ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ

Учебные вопросы

1. Общие сведения о современных технологиях программирования. Функциональное и логическое программирование.
2. Основные понятия языка программирования С.
3. Константы и переменные.
4. Арифметические выражения.
5. Встроенные функции.

1. Общие сведения о современных технологиях программирования. Функциональное и логическое программирование

Программирование для ЭВМ за свою недолгую историю претерпело существенное изменение. Оно начиналось как искусство, однако в результате научных исследований к настоящему времени уже определились некоторые теоретические положения и общие принципы, позволяющие выделить ядро знаний о методологии программирования.

К методологии программирования относится, в частности, круг вопросов, связанных с методами разработки “хороших” программ, обладающих такими важными свойствами, как правильность, надежность, понятность, эффективность и т.п. Подходы к разработке таких программ основаны на методах программирования. Существуют следующие методы программирования:

1. операторное (процедурное) программирование;
2. структурное программирование;
3. модульное программирование;
4. функциональное программирование;
5. логическое программирование;
6. объектно-ориентированное программирование.

К языкам операторного программирования относятся такие языки программирования как, Algol, Fortran, PL/1.

Технологию структурного программирования реализуют языки Pascal, C. Идеи структурного программирования были высказаны Э. Дейкстрой в 1965 году. Понятие структурного программирования включает определенные принципы проектирования, кодирования, тестирования и документирования программ в соответствии с заранее определенной жесткой дисциплиной. Применение методов структурного программирования улучшает ясность и читабельность программ. Программы, написанные с использованием традиционных (операторных) методов, обычно имеют хаотическую структуру, что затрудняет их чтение и понимание. Сколько нужно потратить времени и сил, чтобы разобраться в программе написанной на языке Basic и использующей множество операторов goto и gosub. Структурированные программы можно читать как обычный текст сверху вниз без перерыва, так как они имеют последовательную организацию. Кроме того, структурирование облегчает создание программной документации в связи с тем, что сам текст программы уже несет основную часть информации о решаемой задаче.

К основным методам структурного программирования относятся:

1. отказ от бессистемного употребления оператора GOTO;
2. преимущественное использование структурных операторов;
3. нисходящие проектирование и разработка программ;
4. пошаговая детализация.

Со временем внимание при проектировании программ стало

перемещаться с разработки процедур к организации данных, что было связано с увеличением размера программ и сложности обрабатываемых данных. Это привело к развитию модульного программирования, при котором множество связанных процедур и обрабатываемых ими данных оформлялось в модуль. При модульном программировании акцент делается на разбиение программ на модули таким образом, что данные, обрабатываемые модулем, были спрятаны в нем. К языкам, поддерживающим модульное программирование, можно отнести языки Modula-2, C. Модульное программирование было шагом вперед, так как вело к более структурированным программам.

Под функциональным программированием понимается программирование на языках, в основу которых положено понятие функции. Все вычисления, преобразования и управления программы на таких языках осуществляются с помощью элементарных функций или функций, определяемых программистом в процессе написания программ. Программа в целом, как правило, представляет собой результат суперпозиции некоторых функций и в свою очередь, может быть использована как функция другими программами. К таким языкам программирования относится язык Lisp, C.

Под логическим программированием понимается программирование на языках, в которых задается условие решение задачи и необходимый результат. К таким языкам программирования относится в частности язык Prolog.

Следующим шагом в развитии языков программирования были языки, предоставляющие возможность определения типов данных, которые вели бы себя почти таким же образом, как встроенные типы (языки Ada, Clu, C++). Подобные типы обычно называют абстрактными типами данных, другими словами – типами, определяемыми пользователем. Они объединяют данные и функции, манипулирующие этими данными, в один тип.

Абстракции данных значительно облегчили обработку сложных структур данных. Но при разработке программ, требующих несколько пользовательских типов, которые имеют много общих свойств, оказалось, что часто приходится дублировать большие фрагменты программы. Избежать этого позволили языки объектно-ориентированного программирования. К ним относятся Simula-67, Smalltalk, C++.

Таким образом, объектно-ориентированное программирование явилось результатом эволюции методологий программирования. Оно в большей степени, чем модульное программирование, предоставляет возможность создавать программы, обладающие структурированностью, модульностью и абстракциями данных. Объектно-ориентированный язык программирования характеризуют три основных свойства:

1. инкапсуляция;
2. наследование;
3. полиморфизм.

Инкапсуляция (сокрытие информации) – определение пользователем новых типов данных. Каждый такой тип содержит определение набора

значений и операций, которые могут быть выполнены над этими значениями, и образует так называемый абстрактный тип данных. При этом никакие другие функции, кроме набора операций, не должны иметь доступа к значениям типа.

Например, новый тип – “животные” может содержать значения характеризующие вес, размер, цвет, а также действия, такие как передвижение, размножение, насыщение.

Наследование – это механизм, позволяющий строить иерархию типов. Механизм наследования предполагает определение базового типа (или родительского), а затем использование этого типа для построения производных типов (или потомков). Причем каждый производный тип наследует все свойства базового типа, включая характеристики (данные) и набор операций (функции).

Например, из базового типа животные могут быть порождены типы: “млекопитающие”, “рыбы”, “птицы”. Все эти типы будут содержать характеристики типа “животные” и иметь свои, отличные от других, характеристики: у типа “млекопитающие” – ноги, у типа “рыбы” – плавники, у типа “птицы” – крылья.

Сущность ***полиморфизм***, что в переводе с греческого означает много форм, заключается в обозначении общего действия одним именем (функцией), которое используется во всей иерархии типов. Причем каждый тип в этой иерархии реализует это действие своим собственным, пригодным для него способом. Например, все типы в иерархии “животные” содержат операцию “передвижение”. Но каждый из производных типов выполняет это действие своим способом:

- млекопитающие бегают;
- рыбы плавают;
- птицы летают.

Новизна объектно-ориентированного программирования заключается в том, что оно заставляет отбросить стандартные способы мышления и привычки, которые в течении многих лет использовались в традиционном программировании.

При структурном подходе к программированию каждый новый проект трактуется как новая задача, никогда ранее не рассматривавшаяся, анализ и разработка начинаются с чистого листа, тогда как при использовании объектно-ориентированных методов анализ проекта начинается с попытки трактовать новый проект как расширение. То есть предполагается, что новый проект – это усовершенствование чего-то уже разработанного.

2. Основные понятия языка программирования С

При написании программ в языке С используются следующие понятия:
алфавит,
константы,
идентификаторы,
ключевые слова,
комментарии.

Алфавитом языка называется совокупность символов, используемых в языке. Алфавит языков С/С++ состоит из английских строчных и прописных букв, цифр и специальных знаков.

Очень важно знать и помнить, что языки С/С++ различают прописные и строчные буквы. Языки С/С++, как говорят, является чувствительным к регистру (case sensitive). В языке С/С++ имена COLOR, Color и color определяют три различных имени переменных. При написании программ будьте внимательны к использованию регистров при написании имен переменных.

Borland C++ требует от вас соблюдения следующих правил в отношении идентификаторов:

- Первый символ должен быть буквой или подчеркиванием (_).
- Последующие символы могут быть буквами, цифрами или подчеркиваниями.
- Максимальная длина идентификатора составляет по умолчанию 32 символа (это может быть изменено в опциях компилятора).
- В идентификаторах С++ имеет значение регистр букв. Таким образом, имена rate, RATE и Rate относятся к трем различным идентификаторам.
- Идентификаторами не могут быть зарезервированные слова, например, int, double или static.

Далее следуют примеры допустимых идентификаторов:

```
x x
aString
DAYS_IN_WEEK
BinNumber0
bin_number_0
bin0NumberS
_length
```

А здесь приводятся некоторые из недопустимых:

```
123aNumber
const
NoSpaces Allowed
NorAre*Most+Symbols
```

Используйте описательные имена разумной длины.

Не используйте слишком коротких или слишком длинных имен

идентификаторов. Короткие имена неудобочитаемы, а длинные имена предрасполагают к ошибкам при их написании. Из достойных упоминания исключений можно было бы назвать счетчики, используемые в циклах. В этих случаях часто используются простые i или ix. Для представления временных переменных на скорую руку также часто используется только один символ, как, например, простое s для строки.

В именах переменных можно использовать символ подчеркивания. Обычно с символа подчеркивания начинаются имена системных зарезервированных переменных и констант.

В библиотечных функциях также часто используются имена, начинающиеся с этого символа. Это делается в предположении, что пользователи вряд ли будут применять этот символ в качестве первого символа, Старайтесь не использовать имен, начинающихся с символа подчеркивания, и вам удастся избежать возможных конфликтов и взаимопересечений с множеством библиотечных имен.

Идентификаторы в языке программирования используются для обозначения имен переменных, функций и меток, применяемых в программе. Идентификатором может быть произвольная последовательность латинских букв (прописных и строчных), цифр и символа подчеркивания, которая начинается с буквы или символа подчеркивания. В языке С идентификатор может состоять из произвольного числа символов, однако два идентификатора считаются различными, если у них различаются первые 32 символа. В языке С++ это ограничение снято.

В языках С и С++ некоторые идентификаторы употребляются как служебные слова (keywords), которые имеют специальное значение для компилятора. Их употребление строго определено, и эти слова не могут использоваться иначе. Ключевыми словами стандарта ANSI языка С являются:

auto	**double**	**int**	**struct**
break	**else**	**long**	**switch**
case	**enum**	**register**	**typedef**
char	**extern**	**return**	**union**
const	**float**	**short**	**unsigned**
continue	**for**	**signed**	**void**
default	**goto**	**sizeof**	**volatile**
do	**if**	**static**	**while**

Каждый компилятор может увеличивать количество ключевых слов, так как компилятор учитывает дополнительные возможности того типа компьютеров, для которых он создан. Например, компилятор Borland С++ 3.1 добавляет к ключевым словам стандарта языка С дополнительные слова, предназначенные для работы с памятью и регистрами процессоров семейства Intel, а также позволяющих использовать прерывания и фрагменты программ на другом языке. Эти дополнительные ключевые слова приведены ниже:

asm	**_asm**	**__asm**	**cdecl**
_cdecl	**__cdecl**	**_cs**	**__cs**
_ds	**__ds**	**_es**	**__es**
_export	**__export**	**far**	**_far**
__far	**_fastcall**	**__fastcall**	**huge**
_huge	**__huge**	**interrupt**	**_intrerrupt**
__interrupt	**_loadds**	**__loadds**	**near**
_near	**__near**	**pascal**	**_pascal**
__pascal	**_saveregs**	**__saveregs**	**_seg**
__seg	**_ss**	**__ss**	

Язык C++ в дополнение к ключевым словам языка C добавляет еще несколько:

asm	**catch**	**class**	**friend**	**inline**
operator	**private**	**protected**	**public**	**template**
throw	**try**	**virtual**	**new**	**this**

Программа на C++ состоит из объявлений (переменных, констант, типов, классов, функций) и описания функций. Среди функций всегда имеется главная – main для консольных приложений или WinMain для приложений Windows. Именно эта главная функция выполняется после начала работы программы. Обычно эта функция очень короткая и выполняет только некоторые подготовительные операции, необходимых для начала работы. А далее, при объектно-ориентированном подходе, работа приложения определяется происходящими событиями и реакцией на них объектов.

Как правило, программы стоятся по модульному принципу и состоят из множества модулей. Принцип модульности очень важен для создания надежных и относительно легко модифицируемых и сопровождаемых приложений. Четкое соблюдение принципов модульности в сочетании с принципом скрытия информации позволяет внутри любого модуля проводить какие-то модификации, не затрагивая при этом остальных модулей и головную программу.

Согласно принципам скрытия информации, обычно текст модуля разделяют на заголовочный файл интерфейса, который содержит объявления классов, функций, переменных и т.п., и файл реализации, в котором содержится описание функций. Стандартное расширение файлов реализации – cpp, а заголовочный файлов – h (hpp).

После того, как программа написана, на ее основе должен быть создан выполняемый файл (модуль). Этот процесс осуществляется в несколько этапов.

Сначала работает препроцессор, который преобразует исходный текст. Препроцессор осуществляет преобразования в соответствии со специальными директивами препроцессора, которые размещаются в исходном тексте. Препроцессор может в соответствии с этими директивами включать тексты одних файлов в тексты других, развертывать макросы – сокращенные обозначения различных выражений и выполнять множество

других преобразований.

После окончания работы препроцессора начинает работать компилятор. Его задача – перевести тексты модулей в машинный (объектный) код. В результате для каждого исходного файла .cpp создается объектный файл, имеющий расширение .obj.

После окончания работы компилятора работает компоновщик, который объединяет объектные файлы в единый выполняемый загрузочный модуль, имеющий расширение .exe. Этот модуль можно запускать на выполнение.

Как было отмечено выше, любая программа на C/C++ состоит из одной или нескольких функций. Выполнение программы всегда начинается с функции main(). Это самая главная функция в языке C/C++, без нее программа не будет выполняться. Перед использованием функций в программе, будь то библиотечные, или разработанные программистом, они должны быть объявлены, то есть иметь прототип. Прототипы функций, а также объявление констант и глобальных переменных принято выносить в заголовочные файлы, которые имеют расширение *.h или для C++ *.hpp. Рассмотрим структуру простейшей программы на C/C++ представленной в листинге 1.

```
// Простейшая программа на C++
#include <iostream.h>
int main()
{
	cout << “Я программирую!” << endl;
	return 0;
}
```

Листинг 1.

Первая стока программы начинается с комментария. ***Комментариями*** называются пояснения, помещенные в тексте программы для того, чтобы объяснить или описать некоторые ее части. Транслятор игнорирует комментарии, но программист использует их, чтобы знать, что делает программа, особенно если она не использовалась длительное время, и о ее особенностях забыли. C++ использует символы // для комментария, который продолжается до конца строки. C++поддерживает комментарии языка C, которые начинаются с символов /* и заканчиваются символами */, такие комментарии могут переходить на следующие строки.

Программа на C++ не имеет никаких зарезервированных ключевых слов, которые обозначают ее конец. C++ использует довольно простую схему организации программы. Эта схема поддерживает два уровня кода: глобальный и уровень функций. Кроме того, функция main, определяемая в 3 строке, играет очень специфическую роль, потому что, как было отмечено выше, выполнение программы всегда начинается с этой функции. Следовательно, в программе может быть только одна функция main.

Функция main может располагаться в любом месте программы.

Строки и символы в C++ заключаются соответственно в двойные и одинарные кавычки. Таким образом ‘A’ является одиночным символом, в то время как “A” – строка, состоящая всего из одного символа. Смешивание в C++ односимвольных строк и символов запрещено.

Строки могут содержать любое количество символов, в том числе ни одного. Строка, не имеющая символов, называется пустой строкой.

C++ определяет операторные блоки, ограниченные символами { и }, стоки с 4 по 7.

Каждый оператор в программе на C++ должен заканчиваться точкой с запятой (;). Внимательно изучив листинг 1, вам может показаться, что имеются исключения из этого правила, но это не так. Это мы обсудим позже.

Вторая строка программы содержит директиву препроцессора #include, в которой компилятору Borland C++ дается указание включить в текст программы заголовочный файл IOSTREAM.H. В C++ расширено понятие потоков, которые существовали в языке C. IOSTREAM.H обеспечивает операции, которые поддерживают базовый потоковый ввод/вывод. C++ не имеет встроенных операций ввода/вывода. Вместо этого язык полагается на библиотеки, специализирующиеся в различных типах ввода/вывода.

Директива компилятора – одна из специальных команд компилятора, которые мы рассмотрим при изучении директив препроцессора.

Файл заголовка или заголовочный файл содержит объявление констант, типов данных, переменных и прототипов функций.

Поток – последовательность данных, передающихся из одной части компьютера в другую.

Программа в строке 5 выводит строку “Я программирую!” в стандартный поток вывода cout, при этом программа использует потоковую операцию вывода <<, направляющую выводимую строку в выходной поток. Оператор endl переводит курсор на новую строку.

Функция main должна возвращать значение, которое отражает состояние программы на C++. Возвращаемое значение 0 сообщает операционной системе о том, что программа завершилась без ошибок. Это делается оператором return в 6 строке.

3. Константы и переменные

Для представления логических значений, целых чисел, символов, чисел с плавающей точкой обычной точности, чисел с плавающей точкой двойной точности и незначимых данных Borland C++ предлагает соответственно типы данных bool, int, char, float, double и void. В языке C++ тип void для возвращаемого функцией значения используется для указания на то, что функция не вырабатывает значимого результата, то есть функция действует как процедура.

В языке C++ гибкость в отношении типов данных увеличивается благодаря возможности применения модификаторов типов данных. Модификаторами типа являются: signed, unsigned, short и long. В таблице 3 показаны предопределенные типы данных языка C++, а также их размеры и диапазоны значений. Обратите внимание, что int и unsigned int являются системно-зависимыми.

Модификатор типа данных изменяет точность представления данных и диапазон их значений.

Таблица 1. Предопределенные типы данных C++

Тип данных	Размер	Диапазон	Примеры
bool	1	false и true	false, true
char	1	от -128 до 127	‘A’, ‘!”
wchar_t	2	от -32768 до 32767	L‘A’, L‘!’
signed char	1	от -128 до 127	23
unsigned char	1	от 0 до 255	200, 0x1A
int	2 или 4	Зависит от платформы: от -32768 до 32767 для 16-бит от -2147483647 до 2147483647 для 32-бит	3000
unsigned int	2 или 4	Зависит от платформы: от 0 до 65535 для 16-бит от 0 до 4294967295 для 32-бит	0xFFFF, 65535
short int	2	от -32768 до 32768	100
unsigned short int	2	от 0 до 65535	0xFF, 40000
long int	4	от -2147483648L до 2147483647L	0xFFFFL, -123456L
unsigned long int	4	от 0L до 4294967295L	123456L
float	4	от 3.4E-38 до 3.48E+38 и от -3.4E-38 до -3.4E+38	2.35, -52.35, 1.3e+10
double	8	от 1.7E-308 до 1.7E+308 и от -1.7E_308 до 1.7E+308	12.354, -2.5e+100
long double	10	от 3.4E-4932 до 3.4E+4932 и от -1.1E-4932 до -3.4E+4932	8.5e-30000

В языке C++поддерживаются шестнадцатеричные числа. Такие числа начинаются с символов 0x, за которыми следует шестнадцатеричное значение, например, число 0xff (0xFF) является шестнадцатеричным эквивалентом десятичного числа 255.

Операторы языка C и C++ манипулируют переменными и константами в виде выражений. Имена, которые используются для обозначения

переменных, функций, меток и других определенных пользователем объектов называются идентификаторами (identifies). Идентификатор может содержать латинские буквы, цифры и символы подчеркивания, и начинаться должен только с буквы или символа подчеркивания. В стандарте ANSI языка, идентификатор определяется своими первыми 32 символами. Строчные и прописные буквы рассматриваются как различные символы. Идентификатор не должен совпадать с ключевыми словами языка С и С++.

Многие языки, такие, как BASIC (в его последних реализациях), Modula-2, Ada, C, Pascal и C++, поддерживают константы. Невозможно отрицать, что константы повышают удобочитаемость программы, заменяя числовые константы описательными идентификаторами. Более того, использование констант дает возможность изменять значение параметра посредством простого изменения этого параметра в одном месте программы. Этот подход более удобен и менее подвержен ошибкам, которые могут появляться, при замене числовых данных.

В языке С и С++ константы представляют фиксированную величину, которая не может быть изменена в программе. Константы могут быть любого базового типа.

Константы являются идентификаторами, ассоциированными с фиксированными значениями. С++ предоставляет два варианта констант: константы, определенные с помощью макросов, и формальные константы. Определенные с помощью макросов константы унаследованы из языка С и используют директиву компилятора #define.

Общий синтаксис для задания формальной константы:

```
const тип_данных имя_константы = значение константы;
```

Элемент тип_данных является необязательным и определяет тип данных значения константы.

Пример:

```
const unsigned char ASCII_A = 65;
const int DAY_IN_WEEK = 7;
const char FIRST_DISK_DRIVE = 'A';
```

Используйте имена констант, набранные в верхнем регистре. Этот стиль присвоения имен дает вам возможность быстро определять, является ли идентификатор константой. Не думайте, что те, кто будет читать ваш код, будут знать о смысле включенных в него чисел. Раз уж на то пошло, не думайте, что вы сами будете помнить о смысле этих чисел, когда будете читать этот код снова через несколько месяцев. Используйте объявление констант для повышения удобочитаемости ваших программ.

Рассмотрим пример, в котором используются константы, определенные с помощью макросов. В листинге 2 показан исходный текст программы CONST1.CPP. Программа запрашивает ввод текущего времени в часах (24 часовой формат), минутах и секундах, и выдает количество секунд, прошедших с полуночи.

```
//Программа C++, иллюстрирующая использование констант в стиле C
#include <iostream.h>
#define SEC_IN_MIN 60
#define MIN_IN_HOUR 60
int main()
{
        long hours, minutes, seconds;
        long totalSec;
        cout << “Введите часы: ”;
        cin >> hours;
        cout << “Введите минуты: ”;
        cin >> minutes;
        cout << “Введите секунды: ”;
        cin >> seconds;
        totalSec = ((hours * MIN_IN_HOUR + minutes) * SEC_IN_MIN) +
seconds;
        cout << endl << totalSec << “ прошло с полуночи”;
        return 0;
}
```

Листинг 2. Исходный текст программы CONST1.CPP

Программа использует директиву #include для включения заголовочного файла IOSTREAM.H, в котором содержаться объявления и определения, необходимые для ввода и вывода (ввода-вывода) информации. В следующих двух строках содержаться директивы #define, в которых объявляются определенные с помощью макросов константы SEC_IN_MIN и MIN_IN_HOUR. Обе константы имеют одинаковое значение 60, но смысл этих значений различен. В функции main, которая начинается со следующей строки, объявлены четыре переменных типа long: hours, minutes, seconds и totalSec.

Функция main использует пары операторов для отображения на экране запроса и для приема введенной информации. Строка 9 содержит потоковый оператор вывода, который запрашивает количество часов. Идентификатор cout является именем стандартного потока вывода (как правило, экран монитора) и использует операцию << для вывода запроса. Строка 10 содержит потоковый оператор ввода. Идентификатор cin является именем стандартного входного потока, который использует операцию >> для считывания данных с клавиатуры и сохранения их в переменной hours. Операторы ввода-вывода в следующих строках выполняют аналогичную задачу, связанную с запросом входных данных и получения ввода с клавиатуры.

В строках 15 и 16 содержится оператор, который вычисляет общее число секунд, прошедших с полуночи, и сохраняет результат в переменной totalSec. Оператор использует определенные с помощью макросов константы

MIN_IN_HOUR и SEC_IN_MIN. Использование этих констант делает оператор более понятным по сравнению со случаем использования вместо обеих констант числа 60. В следующей строке содержится потоковый оператор вывода, который отображает на экране общее число секунд, прошедших с полуночи (хранимое в переменной totalSec), за которым следует поясняющий текст.

В листинге 3 показан исходный текст программы CONST2.CPP.

```
//Программа C++, константы в стиле C++
#include <iostream.h>
const int SEC_IN_MIN = 60;              //Глобальная константа
int main()
{
      const int MIN_IN_HOUR = 60;      //Локальная константа
      long hours, minutes, seconds;
      long totalSec;
      cout << “Введите часы: ”;
      cin >> hours;
      cout << “Введите минуты: ”;
      cin >> minutes;
      cout << “Введите секунды: ”;
      cin >> seconds;
      totalSec = ((hours * MIN_IN_HOUR + minutes) * SEC_IN_MIN) +
seconds;
      cout << endl << totalSec << “ прошло с полуночи”;
      return 0;
}
```

Листинг 3. Исходный текст программы CONST2.CPP

Программы, приведенные в листингах 2 и 3 похожи. Различие между ними состоит в способе объявления констант. В листинге 3 для объявления констант используется формальный синтаксис языка C++. Кроме того, константа SEC_IN_MIN объявлена вне функции main. Такой тип объявления делает константу глобальной; это значит, что если бы в программе имелась другая функция, в ней можно было бы использовать константу SEC_IN_MIN. В противоположность этому, константа MIN_IN_HOUR объявлена внутри функции main, что делает эту константу локальной для функции main.

Объявление переменных

Объявление переменных требует от вас определения типа и имени этой переменной. Слово “переменная” указывает на то, что вы можете изменять содержимое этого контейнера данных.

Переменные представляют собой идентификаторы, используемые для сохранения и последующего извлечения информации.

Чтобы лучше понять концепцию переменных, вообразите, что имеется

ряд ящиков, в каждом из которых может содержаться только один, определенный тип объектов. Эти объекты называются значениями типа int, char и так далее. Вы можете иметь любое количество ящиков любого типа, но у вас должно быть по одному ящику для каждого объекта, который вы хотите хранить – вы не можете хранить объект без соответствующего ящика. Так, например, если вы хотите хранить значение типа int, вам потребуется переменная соответствующего типа, в котором вы могли бы хранить это число. Затем, когда в последствии вы захотите использовать это число, вам достаточно будет только обратиться к соответствующему ящику и узнать, что в нем содержится.

Общий синтаксис объявления переменных имеет вид:

тип имяПеременной;

тип имяПеременной = начальноеЗначение;

тип перем1 [= нач_знач1], перем2 [= нач_знач2], …;

Пример:

```
int i;
double z = 32.314;
double y = 2 * x, z = 4.5, a = 45.7;
```

Инициализирующие значения могут содержать значения других переменных, которые были определены раньше.

Не забывайте использовать описательные имена для ваших переменных. Если вы постоянно используете односимвольные имена для ваших переменных, то, вернувшись когда-нибудь снова к вашему коду, вы испытаете немалые трудности, пытаясь понять, что же это вы там имели в виду.

Не объявляйте переменных внутри одного и того же программного модуля с именами, различающимися только регистром, в котором набраны символы, например rate и Rate. Это может создать ужасную путаницу, если вам впоследствии придется вернуться к вашему коду, и граничит с садизмом по отношению к тем, кому придется читать ваш код.

4. Арифметические выражения

Операции и выражения

Манипулирование данными сопряжено с построением выражений из операндов и операций. Язык C++ поддерживает различные виды операций и выражений.

Операции являются специальными символами, при помощи которых из значений операндов получается новое значение.

Арифметические операции

В таблице 4 показаны арифметические операции языка C++. Они используются в арифметических выражениях, например,

z = y / x.

Переменные y и x являются операндами, а знак / операцией. Результат выражения зависит от типа данных операндов и сохраняется в z. В зависимости от типа операндов компилятор выполняет деление с плавающей точкой или целочисленное деление, причем, если оба операнда имеют целочисленный тип – будет выполнено целочисленное деление, если один или оба операнда являются выражениями с плавающей точкой, то компилятором будет генерироваться код для деления с плавающей точкой.

Таблица 2. Арифметические операции языка C++

Оператор C++	Назначение	Тип данных	Пример
+	Унарный плюс	Числа	**x = +y + 3;**
-	Унарный минус	Числа	**x = -y;**
+	Сложение	Числа	**z = y + x;**
-	Вычитание	Числа	**z = y – x;**
*	Умножение	Числа	**z = y * x;**
/	Деление	Числа	**z = y / x;**
%	Деление по модулю	Целые числа	**z = y % x;**

Арифметические выражения

Наиболее простыми типами выражений являются выражения, которые содержат литералы, как, например,

-12

34.45

‘A’

“Краснодар”.

Арифметическое выражение является частью оператора программы, в которой содержится значение.

Литеральные константы –12 и 34.45 являются простыми арифметическими выражениями. Следующий уровень арифметических выражений включает отдельные переменные или константы, например,

```
DAY_IN_WEEK // константа
i
x
```

Еще один уровень арифметических выражений содержит отдельные операции, использующие в качестве операндов числа, константы и переменные

```
355 / 113
4 * i
45.67 + x
```

Наиболее сложные арифметические выражения содержат несколько операций, круглые скобки и даже функций, как, например

```
(355 / 113) * square(radius)
PI * square(radius)
((2 * x – 3) * x + 2) * x – 5
(1 + x) / (3 – x)
```

Операции инкремента и декремента (изменения на 1)

Язык C++ поддерживает специальные операции инкремента (увеличения на 1) и декремента (уменьшения на 1).

Операции инкремента (++) и декремента (--) соответственно увеличивают или уменьшают на 1 хранимое в переменной значение.

Общий синтаксис и пример операций инкремента и декремента приведен в таблице 5.

Общий синтаксис показывает, что имеются два способа применения операций ++ и - -. Помещение этих операций слева от операнда приводит к изменению значения операнда, прежде чем значение операнда будет использовано в выражении. Подобным образом, помещение этих операций справа от операнда приводит к изменению значения операнда после того, как его значение используется в выражении.

Таблица 3. Синтаксис операций инкремента и декремента

Синтаксис	Описание	Пример
переменная++	Пост-инкремент	lineNumber++
++переменная	Пред-инкремент	++index
переменная--	Пост-декремент	lineNumber--
--переменная	Пред-декремент	--index

Если операция ++ или – является единственным знаком операции в операторе программы, то практически нет разницы между использованием его префиксной или постфиксной формы.

Следующий код иллюстрирует использование операций ++ и --:

```
int n, m, t = 5;
t++;          // t теперь равно 6, тот же результат, что и ++t
--t;          // t теперь равно 5, тот же результат, что и t--
n = 4 * t++;// t теперь равно 6, а n равно 20
t = 5;
```

```
m = 4 * ++t;        // t теперь равно 6, а m равно 24
```

Во второй строке для увеличения значения переменной t используется операция инкремента ++ в постфиксной форме. Если вместо этого записать оператор ++t, то после его выполнения получится тот же результат. В третьей строке использована операция декремента в префиксной форме. Если ее заменить на постфиксную форму, получим тот же самый результат. В четвертой строке используется постфиксная операций инкремента в простом арифметическом выражении. Здесь 4 умножается на текущее значение переменной t (равное 5), результат 20 присваивается переменной n и затем происходит увеличение переменной t до 6. выполнение двух последних строк дает результат отличный от предыдущего. Оператор сначала увеличивает значение переменной t (значение переменной t становится равным 6), затем выполняет умножение и, наконец, присваивает результат 24 переменной m.

Операции присваивания

Язык C++ предусматривает специальные операции присваивания, которые объединяются с простыми математическими операциями. Они дают возможность использовать сокращенную запись для воздействия на отдельный операнд. В таблице 6 показаны арифметические операции присваивания.

Таблица 4. Арифметические операции присваивания

Операция присваивания	Длинная форма	Пример
x += y	**x = x + y**	**x += 12;**
x -= y	**x = x – y**	**x -= 24 + y;**
x *= y	**x = x *y**	**scale *= 10;**
x /= y	**x = x / y**	**z /= 34 * y;**
x %= y	**x = x % y**	**z %= 2;**

Операция sizeof

Часто программам требуется знать размер в байтах для типа данных или переменной. В языке C++ предусмотрена операция sizeof, которая принимает в качестве параметра либо тип данных, либо имя переменной (скаляр, массив, структуру и т.д.). Синтаксис операции sizeof представлен в таблице 7.

Операция sizeof обычно используется для определения размера структуры и класса до того, как они будут записаны в файл. Операция sizeof просуммирует все компоненты рассматриваемого объекта и возвратит его полный размер. Эта операция может, однако, использоваться и в случае простых переменных и типов данных.

Таблица 5. Синтаксис операции sizeof

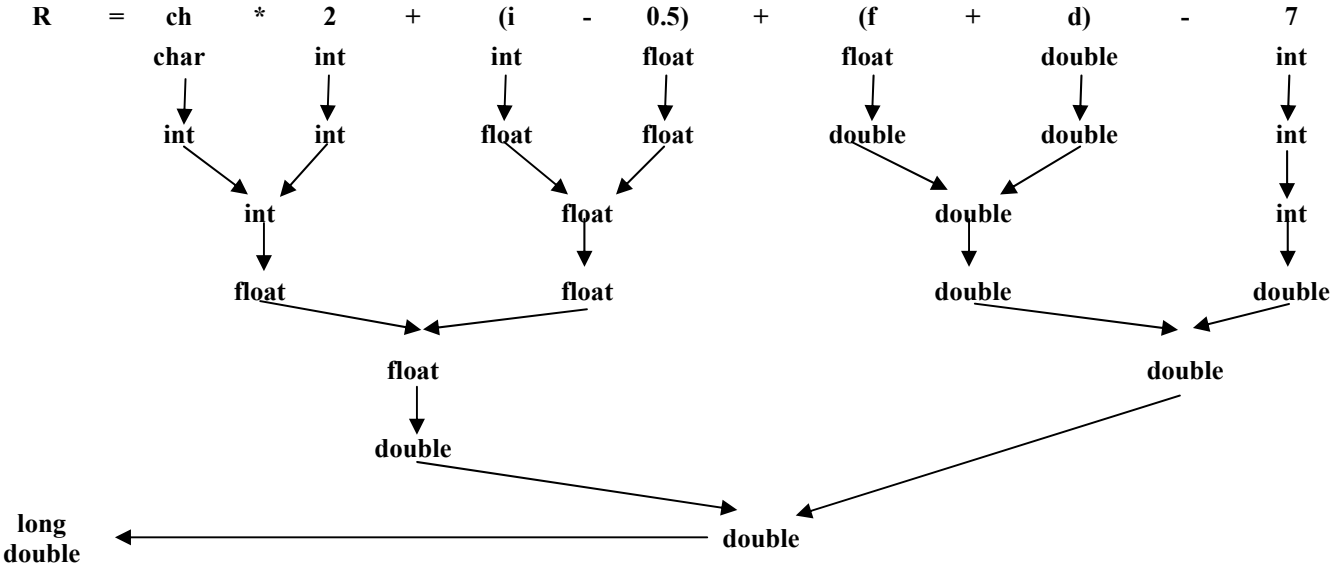
Общий синтаксис операции	Пример
sizeof({имя_переменной | тип_данных})	int sizeDifferent = sizeof(double) – sizeof(float);
sizeof{имя_переменной | тип_данных}	int intSize = sizeof int;

Используйте операцию sizeof с именем переменной, а не с типом данных. Этот подход является безопасным, так как если вы измените тип этой переменной, то операция sizeof по-прежнему возвратит правильный ответ. В противоположность этому, если вы примените операция sizeof к типу данных переменной и позже измените ее тип (и при этом не модифицируете параметр операции sizeof), Вы внесете в программу ошибку.

Не используйте числовые константы для представления размера переменной. Этот подход часто служит причиной возникновения ошибок.

Приведение типа

Одной из обязанностей компилятора является автоматическое преобразование значения из одного типа данных в другой, совместимый с ним. Такое преобразование данных упрощает выражения и облегчает работу как программистам-ветеранам, так и новичкам. С преобразованием данных, происходящим за сценой (не явное преобразование типов), отпадает необходимость в исследовании в вашей программе каждого выражения, в котором происходит смешение совместимых типов данных. Например, компилятор обрабатывает большинство выражений, в которых происходит смешение различных типов целых типов или целых типов чисел и чисел с плавающей точкой. Если Вы попытаетесь сделать что-нибудь запрещенное, Вы получите ошибку времени компиляции. Следующая схема иллюстрирует неявное (автоматическое) преобразование типов.



Приведение типа является свойством языка, которое дает возможность явно определять, каким образом некоторое значение будет преобразовано из первоначального типа данных в совместимый с ним тип.

Таким образом, приведение типа дает компилятору указание, чтобы он выполнял именно то преобразование, которое желаете Вы, а не то, которое он считает необходимым. В таблице 8 представлен синтаксис и пример приведения типов.

Таблица 6. Синтаксис приведения типов

Общий синтаксис	Пример

приведение_к_типу (выражение)	int i = 2;
	float a, b;
	a = float(i);
	b = (float)i;

(приведение_к_типу) выражение	

Операции отношений и логические операции

В таблице 9 показаны операции отношений и логические операции языка C++. В языке C++ не поддерживается логическая операция XOR (исключающее ИЛИ).

Таблица 7. Операции отношений и логические операции языка C++

Операция C++	Значение	Пример
&&	Логическое И (AND)	if (i > 1 && i < 10)
||	Логическое ИЛИ (OR)	if (c = = 0 || c = = 9)
!	Логическое НЕ (NOT)	if (! (c > 1 && c < 9))
<	Меньше чем	if (i < 0)
<=	Меньше или равно	if (i <= 0)
>	Больше чем	if (j > 0)
>=	Больше или равно	if (x >= 8.2)
= =	Равно	if (c = = ‘\0’)
!=	Не равно	if (c != ‘\n’)
?:	Условное присваивание	k = (i<1) ? 1:i;

Условное выражение является сокращенной формой простого оператора if-else с двумя альтернативами.

Операции отношений (меньше чем, больше чем и равно) и логические операции (И, ИЛИ, НЕ) являются базовыми строительными блоками в конструкциях принятия решений в любом языке программирования.

Не используйте символ = в качестве операции отношения равенства. Этот распространенный недосмотр является источником логических ошибок в программе на C++.

Булевы выражения

Часто для того, чтобы сформулировать нетривиальное условие, необходимо использовать совокупность операций отношений и логических операций, например:

```
x < 0 || x > 11
(i != 0 || i > 100) && (j != i || j > 0)
x != 0 && x != 10 && x != 100
```

Булевы выражения (также называемые логическими) – это выражения, которые содержат логические операции и/или операции отношений,

результат которых имеет тип bool.

Результатами всех операций отношения являются выражения типа bool.

Необходимо делать тщательную проверку, чтобы избежать булевых выражений, которые либо всегда являются истинными, либо всегда ложными. Например, выражение (x < 0 && x > 10) всегда является ложным, поскольку никакое значение x не может быть одновременно отрицательным и большим 10. Не используйте операцию присваивания = для проверки на равенство. Это распространенная ошибка начинающих программистов на C++!

Операции манипулирования битами

C++ является языком программирования, который подходит для системных разработок. Системные разработки требуют операций манипулирования битами. В таблицах 10 и 11 представлены операции манипулирования битами.

Операции манипулирования битами переключают, устанавливают, опрашивают и сдвигают биты любого числового типа, исключая типы с плавающей точкой (float и double).

Таблица 8. Операции манипулирования битами языка C++

Операция C++	Смысл	Пример
&	Побитовое И (AND)	i & 128
|	Побитовое ИЛИ (OR)	j | 64
^	Побитовое ИЛИ НЕ (XOR)	j ^ 12
~	Побитовое НЕ (NOT)	~w
<<	Побитовый сдвиг влево	i << 2
>>	Побитовый сдвиг вправо	j >> 4

Таблица 9. Операции манипулирования битами с присваиванием

Операция C++	Длинная форма	Пример
x &= y	x = x & y	i &= 128
x |= y	x = x | y	j |= 64
x ^= y	x = x ^ y	k ^= 15
x <<= y	x = x << y	i <<= 2
x >>= y	x = x >> y	j >>= 4

Операция-запятая

Операция-запятая требует, чтобы программа полностью завершила оценку первого выражения перед тем, как начать оценку второго выражения. Операция-запятая, выполняя свою особую роль, служит специфической и очень важной цели в организации цикла for.

Операция-запятая дает возможность создавать набор выражений, инициализирующих несколько переменных цикла, например:

```
for(i = 0, j = 0; i < 10; i++, j++)
```

Условное выражение

Операция условия (условное выражение) – единственная операция языка С, имеющая три операнда (тернарная). Общий синтаксис операции условия имеет вид:

(Exp_1) ? (Exp_2) : (Exp_3);

Вычисляется выражение Exp_1. Если это выражение имеет ненулевое значение (истина), то вычисляется выражение Exp_2. результатом операции будет значение выражения Exp_2.

Если значение выражения Exp_1 равно нулю, то вычисляется выражение Exp_3 и его значение будет результатом операции. В любом случае вычисляется только одно из выражений: Exp_2 и Exp_3.

Например, операцию условное выражение удобно применять для нахождения наибольшего из двух чисел:

double max = (x > y) ? x : y;

или для нахождения абсолютной величины числа:

double absolute = (x > 0) ? x : -x;

Если второй и третий операнды являются величинами типа lvalue, т.е. могут стоять в левой части операции присваивания, то результат операции является величиной lvalue.

Следующий код присваивает переменной имеющей наибольшее значение, значение равное единице:

(x > y) ? x : y = 1;

Приоритет операций и последовательность оценки

Таблица 10. Операции языка C++ и их приоритет

Категория	Название	Символ	Посл. оценки	Приоритет
Одноместные				
	Пост-инкремент	++	Слева направо	2
	Пост-декремент	--	Слева направо	2
	Адрес	&	Справа налево	2
	Побитовое NOT	~	Справа налево	2
	Приведение типа	(тип)	Справа налево	2
	Логическое NOT	!	Справа налево	2
	Отрицание	-	Справа налево	2
	Знак плюс	+	Справа налево	2
	Пред-инкремент	++	Справа налево	2
	Пост-декремент	--	Справа налево	2
	Размер данных	sizeof	Справа налево	2
Мультипликативные				
	Деление по модулю	%	Слева направо	3

	Умножение	*	Слева направо	3
	Деление	/	Слева направо	3
Аддитивные				
	Сложение	+	Слева направо	4
	Вычитание	-	Слева направо	4
Побитовые сдвиги				
	Сдвиг влево	<<	Слева направо	5
	Сдвиг вправо	>>	Слева направо	5
Отношения				
	Меньше чем	<	Слева направо	6
	Меньше или равно	<=	Слева направо	6
	Больше чем	>	Слева направо	6
	Больше или равно	>=	Слева направо	6
	Равно	= =	Слева направо	7
	Не равно	!=	Слева направо	7
Побитовые				
	AND	&	Слева направо	8
	XOR	^	Слева направо	9
	OR	|	Слева направо	10
Логические				
	AND	&&	Слева направо	11
	OR	||	Слева направо	12
Тернарная				
	Условное выражение	?:	Справа налево	13
Присваивания				
	Арифметическое	=	Справа налево	14
		+=	Справа налево	14
		-=	Справа налево	14
		*=	Справа налево	14
		/=	Справа налево	14
		%=	Справа налево	14
	Сдвига	>>=	Справа налево	14
		<<=	Справа налево	14
	Побитовое	&=	Справа налево	14
		|=	Справа налево	14
		^=	Справа налево	14
	Запятая	,	Справа налево	15

5. Встроенные функции

Программу, как и любой объект, можно компоновать из элементов различной сложности. Можно создавать ее из отдельных инструкций. В этом случае каждую новую программу придется писать заново в полном объеме, что неэффективно. Поэтому чаще программы разрабатывают с использованием законченных программных модулей, состоящих из многих инструкций и ориентированных на решение некоторых локальных задач. Это позволяет существенно упростить и сократить время разработки программ. Такие законченные модули называются функциями или подпрограммами. Обращаться к ним можно многократно по имени, передавая исходные данные и получая результаты вычислений.

Программы на языке С обычно состоят из нескольких отдельных функций, которые имеют небольшие размеры и могут находиться как в одном, так и в нескольких файлах.

В языке С имеется обширная библиотека встроенных функций. Некоторые из них были рассмотрены ранее. Это функции ввода-вывода для стандартных файлов и функции для работы с файлами данных. Другие библиотечные функции будут рассмотрены в следующих вопросах. Это функции взаимодействия с операционной системой, графические функции языка, специальные функции.

Все библиотечные функции разработаны заранее, помещены в особые файлы с расширением .LIB и автоматически включаются системой программирования. Для вызова любой такой функции в программе указывается ее имя и список параметров в круглых скобках. Если функция возвращает нецелое значение, нужно объявить его тип. Для некоторых функций такие объявления помещены в стандартные файлы, например для функций ввода-вывода - scanf, printf, fscanf, fprintf - в файл stdio.h.

Рассмотрим библиотечные математические функции языка С++. Их список и назначение даны в таблице 13. В первой графе таблицы представлены имена функций, во второй - их прототипы, объявленные в файле заголовков, в третьей графе – действия, выполняемые функциями. Большинство математических функций отличаются от аналогичных функций языков турбо Паскаль и Бейсик только именами и не требуют пояснений. Новыми, ранее не изучавшимися, являются функции cabs, sinh, cash, fanh, hypot, ldexp и poly. Эти функции представляют дополнительные возможности при программировании арифметических выражений. Функция cabs вычисляет абсолютное значение комплексного числа. Функции sinh, cash и tanh вычисляют значения гиперболических синуса, косинуса и тангенса соответственно. Вычисления осуществляются по формулам:

$$sh\,x = \frac{e^{x} - e^{-x}}{2},\ ch\,x = \frac{e^{x} + e^{-x}}{2},\ th\,x = \frac{sh\,x}{ch\,x} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Функция hypot вычисляет гипотенузу прямоугольного треугольника. В качестве ее аргументов указывается длина катетов треугольника.

Функция ldexp вычисляет значение выражения $\sqrt{2^e}$.

Функция poly предоставляет возможность работы с многочленами.

Таблица 11. Математические функции языка C++

Функция	Прототип	Выполняемые действия
abs	int abs (int i);	Возвращает абсолютное значение целого аргумента i
acos	double acos (double x);	Функция арккосинуса. Значение аргумента должно находиться в диапазоне от -1 до 1.
asin	double asin (double x);	Функция арксинуса. Значение аргумента должно находиться в диапазоне от -1 до 1.
atan	double atan (double x);	Функция арктангенса
atan2	double atan2(double y, double x);	Функция арктангенса y / x
cabs	double cabs (struct complex znum);	Вычисляет абсолютное значение комплексного числа znum. Описание записи типа complex дано в файле math.h.
cos	double cos (double x);	Функция косинуса. Угол задается в радианах.
cosh	double cosh (double x);	Возвращает значение гиперболического косинуса.
exp	double exp (double x);	Вычисляет значение e^x.
fabs	double fabs (double x);	Возвращает абсолютное значение вещественного аргумента x двойной точности.
floor	double floor (double x);	Находит наибольшее целое, не превышающее значения x. Возвращает его в форме double
fmod	double fmod (double x, double y);	Возвращает остаток от деления x на y
hypot	double hypot (double x, double y);	Вычисляет гипотенузу прямоугольного треугольника из выражения $z^2=x^2+y^2$
labs	long labs (long x);	Возвращает абсолютное значение длинного целого аргумента x
log	double log (double x);	Возвращает значение натурального логарифма x
log 10	double log 10 (double x);	Возвращает значение десятичного логарифма
pow	double pow(double x, double y);	Возвращает значение x^y
pow 10	double pow 10 (int p);	Возвращает значение 10^p
sin	double sin (double x);	Функция синуса. Угол задается в радианах
sinh	double sinh (double x);	Возвращает значение гиперболического синуса x
sqrt	double sqrt (double x);	Возвращает значение $\sqrt{x}$

tan	double tan (double x);	Функция тангенса. Угол задается в радианах
tanh	double tanh (double x);	Возвращает значение гиперболического тангенса x
ldexp	double ldexp (double v, int e);	Возвращает значение $\sqrt{2^e}$
poly	double poly (double x, int n, double c[ ]);	Вычисляет полином вида $c[n]x^n+c[n-1]x^{n-1}+...+c[1]x+c[0]$

Рассмотрим пример программы вычисления арифметического выражения с использованием встроенных математических функций.

Пусть необходимо вычислить:

$$y = \sin\left(x^2\right) + \frac{\cos 2x}{\sqrt{x}} + 3x^2$$

Программа будет иметь вид, представленный в листинге 4.

```
#include <stdio.h>
#include <math.h>
int main( )
{
        double x, y;
        printf("введите x");
        scanf("%lf", &x);
        y = sin(x * x) + cos(2 * x) / sqrt(x) + 3 * x * x * x;
        printf("y=%f", y);
        return 0;
}
```

Листинг 4. Пример вычисления функции $y = \sin\left(x^2\right) + \frac{\cos 2x}{\sqrt{x}} + 3x^2$.

В начале текста программы специальными командами #include подключаются стандартные заголовочные файлы, в которых содержатся прототипы функций и отдельные определения, используемые функциями.

Функция printf() обеспечивает форматный вывод информации, а функция scanf() – форматный ввод. Подробно эти функции будут рассмотрены на далее.

Заключение

Сегодня вы познакомились с базовыми компонентами программирования на языке C/C++. В их число входят типы данных, переменные, константы и выражения.

Предопределенные типы данных в Borland C++ включают в себя типы bool, int, char, float, double и void. В языке C++ гибкость типов данных увеличивается благодаря применению модификаторов типов. Эти модификаторы изменяют точность представления и диапазон значений переменных. Модификаторами типа являются signed, unsigned, short и long.

Идентификаторы в Borland C++ могут иметь длину до 32 символов и должны начинаться с буквы или подчеркивания. Последующие символы идентификатора могут быть буквой, цифрой или подчеркиванием. Идентификаторы C++ чувствительны к регистру. Ограничение на длину в 32 символа может быть изменено путем установки соответствующей опции компилятора.

Директива #include является специальной командой компилятора. Она предписывает компилятору включить в программу содержимое определенного файла, как если бы вы сами ввели его в текущий исходный файл.

Объявление констант предусматривает использование директивы #define для объявления констант, определяемых с помощью макросов, или использование ключевого слова const для объявления формальных констант. Формальные константы требуют определения их типа (значение по умолчанию является int), имени и ассоциированного с ними значения.

Объявление переменной требует задание ее типа и имени. C++ дает возможность инициализировать переменную при ее объявлении. Вы можете объявить несколько переменных в одном операторе объявления.

Арифметическими операциями являются +, -, *, / и % (деление по модулю).

Арифметические выражения различаются по сложности. Самое простое выражение содержит единственный элемент данных (литерал, константу или переменную). Сложные выражения включают набор операций, функции, литералы, константы и переменные.

Операции инкремента и декремента используются в префиксной и постфиксной формах. Язык C++ дает вам возможность применять эти операции к переменным, в которых хранятся символы, целые числа и даже числа с плавающей точкой.

Арифметические операции присваивания дают вам возможность записывать более короткие арифметические выражения, в которых первый операнд является также переменной, принимающей результат вычислений.

Оператор sizeof возвращает как для типов данных, так и для переменных их размер в байтах.

Механизм приведения типа дает вам возможность форсировать преобразование типа выражения.

Операции отношения и логические операции позволяют строить логические выражения.

Булевы выражения объединяют операции отношений и логические операции для формулирования нетривиальных условий. Эти выражения позволяют программе принимать сложные решения.

Условное выражение предлагает короткую форму для простого оператора if-else с двумя альтернативами.

Операции манипулирования битами выполняют поразрядные операции AND, OR, XOR и NOT. Кроме того, в C++ поддерживаются поразрядные операции сдвига << и >>.

Операции манипулирования битами с присваиванием предлагают короткие формы для простых операций манипулирования битами.