2.6.3. Массивы

Паскаль	Си
Объявление:	
Var	
a: array[1..10] of char;	char a[10];
b: array[1..5, 1..10] of char;	char b[5][10]; целые и char совместимы в Си
Обращение к элементам:	
a[1]:=’1’;	a[0]=’1’; или a[0] = 25;
b[1,2]:=’z’;	b[0][1]=’z’; или b[0][1] = 25;

Замечания по массивам в Си:

1) индексы элементов по каждой размерности заключаются в **отдельную пару** квадратных скобок. Если этого не сделать, то будет выполнена операция запятая, и получим, что b[0 , 1] эквивалентно b[1], потому что то значение, что справа (1), берется в качестве результата, а 0 при этом не учитывается;

2) индексы элементов **начинаются с нуля**;

3) в качестве индексов в Паскале могут использоваться переменные любого (кроме longint) порядкового типа, а в Си - целого, символьного или перечисляемого (enum);

4) при объявлении массива указывается **число элементов** по каждой размерности

char a[2][3]

a[0][0]	a[0][1]	a[0][2]
a[1][0]	a[1][1]	a[1][2]

1

2 строки 3 колонки

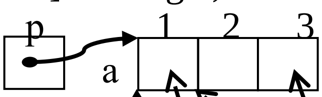
5) в отличие от Паскаля **в Си разрешена адресная арифметика**, и следующие выражения идентичны (для одномерного массива a):

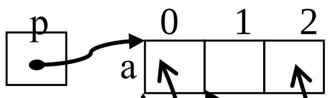
```
   Паскаль                       Си
  -----------              -------------
     a[i]                   a[i]
                            *(a + i)
```

имя ***a*** интерпретируется как указатель на первый элемент массива (элемент с индексом 0)

Пример: использование индексов и указателей при работе с массивами

Паскаль	Си
Type	typedef
t=array[1..3] of integer;	int t[3]; (имя типа)
Var	
a: t;	t a;
p: ^t;	t *p; ≠ int *p;
	void main(void)
begin	{
1) p:=@a;	1) p=a; или p=&a[0];
3) ~~p:=p+1;~~ так нельзя в Паскале	3) p=p+1; или p++;
2) ~~(p^):=25;~~	2) *p=25;
4) (p^)[3]:=25;	4) a[2]=10; или *(p+2)=10;
обращение к последнему элементу	
end.	}





6) имя массива является **константой-указателем (адресом)** на начало области памяти, выделенной под массив. Из этого факта имеются два очевидных следствия:

- нельзя, как в Паскале, присвоить один массив другому;
- имени массива нельзя присвоить результат операции адресной арифметики (увеличение и уменьшение) или нельзя присвоить строковый литерал:

char a[10]; ***a = "строка";*** //присваивание константе недопустимо

Type
  t=array[1..3] of integer;
Var
  a,b: t;
Var
  p: ^t;
 	
begin
  p:=@a;
  a:=b; ----Можно!
  a:=a+1; -------- Нельзя!
  p:=p+1; -------- Нельзя!
end.
	typedef
	  int t[3];
	  имя типа
	t a,b;
	
	t *p;
	void main(void)
	{
	p=a;
	a=b; ---------- Нельзя!
	a=a+1; -------- Нельзя!
	p=p+1; или p++; ------- Можно!
	}

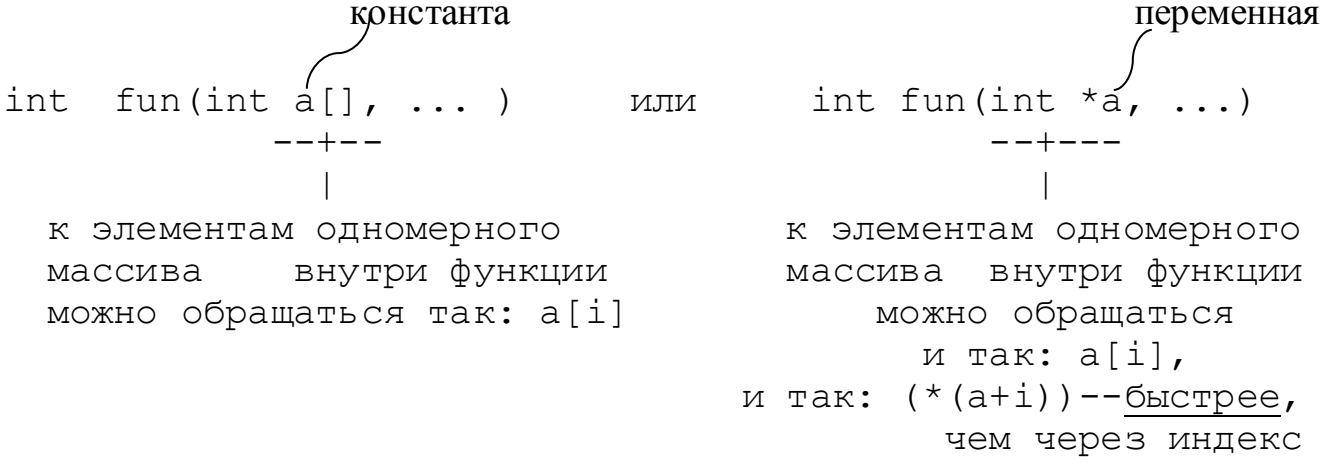
7) в программе на Си при описании **массива как параметра** (в виде указателя на тип элементов массива) в функцию она (функция) не получает сведений о размере массива (массив передается не как набор значений, а как **адрес нулевого элемента массива)**. Следующие два описания прототипа функции *f* эквивалентны:

2

```
int f(int *p );
int f(int p[] );
```

Если в каждом из этих случаев при вызове функции в качестве фактического параметра передать имя массива, то внутри тела функции указатель *p* будет указывать на 0-й элемент переданного массива.

Достоинством данной особенности (описания массива) является то, что это позволяет (в отличие от Паскаля) передавать в одну и ту же функцию массивы разной размерности (но одного типа элемента). **Заголовок функции**, которой передается массив, обычно имеет вид:



Недостатки данной особенности: надо не забыть кроме имени массива передать в функцию и информацию о числе элементов массива в виде значения отдельного фактического параметра (или использовать в теле функции вызов функции *sizeof(имя массива)*);

8) как и в Паскале, в Си можно ссылаться или на весь массив, или на отдельный элемент массива, или на **подмассив**: если для К-мерного (К индексов) массива указать только первые Р индексов, то это будет ссылка на (К-Р)-мерный подмассив данного массива. Так для массива вида

int a[N1][N2][N3]
К==3 (К-Р) == 2

a[i] - это ссылка на 2-мерный подмассив массива *a* размерностью N2*N3, а a[i][j] - ссылка на одномерный подмассив массива *a* размерностью N3;

p==1 К-Р==1 p==2

9) в Си глобальные массивы можно инициализировать при объявлении (как типизированные константы - массивы в Паскале):

Паскаль	Си
`const`	
`a:array[1..4] of integer=(1,2,3,4);`	`int a[4] = {1,2,3,4};`
`const`	
`b:array[1..3][1..4] of integer= ((1,1,1,1), (2,2,2,2), (3,3,3,3));`	`int b[3][4] = {{1,1,1,1}, --- 1-я строка {2,2,2,2}, --- 2-я строка {3,3,3,3}};--- 3-я строка`

3

При **неполной инициализации** (нельзя было на Паскале) те элементы, которым не заданы начальные значения, **получат нулевые значения:**

```
int c[3][3] = {
              {1},      /*  задаётся c[0][0],  не задаются c[0][1] и c[0][2]  */
              {0,1},    /*  задаются c[1][0] и c[1][1],  не задается c[1][2]  */
              {1}       /*  задаётся c[2][0],  не задаются c[2][1] и c[2][2]  */
             }
```

2.6.4. Строки

В отличие от Паскаля в Си нет встроенного строкового типа. Правда, в C++ есть класс string.

Паскаль	Си
var	указатель на char
a: pchar;	char *a; /*строка **переменной** длины*/ 1)
b: string[10];	char b[11]; /*строка (массив символов) 2) **постоянной** длины*/
s: string; (≡ string[255])	char s[256]; /*строка (массив символов) 3) **постоянной** длины*/
	имя типа; добавили 1 символ для конца строки
type t = string[255];	typedef char t[256];
Var S : ^t;	t *s;
	имя типа; указатель на массив символов

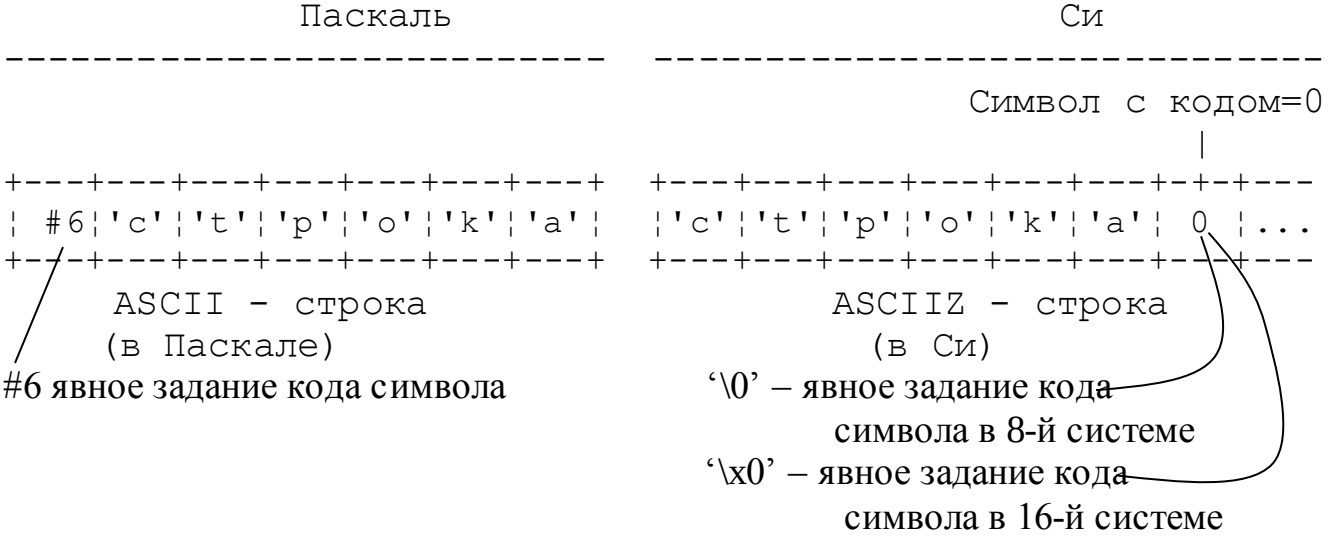
Замечания по строкам в Си:

1. Для работы со строками используется либо **указатель** (указатель на char) на начало строки переменной длины, либо **символьный массив**.

2. Хранятся строки в Паскале и Си по-разному. В Си длина строки не хранится в самой строке, а опознается по положению **нуль-символа** (он является признаком конца строки). Под этот символ надо не забывать выделить память (при объявлении или при выполнении - динамически).

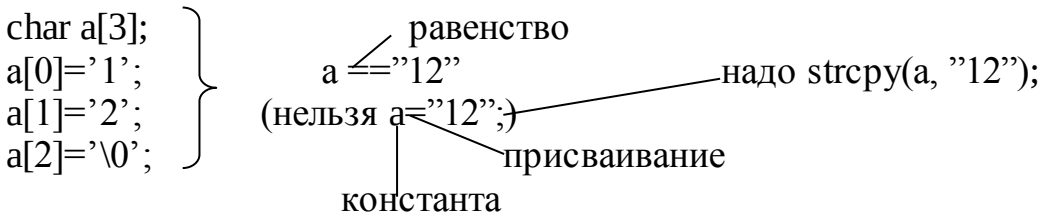
Замечание: в строковую константу символ кодом ноль входит (хотя не изображается в записи константы). Например, строка “12” занимает 3 байта в памяти.

4



Все функции для работы со строками в Си ориентированы именно на **наличие в конце** последовательности символов символа с кодом 0.

3. Если строка **формируется посимвольно** (инициализируется присваиванием), то нуль-символ должен обязательно вручную записываться в конец строки. С учетом необходимости записи в конец строки этого символа в случае если строка хранится как **массив символов**, то число элементов такого массива должно выбираться на 1 больше (под символ с кодом 0), чем нужно для собственно символов в строке.



4. Несмотря на различия в хранении строк, можно и в Паскале, и в Си обращаться к элементам строки как к элементам одномерного массива (через индекс, начиная в Си с нуля).

5. Необходимо **различать**:

c - имя переменной-носителя строки (указатель или массив);

'c' - символьная константа. Занимает **один байт**;

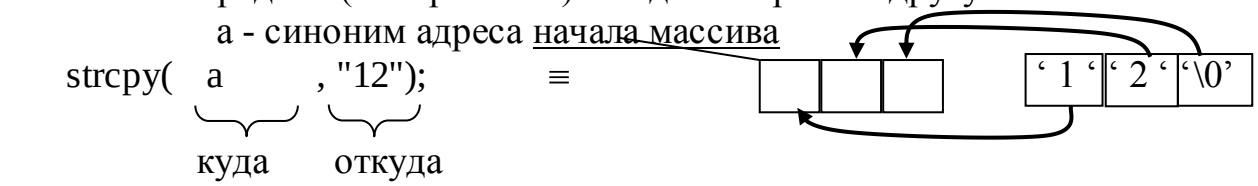
"c" - строковая константа. Занимает **два байта** (один под символ с кодом ноль).

6. При **объявлении строки как массива символов** память выделяется под массив символов, имя массива является **константным указателем** на начало области памяти (последовательности байт), и **<u>нельзя прямо присвоить массиву символов строковый литерал</u>**:

```
char a[3];
a = "12"; --- нельзя
a++ ≡ a = a+1; --- нельзя! (но char a[3] = {'1','2','3','\0'}; -- можно и
                            char a[]  = "12"; - тоже можно)
(a – константный указатель)
```

5

Вместо этого надо использовать функцию strcpy (или подобные) для побайтной передачи (копирования) из одной строки в другую:



При этом нужно не забыть подключить заголовочный файл #include <string.h> для функций, работающих со строками:

strcpy() ≡ copy

strlen() ≡ length

strcat() ≡ concat .

Однако **можно напрямую читать в *s***, используя функции scanf() или gets():

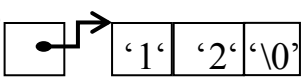
```
char s[80];
gets(s);  или   scanf( "%s", &s );
```

формат (строка) адрес места, куда надо ввести.

7. При **объявлении строки как указателя** на char

```
char *s;
```

память выделяется только под указатель (этот указатель является переменной, не константой), под сам массив символов память не выделяется.



При этом **можно** выполнить присваивание вида (строковый литерал)

```
s = "12";
```

при котором указателю *s* будет присвоен адрес начала размещения строкового литерала (с нулем в конце) в памяти. Операцию s++; в данном случае **можно** использовать, так как s – переменная, а не константа.

Можно puts(s); и printf("s[0] =%c", s[0]);.

Но при этом **нельзя** как было выше char s[] = {'1','2','3','\0'};

Также **нельзя** s[0] = '3'; //s[0] - это часть константы (строкового литерала)

Также при этом **нельзя** (если **не выделено место в памяти под хранение символов строки**) использовать

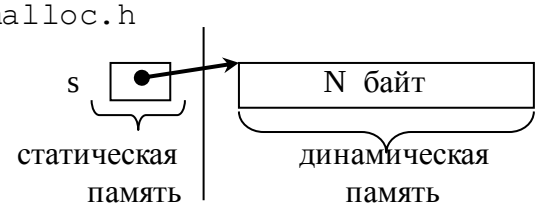
gets(s); или scanf("%s", &s);

.

В данной ситуации (при необходимости ввода значения в строку переменной длины) надо будет предварительно выделить память под будущие элементы строки:

```
s = (char *) malloc(N); //malloc возвращает адрес типа void
```

указатель; приведение типа; нужен malloc.h



6

Здесь выделяется N байт памяти (из кучи) и *s* присваивается адрес начала этой области.

То же самое в Си++ можно записать так:

```
s=new char [N]; // потом надо будет освободить: delete [ ] s;
```

оператор выделения динамической памяти

Замечание: результатом функции malloc() является значение типа void (неопределенный тип), поэтому к нему (результату) необходимо применить операцию **явного преобразования типа** (вида тип *).

Отметим некоторые **аналогичные функции для работы со строками** в Паскале и Си:

Паскаль	Си
length	strlen
copy	strcpy
concat	strcat
str	sprintf
val	atoi

Прототипы этих функций в Си находятся в string.h и stdlib.h.