C++

Единственный способ изучать новый язык программирования - писать на нем программы.

Брайэн Керниган

История языка Си++

Язык Си++ был разработан в начале 1980-х гг. Бьерном Страуструпом из компании AT&T Bell Laboratories. Си++ основан на языке Си. Два символа "++« в названии – это игра слов, символами "++" в языке Си обозначается операция инкремента (увеличение значения переменной на 1).

История языка Си++

Таким образом, Си++ был задуман как язык Си с расширенными возможностями. Большая часть языка Си вошла в Си++ как подмножество, поэтому многие программы на Си можно скомпилировать (т.е. превратить в набор низкоуровневых команд, которые компьютер может непосредственно выполнять) с помощью компилятора Си++.

Состав языка

- В тексте на любом естественном языке можно выделить четыре основных элемента: символы, слова, словосочетания и предложения.
- Подобные элементы содержит и алгоритмический язык, только слова называют лексемами (элементарными конструкциями), словосочетания — выражениями, а предложения — операторами.

Состав языка

Лексемы образуются из символов, выражения — из лексем и символов, а операторы — из символов, выражений и лексем.

Состав языка

- Операторы бывают исполняемые и неисполняемые.
- *Исполняемые операторы* задают действия над данными.
- *Неисполняемые операторы* служат для описания данных, поэтому их часто называют операторами описания или просто описаниями.

Состав языка

- Принято исходный код программ на C++ сохранять с расширением .cpp после имени файла (происходит такая идея от названия «C Plus Plus» и от того, что во многих операционных системах знак плюс нельзя использовать в именах файлов и каталогов).

Состав языка

- Для C++ существует масса IDE (например, NetBeans). Внутри IDE (интегрированной среды разработки) процесс компиляции и запуска автоматизирован и, как правило, скрыт от разработчика. Но эти процессы всё равно, происходят каждый раз при попытке запустить программу, притом строго в рассмотренной нами последовательности, т.е. самое минимальное изменение в программном коде требует пересохранения файла с исходным кодом, перекомпиляции и перезапуска программы.

Состав языка

- Перед тем, как создаётся исполняемый код, программа анализируется отладчиком, который ищет в исходном коде существующие и потенциальные ошибки. Если ошибок не найдено, то команда make ничего не выведет на экран в результате своей работы, иначе — будет представлена информация об ошибках с указанием строк, в которых они присутствуют. Пока ошибки не будут исправлены, исполнимый файл не будет создан (или не обновлён, если существовал ранее).

Алфавит языка

Алфавит C++ включает:

1. прописные и строчные латинские буквы и знак подчеркивания;
2. арабские цифры от 0 до 9;
3. специальные знаки, например, {, %, # и т.д.
4. пробельные символы: пробел, символы табуляции, символы перехода на новую строку.

Алфавит языка

Из символов алфавита формируются *лексемы* языка:

- идентификаторы;
- ключевые (зарезервированные) слова;
- знаки операций;
- константы;
- разделители (скобки, точка, запятая, пробельные символы).

Алфавит языка

В тесте программы можно использовать **комментарии**.

Если текст с двух символов «косая черта» // и заканчивается символом перехода на новую строку или заключен между символами /* и */, то компилятор его игнорирует.

Для хранения данных в C++ используются различные сущности, наиболее простыми из них являются литералы, константы и переменные.

Литералом называется явно указанное в исходном коде программы значение определенного типа.

```
cout << 1024; // выводим на экран целочисленный литерал
cout << "mir"; // выводим на экран строковый литерал
```

Идентификаторы

- **Переменной** называется именованная область памяти компьютера (имя которой задаёт разработчик) в которую можно записывать (в том числе повторно, замещая ранее хранимое значение) значения определенного типа и откуда эти значения можно читать.
- При создании любой переменной требуется указать её тип и задать имя.

Идентификаторы

Например:

```
int per1; // создали переменную типа int с именем per1
per1 = 25; // сохранили в переменную целое число 25
int b; // создали переменную типа int с именем b
b = 3 + per1; // прочитали значение 25, сложили его с 3 и сумму записали в b
cout << b; // прочитали из переменной b значение 28 и вывели его на экран
```

Идентификаторы

Использованный в примере тип **int** позволяет хранить целочисленные значения из некоторого диапазона (диапазон будет представлен в таблице далее). Перед тем как использовать переменную (т.е. записывать в неё значение или читать из ней значение) её обязательно нужно объявить (указав её тип и задав имя). Изменить тип переменной или повторно создать переменную — невозможно.

Идентификаторы

Идентификатор — это имя программного объекта.

В идентификаторе могут использоваться латинские буквы, цифры и знак подчеркивания.

Прописные и строчные буквы различаются.

Первым символом идентификатора может быть буква или знак подчеркивания.

Идентификаторы

Длина идентификатора по стандарту не ограничена. Идентификатор создается на этапе объявления переменной, функции, типа и т.п., после этого его можно использовать в последующих операторах программы.

Константы

Константной называется именованная область памяти, в которую при создании можно записать значение определенного типа, но далее по ходу программы это значение можно только читать (и нельзя изменять).

```
const int k1 = 13; // создали константу типа int с именем k1 и записали в неё значение
cout << k1 = 12; // но нельзя изменить, это приведёт к ошибке
```

Идентификаторы

При выборе идентификатора необходимо иметь в виду следующее:

1. идентификатор не должен совпадать с ключевыми словами и именами используемых стандартных объектов языка;
2. не рекомендуется начинать идентификаторы с символа подчеркивания;
3. на идентификаторы, используемые для определения внешних переменных, налагаются ограничения компоновщика.
4. Для улучшения читаемости программы следует давать объектам осмысленные имена.

Концепция типа данных

Тип данных определяет:

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

Все типы языка C++ можно разделить на *простые* (скалярные), *составные* (агрегатные) и *функциональные*. Простые типы могут быть стандартными и определенными программистом.

Концепция типа данных

В языке C++ определено шесть стандартных простых типов данных для представления целых, вещественных, символьных и логических величин.

На основе этих типов, а также массивов и указателей (указатель не является самостоятельным типом, он всегда связан с каким-либо другим конкретным типом), программист может вводить описание собственных простых или структурированных типов.

К структурированным типам относятся перечисления, функции, структуры, объединения и классы.

Простые типы данных

Простые типы делятся на целочисленные типы и типы с плавающей точкой.

Для описания *стандартных типов* определены следующие ключевые слова:

1. int (целый);
2. char (символьный);
3. wchar_t (расширенный символьный);
4. bool (логический);
5. float (вещественный);
6. double (вещественный с двойной точностью).

Простые типы данных

Существует четыре *спецификатора типа*, уточняющих внутреннее представление и диапазон значений стандартных типов:

1. short (короткий);
2. long (длинный);
3. signed (со знаком);
4. unsigned (без знака).

Простые типы данных

Диапазоны значений простых типов данных для IBM PC

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`	$1.7\ e^{-308}$ .. $1.7\ e^{+308}$	8
`long double`	$3.4\ e^{-4932}$ .. $3.4\ e^{+4932}$	10

Простые типы данных

Символьный тип

Данные типа **char** в памяти компьютера всегда занимают 1 байт. Символьный тип может быть со знаком или без него. В величинах со знаком *signed char* можно хранить значение от -128 до 127. Соответственно значения переменных типа *unsigned char* могут находиться в диапазоне от 0 до 255.

При работе с символьными данными нужно помнить, что если в выражении встречается одиночный символ, то он должен быть заключен в одинарные кавычки (‘a’).

Простые типы данных

Целочисленный тип

Переменная типа **int** в памяти компьютера может занимать либо 2, либо 4 байта. Это зависит разрядности процессора. По умолчанию все целые типы считаются знаковыми, то есть спецификатор *signed* можно не указывать. Спецификатор *unsigned* позволяет представлять только положительные числа.

Простые типы данных

Вещественный тип

Число с плавающей точкой представлено в форме mE +- p, где m — мантисса (целое или дробное число с десятичной точкой), p — порядок (целое число). Обычно величины типа float занимают 4 байта, а double 8 байт.

Таблица диапазонов значений вещественного типа:

float	3,4E-38…3,4E+38	4 байта
double	1,7E-308…1,7E+308	8 байт
long double	3,4E-4932…3,4E+4932	8 байт

Простые типы данных

Логический тип

Переменная типа **bool** может принимать только два значения *true (*истина*)* или *fasle (*ложь*).* Любое значение, не равное нулю, интерпретируется как *true.* Значение *false* представлено в памяти как 0.

Простые типы данных

Тип void

Тип void используется для определения функций, которые не возвращают значения, для указания пустого списка аргументов функции, как базовый тип для указателей и в операции приведения типов.

Выражения

Из констант, переменных, разделителей и знаков операций можно конструировать выражения.

- ✔ Каждое выражение представляет собой правило вычисления нового значения.
- ✔ Если выражение формирует целое или вещественное число, то оно называется арифметическим.
- ✔ Пара арифметических выражений, объединенная операцией сравнения, называется отношением.
- ✔ Если отношение имеет ненулевое значение, то оно – истинно, иначе – ложно.

Выражения

Приоритеты операций в выражениях

Ранг	Операции
1	( ) [ ] -> .
2	! ~ — ++ — & * (тип) sizeof тип( )
3	* / % (мультипликативные бинарные)
	+ — (аддитивные бинарные)
5	<< >> (поразрядного сдвига)
6	< > <= >= (отношения)
7	== != (отношения)
8	& (поразрядная конъюнкция «И»)
9	^ (поразрядное исключающее «ИЛИ»)
10	| (поразрядная дизъюнкция «ИЛИ»)
11	&& (конъюнкция «И»)
12	|| (дизъюнкция «ИЛИ»)
13	?: (условная операция)
14	= *= /= %= -= &= ^= |= <<= >>= (операция присваивания)
15	, (операция запятая)

Основные библиотеки

Библиотеки - гардеробы, из которых умелые люди могут извлекать кое-что для украшения, многое - для любопытства и еще больше для употребления.

Библиотека iostream

В первой строке программы с помощью директивы **#include** происходит подключение заголовочного файла **iostream**.

Заголовочные файлы содержат описание функций и других готовых элементов, которые можно использовать в своих программах после того, как заголовочный файл подключён. iostream входит в стандартную библиотеку C++, но заголовочные файлы можно создавать и самостоятельно, помещая туда часто используемые функции, шаблоны и прочие заготовки.

Библиотека iostream

Заголовочный файл **iostream** содержит набор готовых функций для потокового ввода и вывода.

Библиотека iostream

Ввод данных

Ввод данных в C++ осуществляется с помощью команды **cin** (**C**onsole **In**put).

Аргумент этой функции передаётся не в круглых скобках, а через оператор >> (аналог перенаправления в GNU/Linux).

Вывод данных

Вывод данных в C++ осуществляется с помощью команды **cout** (**C**onsole **Out**put).

Пространство имен

std — это пространство имён, определённое для всей стандартной библиотеки C++, а «::» — это оператор разрешения области видимости, который указывает, из какого пространства имён должен браться следующий за ним идентификатор.

Пространство имен

Пространство имён — группа идентификаторов, внутри которой все идентификаторы уникальны (не повторяются).

Пространство имен

С помощью идентификаторов, состоящих из латинских букв и цифр можно давать имена различным сущностям программы: переменным, функциям, методам, классам. С помощью разных пространств имён можно использовать одни и те же имена в одной и той же программе. Кроме того, пространство имён позволяет решить следующую программу: в собственных программах для создания сущностей мы можем использовать те же имена, что задействованы в сторонних библиотеках, в том числе, в стандартной библиотеке C++.

Пространство имен

Если отсутствует необходимость в использовании разных пространств имён в рамках одной программы, то можно однажды задать пространство и далее обращаться ко всем именам без его указания.

using namespace std;

Пространство имен

На всей стандартной библиотеке единое пространство имён std помогает объединить описанные в библиотеке ресурсы в единое целое. Стандартная библиотека устроена так, что в разных её файлах нет повторяющихся идентификаторов на одном уровне. Стандартная библиотека распределена по разным заголовочным файлам, каждый из которых мы можем подключить к программе, но пространство имён везде общее — std.

Библиотека iostream

```
#include <iostream>
using namespace std;
int main()
{
   setlocale(LC_ALL,””);
   int i;
   cout << "Введите целое число\n";
   cin >> i;
   cout << "Вы ввели число" << i << ", спасибо!";
return 0;
}
```

Библиотека iostream

Пример программы, выводящей на экран квадрат числа, введённого пользователем с клавиатуры:

```
#include <iostream>
using namespace std;
int main()
  { setlocale(LC_ALL,””);
    cout << “Введите число: ";
    int s;
    cin >> s;
    cout << «Квадрат числа: ";
    cout << s*s << endl;
    return 0;
   }
```

Библиотека iostream

return 0

Ноль означает отсутствие ошибки.

Для функции main, даже объявляя её как возвращающую целое число, и только для неё, можно ничего не возвращать.

Библиотека iostream

Пример программы, выводящей на целую часть вещественного числа, введённого пользователем с клавиатуры:

```
#include <iostream>
using namespace std;
int main() {
    double num;
    cout << "Vvedite chislo: ";
    cin >> num;
    int ch = (int) num;
    cout << "Tselaya chast chisla: " << ch << endl;
    return 0;
}
```

Преобразование типов данных

В C++ различают два вида преобразования типов данных: явное и неявное.

- **Неявное преобразование** происходит автоматически. Это выполняется во время сравнения, присваивания или вычисления выражения различных типов.

Наивысший приоритет получает тот тип, при котором информация теряется менее всего. Не стоит злоупотреблять неявным преобразованием типов, так как могут возникнуть разного рода непредвиденные ситуации.

Преобразование типов данных

Явное приведение осуществляется с помощью указания целевого типа данных (того, к которому нужно привести) в круглых скобках перед выражением:

```
double s = 2.71;
int t = (int) s;
cout << t << endl; // 2
cout << (int) 3.14 << endl; // 3
cout << (int) (2.5 + t) << endl; // 4
```

Приведение к целым числам от вещественных осуществляется путём отбрасывания целой части (не округлением).

Преобразование типов данных

В C++ к тому же возможно приведение между логическим и числовыми типами.

Любое ненулевое число приводится к true, число 0 или 0.0 — к false. И, наоборот, false преобразуется в 0, а true — в 1.

```
bool b = true;
int t = (int) b;
cout << t << endl; // 1
cout << (int) false << endl; // 0
bool n = (bool) 42; // true
bool m =  (bool) (5 * 2 - 10); // false
bool k = (bool) 0.43; // true
double num1 = 2 + (double) (k || m); // 3.0
```

Библиотека math.h

- Чтобы воспользоваться сложными математическими действиями, нам нужно подключить в программу библиотеку, в которой и содержаться эти функции, а именно:

#include<math.h>

Библиотека math.h

Рассмотрим, какие функции содержатся в этой библиотеки.

- abs – это модуль, возвращает положительное число
- acos (xxx)- арккосинус
- asin (sss) — арксинус
- atan (poiy) — арктангенс
- cos (sgrgrg) — косинус
- Random- вывод случайных чисел
- exp — экспонента
- log (56) — натуральный логарифм
- log10 (45,755) — это логарифм по основанию десять.
- pow(xx,yyy)- возведение в степень
- sin — синус
- tan — тангенс

Библиотека cmath

Имя функции	Описание
abs	Возвращает абсолютную величину (модуль) целого числа
acos	арккосинус
asin	арксинус
atan	арктангенс
ceil	округление до ближайшего большего целого числа
cos	косинус
cosh	гиперболический косинус
exp	вычисление экспоненты
fabs	абсолютная величина (для чисел с плавающей точкой)
floor	округление до ближайшего меньшего целого числа
fmod	вычисление остатка от деления нацело для чисел с плавающей точкой
log	натуральный логарифм
Log10	логарифм по основанию 10
pow(x,y)	результат возведения x в степень y, x^y
sin	синус
sinh	гиперболический синус
sqrt	квадратный корень
tan	тангенс
tanh	гиперболический тангенс

Также в C++ доступны две константы: число «пи» и число «е» (основание экспоненциальной функции или число Неппера). Их можно получить с помощью констант

M_PI и M_E

Если гора не идет к Магомету, то Магомет идет к горе.

Условный оператор

Пойдет направо – песнь заводит
Налево – сказку говорит…

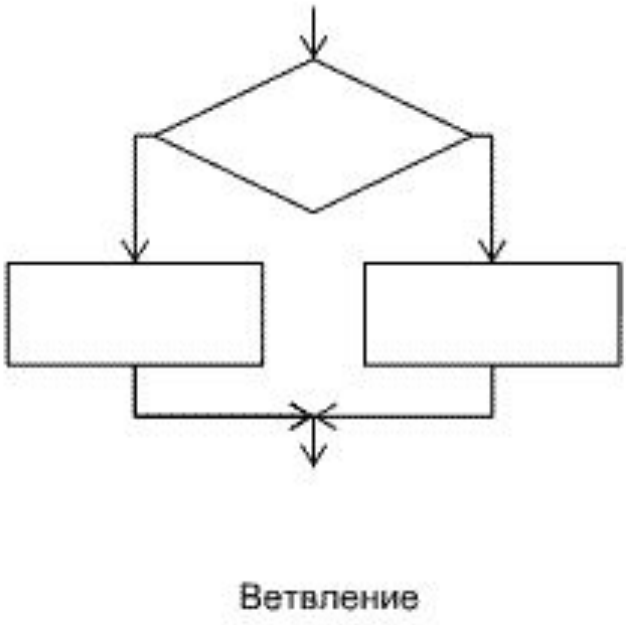
А.С. Пушкин

Условный оператор

Условный оператор ***if*** используется для разветвления процесса вычислений на два направления.

Формат оператора:

if (выражение) оператор_1;
[else оператор_2;]



Структурная схема оператора

Условный оператор

- Сначала вычисляется выражение, которое может иметь арифметический тип или тип указателя.
- Если оно не равно нулю, выполняется первый оператор, иначе — второй. После этого управление передается на оператор, следующий за условным.

Условный оператор

- Одна из ветвей может отсутствовать.
- Если в какой-либо ветви требуется выполнить несколько операторов, их необходимо заключить в блок. Блок может содержать любые операторы, в том числе описания и другие условные операторы.

Условный оператор

Знаки сравнения:

==	равно
!=	не равно
<	меньше, чем
>	больше, чем
<=	меньше или равно
>=	больше или равно

Условный оператор

Программа:

```
//Площадь треугольника
#include <iostream.h>
#include <math.h>
main()
{float a,b,c,P,S;
Cout<<”\na=”; cin>>a;
Cout<<”\nb=”; cin>>b;
Cout<<”\nc=”; cin>>c;
If (a>0 && b>0 && c>0 && a+b>c && a+c>b && b+c>a)
    {
     P=(a+b+c)/2;
     S=sqrt(P*(P-a)*(P-b)*(P-c));
     Cout<<”\nПлощадь треугольника=”<<S;
     }
else cout (<<”\n Неверные исходные данные.);
```

Условный оператор

Туристы вышли из леса на шоссе неподалеку от километрового столба с отметкой А км и решили пойти на ближайшую автобусную остановку. Посмотрев на план местности, руководитель группы сказал, что автобусные остановки расположены на километре В и на километре С. Куда следует пойти туристам?

Условный оператор

- Даны три действительных числа a, b, c. Найти наибольшее из них.

Оператор множественного выбора

SWITCH

Использование оператора SWITCH

Инструкция множественного выбора switch позволяет выполнять различные части программы в зависимости от того,

какое значение будет иметь некоторая целочисленная переменной (её называют «переменной-переключателем», а «switch» с английского переводится как раз как «переключатель»).

Использование оператора SWITCH

Оператор switch состоит из двух частей.

- Первая часть оператора switch представляет собой условие, которое появляется после ключевого слова switch.
- Вторая часть представляет собой возможные варианты соответствия. Когда программа встречает оператор switch, она сначала исследует условие, а затем пытается найти среди возможных вариантов тот, который соответствует условию.

Использование оператора SWITCH

Схема инструкции такова:

```
switch (переключатель)  {
  case значение1:
    инструкция1;
    break;
  case значение2:
    инструкция2;
    break;
  …
  default:
    инструкция_по_умолчанию;
}
```

Использование оператора SWITCH

Рассмотрим все элементы оператора:

- переключатель — это целочисленная переменная или выражение дающее целочисленный результат;

Использование оператора SWITCH

- **значение1**, **значение2**, … — это целочисленные литералы, с которыми будет сравниваться значение переключателя.

Если переключатель равен **значениюN**, то программа будет выполняться со строки, следующей за case **значениеN**: и до ближайшего встреченного break, либо до конца блока switch (если break не встретится);

Использование оператора SWITCH

- Если программа находит соответствие, выполняются указанные операторы.
- Если же ни один из указанных вариантов не соответствует условию, то выполняется вариант default.

Обратите внимание на использование оператора break в каждом варианте предыдущей программы.

Использование оператора SWITCH

- Оказывается, если C++ встречает вариант, соответствующий условию оператора switch, то он подразумевает, что все последующие варианты тоже соответствуют условию.

Использование оператора SWITCH

- Оператор break указывает C++ завершить текущий оператор switch и продолжить выполнение программы с первого оператора, следующего за оператором switch. Если удалить операторы break из программы, то программа выведет не только требуемое сообщение, но и сообщение для всех последующих вариантов (потому что если один вариант является истинным, то и все последующие варианты в C++ рассматриваются как истинные).

Использование оператора SWITCH

Такое поведение программы можно использовать во благо. Например, можно разместить несколько подряд идущих меток с разными литералами, для которых будет выполняться один и тот же код:

Использование оператора SWITCH

```
switch(ans) {
case 'Д':
case 'д':
case 'Y':
case 'y':
   cout << "Продолжаем программу";
case 'Н':
case 'н':
case 'N':
case 'n':
    cout << "Останавливаем программу";
    break;
default:
    cout << "Вы ввели неподходящий символ";
}
```

Использование оператора SWITCH

- **default:** — это метка инструкции после которой будут выполняться в том случае, если выше ни одно из **значенийN** не совпало с переключателем. Метка default — необязательная: можно её не включать в блок switch меток или не выполнять после неё никаких команд;
- **инструкцияN** — простая или составная инструкция. Притом в случае составной несколько команд не обязательно объединять в блок, можно их просто написать друг за другом разделяя с помощью «;» (и начиная новые строки для удобства).

Использование оператора SWITCH

Если мы захотим как-то оповестить пользователя о том, что он ввёл неподходящий символ, то пригодится метка default:

```
switch(ans) {
    case 'Д':
        cout << "Продолжаем программу";
        break;
    case 'Н':
        cout << "Останавливаем программу";
        break;
    default:
        cout << "Вы ввели неподходящий символ";
}
```

Использование оператора SWITCH

Вывод словесного описания оценки, основываясь на текущей оценке ученика

```
#include <iostream.h>
int main()
{
     int grade;
    cout << "Введите оценку" << endl;
    cin >> grade;
    switch (grade)
       { case 5: cout << "Поздравляем, вы получили пять" << endl;
    break;
          case 4: cout << "Хорошо, у вас хорошо" << endl; break;
          case 3: cout << "У вас всего лишь удовлетворительно" <<
    endl; break;
          case 2: cout << "Плохо, у вас двойка" << endl; break;
          default: cout << "Ужасно! Учите лучше!" << endl; break;
       }
}
```

Использование оператора SWITCH

Switch позволяет только сравнивать переключатель с конкретными значениями, но не позволяет для какой-то из веток задать условие в виде целого диапазона значений с использованием операторов сравнения (например, с использованием строгих неравенств и логических операторов «и» или «или»).

В качестве переключателя могут выступать только целочисленные переменные или выражения.

Циклы в C++

С другой стороны, мы не можем игнорировать эффективность

Циклы

- Существует два типа циклов: типа «пока» и типа «n-раз».
- Первый тип «пока» предназначен для повторения некоторых действий до тех пор, пока выполняется некоторое условие.
- Второй тип «n-раз» предназначен для повторения некоторых действий заранее известное количество раз.

Цикл типа «n-раз» (оператор for)

Оператор for содержит три параметра.

- Первый называется инициализацией,
- второй — условием повторения,
- третий — итерацией.

```
for (инициализация; условие; итерация)
    {
        //тело цикла, т. е. действия
повторяемые циклично
    }
```

Цикл типа «n-раз» (оператор for)

В первом параметре обычно выбирают какую-то переменную, с помощью которой будет подсчитываться количество повторений цикла. Её называют счетчиком. Счётчику задают некоторое начальное значение (указывают, начиная с какого значения он будет изменяться).

Цикл типа «n-раз» (оператор for)

- Во втором параметре указывают некоторое ограничение на счётчик (указывают, до какого значения он будет изменяться).

Цикл типа «n-раз» (оператор for)

В третьем параметре указывают выражение, изменяющее счётчик после каждого шага цикла. Обычно это инкремент или декремент, но можно использовать любое выражение, где счётчику будет присваиваться некоторое новое значение.

Цикл типа «n-раз» (оператор for)

Перед первым шагом цикла счётчику присваивается начальное значение (выполняется инициализация). Это происходит лишь однажды.

Представленная программа выводит на экран числа от 1 до 100:

```
for (int i = 1; i <= 100; i++)
    {
       cout << i << " ";
    }
```

Цикл типа «n-раз» (оператор for)

Перед каждым шагом цикла (но после инициализации) проверяется условие повторения, если оно истинно, то в очередной раз выполняется тело цикла. При этом, тело цикла может не выполниться ни разу, если условие будет ложным в момент первой же проверки.

```
for (int i = 1; i <= 100; i++)
   {
      cout << i << " ";
   }
```

Цикл типа «n-раз» (оператор for)

После завершения каждого шага цикла и перед началом следующего (и, значит, перед проверкой условия повторения) выполняется итерация.

```
for (int i = 1; i <= 100; i++)
   {
      cout << i << " ";
   }
```

Цикл типа «n-раз» (оператор for)

Представленная программа выводит на экран числа от 10 до −10:

```
for (int s = 10; s > -11; s--)
   {
      cout << s << " ";
   }
```

Представленная программа выводит на экран нечётные числа от 1 до 33:

```
for (int i = 1; i <= 33; i = i + 2)
   {
      cout << i << " ";
   }
```

Цикл типа «n-раз» (оператор for)

Представленная программа вычислит сумму элемнтов фрагмента последовательности 2, 4, 6, 8,… 98, 100. Итак:

```
int sum = 0; // Сюда будем накапливать результат
for (int j = 2; j <= 100; j=j+2) {
  sum = sum + j;
}
cout << sum;
```

Цикл типа «n-раз» (оператор for)

Представленная программа будет возводить число из переменной a в натуральную степень из переменной n:

```
double a = 2;
int n = 10;
double res = 1; // Сюда будем накапливать результат
for (int i = 1; i <= n; i++) {
  res = res * a;
}
cout << res;
```

Цикл типа «n-раз» (оператор for)

Представленная программа выведет на экран 10 первых элементов последовательности 2n+2, где n=1, 2, 3…:

```
for (int i = 1; i < 11; i++)
  {
     cout << 2*i + 2 << " "
  }
```

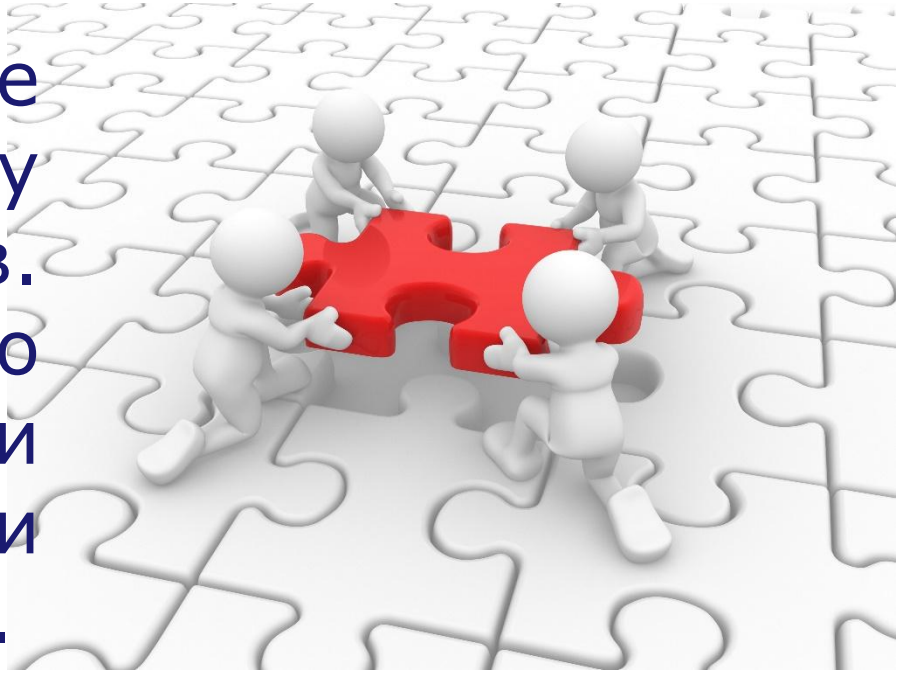
Цикл типа «n-раз» (оператор for)

Представленная программа выведет на экран 10 первых элементов последовательности $2a_{n-1}+3$, где $a_1=3$:

```
int a = 3;
for (i=1; i<=10;i++)
   {
      cout << a << " ";
      a = 2*a + 3;
   }
```

Цикл типа «n-раз» (оператор for)

В одном цикле можно задавать сразу несколько счётчиков. При этом несколько выражений в итерации и в инициализации разделяются запятыми.



Условие повторения можно задавать только одно, но оно может быть выражением, содержащим сразу несколько счётчиков.

Цикл типа «n-раз» (оператор for)

Представленная программа выведет на экран 10 первых элементов последовательности $2a_{n-1}-2$, где $a_1=3$:

```
for (int a=3, i=1; i<=10; a=2*a-2, i++)
    {
       cout << a << " ";
    }
```

Цикл типа «n-раз» (оператор for)

Представленная программа выведет на экран такую последовательность «0 -1 -4 -9 -16 -25»:

```
for (int a=0, b=0; a-b<=10; a++, b--)
   {
      cout << a*b << " ";
   }
```

Досрочное завершение цикла (оператор break)

Как цикл типа «пока» так и цикл типа «n-раз» можно завершить досрочно, если внутри тела цикла вызвать оператор **break**.

При этом произойдёт моментальный выход из цикла, не будет закончен даже текущий шаг (т. е. если после break присутствовали какие-то ещё операторы, то они не выполнятся).

Досрочное завершение цикла (оператор break)

В результате работы следующего примера на экран будут выведены только числа «1 2 3 4 Конец»:

```
for (int a=1; a<=10; a++) {
  if(a == 5) {
    break;
  }
  cout << a << " ";
}
cout << "Конец";
```

Досрочное завершение цикла (оператор break)

Когда программа будет выполнять цикл в пятый раз(войдёт в цикл с счётчиком равным 5), сразу же будет проверено и окажется истинным условие при котором выполнится оператор break.

Оставшаяся часть тела цикла (вывод на экран) уже производится не будет: программа сразу перейдёт к выполнению операций указанных после цикла и далее.

Досрочное завершение цикла (оператор break)

С помощью оператор break можно прервать заведомо бесконечный цикл. Пример (на экран выведется «100 50 25 12 6 3 1 0 » и после этого цикл остановится):

```
int s = 100;
while (true) {
  cout << s << " ";
  s = s / 2;
  if(s == 0) {
    break;
  }
}
```

Досрочное завершение цикла (оператор break)

Оператор break имеет смысл вызывать только при наступлении какого-то условия, иначе цикл будет завершен досрочно на первом же своём шаге.

```
int a;
for (a=25; a>0; a--)
 {
   break;
   cout << a << " ";
 }
cout << "a=" << a;
```

Досрочное завершение цикла (оператор break)

В представленном выше примере вывода в цикле на экран не произойдёт ни разу, а когда переменная a выведется на экран после цикла, то окажется, что её значение ни разу не менялось, т. е. выведено будет «a=25» (и ничего больше).

Обратите внимание также на то, что переменная была объявлена до начала цикла. Когда переменная объявляется в параметрах цикла, то она оказывается недоступной за его пределами, а в данном случае требовалось иное — узнать какое значение будет у счётчика после завершения цикла.

Задачи

- Создайте программу, выводящую на экран все четырёхзначные числа последовательности 1000 1003 1006 1009 1012 1015 ….
- Создайте программу, выводящую на экран первые 55 элементов последовательности 1 3 5 7 9 11 13 15 17 ….
- Создайте программу, выводящую на экран все неотрицательные элементы последовательности 90 85 80 75 70 65 60 ….
- Создайте программу, выводящую на экран первые 20 элементов последовательности 2 4 8 16 32 64 128 ….

Задачи

- Выведите на экран все члены последовательности $2a_{n-1}-1$, где $a_1=2$, которые меньше 10000.
- Выведите на экран все двузначные члены последовательности $2a_{n-1}+200$, где $a_1= -166$.
- Создайте программу, вычисляющую факториал натурального числа n, которое пользователь введёт с клавиатуры.

Задачи

- Выведите на экран все положительные делители натурального числа, введённого пользователем с клавиатуры.

Проверьте, является ли введённое пользователем с клавиатуры натуральное число — простым. Постарайтесь не выполнять лишних действий (например, после того, как вы нашли хотя бы один нетривиальный делитель уже ясно, что число составное и проверку продолжать не нужно). Также учтите, что наименьший делитель натурального числа n, если он вообще имеется, обязательно располагается в отрезке [2; √n].

Задачи

- Для введённого пользователем с клавиатуры натурального числа посчитайте сумму всех его цифр (заранее не известно сколько цифр будет в числе).

- Пользователь вводит с клавиатуры последовательность ненулевых целых чисел. Программа должна вывести на экран максимальный и минимальный элементы последовательности сразу после того, как пользователь введёт 0 (т.е. заранее длина последовательности неизвестна).

Задачи

- Пользователь вводит с клавиатуры арифметический пример в таком формате «2+3.5» или «3.14*8», программа должна вычислить и вывести правильный ответ на экран. В примере должны быть допустимы операции сложения, умножения, вычитания, деления (с остатком). После вывода ответа программа должна спросить пользователя, требуется ли решить другой пример? Если пользователь введёт «у» программа должна запуститься повторно, иначе — завершиться.

Цикл типа «пока» (оператор while)

Оператор **while** повторяет указанные действия до тех пор, пока его параметр имеет истинное значение.

Например, такой цикл выполнится 4 раза, а на экран будет выведено «1 2 3 4 »:

```
int i = 1;
while (i < 5)
  {
    i++;
    cout << i << " ";
  }
```

Цикл типа «пока» (оператор while)

Такой цикл не выполнится ни разу и на экран ничего не выведется:

```
int i = 1;
  while (i < 0)
    {
        i++;
        cout << i << " ";
    }
```

Цикл типа «пока» (оператор while)

Такой цикл будет выполняться бесконечно, а на экран выведется «1 2 3 4 5 6 7 …»:

```
int i = 1;
while (true)
  {
    i++;
    cout << i << " ";
  }
```

Цикл типа «пока» (оператор while)

Условие, определяющее будет ли цикл повторятся снова, проверяется перед каждым шагом цикла, в том числе перед самым первым.

Таким образом происходит *предпроверка* условия.

Цикл типа «пока» (оператор do...while)

Бывает цикл типа «пока» с постпроверкой условия. Для его записи используется конструкция из операторов do…while.

Такой цикл выполнится 4 раза, а на экран будет выведено «2 3 4 5 »:

```
int i = 1;
do {
    i++;
    cout << i << " ";
} while (i < 5);
```

Цикл типа «пока» (оператор do...while)

Такой цикл выполнится 1 раз, а на экран будет выведено «2 »:

```
int i = 1;
do {
    i++;
    cout << i << " ";
} while (i < 0);
```

Цикл типа «пока» (оператор do...while)

Тело цикла **do...while** выполняется по крайней мере один раз. Этот оператор удобно использовать, когда некоторое действие в программе нужно выполнить по крайней мере единожды, но при некоторых условиях придётся повторять его многократно.

Задачи

- В американской армии считается несчастливым число 13, а в японской — 4. Перед международными учениями штаб российской армии решил исключить номера боевой техники, содержащие числа 4 или 13 (например, 40123, 13313, 12345 или 13040), чтобы не смущать иностранных коллег. Если в распоряжении армии имеется 100 тыс. единиц боевой техники и каждая боевая машина имеет номер от 00001 до 99999, то сколько всего номеров придётся исключить?