*C++*

# // ЯЗЫК ПРОГРАММИРОВАНИЯ

# ПРИМЕР ПРОГРАММЫ НА ЯЗЫКЕ C++

Имя файла исходного кода программы - *First.CPP*
Программа *First* выводит сообщение на экран
и демонстрирует типичную структуру программы
на языке C++:

```
#include <iostream>
using namespace std;
 int  main()
{
   cout <<”У каждой эпохи свой язык \n” ;
   return 0;
}
```

# СТРУКТУРА ПРОГРАММЫ С++

- Функции представляют собой основу, на которой строится любая программа на С++. Программа *First* состоит из единственной функции с названием main();
- Каждая программа содержит одну или несколько функций;
- Каждая функция содержит 4 основных элемента:

1. тип возвращаемого значения; int
2. имя функции; main()
3. список параметров, заключённый в круглые скобки
4. тело функции { …… }

return 0; - - эта строка значит "*вернуть операционной системе в качестве сигнала об успешном завершении программы значение 0*".

# ОРГАНИЗАЦИЯ КОНСОЛЬНОГО – ВВОДА/ВЫВОДА ДАННЫХ (Т.Е. В РЕЖИМЕ ЧЁРНОГО ЭКРАНА)

```
#include <iostream>;    //директива  препроцессора, предназначена для включения в
```
*исходный текст содержимое заголовочного файла, имя которого< iostream>, содержащий описания функций стандартной библиотеки ввода/вывода для работы с клавиатурой и экраном.*

```
using namespace std;    //директива означ.что все определённые ниже имена будут отн-ся
```
*к пространству имён* ***std***

```
int  main()     //имя функции, которая не содержит параметров и должна возвращать
```
*значение типа  int (целый)*

```
{int a,b;        //объявление двух переменных  a ,b
```
*целого типа* ***int***

```
cout <<"введите два целых числа"<<endl;      //оператор вывода  данных ( в нашем случае
```
*- текстового сообщения ) на экран ,*

*<< - операция помещения данных в выходной поток;*

***endl*** *- манипулятор, который вставляет в символьный поток символ окончания строки, поэтому весь последующий текст будет печататься с новой строки.*

```
cin >>a >>b;     //оператор для работы со стандартным потоком ввода. Этот поток
```
*содержит данные, вводимые с клавиатуры.*

*>> - операция извлечения данных из входного потока, читает значения из* ***cin*** *и сохраняет их в переменных* a *и* b .

```
cout <<"их сумма равна"<<a+b<<endl;  //оператор вывода
return 0;}   //оператор , возвращающий значение  0  операционной системе, вызывающей
```
*функцию* main().

# СТАНДАРТНЫЕ ТИПЫ ДАННЫХ

- Целые типы данных – **short, int, long** и спецификаторы ***(signed,unsigned);***
- Вещественные типы – **float, double, long double;**
- Символьные типы – **char, wchar_t;**
- Логический тип – **bool**, принимающий значения (true-истина, false-ложь);

**Константы (const)**

*a=+2.5;* //неименованная константа;

*b=1L;* – целочисленная константа(тип long);

*'1L'*– целочисленный литерал (тип long);*'8'* – целочисл.литерал (тип int);

*'f'*, – символьный литерал(тип char),

***'\n'***- пример управляющей последовательности – переход на новую строку

Строковые константы записываются в двойных кавычках.

Формат описания именованной константы:

[<класс памяти>]const <тип> <имя именованной константы>=<выражение>;

```
const int I= - 124;
const float  k1=2.345, k=1/k1;
```

*Класс памяти*- это спецификатор, определяющий время жизни и область видимости данного объекта.

*Выражение* определяет значение константы, т.е. инициализирует её.

# ПЕРЕМЕННЫЕ

Формат описания переменных:
**[<класс памяти>]<тип><имя>[=<выражение>|(<выражение>)];**

*Пример*:

```
int I,j;
double x;
```

Значение переменных должно быть определено с помощью:

1. оператора присваивания:
```
int a; //описание переменной
a= 5; //присваивание значения переменной
```

2. оператора ввода:
```
int a; //описание переменной
cin>>a; // ввод значения в переменную
```

3. инициализация – определение значения переменной на этапе описания.
```
int i=100; //прямая инициализация
```

# УПРАВЛЕНИЕ ФОРМАТОМ ВЕЩЕСТВЕННЫХ ТИПОВ ДАННЫХ

Существуют три аспекта оформления значения с плавающей запятой, которыми можно управлять:

- *точность*( кол-во отображаемых цифр), изменяется с помощью манипулятора *setprecision*;
- *форма записи* (десятичная или экспоненциальная);
- *указание десятичной точки для значения с пл.запятой, являющихся целыми числами .*

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{ double i=12345.6789;
   cout << setprecision(3)<< i <<endl;
   cout << setprecision(6)<< i <<endl;
   cout << setprecision(9)<< i <<endl;
return 0;}
```

Результат работы программы:

```
1.23e+004
12345.7
12345.6789
```

- (для использования манипуляторов с аргументами (например, setprecision) требуется подключить заголовочный файл *iomanip*)

# УПРАВЛЕНИЕ РАЗМЕЩЕНИЕМ ДАННЫХ НА ЭКРАНЕ

- **Используются манипуляторы**:

1. left – выравнивает вывод по левому краю;
2. right – выравнивает вывод по правому краю;
3. internal – контролирует размещение отрицательного значения: выравнивает знак по левому краю, а значение по правому, заполняя пространство между ними пробелами;
4. setprecision(int w) – устанавливает max кол-во цифр в дробной части для вещественных чисел;
5. setw(int w) – устанавливает ширину поля для вывода данных;

*Пример:*

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
 {  cout <<“1.” <<setw(10) <<“Ivanov” <<endl;
  cout <<“2.” <<setw(10) <<left<<“Ivanov” <<endl;
  cout <<“3.” <<setw(10) <<right<<“Ivanov” <<endl;
 return 0;}
```

*Получим:*

```
1.     Ivanov
2.Ivanov
 3.    Ivanov
```

# ЗАДАНИЕ

- С помощью данных манипуляторов запишите программу, где выравнивание отрицательного числа -23,4567 будет только по правому краю.

Должно получиться:

1. -23,4567
2. -23,5
3. - 23,5

# УНАРНЫЕ ОПЕРАЦИИ : инкремент (++) и декремент (- -)

- *Операции увеличения (инкремента) и уменьшения (декремента) на 1(++ и --);* записываются в двух формах:

*Префиксия* – операция записывается перед операндом и **выполняется ранее** текущей операции, например ранее вывода значения переменной в справа приведенном примере

*Постфиксия* – операция записывается после операнда, и **выполняется после** текущей операции, например после вывода значения переменной в справа приведенном примере.

Пример:

```
#include <iostream>
using namespace std;
int main()
{ int x=3, y=4;
  cout <<++x<<'\t'<<--y<<endl;
  cout <<x++<<'\t'<<y--<<endl;
  cout <<x<<'\t'<<y<<endl;
  return 0;}
```

Результат:

```
 4   3  (x=4(3+1), y=3(4-1))
4 3  (x=5(4+1), y=2(3-1))
  5 2
```

# Операции отрицания (- и !)

- *(-) - унарный минус – изменяет знак операнда целого или вещественного типа на противоположный;*
- *(!) – логическое отрицание, даёт в результате значение 0(ложь), если операнд отличен от 0 (истина), и, наоборот, значение истина, если операнд был равен 0 (ложь);*
- *тип операнда может быть любой.*

*Пример:*

```
#include <iostream>
using namespace std;
int main()
{ int x=3, y=0;
   bool f=false, v=true;
  cout <<-x<<'\t'<<!y<<endl;
  cout <<-y<<'\t'<<!y<<endl;
  cout <<v<<'\t'<<!v<<endl;
 return 0;}
```

# БИНАРНЫЕ ОПЕРАЦИИ

- *Арифметические операции: умножение(*), деление(/), остаток от деления(%), сложение(+), вычитание(-).*

Рассмотрим операции деления и остаток от деления:

```
#include <iostream>
using namespace std;
int main()                                  // результаты:
{ cout <<100/24<<'\t'<<100./24<<endl;       // 4 4.1666
  cout <<100/21<<'\t'<<100.0/21<<endl;      // 4 4.7619
  cout <<21%3<<'\t'<<21%6<<-21%8<<endl;     // 0 3    -5
  return 0;}
```

- *Операции отношения: (<, <=, >, >=, == !=), меньше, меньше или равно, больше, больше или равно, равно, не равно, не равно соответственно).*

*Результатом операций являются значения true, false.*

# *Логические операции (&& и ||)*

*И (&&) - возвращает значение **истина** тогда и только тогда, когда оба операнда принимают значение истина, в противном случае операция возвращает значение **ложь**.*

*ИЛИ || - возвращает значение **истина** тогда и только тогда, когда хотя бы один операнд принимает значение истина, в противном случае операция возвращает значение **ложь.***

*Логические операции выполняются слева направо;*

*Приоритет операции && выше ||.*

*Пример:*

```
#include <iostream>
using namespace std;
int main()
 {  cout <<“x\t y\t &&\t||” <<endl;
  cout <<“0\t 0\t”<<(0 && 0)<<‘\t’<<(0||0)<<endl;
  cout <<“0\t 1\t”<<(0 && 1)<<\t’<< (0||1)<<endl;
  cout <<“1\t 0\t”<<(1 && 0)<<‘\t’<<(1||0)<<endl;
  cout <<“1\t 1\t”<<(1 && 1)<<‘\t’<<(1||1)<<endl;
  return 0;}
```

# ОПЕРАЦИИ ПРИСВАИВАНИЯ

- формат операции простого присваивания (=):
  операнд_1 = операнд_2

*пример*: a=b=c=100, это выражение выполняется справа налево, результатом выполнения c=100, является число 100, которое затем присвоится переменной b, потом a.

- *Сложные операции присваивания*:
- (*=) – умножение с присваиванием,
- ( /=) – деление с присваиванием,
- (%= ) – остаток от деления с присваиванием,
- (+=) – сложение с присваиванием,
- (-=) – вычитание с присваиванием.

*пример:* к операнду _1 прибавляется операнд_2 и результат записывается в операнд_1

c = c + a, тогда компактная запись c += a

# ТЕРНАРНАЯ ОПЕРАЦИЯ

- ***Условная** операция (**? :**)*
- Формат условной операции: операнд_1 **?** операнд_2 **:** операнд_3
- Операнд_1 - это логическое или арифметическое выражение;
- Оценивается с точки зрения эквивалентности константам true и false;
- Если результат вычисления операнда_1 равен true,то результатом условной операции будет значение операнда_2, иначе операнда_3;
- Тип может различаться;
- Условная операция является сокращённой формой условного оператора if;

Пример:

```
#include <iostream>
  using namespace std;
int main()
  { int x, y, max;
    cin >>x>>y;
    (x>y)? cout <<x : cout<<y<<endl;      //1
    max=(x>y)? x:y;                       //2
    cout<<max<<endl;
    return 0;}
```

Результат:

для x=12 и y=9

12

12

*нахождение наибольшего значения для двух целых чисел;*

# ПРАВИЛА ПРЕОБРАЗОВАНИЯ ТИПОВ

*Примеры выражений:*

(a+0.12)/6;  x && y || !z;

(t*sin(x)-1.05e4)/((2*k+2)*(2*k+3))*4;

- операции выполняются в соответствии с приоритетом;
- если в одном выражении имеются несколько операций одинакового приоритета, то унарные операции выполняются- *справа*
- *налево*, остальные –*слева направо*
- *Т.е : a= b+c значит a=(b+c),*
- *a + b+c значит (a +b) +c*
- *в выражения могут входить операнды различных типов;*
- *при одинаковом типе операндов, результат будет иметь тот же тип;*
- *если разного типа операнды, то операнд с более «низким» типом будет преобразован к более «высокому» типу для сохранения значимости и точности:*

| Тип данных | Старшинство |
|---|---|
| bool | высший |
| double | |
| float | |
| long | |
| Int | |
| short | |
| char | низший |

Неявное преобразование типов:

```
#include <iostream>
using namespace std;
 int main()
 { int a=100, b;   float c=4.5, d;
  d=a/c;       //1- без потери
     точности
  cout << “d=” <<d<<endl;
  b=a/c;       //2- с потерей
     точности
  cout <<“b=”<<b<<endl;
 return 0;}
```

- **Задания:**

1.Составить программу, которая для заданного значения x вычисляет значение выражения $x^2$+sin(x+1)/25 , с учётом приоритета операций в С++:

(pow(x,2)+sin(x+1)/25;

2.Написать программу, подсчитывающую площадь квадрата, периметр которого равен p.

Пусть дан квадрат со стороной *a*, тогда: *p = 4a, a =p/4……*

$s=a^2$ ……S=

# ОПЕРАТОРЫ ЯЗЫКА С++

- Программа на языке С++ состоит из последовательности операторов, большинство из которых приказывают компьютеру совершить какое-либо действие. Окончание оператора обозначается знаком «точка с запятой»;
- Все операторы разделены на 4 группы:

  - операторы следования;
  - операторы ветвления;
  - операторы цикла;
  - операторы передачи управления.

# ОПЕРАТОРЫ СЛЕДОВАНИЯ

- К ним относятся : ***оператор -выражение и составной оператор.***
- ***Выражение,*** завершающееся точкой с запятой, рассматривается как оператор (при вычислении значения выражения или выполнении законченного действия);

  **++i;** *//оператор инкремента*

  **x+=y**; *//оператор сложение с присваиванием*

  **f(a, b)** //вызов функции

  **x= max (a, b) + a*b;** *//вычисление сложного выражения*
- Частным случаем оператора выражения является ***пустой оператор*** *;*

(используется, когда по синтаксису оператор требуется, а по смыслу — нет)
- ***Составной оператор*** или ***блок*** представляет собой последовательность операторов, заключенных в фигурные скобки { }.
- **Блок обладает собственной *областью видимости:*** объявленные внутри блока имена доступны только внутри блока;
- Составные операторы применяются в случае, когда правила языка предусматривают наличие только одного оператора, а логика программы требует выполнения нескольких операторов.

# ОПЕРАТОРЫ ВЕТВЛЕНИЯ

- К ним относятся : *условный оператор* ***if*** *и оператор выбора* ***switch.***

Они позволяют изменить порядок выполнения операторов в программе;

***Условный оператор if***

- используется для разветвления процесса обработки данных на два направления.
- имеет формы: *сокращенную* или *полную.*

- **Формат *сокращенного оператора if:*** **if (B) S;**

  ***B*** –логич. или арифметич. выражение, истинность которого проверяется;
  ***S -*** один оператор: простой или составной.
- При выполнении сокращенной формы оператора if сначала вычисляется выражение *B,* затем проводится анализ его результата: если *B* истинно, то выполняется оператор S; если *B* ложно, то оператор *S* пропускается.
- С помощью сокращенной формы оператора If можно либо выполнить оператор S, либо пропустить его.
- **Формат *полного оператора if:*** **if (B) S1 ; else S2;**

  ***SI, S2-*** один оператор: простой или составной.
- При выполнении полной формы оператора *if* сначала вычисляется выражение *B,* затем анализируется его результат: если *B* истинно, то выполняется оператор S1 а оператор *S2* пропускается; если *B* ложно, то выполняется оператор S2, а *S1* - пропускается.
- С помощью полной формы оператора *if* можно выбрать одно из двух альтернативных действий процесса обработки данных.

# Примеры записи условного оператора *if*.

- if (a > 0)  x=y;  *// сокращенная форма с простым оператором*
- if (++i) {x=y; y=2*z;}  *// сокращенная форма с составным оператором*
- if (a > 0 || b<0) x=y;  else x=z;  *//полная форма с простыми операторами*
- if (i+j-1) { x= 0; y= 1;} else {x=1; y:=0;} *//полная форма с составными операторами*

- Операторы S1 и S2 могут сами являться условными операторами ***if***,  такие операторы называются  *вложенные*;
- Ключевое слово *else* связывается с ближайшим предыдущим словом ***if***, которое ещё не связано ни с одним *else*.

- **Примеры алгоритмов с использованием вложенных условных операторов:**

Пример1  *Уровни вложенности If*

```
if(A<B)
    if (C < D)
        if(E<F) X=Q;
else X = R;
else X=Z;
else X = Y;
```

(3, 2, 1)

Пример 2  *Уровни вложенности if*

```
if (A<B)
   if (C<D) X=Y
   else  X=Z:
    else
    if (E<F) X=R;
    else X=Q;
```

(2, 2, 1)

# Оператор выбора *switch*

*предназначен для разветвления процесса вычислений на несколько направлений.*

- **Формат оператора:**

**switch (<выражение>)**
**{case <константное_выражение_1>: [<оператор 1>;]**
**case <константное_выражение_2>: [<оператор 2>;]**
**……..**
**case <константное_выражение_n>: [<оператор n>;]**
**[default: <оператор> ;]}**

- Выражение, стоящее за ключевым словом *switch,* должно иметь арифметич. тип или тип указатель.

- Все константные выражения должны иметь разные значения, но совпадать с типом выражения, стоящим после *switch.*

- Ключевое слово *case* и расположенное после него константное выражение называют также меткой *case.*

- Выполнение оператора начинается с вычисления выражения, расположенного за ключевым словом *switch*.
- Полученный результат сравнивается с меткой *case*.
- Если результат выражения соответствует метке *case*, то выполняется оператор, стоящий после этой метки.
- Затем последовательно выполняются все операторы до конца оператора *switch* , если только их выполнение не будет прервано с помощью оператора передачи управления *break*
- При использование оператора *break* происходит выход из *switch* и управление переходит к первому после него оператору.
- Если совпадения выражения ни с одной меткой *case* не произошло, то выполняется оператор, стоящий после слова *default*, а при его отсутствии управление передается следующему за *switch* оператору.

- Пример. Известен порядковый номер дня недели. Вывести на экран его название.

```
#include <iostream>
using namespace std;
 int main()
{ int x; cin >>x;
switch (x)
{ case 1: cout <<"понедельник";
     break;
  case 2: cout <<"вторник"; break;
  case 3: cout <<"среда"; break;
  case 4: cout <<"четверг"; break;
  case 5: cout <<"пятница"; break;
   case 6: cout <<"суббота"; break;
   case7: cout <<"воскресенье";
     break;
   default: cout <<“Вы ошиблись!";
}
return 0;}
```

# ОПЕРАТОРЫ ЦИКЛА

- *Операторы цикла используются для организации многократно повторяющихся вычислений. В С++ существуют 3 типа циклов:*

- цикл с предусловием **while**,
- цикл с постусловием **do while**
- цикл с параметром **for**.

***Цикл с предусловием while:***

- Оператор цикла *while* организует выполнение одного оператора (простого или составного) неизвестное заранее число раз.
- ***Формат цикла:*** **while (B) S;**
- ***B*** - выражение, его истинность проверяется(условие выполнения цикла)
- ***S*** - тело цикла: один оператор (простой или составной).
- Перед каждым выполнением тела цикла анализируется значение выражения *B:*

- если оно истинно, то выполняется тело цикла, и управление передается на повторную проверку условия B;
- если значение B ложно - цикл завершается и управление передается на оператор, следующий за оператором S.
- если результат выражения B окажется ложным при первой проверке, то тело цикла не выполнится ни разу
- если условие B во время работы цикла не будет изменяться, то возможна ситуация зацикливания, то есть невозможность выхода из цикла.

# ОПЕРАТОРЫ ЦИКЛА (продолжение)

Внутри тела должны находиться операторы, приводящие к изменению значения выражения *B* так, чтобы цикл мог завершиться.

***Рассмотрим программу вывода на экран целых чисел из интервала от 1 до n.***

```
#include <iostream>
using namespace std;
 int main()
 { int n, i=1;
  cout << "n="; cin >> n;
  while (i<=n)   //пока i меньше или равно n
 { cout << i << '\t';     //выводим на экран значение i
   ++i;}       //увеличиваем i на единицу
 return 0;}
```

*Результаты( при n=10)*

*n=10*

1 2 3 4 5 6 7 8 9 10

- Замечание: используя операцию постфиксного инкремента, тело цикла можно заменить одной командой ***cout <<i++ <<'\t';***

# ЦИКЛ С ПОСТУСЛОВИЕМ DO WHILE

- В отличие от цикла *while* условие завершения этого вида цикла проверяется после выполнения тела цикла.
- **Формат цикла *do while:*** **do S while (B);**

***B*** - выражение, истинность которого проверяется (условие завершения цикла);

**S** - тело цикла: один оператор (простой или блок).

- Сначала выполняется оператор *S,* а затем анализируется значение выражения *B:*

- если оно истинно, то управление передается оператору *S,*
- если ложно - цикл заверш. и управление передается на оператор, следующий за условием *B.*

**Пример(*do while):* программа вывода на экран целых чисел из интервала от 1 до *n.***

```
#include <iostream>
using namespace std;
 int main()
{int n, i=1;
cout <<"n="; cin >>n;
do      //выводим на экран i, а замет увеличиваем
  cout << i++ << '\t';     //ее значении на единицу
whiie (i<=n);  //до тех пор пока i меньше или равна n
return 0;}
```

*Результаты работы программы:*

n=10

1 2 3 4 5 6 7 8 9 10

# ЦИКЛ С ПАРАМЕТРОМ FOR

- Цикл с параметром имеет следующую структуру:
- **for (<инициализация>; <выражение>; <модификации>) <оператор>;**
- ***Инициализация*** *используется для объявления и присвоения начальных значений величинам, используемым в цикле.*
- В этой части можно записать несколько операторов, разделенных запятой. Областью действия переменных, объявленных в части инициализации цикла, является цикл и вложенные блоки.
- ***Выражение*** *определяет условие выполнения цикла:*

- *если его результат истинен, цикл выполняется.*

- *Истинность* выражения проверяется перед каждым выполнением тела цикла, таким образом, цикл с параметром реализован как цикл с предусловием.
- ***Модификации*** *выполняются после каждой итерации цикла и служат обычно для изменения параметра цикла.*
- В части модификаций можно записать несколько операторов через запятую.
- ***Оператор*** (простой или составной) представляет собой тело цикла.

# ЦИКЛ С ПАРАМЕТРОМ FOR (продолжение)

- Любая из частей оператора *for* (инициализация, выражение, модификация, оператор) может отсутствовать, но точку с запятой, определяющую позицию пропускаемой части, надо оставить.

```
#include <iostream>
using namespace std;
int main()
{ int n;
cout <<"n="; cin >>n;
  for (int i=1; i<=n; i++)  //для i от 1 до n с шагом 1
cout << i <<"\t";     //выводить на экран значение i
return 0;}
```

***Результаты работы программы:***

n=10

1 2 3 4 5 6 7 8 9 10

- *Замечание*. Используя операцию постфиксного инкремента при выводе данных на экран, цикл *for* можно преобразовать следующим образом:
- ***for (int i=1 ;i<=n;) cout<<i+ + <<"\t";***
- В этом случае в заголовке цикла *for* отсутствует блок модификации.