Лекция 2
Спецификации, базовые типы, операции, операторы и выражения языка C/C++

Идентификаторы и ключевые слова

Идентификатор (ID) — это имя переменной, функции, класса или другого объекта в C/C++. Можно определять идентификаторы любыми словами/именами. Тем не менее, есть несколько общих правил, которые необходимо соблюдать:

Идентификатор не может быть ключевым словом. Ключевые слова зарезервированы.

Идентификатор может состоять только из букв (нижнего или верхнего регистра), цифр или символов подчёркивания. Это означает, что все другие символы и пробелы — запрещены.

Идентификатор должен начинаться с буквы (нижнего или верхнего регистра). Он не может начинаться с цифры.

C++ различает нижний регистр от верхнего. nvalue отличается от nValue и отличается от NVALUE.

При именовании объектов следует придерживаться общепринятых соглашений:

– **ID переменных и функций** обычно пишутся строчными (малыми) буквами – index, max();

– **ID типов** пишутся с большой буквы, например, Spis, Stack;

– **ID констант (макросов)** – большими буквами – INDEX, MAX_INT;

– идентификатор должен нести смысл, поясняющий назначение объекта в программе, например, birth_date – день рождения, sum – сумма;

– если ID состоит из нескольких слов, как, например, birth_date, то принято либо разделять слова символом подчеркивания, либо писать каждое следующее слово с большой буквы – birthDate.

C++ имеет зарезервированный набор из 92 слов (стандарт C++20) для собственного использования. Эти слова называются **ключевыми словами**, каждое из которых имеет своё особое значение. Ключевые (зарезервированные) слова не могут быть использованы в качестве идентификаторов.

Список всех ключевых слов в C++ (стандарт C++20):

alignas (C++11)	const_cast	int	static_assert (C++11)
alignof (C++11)	continue	long	static_cast
and	co_await (C++20)	mutable	struct
and_eq	co_return (C++20)	namespace	switch
asm	co_yield (C++20)	new	template
auto	decltype (C++11)	noexcept (C++11)	this
bitand	default	not	thread_local (C++11)
bitor	delete	not_eq	throw
bool	do	nullptr (C++11)	true
break	double	operator	try
case	dynamic_cast	or	typedef
catch	else	or_eq	typeid
char	enum	private	typename
char8_t (C++20)	explicit	protected	union
char16_t (C++11)	export	public	unsigned
char32_t (C++11)	extern	register	using
class	false	reinterpret_cast	virtual
compl	float	requires (C++20)	void
concept (C++20)	for	return	volatile
const	friend	short	wchar_t
consteval (C++20)	goto	signed	while
constexpr (C++11)	if	sizeof	xor
constint (C++20)	inline	static	xor_eq

Основные типы данных

Данные отображают в программе окружающий мир. Цель программы состоит в обработке данных. Данные различных типов хранятся и обрабатываются по-разному. Тип данных определяет:

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

В зависимости от требований задания программист выбирает тип для объектов программы. Типы C/C++ можно разделить на простые и составные. К простым типам относят типы, которые характеризуются одним значением. В C/C++ определено 6 простых типов данных:

Существует 4 спецификатора типа, уточняющих внутреннее представление и диапазон стандартных типов:

short (короткий)
long (длинный)
signed (знаковый)
unsigned (беззнаковый)

Тип int

Значениями этого типа являются целые числа.

Размер типа int не определяется стандартом, а зависит от компьютера и компилятора. Для 16-разрядного процессора под него отводится 2 байта, для 32-разрядного – 4 байта.

Если перед int стоит спецификатор short, то под число отводится 2 байта, а если спецификатор long, то 4 байта. От количества отводимой под объект памяти зависит множество допустимых значений, которые может принимать объект:

short int — занимает 2 байта, следовательно, имеет диапазон –32768 ..+32767;

long int – занимает 4 байта, следовательно, имеет диапазон –2 147 483 648..+2 147 483 647

Тип int совпадает с типом short int на 16-разрядных ПК и с типом long int на 32-разрядных ПК.

Модификаторы signed и unsigned также влияют на множество допустимых значений, которые может принимать объект:

unsigned short int — занимает 2 байта, следовательно, имеет диапазон 0 ..65536;

unsigned long int – занимает 4 байта, следовательно, имеет диапазон 0..+4 294 967 295.

Тип char

Значениями этого типа являются элементы конечного упорядоченного множества символов. Каждому символу ставится в соответствие число, которое называется кодом символа. Под величину символьного типа отводится 1 байт. Тип char может использоваться со спецификаторами signed и unsigned. В данных типа signed char можно хранить значения в диапазоне от –128 до 127. При использовании типа unsigned char значения могут находиться в диапазоне от 0 до 255. Для кодировки используется код ASCII (American Standard Code foe International Interchange). Символы с кодами от 0 до 31 относятся к служебным и имеют самостоятельное значение только в операторах ввода-вывода.

Величины типа char также применяются для хранения чисел из указанных диапазонов.

Тип wchar_t

Предназначен для работы с набором символов, для кодировки которых недостаточно 1 байта, например Unicode. Размер этого типа, как правило,

соответствует типу short. Строковые константы такого типа записываются с префиксом L: L “Gates”.

Тип bool

Тип bool называется логическим. Его величины могут принимать значения true и false. Внутренняя форма представления false – 0, любое другое значение интерпретируется как true.

Типы с плавающей точкой.

Внутреннее представление вещественного числа состоит из 2 частей: мантиссы и порядка. В IBM-совместимых ПК величины типа float занимают 4 байта, из которых один разряд отводится под знак мантиссы, 8 разрядов под порядок и 24 – под мантиссу. Мантисса — это число, большее 1.0, но меньшее 2.0. Поскольку старшая цифра мантиссы всегда равна 1, она не хранится.

Величины типы double занимают 8 байтов, под порядок и мантиссу отводятся 11 и 52 разряда соответственно. Длина мантиссы определяет точность числа, а длина порядка его диапазон.

Если перед именем типа double стоит спецификатор long, то под величину отводится 10 байт.

Тип void

К основным типам также относится тип void. Множество значений этого типа – пусто.

Таблица основных типов данных

Тип данных	Объем памяти (байт)	Диапазон значений
char	1	–128 … 127
int	4	–32768 … 32767
short	2	–32768 … 32767(–128 … 127)
long	4	–2147483648 … 2147483647
unsigned int	4	0 … 65535
unsigned long	4	0 … 4294967295
float	4	$3{,}14\cdot10^{-38}$ … $3{,}14\cdot10^{38}$
double	8	$1{,}7\cdot10^{-308}$ … $1{,}7\cdot10^{308}$
long double	10	$3{,}4\cdot10^{-4932}$ … $3{,}4\cdot10^{4932}$
bool (только в C++)	1	true, false

Сложные типы данных – массивы, структуры – ***struct***, объединения – ***union***, перечисления – ***enum***.

Декларация объектов

Все объекты, с которыми работает программа, необходимо декларировать, т.е. объявлять компилятору об их присутствии. При этом возможны две формы декларации:

– описание, не приводящее к выделению памяти;

– определение, при котором под объект выделяется объем памяти в соответствии с его типом; в этом случае объект можно инициализировать, т.е. задать его начальное значение.

Кроме констант, заданных в исходном тексте, все объекты программы должны быть явно декларированы по следующему формату:

<атрибуты> <список ID объектов>;

элементы *списка ID объектов* разделяются запятыми, а *атрибуты* – разделителями, например: *int i, j, k*; *float a, b*;

Объекты программы могут иметь следующие атрибуты:

класс памяти – характеристика способа размещения объектов в памяти (статическая, динамическая); определяет область видимости и время жизни переменной (по умолчанию – ***auto***) (*данные атрибуты будем рассматривать позже*).

тип – тип будущих значений декларируемых объектов (по умолчанию устанавливается тип ***int***).

Класс памяти и тип – атрибуты необязательные и при отсутствии одного из них (но не обоих одновременно) устанавливаются атрибуты по умолчанию.

Примеры декларации простых объектов:

int i, j, k; *char r;* *double gfd;*

Константы

Константами называют величины, которые не изменяют своего значения во время выполнения программы, т.е. это объекты, не подлежащие использованию в левой части операции присваивания, т.к. константа – это неадресуемая величина и, хотя она хранится в памяти компьютера, не существует способа определить ее адрес. В языке C/C++ константами являются:

– самоопределенные арифметические константы целого и вещественного типов, символьные и строковые данные;

– идентификаторы массивов и функций;

– элементы перечислений.

Целочисленные константы

Общий формат записи: **±*n*** (+ обычно не ставится).

Десятичные константы – это последовательность цифр 0...9, первая из которых *не должна быть 0*. Например, 22 и 273 – обычные целые константы, если нужно ввести длинную целую константу, то указывается признак *L*(*l*) – 273*L* (273*l*). Для такой константы будет отведено – 4 байта. Обычная целая константа, которая слишком длинна для типа *int*, рассматривается как *long*.

Существует система обозначений для восьмеричных и шестнадцатеричных констант.

Восьмеричные константы – это последовательность цифр от 0 до 7, первая из которых *должна быть 0*, например: $020_8 = 16_{10}$.

Шестнадцатеричные константы – последовательность цифр от 0 до 9 и букв от *A* до *F* (*a*...*f*), начинающаяся символами 0*X* (0*x*), например: $0X1F_{16}$ $(0x1f)_{16} = 31_{10}$.

Восьмеричные и шестнадцатеричные константы могут также заканчиваться буквой *L*(*l*) – *long*, например, 020*L* или 0*X*20*L*.

Примеры целочисленных констант:

1992	777	1000*L*	– десятичные;
0777	00033	01*l*	– восьмеричные;
0*x*123	0*X*00*ff*	0*xb*8000*l*	– шестнадцатеричные.

Константы вещественного типа

Данные константы размещаются в памяти в формате *double*, а во внешнем представлении могут иметь две формы:

1) с фиксированной десятичной точкой, формат записи: ***±n.m***, где *n*, *m* – целая и дробная части числа;

2) с плавающей десятичной точкой (экспоненциальная форма) представляется в виде мантиссы и порядка. Мантисса записывается слева от знака экспоненты (*E* или *e*), а порядок – справа. Значение константы определяется как произведения мантиссы и числа 10, возведенного в указанную в порядке степень.

Общий формат таких констант: ***±n.mE±p***, где *n*, *m* – целая и дробная части числа, *p* – порядок; ±0.*xxxE*±*p* – нормализованный вид, например, $1{,}25 \cdot 10^{-8} = 0.125E{-}7$.

Примеры констант с фиксированной и плавающей точками:

1.0 –3.125 100*e*–10 0.12537*e*+12.

Пробелы внутри чисел не допускаются, а для отделения целой части числа от дробной используется ***точка***. Можно опустить нулевую дробную или целую части числа, но не обе сразу, например, 1.0 ↔ 1. или 0.5 ↔ .5 .

В любом случае при использовании вещественных констант наличие так называемой десятичной точки обязательно.

Символьные константы

Символьная константа – это символ, заключенный в одинарные кавычки: '*A*', '*x*' (тип *char* занимает в памяти один байт).

Также используются специальные последовательности символов – управляющие (*escape*) последовательности:

n	новая строка;
t	горизонтальная табуляция;
b	шаг назад;

\r	возврат каретки;
\v	вертикальная табуляция;
\f	перевод формата (переход на новую строку);
\\	обратный слеш;
\'	апостроф;
\"	– кавычки;
\0	– символ «пусто», не путать с символом ‘0’.

Символьная константа '\0' – это нулевой байт, каждый бит которого равен нулю.

При присваивании символьным переменным значений констант значения констант заключаются в апострофы, например:

char ss = ‘У’;

Текстовые символы непосредственно вводятся с клавиатуры, а специальные и управляющие – представляются в исходном тексте парами символов, например: \\ , \' , \" .

Примеры символьных констант: '*A*', '9', '$', '*n*'.

Строковые константы

Строковая константа представляет собой последовательность символов кода *ASCII*, заключенную в кавычки (”). Во внутреннем представлении к строковым константам добавляется пустой символ '\0', который не является цифрой 0, на печать не выводится (в таблице кодов *ASCII* имеет код = 0) и является признаком окончания строки.

Кавычки не являются частью строки, а служат только для ее ограничения. Строка в языке C/C++ представляет собой массив, состоящий из символов. Внутреннее представление константы "1234*ABC*": '1' '2' '3' '4' '*A*' '*B*' '*C*' '\0' .

Примеры строковых констант:

"Система", "*n**t* Аргумент *n*", "Состояние \"*WAIT* \" ".

Строковые константы еще называют *строковыми литералами*.

В конец строковой константы компилятор автоматически помещает нуль-символ.

Длинную строковую константу можно разбить на несколько, используя символ переноса – обратный слеш (\). Например:

“*Вы студенты 1 курса* \
Инженерно-экономического факультета \
Белорусского государственного университета информатики и радиоэлектроники”

Компилятор C/C++ воспримет такую запись как единое целое, игнорируя символы обратного слеша.

Операции, выражения

Выражения используются для вычисления значений (определенного типа) и состоят из операндов, операций и скобок. Каждый операнд может быть, в свою очередь, выражением или одним из его частных случаев – константой или переменной. Операнды задают данные для вычислений.

Знак операции – это один или более символов, определяющих действие над операндами, т.е. операции задают действия, которые необходимо выполнить. Внутри знака операции пробелы не допускаются.

Операции делятся на унарные, бинарные и тернарные – по количеству участвующих в них операндов, и выполняются в соответствии с приоритетами. Для изменения порядка выполнения операций используются круглые скобки.

Большинство операций выполняются слева направо, например,

$$a+b+c \rightarrow (a+b)+c.$$

Исключение составляют унарные операции, операции присваивания и условная операция (?:), которые выполняются справа налево.

Арифметические операции

Обозначения арифметических операций:

\+ (сложение);

– (вычитание);

/ (деление, для *int* операндов – с отбрасыванием остатка);

* (умножение);

% (остаток от деления целочисленных операндов со знаком первого операнда – деление «по модулю»).

Операндами традиционных арифметических операций (+ – * /) могут быть константы, переменные, обращения к возвращающим значения функциям, элементы массивов, любые арифметические выражения, указатели (с ограничениями).

Порядок выполнения действий в арифметических выражениях следующий: выражения в круглых скобках; операции *, /, %; операции **+, –.**

Унарные операции «знак числа» (+, –) обладают самым высоким приоритетом и определены для операндов числовых типов (имеющих числовой результат), при этом «+» носит только информационный характер, «–» меняет знак операнда на противоположный (неадресная операция).

Операции *, /, % обладают высшим приоритетом над операциями **+, –**, поэтому при записи сложных выражений нужно использовать общепринятые

математические правила: $x + y \cdot z - \frac{a}{b+c}$ ↔ *x*+*y***z*–*a* /(*b*+*c*) , т.е. использовать круглые скобки.

Операция присваивания

Формат операции присваивания:

***Операнд*_1 = *Операнд*_2 ;**

*Операндом*_1 (левый операнд) может быть только переменная. Левый операнд операции присваивания получил название ***L–значение***, (*L–value*, *Left–value*) – *адресное выражение*. Так в C/C++ называют любое выражение, адресующее некоторый участок оперативной памяти, в который можно записать некоторое значение. Переменная – это частный случай адресного выражения.

*Операндом*_2 (правый операнд) могут быть: константа, переменная или любое выражение, составленное в соответствии с синтаксисом языка C/C++. Правый операнд операции присваивания назвали ***R–значение***, (*R–value*, *Right–value*).

Присваивание значения в языке C/C++, в отличие от традиционной интерпретации, рассматривается как выражение, имеющее значение левого операнда после присваивания. Таким образом, присваивание может включать несколько операций присваивания, изменяя значения нескольких операндов, например:

```
int i, j, k;
float x, y, z;
    ...
i = j = k = 0;              ↔   k = 0, j = k, i = j;
x = i + (y = 3) – (z = 0);  ↔   z = 0, y = 3, x = i + y – z;
```

Примеры недопустимых выражений:

– присваивание константе: 2 = *x* + *y*;
– присваивание функции: *getch*() = *i*;
– присваивание результату операции: (*i* + 1) = 2 + *y*;

Сокращенная запись операции присваивания

В языке C/C++ используются два вида сокращенной записи операции присваивания:

1) вместо записи: ***v* = *v* # *e***;

где # – любая арифметическая операция (операция над битовым представлением операндов), рекомендуется использовать запись ***v* #= *e***;

Например, *i* = *i* + 2; ↔ *i* += 2; (***знаки операций – без пробелов***);

2) вместо записи: *x* = *x* # 1;

где # – символы, обозначающие операцию инкремента (+1), либо декремента (–1), *x* – целочисленная переменная (или переменная-указатель), рекомендуется использовать запись:

##x; – префиксную, или ***x##;*** – постфиксную.

Если эти операции используются в чистом виде, то различий между постфиксной и префиксной формами нет. Если же они используются в выражении, то в префиксной форме (*##x*) сначала значение *x* изменится на 1, а затем полученный результат будет использован в выражении; в постфиксной форме (*x##*) – сначала значение переменной *x* используется в выражении, а затем изменится на 1.

Пример 1:

int i, j, k;	Смысл записи
float x, y;	
...	
x *= *y*;	*x* = x*y;
i += 2;	*i* = *i* + 2;
x /= *y*+15;	*x* = *x*/(*y* + 15);
--*k*;	*k* = *k* – 1;
k--;	*k* = *k* – 1;
j = *i*++;	*j* = *i*; *i* = *i* + 1;
j = ++*i*;	*i* = *i* + 1; *j* = i;

Пример 2:

int n, a, b, c, d;	Значения
n = 2; *a* = *b* = *c* = 0;	
a = ++*n*;	*n*=3, *a*=3
a += 2;	*a*=5
b = *n*++;	*b*=3, *n*=4
b –= 2;	*b*=1
c = --*n*;	*n*=3, *c*=3
c *= 2;	*c*=6
d = *n*--;	*d*=3, *n*=2
d %= 2;	*d*=1

Преобразование типов операндов арифметических операций

Если операнды арифметических операндов имеют один тип, то и результат операции будет иметь такой же тип.

Но, как правило, в операциях участвуют операнды различных типов. В этом случае они преобразуются к общему типу в порядке увеличения их «размера памяти», т.е. объема памяти, необходимого для хранения их значений. Поэтому неявные преобразования всегда идут от «меньших» объектов к «большим».

Схема выполнения преобразований операндов арифметических операций выглядит следующим образом:

short, char → ***int*** → ***unsigned*** → ***long*** → ***double***
float → ***double***

Стрелки отмечают преобразования даже однотипных операндов перед выполнением операции. То есть действуют следующие правила:

– значения типов *char* и *short* всегда преобразуются в *int*;

– если один из операндов имеет тип *double*, то и другой преобразуется в *double*;

– если один из операндов *long*, то другой преобразуется в *long*.

Внимание. Результатом операции **1/3** будет значение **0**, чтобы избежать такого рода ошибок, необходимо явно изменить тип хотя бы одного операнда, т.е. записать, например: 1./3

Типы *char* и *int* могут свободно смешиваться в арифметических выражениях. Каждая переменная типа *char* автоматически преобразуется в *int*, что обеспечивает значительную гибкость при проведении преобразований, т.к. над типом *int* действия выполняются быстрее, чем над любым другим типом.

При выполнении операции присваивания значение правого операнда преобразуется к типу левого, который и является типом полученного результата. И здесь необходимо быть внимательным, т.к. при некорректном использовании операций присваивания могут возникнуть неконтролируемые ошибки. Так, при преобразовании *int* в *char* старший байт просто отбрасывается.

Пусть: *float x*; *int i*; тогда и $x = i$; и $i = x$; приводят к преобразованиям, причем *float* преобразуется в *int* отбрасыванием дробной части.

Тип *double* преобразуется в *float* округлением.

Длинное целое преобразуется в более короткое целое и *char* посредством отбрасывания бит в старших разрядах.

Итак, безопасным преобразованием типов является преобразование в порядке увеличения «размера памяти», обратное преобразование может привести к потере значащих разрядов.

Операция приведения типа

В любом выражении преобразование типов может быть осуществлено явно, для этого достаточно перед выражением поставить в круглых скобках атрибут соответствующего типа:

(*тип*) *выражение*;

ее результат – значение *выражения*, преобразованное к заданному *типу*.

Операция приведения типа вынуждает компилятор выполнить указанное преобразование, но ответственность за последствия возлагается на программиста. Использовать эту операцию рекомендуется везде, где это необходимо, например:

double x;
int n = 6, *k* = 4;
$x = (n + k)/3$; $\rightarrow x = 3$, т.к. дробная часть будет отброшена;
$x = (double)(n + k)/3$; $\rightarrow x = 3.333333$ – использование операции приведения типа позволило избежать округления результата деления целочисленных операндов.

Операции сравнения

В языке C/C++ используются следующие операции сравнения, т.е. отношения между объектами:

==	– равно или эквивалентно;	**!=**	– не равно;
<	– меньше;	**<=**	– меньше либо равно;
>	– больше;	**>=**	– больше либо равно.

Пары символов соответствующих операций разделять нельзя.

Общий вид операций отношений:

Операнд_1 Знак операции Операнд_2

Указанные операции выполняют сравнение значений первого операнда со вторым. Операндами могут быть любые арифметические выражения и указатели.

Значения арифметических выражений перед сравнением вычисляются и преобразуются к одному типу.

Арифметические операнды преобразуются по правилам, аналогичным для арифметических операций. Операнды-указатели преобразуются в целые числа необходимого типа. Результат сравнения указателей будет корректным в арифметическом смысле лишь для объектов одного массива.

Операции сравнения на равенство и неравенство имеют меньший приоритет, чем остальные операции отношений.

Примеры использования операций отношений:

$y > 0$, $x == y$, $x != 2$.

Отношения между объектами сложных типов проверяются либо посредством последовательного сравнения их элементов (для массивов), либо используя стандартные библиотечные функции.

Логические операции

Логические операции в порядке убывания относительного приоритета:

! – отрицание (логическое «НЕТ»);
&& – конъюнкция (логическое «И»);
|| – дизъюнкция (логическое «ИЛИ»).

Операндами (выражениями) логических операций могут быть любые скалярные типы.

В Си ненулевое значение операнда трактуется как «истина», а нулевое – «ложь». Результатом логической операции, как и в случае операций отношения, может быть 1 или 0.

В C++ результатом логической операции будет true или false.

Общий вид операции *отрицания:* **! *выражение***

Общий вид операций *конъюнкции* и *дизъюнкции*:
Выражение_1 знак операции Выражение_2

Особенность операций конъюнкции и дизъюнкции – экономное последовательное вычисление выражений-операндов:

– если выражение_1 операции «конъюнкция» ложно, то результат операции – ноль / false и выражение_2 не вычисляется;

– если выражение_1 операции «дизъюнкция» истинно, то результат операции – единица / true и выражение_2 не вычисляется.

Например:

$y > 0$ && $x = 7$ → истина, если оба выражения истинны;

$e > 0$ || $x = 7$ → истина, если хотя бы одно выражение истинно.

Старшинство операции «И» выше, чем «ИЛИ» и обе они младше операций отношения и равенства.

Относительный приоритет логических операций позволяет пользоваться общепринятым математическим стилем записи сложных логических выражений, например:

$0 < x < 100$ ↔ $0 < x$ && $x < 100$;

$x > 0, y \le 1$ ↔ $x > 0$ && $y <= 1$.

Операция «,» (запятая)

Данная операция используется при организации строго гарантированной последовательности вычисления выражений (обычно используется там, где по синтаксису допустима только одна операция, а необходимо разместить две и более, например, в операторе *for*). Форма записи:

выражение_1, …, выражение_N;

выражения 1, 2,…, *N* вычисляются последовательно друг за другом и результатом операции становится значение последнего выражения *N*, например:

$m = (i = 1, j = i ++, k = 6, n = i + j + k);$

получим последовательность вычислений: **$i = 1, j = i = 1, i = 2, k = 6, n = 2+1+6,$** и в результате **$m = n = 9.$**

Операторы

Операторы языка Си можно разделить на три группы: операторы-декларации (рассмотрены ранее), операторы преобразования объектов, и операторы управления процессом выполнения алгоритма.

Язык программирования С++ поддерживает все операторы своего прародителя Си и дополнен новыми операторами (например, операторами

приведения типа: const_cast, static_cast, dynamic_cast, reinterpret_cast) и возможностями (например, возможность перегрузки операторов).

Программирование процесса ***преобразования объектов*** производится посредством записи операторов (инструкций).

Простейший вид операторов – выражение, заканчивающееся символом «;» (точка с запятой). Выполнение такого оператора заключается в вычислении некоторого выражения.

Простые операторы:
оператор присваивания (выполнение операций присваивания),
оператор вызова функции (выполнение операции вызова функции),
пустой оператор «;» – частный случай выражения. Пустой оператор используют тогда, когда по синтаксису оператор требуется, а по смыслу – нет.

Примеры операторов «выражение»:
i++; – выполняется операция инкремента (увеличение на 1);
x+y; – выполняется операция сложения (результат будет утерян);
a=b–c; – выполняется операция вычитания с одновременным присваиванием.

Операторы записываются в свободном формате с использованием разделителей между ключевыми словами. Любой оператор может помечаться меткой – идентификатор и символ «:» (двоеточие). Область действия метки – функция, где эта метка определена.

К ***управляющим операторам*** относятся: операторы условного и безусловного переходов, оператор выбора альтернатив (переключатель), операторы организации циклов и передачи управления (перехода).

Каждый из управляющих операторов имеет конкретную лексическую конструкцию, образуемую из ключевых слов языка C/C++, выражений и символов-разделителей.

Допускается вложенность операторов. В случае необходимости можно использовать составной оператор – блок, состоящий из любой последовательности операторов, заключенных в фигурные скобки – { и }, после закрывающей скобки символ «;» не ставится.

Разветвляющиеся алгоритмы (условные операторы)

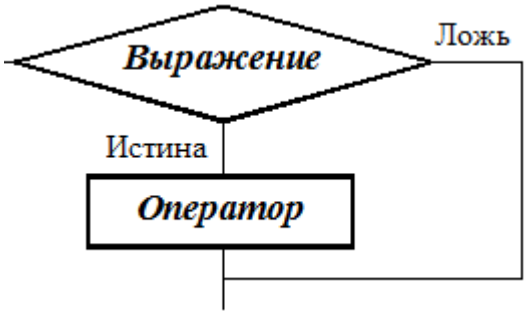
Условный оператор *if* используется для разветвления процесса выполнения кода программы на два направления.

Имеется две разновидности условного оператора: простой и полный. Синтаксис простого оператора:

if (выражение) оператор;

выражение – логическое или арифметическое выражение, вычисляемое перед проверкой, и, если выражение истинно (не равно нулю), то выполняется оператор, иначе он игнорируется; оператор – простой или составной (блок) оператор языка C/C++. Если в случае истинности выражения необходимо выполнить несколько операторов (более одного), их необходимо заключить в фигурные скобки.

Структурная схема простого оператора.



Примеры записи условного оператора if:

```
if (x > 0) x = 0;

if (i != 1) {
j++;  s = 1;    – последовательность операций (блок);
}
```

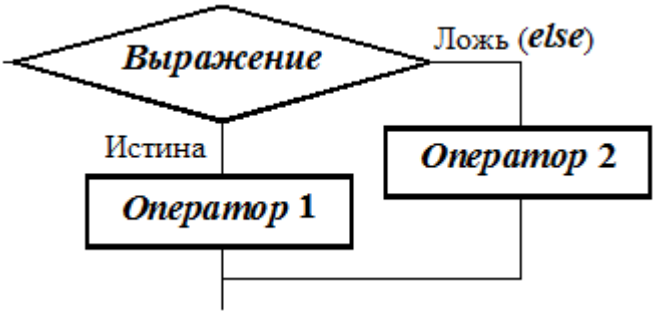
Синтаксис полного оператора условного выполнения:

if (выражение) оператор 1;
else оператор 2;

Если выражение не равно нулю (истина), то выполняется оператор 1, иначе – оператор 2. Операторы 1 и 2 могут быть простыми или составными (блоками).

Наличие символа «;» перед словом else в языке C/C++ обязательно.

Структурная схема оператора:



Примеры записи:

```
if (x > 0) j = k+10;
else m = i+10;
```

```
if ( x>0 &&  k!=0 ) {
j = x/k;
x += 10;
}
else m = k*i + 10;
```

Операторы 1 и 2 могут быть любыми операторами, в том числе и условными. Тогда, если есть вложенная последовательность операторов if – else, то фраза else связывается с ближайшим к ней предыдущим if, не содержащим ветвь else. Например:

```
if (n > 0)
    if(a > b)  z = a;
    else   z = b;
```

Здесь ветвь else связана со вторым if (a > b). Если же необходимо связать фразу else с внешним if, то используются операторные скобки:

```
if(n > 0) {
    if(a > b)  z = a;
}
else  z = b;
```

Условная операция «? :»

Условная операция – тернарная, т.к. в ней участвуют три операнда. Формат написания условной операции следующий:

Выражение 1 ? выражение 2 : выражение 3;

если выражение 1 (условие) отлично от нуля (истинно), то результатом операции является значение выражения 2, в противном случае – значение выражения 3. Каждый раз вычисляется только одно из выражений 2 или 3.

Условное вычисление применимо к арифметическим операндам и операндам-указателям.

Пример. Участок программы для нахождения максимального значения z из двух чисел a и b, используя оператор if и условную операцию.

1. Запишем оператор if :

```
if (a > b)  z = a;
else  z = b;
```

2. Используя условную операцию, получим

```
z = (a > b) ? a : b;
```

Оператор выбора альтернатив (переключатель)

Оператор switch (переключатель) предназначен для разветвления процесса вычислений на несколько направлений.

Общий вид оператора:

switch (выражение) {
case константа1: список операторов 1
case константа2: список операторов 2
...
case константаN: список операторов N
default: список операторов N+1 – необязательная ветвь;
}

Выполнение оператора начинается с вычисления выражения, значение которого должно быть целого или символьного типа. Это значение сравнивается со значениями констант и используется для выбора ветви, которую нужно выполнить.

В данной конструкции константы фактически выполняют роль меток. Если значение выражения совпало с одной из перечисленных констант, то управление передается в соответствующую ветвь. После этого, если выход из переключателя в данной ветви явно не указан, то последовательно выполняются все остальные ветви.

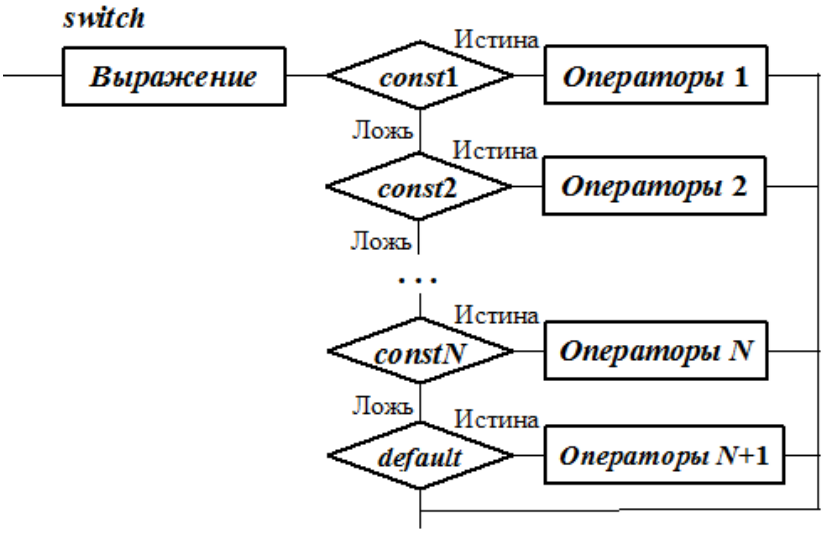
Все константы должны иметь разные значения, но быть одного и того же типа. Порядок следования ветвей не регламентируется.

В случае несовпадения значения выражения ни с одной из констант выбора происходит переход на метку default либо, при ее отсутствии, к оператору, следующему за оператором switch.

Управляющий оператор **break** (разрыв) выполняет выход из оператора switch. Если по совпадению с каждой константой должна быть выполнена одна и только одна ветвь, схема оператора switch следующая:

```
switch (выражение) {
        case константа1: операторы 1;  break;
        case константа2: операторы 2;  break;
        ...
        case константаN: операторы N;  break;
        default:  операторы (N+1);  break;
 }
```

Структурная схема рассмотренной конструкции (с использованием оператора break):



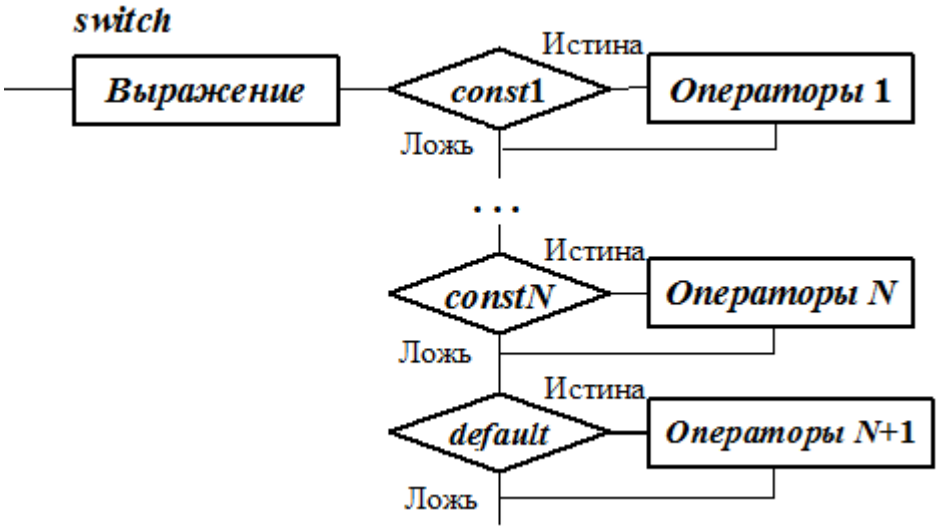
Пример оператора switch с использованием оператора break:

```
int i = 2;
switch(i)      {
        case 1:  puts ( "Случай 1. "); break;
        case 2:  puts ( "Случай 2. "); break;
        case 3:  puts ( "Случай 3. "); break;
        default: puts ( "Случай default. "); break;
}
```

Результатом выполнения данной программы будет:

Случай 2.

Аналогичный пример без использования оператора break.
схема общего вида такой конструкции:




```
int i = 2;
switch(i)      {
        case 1: puts ( "Случай 1. ");
```

```
        case 2: puts ( "Случай 2. ");
        case 3: puts ( "Случай 3. ");
        default: puts ( "Случай default. ");
    }
```

В данном случае результат будет следующим:

Случай 2.
Случай 3.
Случай default.

Пример реализации простейшего калькулятора на четыре действия с контролем правильности ввода символа нужной операции. Ввод данных осуществляется следующим образом: операнд 1, символ нужной операции, операнд 2.

Текст программы может быть следующим:

```
#include <stdio.h>
void main(void)
{
double a, b, c;
char s;
m1:  fflush(stdin);                  // Очистка буфера ввода stdin
printf("\n Введите операнд 1, символ операции, операнд 2:");
scanf("%lf%c%lf", &a, &s, &b);
switch(s)  {
        case '+':       c = a+b;       break;
        case '–':       c = a–b;       break;
        case '*':       c = a*b;       break;
        case '/':       c = a/b;       break;
        default: printf("\n Ошибка, повторите ввод! "); goto m1;
        }
  printf("\n a %c b = %lf ", s, c);
  printf("\n Продолжим? (Y/y) ");
  s = getch();
  if ( (s=='Y') || (s=='y') ) goto m1;
  printf("\n До свидания! ");
}
```

Void:

Если ключевое слово void указывает возвращаемый тип функции, оно означает, что данная функция не возвращает никакого значения. Если оно используется для списка параметров функции, оно означает, что функция не принимает никаких параметров.

После запуска программы на экран выводится подсказка, нужно набрать соответствующие значения без пробелов, например, как показано ниже, и нажать клавишу Enter:

Введите операнд 1, символ операции, операнд 2: ***2.4+3.6***

На экран будет выведен результат и дальнейший диалог:
a + b = 6.000000
Продолжим? (Y/y)

Введя символ **y (Y),** вернемся в начало функции и на экране вновь появится:
Введите операнд 1, символ операции, операнд 2:

Если ошибочно ввести – 2r3 , появятся следующие сообщения:
Ошибка, повторите ввод!
Введите операнд 1, символ операции, операнд 2: ***2 * 3***
*a*b = 6.000000*
Продолжим? (Y/y)

Нажимаем любую клавишу, кроме y или Y – следует сообщение
До свидания!

Программа закончена.

Циклические алгоритмы

Практически все алгоритмы решения задач содержат циклически повторяемые участки. Цикл – это одно из фундаментальных понятий программирования. Под циклом понимается организованное повторение некоторой последовательности операторов.

Любой цикл состоит из кода цикла, т.е. тех операторов, которые выполняются несколько раз, начальных установок, модификации параметра цикла и проверки условия продолжения выполнения цикла.

Один проход цикла называется шагом или итерацией. Проверка условия продолжения цикла происходит на каждой итерации либо до выполнения кода цикла (с предусловием), либо после выполнения (с постусловием).

Для организации циклов используются специальные операторы. Перечень разновидностей операторов цикла языка C/C++ следующий:

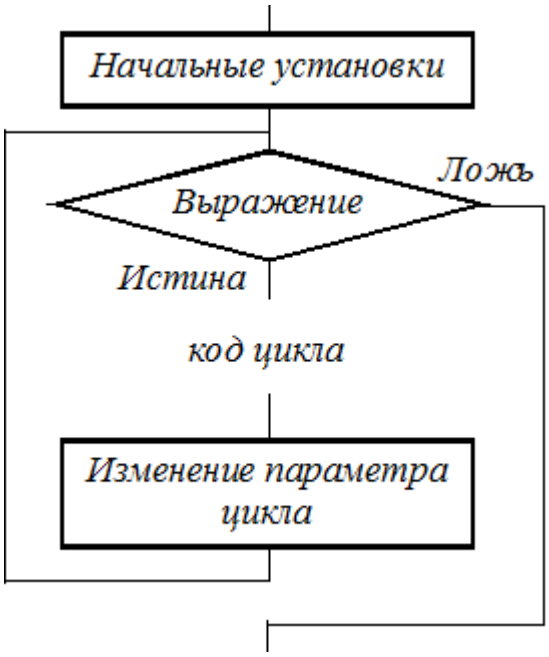
– оператор цикла с предусловием;
– оператор цикла с постусловием;
– оператор цикла с предусловием и коррекцией.

Оператор с предусловием while

Цикл с предусловием имеет вид

while (выражение)
код цикла;

Структурная схема:



Выражение определяет условие повторения кода цикла, представленного простым или составным оператором.

Если выражение в скобках – истина (не равно 0), то выполняется код цикла. Это повторяется до тех пор, пока выражение не примет значение 0 (ложь). В этом случае происходит выход из цикла и выполняется оператор, следующий за конструкцией while. Если выражение в скобках изначально ложно (равно 0), то цикл не выполнится ни разу.

Код цикла может включать любое количество операторов, связанных с конструкцией while, которые нужно заключить в фигурные скобки (организовать блок), если их более одного.

Переменные, изменяющиеся в коде цикла и используемые при проверке условия продолжения, называются ***параметрами цикла***. Целочисленные параметры цикла, изменяющиеся с постоянным шагом на каждой итерации, называются ***счетчиками цикла***.

Начальные установки могут явно не присутствовать в программе, их смысл состоит в том, чтобы до входа в цикл задать значения переменным, которые в этом цикле используются.

Цикл завершается, если условие его продолжения не выполняется. Возможно принудительное завершение как текущей итерации, так и цикла в целом.

Для этого используют оператор **continue** – переход к следующей итерации цикла и **break** – выход из цикла.

Передавать управление извне внутрь цикла не рекомендуется, так как получите непредсказуемый результат.

Например, необходимо сосчитать количество символов в строке. Предполагается, что входной поток настроен на начало строки. Тогда подсчет символов выполняется следующим образом:

```
int count = 0;
char ch = getchar();
while ( ch != \n') {
      count++;
      ch = getchar();
 }
```

В языке C/C++ в выражение, управляющее циклом, можно включить и оператор присваивания переменной ch, например:

```
char ch;
int count = 0;
while (( ch=getchar()) != '\n') count++;
```

Здесь переменная ch применяется только в выражении, управляющем циклом, поэтому от ch можно отказаться:

```
int count = 0;
while ( getchar() !='\n') count ++;
```

Полезные примеры

1. Организация выхода из бесконечного цикла по нажатии клавиши Esc

```
while (1) {                          // Бесконечный цикл
            ...
      if (kbhit() && getch()==27 ) break;
            ...
}
```

Функция kbhit() возвращает значение > 0, если нажата любая клавиша, а функция getch() возвращает код нажатой клавиши (код клавиши Esc равен 27). В результате выполнения оператора if, если будет нажата клавиша Esc, выполнится оператор break и произойдет выход из цикла.

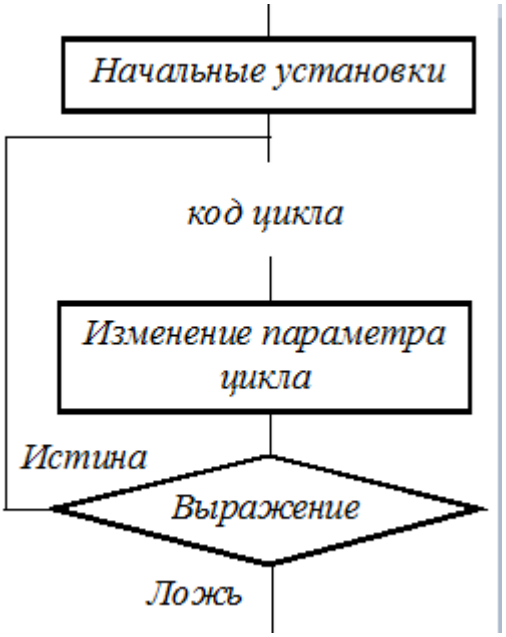
2. Организации паузы в работе программы с помощью цикла, выполняющегося до тех пор, пока не нажата любая клавиша

```
            ...
while (!kbhit());
            ...
```

Оператор цикла с постусловием do – while

Цикл с постусловием имеет вид

do
код цикла;
while (выражение);



Код цикла будет выполняться до тех пор, пока выражение истинно. Все, что говорилось выше, справедливо и здесь, за исключением того, что данный цикл всегда выполняется хотя бы один раз, даже если изначально выражение ложно.

Здесь сначала выполняется код цикла, после чего проверяется, надо ли его выполнять еще раз.

Следующая программа будет «вас приветствовать» до тех пор, пока будем вводить символ Y или y (Yes). После введения любого другого символа цикл завершит свою работу.

```
#include <stdio.h>
void main(void)
{
 char answer;
 do {
        puts(" Hello! => ");
        scanf(" %c ", &answer);
 }
 while ((answer=='y')||(answer=='Y'));
}
```

Результат выполнения программы:

```
Hello! => Y
Hello! => y
Hello! => d
```

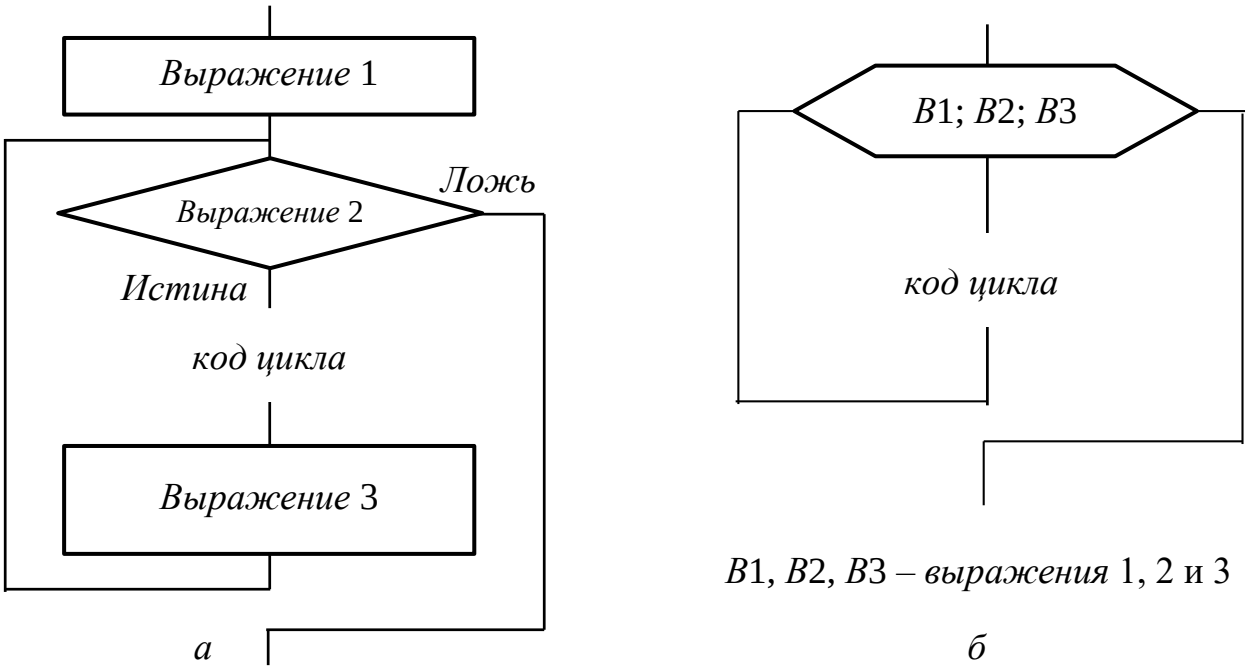
Оператор цикла с предусловием и коррекцией for

Общий вид оператора:

for (выражение 1; выражение 2; выражение 3)
код цикла;

где выражение 1 – инициализация счетчика (параметр цикла);
выражение 2 – условие продолжения счета;
выражение 3 – коррекция счетчика.

Схемы оператора цикла for:



а – схема работы; б – блок-схема

Инициализация используется для присвоения счетчику (параметру цикла) начального значения.

Выражение 2 определяет условие выполнения цикла. Как и в предыдущих случаях, если его результат не нулевой («истина»), – то цикл выполняется, иначе – происходит выход из цикла.

Коррекция выполняется после каждой итерации цикла и служит для изменения параметра цикла.

Выражения 1, 2 и 3 могут отсутствовать (пустые выражения), но символы «;» опускать нельзя.

Например, для суммирования первых N натуральных чисел можно записать такой код:

```
sum = 0;
for ( i = 1; i<=N; i++)  sum+=i;
```

Заметим, что в выражении 1 переменную-счетчик можно декларировать. Например:

```
for (int i = 1; i<=N; i++)
```

Областью действия такой переменной будет код цикла.

В заголовке оператора for может использоваться операция «запятая». Она позволяет включать в его выражения несколько операторов. Тогда рассмотренный

пример суммирования первых N натуральных чисел можно записать в следующем виде:

```
for ( sum = 0 , i = 1; i<=N; sum+= i , i++) ;
```

Оператор for имеет следующие возможности:
– можно вести подсчет с помощью символов, а не только чисел:

```
for (ch = 'a'; ch <= 'z'; ch++) ... ;
```

– можно проверить выполнение некоторого произвольного условия:

```
for (i= 0; s[i] >= '0' && s[i] < '9'; i++) ... ;
```

или

```
for (n = 1; n*n*n <= 216; n++) ... ;
```

Первое выражение необязательно должно инициализировать переменную. Необходимо только помнить, что первое выражение вычисляется только один раз, перед тем как остальные части начнут выполняться.

```
for (printf(" вводить числа по порядку! \n"); num!=6;)
        scanf("%d", & num);
printf(" последнее число – это то, что нужно. \n");
```

В этом фрагменте первое сообщение выводится на печать один раз, а затем осуществляется прием вводимых чисел, пока не поступит число 6.

Операторы и функции передачи управления

Формально к операторам передачи управления относятся:
– оператор безусловного перехода goto;
– оператор перехода к следующему шагу (итерации) цикла continue;
– выход из цикла, либо оператора switch – break;
– оператор возврата из функции return.

Оператор безусловного перехода goto

В языке C/C++ предусмотрен оператор goto, общий вид которого

goto метка ;

Он предназначен для передачи управления оператору, помеченному указанной меткой. Метка представляет собой идентификатор, оформленный по всем правилам идентификации переменных с символом «двоеточие» после него, например, пустой помеченный меткой m1 оператор:

```
m1:   ;
```

Область действия метки – функция, где эта метка определена. В случае необходимости можно использовать блок.

Циклы и переключатели можно вкладывать вдруг в друга и наиболее характерный оправданный случай использования оператора goto – выполнение прерывания (организация выхода) во вложенной структуре. Например, при возникновении грубых неисправимых ошибок необходимо выйти из двух (или более) вложенных структур (где нельзя использовать непосредственно оператор break, т.к. он прерывает только самый внутренний цикл):

```
for (...)
    for (...) {
        ...
        if (ошибка)  goto error;
    }
        ...
error:  операторы для устранения ошибки;
```

Второй оправданный случай: организация переходов из нескольких мест функции в одно, например, когда перед завершением работы функции необходимо сделать одну и ту же операцию.

Операторы continue, break и return

Оператор continue может использоваться во всех типах циклов (но не в операторе-переключателе switch). Наличие оператора continue вызывает пропуск «оставшейся» части итерации и переход к началу следующей, т.е. досрочное завершение текущего шага и переход к следующему шагу.

В циклах while и do-while это означает непосредственный переход к проверочной части. В цикле for управление передается на шаг коррекции, т.е. модификации выражения 3.

Оператор continue часто используется, когда последующая часть цикла оказывается слишком сложной, так что рассмотрение условия, обратного проверяемому, приводит к слишком высокому уровню вложенности программы.

Оператор break производит досрочный выход из цикла или оператора-переключателя switch, к которому он принадлежит, и передает управление первому оператору, следующему за текущим оператором. То есть break обеспечивает переход в точку кода программы, находящуюся за оператором, внутри которого он (break) находится.

Оператор return производит досрочный выход из текущей функции. Он также возвращает значение результата функции:

return выражение;

Выражение должно иметь скалярный тип.

Функции exit и abort

Функция exit выполняет прерывание программы и используется для нормального, корректного завершения работы программы при возникновении какой-либо внештатной ситуации, например, ошибка при открытии файла. При этом записываются все буферы в соответствующие файлы, закрываются все потоки и вызываются все зарегистрированные стандартные функции завершения.

Прототип этой функции приведен в заголовочном файле stdlib.h и выглядит так:

void exit (int exit_code);

Параметр данной функции – ненулевое целое число, передаваемое системе программирования (служебное сообщение о возникшей внештатной ситуации).

Для завершения работы программы также может использоваться функция
void abort (void);

действия которой аналогичны функции exit(3).

Стандартная библиотека языка C/C++

В любой программе кроме операторов и операций используются средства библиотек, входящих в среду программирования. Часть библиотек стандартизирована и поставляется с компилятором. Функции, входящие в библиотеку языка C/C++, намного облегчают создание программ. Расширение библиотечных файлов *.lib.

В стандартную библиотеку входят также прототипы функций, макросы, глобальные константы. Это заголовочные файлы с расширением *.h, которые хранятся в папке include и подключаются на этапе предпроцессорной обработки исходного текста программ.

Стандартные математические функции

Математические функции языка C декларированы в файлах math.h и stdlib.h.

Математические функции языка C++ декларированы в файлах cmath.h и cstdlib.h.

В приведенных здесь функциях аргументы и возвращаемый результат имеют тип double. Аргументы тригонометрических функций должны быть заданы в радианах (2π радиан = 360º).

Математическая функция	ID функции в C/C++
$\sqrt{x}$	sqrt(x)
|x|	fabs(x)
ex	exp(x)
xy	pow(x,y)
ln(x)	log(x)

lg10(x)	log10(x)
sin(x)	sin(x)
cos(x)	cos(x)
tg(x)	tan(x)
arcsin(x)	asin(x)
arccos(x)	acos(x)
arctg(x)	atan(x)
arctg(x / y)	atan2(x)
sh(x)=0.5 (ex–e-x)	sinh(x)
ch(x)=0.5 (ex+e-x)	cosh(x)
tgh(x)	tanh(x)
остаток от деления x на y	fmod(x,y)
наименьшее целое >=x	ceil(x)
наибольшее целое <=x	floor(x)

Функции вывода данных на экран

В языке C/C++ нет встроенных средств ввода/вывода данных. Ввод/вывод информации осуществляется с помощью библиотечных функций и объектов.

Декларации ***функций ввода/вывода языка Си*** приведены в заголовочном файле stdio.h.

Для вывода информации на экран монитора (дисплей) в языке Си чаще всего используются функции: printf() и puts().

Формат функции форматного вывода на экран:

Printf (управляющая строка, список объектов вывода);

В управляющей строке, заключенной в кавычки, записывают: поясняющий текст, который выводится на экран без изменения (комментарии), список модификаторов форматов, указывающих компилятору способ вывода объектов (признак модификатора формата – символ %) и специальные символы, управляющие выводом (признак – символ \).

В списке объектов вывода указываются идентификаторы печатаемых объектов, разделенных запятыми: переменные, константы или выражения, вычисляемые перед выводом.

Количество и порядок следования форматов должен совпадать с количеством и порядком следования выводимых на экран объектов.

Функция printf выполняет вывод данных в соответствии с указанными форматами, поэтому формат может использоваться и для преобразования типов выводимых объектов.

Если признака модификации (%) нет, то вся информация выводится как комментарии.

Основные модификаторы формата:

%d (%i) – десятичное целое число;

%c – один символ;

%s – строка символов;

%f – число с плавающей точкой, десятичная запись;
%e – число с плавающей точкой, экспоненциальная запись;
%g – используется вместо f, e для исключения незначащих нулей;
%o – восьмеричное число без знака;
%x – шестнадцатеричное число без знака.

Для чисел long добавляется символ l, например, %ld – длинное целое, %lf – число вещественное с удвоенной точностью – double.

Если нужно напечатать сам символ %, то его нужно указать 2 раза:
printf ("Только %d%% предприятий не работало. \n",5);

Получим: Только 5% предприятий не работало.

Управляют выводом специальные последовательности символов: \n – новая строка; \t – горизонтальная табуляция; \b – шаг назад; \r – возврат каретки; \v – вертикальная табуляция; \\ – обратная косая; \' – апостроф; \" – кавычки; \0 – нулевой символ (пусто).

Пример.

```
#define  PI   3.14159
. . .
int number = 5;
float bat = 255;
int cost = 11000;
. . .
printf(" %d студентов съели %f бутербродов. \n", number, but);
printf(" Значение числа pi равно %f. \n", pi);
printf(" Стоимость этой вещи %d %s. \n", cost, "Руб.");
. . .
```

В модификаторах формата функции printf после символа % можно указывать число, задающее минимальную ширину поля вывода, например, %5d – для целых, %4.2f – для вещественных – две цифры после запятой для поля шириной 4 символа. Если указанных позиций для вывода целой части числа не хватает, то происходит автоматическое расширение.

Если после «%» указан знак «минус», то выводимое значение будет печататься с левой позиции поля вывода, заданной ширины, например: % – 10d.

Использование функции printf для преобразования данных:

1) printf("%d", 336.65); получим: 336;
2) printf("%o", 336); получим: 520, т.е. $5*8^2+2*8+0*1 = 336$;
3) printf("%x", 336); получим: 150 (шестнадцатеричное).

Можно использовать функцию printf для нахождения кода ASCII некоторого символа:

printf (" %c – %d\n", 'a', 'a');

получим десятичный код ASCII символа a: a – 65 .

Функция **puts(ID строки);** выводит на экран дисплея строку символов, автоматически добавляя к ней символ перехода на начало новой строки (\n).

Аналогом такой функции будет: printf("%s \n", ID строки);

Функция putchar() выдает на экран дисплея один символ без добавления символа '\n'.

Функции ввода информации

Функция, предназначенная для форматированного ввода исходной информации с клавиатуры:

scanf (управляющая строка, список адресов объектов ввода);

в управляющей строке указываются только модификаторы форматов, количество и порядок следования которых должны совпадать с количеством и порядком следования вводимых объектов, а тип данных будет преобразовываться в соответствии с модификаторами.

Список объектов ввода представляет собой адреса переменных, разделенные запятыми, т.е. для ввода значения переменной перед ее идентификатором указывается символ &, обозначающий операцию «взять адрес».

Если нужно ввести значение строковой переменной, то использовать символ & не нужно, т.к. строка – это массив символов, а ID массива является адресом его первого элемента.

Например:

```
int course;
double grant;
char name[20];
printf (" Укажите курс, стипендию, имя \n ");
scanf ("%d %lf %s", &course, &grant,  name);
```

Вводить данные с клавиатуры можно как в одной строке через пробелы, так и в форме разных строк, нажимая после ввода текущего объекта клавишу Enter.

Функция scanf() использует практически тот же набор модификаторов форматов, что и printf(); отличия от функции вывода следующие:

отсутствует формат %g,

форматы %e,%f – эквивалентны.

Для ввода коротких целых чисел введен модификатор формата %h.

Внимание! Функцией scanf() по формату %s строка вводится только до первого пробела.

Для ввода фраз, состоящих из слов, разделенных пробелами, используется функция **gets (ID строковой переменной);**

Символы вводятся при помощи функции **getch().** Причем простой ее вызов организует паузу, при которой система программирования приостановит выполнение программы и будет ждать нажатия любой клавиши. Так поступают в том случае, когда нужно просмотреть какие-то результаты работы, при выводе их на экран монитора.

Если же использовать ее в правой части операции присваивания, например:

```
char c;
        . . .
c = getch();
```

то символьная переменная *c* получит значение кода нажатой клавиши.

С началом работы любой программы автоматически открываются стандартные потоки для ввода (stdin) и вывода данных (stdout), которые по умолчанию связаны с клавиатурой и экраном монитора соответственно.

Внимание! Ввод данных функциями gets(), getch() выполняется с использованием потока stdin. Если указанная функция не выполняет своих действий (проскакивает), перед использованием необходимо очистить поток (буфер) ввода с помощью функции fflush(stdin);

Ввод-вывод с помощью потоков STL (C++)

Для использования консольного ввода-вывода с помощью потоков (stream) STL в программу необходимо включить заголовочный файл <iostream>:

```
#include <iostream>
using namespace std;
int main() {
   cout << "Hello, world!\n";
}
```

При запуске консольного приложения неявно открываются четыре потока: *cin* — для ввода с клавиатуры, *cout* — для буферизованного вывода на монитор, *cerr* — для небуферизованного вывода на монитор сообщений об ошибках и *clog* — буферизованный аналог *cerr*. Эти четыре символа определены посредством <iostream>.

Потоки *cin*, *cout* и *cerr* соответствуют потокам *stdin*, *stdout* и *stderr* соответственно.

Приложение 1

Операции языка C/C++

Операции приведены в порядке убывания приоритета, операции с разными приоритетами разделены чертой.

Операция	Краткое описание	Использование	Порядок выполнения
	Первичные (унарные) операции		
.	Доступ к члену	*объект* **.** *член*	Слева направо
->	Доступ по указателю	*указатель* -> *член*	
[ ]	Индексирование	*переменная* [*выражение*]	
( )	Вызов функции	ID_*функции*(*список*)	
	Унарные операции		
++	Постфиксный инкремент	*lvalue*++	Справа налево
--	Постфиксный декремент	*lvalue*--	
sizeof	Размер объекта (типа)	sizeof(ID *или тип*)	
++	Префиксный инкремент	++*lvalue*	
--	Префиксный декремент	--*lvalue*	
~	Побитовое НЕ	~*выражение*	
!	Логическое НЕ	!*выражение*	
– (+)	Унарный минус (плюс)	– (+)*выражение*	
*****	Разадресация	**выражение*	
&	Адрес	&*выражение*	
()	Приведение типа	(*тип*)*выражение*	
	Бинарные и тернарная операции		
*****	Умножение	*выражение* * *выражение*	Слева направо
/	Деление	*выражение* / *выражение*	
%	Получение остатка	*выражение* % *выражение*	
+	Сложение	*выражение* + *выражение*	
–	Вычитание	*выражение* – *выражение*	
<<	Сдвиг влево	*выражение* << *выражение*	
>>	Сдвиг вправо	*выражение* >> *выражение*	
<	Меньше	*выражение* < *выражение*	
<=	Меньше или равно	*выражение* <= *выражение*	
>	Больше	*выражение* > *выражение*	
>=	Больше или равно	*выражение* >= *выражение*	
==	Равно	*выражение* == *выражение*	
!=	Не равно	*выражение* != *выражение*	
&	Побитовое И	*выражение* & *выражение*	
^	Побитовое исключ. ИЛИ	*выражение* ^ *выражение*	
|	Побитовое ИЛИ	*выражение* | *выражение*	Слева направо
&&	Логическое И	*выражение* && *выражение*	
||	Логическое ИЛИ	*выражение* || *выражение*	

Операция	**Краткое описание**	**Использование**	**Порядок выполнения**
?:	Условная операция (***тернарная***)	*выражение* ? *выражение* : *выражение*	Справа налево
=	Присваивание	*lvalue* = *выражение*	
*=	Умножение с присваиванием	*lvalue* *= *выражение*	
/=	Деление с присваиванием	*lvalue* /= *выражение*	
%=	Остаток от деления с присваиванием	*lvalue* %= *выражение*	
+=	Сложение с присваиванием	*lvalue* += *выражение*	
- =	Вычитание с присваиванием	*lvalue* -= *выражение*	
<<=	Сдвиг влево с присваиванием	*lvalue* <<= *выражение*	
>>=	Сдвиг вправо с присваиванием	*lvalue* >>= *выражение*	
&=	Поразрядное И с присваиванием	*lvalue* &= *выражение*	
|=	Поразрядное ИЛИ с присваиванием	*lvalue* |= *выражение*	
^=	Поразрядное ИСКЛЮЧАЮЩЕЕ ИЛИ с присваиванием	*lvalue* ^= *выражение*	
,	Последовательное вычисление	*выражение*, *выражение*	Слева направо