

Методические указания

Тематическое занятие 6 **Массивы и указатели.**

Содержание

Указатели и операции для работы с ними	1
<i>Переменная-указатель</i>	<i>1</i>
<i>Операция получения адреса (&).....</i>	<i>2</i>
<i>Операция раскрытия ссылки (*).....</i>	<i>2</i>
<i>Использование операций с указателями</i>	<i>2</i>
Массивы.....	3
<i>Определение.....</i>	<i>3</i>
<i>Объявление одномерного массива</i>	<i>3</i>
<i>Обращение к элементам массива через индексы</i>	<i>3</i>
<i>Инициализация массива</i>	<i>4</i>
<i>Ошибки при работе с индексами массива</i>	<i>4</i>
Работа с массивом с помощью указателей	4
<i>Связь массивов и указателей.....</i>	<i>4</i>
<i>Указатели на элементы массива</i>	<i>5</i>
<i>Адресная арифметика</i>	<i>5</i>
Случайные числа	6
<i>Функция rand()</i>	<i>6</i>
<i>Функция srand()</i>	<i>6</i>
Упражнения	6
<i>Упражнение 6.1.....</i>	<i>6</i>
<i>Упражнение 6.2.....</i>	<i>6</i>
<i>Упражнение 6.3.....</i>	<i>6</i>

Указатели и операции для работы с ними

Переменная-указатель

Указатель — это адрес объекта в памяти. Переменная типа «указатель» (*переменная-указатель*) – это переменная, в которой хранится адрес переменной определенного типа. Адрес соответствует некоторой конкретной ячейке памяти, и каждая ячейка доступной памяти имеет свой уникальный адрес.

Далее переменные-указатели будем также называть просто *указателями*. В языке C указатели являются мощнейшим средством программирования и широко используются для самых разных целей.

При объявлении переменной-указателя перед ее именем ставится знак *.
Общая форма объявления указателя:

тип *имя;

где тип – базовый тип указателя (например, один из стандартных типов), а имя – имя переменной-указателя. Примеры:

```
int *p; /* p – указатель на переменную типа int */  
float x, *y, z; /* x и z – обычные переменные, y – указатель */
```

Здесь переменные-указатели p и y имеют тип «указатель» и хранят адреса объектов базовых типов (int и float соответственно).

Операция получения адреса (&)

Операция получения адреса & – унарная операция, возвращающая адрес операнда в памяти. Например:

```
int a, *p;  
p = &a; /* в указатель p помещается адрес переменной a */
```

Здесь указатель p ссылается на переменную a, т.е. содержит адрес ячейки памяти, в которой размещена переменная a. Поэтому переменные-указатели часто называют **ссылками**.

Операция раскрытия ссылки (*)

Обратной к получению адреса является унарная операция раскрытия ссылки *, которая возвращает значение объекта, расположенного по указанному адресу. Другое название этой операции – **разыменование** адреса. Например:

```
int a=5, b, *p;  
p = &a; /* указателю p присваивается адрес переменной a */  
b = *p; /* переменная b получает значение, расположенное  
по адресу p */
```

В последней строке переменной b присваивается значение, на которое ссылается указатель p, т.е. значение переменной a. В результате в обеих переменных a и b будет содержаться значение 5.

Адрес и значение переменной – совершенно разные понятия. В приведенном примере переменные a и b имеют различные адреса в памяти, но принимают одинаковое значение 5. Теперь в этом же примере изменим значение переменной a с помощью указателя p:

```
*p = 7; /* переменная a, расположенная по адресу p,  
получает значение 7 */
```

Использование операций с указателями

Операции & и * имеют более высокий приоритет, чем любая арифметическая операция (кроме унарного минуса, который имеет тот же приоритет). Продолжая предыдущий пример:

```
b = *p + 3; /* переменная b получает значение 7+3=10 */  
*p += b; /* переменная a получает значение 7+10=17 */
```

Для операции инкремента:

```
p = &b; /* указателю p присваивается адрес переменной b */
++*p; /* переменная b получает значение 10+1=11 */
(*p)++; /* переменная b получает значение 11+1=12 */
```

В последней строчке скобки необходимы, поскольку обе используемые операции унарные и выполняются справа налево. В варианте без скобок `*p++` инкрементируется сам указатель, а не то, на что он указывает (см. адресную арифметику).

Указатели могут использоваться непосредственно, как переменные:

```
int *p, *q;
...
p = q; /* указатели p и q ссылаются а один и тот же объект */
```

Массивы

Определение

Массив – статическая структура данных, которая представляет собой однородную, фиксированную по размеру и конфигурации совокупность элементов, упорядоченных по номерам (*индексам*).

В языке C массивы располагаются в отдельной непрерывной области памяти. Первый элемент массива располагается по самому меньшему адресу этой области, а последний элемент – по самому большому.

Объявление одномерного массива

Общая форма объявления одномерного массива:

```
типЭлем имяМассива [количЭлем];
```

Здесь `типЭлем` – тип элементов массива (например, один из стандартных типов, а `количЭлем` – количество элементов массива.

Пример объявления массива из ста вещественных чисел:

```
double a[100];
```

Обращение к элементам массива через индексы

Доступ к элементу массива осуществляется с помощью индекса элемента, который указывается в квадратных скобках:

```
a[3] = 12.34;
b = a[4];
a[i] = 0.0;
printf("a[n]=%f", a[n]);
```

Индекс первого элемента любого массива в языке C равен нулю (0).

Поэтому в объявленном массиве `a` содержатся элементы от `a[0]` до `a[99]`.

Пример заполнения массива целыми числами от 1 до 20 и вывода на экран:

```
#include <stdio.h>
int main(void) {
    int m[20]; /* объявление массива из 20 элементов */
    int i;
    for (i=0; i<=19; i++)
        m[i]=i+1; /* заполнение массива числами */
}
```

```

for (i=0; i<=19; i++)
    printf("m[%2d]=%3d\n", i, m[i]); /* вывод на экран */
return 0;
}

```

Инициализация массива

При объявлении массива можно задать начальные значения всех его элементов (провести инициализацию массива). Пример:

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

Если при этом не указать размер массива, то в качестве значения размера будет взято количество инициализирующих значений. Например, массив

```
int arr[] = {4, 3, 2, 1};
```

будет состоять из 4-х элементов.

Ошибки при работе с индексами массива

В языке С **не проверяется содержимое** массива. В непрерывную область памяти, занятую массивом, может быть записано что угодно.

Также в языке С **не проверяется соблюдение границ** массива. Программист сам должен следить за тем, чтобы индексы элементов массива не выходили за свои пределы.

Следующий пример программы компилируется без ошибки, но во время выполнения происходит выход за границы массива и нарушение соседних участков памяти:

```

int mas[10];
for (i=0; i<100; i++) /* ОШИБКА выход за границы индексов */
    mas[i]=i;

```

Работа с массивом с помощью указателей

Связь массивов и указателей

Массивы и указатели в языке С тесно связаны. Значение переменной типа «массив» является адресом 1-го элемента массива (элемента с нулевым индексом). То есть имя массива – это синоним адреса его 1-го элемента. Поэтому, например, для объявлений

```

int *p; /* указатель на переменную типа int */
int a[10]; /* массив из 10 элементов */

```

после выполнения присваивания:

```
p = a; /* указателю p присваивается адрес 1-го элемента массива a */
```

в обеих переменных (p и a) хранится адрес 1-го элемента массива a[0]. Отличаются эти переменные тем, что значение a изменить нельзя, а p – можно.

Адрес 1-го элемента массива a[0] также можно получить, используя оператор &. Результат следующего присваивания будет тем же:

```
p = &a[0]; /* то же самое, что p=a; */
```

Указатели на элементы массива

Указатель, базовый тип которого совпадает с типом элементов массива, может указывать на любой элемент массива. Для приведенного примера:

```
int *q;  
q = &a[2];  
a[3] = *q; /* то же самое, что a[3]=a[2]; */
```

Любую операцию, выполняемую с помощью индексации массива, можно проделать с применением указателей. Причем код с применением указателей обычно **работает быстрее**, но несколько более сложен для восприятия.

Адресная арифметика

Для указателей разрешены арифметические операции сложения и вычитания указателя и целого числа. Если p указывает на некоторый элемент массива a , то выражение $p+1$ указывает на следующий элемент. Выражение $p+i$ указывает на i -й элемент после p , а выражение $p-i$ – на i -й элемент перед p .

В рассмотренном примере p указывает на элемент $a[0]$, тогда $p+i$ – это адрес элемента $a[i]$. Применяя операцию разыменования $*(p+i)$, получим содержимое ячейки памяти по данному адресу, т.е. значение элемента $a[i]$.

```
int i=4;  
printf("%d = %d", a[i], *(p+i)); /* одно и то же значение */
```

Более того, вычисляя выражение $a[i]$ компилятор языка C преобразует его в форму $*(a+i)$, поскольку обе эти формы тождественны. Применяя к ним операцию $\&$, получаем, что $\&a[i]$ и $a+i$ также тождественны.

Также для указателей разрешены операции инкремента (декремента) и сложения (вычитания) с присваиванием:

```
p = &a[5];  
p++; /* тождественно p=p+1; или p=&a[6]; */  
p-=2; /* тождественно p=p-2; или p=&a[4]; */
```

Кроме того, указатели на элементы одного и того же массива можно сравнивать. Например, здесь $p>q$ – истина, поскольку индекс элемента, на который указывает p , больше индекса элемента, на который указывает q .

Также возможно вычитание двух указателей одного и того же массива. В рассматриваемом примере выражение $p-q+1$ дает количество элементов от q до p включительно (в данном случае оно равно 3).

Рассмотренный выше пример заполнения массива целыми числами от 1 до 20 и вывода на экран с использованием указателей и адресной арифметики:

```
#include <stdio.h>  
int main(void) {  
    int m[20]; /* объявление массива из 20 элементов */  
    int i, *p; /* объявление указателя p */  
    p = m;  
    for (i=0; i<=19; i++)  
        *(p+i)=i+1; /* заполнение массива числами */  
    for (i=0; i<=19; i++)  
        printf("m[%2d]=%3d\n", i, *(p+i)); /* вывод на экран */  
    return 0;  
}
```

Случайные числа

Функция *rand()*

Стандартная функция `rand()`, которая генерирует последовательность псевдослучайных чисел, описана в заголовочном файле стандартной библиотеки `<stdlib.h>`. Функция `rand()` возвращает целое число в интервале от нуля до значения `RAND_MAX`, определенного в `<stdlib.h>`.

Например, генерировать случайные вещественные числа из интервала $[0; 1)$ позволяет следующее выражение

```
(double) rand() / (RAND_MAX + 1.0)
```

Функция *srand()*

Функция `srand()` (также описанная в заголовочном файле `<stdlib.h>`) устанавливает инициализирующее значение для функции `rand()`.

Следующая программа использует в качестве параметра функции `srand()` системное время:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(void) {
    int stime;
    long int ltime;
    ltime=time(NULL);
    stime=(unsigned) ltime/2;
    srand(stime);
    printf("%d\n", rand());
    return 0;
}
```

Упражнения

Упражнение 6.1

Составить программу, которая создает массив из 15 элементов, заполняет его случайными целыми числами из диапазона $[1, 12]$. Вывести массив на экран.

Упражнение 6.2

Составить программу, которая находит среднее арифметическое значение всех его элементов массива, описанного в упражнении 6.1.

Упражнение 6.3

Составить программу, которая находит максимальный элемент массива, описанного в упражнении 6.1. Если таких элементов несколько, то определить их количество.