Си - простой, изящный язык программирования, на котором останавливает свой выбор все большее число программистов. Основные достоинства языка: это - мобильный язык, мощный и гибкий (например, большая часть популярной ОС UNIX, компиляторы и интерпретаторы языков Фортран, АПЛ, Паскаль, Лисп, Бейсик написаны на Си), удобный, эффективный, обладает рядом замечательных конструкций управления, обычно ассоциируемых с ассемблером.

Алфавит языка

Как и любой другой язык программирования, C++ имеет свой алфавит — набор символов, разрешенных к использованию и воспринимаемых компилятором. В алфавит языка входят:

1. Латинские строчные и прописные буквы:

A,B…Z и a,b…z

2. Цифры от 0 до 9.
3. Символ подчеркивания «_» (код ASCII номер 95). Из этих символов (и только из них!) конструируются **идентификаторы** — имена типов, переменных, констант, процедур, функций и модулей, а также меток переходов. Имя может состоять из любого числа перечисленных выше символов, но должно начинаться с буквы, например:

X Char My_Int_Var C_Dd16_32m

Прописные и строчные буквы различаются: идентификаторы **FILENAME и filename** — это не одно и тоже. Длина имен формально не ограничена, но различаются в них «лишь» первые 63 символа (остальные игнорируются).

4. Символ «пробел» (код 32). Пробел является разделителем в языке

C=2+2; и C = 2 + 2 ;

для компилятора эквивалентны.

5. Символы с кодами ASCII от 0 до 31 (управляющие коды). Они могут участвовать в написании значений символьных и строчных констант. Некоторые из них (7,10,13,8,26) имеют специальный смысл при проведении ряда операции с ними. Символы, замыкающие строку (коды 13 и 10), символ табуляции (код 9) также могут быть разделителями
6. Специальные символы, участвующие в построении инструкций языка:

+ - * / = < > [ ] . , ( ) : ; { } ‘ “ ! % # & |

7. Составные символы, воспринимаемые как один символ:

<= >= == /* */ -> << >> :: && != ||

Из символов алфавита формируются *лексемы языка:*

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

Границы лексем определяются другими лексемами, такими, как разделители или знаки операций.

Символы из расширенного кода ASCII, т.е. символы с номерами от 128 до 255 (а именно в этот диапазон входит алфавит кириллицы на IBM совместимых ПЭВМ), а также некоторые другие из основного набора клавиатуры (^ @ и др.) не входят в алфавит языка.

C++ имеет большое количество зарезервированных (или ключевых) слов. Например **external, file, string, for, while, do, switch,case, typedef, const, goto, constructor, destructor, if, else, inline, virtual, class и др**. Эти слова не могут быть использованы в качестве имен (идентификаторов) в программе.

Объявления **– это способ установления связи объявляемого идентификатора с некоторым типом данных.**

Выражение **состоит из операндов, операции и скобок. Операции находятся между операндами и обозначают действия, которые выполняются над операндами. В качестве операндов выражения можно использовать: переменную, константу, функцию или другое выражение. При вычислении значений выражений следует учитывать, что операции имеют разный приоритет.**

c= a + b*d - sin(x);

```
/*заголовок*/
# include <stdio.h>
void main ()
```

директива препроцессора

Имя функции

```
/*тело функции*/
{  int m;
m=1;
printf (“%d нач. знач \n”,m);
return 0;
}
```

Оператор описания

Оператор присваивания

оператор вызова стандартной функции

Классификация типов данных, объявление типов

Классификация типов связана с их разбиением на *основные* и *производные* типы. Стандарт С89 определяет пять базовых типов данных:

int – *целочисленный тип*, *целое число*;

float – *вещественное число* одинарной точности с плавающей точкой;

double – *вещественное число* двойной точности с плавающей точкой;

char – символьный тип для определения одного символа;

void – тип без значения.

Литерал	Описание	Примеры
Символьный	Одиночный символ, заключенный в апострофы	'W', '&', 'Ф' '7'
Строковый	Последовательность символов, заключенная в обычные (двойные) кавычки	"Это строка \n" "7"
Целый	Десятичный — последовательность цифр, не начинающаяся с нуля	123 7 1999
	Восьмеричный — последовательность цифр от нуля до семерки, начинающаяся с нуля	011, 0177 037777777777
	Шестнадцатеричный — последовательность шестнадцатеричных цифр (0 - 9 и A - F), перед которой стоит 0X или 0x	0X9A, 0xffff 0xFFFFFFFF
Вещественный	Десятичный — [цифры].[цифры]	123.0, 3.14, 0.99 7.0
	Экспоненциальный — [цифры]E|e[+|-] цифры	3e-10, 1.17E6

Тип данных	размер (байт)	диапазон
char	8 бит – 1 байт	-128 ... +127
unsigned char	8 бит – 1 байт	0 ... 255
short int	16 бит – 2 байт	-32768 ... 32767
unsigned short	16 бит – 2 байт	0 ... 65 535
int	32 бит – 4 байт	-32768 ... 32767 Или как Long
unsigned int	32 бит – 4 байт	0 ... 4294967295
long	32 бит – 4 байт	-2 147 483 648 ... 2 147 483 647
unsigned long	32 бит – 4 байт	0 ... 4294967295
float	32 бит – 4 байт	$3.4*10^{-38}$ … $3.4*10^{38}$
double	64 бит – 8 байт	$1.7*10^{-308}$ … $1.7*10^{308}$
long double	80 бит – 10 байт	$3.4*10^{-4932}$…$3.4*10^{4932}$
bool	8 бит – 1 байт	0 или 1; false или true

Производные **типы включают в себя указатели и ссылки на какие-то типы, массивы каких-то типов, типы функций, классы, структуры, объединения. Эти типы считаются производными, поскольку, например, классы, структуры, объединения могут включать в себя объекты различных типов.**

П*орядковые типы,* в которых значения упорядочены и для каждого из них можно указать предшествующее и последующее. К ним относятся целые, символы, перечислимые типы .

Типы данных указываются при объявлении любых переменных и функций. Например:

```
double a = 5.4, b = 2;
int c;
void F1(double A);
```

Пользователь может вводить в программу свои собственные типы. Объявления типов могут делаться в различных местах кода.Место объявления влияет на область видимости или область действия так же, как и в случае объявления переменных.

Синтаксис объявления типа пользователя:

typedef определение_типа идентификатор;

Здесь идентификатор — это вводимое пользователем имя нового типа, а определение_типа — описание этого типа.

Например, оператор

```
typedef   int   Ar[10];
```

объявляет тип пользователя с именем Ar как массив из 10 целых чисел. В дальнейшем на этот тип можно ссылаться при объявлении переменных. Например:

```
Ar   A =  {l, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

Объявление типа с помощью typedef можно использовать и для создания нового типа, имя которого будет являться псевдонимом стандартного типа *C*++. В C++Builder многие встроенные типы компонентов Object Pascal приведены к типам, характерным для C++. Эти переопределения типов содержатся в файле sysdefs.h.

Например:

```
typedef   bool    Boolean;
typedef   int     Integer;
```

Приведение типов

В арифметических выражениях, содержащих элементы различных арифметических типов, в процессе вычислений автоматически осуществляется преобразование типов. Это стандартное преобразование всегда осуществляется по принципу: если операция имеет операнды разных типов, то тип операнда «младшего» типа приводится к типу операнда «старшего» типа. Таким образом, после подобного приведения типов оба операнда оказываются одного типа. И результат применения операции имеет тот же тип.

• Любые операнды типа char, unsigned char или short преобразуются к типу int.

•Любые два операнда становятся либо int, либо float, double или long double:

• Если один из операндов имеет тип long double, то другой преобразуется к типу long double.

• Если один из операндов имеет тип double, то другой преобразуется к типу double.

• Если один из операндов имеет тип float, то другой преобразуется к типу float.

• Иначе, если один из операндов имеет тип unsigned long, то другой преобразуется к типу unsigned long.

• Иначе, если один из операндов имеет тип long, то другой преобразуется к типу long.

• Иначе, если один из операндов имеет тип unsigned, то другой преобразуется к типу unsigned.

• Иначе оба операнда должны иметь тип i nt.

Все это относится к арифметическим операциям, но не относится к операции присваивания. Присваивание сводится к приведению типа результата выражения к типу левого операнда. Если тип левого операнда «младше», чем тип результата выражения, возможна потеря точности или вообще неправильный результат.

```
double a = 5.4,  b = 2;
int c = a * b;          //  c будет = 10
int m = 1,  n = 2; double A = m / n;       // A=0
int  m  =  1; double  n  =  2; double  B  =  m  /  n;
// B = 0.5
```

Неявное автоматическое приведение типов не всегда дает желаемый результат. Это можно исправить, применив операцию **явного приведение типов**.

Она записывается в виде

(тип)

перед той величиной, которую вы хотите привести к указанному типу. Этот способ конвертации типов называется **конвертацией C-style.** Пример, который давал неверное значение переменной A

```
int m = 1,  n = 2; double A = m / n;
```

можно исправить, применив во втором операторе явное приведение типа:

```
double A =  (double) m / n;
```

Или

```
double A =  double (m ) / n;
```

Конвертация C-style не проверяется компилятором во время компиляции, поэтому она может быть неправильно использована, например, при конвертации типов const или изменении типов данных, без учета их диапазонов (что приведет к переполнению).

Есть еще одна ситуация, которая требует явного приведения типов: в некоторых случаях компилятор не может выбрать среди перегруженных функции, если под данный тип параметра подходит несколько из них.

```
TPoint   P;
P.x = 5;    P.y = 1;
Label1->Caption="Координата  x  = "  +  IntToStr(P.x);
```

то получите сообщение компилятора об ошибке: «Ambiguity between '_fastcall Sysutils::IntToStr(int64)' and '_fastcall Sysutils::IntToStr(int)'» (Неоднозначность применения функции IntToStr к параметрам типов int64 и int). Помочь компилятору легко, применив в последнем из приведенных операторов явное указание типа int:

```
Label1->Caption="Координата x=" + IntToStr((int)P.x);
```

```
#include <iostream>
#include <string>
int main()
{
float j = 5.1;
int i= 6;
float a;
 char c='7';
a = i+j;
std::cout << "a=" << a <<"\n";
a = i + (int) j;
std::cout << "integer " << a <<"\n";
  char n='b';
a= a+ (int) n;
  std::cout << "a=" << a << "  n= "<<n<<"\n”;
}
```

Оператор static_cast.

Синтаксис static_cast<тип>(выражение)

```
char c = 97;
std::cout << static_cast<int>(c) << std::endl; // в результате выведется 97, а не 'a'
```

Оператор static_cast лучше всего использовать для конвертации одного фундаментального типа данных в другой:

```
int i1 = 11;
int i2 = 3;
float x = static_cast<float>(i1) / i2;
```

Основным преимуществом оператора static_cast является проверка его выполнения компилятором во время компиляции, что усложняет возможность возникновения непреднамеренных проблем.

Неименованные константы

Константы могут использоваться непосредственно в тексте программы в любых операторах и выражениях. Имеется 4 типа констант: целые, с плавающей запятой, символьные (включая строки) и перечислимые.

Символьные константы должны заключаться в одинарные кавычки. Эти *кон*станты хранятся как **char, signed char** или **unsigned char.**

Строковые константы заключаются в двойные кавычки. Они хранятся как последовательность символов, завершающаяся нулевым символом "0". Пустая строка содержит только нулевой символ.

Если две строковые константы разделены в тексте только пробельным символом, они склеиваются в одну строку. Например:

"Это начало строки, ” "а это ее продолжение"

или

"Это начало строки, ""*а* это ее продолжение"

воспримутся как константа

"Это начало строки, а это ее продолжение"

Перенос длинной строки с одной строчки кода в другую можно делать не только так, как показано выше, но и помещая в конец первой строчки одиночный символ обратного слэша '\'. Например, запись

printf ("Это начало строки, ");

printf("*a* это ее продолжение ");

воспримется как одна строка

"Это начало строки, а это ее продолжение".

В строковой константе можно использовать управляющие символы, предворяемые обратным слэшем. Например, константа

"\"Имя\"\t \tАдрес\n Иванов\t \tМосква" будет при отображении на экране выглядеть так

"Имя" Адрес

Иванов Москва

Кавычки после символа \ воспринимаются как символ кавычек, а не окончание строки. Символы \t и \n означают соответственно табуляцию и перенос строки.

Наиболее часто используемые управляющие символы

\ b — смещение курсора на одну позицию влево;

\ n — переход курсора на новую строку;

\ r — возврат каретки (перевод курсора на начало строки);

\ t — горизонтальная табуляция;

\ v — вертикальная табуляция.

Если в константу должен быть включен обратный слэш "\", то надо поместить подряд два слэша.

Например, строка

"c:\\test\\test.cpp" будет откомпилирована как

"c:\test\test.cpp"

Константы перечислимого типа объявляются следующим образом:

enum имя {значения};

Например, оператор

enum color { red, yellow, green };

объявляет переменную с именем **color,** которая может принимать константные значения **red, yellow** или **green.**

Эти значения в дальнейшем можно использовать как константы для присваивания переменной **color** или для проверки ее значения. Этим константам соответствуют целые значения, определяемые их местом в списке объявления: **red** - 0, **yellow - 1, green** - 2. Эти значения можно изменить, если инициализировать константы явным образом. Например, объявление

enum color { RED, YELLOW = 3, GREEN = RED + 1, GRAY, WHITE };

приведет к тому, что значения констант будут равны: RED - 0, YELLOW – 3, GREEN – 1, GRAY – 2, WHITE – 3.

При этом не обязательно должна соблюдаться уникальность значений. Несколько констант в списке могут иметь одинаковые значения.

В C++Builder имеется ряд предопределенных констант, основные из которых **true** — истина, **false** — ложь, **NULL** — нулевой указатель.

```
#include <iostream>
#include <string>
enum Colors
{
    YELLOW, // присваивается 0
    WHITE, // присваивается 1
    ORANGE=8, // присваивается 8
    GREEN=WHITE+1, // присваивается 2
    RED, // присваивается 3
    GRAY, // присваивается 4
    PURPLE=8,// присваивается 8
    BROWN // присваивается 9
};

int main()
{   Colors paint (RED); int paint1=GRAY;
    std::cout << paint<<  "\t"<<paint1;
    return 0;    }
```

Именованные константы

Именованная константа — это константа, которой присвоен некоторый идентификатор. Объявление именованной константы является указателем для компилятора заменить во всем тексте этот идентификатор значением константы. Такая замена производится только в процессе компиляции и не отражается на исходном тексте.

Именованные константы объявляются так же, как переменные, но с добавлением модификатора **const:**

const тип имя_константы = значение;

Например:

const float Pi = 3.14159;

В качестве значения константы можно указывать и константное выражение, содержащее ранее объявленные константы. Например,

const float Pi2 =2 * Pi; *//удвоенное число π*

const float Kd = Pi/180; *// коэффициент пересчета градусов в радианы*

Для целых констант тип можно не указывать:

const maxint = 12345;

Не забывайте указывать тип для констант, тип которых отличен от int. Например, объявление **const Pi = 3.14159;** присвоит константе Pi значение 3.

Попытка где-то в тексте изменить значение именованной константы приводит к ошибке компиляции с выдачей соответствующего сообщения. Приведем еще примеры объявления именованных констант:

char *const str1= "Привет!";

char const *str2 = "Всем привет!";

Первое объявление вводит константу str1, являющуюся постоянным указателем на строку. Второе объявляет указатель str2 на строковую константу. Этот указатель не является константой. Его в процессе выполнения программы можно изменить так, чтобы он указывал на другую строковую константу. Иначе говоря, оператор

str2 = str1; допустим, а оператор

str1 = str2; вызовет ошибку компиляции.

Объявление переменных

Переменная является идентификатором, обозначающим некоторую область в памяти, в которой хранится значение переменной. Это значение может меняться во время выполнения приложения.

Объявление переменной имеет вид:

тип список_идентификаторов_переменных;

Список идентификаторов может состоять из идентификаторов переменных, разделенных запятыми. Например:

int xl, x2;

Одновременно с объявлением некоторые или все переменные могут быть инициализированы.

тип имя_переменной [= инициализирующее_значение][, ...];

Например: **int xl = 1, x2 = 2; float fl, factor = 3.0, f2;**

Для инициализации можно использовать не только константы, но и произвольные выражения, содержащие объявленные ранее константы и переменные. Например,

int xl = 9, x2 = 2 * xl;

Объявление переменных может быть отдельным оператором или делается это внутри таких операторов, как, например, оператор цикла:

for (int i = 0; i < 10; i++)

Для объявления массива после имени массива указывается число элементов в квадратных скобках.

Наример,

float mas [20];

//массив из 20-ти элементов типа float первый элемент имеет индекс 0.

Так как в Си не существует встроенного строкового типа данных, то для создания переменных строкового типа необходимо создать массив символов. Причем последним элементом этого массива всегда будет нуль-символ '\0'. Он используется для того, чтобы отмечать конец строки. Наличие нуль-символа означает, что количество ячеек массива должно быть по крайней мере на одну больше, чем число символов, которые необходимо размещать в памяти.

Инициализация строк может производиться следующим образом:

```
char  m1[ ]="строка символов";
char m[44]="Только ограничтесь одной строкой";
```

Причем нельзя инициализировать автоматические массивы, необходимо для этого использовать статические или внешние массивы.

Так как массив - это указатель на первый элемент массива, то для определения строковых переменных можно использовать указатели:

```
char *str = "Строка";
```

```
int m=5, k=2;
double A =  (double) m / k;
double  a  =  300,  b  =  200;
short  c  =  a  *  b;
// c=-5536
int p, n;
double pReal = 2.718281828;
p = pReal; // p получает значение 2
pReal = p; // pReal теперь равно 2.0
p = n + (int)(pReal + 0.5); // Округление pReal
```

Инструкция присваивания **является основной вычислительной инструкцией. Если в программе надо выполнить вычисление, то нужно использовать инструкцию присваивания.**

В результате выполнения инструкции присваивания значение переменной меняется, ей присваивается значение.

В общем виде инструкция присваивания выглядит так:

Индентификатор = Выражение;

где:

Индентификатор **- имя переменной, значение которой изменяется в результате выполнения инструкции присваивания;**

= — символ инструкции присваивания.

выражение **— выражение, значение которого присваивается переменной, имя которой указано слева от символа инструкции присваивания.**

Пример

```
#include <iostream.h>
int main()
{
int i = 5;
i = i + 4;
cout << "I = " << i ;
return 0;
}
```

Пример

```
#include <stdio.h>
int main()
{
int i;
printf("Введите целое
число\n");
scanf("%d", &i);
printf("Вы ввели число
%d, спасибо!", i);
return 0;
}
```

```
#include <iostream.h>
int main()
{
int i;
cout << "Введите целое
число \n";
cin >> i;
cout << "Вы ввели число
" << i << ", спасибо!":
return 0;
}
```

Пример . $y = a^b + \sqrt{a^2 + b^2} - 3|b|$

```
#include <stdio.h>
#include <math.h>
int main()
{
double a,b,y;
printf("Введите два числа a и b \n");
scanf("%f  %f", &a,&b);
 y= pow(a,b) + sqrt(a*a+b*b) - 3*fabs(b);
printf("y=   %5.2f", y);
return 0;
}
```

```
#include <stdio.h>
#include <math.h>
int main()
{
double a,b,y;
printf("Введите два числа
a и b \n");
scanf("%f  %f", &a,&b);
 y= pow(a,b) +
sqrt(a*a+b*b) - 3*fabs(b);
printf("y=   %5.2f", y);
return 0;
}
```

```
#include <iostream.h>
#include <math.h>
int main()
{
double a,b,y;
cout << "Введите два
числа a и b \n";
cin >> a>> b;
 y= pow(a,b) +
sqrt(a*a+b*b) - 3*fabs(b);
cout << fixed;
cout.width(5);
cout.precision(2);
cout << "y=  " << y;
return 0;
}
```