ЭЛЕМЕНТЫ ЯЗЫКА СИ

Под элементами языка понимается набор его базовых конструкций, используемых при написании программ.

Элементы языка Си:

1. АЛФАВИТ - это набор допустимых символов языка:
   - ✓ прописные и строчные латинские буквы (a-z, A-Z);
   - ✓ арабские цифры (0-9);
   - ✓ знаки пунктуации (. , , , ! , ‘ , “ , ; , ?);
   - ✓ специальные символы (#, \, /, +, -, *, _, % и др.);
   - ✓ скобки: ( ), { }, [ ].

При формировании строковых и символьных данных можно использовать буквы русского алфавита. В текстах программ допускаются комментарии и пробелы.

Комментарии - это текст, поясняющий назначение переменных и действия программы. Комментарий ограничивается одним из двух способов:

а) /*Это комментарий*/ - может занимать несколько строк

б) // Это тоже комментарий - одна строка

Комментарии воспринимаются компилятором языка, как пробельные символы и игнорируются. Внутри комментариев нельзя использовать сочетание символов */, т.е. комментарии не могут быть вложенными.

Комментарии могут быть следующих видов:

1) после заголовка главной функции – общая информация о программе: ее назначение, входные данные и результаты, метод решения задачи, ФИО программиста, дата написания программы и т.д.;
2) если есть несколько функций, то комментарий должен быть для каждой из них; он может также определять начало и завершение определенной функционально законченной программы;
3) при объявлении данных функции – пояснения назначения используемых в ней переменных;
4) пояснения логически сложных частей программы, содержащих разветвления, циклы.

Однако количество комментариев не должно быть чрезмерным.

Буквы и цифры используются при формировании констант, меток, идентификаторов и ключевых слов.

2. ИДЕНТИФИКАТОРЫ - это последовательность латинских букв, цифр и символа подчеркивания, начинающаяся с буквы или символа подчеркивания. Идентификаторы могут иметь произвольную длину, но используются в программе только первые 32. В идентификаторе прописные и строчные буквы различаются, т.е. name, Name и NAME являются различными. Как правило, в языке Си в именах переменных используется только строчные буквы, а для именованных констант - заглавные.

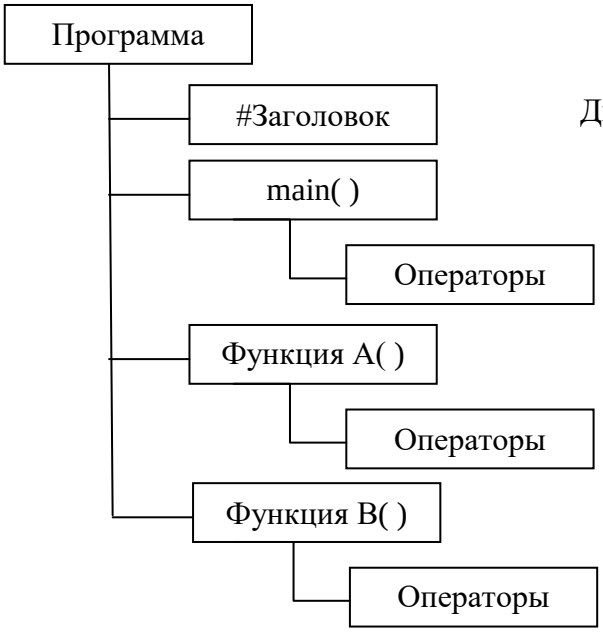
3. КЛЮЧЕВЫЕ (служебные, зарезервированные) СЛОВА - это идентификаторы, которые имеют специальное значение для компилятора и не могут быть идентификаторами переменных и данных пользователя.

auto	автоматический	int	целое
break	завершить	long	длинное
case	вариант	register	регистровый
char	символьный	return	возврат
const	константа	short	короткий
continue	продолжить	signed	знаковый
default	по умолчанию	sizeof	размер
do	выполнить	static	статический
double	двойной точности	struct	структура
else	иначе	switch	переключатель
enum	перечисляемый	typedef	определение типа
extern	внешний	union	объединение
float	плавающее	unsigned	беззнаковый
for	для	void	пустой
goto	перейти	volatile	изменчивый
if	если	while	пока

4. МЕТКИ. Любой оператор в программе может быть помечен меткой. Метка ставится перед оператором и отделяется от него двоеточием. Метка в Си – это идентификатор. Метки, использованные в теле функции, локальны в ней. Область действия метки - функция. Поэтому в разных функциях можно использовать одинаковые метки.

Примеры меток: vvod, vivod, vhod

СТРУКТУРА ПРОГРАММЫ НА ЯЗЫКЕ СИ



Директивы препроцессора и указания компилятору

Любая программа на языке Си начинается с инструкций подключения к программе заголовочных файлов, обеспечивающих подключение библиотечных функций. Эти инструкции имеют вид:

#include <имя заголовочного файла>

Все заголовочные файлы имеют расширение **.h** (от англ. *header* – заголовок). Поэтому файлы с расширением **.h** называются **файловыми заголовками**. Использование символов < > дает указание компилятору, что поиск этого файла будет осуществлен в системном каталоге. Если нужно подключить файл из текущего каталога, то вместо символа < > используются " ".

Символ # означает, что строка должна быть обработана препроцессором языка Си. **Препроцессор** - это программа, выполняющая предварительную обработку исходного текста программы перед началом компиляции.

Каждая инструкция include (директива препроцессора) записывается с новой строки. В конце инструкции никакие разделительные знаки не используются.

После всех инструкций подключения заголовочных файлов обычно помещают описание главной функции программы. Оно начинается с заголовка функции **main ()**, за которым никакие знаки не ставятся. Главная функция определяет действия выполняемые программой и вызывает на выполнение другие функции. Слово **void** пе-

ред **main** обозначает, что главная функция не возвращает никаких значений в результате своего выполнения. Пустые скобки обозначают отсутствие аргументов.

За заголовком следует тело этой функции, заключенное в фигурные скобки **{}**. Вначале записываются *объявления и определения переменных*. **Объявление переменных** - это описание переменных, которое задает им имена и атрибуты, также это приводит к выделению под переменные области памяти. **Определение (инициализация) переменных** задает им начальное значение.

Далее следует последовательность выполняемых операторов. Каждый оператор завершается символом «**;**». Обычно каждый оператор записывается с новой строки, что обеспечивает хороший обзор программы.

Если программа содержит несколько функций, то они располагаются последовательно одна за другой, причем функция main( ) может занимать в этой последовательности произвольное место. Выполнение программы в любом случае начинается с выполнения функции main( ).

```
#include  <stdio.h>  /* подключение функций ввода-вывода при работе с экраном
  main( )                    и файлами*/
{
   int num=1;
   printf ("Я простая");
   printf ("вычислительная ЭВМ. \n ");
   printf ("Мое любимое число %d, потому что оно самое первое. \n ",  num);
}
```

ДАННЫЕ. ТИПЫ ДАННЫХ

Алгоритм, реализующий решение некоторой конкретной задачи, всегда работает с данными. Данные - это любая информация, представленная в формализованном виде и пригодная для обработки алгоритмом. Данные, известные перед выполнением алгоритма, являются начальными, ***исходными данными.*** Результат решения задачи - это конечные, ***выходные данные.*** Данные делятся на переменные и константы. С данными тесно связано понятие типа данных. Любой константе, переменной, выражению (с точки зрения обработки на ЭВМ) всегда сопоставляется некоторый тип. ***Тип данных характеризует***:

1. ***Множество значений,*** к которым относится константа и которые может принимать переменная или выражение. Например, если переменная в некотором алгоритме может принимать только значения из множества целых чисел, то ей ставится в соответствие целый тип данных.

2. ***Множество операций,*** которые можно к этим данным применять, и правила выполнения этих операций.

3. ***Количество памяти***, выделяемое под переменные данного типа.

В языке Си имеется четыре базовых арифметических (числовых) типа данных. Из них два целочисленных – char, int – и два плавающих (вещественных) – float и double. Существуют два модификатора размера – short (короткий) и long (длинный) – и два модификатора знаков – signed (знаковый) и unsigned (беззнаковый). Знаковые модификаторы могут применяться только к целым типам.

Основные характеристики различных типов данных

Наименование типа	Тип	Память в байтах	Диапазон значений
Целые			
целый	int	2/4	- 32768..32767
беззнаковый целый	unsigned int	2/4	0..65535
короткий	short int	2	- 32768..32767
беззнаковый короткий	unsigned short	2	0..65535

длинный	long int	4	-2147483648..2147483647
беззнаковый длинный	unsigned long	4	0..4294967295
Вещественные			
вещественный	float	4	3.4E-38..3.4E+38
вещественный с двойной разрядностью	double	8	1.7E-308..1.7E+308
длинный вещественный	long double	10	3.4E-4932..1.1E+4932
Символьные			
символьный	char	1	-128..127
беззнаковыйсимвольный	unsigned char	1	0..255

Если не указан модификатор знаков, то по умолчанию подразумевается signed; с базовым типом float модификаторы не употребляются; модификатор short применим только к базовому типу int.

Величины типа char в языке Си могут рассматриваться в программе и как символы, и как целые числа. Все зависит от контекста, т.е. от способа использования этой величины. В случае интерпретации величины типа char как символа ее числовое значение является ASCII-кодом. Например:

```
char a=65;
printf ("%c",a);          //на экране появится символ A
printf ("%d",a);          //на экране появится число 65
```

В языке Си отсутствует логический тип данных. В качестве логических величин выступают целые числа. Интерпретация их значений в логические величины происходит по правилу: равно нулю – ложь, не равно нулю – истина.

ПЕРЕМЕННЫЕ. ОБЪЯВЛЕНИЕ ПЕРЕМЕННЫХ

Переменные - это величины, значение которых может изменяться в процессе выполнения программы. Объявления (описания) переменных могут быть в любой последовательности. Можно группировать переменные по типам или объявлять каждую из них отдельно.

Описание переменных в программах на Си имеет вид (синтаксис):

имя типа список переменных;

Например,

```
int i, j,k;       // описание целых переменных
float x,y;        // описание вещественных переменных
char sim, s;  // описание символьных переменных
```

Присвоение переменным начальных значений называется инициализацией переменных. Инициализацию можно проводить двумя способами: при описании переменных, в операторе присваивания.

Описание переменных с инициализацией имеет следующий вид:

тип имя переменной=начальное значение;

Например,

```
int a=0, b=1;          // инициализация целых переменных
float x=1.2;           // инициализация вещественных переменных
char sim='*';          // инициализация символьных переменных
```

КОНСТАНТЫ. ТИПЫ КОНСТАНТ

Константа (постоянная) - это величина (число, символ, символьная строка), которая не изменяет своего значения в процессе выполнения программы.

Запись целых констант.

В языке Си целые константы могут быть представлены в одной из трех систем счисления: десятичной, восьмеричной, шестнадцатеричной.

Целые десятичные числа, начинающиеся не с нуля (973, -25).

Целые восьмеричные числа, запись которых начинается с нуля (045, 011).

Целые шестнадцатеричные числа, запись которых начинается с символов 0x (0x16, 0xA2).

Тип константы компилятор определяет по следующим правилам: если значение константы лежит в диапазоне типа int, то она получит тип int, в противном случае проверяется, лежит ли константа в диапазоне unsignedint, в случае положитель-

ного ответа она получит этот тип, если не подходит и он, то пробуется тип long и, наконец, unsigned long. Если значение числа не укладывается в диапазон типа unsigned long, то возникает ошибка компиляции.

Запись вещественных констант.

Если в записи числовой константы присутствует десятичная точка (2.5) или экспоненциальное расширение (1E-8), то компилятор рассматривает ее как вещественное число и ставит ей в соответствие тип double.

Использование суффиксов.

Можно явно задать тип константы, используя для этого суффиксы: F (f) - float, U (u) - unsigned, L (l) – long (для целых и вещественных констант). Допускается совместное использование суффиксов U и L в вариантах UL или LU.

Примеры:

3.14159F – константа типа float, под которую выделяется 4 байта памяти;

3.14L - константа типа long double, занимает 10 байт;

50000U - константа типа unsigned int, занимает 2 байта памяти (вместо четырех без суффикса);

0LU - константа типа unsigned long, занимает 4 байта;

24242424UL - константа типа unsigned long, занимает 4 байта.

Запись символьных и строковых констант.

Символьная константа представляет собой одиночный символ, заключенный в апострофы ('A', '%', '3'). Значение символьной константы равно значению кода представляемого ею символа, символьные константы имеют тип int.

Строковые константы представляют собой последовательность символов, заключенную в кавычки. Это массив символов, каждый элемент которого является отдельным символом. В конце каждой символьной строки компилятор помещает пустой символ ‘\0’, который является признаком конца строки.

Следует различать символьные и строковые константы.

Константа	- 'A'	char	1байт
Строка	- “A”	char [ ]	2байта (“A\0”)

Особую разновидность символьных констант представляют так называемые управляющие символы. Их назначение – управление выводом на экран. В программах на Си они изображаются парой символов, первый из которых '\'.

В языке Си можно использовать *именованные* и *неименованные* константы. Неименованные константы – это константы, представленные своим значением. Если значение одной и той же константы используется в программе многократно, применяют именованные константы.

Синтаксис записи именованных констант:

const имя типа имя константы = выражение;

Например: const float e=2.71;

Служебное слово const принято называть квалификатором доступа. Он указывает на то, что данная величина не может изменять своего значения во время работы программы и не может использоваться в левой части оператора присваивания.

Ещё одной возможностью ввести именованную константу является использование препроцессорной директивы #define в следующем формате:

#define имя константы значение константы

Например:

#define N 100

Тип констант явно не указывается и определяется по форме записи. В конце директивы не ставится точка с запятой. На стадии препроцессорной обработки указанные имена заменяются на соответствующие значения.

Последовательность целочисленных именованных констант можно определить с помощью перечисляемого типа.

Синтаксис описания перечисляемого типа:

enum {список значений};

Например:

enum {A,B,C,D};

По умолчанию значение первой константы равно нулю (т.е. A=0,B=1,C=2,D=3).

Для любой константы можно явно указать значение.

Например:

enum {A=10,B,C,D};

Возможен и такой вариант определения перечисления:

enum {A=10,B=20,C=35,D=100};

Если перечисляемому типу дать имя, то его можно использовать в описании переменных.

Например:

enum metal {Fe,Co,Na,Cu,Zn};

metal Met1,Met2;

Здесь идентификатор metal становится именем типа. После такого описания в программе возможны следующие операторы:

Met1=Na; Met2=Zn;

ВЫРАЖЕНИЯ

Основное назначение программы состоит в выполнении действий, необходимых для решения поставленной задачи. Решение любой задачи представляет собой процесс формирования результатов из заданных исходных данных. Правила формирования новых значений задаются с помощью выражений.

Во всех языках программирования под выражением подразумевается конструкция, составленная из операндов (констант, переменных, функций), операций и круглых скобок, с помощью которых можно задать требуемый порядок выполнения операций. Выражение определяет порядок вычисления некоторого значения.

В языке Си используются следующие типы выражений: арифметические, логические, условные и над адресами. Тип выражения определяется типом результата, то есть значением, которое формируется при выполнении последней операции выражения. Результатом выполнения выражения может быть значение одного из допустимых типов: арифметического (целого или вещественного) или символьного. Тип результата определяется как типом операндов, так и типом операций, выполненных над операндами.

Если есть вложенные скобки, то вычисление выражения начинается с самых внутренних скобок. Внутри скобок и если в выражении не используются круглые скобки, последовательность выполнения операций определяется с учетом относительного приоритета операций.

Все операции в Си разделены по приоритетам на 16 категорий. Операции одной категории имеют один и тот же приоритет.

Приоритет	Операция	Комментарий	Порядок выполнения
1	(), []	Скобки	
Все унарные (кроме постфиксных)			
2	++	Увеличение операнда на единицу	←
	- -	Уменьшение операнда на единицу	←
	-	Изменение знака	←
	*	Определение значения по адресу	←
	&	Определение адреса	←
	!	Логическое отрицание	←
	~	Поразрядное логическое отрицание**	←
	(тип)	Явное преобразование типа	←
	sizeof	Определение размера	←
Бинарные арифметические			

3	* / %	Умножение Деление Остаток от деления**	→ → →
4	+ -	Сложение Вычитание	→ →
5	<< >>	Сдвиг влево** Сдвиг вправо**	→ →
6	< < = > > =	Меньше Меньше или равно Больше Больше или равно	→ → → →
7	== !=	Равно Не равно	→ →
Поразрядные операции			
8	&	Поразрядное И**	→
9	^	Поразрядное исключающее ИЛИ**	→
10	|	Поразрядное ИЛИ**	→
Логические			
11	&&	Логическое И	→
12	||	Логическое ИЛИ	←
Условная			
13	?	Условная	→
Присваивание и постиксные бинарные			
14	= * = / = % = + = - = << = >> =	Простое присваивание Умножение с присваиванием Деление с присваиванием Остаток от деления с присваиванием** Суммирование с присваиванием Вычитание с присваиванием Сдвиг влево с присваиванием Сдвиг вправо с присваиванием	← ← ← ← ← ← ← ←
Постфиксные унарные			
15	++ - -	Увеличение операнда на единицу Уменьшение операнда на единицу	← ←
Операция Запятая			
16	,	Последовательность выполнения	←

Примечания:
** - операция определена только для данных целого типа;
→ - порядок выполнения операции одного приоритета слева направо;
← - порядок выполнения одного приоритета справа налево

Последовательность вычисления функций в выражениях Си не задана.

Порядок вычисления выражения всегда можно отрегулировать скобками, их можно ставить сколько угодно, но здесь важно соблюдать золотую середину. При большом количестве скобок снижается наглядность выражения и легко ошибиться.

Если выражение со скобками выглядит корректно, то компилятор может отследить только парность скобок, но не правильность их расстановки.

При рассмотрении выражений различных типов определяют по каждому типу:

- допустимые операции;
- стандартные функции, допустимые для данных определенного типа.

АРИФМЕТИЧЕСКИЕ ВЫРАЖЕНИЯ

Арифметические выражения – аналог алгебраических выражений математики. Они используются:

1) в операторах присваивания;
2) в качестве фактических параметров функций;
3) в операторах заголовка цикла.

Арифметические операции:

Операция	Назначение	Тип операнда	Тип результата
+	Плюс (сложение)	Целый, вещественный	Совпадает с типом операнда
-	Унарный минус (вычитание)	Целый, вещественный	Совпадает с типом операнда
*	Умножение	Целый, вещественный	В соответствии с иерархией типов
/	Деление	Целый, вещественный	В соответствии с иерархией типов
%	Остаток от деления (деление по модулю)	Целый	Целый
++	Инкремент (увеличение на 1)	Целый, вещественный	Совпадает с типом операнда
--	Декремент (уменьшение на 1)	Целый, вещественный	Совпадает с типом операнда

Приоритет: 1) ++, -- 2) – 3) *, /, % 4) +, -

Сложение, вычитание и умножение значений выполняются как обычно.

Операция *«деление по модулю»* применяется только к данным целого типа, знак совпадает со знаком первого из операндов.

Операция *деления* дает результат с плавающей точкой, если хотя бы один из операндов является вещественным, если оба операнда целые, то результат тоже будет целым - произойдет отбрасывание дробной части (аналогично функции DIV в Паскале).

Например:

5/2= 2 5/2.0=2.5 5.0/2=2.5 (13+6)/2*3=27

В одном выражении могут быть операнды различных типов. Если арифметическая операция выполняется над двумя операндами различных типов, то они сначала преобразуются к одному и тому же общему типу в соответствии со схемой иерархии типов, и результат выполнения операции будет того же типа. Схема иерархии типов определяет тип результата: это более высокий тип одного из двух операндов. Самый высокий тип операнда – **double**.

В арифметических выражениях операндами могут быть переменные целого типа и типа **char**, так как на машинном уровне и те, и другие имеют одинаковое целое представление. Поэтому для данных типа char применимы все операции, операндами которых могут быть целые **int**.

double ← float
↑
long ← int ← char, short, enum
↑
unsigned ← unsigned char, unsigned short

Для использования математических функций в программах нужно подключать библиотеку математических функций - **math.h.**

#include <math.h>

Математические функции:

Функция	Название	Тип аргумента	Тип результата
abs(x)	абсолютное значение целого x (модуль x)	int	int
fabs(x)	абсолютное значение вещест-	double	double

	венного x (модуль x)		
atan(x)	арктангенс (радианы)	double	double
sin(x)	синус (угол в радианах)	double	double
asin(x)	арксинус (угол в радианах)	double	double
cos(x)	косинус (угол в радианах)	double	double
acos(x)	арккосинус (угол в радианах)	double	double
tan(x)	тангенс (угол в радианах)	double	double
exp(x)	экспонента - e^x	double	double
log(x)	натуральный - ln x	double	double
log10(x)	логарифм десятичный - lg x	double	double
sqrt(x)	корень квадратный	double	double
pow(x,y)	значение x в степени y - x^y	double double	double
fmod(x,y)	остаток от деления нацело x на y	double double	double
floor(x)	ближайшее меньшее целое, <=x	double	double
ceil(x)	ближайшее большее целое, >=x	double	double

x^y = pow(x,y) = exp(y*ln(x))

ОПЕРАЦИИ «ИНКРЕМЕНТ, ДЕКРЕМЕНТ»

Операция *инкремент* ++ увеличивает значение переменной на 1, операция *декремент* -- уменьшает значение переменной на 1.

В зависимости от положения знака бывают:

префиксные операции

- ✓ предъинкремент ++a
- ✓ преддекремент --a

постфиксные операции

- ✓ постъинкремент a++
- ✓ постдекремент a--

И та и другая операция увеличивают или уменьшают величину **a** на 1, но разница заключается в том, что при выполнении постфиксных операций значение **a** сначала используется в выражении, а потом изменяется на 1. А при выполнении

префиксных операций значение переменной сначала изменяется на 1, а потом используется в выражении.

Различие между ++a и a++ имеет место только в выражениях. Если же ++a и a++ используются как отдельные операторы, то разницы между ними нет.

a++ - это выражение **a++;** - это оператор

Таким образом, ++a; a++; a=a+1; дают один и тот же ответ.

Например: Определить значение переменных a,b,c

a=3;	a=4	a =3;	a =4
b=2;	b=3	b=2;	b=3
c=a++*b++;	c=6	c=++a*++b;	c=12

Операции инкремент и декремент могут применяться только к переменным и не могут к константам или выражениям, нельзя написать 5++ или (a + b)++.

ОПЕРАЦИИ ПРИСВАИВАНИЯ

Присваивание в языке Си является операцией, а не оператором.

Простая операция присваивания записывается знаком равенства, слева от которого стоит переменная, а справа выражение, совместимое с типом переменной:

x=3.5 y=2*(x–0.567)/(x + 2)

К *составным операциям присваивания* относятся +=, -=, *= и /=, а также операции инкремент ++ и декремент --. Символы записываются без пробелов, нельзя переставлять их местами. Операции +=, -=, *= и /= являются укороченной формой записи оператора присваивания.

a+=b означает a=a+b a-=b означает a=a-b

a*=b означает a=a*b a/=b означает a=a/b

Все операции присваивания присваивают переменной результат вычисления выражения. Если тип левой части присваивания отличается от типа правой части, то тип правой части приводится к типу левой.

В одном операторе операция присваивания может встречаться несколько раз. Вычисления производятся справа налево.

Например:

a = b = c*d;

вначале выполняется операция умножения, затем переменной присваивается ее результат, далее значение переменной b присваивается переменной a.

ОПЕРАЦИИ ОТНОШЕНИЯ (СРАВНЕНИЯ)

Операции сравнения в языке Си:

- ✓ равно ==
- ✓ не равно !=

Операции отношения:

- ✓ больше >
- ✓ меньше <
- ✓ больше или равно >=
- ✓ меньше или равно <=

Сдвоенные символы записываются без пробелов, их нельзя переставлять местами. Числа сравниваются по величине, у символов сравниваются их коды. Результат сравнения – целое число 0 или 1 типа **int**. Если сравнение верно, то получаем 1, если неверно - 0. Для записи сложных сравнений следует использовать логические операции.

ЛОГИЧЕСКИЕ ОПЕРАЦИИ

1. Логическое отрицание НЕ - !
2. Логическое умножение И - &&
3. Логическое сложение ИЛИ - ||

Правила их выполнения определяются соответствующими таблицами истинности. Результатом выполнения логической операции может быть 0 - ложь или 1 - истина.

Операнды логических операций могут иметь целый тип, плавающий или быть указателями. Типы операндов в выражении могут быть различными. Логические операции не выполняют преобразований по умолчанию. Тип результатов всегда **int**.

ОПЕРАЦИЯ ЯВНОГО ПРЕОБРАЗОВАНИЯ ТИПА (ОПЕРАЦИЯ «ТИП»)

Синтаксис записи:

(имя типа) операнд

Операндом могут быть константа, переменная, выражение. В результате значение операнда преобразуется к указанному типу.

```
float c;
int a=1, b=2;
c=(float)a/b;
c=0.5
```

ОПЕРАЦИЯ sizeof

Синтаксис записи:

sizeof (тип)

sizeof (выражение)

Результатом операции является целое число, равное количеству байтов, которое занимает в памяти величина явно указанного типа или величина, полученная в результате вычисления выражения. Последняя определяется также по типу результата выражения.

Примеры:

```
sizeof (int)        результат 2
sizeof (1)          результат 2
sizeof (0.1)        результат 8
```

sizeof (1L) результат 4

sizeof (char) результат 1

sizeof (‘a’) результат 1

ОПЕРАЦИЯ «ЗАПЯТАЯ»

Эта необычная операция используется для связывания нескольких выражений в одно. Несколько выражений, разделенных запятыми, вычисляются последовательно слева направо. В качестве результата такого совмещенного выражения принимается значение самого правого выражения.

Например, если переменная a имеет тип int, то значение выражения (a=3, 5*a) будет равно 15.

УСЛОВНАЯ ОПЕРАЦИЯ

Это единственная операция, которая имеет три операнда.

Синтаксис записи:

выражение 1? выражение 2: выражение 3;

Данная операция реализует алгоритмическую структуру ветвления.

Сначала вычисляется значение выражения 1, которое обычно представляет собой некоторое условие. Если оно истинно (т.е. имеет ненулевое значение), то вычисляется выражение 2, и его значение будет результатом всей операции. В противном случае в качестве результата берется значение выражения 3. В любом случае вычисляется выражение 2 или 3, но не оба сразу.

Выражение 2 и 3 в свою очередь тоже могут быть условными выражениями. В этом случае имеет место вложенность условных выражений.

Условная операция поначалу кажется странной, но она очень удобна для записи небольших разветвлений и очень часто применяется в языке C.

Например:

1) Вычисление абсолютной величины переменной x

x<0? - x: x;

2) int i=3, j;

(i<0)? j=i: j=-i; лучше j=(i<0)? i: (-i); => j=-3

3) Выбор большего значения из двух переменных a и b

max=(a>b)? a: b;

ОПЕРАЦИИ ВВОДА-ВЫВОДА В ЯЗЫКЕ СИ

Язык Си не включает в себя стандартных средств ввода-вывода данных. Эти операции выполняются с помощью функций. Функции ввода-вывода сгруппированы в трех библиотеках: **stdio.h** (std - стандартная, i - ввод, o - вывод); **io.h** и **conio.h**.

Каждая программа на языке Си, которая обращается к функциям из стандартной библиотеки, должна в заголовке содержать директиву препроцессора

#include <stdio.h>

ФУНКЦИЯ ВЫВОДА printf( )

Функция printf() (print format – форматная печать) осуществляет преобразование и печать данных в соответствии с указанным форматом представления данных.

Синтаксис записи:

printf ("формат", аргумент1,…,аргумент n);

где **аргументы** - это выводимые на печать значения, которые могут быть константами, переменными или выражениями. Список аргументов может отсутствовать.

формат - символьная строка, содержащая объекты трех типов:

обычные символы, которые просто копируются в выходной поток;

управляющие символы – влияют на расположение на экране выводимых знаков. Признаком управляющего символа является значок \ .

Специальные управляющие символы

Символ	Назначение
\n	Переход на новую строку

\r	Возврат курсора к началу строки
\t	Табуляция (курсор сдвигается вправо на 8 позиций)
\b	Возврат на одну позицию
\f	Переход к новой странице
\\	Выводит на экран символ \ (слэш)
\’	Выводит на экран апостроф
\”	Выводит на экран кавычки

спецификации преобразований, каждая из которых вызывает преобразование и печать очередного аргумента.

Если в списке аргументов содержится аргументов больше, чем спецификаций, то лишние аргументы отбрасываются. Если же спецификаций преобразований больше, чем аргументов, то результат является неопределенным.

Синтаксис записи спецификации преобразования:

% [выравнивание] [ширина поля] символ преобразования

Например: printf (“Значение переменной a=%3d”,a);

Символы преобразования

Формат	Тип выводимых значений
d	Ввод-вывод целого значения
c	Ввод-вывод одного символа
s	Ввод-вывод символьной строки
f	Ввод-вывод значения с плавающей точкой
e (E)	Ввод-вывод значения с плавающей точкой в экспоненциальной форме вида [-] число e (или)[-]xx (для числа по умолчанию берется: 1 позиция под целую часть, 6 позиций под дробную часть)
g (G)	Ввод-вывод по формату f или e (E) в зависимости от того, какая запись короче
u	Ввод-вывод беззнакового целого
o	Ввод-вывод беззнакового восьмеричного целого
x	Ввод-вывод беззнакового шестнадцатиричного целого

Наличие пробелов между спецификациями гарантирует, что даже в том случае, если все поле будет заполнено, то символы, соответствующие данному числу не перейдут в следующее поле.

Выравнивание по умолчанию производится по правому краю ширины поля. Если в позиции выравнивания поставить знак «-», то выравнивание производится по левому краю ширины поля.

Ширина поля задает количество позиций, отводимых на экране для изображения значения.

Если заданная ширина поля недостаточна для изображения числа, то этот параметр игнорируется, и величина будет выводиться полностью.

Если ширина поля велика для выводимого числа, то поле заполняется пробелами слева или справа в зависимости от выравнивания.

Если печатаемое число включено в некоторую фразу, то часто при его выводе удобно задавать ширину поля, равной или меньшей требуемой. Это дает возможность включить число в фразу без добавления лишних пробелов.

Вместо ширины поля можно поставить *, которая означает, что число выводимых символов определяется текущим значением переменной.

При выводе символьных строк ширина поля определяет максимальное число печатаемых символов.

При выводе чисел с плавающей точкой возможно задание точности (число позиций под дробную часть числа). Она задается через точку от ширины поля. Если точность превышает количество символов в дробной части, то лишние позиции дополняются нулями. Если же количество цифр в дробной части числа больше точности, то производится округление дробной части. Если в спецификации f не указана точность, то по умолчанию после точки берется 6 цифр.

ФУНКЦИЯ ВВОДА scanf()

Функция scanf( ) (форматный ввод) осуществляет ввод данных с клавиатуры и преобразование их во внутреннее представление в соответствии с типом величин.

Синтаксис записи:

```
scanf ("формат", аргумент 1,...,аргумент n);
```

Форматная строка и список аргументов присутствуют обязательно.

где **аргументы** – это перечень вводимых переменных, причем перед именеми каждой переменной обязательно указывается символ **&**(это знак операции «взятия адреса переменной»).

формат - символьная строка, которая может содержать символы трех типов:

1. Пробелы, символы табуляции, символ перехода на новую строку, которые компилятором игнорируются.

2. Обычные символы. В этом случае пользователь программы должен набрать на клавиатуре перед задаваемым значением указанную перед форматом последовательность исмволов, иначе ввод прекратится при первом же несовпадающем символе.

3. Спецификации преобразования – указывают необходимые преобразования вводимых данных для очередного аргумента.

В спецификации преобразования указывается символ % и символ преобразования.

ОПЕРАТОРЫ ЯЗЫКА СИ

Оператором в Си называют любое выражение, вслед за которым стоит точка с запятой. Операторы управляют процессом вычисления в программе и задают ход выполнения программ. Все операторы принято группировать в следующие классы: присваивания, вызов функции, ветвления и цикла. Отличительная особенность языка Си: присваивание реализовано не как оператор, а как операция.

ПУСТОЙ ОПЕРАТОР

Синтаксис оператора:

```
;
```

Оператор может появиться в любом месте программы, где по синтаксису требуется оператор. Выполнение пустого оператора не изменяет состояние программы.

Он может использоваться:

1) для установки метки (например, чтобы пометить закрывающуюся фигурную скобку составного оператора, который не является оператором, нужно поставить перед ней помеченный пустой оператор);

2) после ключевого слова else оператора if, вложенного в другой if, внешний по отношению к данному;

3) после оператора заголовка цикла, когда действия, которые должны быть выполнены в теле цикла, помещаются в заголовок цикла.

СОСТАВНОЙ ОПЕРАТОР

Составной оператор – это блок, ограниченный фигурными скобками. Он может содержать объявление переменных, которые будут действовать только в пределах этого блока, и выполняемые операторы любого типа (перед закрывающейся фигурной скобкой точку с запятой надо ставить обязательно).

Синтаксис оператора:

```
{
  [объявление;]
  [операторы;]
}
```

Используется в программе там, где по синтаксису требуется один оператор, а нужно выполнить несколько действий. Например, его можно использовать для ограничения операторов тела функции, для ограничения групп операторов в условных операторах и в операторах циклов.

В составном операторе могут быть другие составные операторы. Любой оператор внутри составного оператора может быть помечен меткой. Передача управления внутрь составного оператора возможна, но если составной оператор содержит объявление переменных и их инициализацию, то при входе в блок по метке, эта инициализация не будет выполнена, и значение переменных будет неопределено.

ОПЕРАТОР ПЕРЕХОДА

Оператор goto передает управление непосредственно на оператор, помеченный меткой.

Синтаксис оператора:

goto метка;

где метка – это идентификатор.

Например:

```
    goto m1; …..
m1: a=b; …...
```

Область действия метки ограничивается функцией, которой она определена. Из этого следует: 1) каждая метка должна быть отлична от других меток той же самой функции; 2) нельзя передавать управление оператором goto в другую функцию.

Помеченный оператор выполняется сразу после выполнения оператора goto. Если оператор с данной меткой отсутствует или существует более одного оператора, помеченных одной и той же меткой, то это приводит к ошибочному результату.

Не рекомендуется переход внутрь операторов: составного, условного, варианта, цикла.

Для программирования ветвлений в языке Си используют условную операцию **?**, условный оператор **if** и оператор выбора варианта **switch**.

УСЛОВНЫЙ ОПЕРАТОР

Синтаксис оператора:

```
if  (выражение) оператор1;
  [else оператор2;]
```

где выражение – это некоторое условие, содержащее операции отношения и логические операции. Значение выражения приводится к целому и интерпретируется в соответствии с правилом: =0 – ложь, ≠0– истина. Выражение обязательно записывается в круглых скобках.

Перед ключевым словом else точка с запятой:

- ставится, если оператор 1 простой;

- не ставится, если оператор 1 – составной оператор.

Условный оператор служит для вычислений по одной из двух ветвей в зависимости от значения выражения. Сначала вычисляется выражение, и если оно истинно, то выполняется <оператор 1>, если ложно, то выполняется <оператор 2>, непосредственно следующий за ключевым словом else.

Если значение выражения ложно, а конструкция else отсутствует, то управление передается на оператор, следующий в программе за оператором if.

Например:

```
if  (i>0)
   y=x/i;
else { y=1;
         i++; }
```

На месте оператора 1 и 2 может быть оператор if. В этом случае имеет место вложенный условный оператор.

Например:

```
if  (x<100)
   { y=0;
     z=5; }
else if (x<1000)
        y=1;
     else if (x<2000)
             y=2;
          else y=3;
```

При вложенности операторов if рекомендуется для ясности программы использовать фигурные скобки, ограничивающие операторы 1 и 2. Если фигурные скобки отсутствуют, то ключевое слово else будет относиться к ближайшему оператору if, у которого отсутствует конструкция else.

Пример 1.

```
if  (i>j)
   if (j>0)
     x=1;
   else x=2;
```

Пример 2.

```
if  (i>j)
   { if (j>0)
        x=1; }
else x=2;
```

/* else ассоциируется с внутренним if */ /* else ассоциируется с внешним if */

ОПЕРАТОР ВЫБОРА ВАРИАНТА

Синтаксис оператора:

```
switch (выражение целого или символьного типа)
{ case константное выражение 1: оператор1;[break;]
  case константное выражение 2: оператор 2;[break;]
                    . . . . . .
  case константное выражение N: оператор N;[break;]
[default: оператор;]
}
```

Оператор switch служит для выбора одного из нескольких вариантов; он проверяет совпадения значения выражения в скобках с одним из заданных константных выражений во всех вариантах case и передаёт управление оператору, который соответствует значению какого-либо из выражений. Каждый вариант case может быть помечен целой или символьной постоянной или выражением. Операторы, связанные с меткой default выполняются в том случае, если ни одна из констант не равна значению выражения. Вариант default не обязательно указывать в операторе и не обязательно последним. Если ни одно из константных выражений не соответствует значению выражения и вариант default отсутствует, то оператор switch не выполняет никаких действий.

Ключевое слово case вместе с константным выражением служит просто меткой. Если будут выполнены операторы для некоторого варианта, то далее выполняются операторы всех последующих вариантов до тех пор, пока не встретится оператор break. Это позволяет связать одну последовательность операторов с несколькими.

Если действия двух вариантов совпадают, то эти варианты могут быть объединены, в этом случае оператор switch имеет вид:

```
switch (выражение целого символьного типа)
```

```
{ case константное выражение 1:
  case константное выражение 2: оператор1;[break;]
[default: оператор;]
}
```

ОПЕРАТОР РАЗРЫВА

Синтаксис оператора:

```
break;
```

Оператор разрыва прерывает выполнение операторов цикла или оператора switch, в которых он появляется. Управление передаётся оператору, следующему за прерванным оператором. Если операторы цикла вложены, то break вызывает непременное прекращение того цикла в котором он находится. Появление break вне перечисленных операторах приводит к ошибке.

ОПЕРАТОРЫ ЦИКЛА

Циклы используют в случае, если некоторые действия надо выполнить многократно, каждый раз с новыми данными. В Си имеются операторы цикла while, do while и for.

ОПЕРАТОР ПРОДОЛЖЕНИЯ

Синтаксис оператора:

```
continue;
```

Оператор продолжения может использоваться в операторах цикла, предназначен для перехода за последний оператор тела цикла, т.е. на корректировку параметров цикла for (вычисляется выражение приращения) или на анализ конца циклов while и do while (происходит вычисление условного выражения).

ОПЕРАТОР ЦИКЛА С ПАРАМЕТРОМ (С ШАГОМ)

Оператор цикла с параметром используется, если заранее известно начальное значение параметра (параметров) цикла, условие его завершения и коррекция – выражение (выражения) для изменения параметра (параметров) цикла.

Синтаксис оператора:

```
for ([начальное значение];[условное выражение];[выражение приращения])
    оператор;
```

где

- начальное значение – список операторов, инициирующих начальные значения; выполняется один раз, до начала выполнения тела цикла; как правило, используется для установки начальных значений параметров цикла;

- условное выражение – список операторов и выражений для проверки конца цикла; конец цикла обычно определяется на основе анализа значения параметра цикла; выполняется перед каждым выполнением тела цикла; если значение последнего выражения истинно (≠0), тело цикла выполняется, а если ложно (=0) – завершается;

- выражение приращения – список операторов и (или) выражений для корректировки параметров цикла; выполняется после каждого выполнения тела цикла;

- оператор – простой или составной оператор тела цикла.

Правила выполнения цикла for:

1) производится вычисление *начального выражения*;

2) выполняется *условное выражение* и производится его оценка:

а) если оно истинно (≠0), то выполняется тело цикла;

б) если оно ложно (=0), то выполнение цикла завершается;

если оно ложно до первого выполнения тела цикла, то тело цикла не выполняется ни разу;

3) после выполнения тела цикла выполняется *выражение приращения* и осуществляется переход к п.2;

4) появление в любом месте тела цикла оператора continue дает переход к выполнению выражения приращения, т.е. к п.3;

5) появление в любом месте тела цикла оператора break вызывает переход к оператору, следующему после оператора цикла;

после выхода из цикла по оператору break или goto параметр цикла сохраняет значение, при котором произошло завершение выполнения цикла;

6) после нормального завершения цикла значение параметра цикла равно значению, которое привело к завершению выполнения цикла.

7) если начальное значение и (или) условное выражение отсутствуют, их ; должны остаться в операторе заголовка цикла;

Например:

for (; ;) for (i=1; ;i++) - бесконечные циклы, которые могут завершиться только при выполнении в его теле операторов break, goto, return.

Пример оператора цикла for:

```
for (i=0; i<10; i++)
    c+=i;
```

С помощью сокращенной формы оператора for можно реализовать временную задержку процесса выполнения программы. Например, для ограничения времени ожидания ответа пользователя - **for (n=1; n<=10000; n++);**

Этот цикл считает значения **n** от **1** до **10000,** ничего больше не делая. Символ **;** после заголовка цикла – это пустой оператор.

Любой список заголовка может содержать оператор запятая, т.е. несколько операторов и выражений, разделенных запятыми, которые выполняются слева направо. Таким образом, оператор цикла for может иметь как бы несколько параметров цикла, изменяющихся одновременно. Чаще всего *начальное выражение* и *выражение приращения* – операторы присваивания или обращение к функциям (если их нет, то параметр цикла как бы не рассматривается); а *условное выражение* содержит выражение отношения или логическое выражение (если его нет (остается только ;), то считается что условие проверки конца цикла истинно, при этом цикл

не может быть завершен по условию конца цикла в заголовке цикла, а только по операторам break, goto, return.

ОПЕРАТОР ЦИКЛА С ПРЕДУСЛОВИЕМ

Оператор цикла с предусловием (анализ конца цикла производится до выполнения операторов тела цикла) используется, когда количество повторений операторов тела цикла заранее не известно и определяется в процессе выполнения цикла.

Синтаксис оператора:

```
while (выражение)
 оператор;
```

где

выражение – выражение любого типа;

оператор - простой или составной оператор тела цикла, он должен включать оператор изменения операндов логического выражения.

Вначале вычисляется выражение (условие), если оно истинно, то выполняется тело цикла, и выражение вычисляется снова; если ложно – тело цикла не выполняется. Цикл продолжается до тех пор, пока значение выражения не станет ложным и управление сразу передаётся следующему оператору. Если выражение ложно изначально, то тело цикла не будет выполнено ни разу.

Например:

```
i=20;
while (i>=1)
  {x=i;
   i - -;
  }
```

ОПЕРАТОР ЦИКЛА С ПОСТУСЛОВИЕМ

Оператор цикла с постусловием (анализ конца цикла производится после выполнения операторов тела цикла) используется, когда количество повторений опе-

раторов тела цикла заранее не известно и определяется в процессе выполнения цикла.

Синтаксис оператора:

```
do
  оператор
while (выражение);
```

где

выражение – выражение любого типа;

оператор - простой или составной оператор тела цикла.

Тело цикла выполняется хотя бы один раз. После каждого выполнения тела цикла анализируется значение выражения: если оно истинно, то тело цикла выполняется снова, и снова вычисляется значение выражения; если ложно - цикл завершается.

Например:

```
do
  {y=x-5;
    x - -}
while (x>0);
```

МАССИВЫ В ЯЗЫКЕ СИ

Массив – это совокупность однотипных элементов, занимающих непрерывную область памяти. Элементы массива образуют последовательность, упорядоченную по номерам их элементов. С массивом связаны следующие его свойства: имя, тип, размерность.

Синтаксис объявления массива:

```
тип_элементов имя_массива [константное_выражение];
```

где

константное выражение определяет размер массива, т.е количество элементов массива. Например:

1) int a[10];

объявлен одномерный массив с именем a, содержащий 10 элементов целого типа.

2) float b[3][3];

объявлен двумерный массив b, состоящий из 3 строк и 3 столбцов, т.е. из 9 элементов вещественного типа.

В отличие от языка Паскаль в языке Си нельзя определять произвольные диапазоны для индексов. Размер массива, указанный в объявлении, всегда на единицу больше максимального значения индекса.

Для обращения к элементам массива используется имя массива, после которого в квадратных скобках стоит индекс в виде выражения определяющего значение индекса - порядкового номера элементов массива. Индексов в квадратных скобках может быть столько, сколько измерений в массиве. Каждый индекс указывается в своих скобках. Значения индекса должны лежать в диапазоне от 0 до величины на единицу меньшей, чем размер массива указанный при его объявлении.

Например: a[0], a[1], a[2], a[3], a[4], a[5], a[6], a[7], a[8], a[9].
b[0][0], b[0][1], …, b[3][3]

В памяти элементы массива располагаются так, что при переходе от элемента к элементу наиболее быстро меняется самый правый индекс массива. Т.е. матрица располагается в памяти по строкам. Между элементами массива в памяти разрывы отсутствуют. Операций над массивами в Си нет. Пересылка элементов одного массива в другой может быть реализована только поэлементно, с помощью цикла. Над элементами массива допускаются те же операции, что и над скалярными переменными того же типа. Форматный ввод-вывод значений элементов массива можно производить только поэлементно.

Пример 1. Ввод с клавиатуры и вывод на экран одномерного массива.

```
while(1)//открытие беск.цикла для проверки размерности
    {
     printf("Ведите размерность вектора:");
     scanf("%d",&r);
     if((r>0)&&(r<=R)) break;
       printf("Размерность вне диапазона,повторите ввод\n");
    } //закрытие беск.цикла для проверки размерности
```

```
printf("\n\t Введите вектор\n");// ввод вектора
for(i=0;i<r;i++)
    {
     printf("a[%d]=>",i);
     scanf("%f",&a[i]);
    } //конец ввода вектора
printf("\n\t Исходный вектор \n");//вывод вектора
for(i=0;i<r;i++)
   printf("%5.2f",a[i]);
printf("\n");//конец вывода вектора
```

Пример 2. Ввод с клавиатуры и вывод на экран двумерного массива.

```
while(1)
    {
     printf("Введите размерность матрицы по строкам:");
     scanf("%d",&ri);
     if ((ri>0)&&(ri<=R)) break;
       printf("Размерность вне диапазона, повторите ввод\n");
    }
while(1)
    {
     printf("Введите размерность матрицы по столбцам:");
     scanf("%d",&rj);
     if ((rj>0)&&(rj<=R)) break;
       printf("Размерность вне диапазона, повторите ввод\n");
    }
printf("\n\t Введите матрицу\n");
for(i=0;i<ri;i++)
    for(j=0;j<rj;j++)
       {
         printf("a[%d][%d]=> ",i,j);
         scanf("%f",&a[i][j]);
       }
printf("\n\t Исходная матрица \n");
for(i=0;i<ri;i++)
     {
       for(j=0;j<rj;j++)
          printf("%5.2f ",a[i][j]);
       printf("\n");
     }
```

При объявлении массива можно указывать инициализацию массива, т.е. присвоить значение каждому элементу массива.

Синтаксис инициализации значений:

тип_элементов имя_массива [размер]={список значений};

Например:

float a[5] = {1, 0.5, 3, 2.1, 6};

Размер массива может не указываться, если при объявлении производится инициализация значений элементов;

Например: int p[] = {2, 4, 6, 10, 1};

При инициализации многомерных массивов используются вложенные списки значений в фигурных скобках, разделенные запятыми.

```
int b[2][4] = {{1, 2, 3, -4}      // элементы 0 строки
               {5, 6, -7, 8}};    // элементы 1 строки
```

Если количество элементов массива известно до выполнения программы, можно определить его как константу с помощью директивы #define, а затем использовать ее в качестве границы массива. Именованные константы, определенные спецификатором const, нельзя использовать при объявлении границ массива.

СТРОКИ (СИМВОЛЬНЫЕ МАССИВЫ)

В языке Си нет специально определенного строкового типа данных (как в Паскале). Символьные строки организуются, как массивы символов, т.о. на длину строки в Си нет ограничения. В этом состоит их важное преимущество перед строками в Паскале, где размер строки не может превышать 255.

Каждый символ строки хранится в отдельном байте ОП. В конце каждой строки компилятор помещает нулевой символ **\0** (null – в явном виде он не отображается), поэтому каждая строка занимает на 1 байт ОП больше, чем символы в строке. По этой же причине строка, содержащая один символ, не совпадает с символьной константой.

Строка – это последовательность символов, заключенных в кавычки. Строка может содержать любые символы алфавита, кроме кавычек, символа \ и символа

конца строки \n (переход на новую строку). Для размещения в строке данных символов необходимо перед ними разместить символ \ .

Например: **“Он сказал: \”Привет!\””**

Каждая строковая константа, даже если она идентична другой строковой константе, сохраняется в отдельном месте памяти.

Если две строковые константы записаны одна за другой, то во время компиляции компилятор объединяет их:

“Hello, “, “Friends” => **“Hello, Friends”**

Такой механизм предоставляет удобное средство для записи длинных строковых констант.

Строка описывается, как символьный массив.

Синтаксис описания:

char имя строки [максимальный размер];

например: **char str[20];**

Одновременно с описанием строка может инициализироваться. Возможны два способа инициализации строки:

1) с помощью строковой константы (последовательности символов, заключенной в кавычки):

При таком способе инициализации в конце строки компилятор сам ставит символ **\0.**

char s[10] = “строка”; - начальное значение задавать необязательно

под строку s будет выделено 10 байт памяти, из них первые 7 получат значения при инициализации (седьмой – нулевой символ).

char s[] = ”строка”; - начальное значение обязательно

будет сформирована строка s из 7 символов.

2) в виде списка символов (элементов массива):

char s[10] = {‘с’,‘т’,’р’,’о’,’к’,’а’,‘\0’}; - описание равнозначно первому.

Символьный массив можно определить без нулевого символа в конце:

char s[10] = {‘с’,‘т’,’р’,’о’,’к’,’а’};

но это приведет к проблемам с обработкой такой строки, так как будет отсутствовать ориентир на ее окончание.

Отдельные символы строки идентифицируются индексированными именами, например: **s[0]='c', s[5]='a'**

Инициализация двумерного символьного массива строками:

```
char b[4][30]=
{
"Результаты сессии",
"_____________________",
" ФИО   | Оценки ",
"_____________________"
};
```

Тексты в кавычках эквивалентны скобочной записи:
{{…},{…},…};

ВВОД-ВЫВОД СИМВОЛЬНЫХ СТРОК

Прототипы функций находятся в библиотеке стандартного ввода-вывода **stdio.h**.

gets(); Аргументом функций указывается имя строки.	**чтение строки, введенной с клавиатуры** Функция gets() читает строку из стандартного потока ввода stdin (высокоуровневый ввод-вывод) и помещает её по адресу, задаваемому параметром функции. Ввод строки заканчивается при обнаружении символа новой строки \n, который заменяется на нулевой символ \0.
puts(); Аргументом функций указывается имя строки.	**вывод строки на экран** Функция puts() записывает строку, адрес которой определяется значением параметра в круглых скобках, в стандартный поток вывода stdout. При выводе, завершающий \0 заменяется символом новой строки, то есть после вывода строки на экран курсор переходит в начало следующей строки.
getchar(); **getc();**	**ввод одного символа с клавиатуры** Например: **ch = getchar ();** присваивание переменной символьного типа ch очередного символа, полученного из стандартного ввода.
putchar(); **putc();**	**вывод одного символа на экран** Например: **putchar (ch);**

	выводит символ ch на экран.
getch();	**ввод одного символа с клавиатуры без отображения его на экране**

Пример: Ввести с клавиатуры строку символов и вывести ее на печать.

1 способ.	2 способ.
# include <stdio.h> void main() { char str[100]; printf("Введите строку:\n"); gets(str); printf("Введенная строка\n "); puts(str); }	**# include <stdio.h> void main() { char str[100]; printf("Введите строку:\n"); scanf("%s", &str); printf("Введенная строка\n "); printf("%s\n", str); }**

Ввод и вывод строки можно оформить в цикле:

3 способ.	4 способ.
# include <stdio.h> void main() { char str[100]; int i; printf("Введите строку:\n"); for(i=0; i<10; i++) str[i]=getchar(); printf("Введенная строка\n "); for(i=0; i<10; i++)	**# include <stdio.h> void main() { char str[100]; int i; printf("Введите строку:\n"); for(i=0; i<10; i++) scanf(“%c”, &str[i]); printf("Введенная строка\n "); for(i=0; i<10; i++)**

putchar(str[i]); **}**	**printf(“%c”, str[i]);** **}**

ОБРАБОТКА СТРОК

Обработка строк обычно связана с перебором всех символов от начала до конца. Признаком конца такого перебора является обнаружение нулевого символа.

При манипулировании со строками система не контролирует выход фактической длины строки за пределы максимально допустимого размера, задаваемого при описании. В случае ошибки дополнительные символы будут записаны поверх информации, которая располагается вслед за полем, отведенном для строки.

Для избегания ошибки рекомендуется следить за выходом строки за предусмотренные пределы, вводя достаточные размеры строк при их описании.

Для использования функций обработки строк необходимо подключить в программе библиотеку **string.h** (и **stdlib.h**).

Функция	Назначение	Тип возвращаемого результата
strlen(str);	Определение **длины** строки str (завершающий символ \0 не учитывается).	**int**
strrev(str);	Инвертирование (реверс, **переворот**) строки str.	**char**
strlwr(str);	**Преобразование** букв **верхнего** регистра **в** строке в соответствующие буквы **нижнего** регистра (строчные). Только латинские буквы.	**char**
strupr(str);	**Преобразование** букв **нижнего** регистра **в** строке в соответствующие	**char**

	буквы **верхнего** регистра (прописные). Только латинские буквы.	
strcat(str1, str2);	Присоединение строки str2 к строке str1 (**объединение**, конкатенация). Результат - сцепленная строка str1.	**char**
strncat(str, str2, k);	**Добавление** к строке str1 **k** начальных **символов** из строки str2.	**char**
strcmp(str1, str2);	**Сравнение** строк str1 и str2. Возвращаемое значение **меньше нуля**, если str1<str2; **больше нуля**, если str1>str2; **равно нулю**, если строки равны, т.е. совпадают по текущей, а не по объявленной длине и содержат одни и те же символы. Если длины строк равны, то происходит поэлементное сравнение символов до первого несовпадающего символа, и, та строка считается большей, в которой первый несовпадающий символ имеет наибольший код.	**int**
strncmp(str1, str2, k);	**Сравнение первых k символов** строк str1 и str2.	**int**
strcpy(str1,str2);	**Копирование** строки **str2** (включая \0 символ) **в** строку **str1**. Результатом является строка str1.	**char**
strncpy(str1, str2, k);	**Копирование** в начало строки str1 **k первых символов** из строки str2.	**char**

СТРУКТУРНЫЙ ТИП ДАННЫХ (СТРУКТУРЫ)

В языке Си понятие структуры аналогично понятию записи (record) в языке Паскаль. Структурный тип данных обычно используется при разработке информационных систем, баз данных.

Структура - это структурированный тип данных, представляющий собой поименованную совокупность разнотипных элементов. Структуры позволяют группу связанных между собой переменных трактовать не как множество отдельных переменных, а как единое целое.

В отношении структуры можно выполнять следующие действия: копировать; выполнять операции присваивания; передавать функциям в качестве аргумента, а функции могут возвращать их в качестве результата; инициализировать. Структуры нельзя сравнивать.

Синтаксис описания структуры:

```
struct [имя типа]
        { тип 1 переменная 1;
          тип 2 переменная 2;
          ……..……;
          тип n переменная n;
        };
```

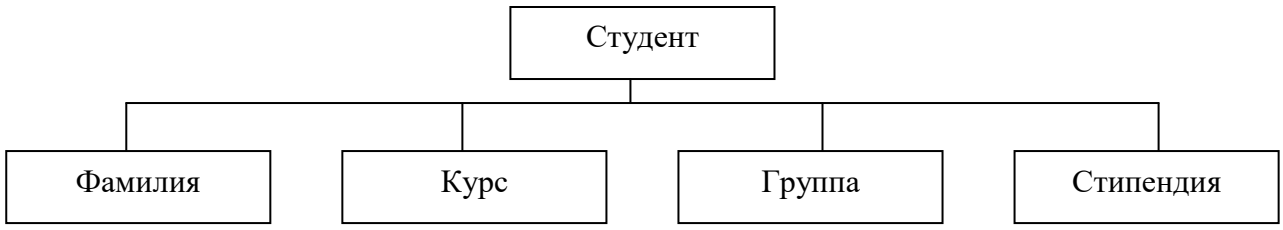
Описание начинается со служебного слова struct, за которым может следовать имя типа структуры. Далее это имя может служить кратким обозначением той части описания, которая заключена в фигурные скобки. В фигурных скобках определяется последовательность переменных величин, называемых элементами структуры, которые могут иметь различные типы. В конце обязательно ставится символ «;» (т.к. это оператор).

Объявления переменных списка имеют тот же самый синтаксис, что и объявление обычных переменных, за исключением того, что объявления не могут содержать спецификаторов классов памяти или начальных значений. Элементы структуры могут быть любого типа (базового, массивом, указателем, объединением или структурой). Их имена могут совпадать с именем обычных переменных, т.к. они всегда различны по контексту.

Имена элементов внутри объявленной структуры должны быть различными, внутри разных структур могут совпадать.

Описание структуры - это тип, то есть когда мы описываем структуру, мы просто описываем новый тип. Описание структуры, не содержащей списка переменных, не резервирует памяти, она просто описывает шаблон-образец структуры. Однако если структура имеет имя, то этим именем далее можно пользоваться при определении структурных объектов.

Например, требуется организовать сведения о выплате студентам стипендии:



Элементы такой структуры (фамилия, курс, группа, стипендия) называются полями. Каждому полю должно быть поставлено в соответствие имя и тип.

Для рассмотренного примера определение соответствующего структурного типа может быть следующим:

```
struct student
{
  char fam[30];
  int kurs;
  char grup[3];
  float stip;
};
```

После этого **student** становится именем структурного типа, который может быть назначен некоторым переменным.

В соответствии со стандартом Си это нужно делать так:

struct student stud1, stud2;

Служебное слово struct можно опускать и писать так:

student stud1, stud2;

где stud1 и stud2 – переменные структурного типа.

Допускаются и другие варианты описания структурных переменных. Можно вообще не задавать имя типа, а описывать сразу переменные:

```
struct
{
  char fam[30];
  int kurs;
  char grup[3];
  float stip;
} stud1, stud2;
```

Обращение к элементам (полям) структурной величины производится с помощью *уточненного* имени следующего формата:

имя структуры.имя элемента

Примеры уточненных имен для описанных выше переменных:

stud1.fam;

stud1.stip;

Значения элементов структуры могут определяться вводом, присваиванием, инициализацией.

Пример инициализации в описании:

student stud1 = {"Кротов", 3, "Ф32", 350};

Поля структуры могут сами иметь структурный тип. Такие величины представляют многоуровневые деревья.

Допускается использование массивов структур. Например, сведения о 100 студентах могут храниться в массиве, описанном следующим образом:

student stud[100];

Тогда сведения об отдельных студентах будут обозначаться, например, так:

stud[0].fam; - фамилия первого студента

stud[4].kurs; - курс пятого студента

stud[9].stip; - стипендия 10 студента и т.п.

Если нужно взять первую букву фамилии 25 студента, то следует писать так:

stud[24].fam[0];

ФУНКЦИИ В ЯЗЫКЕ СИ

Язык Си предоставляет программисту большие возможности в определении структуры программы. Качество программы зависит от умения программиста разбить свою программу на логически обоснованные функции (модули) с хорошо обоснованным интерфейсом и хорошо документированным текстом программы. Функция является основной программной единицей в языке Си, минимальным исполняемым программным модулем. В языке Си нет понятий подпрограммы, вся выполняемая часть программы строится на последовательности определений и функций.

Любая программа на языке Си содержит одну главную функцию с именем ***main()*** и любое количество других функций. Выполнение программы начинается с функции ***main()***, остальные функции выполняются по мере обращения к ним.

В языке Си существует два вида функций:

1) Функции, возвращающие значения в вызывающую программу.

Синтаксис записи:

```
[класс памяти]  тип функции  имя функции (список и определение формальных параметров)
{
  тело функции (операторы);
  return (выражение);
}
```

где

класс памяти - это спецификатор, который определяет класс памяти функции;

тип функции - определяет тип результата возвращаемого функцией, с помощью оператора return;

имя функции – идентификатор, с помощью которого функция вызывается для выполнения;

список формальных параметров – определяет типы и имена формальных параметров, т.е. переменных функции, с помощью которых производится обмен дан-

ными между вызывающей и вызываемой функциями в процессе выполнения программы;

тело функции состоит из определения внутренних переменных и операторов;

return – оператор возврата, которым обязательно заканчивается функция, возвращающая значение в точку её вызова;

выражение преобразуется к типу результата, заданного в заголовке функции и возвращается в точку вызова функции.

Если функция возвращает значение результата с помощью оператора return, то такую функцию можно использовать в выражениях правой части оператора присваивания и во всех выражениях, где допустимо значение результата функции. В качестве результата функция не может возвращать массив или функцию, но может возвращать указатель на массив или функцию.

2) Функции, не возвращающие значения в вызываемую программу.

Синтаксис записи:

```
[класс памяти]  void  имя функции (список и определение формальных параметров)
{
  тело функции (операторы);
  [return();]
}
```

Выполнение такой функции завершается либо оператором возврата без параметров (return();), либо после того как выполняется тело функции.

Если в программе есть обращение к функции, то выше определения функции должно располагаться её описание, такое описание называется прототипом функции, оно полностью совпадает с заголовком функции.

Синтаксис записи прототипа функции:

[класс памяти] тип функции или void имя функции(список и определение формальных параметров);

Прототип функции должен размещаться в разделе внешних параметров, сразу за директивами препроцессора. Прототипы функции указывать не обязательно, если

главная функция ***main()*** записана в программе после всех других функций. Если выше, то обязательно. Если в описании функции не указан класс памяти, то по умолчанию берётся внешний.

Вызов функции активизирует её, т.е. передаёт ей управление и фактические параметры.

Синтаксис вызова функции:

имя функции (список фактических параметров);

или

указатель на функцию (список фактических параметров);

где ***указатель на функцию*** – это выражение, имеющее в своём значении адрес некоторой функции.

Формальные параметры – это переменные, которые принимают значения при обращении к функции в соответствии с порядком следования их имен в списке параметров. Идентификаторы формальных параметров не могут совпадать с идентификаторами, объявленными внутри функции. При вызове функции осуществляется передача фактических параметров в эту функцию, т.е. происходит присваивание значений фактических аргументов формальным параметрам.

Фактические аргументы могут быть выражениями, константами и переменными базового типа. Массивы и функции не могут передаваться как параметры, но могут передаваться указатели на массивы и функции.

Между формальными и фактическими параметрами при вызове функций, должны соблюдаться правила соответствия по последовательности и типам. Передача параметров при вызове функции происходит только по значению, поэтому выполнение функции не может изменить значения переменных, указанных в качестве фактических параметров.