УКАЗАТЕЛИ И МАСИВИ

Ненко Табаков
Пламен Танов
Технологическо училище “Електронни системи”
Технически университет – София

Версия 0.5

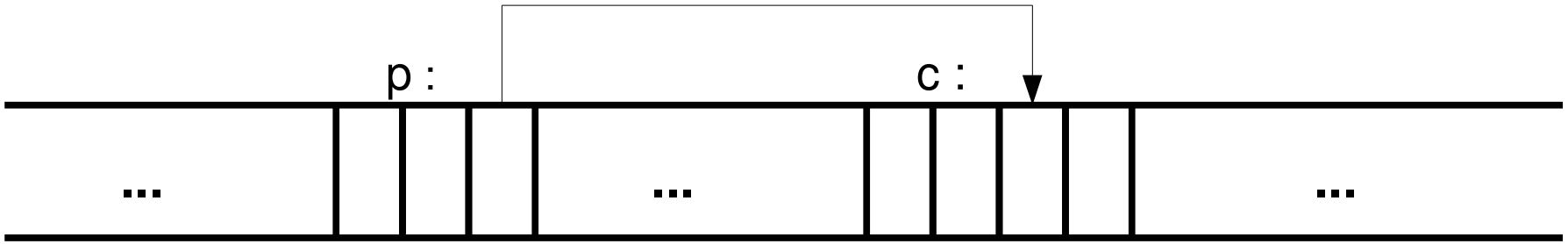
ВЪВЕДЕНИЕ

- Указателите са променливи, които съдържат адреса на други променливи.
- Указателите са широко използвани в C програми. Отчасти това е така, защото това е единственият начин решиш дадения проблем, а също и защото водят до по – ефективен и компактен код.
- ANSI C дефинира ясни правила, как да се използват указатели. Невнимателната им употреба често води до създаване на указатели, които съдържат неочаквани адреси

УКАЗАТЕЛИ И АДРЕСИ

Типично паметта е разделена на клетки. Клетките са последователно подредени и всяка от тях има адрес. Клетките в паметта може да се използват индивидуално или в групи.

Указател е група от клетки, които съдържат адрес.



УКАЗАТЕЛИ И АДРЕСИ

Указатели се дефинират по следния начин:
*тип *име_на_указателя*

- оператор & - връща адреса на променливата
- оператор *- връща стойността на променливата, към която сочи указателя

```
int x = 1, y = 2, z[10];
int *ip; /*ip е указател към int*/
ip = &x; /*ip сочи към x*/
y = *ip; /*y приема стойност 1*/
*ip = 0; /*x приема стойност 0*/
ip = &z[0]; /*ip сочи към z[0]*/
```

УКАЗАТЕЛИ И АДРЕСИ

• Указателите са ограничени да сочат към определен тип данни, който се указва при декларирането им (изключение правят указатели към тип `void`.
• Операторите & и * имат по – висок приоритет от аритметичните оператори

```
(*ip)++ не е еквивалентно на *ip++
```

• Тъй като указателите са променливи, то те може да се използват и без оператора *
В разгледания случай `iq` и `ip` са указатели от тип `int`

```
int *ip;
int *iq;
int n = 10;
ip = &n;
iq = ip; /*iq също сочи към n*/
```

УКАЗАТЕЛИ И АРГУМЕНТИ НА ФУНКЦИИ

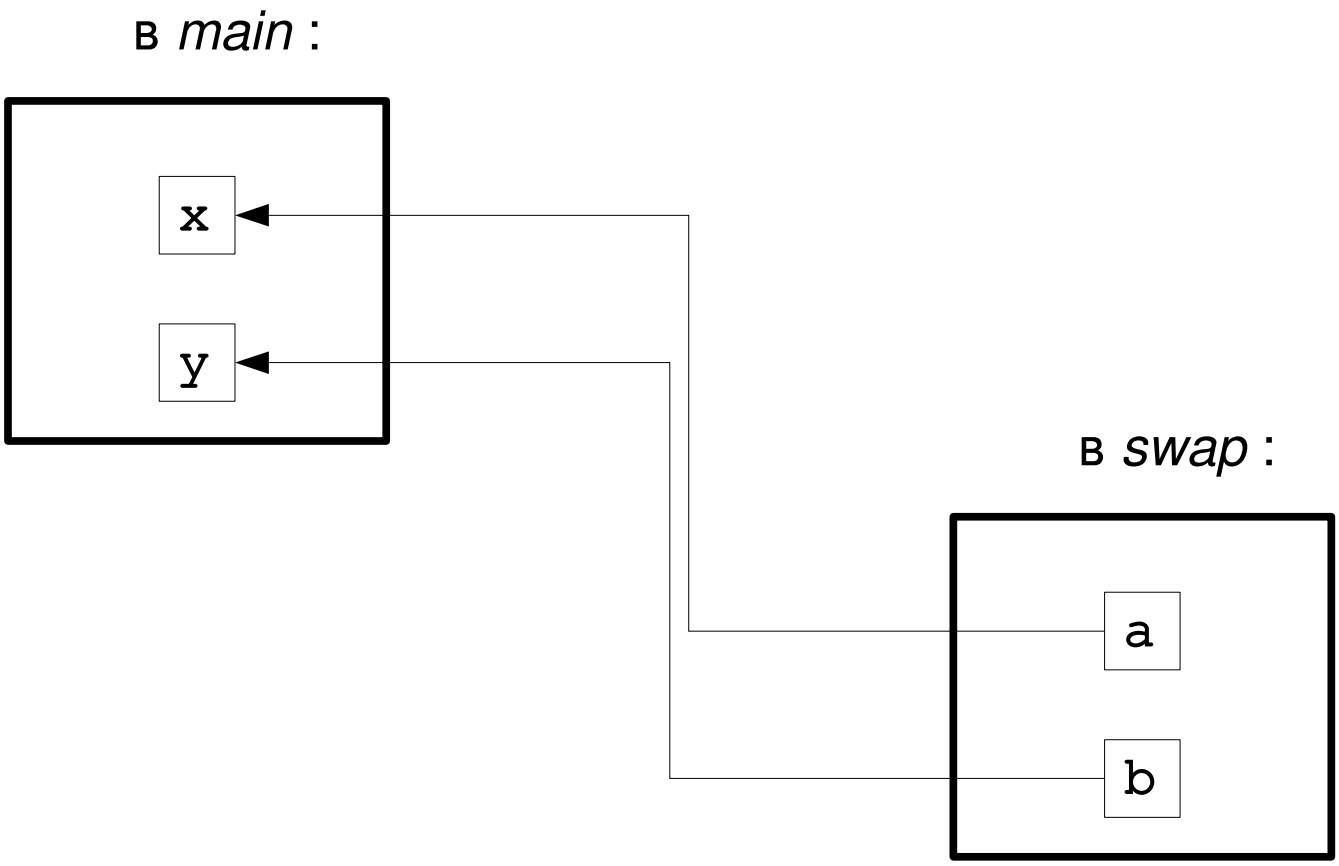
В C аргументите на една функция се предават по стойност, затова няма директен начин за една функция да промени стойността на някой от аргументите си.

```
void swap (int a, int b) /*ГРЕШНО*/
{
   int temp;

   temp = a;
   a = b;
   b = temp;
}
```

ПРИМЕР

```
void swap (int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}

int main ()
{
    int x = 2;
    int y = 4;
    swap (&x, &y)
}
```

ПРИМЕР



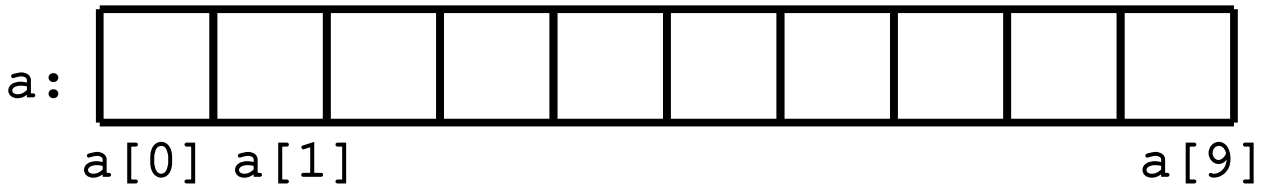
ПРИМЕР

```
/* getint: get next integer from input into *pn */
int getint(int *pn)
{
   int c, sign;
   while (isspace(c = getch())) /* skip white space */
      ;
   if (!isdigit(c) && c != EOF && c != '+' && c != '-') {
      ungetch(c); /* it is not a number */
   return 0;
   }
   sign = (c == '-') ? -1 : 1;
   if (c == '+' || c == '-')
      c = getch();
   for (*pn = 0; isdigit(c), c = getch())
      *pn = 10 * *pn + (c - '0');
   *pn *= sign;
   if (c != EOF)
      ungetch(c);
   return c;
}
```

УКАЗАТЕЛИ И МАСИВИ

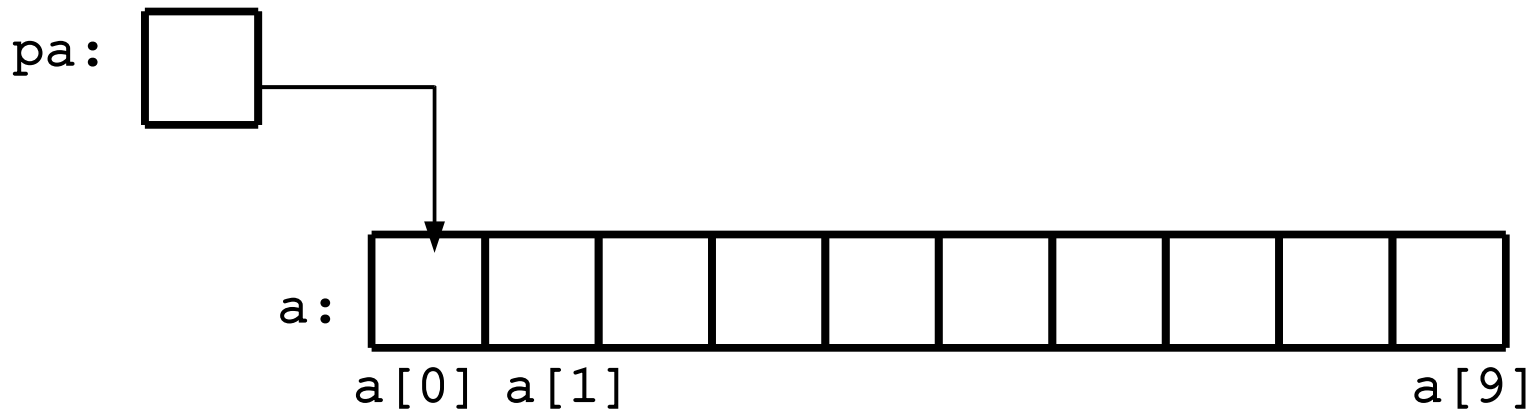
В C има силна връзка между указатели и масиви. Всяка операция, която може да бъде извършена с масиви, може да се направи и с указатели. Като цяло версията с указатели трябва да работи по – бързо, но е по - неразбираема при четене от човек.

```
int a[10];
```



УКАЗАТЕЛИ И МАСИВИ

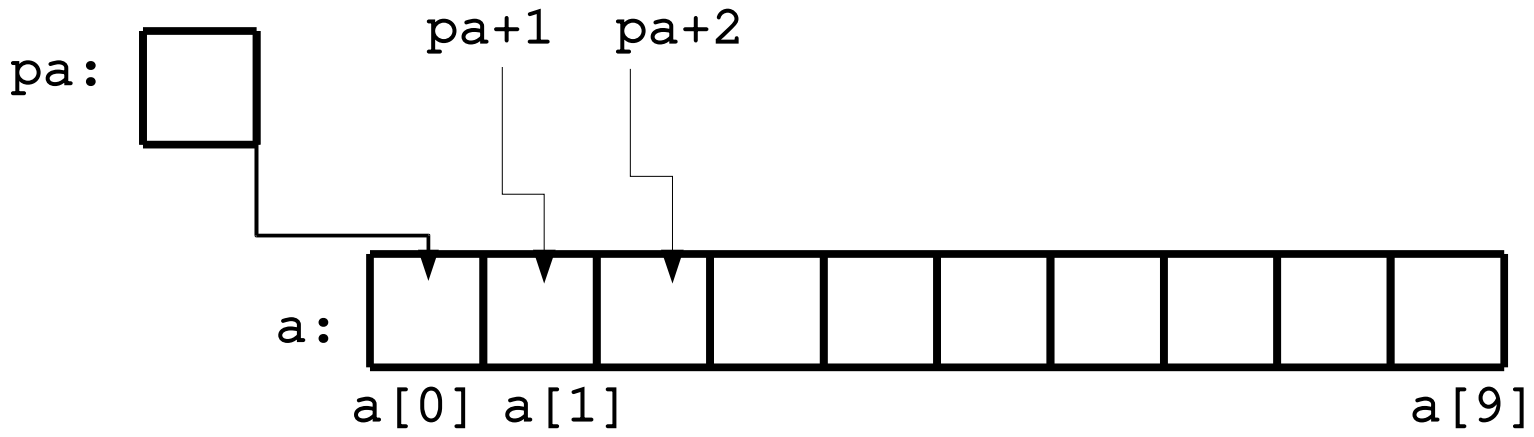
```
int a [10];
int *pa;
int x;
pa = &a[0]; /* pa сочи към първия елемент от масива*/
x = *pa; /* x приема стойността на първия елемент от масива*/
```



УКАЗАТЕЛИ И МАСИВИ

```
x = *(pa + 1);
i = 5;
x = *(pa + i);

for (i = 0; i<10; i++)
   printf („%d“, *(pa+i));
```

```
x = a[1];
i = 5;
x = a[i]

for (i = 0; i<10; i++)
   printf („%d“, a[i]);
```



УКАЗАТЕЛИ И МАСИВИ

Елементите на един масив могат да се достигат чрез индексите му или чрез адресна аритметика. По дефиниция името на масива е указател към първия елемент от него.

```
pa = &a[0];
x = a[5];

еквивалентно на

pa = a;
x = *(a+5);
```

УКАЗАТЕЛИ И МАСИВИ

Важна разлика между името на масив и указател е, че указателят е променлива и изразът

```
pa++;
```

е валиден, докато написан с име на масив е грешка

```
a[i] еквивалентно на *(a+i)

for (i = 0; i<10; i++)
   printf („%d“, *(a+i));

&a[i] еквивалентно на a + i

a++; /*Грешно!*/
```

ПРИМЕР

```
/* strlen: return length of string s */
int strlen(char *s)
{
   int n;
   for (n = 0; *s != '\0', s++)
      n++;
   return n;
}

int main ()
{
   char c[] = „hello world“;
   printf („%d“, strlen(c));
}
```

АДРЕСНА АРИТМЕТИКА

Ако `p` е указател към елемент от масив, то `p++` инкрементира стойността на `p` той вече сочи към следващия елемент от масива. По същия начин `p +=i` увеличава стойността на `p` с `i` и `p` ще сочи към `i`-тия елемент спрямо този, който е сочил преди прибавянето.
Тези и подобни конструкции се наричат адресна аритметика.

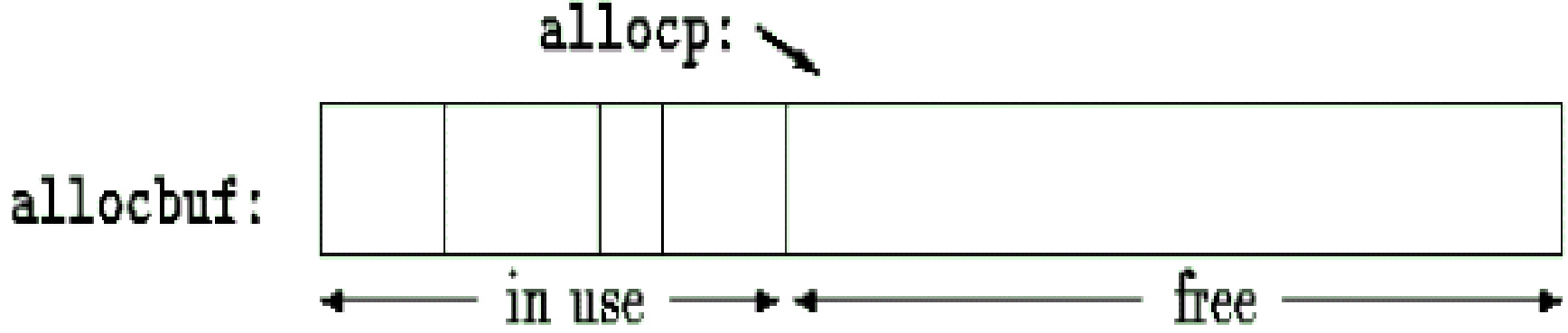
```
int main ()
{
   int a[10];
   int *pa;
   int i = 5;
   pa = a; /* pa сочи към първия елемент от масива*/
   pa++;  /* pa сочи към втория елемент от масива*/
   pa +=i; /* pa сочи към седмия елемент от масива*/
}
```

ПРИМЕР

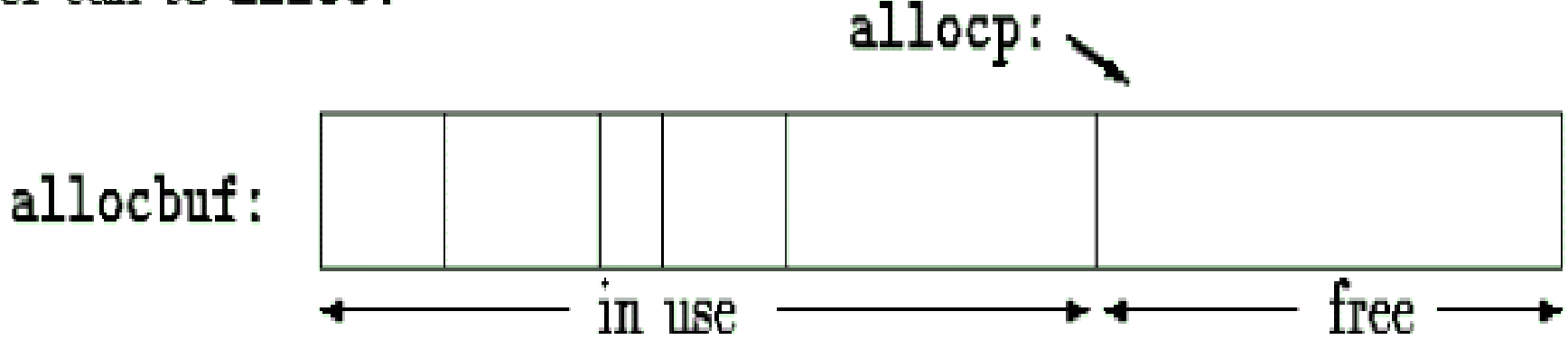
```
/* strlen: return length of string s */
int strlen(char *s)
{
   char *p = s;
   while (*p != '\0')
      p++;
   return p - s;
}

int main ()
{
   printf („%d“, strlen(„hello world“));
}
```

ПРИМЕР

before call to **alloc**:



after call to **alloc**:



ПРИМЕР

```
#define ALLOCSIZE 10000 /* size of available space */
static char allocbuf[ALLOCSIZE]; /* storage for alloc */
static char *allocp = allocbuf; /* next free position */

char *alloc(int n) /* return pointer to n characters */
{
   if (allocbuf + ALLOCSIZE - allocp >= n) { /* it fits */
      allocp += n;
      return allocp - n; /* old p */
   } else /* not enough room */
      return 0;
}

void afree(char *p) /* free storage pointed to by p */
{
   if (p >= allocbuf && p < allocbuf + ALLOCSIZE)
      allocp = p;
}
```

СИМВОЛНИ УКАЗАТЕЛИ И ФУНКЦИИ

- „I am a string“ е константа, чийто край се определя със знака '\0'
- Когато един стринг, който е константа, се предава като аргумент на функция, досъпът до елементите му става чрез символен указател (указател към `char`)

```
int strlen(char *s)
{
   char *p = s;
   while (*p != '\0')
      p++;
   return p - s;
}

int main ()
{
   printf („%d“, strlen(„I am a string“));
}
```

СИМВОЛНИ УКАЗАТЕЛИ И ФУНКЦИИ

```
char *pa;
pa=„I am a string“; /*pa сочи към стринг, който е константа*/
```

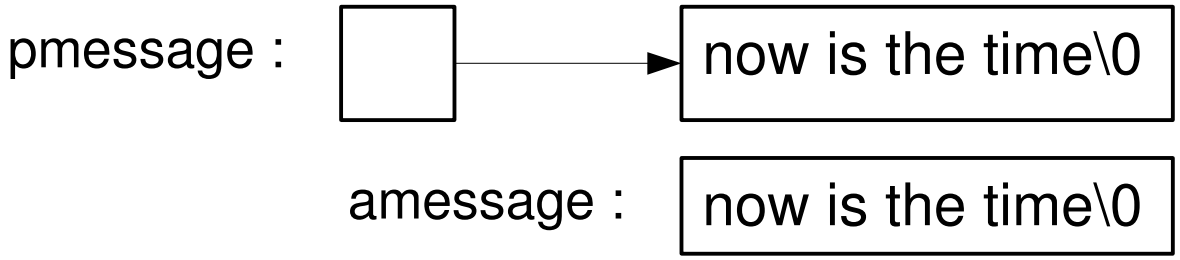
В този указателят `pa` само сочи към стринга. Това не е копие на символния низ. Следващите две дефиниции не са еквивалентни.

```
char amessage[] = „now is the time“; /* масив */
char *pmessage = „now is the time“; /* указател */
```

СИМВОЛНИ УКАЗАТЕЛИ И ФУНКЦИИ

`amessage` е масив, достатъчно голям да съдържа последователността от символи, които завършват с терминиращия знак '\0'. Някои от елементите в масива може да променят стойностите си, но `amessage` винаги ще сочи към едно и също място в паметта.

`pmessage` е указател, който е инициализиран да сочи към стринг. Той може да бъде променен да сочи към друг адрес.



ПРИМЕР

```
/* strcpy: copy t to s; array subscript version */
void strcpy(char *s, char *t)
{
   int i;
   i = 0;
   while ((s[i] = t[i]) != '\0')
      i++;
}
```

```
/* strcpy: copy t to s; pointer version */
void strcpy(char *s, char *t)
{
   int i;
   i = 0;
   while ((*s = *t) != '\0') {
      s++;
      t++;
   }
}
```

ПРИМЕР

```
/* strcpy: copy t to s; pointer version 2 */
void strcpy(char *s, char *t)
{
   while ((*s++ = *t++) != '\0')
      ;
}
```

```
/* strcpy: copy t to s; pointer version 3 */
void strcpy(char *s, char *t)
{
   while (*s++ = *t++)
      ;
}
```

ПРИМЕР

```
/* strcmp: return <0 if s<t, 0 if s==t, >0 if s>t */
int strcmp(char *s, char *t)
{
   int i;
   for (i = 0; s[i] == t[i]; i++)
      if (s[i] == '\0')
         return 0;
   return s[i] - t[i];
}
```

```
/* strcmp: return <0 if s<t, 0 if s==t, >0 if s>t */
int strcmp(char *s, char *t)
{
   for ( ; *s == *t; s++, t++)
      if (*s == '\0')
         return 0;
   return *s - *t;
}
```

МНОГОМЕРНИ МАСИВИ

C подържа многомерни масиви, като няма ограничение колко измерения може да има един масив. Най – често използваният многомерен масив е двумерният.
Един двумерен масив може да бъде разгледан като едномерен масив, на който всеки елемент е масив

```
int arr[10][100]; /*двумерен масив с 10 реда и 100 колони*/

arr[0][5];
arr[i][j];
```

МНОГОМЕРНИ МАСИВИ

Когато двумерен масив се предава като аргумент на функция, декларацията на масива задължително трябва да съдържа броя на колоните.

Броят на редовете не е задължителен.

```
void f (int arr[10][100]){
    ...
}

void f(int arr[][100]){
    ...
}
```

ПРИМЕР

```
static char daytab[2][13] = {
   {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
   {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
};

/* day_of_year: set day of year from month & day */
int day_of_year(int year, int month, int day)
{
   int i, leap;
   leap = (year%4 == 0 && year%100 != 0 || year%400 == 0);
   for (i = 1; i < month; i++)
      day += daytab[leap][i];
   return day;
}
```