НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»
Кафедра информатики и процессов управления (№17)

Дисциплина «Информатика» (углубленный уровень), 1-й курс, 1-й семестр.

Методические указания

Тематическое занятие 9

Функции: работа с массивом.

Содержание

Передача массива в функцию 1
Имя массива как синоним указателя *1*
Массив как параметр функции *2*
Формальные параметры при передаче массива *2*
Доступ к любому элементу массива *3*
Способы передачи массива в функцию 3
Передача размера массива *3*
Передача указателей на начало и конец массива *4*
Защита содержимого массива *5*
Использование `const` *6*
Локальные и внешние массивы 6
Локальный массив *6*
Внешний массив *6*
Одинаковые имена и область действия *7*
Упражнения 8
Упражнение 9.1 ..... *8*
Упражнение 9.2 ..... *8*
Упражнение 9.3 ..... *8*

Передача массива в функцию

Имя массива как синоним указателя

Напомним, что идентификатор (имя) массива является *синонимом указателя* на 1-й элемент массива. Например, при объявлении массива `a`

```
int a[10];  /* массив из 10 элементов типа int */
```

его имя можно использовать для определения адресов элементов массива:

```
int *p1, *p2; /* указатели на переменную типа int */
p1 = &a[2]; /* адрес 3-го элемента массива a[2] */
p2 = a+2; /* адрес 3-го элемента массива a[2] */
```

К указателям `p1` и `p2` можно применять операции адресной арифметики. Но, значение самой переменной типа «массив» `a` изменять нельзя:

```
a = p1; /* НЕДОПУСТИМО! */
a += 3; /* НЕДОПУСТИМО! */
a++; /* НЕДОПУСТИМО! */
```

Массив как параметр функции

В языке С массив в функцию передается ***по адресу***, а не по значению. Для такого способа передачи массива при вызове функции не приходится тратить время и ресурсы на копирование значений всех элементов массива, передаваемого в качестве фактического параметра.

Если в функцию в качестве одного из фактических параметров передается имя массива, то туда поступает адрес первого элемента массива. В вызванной функции соответствующий ему *формальный параметр является указателем*.

Пример передачи массива в функцию:

```
#include <stdio.h>
void func(int a[]);
int main(void) {
     int i, mas[5]={1,2,3,4,5};  /* объявление массива mas */
     func(mas);  /* mas (фактический параметр) - массив */
     for (i=0; i<5; ++i)
          printf("mas[%d]=%d\n", i, mas[i]);
     return 0;
}
void func(int a[]) {  /* a (формальный параметр) - указатель */
     a[2] = 100;
}
```

Поскольку массив `mas` передается в функцию по адресу, то изменение значений его элементов может происходить внутри функции (`a[2]=100;`) с помощью формального параметра-указателя `a[]`.

Количество элементов в квадратных скобах формального параметра `a[]` обычно не указывают, оставляя скобки пустыми. В квадратных скобках можно указать количество элементов массива `void func(int a[5])`, но в этом нет смысла, поскольку тип формального параметра `a` является указателем на одну переменную типа `int`.

Результат работы программы:

```
mas[0]=1  mas[1]=2  mas[2]=100  mas[3]=4  mas[4]=5
```

Заметим, что прототип функции `func()` можно записать без указания имени:

```
void func(int []);
```

Формальные параметры при передаче массива

В приведенном примере реализован вариант, когда имя массива `a[]`, используемое в качестве формального параметра функции, является *указателем*, т.е. локальной переменной, содержащей адрес. При этом значение такого формального параметра `a` можно изменять (в отличие от переменной типа «массив»):

```
void func(int a[]) {  /* a (формальный параметр) - указатель */
     a++; /* Допустимо! */
     a[2] = 100;
}
```

Для этой функции результат:

```
mas[0]=1  mas[1]=2  mas[2]=3  mas[3]=100  mas[4]=5
```

Поскольку в вызванной функции формальный параметр, соответствующий массиву, является указателем, то его можно объявить как указатель в заголовке функции:

```
void func(int *a) {  /* a (формальный параметр) - указатель */
     a++;
     *(a+2) = 100;
}
```

При этом следует изменить прототип функции `func()` таким образом:

```
void func(int *a);
```

На самом деле обе следующие формы эквивалентны, когда употребляются в качестве формальных параметров:

1) `func(int a[])`
2) `func(int *a)`

Вторая форма предпочтительнее, поскольку она явно выражает тот факт, что параметр `a` является указателем. Однако если это удобно и не портит восприятия кода, возможно даже *комбинировать* обе эти формы.

Доступ к любому элементу массива

При вызове `func(mas)` из основной функции `main()` указатель `a` в функции `func()` ссылается на 1-й элемент массива. Но с точки зрения функции `func()` не важно, на какой элемент ссылается указатель `a`, поскольку всегда можно получить доступ к любому элементу массива `mas`.

Например, при вызове `func(mas+3)` (или то же самое `func(&mas[3])`) допускается:

```
void func(int a[]) {
     a[-2] = 10;
     *(a-1) = 20;
     *a = 30;
}
```

если такие обращения не выходят за границы массива. Результат:

```
mas[0]=1  mas[1]=10  mas[2]=20  mas[3]=30  mas[4]=5
```

Способы передачи массива в функцию

Передача размера массива

Для работы внутри функции со всем массивом целиком, ей должно быть известно количество элементов массива. Поэтому при передаче массива в

функцию ***всегда следует использовать два параметра***. Первый способ – передавать ***имя массива и количество элементов в нем***.

Например, составим функцию для вывода массива на экран:

```
#include <stdio.h>
void func(int a[], int k);
int main(void) {
     int mas[5]={1,2,3,4,5};
     print(mas,5); /* второй параметр - количество элементов */
     return 0;
}
void print(int a[], int k){ /* k - количество элементов массива */
     int i;
     for (i=0; i<k; ++i)
          printf("a[%d]=%d\n", i, a[i]);
}
```

Обратим внимание, что функцию `print()` можно вызывать для разных массивов с различным количеством элементов.

Передача указателей на начало и конец массива

Передача количества элементов массива в функцию – на единственный способ сообщить функции размер массива. Другой способ состоит в передаче функции двух параметров – ***указателей на начало и конец массива***.

Пример функции для вывода массива на экран:

```
#include <stdio.h>
#define SIZE 5   /* количество элементов массива */

void print(int *first, int *end);

int main(void) {
     int mas[SIZE]={1,2,3,4,5};
     print(mas, mas+SIZE); /* второй параметр - указатель на конец массива */
     return 0;
}
void print(int *first, int *end){ /* first - указатель на начало массива */
     int i=0;                     /* end - указатель на конец массива */
     while (first < end) {
          printf("mas[%d]=%d\n", i, *first);
          first++;
          i++;
     }
}
```

Здесь в функции `print()` для проверки окончания цикла

```
while (first < end)
```

используется указатель `end`, ссылающийся на ячейку памяти, которая находится сразу же за последним элементом массива. Язык С гарантирует, что при распределении памяти для массива данная ячейка всегда будет доступна.

Использование указателя, ссылающегося за пределы конца массива, позволяет осуществить вызов функции с такими фактическими параметрами

```
print(mas, mas+SIZE);
```

Поскольку индексация массива начинается с нуля, то параметр `mas+SIZE` указывает на элемент, следующий за концом массива.

Рассмотренную программу можно изменить так, чтобы не выходить за пределы массива

```
...
int main(void) {
     ...
     print(mas, mas+SIZE-1);
     ...
}
void print(int *first, int *end){
     ...
     while (first <= end) {
          ...
     }
}
```

но это делает вызов функции `print()` менее наглядным.

Защита содержимого массива

Передача массива в функцию осуществляется по адресу, при этом функция получает доступ к изменению элементов массива и всегда работает с исходными данными. Если функция в ходе своей работы не должна изменять элементы массива, а только иметь к ним доступ, то ей можно запретить изменять исходные данные и помощью ключевого слова `const` при объявлении формального параметра.

Функция `print()` из предыдущего примера только выводит массив на экран, но не изменяет его элементы, поэтому ее заголовок можно изменить:

```
#include <stdio.h>

void print(const int a[], int k);

int main(void) {
     int mas[5]={1,2,3,4,5};
     print(mas,5);
     return 0;
}
void print(const int a[], int k){
     int i;
     for (i=0; i<k; ++i)
          printf("a[%d]=%d\n", i, a[i]);
     a[2] = 10; /* Ошибка при компиляции */
     *(a+2) = 10; /* Ошибка при компиляции */
}
```

При этом изменение элементов массива стало недопустимо, и соответствующие сообщения об ошибках выдаются на этапе компиляции программы.

Использование `const`

Ключевое слово `const` может быть использовано для защиты других видов данных.

Например, для объявления переменной, значение которой изменять запрещено, то есть константы:

```
const double PI = 3.14159;
```

Можно защитить от изменения массив, т.е. все его элементы:

```
const int days[12] = {31,28,31,30,31,30,31,31,30,31,30,31};
```

Если после этого попытаться изменить элемент данного массива

```
days[1] = 29; /* Ошибка при компиляции */
```

то на этапе компиляции будет выдано сообщение об ошибке.

Указатели на константы не могут использоваться для изменения значений:

```
int array[4] = {10,20,30,40};
const int *p = array;
```

Указатель `p` ссылается на начало массива `array`, т.е. на элемент `array[0]`. Вторая строка означает, что значение типа `int`, на которое указывает `p` является константой `const`, поэтому:

```
*p = 100; /* Ошибка при компиляции */
*(p+1) = 200; /* Ошибка при компиляции */
p[2] = 300; /* Ошибка при компиляции */
array[2] = 300; /* Допустимо */
```

При этом изменение самого указателя (т.е. ячейки на которую он указывает) допустимо

```
p++; /* Допустимо */
```

теперь `p` ссылается на `array[1]`.

Локальные и внешние массивы

Локальный массив

Внутри функции можно объявлять *локальные* массивы, которые ведут себя так же, как и любые другие *автоматические* переменные: каждый такой массив создается при вызове функции и освобождает память при выходе из нее.

Внешний массив

Если в программе объявлен *внешний* массив (за пределами всех функций), то он будет существовать в памяти постоянно, подобно любой внешней переменной. Внешний массив, объявленный в самом начале программы, доступен во всей программе *глобально*.

Пример глобального внешнего массива:

```
#include <stdio.h>
int m[5]={1,2,3,4,5}; /* m – глобальный внешний массив */
void func(void);
int main(void) {
     func(); /* вызов функции без параметров */
     printf("m[3]=%d\n", m[3]); /* на экране: m[3]=40 */
     return 0;
}
void func(void){ /* функция без параметров */
     printf("m[3]=%d\n", m[3]); /* на экране: m[3]=4 */
     m[3] = 40;
}
```

Внешние массивы очень часто используются для обмена данными между функциями.

Одинаковые имена и область действия

Если имя глобального массива совпадает с именем локального массива для некоторой функции, то внутри этой функции глобальный массив не виден. Это же правило действует для переменных и других данных.

Иными словами: *в случае **совпадения имен** локального и глобального идентификаторов, **видимым** будет **только локальный** идентификатор, а **глобальный недоступен.***

То есть *при совпадении имен действие внешних идентификаторов отменяется*. Действуют только локальные идентификаторы с тем же именем, независимо от того, совпадают они по типу (и по размеру), или нет.

Пример глобального и локального массивов с одинаковыми именами (**C99**):

```
#include <stdio.h>
int m[3]={1,2,3}; /* объявление глобального внешнего массива m */
void func(char c[]);
int main(void) {
     printf("global      m[2]=%d\n", m[2]); /* m – глоб. int */
     char m[4]={'a','b','c','d'}; /* объявление локального m */
     printf("local(main) m[3]=%c\n", m[3]); /* m – лок. char */
     func(m); /* передача в функцию локального m */
     return 0;
}
void func(char c[]){
     printf("global      m[0]=%d\n", m[0]); /* m – глоб. int */
     double m[5]={0.1,0.2,0.3,0.4,0.5}; /* объявление локал. m */
     printf("local(func) m[4]=%.1f\n", m[4]); /* m – лок. double */
     printf("local(func) c[1]=%c\n", c[1]); /* c – лок. char */
}
```

Результат работы программы:

```
global      m[2]=3
local(main) m[3]=d
global      m[0]=1
local(func) m[4]=0.5
local(func) c[1]=b
```

Обратим внимание, что в функции `func()` объявлен локальный массив `m` из 5-и элементов типа `double`, однако из этой функции также доступен массив с тем же именем `m` из 4-х элементов типа `char`, являющийся локальным массивом функции `main()`. Это возможно благодаря тому, что последний передается в функцию `func()` как параметр и доступен через локальный указатель `c[]`.

В этом примере переменные объявляются после выполнения операторов. Такие объявления запрещены в ANSI C, а разрешены только в поздних стандартах языка C, начиная с **C99**.

Упражнения

Упражнение 9.1

Составить программу, которая создает целочисленный массив из 20-и элементов и заполняет его числами от 1 до 20 с помощью отдельной функции. Внешний массив использовать нельзя.

Упражнение 9.2

Составить программу, которая создает несколько массивов и выводит каждый из них на экран с помощью одной и той же функции. При этом имя каждого массив и количество элементов в нем передаются в функцию в качестве параметров.

Упражнение 9.3

Составить программу, которая содержит две функции для работы с одним внешним массивом из 10-и элементов. Первая функция обнуляет элемент, индекс которого задает пользователь; вторая – вычисляет сумму элементов и выводит массив на экран.