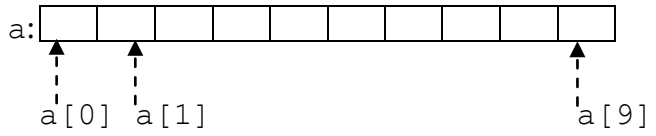
Указатели и массивы

Описание

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

определяет массив a размера 10, т.е. блок из 10 последовательных объектов с именами a[0], a[1] ,... ,a[9].



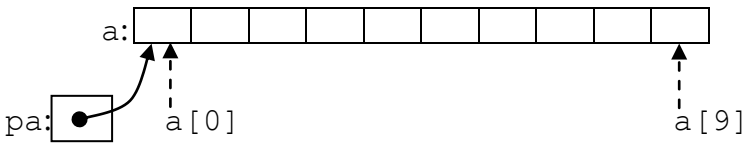
Запись a[i] означает i-й элемент массива. Если pa есть указатель на int, т.е. определен как

```
int *pa;
```

то в результате присваивания

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

pa будет указывать на нулевой элемент a; иначе говоря, pa будет содержать адрес элемента a[0].



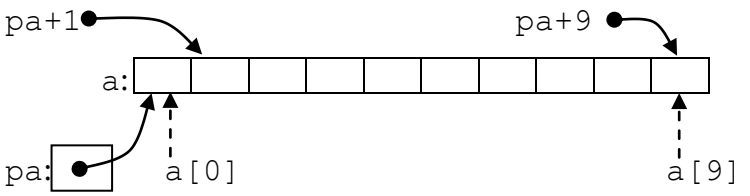
Теперь присваивание

```
x = *pa;
```

будет копировать содержимое a[0] в x.

Если pa указывает «на некоторый элемент массива», то pa+1 по определению указывает на следующий элемент, pa+i — на i-й элемент после pa, а pa-i — на i-й элемент перед pa. Таким образом, если pa указывает на a[0], то *(pa+1)

есть содержимое a[1], pa+i адрес a[i], а *(pa+i) содержимое a[i].



Сделанные замечания верны безотносительно к типу и размеру элементов массива a. Смысл слов «добавить 1 к указателю», как и смысл любой арифметики с указателями, в том, чтобы pa+1 указывал на следующий объект, а pa+i — на i-й после pa.

Между индексированием и арифметикой с указателями существует очень тесная связь. Имя массива означает массив как единый объект, но, если оно не является операндом операции `sizeof` или операндом адресной операции &, то значение массива преобразуется в указатель на его первый элемент, т.е. значением переменной или выражения типа массив (за исключением двух указанных случаев) является адрес нулевого элемента массива. После присваивания

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

`pa` и `a` имеют одно и то же значение. Поскольку имя массива есть не что иное, как адрес его начального элемента, присваивание `pa=&a[0]`

можно также записать в следующем виде:

```
pa = a;
```

В языке Си справедливо следующее утверждение: записи `A[B]` и `*((A)+(B))` эквивалентны. Например, встречая запись `a[i]`, компилятор сразу преобразует ее в `*(a+i)`. Из этого следует, что записи `&a[i]` и `a+i` также будут эквивалентными, т.е. и в том и в другом случае это адрес i-го элемента после `a`. С другой стороны, если `pa` — указатель, то в выражениях его можно использовать с индексом, т.е. запись `pa[i]` эквивалентна записи `*(pa+i)`. Элемент массива одинаково разрешается изображать и в виде указателя со смещением и в виде имени массива с индексом.

Между именем массива и указателем, выступающим в роли имени массива, существует одно различие. Указатель — это переменная, поэтому можно написать `pa=a` или `pa++`. Но имя массива не является переменной и записи типа `a=pa` или `a++` — не допускаются.

Если имя массива передается функции, то последняя получает в качестве аргумента адрес его начального элемента. Внутри вызываемой функции этот аргумент является локальной переменной, содержащей адрес. Мы можем воспользоваться отмеченным фактом и написать еще одну версию функции `strlen`, вычисляющей длину стринга.

Еще одна версия функции `strlen`:

```
/* strlen: возвращает длину стринга s */
int strlen(char *s)
{
 char *p = s;
 while (*p != '\0')     /* можно эту строку еще сократить:  while(*p)  */
        p++;
 return p - s;
}
```

Существует важное различие между следующими определениями:

```
  char amessage[] = "now is the time"; /* массив */
  char *pmessage = "now is the time"; /* указатель */
```

amessage - это массив, имеющий такой объем, что в нем как раз помещается указанная последовательность литер и '\0'. Отдельные литеры внутри массива могут изменяться, но amessage всегда ссылается на одно и то же место памяти. В противоположность ему pmessage есть указатель, инициализированный ссылкой на строковую константу. А значение указателя можно изменить, и тогда последний будет ссылаться на что-либо другое.

```
/* strcpy: копирует t в s; версия с указателями*/
void strcpy(char *s, char *t)
{
while ((*s++ = *t++) )

         ;

}
```

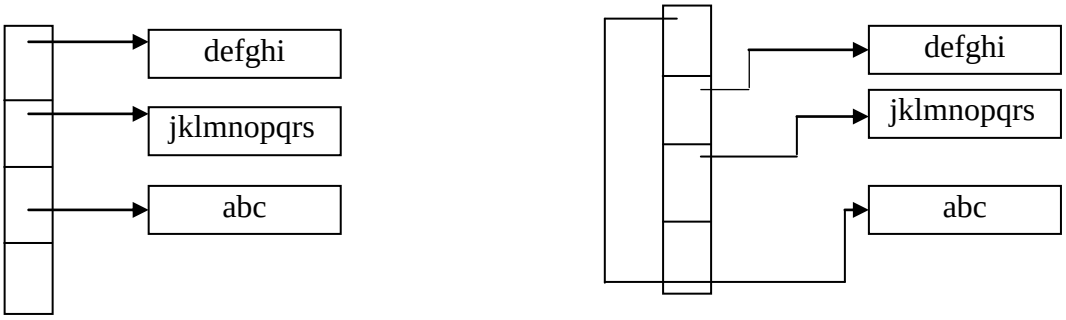
Функция лексикографического сравнения строк

```
/* strcmp: выдает <0 при s<t, 0 при s==t, >0 при 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 (unsigned)s[i] – (unsigned)t[i];
}
```

Та же программа с использованием указателей записывается так:

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

Массивы указателей и указатели на массивы




```
char *lineptr[MAXLINES];
```

В этом описании сообщается, что `lineptr` есть массив из `MAXLINES` элементов, каждый из которых представляет собой указатель на char. Иначе говоря, `lineptr[i]` — указатель на литеру, а `*lineptr[i]` — литера, на которую он указывает (первая литера i-й строки текста).

Пример инициализации многомерного массива

```
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}
};
```

При вызове функции массив не передается, передается указатель на его первый элемент:

```
    f(char daytab[2][13]) { ... }
```

Вместо этого можно записать

```
    f(char daytab[] [13]) { ... }
```

поскольку число строк здесь не имеет значения, или

```
    f(char (*daytab)[13]) { ... }
```

Инициализация массивов указателей

```
/* month_name: возвращает имя n-го месяца */
char *month_name(int n)
{
 static char *name[] ={
  "Неверный месяц",
  "Январь", "Февраль", "Март",
  "Апрель", "Май", "Июнь",
  "Июль", "Август", "Сентябрь",
  "Октябрь", "Ноябрь", "Декабрь"
 };
 return (n < 1 || n > 12) ? name[0] : name[n];
}
```

Цитата из K&P:

«Начинающие программировать на Си иногда не понимают, в чем разница между двумерным массивом и массивом указателей типа name из приведенного примера. Для двух следующих определений:

```
int a[10][20];
int *b[10];
```

записи `a[3][4]` и `b[3][4]` будут синтаксически правильными ссылками на некоторое значение типа `int`. Однако только a является истинно двумерным массивом: для двухсот элементов типа int будет выделена память, а вычисление смещения элемента `a[`*`строка`*`][`*`стобец`*`]` от начала массива будет вестись по формуле `20*`*`строка`*`+`*`столбец`*, учитывающей его прямоугольную природу. Для `b` же определяются только `10` указателей, причем без инициализации. Инициализация должна задаваться явно — либо статически, либо в процессе счета. Предположим, что каждый элемент `b` ссылается на двадцатиэлементный массив, в результате где-то будут выделены пространство, в котором разместятся `200` значений типа `int`, и еще `10` ячеек для указателей. Важное преимущество массива указателей в том, что строки такого массива могут иметь разные длины. Таким образом, каждый элемент `b` не обязательно ссылается на двадцатиэлементный вектор; один может ссылаться на два элемента, другой — на пятьдесят, а некоторые и вовсе могут ни на что не ссылаться.»

Функции работы со строками

Эти функции определены в головном файле **<string.h>**. Если копирование имеет дело с объектами, перекрывающимися по памяти, то поведение функций не определено. Функции сравнения рассматривают аргументы как массивы элементов типа unsigned char.

```
char *strcpy(char *s, const char *ct)
```
копирует строку ct в строку s, включая '\0'; возвращает s.

```
char *strcat (char *s, const char *ct)
```
приписывает ct к s; возвращает s.

```
char strcmp(const char *cs, const char *ct)
```
сравнивает cs с ct; возвращает значение меньшее 0, если cs<ct; равное 0, если cs==ct, и большее 0,если cs>ct.

```
char *strchr(const char *cs, char c)
```
возвращает указатель на первое вхождение с в cs или, если такового не оказалось, NULL.

```
char *strrchr(const char *cs, char c)
```
возвращает указатель на последнее вхождение с в cs или, если такового не оказалось, NULL.

```
char *strstr(const char *cs, const char *ct)
```
возвращает указатель на первое вхождение ct в cs или, если такового не оказалось, NULL.

```
size_t strlen(const char *cs)
```
возвращает длину cs (без учета '\0').

```
char *strtok(char *s, const char *ct)
```
ищет в s лексему, ограниченную литерами из ct.

Последовательные вызовы функции strtok разбивают строку s на лексемы. Ограничителем лексемы может быть любая литера, входящая в строку ct. В первом вызове функции указатель s не равен NULL. Функция находит в строке s первую лексему, состоящую из литер, не входящих в ct; работа этого вызова завершается тем, что поверх следующей литеры пишется '\0' и возвращается указатель на выделенную лексему. Каждый последующий вызов функции strtok, в котором указатель s равен NULL, выдает указатель на следующую лексему, которую функция будет искать сразу за концом предыдущей. Функция возвращает NULL, если далее никакой лексемы не обнаружено. Параметр ct от вызова к вызову может варьироваться.

```
void *memcpy(void *s, const void *ct, size_t n)
```
копирует n литер из ct в s и возвращает s.

```
void *memset(void *s, char c, size_t n)
```
размещает литеру с в первых n позициях строки s и возвращает s.

Функции проверки класса литер

Головной файл **<ctype.h>** предоставляет функции, которые позволяют определить, принадлежит ли литера определенному классу. Параметр каждой из этих функций имеет тип int и должен быть либо значением unsigned char, приведенным к int, либо значением EOF; возвращаемое значение тоже имеет тип int. Функции возвращают ненулевое значение (“истину”), если аргумент принадлежит указанному классу литер, и нуль (“ложь”) – в противном случае.

`int isupper(int c)`	буква верхнего регистра
`int islower(int c)`	буква нижнего регистра
`int isalpha(int c)`	isupper(c) или islower(c) истины
`int isdigit(int c)`	десятичная цифра
`int isalnum(int c)`	isalpha (c) или isdigit (c) истины
`int isxdigit(int c)`	шестнадцатиричная цифра
`int isspace(int c)`	пробел, новая-строка, возврат-каретки, табуляция
`int isgraph(int c)`	печатаемая литера, кроме пробела
`int isprint(int c)`	печатаемая литера, включая пробел

Кроме этих функций в файле есть две функции, выполняющие преобразование букв из одного регистра в другой.

`int tolower(int c)`

если c – буква верхнего регистра, то tolower(c) выдаст эту букву на нижнем регистре; в противном случае она вернет c.

`int toupper(int c)`

если c – буква нижнего регистра, то toupper(c) выдаст эту букву на верхнем регистре; в противном случае она вернет c.