Конструирование программ и языки программирования

Всего 94 часа
54 – лекции;
40 – лабораторные

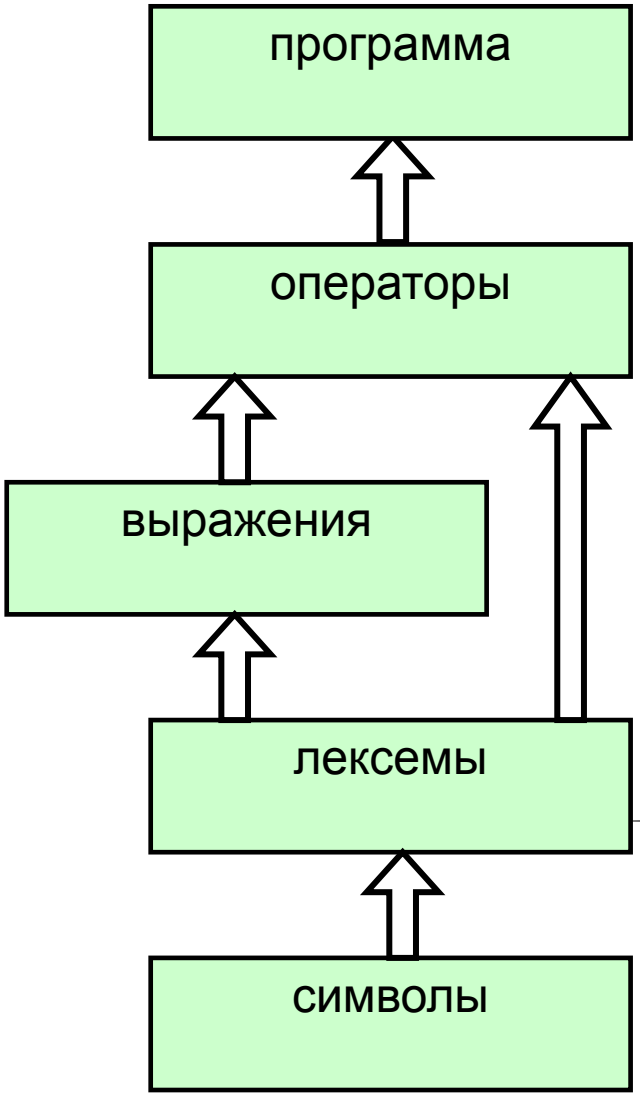
Программирование на C/C++

Базовые средства C/C++

Оглавление

- Типы данных C++
- Структура программы
- Переменные и выражения
- Базовые конструкции структурного программирования (операторы ветвления, цикла и т.д.)

Состав языка



a=b; for (int i=0;i<n;++i)

a++ - --b/c

- идентификаторы
- ключевые слова
- константы
- знаки операций
- разделители

a-z, A-Z, 0-9, “, {,},|,/,%,…

примеры

Пример структуры программы

```
директивы препроцессора
описания
int main() {
    операторы главной функции
}
int f1() {
    операторы функции f1
}
int f2() {
    операторы функции f2
}
```

Константы

Вид	**Примеры**
Целые дес.	8 0 199226
Восьм.	01 020 07155
Шестн.	0xA 0x1B8 0X00FF
Веществ.	5.7 .001 35.
Вещ. с плав. т.	0.2E6 .11e–3 5E10
Символьные	'A‘ 'ю‘ '*‘ 'db‘ '\0' '\n‘ '\012‘ '\x07\x07'
Строковые	"Здесь был Vasia" " \t3начение r=\0xF5\n"

Управляющие последовательности

\a	**7**	**Звуковой сигнал**
\b	8	Возврат на шаг
\f	C	Перевод страницы (формата)
\n	**A**	**Перевод строки**
\r	D	Возврат каретки
\t	**9**	**Горизонтальная табуляция**
\v	**B**	**Вертикальная табуляция**
\\	5C	Обратная косая черта
\'	27	Апостроф
\"	22	Кавычка
\?	3F	Вопросительный знак
\0ddd		Восьмеричный код символа
\0xdd	dd	Шестнадцатиричный код символа

Типы данных

Тип данных определяет:

- *внутреннее представление* данных в памяти компьютера => *множество значений*, которые могут принимать величины этого типа;
- *операции и функции*, которые можно применять к величинам этого типа.

Типы в C++ делятся на *основные* (fundamental) и *составные* (compound). Тип может описывать объект, ссылку или функцию.

Основные (стандартные) типы данных:

int (целый); intergal

char (символьный);

wchar_t (расширенный символьный);

bool (логический);

float (вещественный);

double (вещественный с двойной точностью).

Спецификаторы:

short (короткий);

long (длинный);

signed (знаковый);

unsigned (беззнаковый).

+ void

Составные типы

- *arrays* of objects of a given type;
- *functions*, which have parameters of given types and return void or references or objects of a given type;
- *pointers* to void or objects or functions of a given type;
- *references* to objects or functions of a given type;
- *classes* containing a sequence of objects of various types, a set of types, enumerations and functions for manipulating these objects, and a set of restrictions on the access to these entities;
- *unions*, which are classes capable of containing objects of different types at different times;
- *enumerations*, which comprise a set of named constant values. Each distinct enumeration constitutes a different *enumerated type*;
- *pointers to non-staticclass members*

Диапазоны для IBM PC-совместимых

Тип	Диапазон значений	Размер(байт)
bool	true и false	1
signed char	-128 … 127	1
unsigned char	0 … 255	1
signed short int	-32 768 … 32 767	2
unsigned short *int*	0 … 65 535	2
signed long int	-2 147 483 648 … 2 147 483 647	4
unsigned long int	0 … 4 294 967 295	4
float	3.4e-38 … 3.4e+38	4
double	1.7e-308 … 1.7e+308	8
long double	3.4e-4932 … 3.4e+4932	10

Диапазоны типов по стандарту

- sizeof(float) ≤ sizeof(double) ≤ sizeof(long double)
- sizeof(char) ≤ sizeof(short) ≤ sizeof(int) ≤ sizeof(long)

- Минимальные и максимальные значения определены в файлах:
  - <climits> - целые
  - <cfloat> - вещественные

Явное задание типа констант

- 0X22UL
- 05Lu
- 2E+6L
- 1.82f
- L”Vasia”

Комментарии

- Однострочные

  //…..
- Многострочные

  /* ……..

  …

  */

Описание идентификаторов

[класс памяти] [const] тип имя [инициализатор];

инициализатор: **= значение** или **(значение)**

Примеры описаний:

```
short int a = 1;
const char C = 'C';
char s, sf = 'f';
char t (’54’);
float c = 0.22, x(3), sum;
```

Область видимости

Каждый идентификатор имеет область действия (potential scope) и область видимости (scope), которые, как правило, совпадают (кроме случая описания такого же имени во вложенном блоке).

- Область видимости начинается в точке описания.

```
const int i = 2; { int i[i]; }
```

Имя, описанное внутри блока, локально по отношению к этому блоку. Имя, описанное вне любого блока, имеет глобальную область видимости.

Область действия и класс памяти зависят не только от собственно описания, но и от места его размещения в тексте программы.

Класс памяти

auto — *автоматическая* переменная. Память выделяется в стеке и при необходимости инициализируется каждый раз при выполнении оператора, содержащего ее определение. Освобождение памяти - при выходе из блока

extern — переменная определяется в другом месте программы.

static — *статическая* переменная. Время жизни — постоянное. Инициализируется один раз при первом выполнении оператора, содержащего определение переменной. В зависимости от расположения оператора описания статические переменные могут быть *глобальными и локальными*.

register — аналогично auto, но память выделяется по возможности в регистрах процессора.

Область видимости. Пример 1

```
int a;     // глобальная переменная a
main(){
   int b;// локальная переменная b
   extern int x;  // переменная x определена в другом месте
   static int c;  // локальная статическая переменная c
   a = 1;// присваивание глобальной переменной
   int a;//локальная переменная a
   a = 2;// присваивание локальной переменной
   ::a = 3;  // присваивание глобальной переменной
}
int x = 4;    // определение и инициализация x
```

Область видимости. Пример 2

```
int a;              // глобальная переменная
int main(){
    int b;          // локальная переменная
    static int c = 1;  // локальная статическая переменная
}
```

	Глобальная	Локальная	Статическая
Размещение	с-т данных	с-т стека	с-т данных
Время жизни	вся прогр.	блок	вся прогр.
Область видимости	файл	блок	блок
Обнуление	да	нет	да

Области действия

- блок
- файл
- функция
- прототип функции
- класс
- поименованная область

Базовые конструкции языка C/C++

Переменные

- могут быть **глобальными** (объявляются вне функций), **локальными** (объявляются внутри функций), **формальными параметрами** (объявляются при описании параметров функции).
- Если при объявлении переменных начальное значение не задано, то глобальные переменные инициализируются нулем; локальные переменные имеют неопределенное значение.

Правила задания имени переменной (идентификатора)

- Начинается с буквы или знака _ ;
- Может содержать буквы латинского алфавита, цифры, знак _;
- Строчные и прописные буквы различаются;
- Переменные могут быть описаны в любом месте программы до их использования;
- Имена переменных в операторах описания отделяются запятыми; int i,k,l;
- Возможна инициализация переменных при описании; int i = 256,k,l;

Операторы и операции
Операция присваивания

- **имя_переменной =выражение;**

i = j + k;

многократное присваивание (справа налево)

i = j = k = 0; **a=b=1=k=0;**

или так:

i = 2 + (k = 3);

Операция присваивания

- Сначала вычисляется выражение, а затем результат присваивается имени переменной.
- *Например*: y=(x+2)/(3*x)-5;
- С помощью одного оператора можно присвоить одно значение нескольким переменным, например:
- **x=y=z=0;** /* x, y, z=0 */
- или **z=(x=y)*5**;
- - сначала переменной **x** присваивается значение переменной **y**, далее вычисляется выражение **x*5** , и результат присваивается переменной **z**.

Сокращенная форма

имя_переменной операция=выражение;

- где **операция** – одна из арифметических операций (**+ , -, *, /, %**);

Например:

```
x*=5;      /* x=x*5;*/
s+=7;      /* s=s+7;*/
y/=x+3;      /* y=y/(x+3); */
```

- Сокращенная форма операции присваивания применяется тогда, когда переменная используется в обеих частях полной формы данного оператора.

В языке C++ существует операции

- **Уменьшения (декремент)** (**--**) и **увеличения (инкремент)** (**++**) значения переменной на 1. Операции могут быть **префиксные** (++i и --i) и **постфиксные** (i++ и i--).
- При использовании данной операции в выражении, в случае префиксной операции сначала выполняется сама операция (изменяется значение i), и только потом вычисляется выражение. В случае постфиксной операции – операция применяется после вычисления выражения.

Например:

```
n=1; b=7;
c=b*++n;      /* n=n+1; c=b*n; т.е. c=14 */
c=b*n++;      /* c=b*n; n=n+1; т.е. c=7 */
```

Операции ++ и --, комбинированные операции

```
i = 0;
j = ++i   // j = 1,  i = 1
k = i--   // k =  l, i = 0

i +=j;
i *=j;
i %=j;
```

Арифметические операции

* — умножение

/ — деление

% — остаток от деления (для целых, корректно работает для положительных чисел)

\+ — сложение

\- — вычитание

Приведение типов

```
double avg, sum;
int n;
avg = sum/n; //приведение к double
double num = n;
int a = 3,   b = 2;
double  r = a/b; //приведение к int
```

Операции отношения

>	больше (a > b)
>=	больше или равно (a >= b)
<	меньше (i < 0)
<=	меньше или равно (i <= j)
==	равно (i == к)
!=	не равно (ch != 'y')

Логические операторы

&&	и	(i>j)&&(k!=1)
||	или	(ch==‘y’) || (ch == ‘Y’)
!	не	!(i>1)

Стандартные математические функции

Математическая функция	Имя функции в языке С	Математическая функция	Имя функции в языке С
корень(x)	**sqrt(x)**	**arcsin(x)**	**asin(x)**
|x|	**fabs(x)**	**arccos(x)**	**acos(x)**
e^x	**exp(x)**	**arctg(x)**	**atan(x)**
x^y	**pow(x,y)**	**arctg(x/y)**	**atan2(x,y)**
ln(x)	**log(x)**	**sh(x)=1/2 (e^x-e^{-x})**	**sinh(x)**
lg_{10}(x)	**log10(x)**	**ch(x)=1/2 (e^x+e^{-x})**	**cosh(x)**
sin(x)	**sin(x)**	**tgh(x)**	**tanh(x)**
cos(x)	**cos(x)**	**Ост. от деления x на y**	**fmod(x,y)**
tg(x)	**tan(x)**	**Наим. целое, которое >=x**	**ceil(x)**
		Наиб. целое, которое <=x	**floor(x)**

Библиотечные функции

Синтаксис использования функции в программе:

```
the_root = sqrt(9.0);
```

Обращение к функции

Фактический параметр

Вызов функции в `cout`-инструкциях:

```
cout<<"Длина стороны квадрата, площадь
которого"<<area<<", равна"<< (sqrt(area));
```

Файлы библиотечных функций (директивы препроцессора)

#include <stdio.h> - подключение файла с объявлением стандартных функций файлового ввода-вывода;

#include <conio.h> - функции работы с консолью;

#include <graphics.h> - графические функции;

#include <math.h> - математические функции.

#include<iostream.h> - подключение библиотеки потокового ввода-вывода

Функции вывода информации

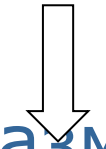
putchar() - обеспечивает вывод одиночного символа без перехода на новую строку.

puts() - используется для вывода строки символов с переходом на начало новой строки.

printf() - форматированный вывод данных.

Формат:

printf (<управляющая строка>, <спис. арг.>);



% <флаг><размер поля . точность>
спецификация

Форматы функции печати (спецификация)

Формат	Тип выводимой информации
d	десятичное целое число
c	один символ
s	строка символов
e	число с плавающей точкой (экспоненциальная запись) 1.2E+21
f	число с плавающей точкой (десятичная запись)
u	десятичное число без знака
o	восьмеричное число без знака
x	шестнадцатеричное число без знака

Примеры форматированного вывода

```
int num=5, cost=11000, s=-777;
float bat=255, x=12.345;
printf ("на %d студентов %f бутербродов\n", num,
   bat);
printf ("Значение числа pi равно%f.\n", PI);
printf ("Любовь и голод правят миром.\n");
printf ("Стоимость этой вещи %d%s.\n", cost,"
   Руб.");
printf ("x=%-8.4f s=%5d%8.2f ", x, s, x);
x=12.3450   s= -777     12.34
```

- Выравнивание по левому краю

8 позиций на целую часть 4 позиции на дробную

Функции ввода информации

getch () ввод одиночных символов.

gets () ввод строки символов до нажатия клавиши ENTER.

scanf форматированный ввод информации любого вида.

Формат:

scanf (<управляющая строка>, <список адресов>);

Примеры форматированного ввода

```
int course; // название переменных
float grant;
char name[20]; // строка символов
printf ( "Укажите ваш курс, стипендию,
   имя"); //может просто быть написана строка символов в
   кавычках
scanf  ( "%d%f", &course, &grant);
scanf  ( "%s", name); //адрес у строк не
   пишется (без амперсанда)
```

Первая программа

```
#include <stdio.h>

void main()

{
  printf (“Hello, world!\n");

}
```

- Включение информации о стандартной библиотеке.
- Определение функции с именем **main**, не получающей никаких аргументов.
- Инструкции **main** заключаются в фигурные скобки.
- Функция **main** вызывает библиотечную функцию **printf** для печати заданной последовательности символов
- **\n** — символ новой строки

Первая программа

```
#include <stdio.h>
int main()
{
  printf("Hello, world!");
  return 0;
}
```

```
#include <stdio.h>
void main()
{
  printf("Hello, world!");
}
```

Пример 1 - простейшая программа

```
#include <stdio.h>
int main(){
   int i;
   printf("Введите целое число\n");
   scanf("%d", &i);
   printf("Вы ввели число %d, спасибо!", i);
}
```

```
#include <cstdio>
using namespace std;
int main(){
   int i;
   printf("Введите целое число\n");
   scanf("%d", &i);
   printf("Вы ввели число %d, спасибо!", i);
}
```

Пример 2 - целые форматы

```
#include <stdio.h>
int main(){
    int int1 = 45, int2 = 13;
    printf("int1 = %d| int2 = %3d| int2 = %-4d|\n",
        int1, int2, int2);
    printf("int1 = %X| int2 = %3x| int2 = %4o|\n",
        int1, int2, int2);
}
```

```
int1 = 45| int2 =  13| int2 = 13  |
int1 = 2D| int2 =   d| int2 =   15|
```

Пример 3 - вещественные форматы

```
#include <stdio.h>
int main(){
float f = 3.621;
double dbl = 2.23;
printf("f = %f| f = %4.2f| f = %6.1f|\n", f, f, f);
printf("f = %g| f = %e| f = %+E|\n", f, f, f);
printf("dbl = %5.2lf| dbl = %e| dbl = %4.1G|\n",
      dbl, dbl, dbl);
}
```

```
f = 3.621000| f = 3.62| f =    3.6|
f = 3.621| f = 3.621000e+000| f = +3.621000E+000|
dbl =  2.23| dbl = 2.230000e+000| dbl =    2|
```

Пример 4 - форматы символов и строк

```
#include <stdio.h>
int main(){
char ch = 'z', *str = "ramambahari";
printf("ch = %c| ch = %3c|\n", ch, ch);
printf("str = %14s|\nstr = %-14s|\nstr = %s|\n",
      str, str, str);
}
```

```
ch = z| ch =   z|
str =    ramambahari|
str = ramambahari   |
str = ramambahari|
```

Пример 5 - классы ввода-вывода

```
#include <iostream.h>
int main(){
   int i;
   cout << "Введите целое число\n";
   cin >> i;
   cout << "Вы ввели число” << i << ", спасибо!";
}
```

```
#include <iostream>
using namespace std;
int main(){
   int i;
   cout << "Введите целое число\n";
   cin >> i;
   cout << "Вы ввели число” << i << ", спасибо!";
}
```

Операции C++ (не все!)

Унарные операции

++ – – sizeof ~ ! – + & * new delete (type)

Бинарные операции

* / % + – << >> < <=
> >= == != & ^ | && || = *= /= %=+= –= <<=
>>= &= |= ^= throw ,

Тернарная операция

? :

Приоритеты операций

Операция	Краткое описание
Унарные операции	
::	доступ к области видимости
.	выбор
->	выбор
[ ]	индексация
( )	вызов функции
<тип>( )	конструирование
++	постфиксный инкремент
– –	постфиксный декремент
typeid	идентификация типа
dynamic_cast	преобразование типа с проверкой на этапе выполнения
static_cast	преобразование типа с проверкой на этапе компиляции
reinterpret_cast	преобразование типа без проверки
const_cast	константное преобразование типа

Приоритеты операций

sizeof	размер объекта или типа
– –	префиксный декремент
++	префиксный инкремент
~	поразрядное отрицание
!	логическое отрицание
–	арифметическое отрицание (унарный минус)
+	унарный плюс
&	взятие адреса
*	разадресация
new	выделение памяти
delete	освобожение памяти
(<тип>)	преобразование типа

Приоритеты операций

.*	выбор
->*	выбор
Бинарные и тернарная операции	
*	умножение
/	деление
%	остаток от деления
+	сложение
–	вычитание
<<	сдвиг влево
>>	сдвиг вправо

<	меньше
<=	меньше или равно
>	больше
>=	больше или равно
==	равно
!=	не равно
&	поразрядная конъюнкция (И)
^	поразрядное исключающее ИЛИ
|	поразрядная дизъюнкция (ИЛИ)
&&	логическое И
||	логическое ИЛИ
? :	условная операция (тернарная)
=	присваивание
*=	умножение с присваиванием
/=	деление с присваиванием

%=	остаток отделения с присваиванием
+=	сложение с присваиванием
–=	вычитание с присваиванием
<<=	сдвиг влево с присваиванием
>>=	сдвиг вправо с присваиванием
&=	поразрядное И с присваиванием
|=	поразрядное ИЛИ с присваиванием
^=	поразрядное исключающее ИЛИ с присваиванием
throw	исключение
,	последовательное вычисление

Операции выполняются в соответствии с *приоритетами*. Для изменения порядка выполнения операций используются круглые скобки. Если в одном выражении записано несколько операций одинакового приоритета, унарные операции, условная операция и операции присваивания выполняются *справа налево*, остальные — *слева направо.*

Операции инкремента и декремента

```
#include <stdio.h>
int main(){
int x = 3, y = 3;
printf("Значение префиксного выражения: %d\n", ++x);
printf("Значение постфиксного выражения: %d\n", y++);
}
```

Результат работы программы:

```
Значение префиксного выражения: 4
Значение постфиксного выражения: 3
```

Операция sizeof

sizeof выражение **sizeof (тип)**

```
#include <iostream.h>
  int main(){
  float x = 1;
  cout << "sizeof (float) :" << sizeof (float);
  cout << "\nsizeof x :" << sizeof x;
  cout << "\nsizeof (x+1.0) :" << sizeof (x+1.0);
  }
```

```
sizeof (float) : 4
sizeof x : 4
sizeof (x+1.0) : 8
```

Поразрядные операции

```
#include <iostream.h>
int main(){
   cout << "\n 6&5 = " << (6&5);
   cout << "\n 6|5 = " << (6|5);
   cout << "\n 6^5 = " << (6^5);
}
```

Результат работы программы:

```
6&5 = 4
6|5 = 7
6^5 = 3
```

Операции деления и остатка от деления

```
#include <stdio.h>
int main(){
    int x = 11, y = 4;
    float z = 4;
    printf(" %d  %f\n", x/y, x/z);
    printf("Остаток: %d\n", x%y);
}
```

```
2  2.750000
Остаток: 3
```

Тернарная операция:

```
i = (i < n) ? i + 1: 1
```

Сложное присваивание:

```
a += b
```

Примеры выражений:

```
(a + 0.12)/6
x && y || !z
(t * sin(x)-1.05e4)/((2 * k + 2) * (2 * k + 3))
```

Приоритеты:

a = b = c означает a = (b = c)

a + b + c означает (a + b) + c

(sin(x + 2) + cos(y + 1))

Преобразования типов

- изменяющие внутреннее представление величин (с потерей точности или без потери точности);
- изменяющие только интерпретацию внутреннего представления.

Явные преобразования типа:

- const_cast
- dynamic_cast
- reinterpret_cast
- static_cast
- приведение в стиле C: (имя_типа)выражение

Правила преобразования типов

Операнды char, unsigned char или short преобразуются к int по правилам:

- char расширяется нулем или знаком в зависимости от умолчания для char;
- unsigned char расширяется нулем; signed char расширяется знаком;
- short, unsigned short и enum при преобразовании не изменяются.

Затем любые два операнда становятся int, или float, double или long double.

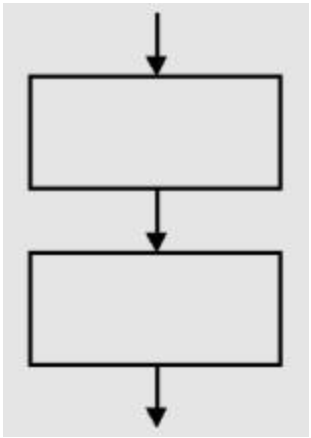
- Если один из операндов имеет тип long double, то другой преобразуется к типу long double.
- Если один из операндов double, другой преобразуется к double.
- Если один из операндов float, другой преобразуется к float.
- Иначе, если один из операндов unsigned long, другой преобразуется к unsigned long.
- Иначе, если один из операндов long, то другой преобразуется к long.
- Иначе, если один из операндов unsigned, другой преобразуется к unsigned.
- Иначе оба операнда должны иметь тип int.

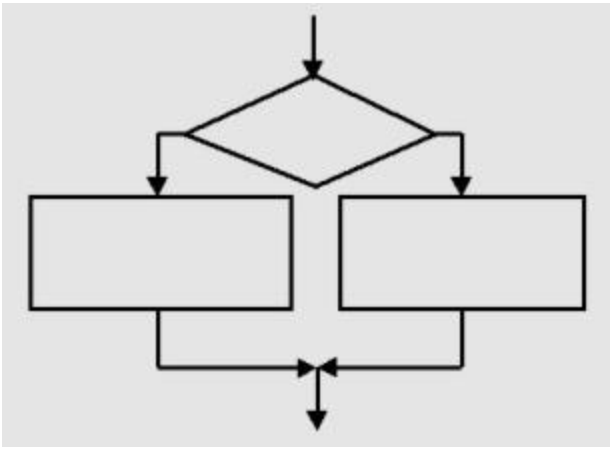
Тип результата тот же, что и тип участвующих в выражении операндов.

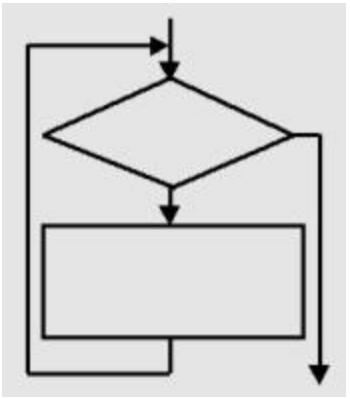
Оператор «выражение»

```
;

i++;

fun(i, k);

a *= b + c;
```

Базовые конструкции структурного программирования







Условный оператор

if (выражение) оператор_1; [else оператор_2;]

```
   if (a<0) b = 1;                  // 1
   if (a<b && (a>d || a==0)) b++;
   else {b* = a; a = 0;}            // 2
   if (a<b){
      if (a<c) m = a;
      else m = c;}
   else {if (b<c) m = b;
         else m = c;}               // 3
   if (a++) b++;                    // 4
if (b>a) max = b;
else max = a;          // max = (b > a) ? b : a;
```

```
if (int i = fun(t)) a -= i; else a += i;
```

Например:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int age;                                    // без переменной никак...

    printf( "Сколько вам лет? " );              // спрашиваем пользователя о его возрасте
    scanf( "%d", &age );                        // ввод пользователем количества лет
    if ( age < 100 ) {                          // если введенный возраст меньше 100
        printf ("Вы очень молоды!/n" );         // просто показываем что программа сработала верно...
    }
    else if ( age == 100 ) {                    // используем else для примера
        printf( "Молодость уже позади/n" );
    }
    else {
        printf( "Столько не живут/n" );         // если ни одно из выше-перечисленных условий не подошло, то программа покажет этот вариант ответа
    }
  return 0;
}
```

Оператор **switch**

```
switch ( выражение ){
   case константное_выражение_1:
        [список_операторов_1]
   case константное_выражение_2:
        [список_операторов_2]
   ...
   case константное_выражение_n:
        [список_операторов_n]
   [default: операторы ]
}
```

Пример оператора switch

```
#include <iostream.h>
int main(){
   int a, b, res; char op; bool f = true;
   cout << "\nВведите 1й операнд : ";      cin >> a;
   cout << "\nВведите знак операции : ";  cin >> op;
   cout << "\nВведите 2й операнд : ";      cin >> b;
   switch (op){
     case '+': res = a + b; break;
     case '-': res = a - b; break;
     case '*': res = a * b; break;
     case '/': res = a / b; break;
     default : cout <<"\nНеизвестная операция"; f = false;
   }
   if (f) cout << "\nРезультат : " << res;
}
```

Оператор цикла **while**

while (выражение) оператор;

```
#include <stdio.h>
int main(){
  float Xn, Xk, Dx;
  printf("Введите диапазон и шаг изм-я аргумента: ");
  scanf("%f%f%f", &Xn, &Xk, &Dx);
  printf("|   X   |   Y   |\n");
  float X = Xn;
  while (X <= Xk){
     printf("| %5.2f | %5.2f |\n", X, X*X + 1);      X
+= Dx;
  }
}
```

while (int x = 0) { /* область действия x */ }

Оператор цикла **do while**

do оператор **while** выражение;

```
#include <iostream.h>
int main(){
    char answer;
    do{
        cout << "\nКупи слоника! ";
        cin >> answer;
    }while (answer != 'y');
}
```

Пример 6 – вычисление функции

$$y_n = \frac{1}{2}\left(y_{n-1} + x/y_{n-1}\right)$$

```
#include <stdio.h>
#include <math.h>
int main(){
   double X, Eps;
   double Yp, Y = 1;
   printf("Введите аргумент и точность: ");
   scanf("%lf%lf", &X, &Eps);
   do{
      Yp = Y;
      Y = (Yp + X/Yp)/2;
   }while (fabs(Y - Yp) >= Eps);
   printf("\n %lf %lf", X, Y);
}
```

Оператор цикла **for**

for (инициализация; выражение; модификации) оператор;

```
for (int i = 1, s = 0; i<=100; i++) s += i;
```

```
#include <iostream.h>
int main(){
    int num;
    cout << "\nВведите число : "; cin >> num;
    for (int half = num / 2, div = 2; div <= half; div++)
     if (!(num % div))cout << div <<"\n";
}
```

- for (int i = 1, s = 0; i<=100; i++) s += i;

- int i = 1, s = 0;
- for(; i<=100;)
- {s += i;
- i++;}

Операторы передачи управления

- оператор безусловного перехода goto;
- оператор выхода из цикла break;
- оператор перехода к следующей итерации цикла continue;
- оператор возврата из функции return.

Оператор goto

- Оператор безусловного перехода goto имеет формат:
- goto метка;
- В теле той же функции должна присутствовать ровно одна конструкция вида:
- метка: оператор;
- Оператор goto передает управление на помеченный оператор. Метка — это обычный идентификатор, областью видимости которого является функция, в теле которой он задан.

Использование оператора безусловного перехода оправдано в двух случаях:

1. принудительный выход вниз по тексту программы из нескольких вложенных циклов или переключателей;
2. переход из нескольких мест функции в одно (например, если перед выходом из функции всегда необходимо выполнять какие-либо действия).

В остальных случаях

- для записи любого алгоритма существуют более подходящие средства, а использование goto приводит только к усложнению структуры программы и затруднению отладки (даже в приведенных случаях допустимо применять goto только в случае, если в этих фрагментах кода не создаются локальные объекты. В противном случае возможно применение деструктора при пропущенном конструкторе, что приводит к серьезным ошибкам в программе).

Пример

- int k; ...
- goto metka; ...
- {int a = 3, b = 4;
- k = a + b;
- metka: int m = k + 1; ...
- }
- После выполнения этого фрагмента программы значение переменной m не определено.

Оператор break

- используется внутри операторов цикла или switch для обеспечения перехода в точку программы, находящуюся непосредственно за оператором, внутри которого находится break .

Оператор continue

- Оператор перехода к следующей итерации цикла continue пропускает все операторы, оставшиеся до конца тела цикла, и передает управление на начало следующей итерации.

Оператор return

- Оператор возврата из функции return завершает выполнение функции и передает управление в точку ее вызова.
- Вид оператора:
- return [выражение];
- Выражение должно иметь скалярный тип. Если тип возвращаемого функцией значения описан как void , выражение должно отсутствовать.

Пример 7 - Вычисление суммы ряда

$$\mathrm{sh}\ x = 1 + x^3/3! + x^5/5! + x^7/7! + \ldots$$

```
#include <iostream.h>
#include <math.h>
int main(){
   const int MaxIter = 500;
   double x, eps;
   cout << "\nВведите аргумент и точность: ";
   cin >> x >> eps;
   bool ok = true;
   double y = x, ch = x;
   for (int n = 0; fabs(ch) > eps; n++){
      ch *= x * x /(2 * n + 2)/(2 * n + 3);
      y += ch;
      if (n > MaxIter){ok = false; break;}
   }
   if (ok) cout << "\nЗначение функции: " << y;
   else    cout << "\nРяд расходится!";
}
```

Пространства имен

В каждой области действия различают пространства имен, в пределах которых идентификатор должен быть уникальным. В разных категориях имена могут совпадать, например:

```
struct Node{
    int Node;
    int i;
}Node;
```

В C++ определено четыре раздельных класса идентификаторов, в пределах которых имя должно быть уникальным:

1. имена переменных, функций, типов typedef и констант перечислений;
2. имена типов перечислений, структур, классов и объединений;
3. элементы каждой структуры, класса и объединения;
4. метки.