

Основы программирования на языке C++

Преподаватель:
Маслов Алексей Владимирович,
доцент кафедры общей физики

Содержание

1. Типы данных, операторы, выражения [см. лек. 1]
2. Управление (инструкции, блоки, циклы, переключатели)
3. Массивы, структуры, указатели
4. Функции и структура программы, шаблоны функций
5. Интерфейс (ввод-вывод)
6. Работа с памятью, пространства имен
7. Классы и объекты
8.

Управление

(инструкции, блоки, циклы, переключатели)

Цикл **for** (для)

Общий вид: **for** (инициализация; проверка; обновление) {
 тело цикла;
}

Примеры: **for** (**int** i = 0; i < 10; i++)
 some_function(i); // одна инструкция не требует {}

```
int i;  
for (i = 0; i < 10; i++) { // начало блока  
    some_function1(i);  
    some_function2(i);  
} // конец блока
```

```
for (; ; ) { // пустой цикл, так тоже работает  
    ...;  
    if (...) break; // нужен способ выхода  
}
```

Операции инкремента ++ и декремента --

Префиксная операция указывается перед операндом: ++x, --x.

Постфиксная операция следует после операнда: x++, x--.

```
int x = 5;
```

```
int y = ++x; // изменить x, затем присвоить его y;  y равно 6, x равно 6
```

```
int x = 5;
```

```
int y = x++; // присвоить y, затем изменить x;  y равно 5, x равно 6
```

x++ означает использование существующего значения x при выполнении операции, а затем увеличение значения на 1.

++x означает увеличение на 1, а затем использование нового значения x при выполнении операции.

Поведение может быть неопределенно стандартом:

```
int x = 1;
```

```
int y = (2 + x++ ) * (1 + x); // x=2, y=3*2 или y=3*3? C++ не гарантирует,  
// что x++ выполняется после (1+x)
```

```
int y = (1 + x) * (2 + x++ ); // а если так?
```

Если использовать опцию компилятора -Wall, то будет предупреждение
warning: operation on 'x' may be undefined.

Составные инструкции, или блоки

```
for (int i = 1; i <= 5; i++ ) {           // начало блока
    cout << "Value " << i << " : "; // ввод числа
    cin >> number;
    sum += number;
} // конец блока
```

```
int x = 20;
{ // начало блока
    int y = 100; // y определено только внутри блока!
    cout << x << endl;
    cout << y << endl; // нормально
} // конец блока
cout << x << endl;
cout << y << endl; // ошибка во время компиляции!
```

Запятая как оператор

Операция запятой (,) позволяет вставлять два выражения туда, где синтаксис C++ допускает только одно.

// записывает строку в обратном порядке

string word; // строка

.....

char temp;

int i, j; // запятая просто разделяет переменные в списке

for (j = 0, i = word.size() - 1; j < i; --i, ++j) { // начало блока

temp = word[i];

word[i] = word[j];

word[j] = temp;

} // конец блока

int j = 0, i = word.size() - 1; // можно и так

--i, ++j; // два выражения считаются одним синтаксически

Выражения отношений

< Меньше чем

<= Меньше или равно

== Равно // не путать с присвоением =

> Больше чем

>= Больше или равно

!= Не равно

```
int quizscores[10] = { 20, 20, 20, 20, 20, 19, 20, 18, 20, 20};
```

```
for (i = 0; quizscores [i] == 20; i++)
```

```
    cout << "quiz " << i << " is a 20\n";
```

```
for (i = 0; quizscores [i] = 20; i++) // Segmentation fault
```

```
    cout << "quiz " << i << " is a 20\n" ;
```


Цикл **while** (пока)

```
while (проверочное условие){  
    ..... тело цикла;  
}
```

```
// Вывод имени посимвольно и в кодах ASCII
```

```
char name[ArSize];
```

```
cout << "Your first name, please: ";
```

```
cin >> name;           // ввод имени
```

```
cout << "Here are the ASCII codes for the letters:\n";
```

```
int i = 0;              // начать с начала строки
```

```
while (name[i] != '\0'){ // обрабатывать до конца строки
```

```
    cout << name[i] << " : " << int (name [i] ) << endl;
```

```
    i++;                // не забудьте этот шаг
```

```
}
```

Цикл **do while** (делать до тех пор, пока)

do {

..... тело цикла;

} **while** (проверочное выражение) ;

do {

cin >> n; // выполнить тело

} **while** (n != 7); // затем проверить

//завершает работу при n равно 7

Цикл **for**, основанный на диапазоне (C++11)

```
double prices[5] = {4.99, 10.99, 6.87, 7.99, 8.49};  
for (double x : prices) // синтаксис с C++11 до C++20  
    cout << x << endl; // цикл отображает все значения,  
                        // включенные в диапазон массива.
```

или

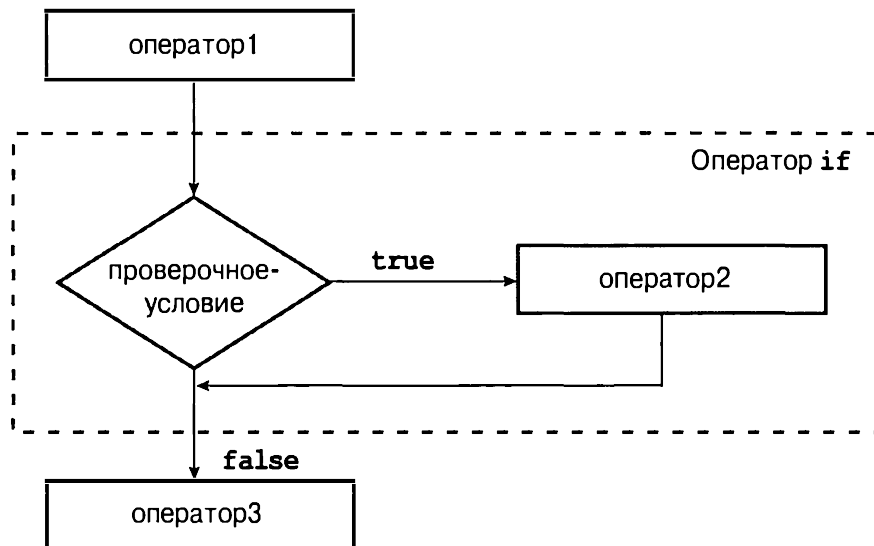
```
string str = "Hello";  
for (char c : str)  
    cout << c << ' '; // цикл отображает все символы
```

(Не работает с динамическими массивами)

Операторы ветвления и логические операции

Операторы **if**, **if else**

```
оператор1;  
if (проверочное условие)  
    оператор2;  
    оператор3;
```



```
if (проверочное условие){  
    оператор2;  
} else {  
    оператор3;  
}  
оператор4
```

```
if (пров. условие 1){  
    оператор2;  
} else if (пров. условие 2){  
    оператор3;  
} else {  
    оператор4;  
}  
оператор5
```

Пример использования оператора **if**

```
char ch;
```

```
.....
```

```
if (ch == 'Z') { // блок, выполняемый, если условие истинно  
    counter++;  
    cout << "is true\n";  
} else { // блок, выполняемый, если условие ложно  
    cout << "is false\n";  
}
```

Логические выражения

Три логических операции, с помощью которых можно комбинировать или модифицировать выражения:

Логическая операция	Запись (символы или ключевые слова)		
ИЛИ (OR)		или	or
И (AND)	&&	или	and
НЕ (NOT)	!	или	not

- 1) Логическая операция И имеет более высокий приоритет, чем логическая операция ИЛИ
- 2) Логические операции ИЛИ, И имеют более низкий приоритет, чем операции сравнения.
- 3) Операция НЕ имеет более высокий приоритет, чем операции сравнения.

((age > 50 || weight > 300) && donation > 1000)

(x != 0 && 1.0 / x > 100.0)

Примеры логических операций:

```
5==5 || 5==9    // true
5>3  || 5>10    // true
5>8  || 5<10    // true
5>8  || 5<2     // false
```

```
5==5 && 9==9    // true
5>3  && 5>10    // false
5>8  && 5<10    // false
5>8  && 5<2     // false
```

```
int i=5, j=6;
```

```
if (i++ < 4 || i==j){ // порядок: <, ++, ==, ||
                      // true, операции в левой части
                      // происходят до операций в правой.
```

```
    printf("true\n");
} else {
    printf("false\n");
}
```

```
int x=3;
```

```
if (!(x > 5)){// true, нужны скобки, if (x <= 5) понятней
```

```
...
}
```

```
if (!x > 5){// false
```

```
...
}
```

Библиотека символьных функций <cctype>

```
#include <cctype> // прототипы символьных функций
```

```
//Проверка, что ch является буквенным символом:
```

```
if ( (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z') ) {
```

```
...
```

```
}
```

`int isalpha(int a)` – возвращает `true` (истина, ненулевое значение), если аргумент – буква и `false` (ложь, нулевое значение) – в противном случае.

С применением функции `isalpha()`:

```
if (isalpha(ch)) {
```

```
....
```

```
}
```


isalnum() возвращает **true**, если аргумент – буква или десятичная цифра.

isalpha() возвращает **true**, если аргумент – буква.

isblank() возвращает **true**, если аргумент – пробел или знак горизонтальной табуляции (`\t`).

iscntrl() возвращает **true**, если аргумент – управляющий символ (`NULL`, `\t`, `\n`, `\v`, `DEL` и т.д.).

isdigit() возвращает **true**, если аргумент – десятичная цифра (0-9).

isgraph() возвращает **true**, если аргумент – любой печатаемый символ, отличный от пробела.

islower() возвращает **true**, если аргумент – символ в нижнем регистре.

isprint() возвращает **true**, если аргумент – любой печатаемый символ, включая пробел.

ispunct() возвращает **true**, если аргумент – знак препинания

isspace() возвращает **true**, если аргумент – стандартный пробельный символ (т.е. пробел, новая строка, возврат каретки (\r), горизонтальная табуляция, вертикальная табуляция).

isupper() возвращает **true**, если аргумент – символ в верхнем регистре

isxdigit() возвращает **true**, если аргумент – символ, используемый в шестнадцатеричной системе (т.е. 0-9, a-f или A-F).

tolower() – если аргумент – символ верхнего регистра, возвращает его вариант в нижнем регистре, иначе возвращает аргумент без изменений.

toupper() – если аргумент – символ нижнего регистра, возвращает его вариант в верхнем регистре, иначе возвращает аргумент без изменений .

Логическая операция ? : вместо if-else

выражение1 ? выражение2 : выражение3;

```
int c;
```

```
if (a > b)
```

```
    c = a;
```

```
else
```

```
    c = b;
```



```
int c = (a > b) ? a : b;
```

```
int x = 5;
```

```
int a = (x > 3) ? 10 : 12; // a = 10
```

```
int b = (x == 9) ? 25 : 18; // b = 18
```

Оператор switch (переключение) (вместо if-else)

```
switch (целочисленное-выражение) {  
    case метка1 : оператор (ы) ;  
        break;  
    case метка2 : оператор (ы) ;  
        break;  
    default : оператор (ы);  
}
```

метки – постоянные выражения
(не значения переменных).

Можно использовать
постоянные, определенные со
словом **const**.

```
int color=1;  
switch (color) {  
    case 0:  
        cout << "0\n";  
        break;  
    case 1:  
        cout << "1\n";  
        break;  
    case 2:  
        cout << "2\n";  
        break;  
    default :  
        cout << "Invalid value\n";  
}
```

Перечисления (enumerations)

enum — альтернативный по отношению к **const** способ создания символьных констант (symbolic constants).

```
enum spectrum {red, orange, yellow, green, blue, violet, indigo, ultraviolet};
```

- Объявляет имя нового типа — **spectrum**; при этом **spectrum** называется перечислением, почти так же, как переменная **struct** называется структурой.
- Устанавливает **red**, **orange**, **yellow** и т.д. в качестве символических констант для **целочисленных** значений 0-7. Эти константы называются перечислителями.

```
spectrum band;    // band — переменная типа spectrum
```

```
band = blue;      // правильно, blue - перечислитель
```

```
band = 2000;      // неправильно, 2000 - не перечислитель
```

```
int val=yellow;   // правильно
```

```
band =2;          // неправильно, int более высокий тип данных
```

```
band=spectrum(2); // используем typecast
```

Значения переменной типа **spectrum** ограничены 8 числами.

Операции для типа **spectrum** в общем случае не определены (нельзя `band++`;))

Перечисления

```
enum bits {one = 1, two = 2, four = 4, eight = 8}; // use int values
```

```
enum bigstep {first, second = 100, third};           // 0, 100, 101
```

```
enum {zero, null = 0, one, numero_uno = 1};         // 0, 0, 1,1
```


```
enum { a, b, c, d }; // постоянные a = 0, b = 1, c = 2, d =3
```

Оператор switch и перечисления

`enum{Red, Green, Blue};` // целочисленные постоянные,
// Red=0, Green=1, Blue=2

```
int color=1;


switch (color) {
case Red: // можно case 0:
    cout << "Red\n";
    break;
case Green:
    cout << "Green\n";
    break;
case Blue:
    cout << "Blue\n";
    break;
default :
    cout << "Invalid value\n";
}
```



Green

```
int color=1;

switch (color) {
case Red:
    cout << "Red\n";
case Green:
    cout << "Green\n";
case Blue:
    cout << "Blue\n";
default:
    cout << "Invalid value\n";
}
```



Green
Blue
Invalid value

Пример использования перечисления

```
#include<stdio.h>
enum day{Sun, Mon, Tue, Wed, Thu, Fri, Sat };

/*****/
void tasks(enum day d){

    printf("d=%d. ", d);
    switch(d){
    case Sun:
        printf("It is Sunday. No tasks to be done!\n");
        break;
    case Mon:
        printf("It is Monday. Classes start at 9:10 am.\n");
        break;
    case Tue : case Wed : case Thu : case Fri : case Sat:
        printf("It is Tuesday, ");
        printf("Wednesday, ");
        printf("Thursday, ");
        printf("Friday, ");
        printf("or Saturday.\n");
        break;
    default:
        printf("The day is wrong\n");
    }
}

/*****/
int main(void){
    enum day d1 = Sat;
    tasks(d1);
    tasks(Mon);
    tasks(Thu);
    tasks(Sun);
    tasks(-1);
    return 0;
}

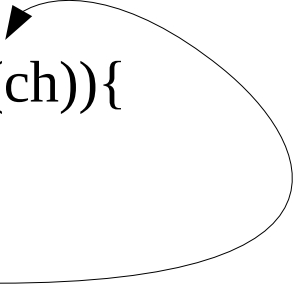
/*****/
```

d=6. It is Tuesday, Wednesday, Thursday, Friday, or Saturday.
d=1. It is Monday. Classes start at 9:10 am.
d=4. It is Tuesday, Wednesday, Thursday, Friday, or Saturday.
d=0. It is Sunday. No tasks to be done!
d=-1. The day is wrong

Операторы **break** и **continue**

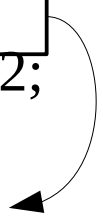
Позволяют программе пропускать часть кода.

```
while(cin.get(ch)){  
    operator1;  
    if(ch=='\n')  
        continue;  
    operator2;  
}  
operator3;
```



Оператор **continue** пропускает остаток тела цикла и начинает новую итерацию

```
while(cin.get(ch)){  
    operator1;  
    if(ch=='\n')  
        break;  
    operator2;  
}  
operator3;
```



Оператор **break** пропускает цикл и переходит на оператор, следующий за циклом.