

# **Курс лекций "Программирование"**

## **Основы программирования на языках С и С++**

---

### **Лекция 3. Конструкции и операции языков С и С++**

**Глухих Михаил Игоревич, к.т.н., доц.**

**[mailto: glukhikh@mail.ru](mailto:glukhikh@mail.ru)**

# Отступление – проверка корректности ввода

---

- ❑ Что делать, если пользователь вводит не то, что от него ожидали?
- ❑ Простой пример – слишком длинное число

# Отступление – вариант с переполнением

---

```
int main(void)
{
    int a, b;
    cin>>a; // Если число слишком большое...
    if (cin.fail())
    {
        cin.clear();
    }
    cin>>b;
    cout<<a<<" "<<b<<endl;
    return 0;
}
```

# Отступление – проверка корректности ввода

---

- ❑ Что делать, если пользователь вводит не то, что от него ожидали?
- ❑ Сложный пример – вообще не число

# СИМВОЛЫ

---

- ❑ Для хранения символов в языке C предусмотрен тип **char**
- ❑ Вместо хранения символа на самом деле хранится 8-разрядное целое число – **код** символа, его номер в таблице кодировки. Например, в кодировке CP1251 пробел имеет код 32, символ 0 – код 48, символ «возврат каретки» – код 13
- ❑ Нуль-символ (не путать с символом 0) - специальный символ, не имеющий отображения, код которого равен 0; в языке C используется для обозначения конца строк

# Примеры определения СИМВОЛОВ

---

```
char c1; // Символ (оди́ночный!)  
// Символьные константы записываются  
// в оди́нарных кавычках  
char c2='a'; // Символ a  
// Варианты escape-последовательностей  
char c3= '\n'; // Новая строка  
char c4= '\r'; // Возврат каретки  
char c5= '\t'; // Табуляция  
char c6= '\\'; // Символ «обратный слэш»  
char c7= '\''; // Символ «оди́нарная кавычка»  
char c8= '\0'; // Нуль-символ
```

# Операции с символами

---

- ❑ С символьным типом можно выполнять все операции, которые можно выполнять с целыми числами (присваивать, складывать, вычитать, сравнивать и так далее)
- ❑ Все эти операции выполняются **над кодами символов**
- ❑  $'0' - ' ' = 16$ , так как код  $'0'$  48, а код  $' '$  32
- ❑  $' ' + 16 = '0'$ , по тем же причинам

# Пример

---

```
// Проверка, что символ является цифрой
if (ch >= '0' && ch <= '9')
{
    // Преобразование символ - соответствующее целое
    // Так можно только потому, что коды цифр идут подряд
    int digit = ch - '0';
    // Вывод кода символа
    // Библиотека вывода «знает»,
    // что char нужно выводить как символ,
    // а int нужно выводить как целое число
    cout<<(int) ch<<endl;
    // ...
}
```



# Отступление – вариант с символами (A)

---

```
int a;
bool success = false;
do
{
    cin>>a;
    if (cin.fail()) {
        cin.clear();
        char ch;
        cin>>ch;
    } else {
        success = true;
    }
} while (!success);
```

# ВВОД И ВЫВОД СИМВОЛОВ

---

- ❑ Выполняется аналогично другим типам с помощью операторов ввода >> и вывода <<
- ❑ При вводе символов по умолчанию пробелы, табуляции, переводы строки (**whitespaces**) игнорируются. При использовании манипулятора `noskipws` все символы вводятся как есть:

```
char ch;
```

```
cin>>noskipws>>ch;
```

```
// skipws - возврат к режиму по умолчанию
```

```
cin>>skipws>>ch;
```

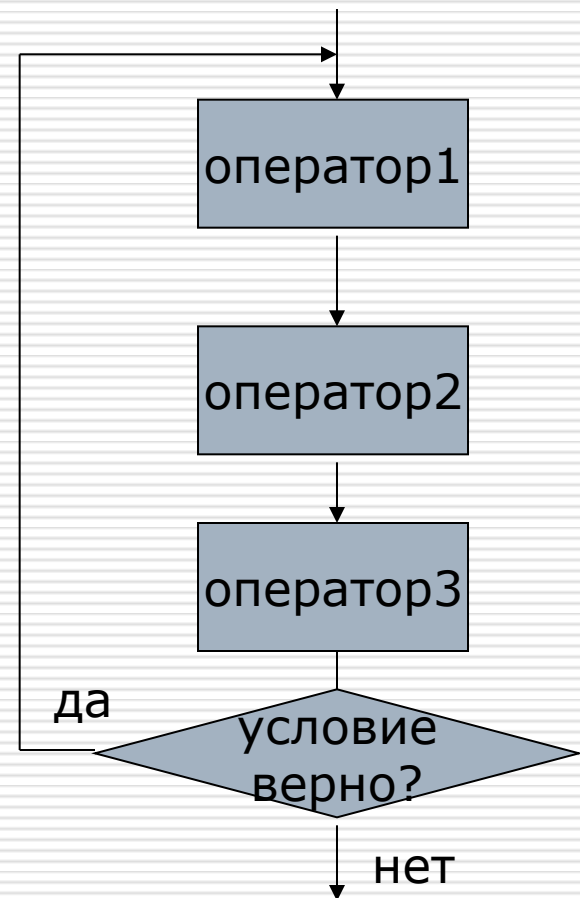
# Отступление – вариант с символами (Б)

---

```
int a;
bool success = false;
do
{
    cin>>a;
    if (cin.fail()) {
        cin.clear();
        char ch;
        cin>>noskipws>>ch;
    } else {
        success = true;
    }
} while (!success);
```

# Конструкция do-while (делай... пока)

```
do
    оператор;
while (условие);
// ИЛИ
do
{
    оператор1;
    оператор2;
    оператор3;
} while (условие);
```



# Особенности

---

- ❑ Первая проверка условия – **после** прохода цикла, поэтому операторы цикла в любом случае выполнятся хотя бы один раз
- ❑ В отличие от языка Паскаль (repeat-until), записывается условие продолжения цикла, а не условие выхода из цикла

# Пример решения задачи

---

- Найти число вхождений цифры  $M$  ( $0 \leq M \leq 9$ ) в неотрицательное целое число  $N$
- Например: цифра  $M=3$  входит в число  $N=5373393$  4 раза

# Вопросы при решении

---

□ Как находить цифры числа?

# Вопросы при решении

---

- Как находить цифры числа?
  - Последнюю цифру можно найти вычислением остатка от деления на 10. Затем, поделив число на 10 нацело, можно отбросить эту последнюю цифру (все это следует из  $N = n_1 + 10 * n_2 + 10^2 * n_3 + \dots$ , где  $n_1, n_2, n_3, \dots$  – цифры числа  $N$ )
- Когда прекратить поиск цифр?



# Текст программы – начало

---

```
#include <iostream>
#include <locale.h>
using namespace std;
int main(void)
{
    int m, n; // цифра и число
    setlocale(LC_ALL, "Russian");
    cout<<"Поиск цифры в числе"<<endl;
    cout<<"Введите цифру и число >0, через пробел: ";
    cin>>m>>n;
    if (m<0 || m>9 || n<0)
    {
        cout<<"Некорректные данные"<<endl;
        return 0;
    }
}
```

# Текст программы – окончание

---

```
int found=0; // Сколько уже найдено цифр
do
{
    int d=n%10; // Очередная цифра
    if (d==m)
        found++;
    n/=10;
} while (n>0);
cout<<"Всего найдено "<<found<<" цифр"<<endl;
return 0;
}
```

# Конструкция for (для..., цикл со счетчиком)

---

```
for (начало; условие продолжения; шаг)  
    оператор;
```

// или

```
for (начало; условие продолжения; шаг)  
{  
    оператор1;  
    оператор2;  
    оператор3;  
}
```

# Эквивалентная конструкция while

---

начало;

**while** (условие продолжения)

{

    оператор1;

    оператор2;

    оператор3;

    шаг;

}

*// На самом деле – не во всех случаях*

# Особенности

---

- Обычно используется, когда известно, сколько раз должен выполняться цикл
- Например: цикл должен выполняться 10 раз

```
for (int n=1; n<=10; n++)
```

```
// Переменная n определяется прямо внутри
```

```
// конструкции for
```

```
{
```

```
...
```

```
}
```

# Особенности

---

- ❑ Любое из полей (начало, условие продолжения, шаг) может быть пустым
- ❑ Если условие продолжения пустое, условие считается всегда верным, цикл становится бесконечным

```
for (int n=1; ; n++)  
{  
    ...  
}
```

# Пример – факториал $n!$

---

```
// Здесь будем вычислять факториал
// Требование:  $n \geq 0$  (контракт)
// Определение функции
double fact(int n) // Заголовок функции
{
    double f=1.0;
    for (int i=2; i<=n; i++)
        f *= i;
    return f;
}
```

# Пример – факториал n!

---

```
int main(void)
{
    setlocale(LC_ALL, "Russian");
    cout<<"Вычисление факториала"<<endl;
    cout<<"Введите целое число: "<<endl;
    int n;
    cin>>n;
    if (n<0) // Проверка корректности
    {
        cout<<"Ошибка - отрицательное число"<<endl;
        return -1;
    }
    cout<<"Факториал "<<n<<" равен "<<fact(n)<<endl;
    return 0;
}
```

---



# Оператор break

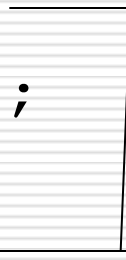
---

- ❑ Прерывает выполнение текущего цикла
- ❑ Обычно используется для того, чтобы проверить условие окончания в **середине** тела цикла, так как другие имеющиеся конструкции позволяют это делать либо **перед** выполнением тела цикла, либо **после** его выполнения

# Например

---

```
for (int n=1; ; n++) // "Бесконечный цикл"  
{  
    оператор1;  
    if (условие) // Если условие верно...  
        break;  
    оператор2;  
}
```



# Пример – определение простоты натурального числа

---

- Простым называется натуральное число, которое делится нацело только на единицу и на себя
- Число, имеющее другие делители, является составным

# Вопросы при решении

---

- Как определить, делится ли число на другое?

# Вопросы при решении

---

- Как определить, делится ли число на другое?
  - Использовать операцию «остаток от деления». Он должен быть равен 0.
- Нам нужно последовательно проверить, делится ли число  $n$  на 2, 3, 4, ...,  $n-1$ . Значит, нам нужно проверить  $n-2$  условия! Как?

# Вопросы при решении

---

- ❑ Как определить, делится ли число на другое?
  - Использовать операцию «остаток от деления». Он должен быть равен 0.
- ❑ Нам нужно последовательно проверить, делится ли число  $n$  на 2, 3, 4, ...,  $n-1$ . Значит, нам нужно проверить  $n-2$  условия! Как?
  - Нужно использовать цикл. Цикл нужно закончить, как только число разделилось хоть на что-нибудь – это значит, что оно составное

# Неверное решение 1

---

...

```
if (n%2==0 || n%3==0 || n%4==0)
    cout<<"Число "<<n<<" составное"<<endl;
else
    cout<<"Число "<<n<<" простое"<<endl;
```

...

- ❑ У числа могут быть и другие делители. Например, число 25 будет ошибочно объявлено простым

# Неверное решение 2

---

```
...  
for (int i=2; i<n; i++)  
{  
    if (n%i==0)  
        cout<<"Число "<<n<<" составное"<<endl;  
    else  
        cout<<"Число "<<n<<" простое"<<endl;  
}
```

...

☐ На консоли появятся сразу  $n-2$  сообщения.



# Правильное решение – начало

---

```
int main(void)
{
    setlocale(LC_ALL, "Russian");
    cout<<"Определение простоты"<<endl<<"Введите число: ";
    int i, n; // Проверяемое число и счетчик цикла
    cin>>n;
    if (n<1)
    {
        cout<<"Число "<<n<<" не является натуральным"<<endl;
        return 0;
    }
    if (n==1)
    {
        cout<<"1 не является ни простым, ни составным"<<endl;
        return 0;
    }
}
```

# Правильное решение – окончание

---

```
for (i=2; i<n; i++)
{
    if (n%i==0)
        break;
}
// Если цикл завершился без помощи break, i будет равно n
// В противном случае i будет меньше
if (i<n)
    cout<<"Число "<<n<<" составное"<<endl;
else
    cout<<"Число "<<n<<" простое"<<endl;
return 0;
}
```

# Вариант с функцией – return вместо break

---

```
// Определение функции
bool isPrime(int n)
{
    // Можно ли быстрее?
    for (int i=2; i<n; i++)
    {
        if (n%i==0)
            return false;
    }
    return true;
}
```

# Можно ли сократить число итераций цикла?

---

- Вообще-то можно.
  - Как минимум, можно проверять делители не до  $n-1$ , а до  $n/2$ . Очевидно, что число  $n$  не может делиться нацело на число в интервале  $[n/2+1; n-1]$

# Можно ли сократить число итераций цикла?

---

- Вообще-то можно.
  - Как минимум, можно проверять делители не до  $n-1$ , а до  $n/2$ . Очевидно, что число  $n$  не может делиться нацело на число в интервале  $[n/2+1; n-1]$
  - А еще можно делить только на простые числа – если  $n$  делится на составное число, то оно делится и на хотя бы одно простое число. Но для этого нужно знать список простых чисел, меньших  $n$

# Можно ли сократить число итераций цикла?

---

- Вообще-то можно.
  - Как минимум, можно проверять делители не до  $n-1$ , а до  $n/2$ . Очевидно, что число  $n$  не может делиться нацело на число в интервале  $[n/2+1; n-1]$
  - А еще можно делить только на простые числа – если  $n$  делится на составное число, то оно делится и на хотя бы одно простое число. Но для этого нужно знать список простых чисел, меньших  $n$
  - Ну, и если идти еще дальше, то можно проверять простые делители только до  $\sqrt{n}$ . Если  $n$  делится на  $m > \sqrt{n}$ , то  $n/m = k$ , причем  $k < \sqrt{n}$  и  $n$  делится на  $k$

# Можно ли решить без break/return?

---

# Можно ли решить без break/return?

---

- Да конечно! И не одним способом.
- ...

```
bool isPrime=true;
for (int i=2; i<=n/2; i++)
    if (n%i==0)
        isPrime=false;
if (isPrime)
    cout<<"Число простое"<<endl;
else
    cout<<"Число составное"<<endl;
```



# А еще можно так...

---

...

```
bool isPrime=true;
```

```
// Цикл с пустым телом (!!!)
```

```
// еще и со сложным условием продолжения
```

```
for (int i=2;
```

```
    i<=n/2 && (isPrime=(n%i > 0)); i++);
```

```
if (isPrime)
```

```
    cout<<"Число простое"<<endl;
```

```
else
```

```
    cout<<"Число составное"<<endl;
```

```
// Будьте проще... а то запутаетесь
```

# Как определить все простые числа в интервале?

---

- ❑ Проверить все числа этого интервала. С ПОМОЩЬЮ ДВУХ **ВЛОЖЕННЫХ** ЦИКЛОВ

```
int min=2, max=1000;
for (int n=min; n<=max; n++)
{
    bool isSimple=true;
    for (int i=2;
        i<=n/2 && (isSimple=(n%i > 0)); i++);
    if (isSimple)
        cout<<"Число "<<n<<" простое"<<endl;
}
```

# Как определить все простые числа в интервале?

---

□ Или так, с помощью функции:

```
int min=2, max=1000;
for (int n=min; n<=max; n++)
{
    if (isPrime(n))
        cout<<"Число "<<n<<" простое"<<endl;
}
```

# Оператор continue


---

- ❑ Прерывает выполнение текущей **итерации** цикла; после этого делается очередной шаг (**for**), проверяется условие продолжения (любой цикл) и, если оно верно, начинается следующая итерация
- ❑ Используется для пропуска окончания итерации
- ❑ Как правило, может быть легко заменена конструкцией if-else

# Например

---

```
for (int n=1; n<10; n++)  
{  
    оператор1;  
    if (условие) // Если условие верно...  
        continue;  
    оператор2;  
}
```



*// Можно ли записать иначе?*

# for/while + continue (найдите разницу)

---

```
for (начало; условие1; шаг)
{
    оператор1;
    if (условие2)
        continue;
    оператор2;
    оператор3;
}
```

# for/while + continue (найдите разницу)

---

начало;

**while** (условие1)

{

оператор1;

**if** (условие2)

**continue;**

оператор2;

оператор3;

шаг;

}

# for/while + continue (найдите разницу)

---

начало;

**while** (условие1)

{

оператор1;

**if** (условие2)

**continue;** // шаг НЕ выполняется

оператор2;

оператор3;

шаг;

}



# Пример использования

---

- ❑ Совершенные числа равны сумме всех своих делителей, кроме самого себя, например:  $6=1+2+3$
- ❑ Найти все совершенные числа в пределах заданного  $N$
- ❑ Вывести их на экран в виде  $6=1+2+3$

# Вопросы при решении

---

- Как находить совершенные числа?

# Вопросы при решении

---

- Как находить совершенные числа?
  - Проверять все потенциальные делители от двух до половины числа. Если сумма всех найденных делителей +1 равна числу – оно совершенное
- Как выводить делители на экран?

# Вопросы при решении

---

- Как находить совершенные числа?
  - Проверять все потенциальные делители от двух до половины числа. Если сумма всех найденных делителей  $+1$  равна числу – оно совершенное
- Как выводить делители на экран?
  - Между делителями нужно выводить плюс, если данный делитель не последний. Последним будет тот делитель, после вывода которого сумма делителей станет равна числу

# Текст программы – начало

---

```
#include <iostream>
#include <locale.h>
using namespace std;
int main(void)
{
    setlocale(LC_ALL, "Russian");
    int n;
    cout<<"Поиск совершенных чисел. Введите максимум: ";
    cin>>n;
    if (n<6)
    {
        cout<<"Совершенных чисел не найдено"<<endl;
        return 0;
    }
}
```

# Текст программы – продолжение

---

```
for (int i=6; i<=n; i++) // Числа
{
    int sum=1; // Сумма делителей
    for (int j=2; j<=i/2; j++)
    {
        if (i%j==0)
        {
            sum += j;
            // Чтобы не тратить зря время
            if (sum > i)
                break;
        }
    }
    if (sum != i)
        continue;
```

# Текст программы – окончание

---

```
    cout<<i<<"=1+"; sum=1;
    for (int j=2; j<=i/2; j++)
        if (i%j==0)
        {
            cout<<j;
            sum += j;
            if (sum==i) // Если последний делитель найден
            {
                cout<<endl;
                break;
            }
            else cout<<"+"; // Делитель не последний
        }
    }
    return 0;
}
```

# Проблемы

---

- Как сократить число вычислений?
  - Есть разные варианты. Хотя все они не очень помогают.
- Разбиение на функции
  - `isPerfect`
  - `printPerfect`



# Функция isPerfect

---

```
bool isPerfect(int n)
{
    int sum=1; // Сумма делителей
    for (int j=2; j<=n/2; j++)
    {
        if (n%j == 0)
        {
            sum += j;
            // Чтобы не тратить зря время
            if (sum > n)
                return false;
        }
    }
    return sum == n;
}
```

---

# Функция printPerfect

---

```
void printPerfect(int n)
{
    cout<<n<<"=1+"; sum=1;
    for (int j=2; j<=n/2; j++)
        if (n%j == 0)
        {
            cout<<j;
            sum += j;
            if (sum == n) // Если последний делитель найден
            {
                cout<<endl;
                return;
            }
            else cout<<" "; // Делитель не последний
        }
}
```

# Программа с функциями

---

```
int main(void)
{
    setlocale(LC_ALL, "Russian");
    int n;
    cout<<"Поиск совершенных чисел. Введите максимум: ";
    cin>>n;
    // ...
    for (int i=6; i<=n; i++)
        if (isPerfect(i)) // Вызов функции
            printPerfect(i); // Вызов функции
    return 0;
}
```

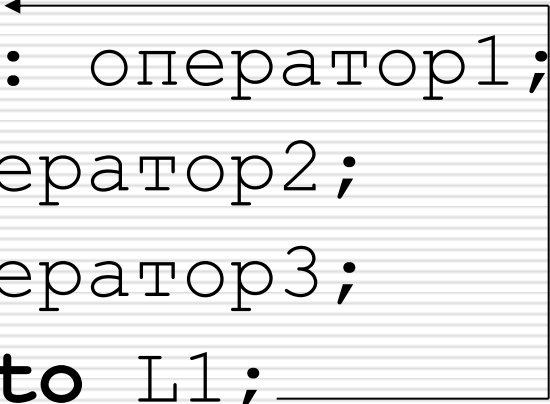
# Команда goto

---

- ❑ Переводит управление на указанную метку

*// Например*

```
L1: оператор1;  
оператор2;  
оператор3;  
goto L1;
```

A diagram consisting of a horizontal line with an arrow pointing left, starting from the 'goto L1;' statement and ending at the 'L1:' label, indicating a jump in the flow of execution.

*// Использовать не рекомендуется*

# Итоги

---

## □ Рассмотрели

- символы
- циклы `do...while`, `for`
- операторы `break`, `continue`
- простые функции

## □ Далее

- системы счисления
- способы хранения чисел,
- классы хранения, файлы