Основы алгоритмизации и программирования

Лекция 3

Знакомство с языком С

Знакомство с языком Си

- Состав языка Си
- Структура программы
- Основные типы данных в Си
- Декларация объектов
- Данные целого типа (integer)
- Данные символьного типа (char)
- Данные вещественного типа (float, double)
- Использование модификаторов
- Константы в программах

Знакомство с языком Си

Любая программа, написанная на языке высокого уровня, **состоит из последовательности инструкций**, оформленных в строгом соответствии с набором правил, составляющих **синтаксис данного языка**

При создании программ разработчик может допустить следующие ошибки

Синтаксические ошибки – это результат нарушения формальных правил написания программы на конкретном языке программирования

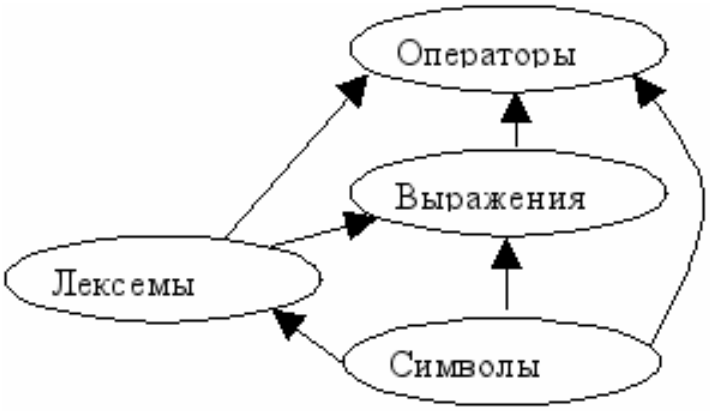
Логические ошибки

Ошибки алгоритма . Причиной является несоответствие построенного алгоритма ходу получения конечного результата сформулированной задачи

Семантические ошибки. Причина – неправильное понимание смысла (семантики) операторов выбранного языка программирования

Состав языка Си

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



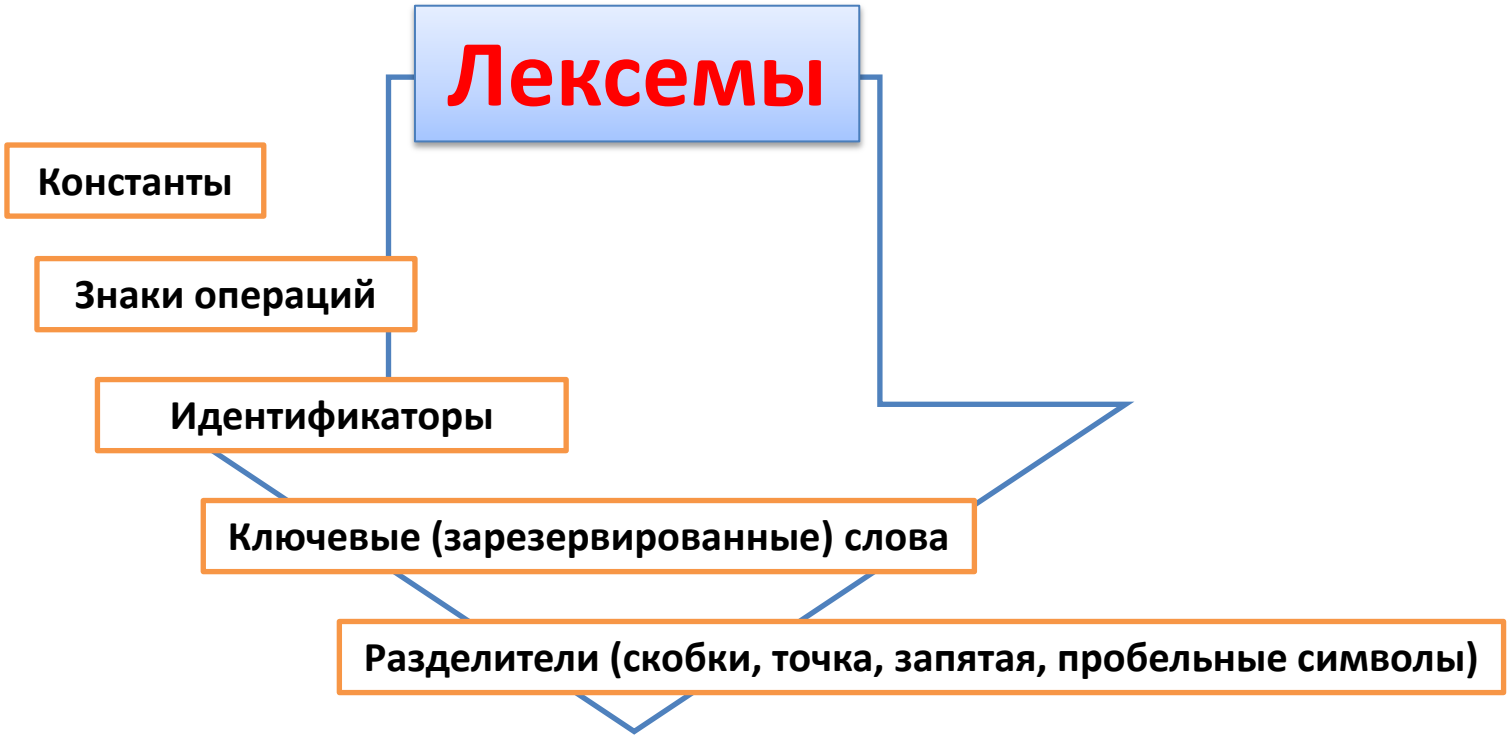
Алфавит любого языка составляет **совокупность символов** – тех неделимых знаков, при помощи которых записываются все тексты на данном языке.

Алфавит языка Си включает:

- Прописные и строчные буквы латинского алфавита и знак подчеркивания (код 95)
- Арабские цифры от 0 до 9
- Специальные символы
- Пробельные (разделительные) символы: пробел, символы табуляции, перевода строки, возврата каретки, новой страницы и новой строки

Состав языка Си

Из символов алфавита формируются **лексемы** (или элементарные конструкции) языка – минимальные значимые единицы текста в программе.



Границы лексем определяются другими лексемами, такими как **разделители** или **знаки операций**, а также **комментариями**

Состав языка Си

Идентификатор (*ID*) – это имя программного объекта (**константы, переменной, метки, типа, функции и т.д.**). В идентификаторе могут использоваться латинские буквы, цифры и знак подчеркивания.

При именовании объектов следует придерживаться общепринятых СОГЛАШЕНИЙ

Первый символ *ID* – не цифра; пробелы внутри *ID* не допускаются.

0001 **ID переменных** и **функций** обычно пишутся **строчными** (малыми) буквами – ***index, max*()**

0010 **ID типов** пишутся с **большой буквы**, например, ***Spis*, *Stack***

0011 **ID констант** (**макросов**) – **большими буквами** – **INDEX, MAX_INT**

0100 **Идентификатор** должен **нести смысл**, поясняющий назначение объекта в программе, например, **birth_date** – день рождения, **sum** – сумма

0101 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**

Состав языка Си

В **Си прописные** и **строчные буквы** – **различные символы**. Идентификаторы **Name**, **NAME**, **name** – **различные объекты!**

Ключевые (**зарезервированные**) слова **не могут быть использованы в качестве идентификаторов**

Список ключевых слов, определенных в стандарте **ANSI Си**

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

Состав языка Си

Еще один базовый элемент языка программирования – **комментарий** – **не является лексемой**. Внутри комментария можно использовать любые допустимые на данном компьютере символы, поскольку компилятор их игнорирует.
В **Си** комментарии ограничиваются парами символов **/*** и ***/**, а в **C++** был введен вариант комментария, который начинается символами **//** и заканчивается символом перехода на новую строку.

Комментарии могут размещаться **везде, где допускается пробел**.
Используйте комментарии, чтобы документировать Ваш код. В этом примере комментарий, принятый компилятором

Комментарий, принятый компилятором:

```
/* Comments can contain keywords such as
 for and while without generating errors. */
```

Комментарии могут размещаться в той же строке, что и оператор программы:

```
printf( "Hello\n" );  /* Comments can go here */
```

Можно размещать описательный блок комментариев перед функциями или модулями программы:

```
/* MATHERR.C illustrates writing an error routine
*   for math functions.
*/
```

Поскольку комментарии не могут содержать вложенные комментарии, этот пример является причиной ошибки:

```
/* Comment out this routine for testing
                /* Open file */
                Fh = _open( "myfile.c", _O_RDONLY );
                .
                .
                .
*/
```

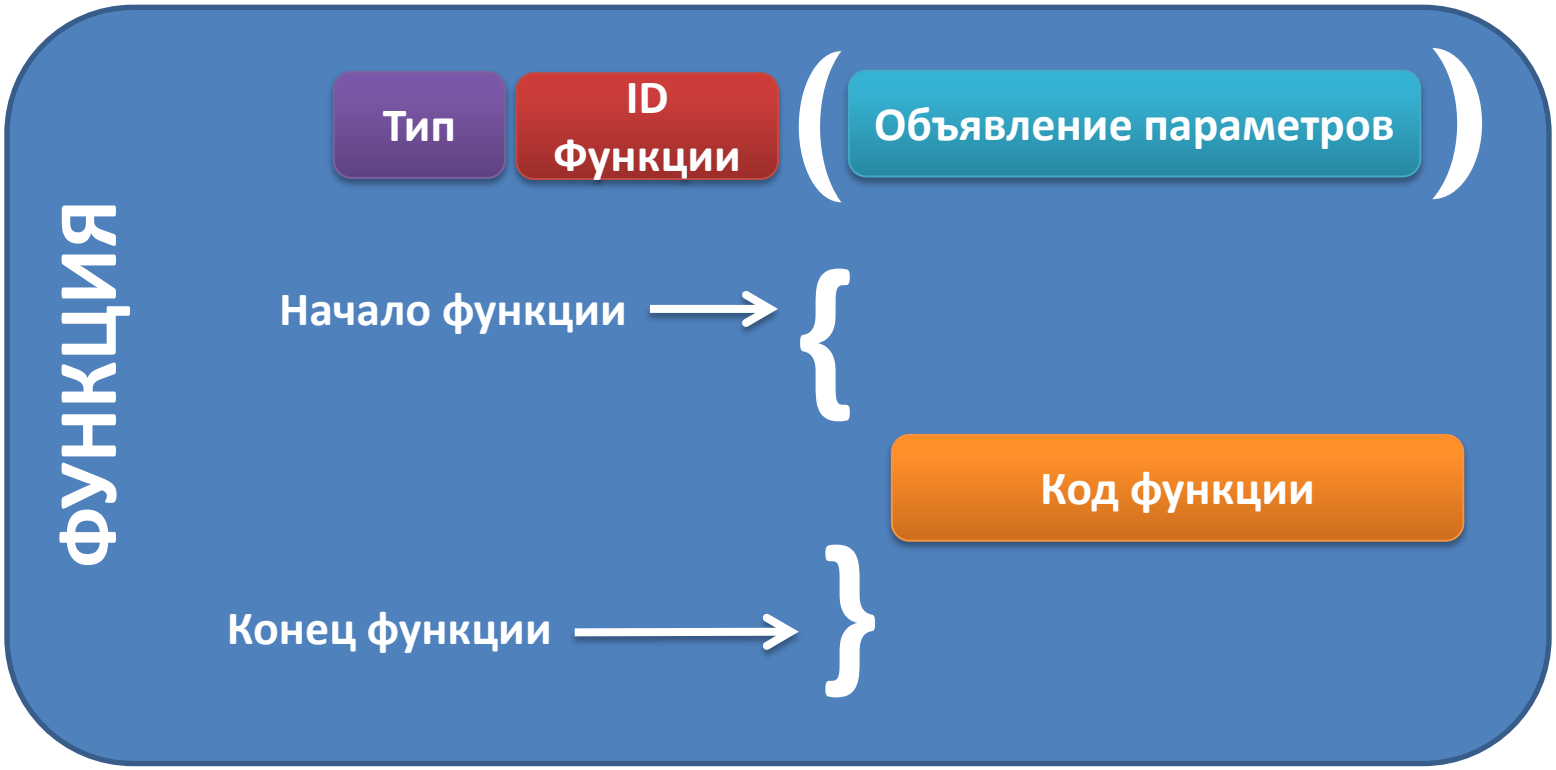
Структура программы

Программа, написанная на языке **Си**, **состоит из одной или нескольких функций**, одна из которых имеет идентификатор **main** – главная (основная). Она является **первой выполняемой функцией** (**с нее начинается выполнение программы**) и ее назначение – управлять работой всей программы (**проекта**).

Общая структура программы на языке Си

Структура программы

В свою очередь, каждая **функция** имеет следующую **структуру**




```
int main(void)
{
        printf(“ Высшая оценка знаний – 10 !”);
        return 0;
}
```

Структура программы

Перед компиляцией программа обрабатывается **препроцессором**, который работает под управлением директив.

Препроцессорные **директивы** начинаются символом **#**, за которым следует наименование директивы, указывающее ее действие.

Препроцессор решает ряд задач по предварительной обработке программы, основной из которых является подключение (**include**) к программе так называемых **заголовочных файлов** (**обычных текстов**) с декларацией стандартных библиотечных функций, использующихся в программе.

Пример:

#include < ID_файла.h>

К наиболее часто используемым библиотекам относятся:

- ***stdio.h*** – содержит стандартные функции файлового ввода-вывода
- ***math.h*** – математические функции
- ***conio.h*** – функции для работы с консолью (клавиатура, дисплей)

Если идентификатор файла заключен в угловые скобки (**< >**), то поиск данного файла производится в стандартном каталоге, если – в двойные кавычки (**” ”**), то поиск файла производится в текущем каталоге.

Структура программы

Второе основное назначение препроцессора – **обработка макроопределений**

Общий вид определителя

#define ID строка

```
#define PI 3.1415927

/* В ходе препроцессорной обработки программы идентификатор PI везде будет заменяться значением 3.1415927 */
```

Простейшая программа на Си

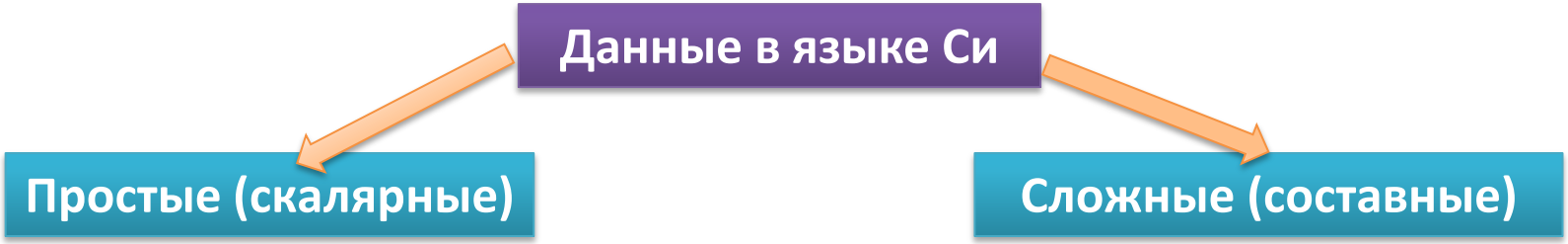
```
#include <stdio.h>
void main(void)
{           // Начало функции main

      printf(" Высшая оценка знаний – 10 !");

}           // Окончание функции main
```

Отличительным признаком функции служат скобки **( )** после ее идентификатора, в которые заключается список параметров. Перед **ID** функции указывается тип возвращаемого ею результата. **Если функция не возвращает результата и не имеет параметров**, указывают атрибуты **void** – **отсутствие значений**

Основные типы данных в Си



Тип данных определяет

- внутреннее представление данных в оперативной памяти
- совокупность значений (диапазон), которые могут принимать данные этого типа
- набор операций, которые допустимы над такими данными

Основные типы базовых данных: целый – ***int*** (***integer***), вещественный с одинарной точностью – ***float*** и символьный – ***char*** (***character***)

В свою очередь, данные целого типа могут быть короткими – ***short***, длинными – ***long*** и беззнаковыми – ***unsigned***, а вещественные – с удвоенной точностью – ***double***

Основные типы данных в Си

Данные целого и вещественного типов находятся в определенных диапазонах, т.к. занимают разный объем оперативной памяти

Тип данных	Объем памяти (байт)	Диапазон значений
char	1	–128 … 127
int	2 (4)	–32768 … 32767
short	1 (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	$3{,}4\cdot10^{-4932}$ … $3{,}4\cdot10^{4932}$

Размер памяти зависит от разрядности процессора, для 16-разрядных объем памяти определяется первой цифрой, для 32-разрядных – второй(в скобках)

Декларация объектов

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

Все **объекты**, с которыми работает программа, необходимо **декларировать**, т.е. объявлять компилятору об их присутствии.

Формат декларации объектов

<атрибуты> <список ID объектов>;

int i, j, k; ***//*** *ID объектов* разделяются запятыми
float a, b; //*атрибуты* – разделителями
char r;
double gfd;

Объекты программы могут иметь следующие атрибуты

Тип – тип будущих значений декларируемых объектов

Класс памяти – характеристика способа размещения объектов в памяти. Определяет область видимости и время жизни переменной

Данные целого типа (integer)

Тип ***int*** – целое число, обычно соответствующее естественному размеру целых чисел. Квалификаторы ***short*** и ***long*** указывают на различные размеры и определяют объем памяти, выделяемый под них

Атрибуты ***signed*** и ***unsigned*** показывают, как интерпретируется старший бит числа – как знак или как часть числа

Данные символьного типа (char)

Под величину **символьного типа** отводится такое количество байт, которое достаточно для любого символа. Поэтому символьная переменная занимает в памяти **один байт**. Закрепление конкретных символов за кодами производится **кодовыми таблицами**

Для персональных компьютеров **(ПК)** наиболее распространена ***ASCII*** (***American Standard Code for Information Interchenge***) таблица кодов

Данные типа **char** рассматриваются компилятором как целые, поэтому возможно использование **signed char**: величины со знаком (по умолчанию) – символы с кодами от **–128** до **+127** и **unsigned char** – беззнаковые символы с кодами от **0** до **255**. Этого достаточно для хранения любого символа из **256-символьного набора *ASCII***. Величины типа **char** применяют еще и для хранения целых чисел из указанных диапазонов.

Данные символьного типа (char)

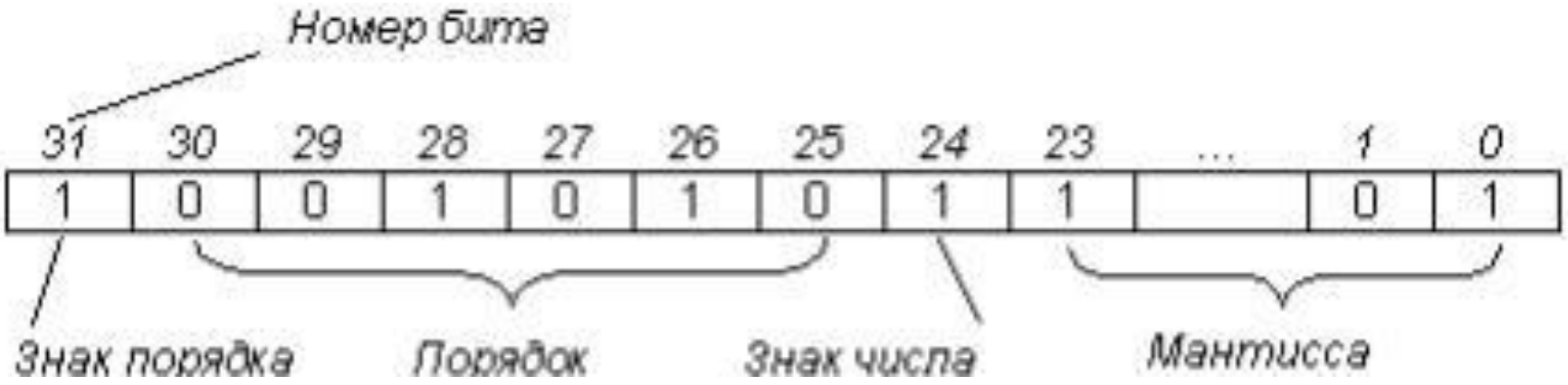
ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Данные вещественного типа (float, double)

Данные вещественного типа в памяти занимают: **float** – **4 байта** (одинарная точность), **double** (удвоенная точность) – **8 байт**; **long double** (повышенная точность) – **10 байт**.

Для размещения данных типа **float** обычно **8 бит** выделено для представления порядка и знака и **24 бита** под мантиссу



Тип	Точность (мантисса)	Порядок
float (4 байта)	7 цифр после запятой	± 38
double (8 байт)	15	± 308
long double (10 байт)	19	± 4932

Данные вещественного типа (float, double)

Типы данных с плавающей десятичной точкой хранятся в оперативной памяти иначе, чем целочисленные. Внутреннее представление вещественного числа состоит из двух частей: **мантиссы** и **порядка**. В **IBM** совместимых ПК, как вы уже знаете, переменная типа **float** занимает **4 байта**, из которых один двоичный разряд отводится под знак мантиссы, **8 разрядов** под **порядок** и **23** под **мантиссу**.
Для того, чтобы **сохранить максимальную точность**, вычислительные машины почти всегда хранят мантиссу в **нормализованном виде**.
Мантисса – это число больше единицы и меньше двух. Поскольку старшая цифра мантиссы всегда равна единице, то ее не хранят.

Значение порядка хранится не как число, представленное в дополнительном коде. Для упрощения вычислений значение порядка в ЭВМ хранится в виде смещенного числа. Это означает, что к действительному значению порядка прибавляется смещение перед записью его в память. Смещенным значение порядка делается для того, чтобы можно было сравнивать значения порядка с помощью обычной операции сравнения чисел с фиксированной точкой без знака. В частности, это полезно при сравнении двух чисел с плавающей точкой.

Данные вещественного типа (float, double)

Для величин типа **double**, занимающих **8 байт**, под порядок и мантиссу отводится **11** и **52 разряда** соответственно. **Длина мантиссы** определяет **точность числа**, а **порядок** – его **диапазон**. При одинаковом количестве байт, отводимом под величины типа **float** и **long int**, диапазоны их допустимых значений **сильно различаются** из-за внутренней формы представления значений таких данных.

При переносе программы с одной платформы на другую нельзя делать предположений, например, о типе **int**, так как для **MS DOS** этот тип имеет размер в **2 байта**, а для **ОС *Windows* – четыре байта**. В стандарте **ANSI** поэтому диапазоны значений для основных типов не задаются, а определяются только соотношения между их размерами, например:

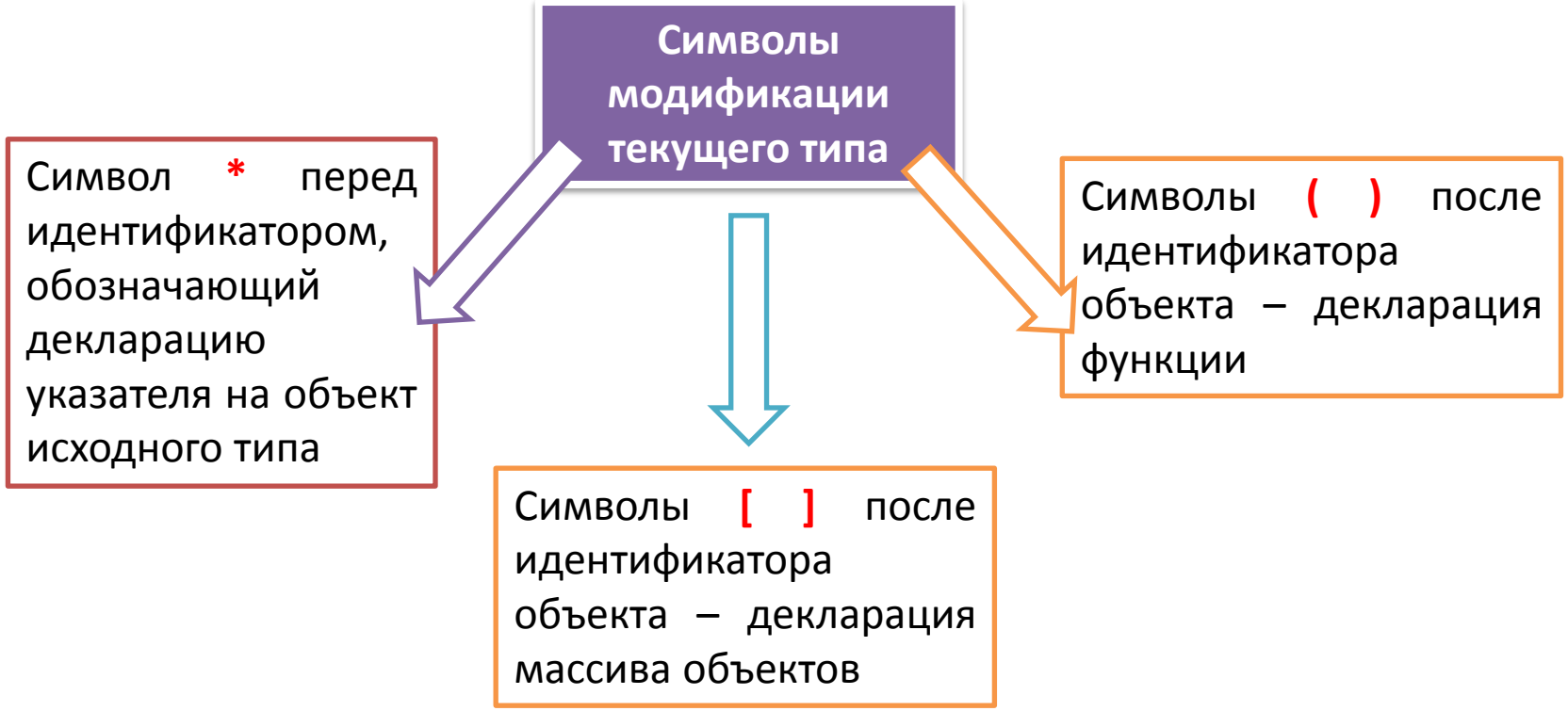
sizeof (***float***) < *sizeof* (***double***) < *sizeof* (***long double***) ,

sizeof (***char***) < *sizeof* (***short***) < *sizeof* (***int***) < *sizeof* (***long***) ,

где операция ***sizeof*** – возвращает количество байт для указанного аргумента – скалярного типа данных.

Использование модификаторов

Ключевые слова **int**, **float**, ***char*** и т.д. называют конечными атрибутами декларации объектов программы. При декларации так называемых производных объектов используют еще дополнительные – промежуточные атрибуты или, как их иногда называют, «**модификаторы**»



Использование модификаторов

Правила использования более одного модификатора типа

Чем ближе модификатор к *ID* объекта, тем выше его приоритет

При одинаковом расстоянии от идентификатора объекта модификаторы **[ ]** и **( )** обладают приоритетом перед атрибутом звездочка *****

Дополнительные круглые скобки позволяют изменить приоритет объединяемых ими элементов описания

Квадратные и круглые скобки, имеющие одинаковый приоритет, рассматриваются слева направо

Использование модификаторов

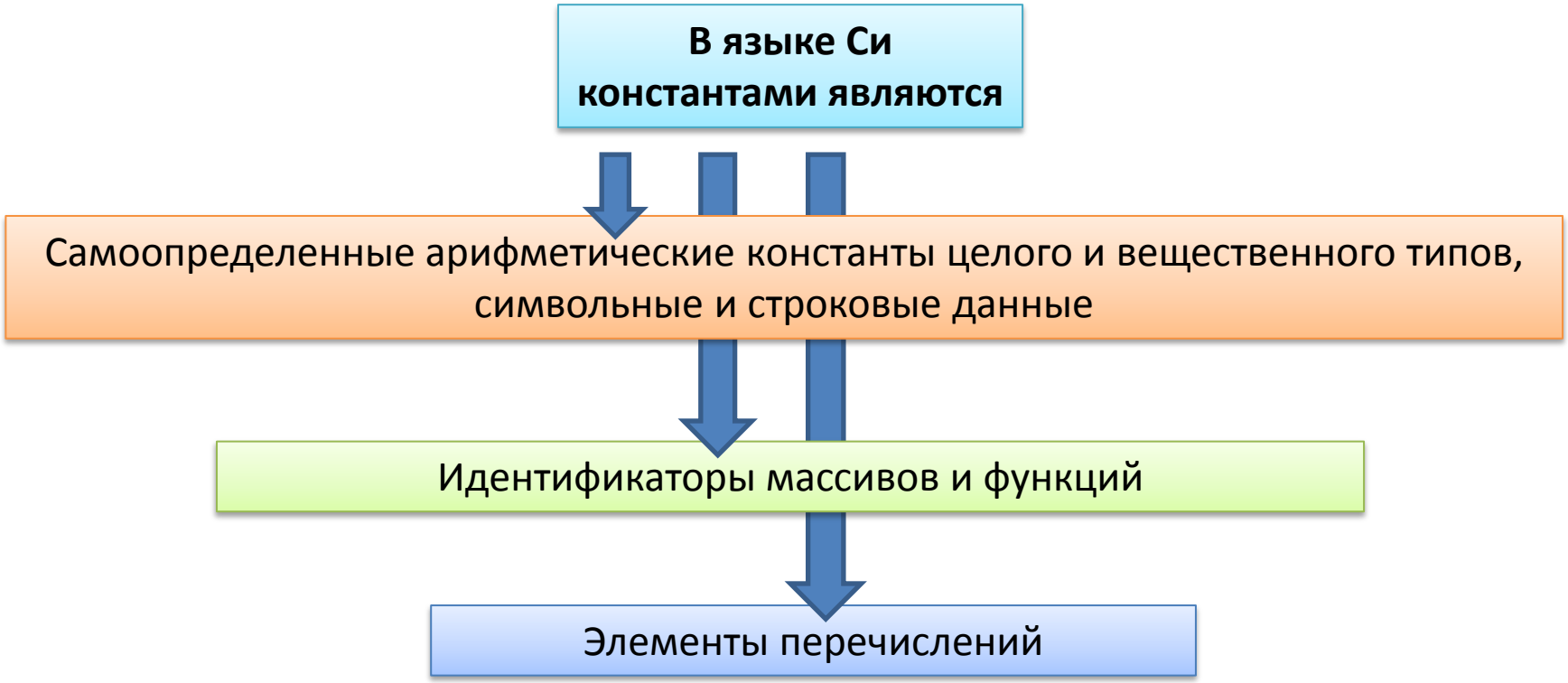
Примеры

***int a*;** – переменная типа *int*;
***int a*[5];** – массив из пяти элементов типа *int*;
***int *a*;** – указатель на объект типа *int*;
***int **a*;** – указатель на указатель на объект типа *int*;
***int *a*[5];** – массив из пяти указателей на элементы типа *int*;
***int* (*a)[10];** – указатель на массив из десяти элементов типа *int*;
***int *a*[3][4];** – 3-элементный массив указателей на одномерные целочисленные массивы по четыре элемента каждый;
***int a*[5][2];** – двухмерный массив элементов типа *int*;
***int a*(*void*);** – функция без параметров, возвращающая значение типа *int*;
***int *a*(*void*);** – функция без параметров, возвращающая указатель на элемент типа *int*;

Существуют недопустимые последовательности промежуточных атрибутов, например, массив не может состоять из функций, а функция не может возвращать массив или другую функцию

Константы в программах

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



Константы в программах

Целочисленные константы

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

Десятичные константы – это последовательность цифр **0...9**, первая из которых *не должна быть* **0**. 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: $\mathbf{020_8 = 16_{10}}$.

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

Пример

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

Константы в программах

Константы вещественного типа

Данные константы размещаются в памяти в формате **double**

Форма представления

С **фиксированной** десятичной точкой,

формат записи: $\pm n.m$, где **n**, **m** – целая и дробная части числа

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

Константы в программах

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’.

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

char ss = ‘У’;

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

Константы в программах

Строковые константы

Строковая константа представляет собой последовательность символов кода **ASCII**, заключенную в кавычки (”)

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

Примеры

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

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**

Строковые константы еще называют **строковыми литералами**.
В конец строковой константы компилятор автоматически помещает **нуль-символ**.