Лекция 3 Базовые средства языка СИ++

3.1. Состав языка

В тексте на любом естественном языке можно выделить четыре основных элемента: **символы, слова, словосочетания и предложения**. Алгоритмический язык также содержит такие элементы, только слова называют **лексемами** (элементарными конструкциями), словосочетания – **выражениями**, предложения – **операторами**.

Лексемы образуются из символов, выражения из лексем и символов, операторы из символов выражений и лексем (Рис. 1.1)

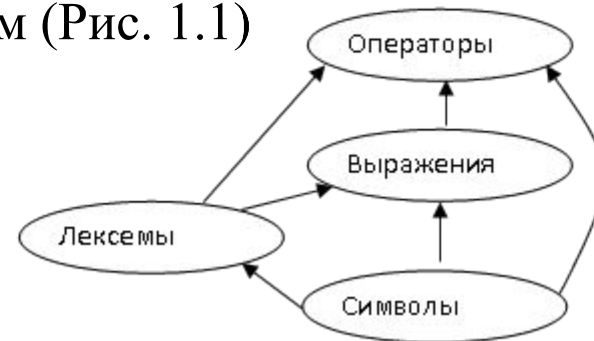


Рис. 1.1. Состав алгоритмического языка

Таким образом, элементами алгоритмического языка являются:

1. Алфавит языка СИ++, который включает

- прописные и строчные латинские буквы и знак подчеркивания;
- арабские цифры от 0 до 9;
- специальные знаки “{},| []()+-/%*.\’:;&?<>=!#^
- пробельные символы (пробел, символ табуляции, символы перехода на новую строку).

2) **Из символов формируются лексемы языка:**

- ***Идентификаторы*** – имена объектов СИ-программ. В идентификаторе могут быть использованы латинские буквы, цифры и знак подчеркивания. Прописные и строчные буквы различаются, например, PROG1, prog1 и Prog1 – три различных идентификатора. Первым символом должна быть буква или знак подчеркивания (но не цифра). Пробелы в идентификаторах не допускаются.
- ***Ключевые*** (зарезервированные) слова – это слова, которые имеют специальное значение для компилятора. Их нельзя использовать в качестве идентификаторов. Полный список служебных слов зависит от реализации языка, т. е. различается для разных компиляторов. Однако существует неименное ядро, которое определено стандартом C++ (табл. 1)
- ***Константы*** – это неизменяемые величины. Существуют целые, вещественные, символьные и строковые константы. Компилятор выделяет константу в качестве лексемы (элементарной конструкции) и относит ее к одному из типов по ее внешнему виду.
- ***Разделители*** – скобки, точка, запятая пробельные символы.
- ***Знаки операций*** – это один или несколько символов, определяющих действие над операндами. Операции делятся на *унарные, бинарные и тернарную* по количеству участвующих в этой операции операндов.

Список служебных слов

Таблица 1

asm	**else**	**private**	**throw**
auto	enum	protected	try
break	extern	public	typedef
case	float	register	typeid
catch	for	return	union
char	friend	short	unsigned
class	goto	signed	virtual
const	if	sizeof	void
continue	inline	static	volatile
default	int	struct	while
delete	long	switch	
do	new	template	
double	operator	this	

3.2. Константы в Си++

Константа – это лексема, представляющая изображение фиксированного числового, строкового или символьного значения.

Константы делятся на 5 групп:

- целые;
- вещественные (с плавающей точкой);
- перечислимые;
- символьные;
- строковые.

Компилятор выделяет лексему и относит ее к той или другой группе, а затем внутри группы к определенному типу по ее форме записи в тексте программы и по числовому значению.

1. Целые константы могут быть десятичными, восьмеричными и шестнадцатеричными. Десятичная константа определяется как последовательность десятичных цифр, начинающаяся не с 0, если это число не 0 (примеры: 8, 0, 192345). Восьмеричная константа – это константа , которая всегда начинается с 0. За 0 следуют восьмеричные цифры (примеры: 016 – десятичное значение 14, 01). Шестнадцатеричные константы – последовательность шестнадцатеричных цифр, которым предшествуют символы 0x или 0X (приме-ры: 0xA, 0X00F).

В зависимости от значения целой константы компилятор по-разному представит ее в памяти компьютера (т. е. компилятор припишет константе соответствующий тип данных).

2. Вещественные константы имеют другую форму внутреннего представления в памяти компьютера. Компилятор распознает такие константы по их виду. Вещественные константы могут иметь две формы представления: **с фиксированной точкой и с плавающей точкой.**

Вид константы с *фиксированной точкой*:[цифры].[цифры] (примеры: 5.7, .0001, 41.).

Вид константы *с плавающей точкой*: [цифры][.][цифры]E|e[+|-][цифры] (примеры:0.5e5, .11e-5, 5E3). В записи вещественных констант может опускаться либо целая, либо дробная части, либо десятичная точка, либо признак экспоненты с показателем степени.

3. Перечислимые константы вводятся с помощью ключевого **слова enum**. Это обычные целые константы, которым приписаны уникальны и удобные для использования обозначения. Примеры: enum { one=1, two=2, three=3,four=4};

enum {zero,one,two,three} – если в определении перечислимых констант опустить знаки = и числовые значения, то значения будут приписываться по умолчанию. При этом самый левый идентификатор получит значение 0, а каждый последующий будет увеличиваться на 1.

enum { ten=10, three=3, four, five, six};

enum {Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday}

4. Символьные константы – это один или два символа, заключенные в апострофы. Символьные константы, состоящие из одного символа, имеют тип **char** и занимают в памяти один байт, символьные константы, состоящие из двух символов, имеют тип **int** и занимают два байта. Последовательности, начинающиеся со знака \ , называются у**правляющими**, они используются:

- Для представления символов, не имеющих графического отображения, например:

  \a – звуковой сигнал,

  \b – возврат на один шаг,

  \n – перевод строки,

  \t – горизонтальная табуляция.
- Для представления символов: \ , ’ , ? , ” (\\, \’ ,\? ,\”).
- Для представления символов с помощью шестнадцатеричных или восьмеричных кодов (\073, \0xF5).

Управляющие символы (или как их ещё называют — escape-последовательность) — символы которые выталкиваются в поток вывода, с целью форматирования вывода или печати некоторых управляющих знаков С++. Основной список управляющих символов языка программирования С++ представлен ниже (см. Таблица 1).

Управляющие символы С++

Таблица 2

Символ	Описание
\r	возврат каретки в начало строки
\n	новая строка
\t	горизонтальная табуляция
\v	вертикальная табуляция
\»	двойные кавычки
\’	апостроф
****	обратный слеш
\0	нулевой символ
\?	знак вопроса
\a	сигнал бипера (спикера) компьютера

Все **управляющие символы**, при использовании, обрамляются двойными кавычками, если необходимо вывести какое-то сообщение, то управляющие символы можно записывать сразу в сообщении, в любом его месте.

Ниже показан код программы, использующей управляющие символы.

```
#include <iostream>
using namespace std;
int main()
{
cout << "\t\tcontrol characters C++"; // две табуляции и печать сообщения
cout << "\rcppstudio.com\n"; // возврат каретки на начало строки и печать сообщения
cout << "\'formatting\' output with \"escape characters\""; // одинарные и двойные
ковычки
cout << "\a\a\a\a\a\a\a\a\a\a\a\a\a\a\a" << endl; //звуковой сигнал биппера
system("pause");
return 0;
}
```

В строке 5 в выходной поток поступают две табуляции \t\t, после чего печатается сообщение control characters C++. В строке 6 управляющий символ \r возвращает каретку в начало строки и печатает сообщение cppstudio.com, причём данное сообщение займет место двух табуляций из строки 5. После этого каретка будет переведена на новую строку, так как в конце сообщения строки 6 стоит символ \n. В строке 7 первое и последнее слова сообщения обрамлены одинарными и двойными кавычками соответственно. В строке 8 в выходной поток сдвигаются управляющие символы \a, эти символы запускают спикер компьютера. Результат работы программы показан ниже:

```
cppstudio.com   control characters C++
'formatting' output with "escape characters"
Для продолжения нажмите любую клавишу .
```

5. Строковая константа – это последовательность символов, заключенная в кавычки. Внутри строк также могут использоваться управляющие символы. Например: “\nНовая строка”,

“\n\”Алгоритмические языки программирования высокого уровня \”” .

3.3. Знаки операций

Знаки операций обеспечивают формирование и последующее вычисление выражений. Один и тот же знак операции может употребляться в различных выражениях и по-разному интерпретироваться в зависимости от контекста. Для изображения операций в большинстве случаев используется несколько символов. **Операнды** — это данные, с которыми работает выражение.

Унарные операции

& операция получения адреса операнда;

* операция обращения по адресу, т.е. раскрытия ссылки, иначе операция разыменования (доступа по адресу к значению того объекта, на который указывает операнд). Операндом должен быть адрес;

- унарный минус - изменяет знак арифметического операнда;

+ унарный плюс (введен для симметрии с унарным минусом); поразрядное инвертирование внутреннего двоичного кода целочисленного аргумента (побитовое отрицание);

~ логическое отрицание (НЕ) значения операнда; применяется к скалярным операндам; целочисленный результат 0 (если операнд ненулевой, т.е. истинный) или 1 (если операнд нулевой, т.е. ложный). В качестве логических значений в языке Си++ используют целые числа: 0 - ложь и не нуль (! 0) - истина. Отрицанием любого ненулевого числа будет 0, а отрицанием нуля будет 1. Таким образом: ! 1 равно 0; ! 2 равно 0; ! (-5) равно 0; ! 0 равно 1;

++ увеличение на единицу (***инкремент*** или авто увеличение): п**рефиксная операция** - увеличение значения операнда на 1 до его использования; **постфиксная операция** - увеличение значения операнда на 1 после его использования.
Операнд не может быть константой либо другим праводопустимым выражением. Записи ++5 или 84++ будут неверными. Операндом не может быть и произвольное выражение. Например, ++(j+k) также неверная запись.

-- уменьшение на единицу (**декремент** или авто уменьшение) -операция, операндом которой не может быть константа и право допустимое выражение:
префиксная операция - уменьшение значения операнда на 1 до его использования;
постфиксная операция - уменьшение значения операнда на 1 после его использования;
Операндом унарных операций ++ и - - должны быть всегда лево допустимые выражения, например, переменные (разных типов);
Операции декремента и инкремента с лёгкостью заменяются арифметическими операциями или операциями присваивания. Но использовать операции инкремента и декремента намного удобнее.
Sizeof - операция вычисления размера (в байтах) для объекта того типа, который имеет операнд. Разрешены два формата операции:
sizeof унарное_выражение sizeof (тип)

Операции инкремента и декремента в С++

Операция	Обозначение	Пример	Краткое пояснение
преинкремент	++	**cout << ++value;**	Значение в переменной value увеличивается после чего оператор cout печатает это значение
предекремент	—	**cout << —value;**	Значение в переменной value уменьшается, после чего оператор cout печатает это значение
постинкремент	++	**cout << value++;**	Оператор cout печатает значение переменной value, затем увеличивает это значение на 1
постдекремент	—	**cout << value—;**	Оператор cout печатает значение переменной value, затем уменьшает это значение на 1

В примерах используется оператор **cout**, чтобы показать, например, в чём различие постфиксных и префиксных операций. Вместо оператора cout можно использовать любой другой, в зависимости от того, какую необходимо выполнить задачу. Разработаем программу на основе выражений из таблицы, которая наглядно покажет, как себя будут вести операции инкремента и декремента.

#Операции инкремента и декремента в C++

```
#include <iostream>
using namespace std;

int main(int argc, char* argv[])
{
int value = 2011;
cout << "value   = " << value << endl; // начальное значение
cout << "++value = " << ++value << endl; // операция преинкремента
cout << "value++ = " << value++ << endl; // операция постинкремента
cout << "value   = " << value << endl; // конечное значение в
переменной value после выполнения операции постинкремента
cout << "--value = " << --value << endl; // операция предекремента
cout << "value-- = " << value-- << endl; // операция постдекремента
cout << "value   = " << value << endl; // конечное значение в
переменной value после выполнения операции постдекремента
system("pause");
return 0;
}
```

1. Д/З – Записать программу и получить результат работы данной программы

Выбрать C:\Users\TEMP\source\repos\Project1\Debug\Project1.exe

```
value   = 2011
++value = 2012
value++ = 2012
value   = 2013
--value = 2012
value-- = 2012
value   = 2011
Для продолжения нажмите любую клавишу . . .
```

```
; // начал
dl; // опе
dl; // опе
; // конеч
dl; // опе
dl; // опе
; // конеч
```

int main(int argc, char* argv[]) // параметры функции main()

Эта строка — заголовок главной функции main(), в скобочках объявлены параметры argc и argv. Так вот, если программу запускать через командную строку, то существует возможность передать какую-либо информацию этой программе, для этого и существуют параметры argc и argv[].
Параметр argc имеет тип данных int, и содержит количество параметров, передаваемых в функцию main. Причем argc всегда не меньше 1, даже когда мы не передаем никакой информации, так как первым параметром считается имя функции.
Параметр argv[] это массив указателей на строки.
Через командную строку можно передать только данные строкового типа.

Указатели и строки — это две большие темы, под которые созданы отдельные разделы.
Так вот именно через параметр argv[] и передается какая-либо информация.

Бинарные операции

Эти операции делятся на следующие группы:

•аддитивные;
•мультипликативные;
•сдвигов;
•поразрядные;
•операции отношений;
•логические;
•присваивания;
•выбора компонента структурированного объекта;
•операции с компонентами классов;
•операция "запятая";
•скобки в качестве операций

Бинарные операции для реализации требуют два операнда.

Аддитивные операции:

\+ бинарный плюс (сложение арифметических операндов или сложение указателя с целочисленным операндом);

\- бинарный минус (вычитание арифметических операндов или указателей).

Мультипликативные операции:

* умножение операндов арифметического типа;

/ деление операндов арифметического типа. Операция стандартна. При целочисленных операндах абсолютное значение результата округляется до целого. Например, 20/3 равно 6, -20/3 равняется -6, (-20) / 3 равно -6, 20/(-3) равно -6;

% получение остатка от деления целочисленных операндов (деление по модулю). При неотрицательных операндах остаток положительный. В противном случае остаток определяется реализацией.

В компиляторах TC++ и BC++:

13%4 равняется 1, (-13)%4 равняется -1, 13 % (-4) равно + 1, а (-13) % (-4) равняется-1

При ненулевом делителе для целочисленных операндов всегда выполняется соотношение: (a/b)*b + a%b равно a.

Побитовые операции выполняются над отдельными разрядами или битами чисел. Данные операции производятся только над целыми числами. (включают операции сдвига и поразрядные операции).

Операции сдвига (определены только для целочисленных операндов).

Формат выражения с операцией сдвига:

операнд_левый операция_сдвига операнд_правый

<< сдвиг влево битового представления значения левого целочисленного операнда на количество разрядов, равное значению правого целочисленного операнда;

>> сдвиг вправо битового представления значения левого целочисленного операнда на количество разрядов, равное значению правого целочисленного операнда.

Применение операций:

```
1int a = 2 << 2;          // 010  на два разрядов влево = 1000 - 8
2int b = 16 >> 3;         // 10000 на три разряда вправо = 10 - 2
```

Поразрядные операции:

& поразрядная конъюнкция (И) битовых представлений значений целочисленных операндов; (Возвращает 1, если оба из соответствующих разрядов обоих чисел равны 1)

| поразрядная дизъюнкция (ИЛИ) битовых представлений значений целочисленных операндов; (Возвращает 1, если хотя бы один из соответствующих разрядов обоих чисел равен 1).

^ поразрядное исключающее ИЛИ битовых представлений значений целочисленных операндов. (Возвращает 1, если хотя бы один из соответствующих разрядов обоих чисел равен 1)

~: поразрядное отрицание или инверсия. Инвертирует все разряды операнда. Если разряд равен 1, то он становится равен 0, а если он равен 0, то он получает значение 1.

Применение операций:

```
1
2 int a = 5 | 2;          // 101 | 010 = 111  - 7
3 int b = 6 & 2;            // 110 & 010 = 10  - 2
4 int c = 5 ^ 2;          // 101 ^ 010 = 111 - 7
```

Следующая программа иллюстрирует особенности **операций сдвига и поразрядных операций.**

```
//операции сдвига и поразрядные операции
#include < iostream.h >
void main ()
{ cout << "\n4<< 2 равняется" << (4<<2);
cout << "\t5>>1 равняется " << (5>>1);
cout << "\n6&5 равняется " << (6&5);
cout << "\t6|5 равняется " << (6 | 5);
cout << "\t6^5 равняется " << (6^5);
}
```

2. Д/З – Записать программу и получить результат работы данной программы

Результат выполнения программы:

4 <<2 равняется 16
5 >>1 равняется 2
6&5 равняется 4
6|5 равняется 7
6^5 равняется 3

Двоичный код для 4 равен 100, для 5 -это 101, для 6 - 110 и т.д. При сдвиге влево на 2 позиции код 100 становится равным 10000 (десятичное значение равно 16).
Остальные результаты операций сдвига и поразрядных операций могут быть прослежены аналогично.
Обратите внимание, что сдвиг влево на n позиций эквивалентен умножению значения на 2n, а сдвиг вправо кода уменьшает соответствующее значение в 2n раз с отбрасыванием дробной части результата. (Поэтому 5>>l равно 2.)

Применение побитовых операторов

Используя побитовые операторы, можно создавать функции, которые позволят уместить 8 значений типа bool в переменную размером 1 байт, что значительно сэкономит потребление памяти. В прошлом такой трюк был очень популярен. Но сегодня, по крайней мере, в прикладном программировании, это не так. Теперь памяти стало существенно больше и программисты обнаружили, что лучше писать код так, чтобы было проще и понятнее его поддерживать, нежели усложнять его ради незначительной экономии памяти. Поэтому спрос на использование побитовых операторов несколько уменьшился, за исключением случаев, когда необходима уж максимальная оптимизация (например, научные программы, которые используют огромное количество данных; игры, где манипуляции с битами могут быть использованы для дополнительной скорости; встроенные программы, где память по-прежнему ограничена).

Операции отношения (сравнения):

< меньше, чем;

> больше, чем;

<= меньше или равно;

>= больше или равно;

== равно;

!= не равно;

Операнды в операциях отношения **арифметического типа или указатели.**

Результат целочисленный: 0 (ложь) или 1 (истина).

Последние две операции (операции сравнения на равенство) имеют более низкий приоритет по сравнению с остальными операциями отношения.

Таким образом, выражение (x < B == A < x) есть 1 тогда и только тогда, когда значение x находится в интервале от A до B. (Вначале вычисляются x < B и A < x, а к результатам применяется операция сравнения на равенство ==.)

Логические бинарные операции:

&& конъюнкция (И) арифметических операндов или отношений. Целочисленный результат 0 (ложь) или 1 (истина);

|| дизъюнкция (ИЛИ) арифметических операндов или отношений. Целочисленный результат 0 (ложь) или 1 (истина).

(Вспомните о существовании унарной операции отрицания ' f '.)

Следующая программа иллюстрирует некоторые особенности операций отношения и логических операций:

```
//операции отношения и логические операции
#include < iostream.h >
void main ()
{ cout << "\n3<5 равняется " << (3<5);
cout << "\t3>5 равняется " << (3>5);
cout << "\3==5 равняется " << (3==5);
cout << " \t3!=5 равняется " << (3!=5);
cout << "\n!=5 || 3==5 равняется " << (3!=5| |3==5);
cout << " \n+4>5 && 3+5>4 && 4+5>3 равняется " << (3+4>5 && 3+5>4 && 4+5>3);
}
```

3. Д/З – Записать программу и получить результат работы данной программы

Результат выполнения программы:

3<5 равняется 1

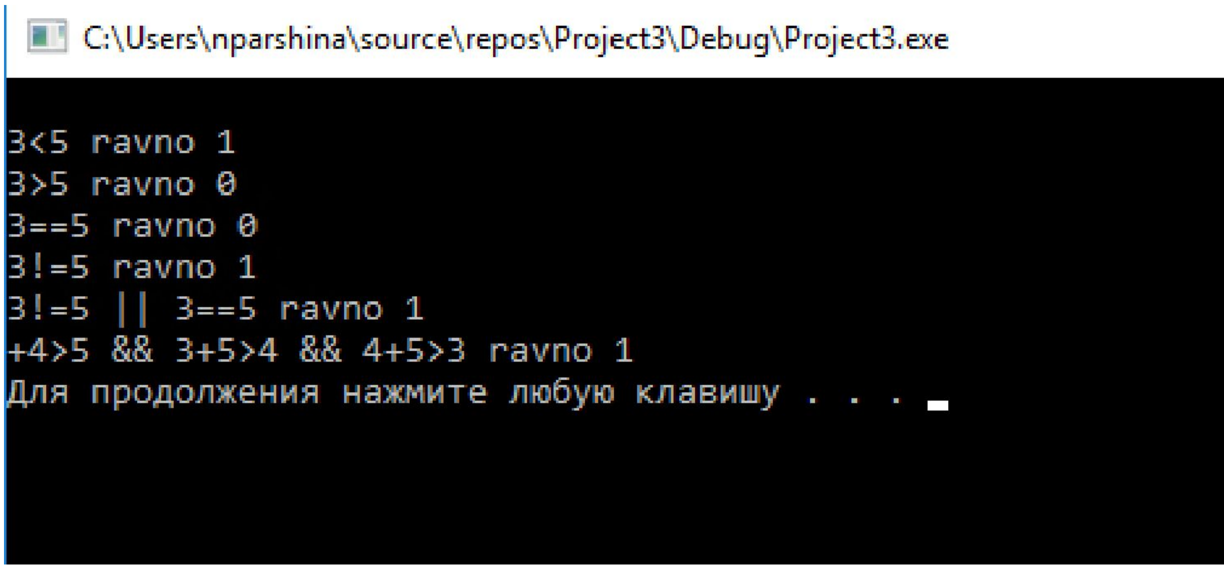
3>5 равняется 0

3==5 равняется 0

3!=5 равняется 1

3!=5 | | 3==5 равняется 1 3+4>5

&& 3+5>4 && 4+5>3 равняется 1



```
#include <iostream>

int main()
{
using namespace std;

cout << "\n3<5 ravno " << (3<5);
cout << "\n3>5 ravno " << (3>5);
cout << "\n3==5 ravno " << (3 == 5);
cout << "\n3!=5 ravno " << (3!= 5);
cout << "\n3!=5 || 3==5 ravno " << (3 != 5||3 == 5);
cout << " \n+4>5 && 3+5>4 && 4+5>3 ravno " << (3 + 4>5 && 3 + 5>4 &&
4 + 5>3) << endl;

system("pause");

return 0;
}
```

Операции присваивания.

В качестве левого операнда в операциях присваивания может использоваться только модифицируемое l-значение - ссылка на некоторую именованную область памяти, значение которой доступно изменениям. Термин l-значение *(left value)*, иначе - лево допустимое выражение, происходит от объяснения действия операции присваивания E = D, в которой операнд E слева от знака операции присваивания может быть только модифицируемым l-значением. Примером модифицируемого l-значения служит имя переменной, которой выделена память и соответствует некоторый класс памяти.

Итак, **перечислим операции присваивания:**

= присвоить значение выражения-операнда из правой части операнду левой части: P = 10.3 - 2*x;

***=** присвоить операнду левой части произведение значений обоих операндов: P *= 2 эквивалентно P = P * 2;

/= присвоить операнду левой части частное от деления значения левого операнда на значение правого: P /= 2.2 - d эквивалентно P = P / (2.2 - d);

%= присвоить операнду левой части остаток от деления целочисленного значения левого операнда на целочисленное значение правого операнда:

N %= 3 эквивалентно N = N % 3;

+= присвоить операнду левой части сумму значений обоих операндов:
A += B эквивалентно A = A + B;

-= присвоить операнду левой части разность значений левого и правого операндов:
x ~= 4.3 - z эквивалентно x = x - (4.3 - z) ;

<<= присвоить целочисленному операнду левой части значение, полученное сдвигом влево его битового представления на количество разрядов, равное значению правого целочисленного операнда: a <<= 4 эквивалентно a = a << 4;

>>= присвоить целочисленному операнду левой части значение, полученное сдвигом вправо его битового представления на количество разрядов, равное значению правого целочисленного операнда: a >>= 4 эквивалентно a = a >> 4;

&= присвоить целочисленному операнду левой части значение, полученное поразрядной конъюнкцией (И) его битового представления с битовым представлением целочисленного операнда правой части: e &= 44 эквивалентно e = e & 44;

| = присвоить целочисленному операнду левой части значение, полученное поразрядной дизъюнкцией (ИЛИ) его битового представления с битовым представлением целочисленного операнда правой части: a | = b эквивалентно a = a | b;

^= присвоить целочисленному операнду левой части значение, полученное применением поразрядной операции исключающего ИЛИ к битовым представлениям значений обоих операндов:
z ^= x + y эквивалентно z = z ^ (x + y).

Для иллюстрации некоторых особенностей выполнения **операций присваивания** рассмотрим следующую программу:

```
//операции присваивания
#include < iostream.h >
void main ()
{ int k ;
cout << "\n k = 35/4 равняется " << (k=35/4);
cout << "\t k /= 1 + 1 + 2 равняется " << (k/=1+1+2);
cout << "\n k *= 5 - 2 равняется " << (k*=5-2);
cout << "\t k %= 3 + 2 равняется " << (k%=3+2);
cout << "\n k += 21/3 равняется " << (k+=21/3);
cout << "\t k -= 6 - 6/2 равняется " << (k-=6-6/2);
cout << "\n k <<= 2 равняется " << (k<<= 2);
cout << "\t k "= 6-5 равняется " << (k"=6-5);
cout << "\n k &= 9 + 4 равняется " << (k&=9+4);
cout << "\t k |= 8 - 2 равняется " << (k|=8-2);
cout << "\n k ^= 10 равняется " << (k^=1O);
}
```

4. Записать программу и получить результат работы данной программы

В первом присваивании обратите внимание на выполнение деления целочисленных операндов, при котором выполняется округление за счет отбрасывания дробной части результата.

Результаты выполнения:

k = 35/4 равняется 8 k /= 1 + 1 + 2 равняется 2

k *= 5 - 2 равняется 6 k %= 3 + 2 равняется 1

k += 21/3 равняется 8 k -= 6 - 6/2 равняется 5

k <<= 2 равняется 20 k <<= 6-5 равняется 10

k &= 9 + 4 равняется 8 k |= 8 - 2 равняется 14

k ^= 10 равняется 4

Условная операция. В отличие от унарных и бинарных операций условная операция используется с тремя операндами.
В изображении условной операции два размещенных не подряд символа **' ? '**, и **' : '** и три операнда-выражения:

выражениие_1 ? выражение_ 2 : выражение_ 3

Первым вычисляется значение выражения_1. Если оно истинно, т.е. не равно нулю, то вычисляется значение выражения_2, которое становится результатом. Если при вычислении выражения_1 получится 0, то в качестве результата берется значение выражения_3. Классический пример:

x < o ? -x : x ;

Выражение возвращает абсолютное значение переменной x.

Условная операция «? :» называется также тернарной операцией

Форма записи тернарной операции в C++

"условие" **?** "выражение 1" **:** "выражение 2"**;**

Если условие истинно, то выполняется выражение 1, иначе (условие ложно) выполняется выражение 2.

Пример:

a > b ? cout << a : cout << b; // если a > b, то выполняется cout << a, иначе выполняется cout << b

Таким образом, если, a > b напечатать a, иначе напечатать b. То есть, программа печатает большее из чисел. Использование условной операции может в некоторых случаях упрощать код, тогда как воспользоваться оператором if else таким же образом не возможно.

Выводы:
Операции могут быть **унарными** – с одним операндом, например, минус;
могут быть **бинарные** – с двумя операндами, например сложение или деление.
В Си++ есть даже одна операция с тремя операндами – **условное выражение**.

Рассмотрим тернарную операцию на примере программы:

```
#include <iostream>
using namespace std;
int main()
 {
int x, y, max;
cin >> x >> y;
(x > y)? cout << x : cout << y << endl; // 1
max = (x > y)? x : y; // 2
cout << max << endl;
return 0;
}
```

Результат работы программы для x = 11 и y = 9:   11 11

Обратите внимание на то, что строки 1 и 2 решают одну и ту же задачу: находят наибольшее значение из двух целых чисел. Но в строке 2 в зависимости от условия (x > y) условная операция записывает в переменную *max* либо x либо y. И после этого значение *max* можно использовать дальше в коде.

5. Записать программу и получить результат работы данной программы

Приоритеты операций в выражениях

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

Грамматика языка Си++ определяет 15 категорий (рангов) приоритетов операций.

Операции одного ранга имеют одинаковый приоритет, и если их в выражении несколько, то они выполняются в соответствии с правилом ассоциативности либо слева направо (->), либо справа налево (<-). Если один и тот же знак операции приведен в таблице дважды, то первое появление (с меньшим по номеру, т.е. старшим по приоритету, рангом) соответствует унарной операции, а второе - бинарной.