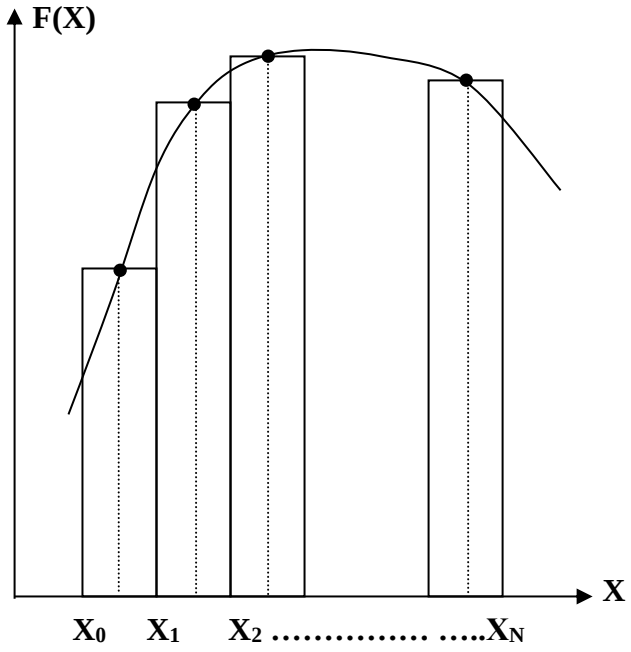
1. Пусть определено начальное приближение к корню $X_{0.}$
2. Следующее приближение к корню точка X1- точка пересечения касательной к функции F(X), проведенная через точку ($X_{0,}$ F_0), и оси абсцисс.

$$X_1 = X_0 - \frac{F(X_0)}{F'(X_0)}$$

3. Считаем точку X_1 в качестве начальной и продолжаем итерационный процесс.
4. С каждой итерацией расстояние между очередным X_{k+1} и предыдущим X_k приближениями к корню будут уменьшаться. Процесс уточнения заканчивается, когда когда выполняется условие:

$$| F(X_k) | < eps$$

ВЫЧИСЛЕНИЕ ИНТЕГРАЛОВ

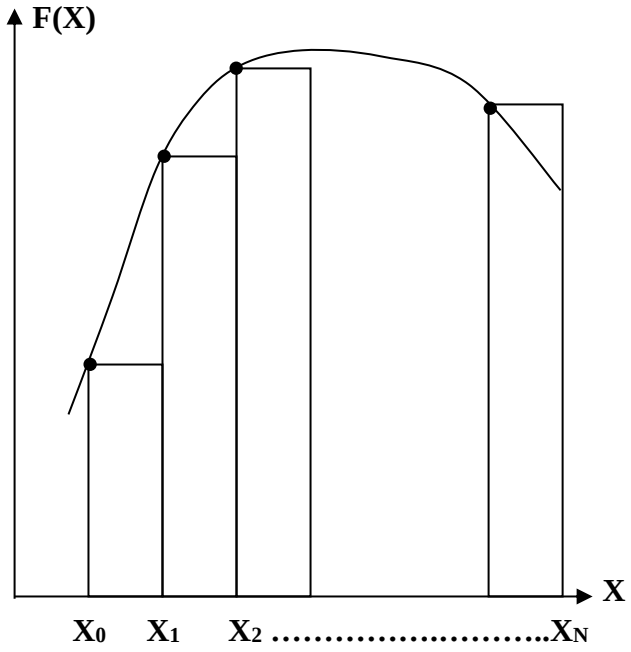


Формула средних прямоугольников

$X_0 = A; X_N = B \qquad h = (B-A)/N$

$X_i = X_{i-1} + h$

$$I = h \cdot \sum_{i=0}^{N-1} F(x_i + \frac{h}{2})$$

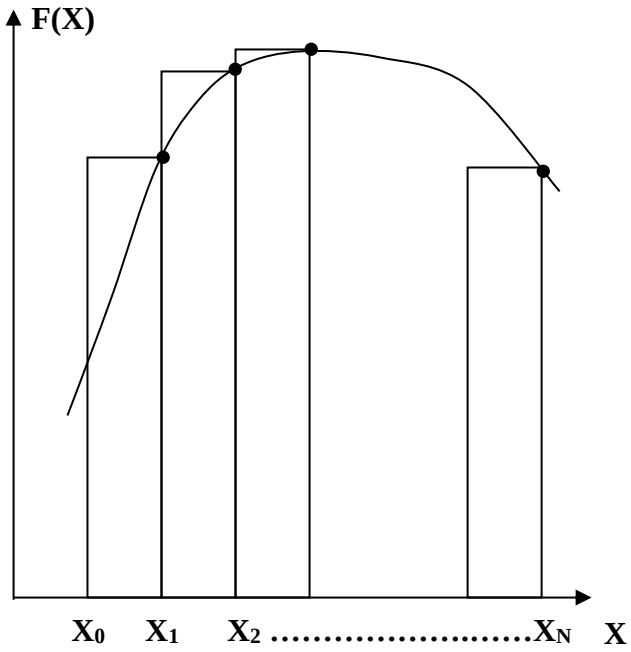


Формула левых прямоугольников

$X_0 = A, X_N = B \qquad h = (B-A)/N$

$X_i = X_{i-1} + h$

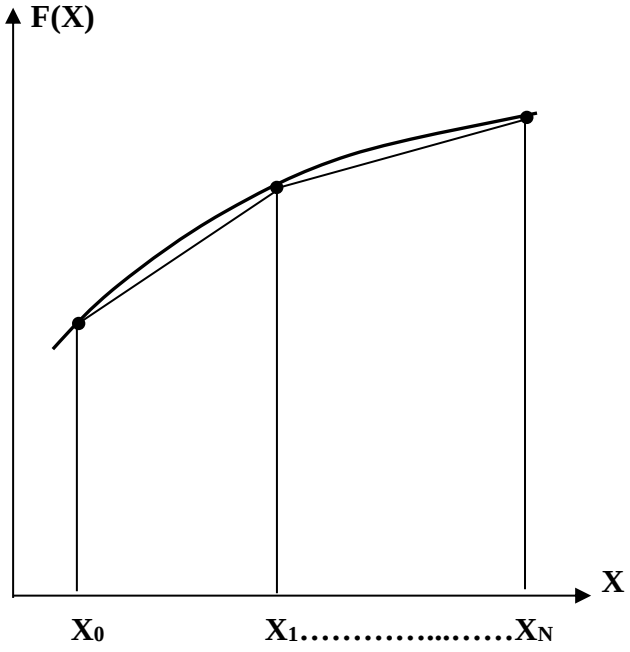
$$I = h \cdot \sum_{i=0}^{N-1} F(x_i)$$



Формула правых прямоугольников

$X_0 = A, X_N = B \qquad h = (B-A)/N$

$$I = h \cdot \sum_{i=1}^{N} F(x_i)$$

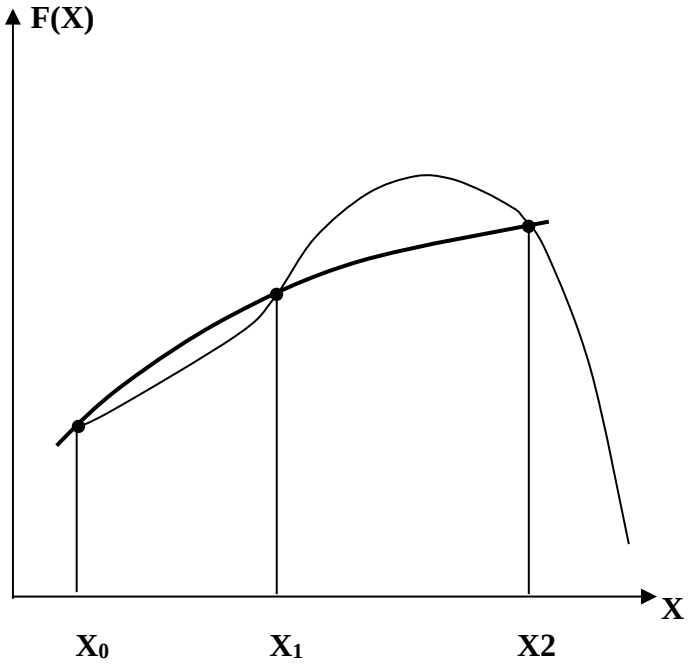


Формула трапеций

$X_0 = A, X_N = B \qquad h = (B-A)/N$

$X_i = X_{i-1} + h$

$$I = h \cdot \left[\frac{F(A) + F(B)}{2} + \sum_{i=1}^{N-1} F(x_i) \right]$$



Формула Симпсона

$X_0 = A, \quad X_N = B,$

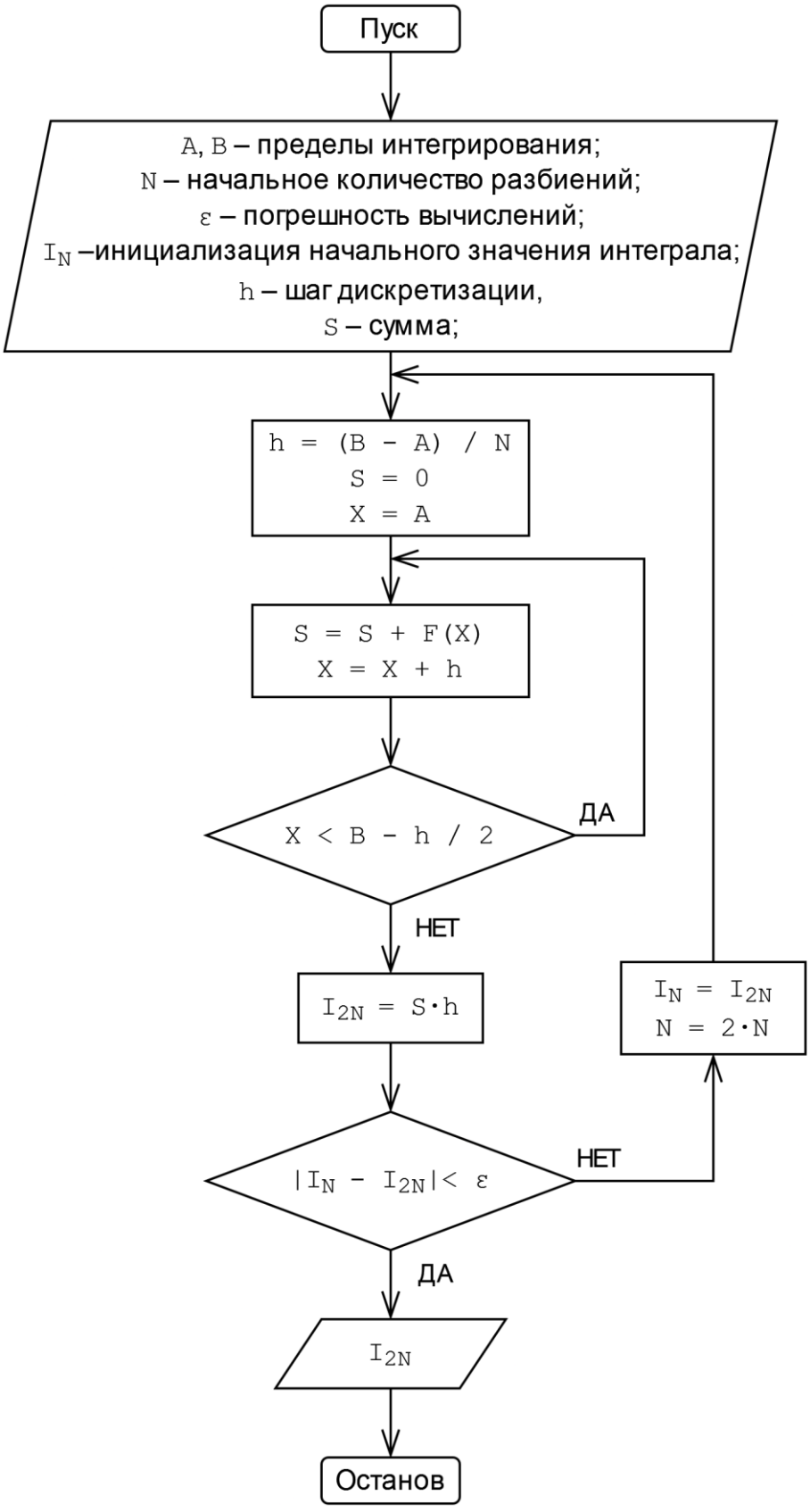
$Yi = F(X_i)$

$h = \frac{B - A}{2 \cdot N};$

$i = 0,1,2,\ldots, 2N; \quad Xi = A + h \cdot i$

$$\mathbf{I \approx \frac{h}{3} \cdot (Y_0 + Y_{2N} + 2 \cdot (Y_2 + Y_4 + \ldots + Y_{2N-2}) + 4 \cdot (Y_1 + Y_3 + \ldots + Y_{2N-1}))}$$

МЕТОД ДВОЙНОГО ПЕРЕСЧЁТА ДЛЯ ВЫЧИСЛЕНИЯ ИНТЕГРАЛОВ МЕТОДОМ ЛЕВЫХ ПРЯМОУГОЛЬНИКОВ



СЛОВАРЬ ПОНЯТИЙ, ИСПОЛЬЗУЕМЫХ В ЗАДАНИЯХ