

PROGRAMMING  
LANGUAGE

C++

POINTER  
FUNCTIONS  
OPERATORS  
OOPS  
BASICS  
INHERITANCE  
ARRAY  
STRINGS  
CONTAINERS  
FUNCTION OVERLOAD

# Процедурное программирование на языке C++

Виталий Полянский,  
преподаватель кафедры  
«Разработка программного обеспечения»  
Одесского филиала Компьютерной Академии  
ШАГ

**Алгоритм – последовательность  
действий, направленных на  
достижение конкретного результата**

**Программа – последовательность команд (инструкций), выполняемых процессором и описывающих алгоритм на языке программирования**

**Данные – информация, которая  
обрабатывается в процессе  
выполнения программы**

# Современные языки программирования



# Машинный код



# Ассемблерный код

```
cs:010A 55          push    bp
cs:010B 89E5          mov     bp,sp
cs:010D 83EC04       sub     sp,0004
cs:0110 51          push    cx
cs:0111 8B4E04       mov     cx,[bp+04]
cs:0114 E31F        jcxz   0135
cs:0116 49          dec     cx
cs:0117 7417        je     0130
cs:0119 51          push    cx
cs:011A E8EDFF       call   010A
cs:011D 8946FC       mov     [bp-04],ax
cs:0120 8956FE       mov     [bp-02],dx
cs:0123 49          dec     cx
cs:0124 51          push    cx
cs:0125 E8E2FF       call   010A
cs:0128 0346FC       add     ax,[bp-04]
cs:012B 1356FE       adc     dx,[bp-02]
cs:012E EB08        jmp     0138
cs:0130 B80100       mov     ax,0001
cs:0133 EB02        jmp     0137
cs:0135 31C0       xor     ax,ax
cs:0137 99          cwd
cs:0138 59          pop     cx
cs:0139 C9          leave
cs:013A C20200       ret     0002
```

# Код на языке C

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  void main()
6  {
7      char arr[101];
8      int help = 0;
9      int i = 0;
10
11     scanf("%s", &arr);
12     printf("Before: %s\n", arr);
13
14     while(arr[i])
15     {
16         if((arr[i] < '0') || (arr[i] > '9'))
17         {
18             for(int j = i; arr[j]; j++)
19                 arr[j] = arr[j + 1];
20             i = 0;
21         }
22         i++;
23     }
24     printf("After: %s", arr);
25 }
```



# Код на языке C#

```
public class Books
{
    public static IEnumerable<Book> GetBooks()
    {
        var books = new List<Book>();

        // Book loading would go here...

        return books;
    }
}
```

# Основные преимущества языков низкого уровня

- Прямой доступ к аппаратным ресурсам
- Малый размер исполняемого файла
- Высокая скорость выполнения

Основной недостаток языков  
низкого уровня – сложность  
программирования из-за огромных  
размеров исходного кода

Главное преимущество языков  
высокого уровня – быстрота  
разработки сложных программных  
продуктов

# Основные недостатки языков высокого уровня

- Ограниченный доступ к аппаратным ресурсам
- Большой размер исполняемого файла
- Сравнительно невысокая скорость

# Основные этапы разработки программы

- Постановка задачи
- Проектирование
- Кодирование

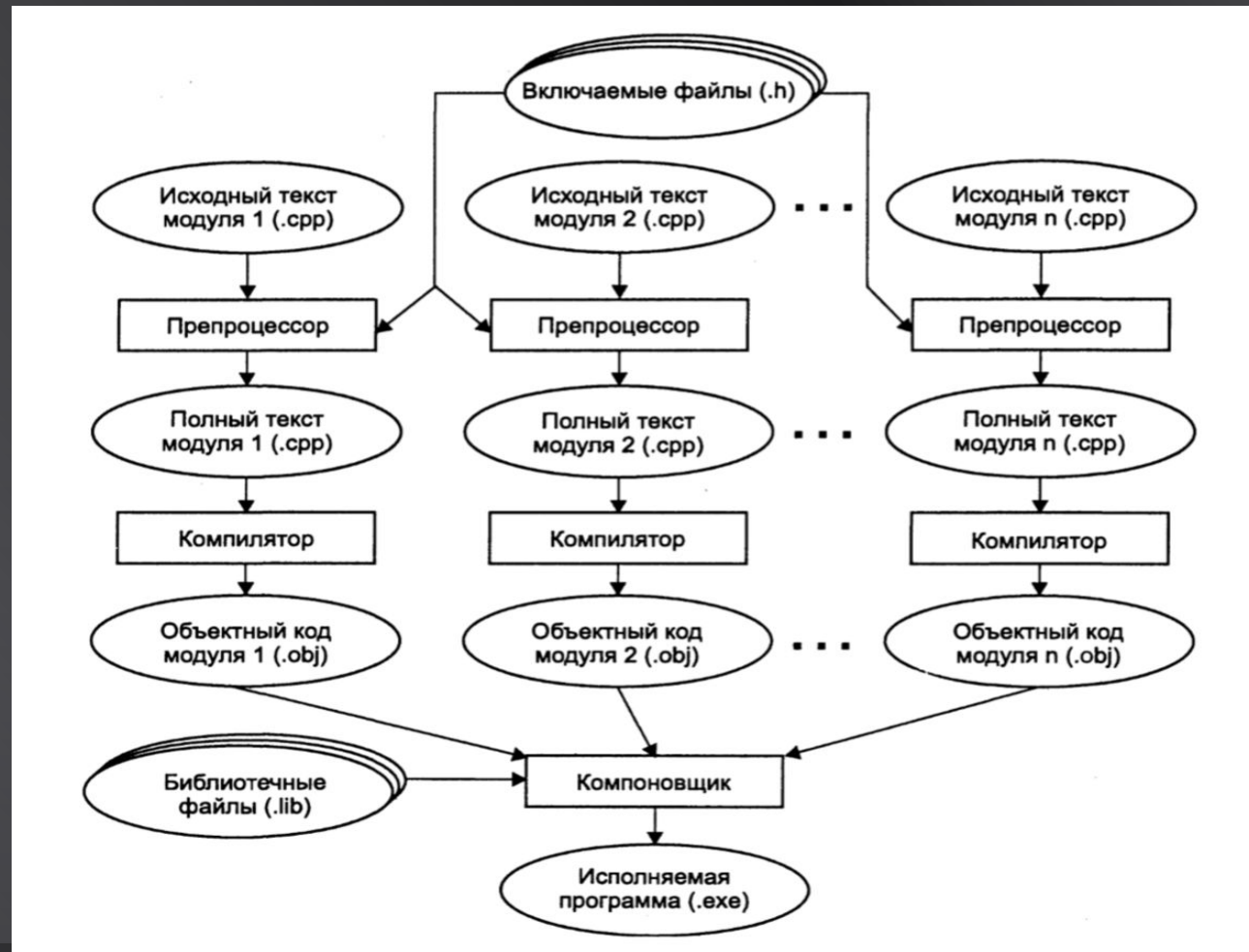
На этапе постановки задачи  
программист выясняет у  
заказчика требования к  
программе

На этапе проектирования  
программист выбирает алгоритм  
для написания программы



На этапе кодирования программист переводит алгоритм на какой-либо язык программирования

# Этапы создания исполняемого файла (.exe)



**Препроцессор** включает в исходный текст программы заголовочные файлы, содержащие описания используемых элементов

**Компилятор выявляет  
синтаксические ошибки и, в случае  
их отсутствия, строит объектный  
модуль (.obj)**

**Компоновщик формирует  
исполняемый модуль программы  
(.exe) из объектных модулей (.obj) и  
библиотечных файлов (.lib)**

# Литералы

- Строковый
- Символьный
- Целочисленный
- Вещественный
- Логический

**Строковый литерал –  
последовательность символов,  
заключенная в двойные кавычки**

**“The C++ Programming Language”**

**Символьный литерал – одиночный символ, заключенный в одинарные кавычки**

**'!'    'A'    '8'**



**Целочисленный литерал – любое  
положительное или отрицательное  
целое число**

**-15    100    0xFF**

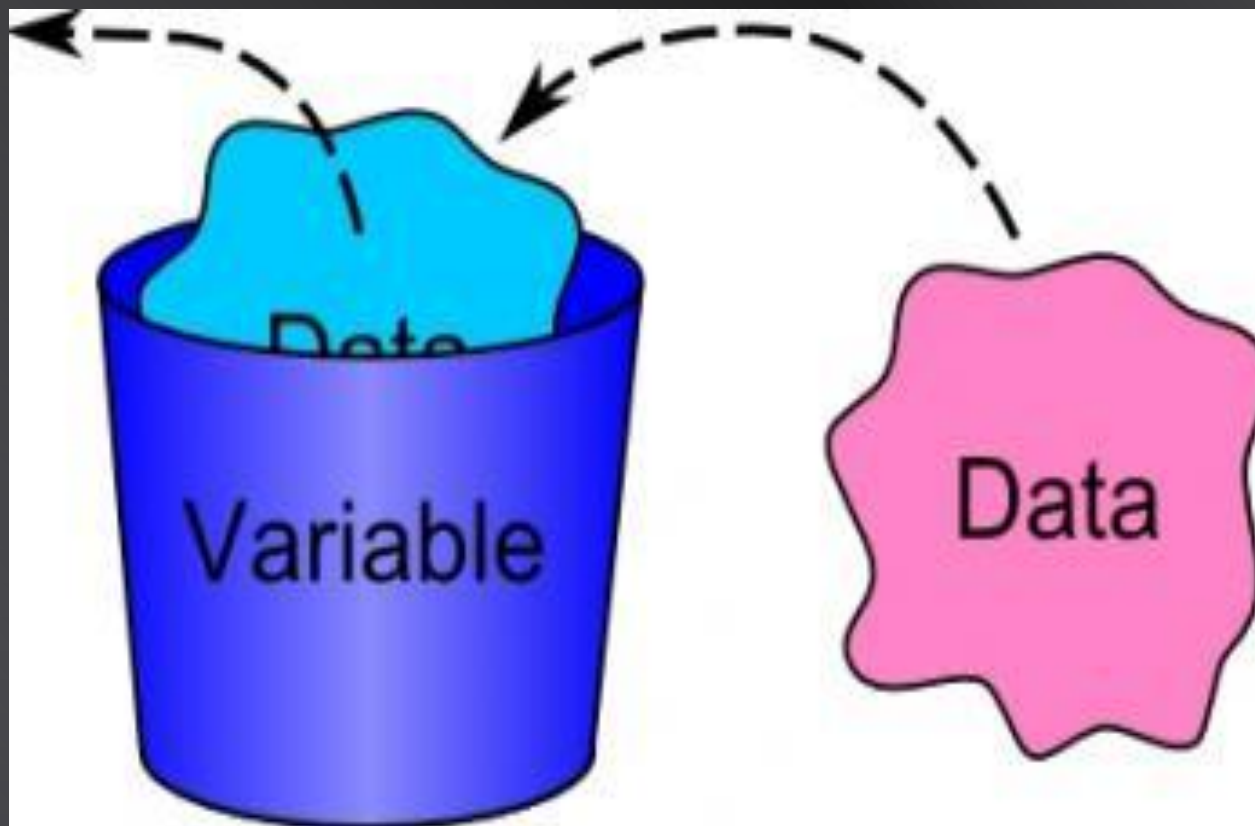
**Вещественный литерал – дробное  
число, представленное в форме с  
десятичной точкой либо в  
экспоненциальной форме**

**-5.7 5e10 11e-3**

# Логический литерал

- `true` (истина)
- `false` (ложь)

# Понятие переменной



**Переменная –**

**именованная область оперативной  
памяти, предназначенная для  
хранения изменяемого значения  
указанного типа**

# Типы данных

- Целочисленный
- Символьный
- Вещественный
- Логический

# Целочисленные типы

- signed short (2 байта)  
-32768 ... 32767
- unsigned short (2 байта)  
0 ... 65535
- signed int (4 байта – равен размеру машинного слова)  
-2 147 483 648 ... 2 147 483 647

# Целочисленные типы

- unsigned int (4 байта – равен размеру машинного слова)

0 ... 4 294 967 295

- signed long (4 байта)

-2 147 483 648 ... 2 147 483 647

- unsigned long (4 байта)

0 ... 4 294 967 295



# Целочисленные типы

- signed long long (8 байт)

-9 223 372 036 854 775 808 ...

9 223 372 036 854 775 807

- unsigned long long (8 байт)

0 ... 18 446 744 073 709 551 615

Тип данных `int` строго не определён  
стандартом языка и должен быть  
равен размеру **машинного слова**

Размер машинного слова  
определяется архитектурой  
(разрядностью) операционной  
системы

Однако в соответствии со стандартом языка тип данных `int` не должен быть меньше `short` (2 байта) и не должен превышать `long` (4 байта)

# Символьные типы

- signed char (1 байт)  
-128 ... 127
- unsigned char (1 байт)  
0 ... 255

Тип данных `char` представляет один символ в кодировке ASCII

**ASCII (American standard code  
for information interchange) — название  
таблицы (кодировки), в которой  
символам сопоставлены числовые  
коды**

Код	Символ	Код	Символ	Код	Символ	Код	Символ	Код	Символ	Код	Символ
0		43	+	86	V	128	A	171	л	214	Г
1	☉	44	,	87	W	129	Б	172	м	215	Г
2	☉	45	-	88	X	130	В	173	н	216	Г
3	▼	46	.	89	Y	131	Г	174	о	217	Г
4	♦	47	/	90	Z	132	Д	175	п	218	Г
5	♣	48	0	91	[	133	Е	176	☄	219	■
6	♣	49	1	92	\	134	Ж	177	☄	220	■
7	•	50	2	93	]	135	З	178	■	221	■
8	□	51	3	94	^	136	И	179		222	■
9		52	4	95	_	137	И	180	┆	223	■
10		52	5	96	`	138	К	181	┆	224	р
11	♂	54	6	97	a	139	Л	182	┆	225	с
12	♀	55	7	98	b	140	М	183	┆	226	г
13		56	8	99	c	141	Н	184	┆	227	у
14	♫	57	9	100	d	142	О	185	┆	228	Ф
15	☼	58	:	101	e	143	П	186		229	х
16	▶	59	;	102	f	144	Р	187	┆	230	ц
17	◀	60	<	103	g	145	С	188	┆	231	ч
18	↑	61	=	104	h	146	Т	189	┆	232	ш
19	!!	62	>	105	i	147	У	190	┆	233	щ
20	¶	63	?	106	j	148	Ф	191	┆	234	Ъ
21	§	64	@	107	k	149	Х	192	┆	235	Ы
22	—	65	A	108	l	150	Ц	193	┆	236	ь
23	↑	66	B	109	m	151	Ч	194	┆	237	э
24	↑	67	C	110	n	152	Ш	195	┆	238	Ю
25	↓	68	D	111	o	153	Щ	196	—	239	я
26	→	69	E	112	p	154	Ъ	197	┆	240	Е
27	←	70	F	113	q	155	Ы	198	┆	241	ё
28	└	71	G	114	r	156	Ь	199	┆	242	Є
29	↔	72	H	115	s	157	Э	200	┆	243	є
30	▲	73	I	116	t	158	Ю	201	┆	244	І
31	▼	74	J	117	u	159	Я	202	┆	245	і
32		75	K	118	v	160	а	203	┆	246	у
33	!	76	L	119	w	161	б	204	┆	247	ў
34	"	77	M	120	x	162	в	205	==	248	°
35	#	78	N	121	y	163	г	206	┆	249	·
36	\$	79	O	122	z	164	д	207	┆	250	·
37	%	80	P	123	{	165	е	208	┆	251	ƒ
38	&	81	Q	124		166	ж	209	┆	252	№
39	'	82	R	125	}	167	з	210	┆	253	▣
40	(	83	S	126	~	168	и	211	┆	254	■
41	)	84	T	127	◊	169	й	212	┆	255	
42	^	85	U			170	к	213	┆		



Например, символу 'D'  
соответствует ASCII-код 68, а  
символу 'd' – ASCII-код 100

# Вещественные типы

- float (4 байта)  
3.4E-38 ... 3.4E+38
- double (8 байт)  
1.7E-308 ... 1.7E+308

# Логический тип

- bool (1 байт)  
true false

Для того чтобы использовать в  
программе переменную, ее  
необходимо объявить

Синтаксис объявления переменных:

**тип\_переменной имя\_переменной;**

Имя переменной называется  
идентификатором

# Правила именования идентификаторов:

- Имя переменной не может начинаться с цифры
- Имя переменной может содержать буквы, цифры и знак подчеркивания «\_»
- Имя переменной не может являться ключевым или служебным словом
- Имя переменной должно быть уникальным
- Имя переменной должно быть

```
// Объявление одной переменной
```

```
int nCount = 0;
```

```
// Объявление нескольких переменных
```

```
double dblNumber1, dblNumber2, dblNumber3;
```

```
// Инициализация - присвоение значения на этапе объявления.
```

```
char chSymbol1 = 'w';
```

```
// Списковая инициализация переменной
```

```
char chSymbol2{ 's' };
```



**Константа –**

**именованная область оперативной  
памяти, предназначенная для  
хранения постоянного значения  
указанного типа**

# Синтаксис объявления константы:

```
const ТИП_КОНСТАНТЫ ИМЯ_КОНСТАНТЫ =  
    значение;
```

```
const double PI = 3.14159265;
```

**Литералы, рассмотренные ранее,  
представляют собой константы,  
непосредственно включаемые в  
текст программы**

# Литералы отличаются от констант тем, что они не имеют идентификаторов

```
const double PI = 3.14159265;
```

```
cout << "PI = " << PI; // константа
```

```
cout << "PI = " << 3.14159265; // литерал
```

# Ввод данных



var\_A

```
cin >> var_A;
```

Синтаксис ввода данных:

```
cin >> имя_переменной;
```

```
int nValue;  
cout << "Enter number: ";  
cin >> nValue;  
  
float fltNumber1, fltNumber2;  
cout << "Enter two numbers: ";  
cin >> fltNumber1 >> fltNumber2;
```

# Понятие оператора





**Оператор – это конструкция языка программирования, которая выполняет определённое действие над аргументами (операндами)**

**Операнд** - это аргумент оператора,  
то есть то значение, над которым  
оператор выполняет действие

В зависимости от количества операндов операторы бывают:

- Унарные
- Бинарные
- Тернарные

# Оператор присваивания

Оператор  
присваивания

x = 5;



Синтаксис оператора присваивания:

**имя\_переменной = выражение;**

```
int nResult;  
nResult = 4;
```

```
cout << "nResult = " << nResult << endl;
```

```
int nValue = 1;  
nResult = nValue;
```

```
cout << "nResult = " << nResult << endl;
```

# Множественные присваивания - присваивания одного и того же значения нескольким переменным одновременно

```
double dblNumber1;  
double dblNumber2;  
dblNumber1 = dblNumber2 = 3.5;  
  
cout << "dblNumber1 = " << dblNumber1 << endl;  
cout << "dblNumber2 = " << dblNumber2 << endl;
```

# Арифметические операторы

- Оператор сложения (бинарный)
- Оператор вычитания (бинарный)
- Оператор умножения (бинарный)
- Оператор деления (бинарный)



# Арифметические операторы

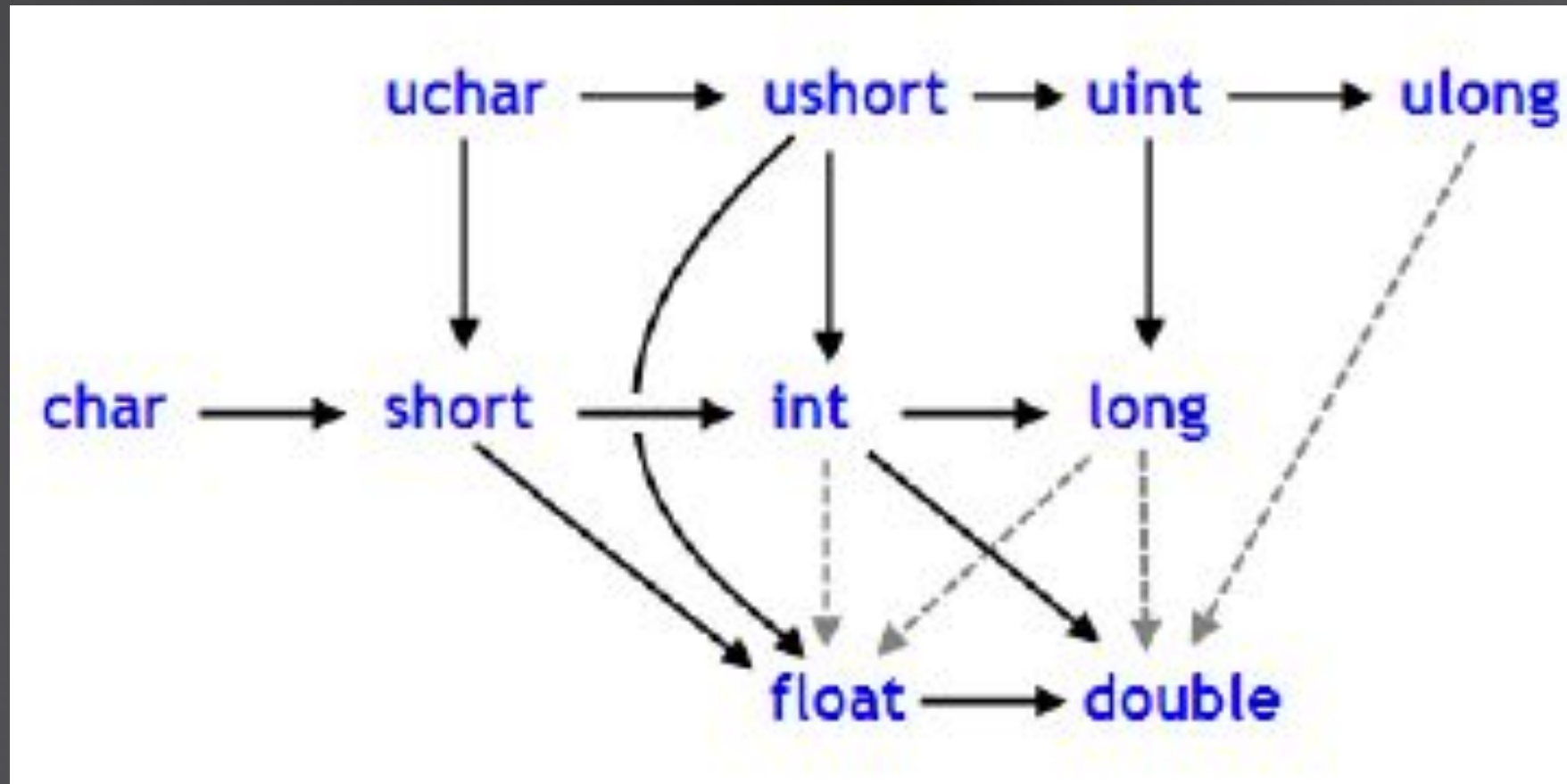
- Оператор остаток от деления (бинарный)
- Оператор минус (унарный)
- Оператор инкремент (унарный)
- Оператор декремент (унарный)

# Сокращённые формы операторов:

`+=` `-=` `*=` `/=` `%=`

```
a += b; // То же самое, что a = a + b.  
a -= b; // То же самое, что a = a - b.  
a *= b; // То же самое, что a = a * b.  
a /= b; // То же самое, что a = a / b.  
a %= b; // То же самое, что a = a % b.
```

# Приведение типов



**Выражение –**

**конструкция, состоящая из  
операндов, объединенных знаками  
операций**

Результат вычисления выражения  
характеризуется значением и типом

В выражение могут входить  
операнды различных типов

Если операнды имеют одинаковый тип, то результат операции будет иметь тот же тип

```
int number1 = 7;  
int number2 = 2;  
int nAddition = number1 + number2;  
cout << "Result = " << nAddition << endl;
```

Если операнды разных типов, то они должны быть приведены к одному типу перед вычислением выражения

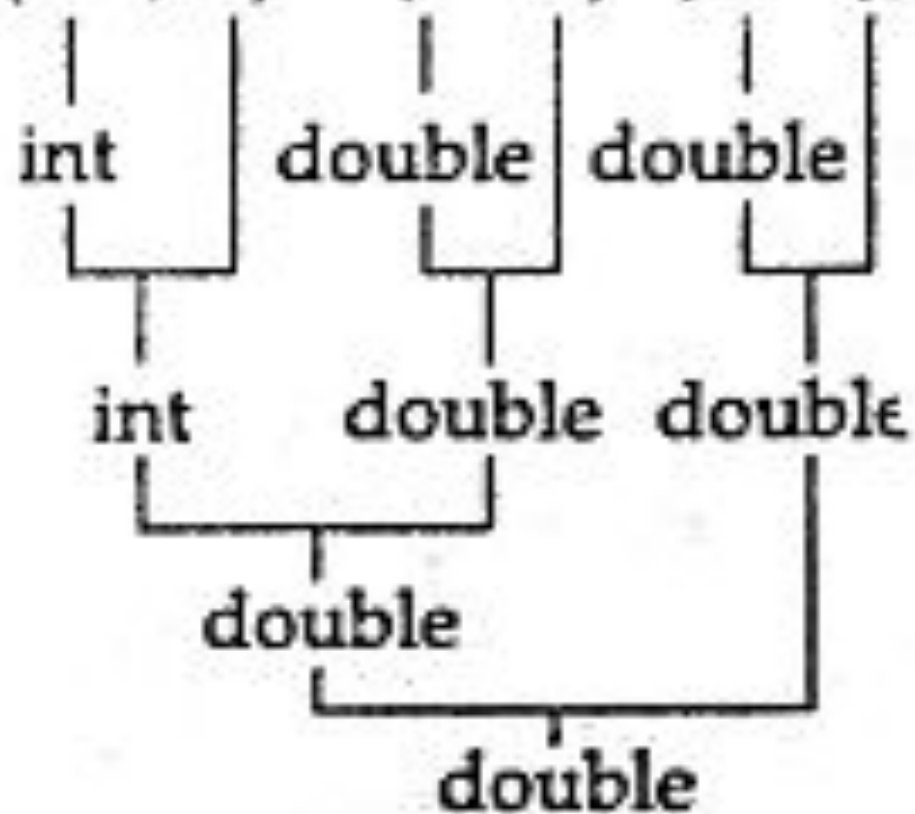


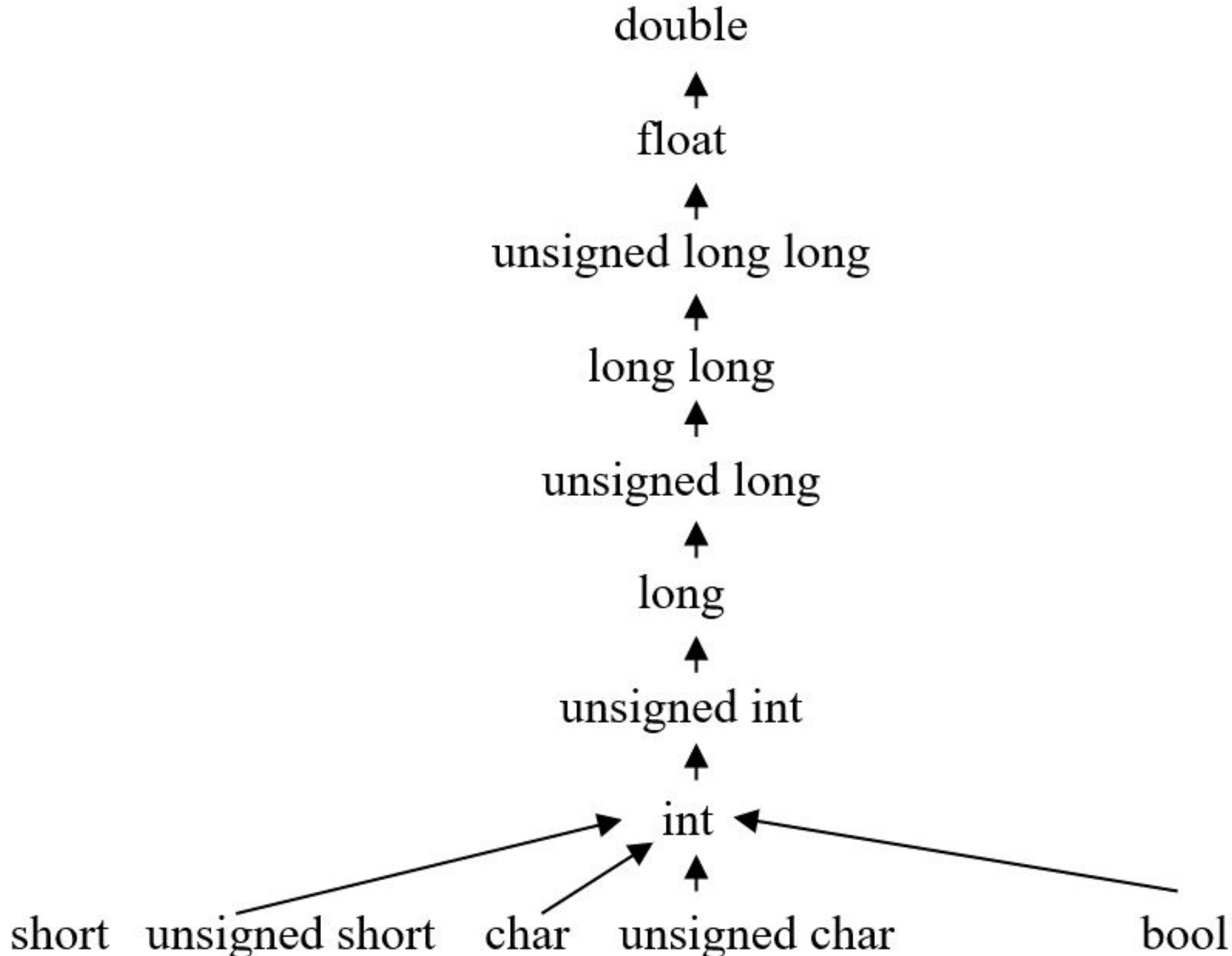
В этом случае выполняется неявное  
(автоматическое) расширяющее  
преобразование типов операндов по  
определенным правилам

Эти правила обеспечивают  
преобразование более коротких  
типов в более длинные для  
сохранения значимости и точности

```
char ch;  
int i;  
float f;  
double d;
```

```
result = (ch / i) + (f * d) - (f + i);
```





Неявное расширяющее преобразование типа может также происходить при присваивании переменной результата выражения

```
long long result;  
result = 'A' + true + 2;
```

Неявное преобразование типа при  
присваивании переменной  
результата выражения может быть  
сужающим

```
int Radius;  
Radius = 3.5;
```

# Сужающие преобразования бывают:

- с потерей точности (данных)
- с переполнением разрядной сетки
- с изменением интерпретации внутреннего представления данных

Приведение типов может быть  
ЯВНЫМ



Явное приведение типов  
указывается программистом в коде  
программы

Синтаксис явного приведения типа:

**(тип) выражение;**

```
// Сужающее преобразование типов
```

```
short nSmall = 1;
```

```
int nBig = 2;
```

```
nSmall = short(nBig);
```

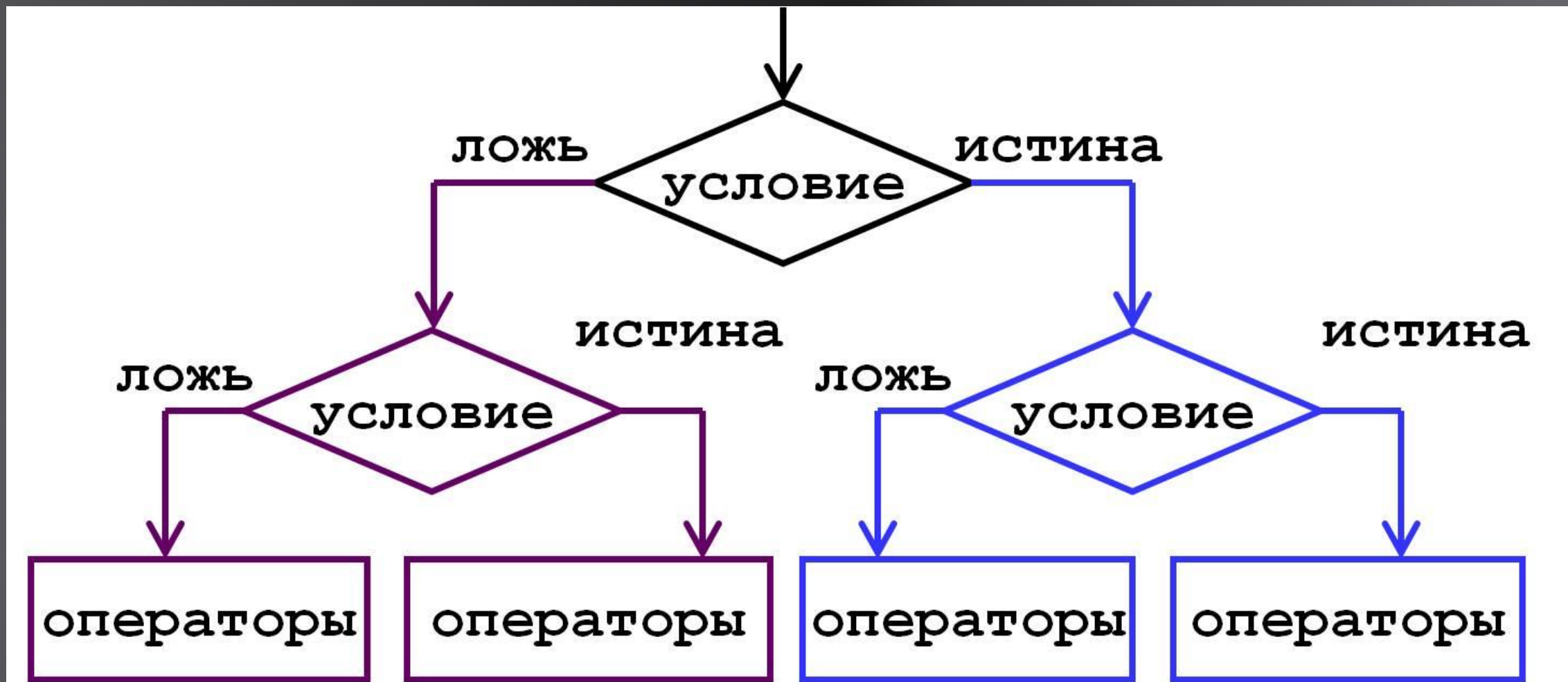
```
// Расширяющее преобразование типов
```

```
float fltSmall = 0.4f;
```

```
double dblBig = 4.4;
```

```
dblBig = (double)fltSmall;
```

# Структура ветвления



# Операторы отношения

- == (равно)
- != (не равно)
- > (больше)
- >= (больше либо равно)
- < (меньше)
- <= (меньше либо равно)

Операторы отношения  
предназначены для составления  
логических выражений

Результат вычисления любого  
логического выражения –  
ИСТИНА или ЛОЖЬ

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

int main()
{
    int a = 10, b = 5;
    bool result;
    result = a == b;    // false
    result = a != b;    // true
    result = a > b;     // true
    result = a >= b;    // true
    result = a < b;     // false
    result = a <= b;    // false

    return 0;
}
```



# Логические операторы

- && (логическое умножение, логическое И)
- || (логическое сложение, логическое ИЛИ)
- ! (логическое отрицание, логическое НЕ)

Логические операторы  
предназначены для объединения  
логических выражений

# Логические операции:

`&&` - логическое И (AND),

`||` - логическое ИЛИ (OR),

`!` - логическое НЕ (NOT).

A	B	A&&B	A  B	!A
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

# Логическое «ИЛИ»

`5 == 5 || 5 == 9`

`// true, потому что первое выражение  
true`

`5 > 3 || 5 > 10`

`// true, потому что первое выражение  
true`

`5 > 8 || 5 < 10`

`// true, потому что второе выражение  
true`

`5 < 8 || 5 > 2`

`// true, потому что оба выражения true`

# Логическое «И»

5 == 5 && 4 == 4

// true, потому что оба выражения true

5 == 3 && 4 == 4

// false, потому что первое выражение  
false

5 > 3 && 5 > 10

// false, потому что второе выражение  
false

5 < 8 && 5 > 2

// true, потому что оба выражения true

5 > 8 && 5 < 2

# Логическое «НЕ»

```
!(10 != 10)
```

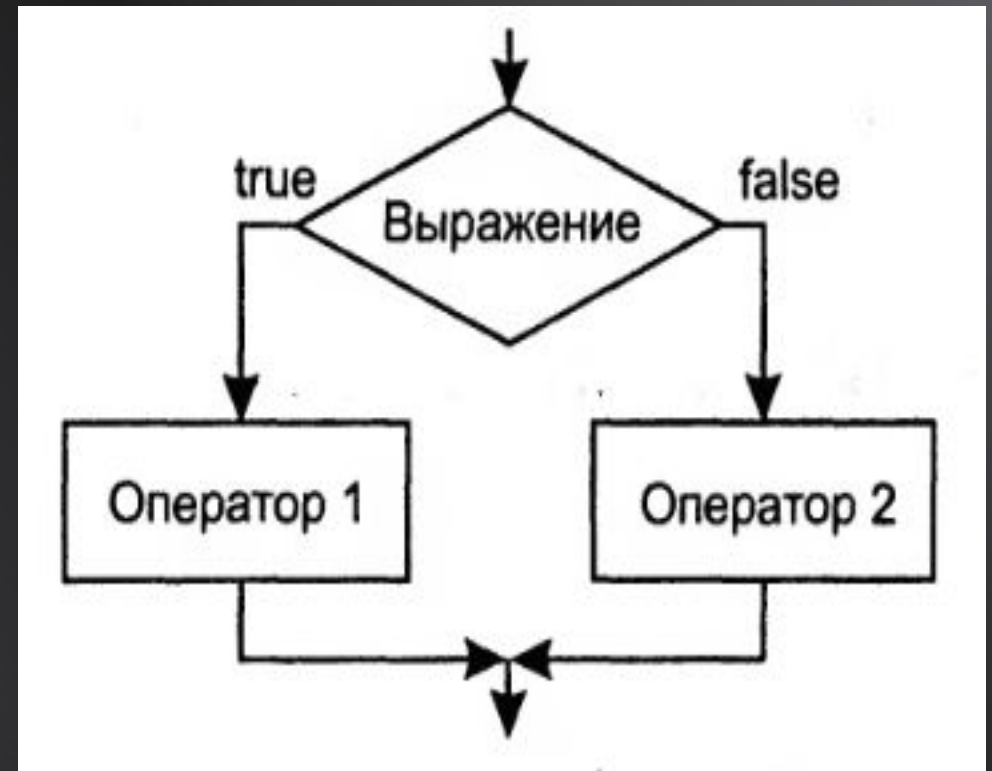
```
// true, потому что выражение false
```

```
!(5 > 3)
```

```
// false, потому что выражение true
```

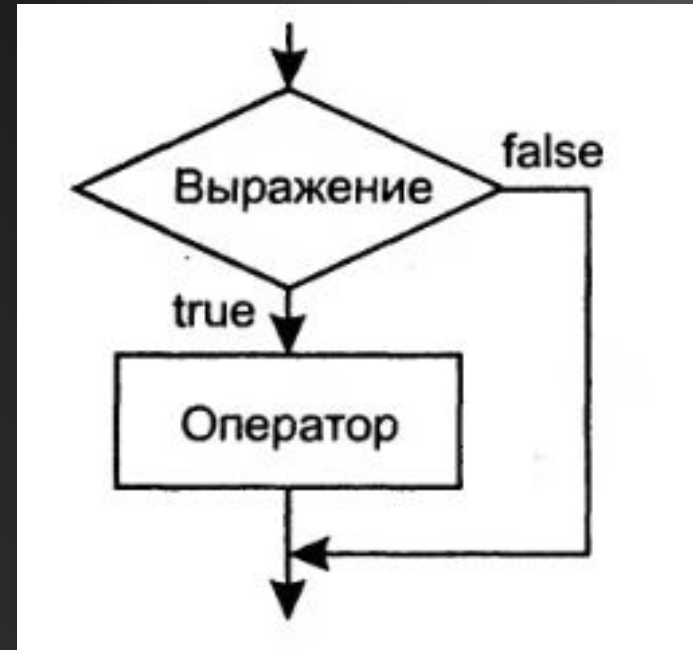
# Оператор ветвления «if»

```
if (выражение)
{
    // оператор_1;
}
else
{
    // оператор_2;
}
```



# Оператор ветвления «if»

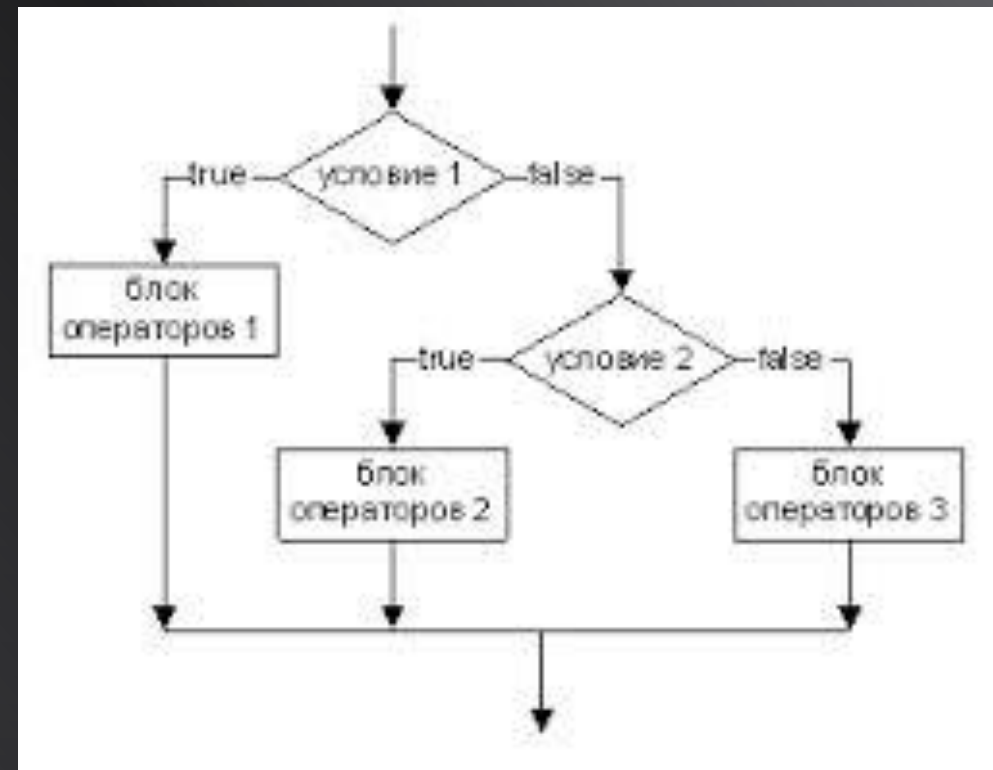
```
if (выражение)  
{  
    // оператор;  
}
```





# Оператор ветвления «if»

```
if (выражение_1)
{ // оператор_1; }
else if (выражение_2)
{ // оператор_2; }
else
{ // оператор_3; }
```



# Таблица приоритетов операций

Приоритет	Оператор	Описание
1	::	Область видимости
2	++ --	Суффиксальный/постфиксный инкремент и декремент
	()	Вызов функции
	[]	Обращение к массиву по индексу
	. ->	Выбор элемента по ссылке Выбор элемента по указателю
3	++ --	Префиксный инкремент и декремент
	+ -	Унарный плюс и минус
	! ~	Логическое НЕ и побитовое НЕ
	(type)	Приведение к типу type
	*	Indirection (разыменование)
	&	Адрес
	sizeof	Размер
	new, new[] delete, delete[]	Динамическое выделение памяти Динамическое освобождение памяти
4	.* ->*	Указатель на член
5	* / %	Умножение, деление и остаток
6	+ -	Сложение и вычитание
7	<< >>	Побитовый сдвиг влево и сдвиг вправо
8	< <=	For relational operators < and ≤ respectively
	> >=	For relational operators > and ≥ respectively
9	== !=	For relational = and ≠ respectively
10	&	Побитовое И
11	^	Побитовый XOR (исключающее или)
12		Побитовое ИЛИ (inclusive or)
13	&&	Логическое И
14		Логическое ИЛИ
15	?:	Тернарное условие
	=	Прямое присваивание (предоставляемое по умолчанию для C++ классов)
	+= -=	Присвоение с суммированием и разностью
	*= /= %=	Присвоение с умножением, делением и остатком от деления
	<<= >>=	Assignment by bitwise left shift and right shift
&= ^=  =	Assignment by bitwise AND, XOR, and OR	
16	throw	Throw оператор (выброс исключений)
17	,	Запятая

# Структура повторения



Структура повторения позволяет программисту определить действие, которое должно повторяться, пока некоторое условие остается ИСТИННЫМ

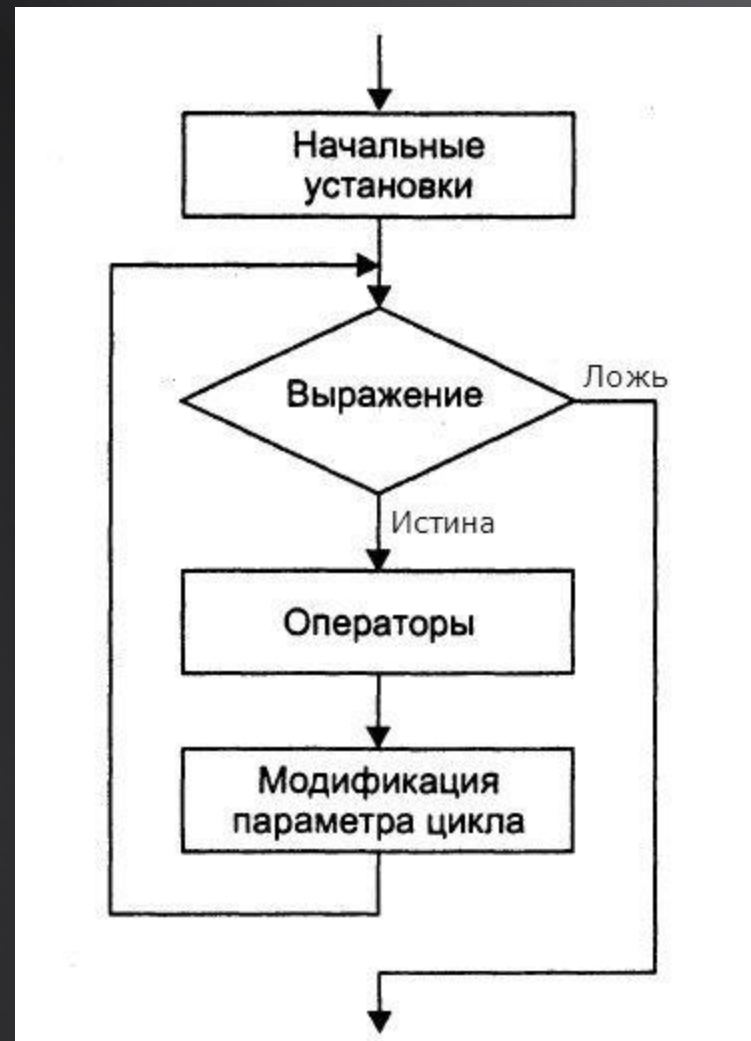
В языке программирования C++  
структура повторения реализуется с  
помощью оператора цикла

# Виды циклов

- Цикл с предусловием (`while`)
- Цикл с постусловием (`do while`)
- Параметрический цикл (`for`)

# Цикл с предусловием while

```
while (выражение)
{
    // оператор
    // или группа
    // операторов
}
```



# Цикл с постусловием do while

```
do  
{  
    // оператор  
    // или группа  
    // операторов  
}  
while (выражение);
```





# Параметрический цикл for



# Массивы



# **Массив –**

**это совокупность элементов,  
объединенных под общим именем  
и имеющих один и тот же тип  
данных**

Каждый элемент массива  
является самостоятельной  
единицей и представляет  
собой переменную или  
константу

Все элементы массива  
располагаются в памяти  
последовательно друг за другом и  
имеют свой индекс – смещение  
относительно начала массива



# Объявление массива

Синтаксис объявления массива:

тип\_данных имя\_массива[количество\_элементов];

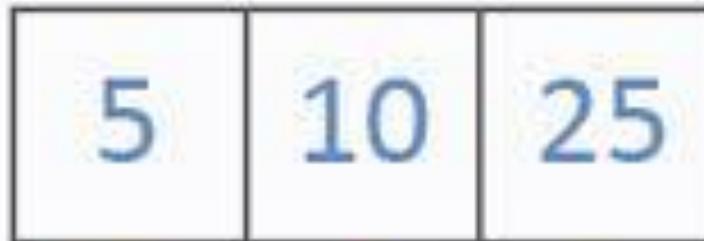
```
int numbers[4];
```

```
const int nSize = 6;  
double arr[nSize];
```

# Инициализация массива

тип\_данных имя\_массива[количество элементов] =  
{значение1, значение2, ... значение n};

```
int array1D[3]={5,10,25};
```



[0]

[1]

[2]

Количество элементов массива  
можно не указывать. Размер  
массива определяется исходя из  
числа элементов в списке  
инициализации.

```
int arrayInit[] = { 2, 33, 4 }; // массив из трех элементов
```



Если значений в списке  
инициализации меньше чем  
количество элементов массива, то  
оставшиеся значения автоматически  
заполняются нулями

```
int arr[6] = { 1, 2, 3 };
```

// такая запись эквивалентна записи:

```
int arr[6] = { 1, 2, 3, 0, 0, 0 };
```

Если значений в списке  
инициализации больше чем  
количество элементов массива, то  
происходит ошибка на этапе  
КОМПИЛЯЦИИ

```
// int arr[2] = {1, 2, 3}; // Ошибка на этапе компиляции
```

# Расположение массива в памяти

Формула, согласно которой производится позиционирование по массиву:  
базовый адрес + размер базового типа \* индекс;



# Операция индексирования массива

Запись значения в массив:

```
имя_массива[индекс_элемента] = значение;
```

Получение значения из массива:

```
cout << имя_массива[индекс_элемента];
```

```
const int nArraySize = 3;
```

```
int ar[nArraySize];
```

```
ar[1] = 7;
```

```
cout << ar[1] << endl;
```

Большинство операций с массивами  
разумно проводить с помощью  
итерационных (переборных) циклов,  
по очереди перебирая элементы

# Циклическая обработка массива

```
#include <iostream>

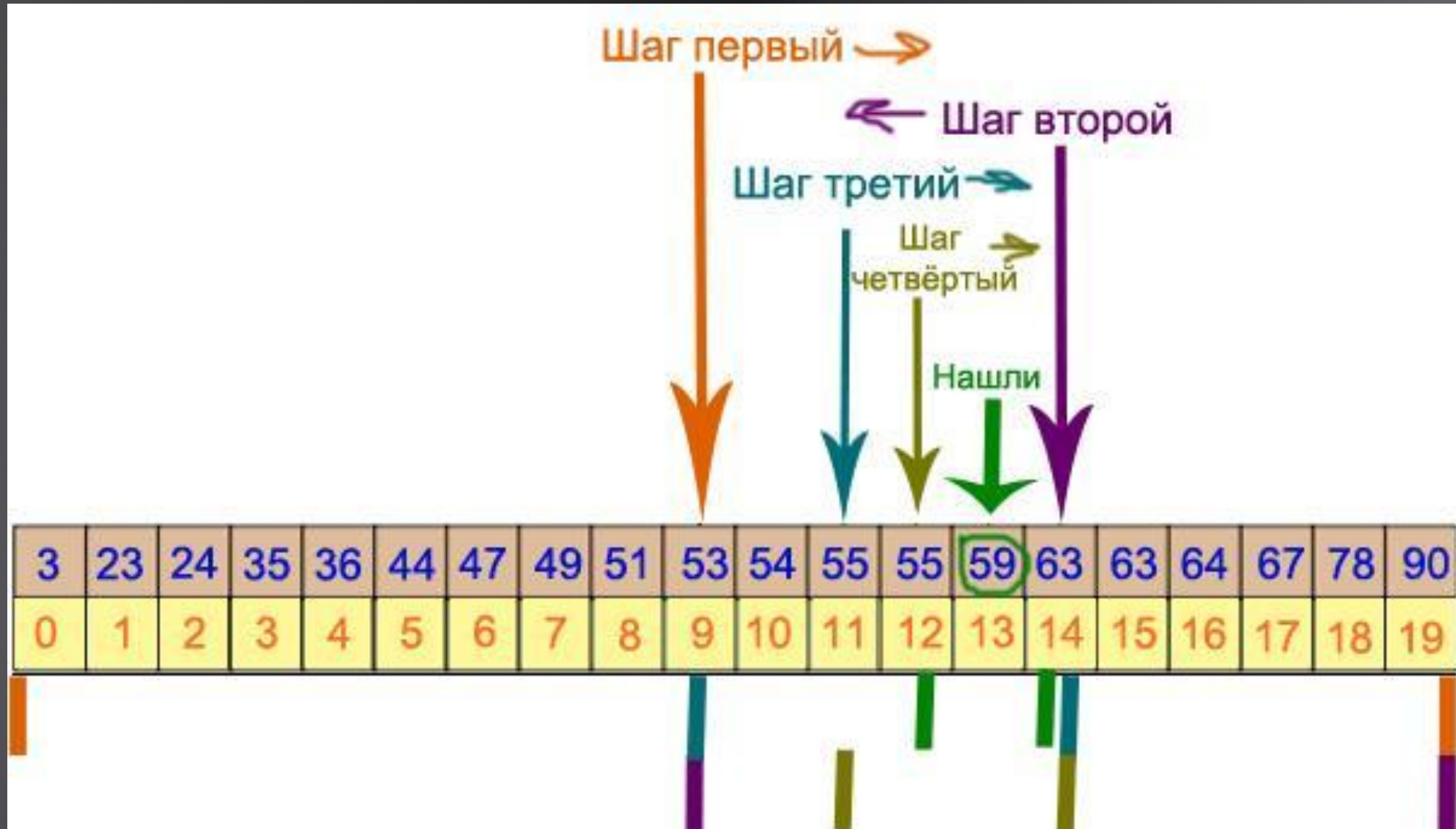
using namespace std;

void main()
{
    const int SIZE = 5;
    int arr[SIZE];
    for (int i = 0; i < SIZE; i++)
    {
        cout << "Enter value at index " << i << " ";
        cin >> arr[i];
    }
    cout << endl;
    for (int i = 0; i < SIZE; i++)
    {
        cout << "array[" << i << "] = " << arr[i] << endl;
    }
}
```

# Линейный поиск элемента в массиве

<b>42</b>	67	93	11	05	<b>18</b>	12	88	26	02
	18								
42	<b>67</b>	93	11	05	<b>18</b>	12	88	26	02
	18								
42	67	<b>93</b>	11	05	<b>18</b>	12	88	26	02
		18							
42	67	93	<b>11</b>	05	<b>18</b>	12	88	26	02
			18						
42	67	93	11	<b>05</b>	<b>18</b>	12	88	26	02
				18					
42	67	93	11	05	<b>18</b>	12	88	26	02
					18				

# Бинарный поиск элемента в массиве





# Сортировка массива прямым обменом (метод «пузырька»)



# Сортировка массива прямыми

## вставками

