Указатели и Массивы.

Шокуров Антон В.
shokurov.anton.v@yandex.ru
http://машинноезрение.рф

26 марта 2017 г.

Версия: 0.12

Аннотация

Указатели. Зачем это? Модификация переданных функции переменных. Индексация переменных. Статические и динамические массивы.

Предварительный вариант!

1 Адрес в памяти

Указатель на переменную дает возможность изменить её значение миную прямого обращения к самой переменной. Важны при вызове функций, когда хотим чтобы была возможность изменить "аргумент"функции.

1.1 Указатель

Сначала будет показано как создать указатель и как его проинициализировать значение, т.е. присвоить значение.

Объявление указателя Указатель модификатор типа данных, т.е. он является некой добавкой к основному типу. Его пишут до объявляемой переменной. Получается как бы что если звездочки нет, то это обычная переменная указанного типа, а если он есть, то переменная превращается в указатель на отмеченный тип.

```
1 int a;//Объявили переменную типа int.
2 //Указатель создается через модификатор *
3 //<тип> *<имя переменной>;//создает переменную значение
4 //которой указывает на переменную указанного типа.
```

```
5 int *b;//b - указатель на переменную типа int.
6 int *c, d;//c - указатель на int, a d - просто int.
```

Указатель можно объявить на любой тип данных, например, на указатель на int:

```
1 int **e;//e - указатель на указатель на int.
```

Указатель на переменную В переменную типа указатель можно присвоить соответствующую числовую характеристику. Для формирования такой числовой характеристики используется операция языка Си – амперсанд (&). Для переменной он возвращает указатель на ячейку памяти, который позволит обращаться к данной ячейке памяти минуя саму переменную.

```
1 int a;//переменная.
2 int *b;//указатель на int
3 b=&a;//Указатель на a присвоен переменной b.
4 b=&(a+1);//Ошибка, указатель не определен.
5 int c=5;
6 b=&(a+c);//Ошибка, указатель не определен.
```

Подчеркну, что указатель можно *вычислить* только для ячеек памяти, например, для переменных. Арифметические выражения не являются ячейками памяти, и поэтому для них указатель не существует (см. 4 и 6). Попытка взятия указателя приведет к ошибки компилирования.

Указатель должен быть совместимого типа.

```
1 double d;//переменная.
2 b=&d;//Нельзя. b - ожидает указатель на int.
3 double *f;
4 f=&d;//Правильно. Указатели совместимы.
5 f=&a;//Естественно нельзя. Указатели на разные типы.
```

Указатели на разные тип данных разные не совместимы. За этим нужно тщательно следить.

Указатель является специальной числовой характеристикой и его можно распечатать использую соответствующий режим печати (`%p`):

```
1 //Значение указателя можно распечатать:
2 printf("pointer ~ %p\n", b);
```

Упражнение. Посоздавай переменных различного типа данных. Напечатай их значение. Можно ли увидеть какую-то закономерность.

По аналогии можно вычислить указатель на указатель:

```
1 int *b, *g;//Указатель на переменную типа int.
2 int **e;//e - указатель на указатель на int.
3 //Корректное присвоение переменной e указателя на
4 e=&b;//переменную b.
5 g=&b;//Ошибка. Указатели не совместимы.
6 e=&f;//Ошибка. Базовый тип разный.
```

В стр. 5 показан ошибочный код. Так, в нем тип у указателей не совместим, а именно – у переменной g тип указатель на int, а у выражения `&b` – тип указатель на указатель на int. Подчеркну, что даже если оба выражения являются указателями на указатели, но базовый тип разный, то они все равно не совместимы (стр. 6).

Разыменование – доступ к переменной Указатель на переменную позволяет получить доступ к переменной, т.е. считать значение и записать новое. Данная операция языка Си называется разыменование. Она обозначается `*` и применяется к уже созданной переменной (в общем случае к адресу):

```
1 //Продолжая предыдущую программу.
2 //b -- указатель на a:
3 //a ~ мусор, следовательно *b ~ мусор.
4 a=5;//В b есть указатель на a, тогда
5 //*b ~ 5.
6 *b = 3;//*b превращается в переменную a, тогда последнее
7 //эквивалентно a = 3, т.е. в переменную a записывается 3.
8 //*b ~ 3.
9 //a ~ 3.
```

Зная указатель можно изменить значение искомой переменной. Даже указатель, теперь возможна следующий код:

```
1 int a;//Переменная типа int.
2 int *b;//Указатель на переменную типа int.
3 int **e;//e - указатель на указатель на int.
4 e=&b;//Присвоили указатель на переменную b.
5 //Можно так:
6 b=&a;
7 //a можно и так:
8 *e=&a;
```

В обоих случая в переменную `b` будет присвоен указатель на переменную `a`.

1.2 В функциях

В функциях находит сильное применение.

Изменение аргументов Применение указателей в качестве аргументов функций позволяет изменять значения аргументов:

```
1  #include <stdio.h>
2  #include <math.h>
3
4  //void, т.е. мы ничего не возвращаем.
5  void my_abs(double *a)
6  {
7    if( (*a) < 0 )
8      (*a) = -(*a);
9  }
10
11 int main()
12 {
13   double a;
14   scanf("%lf", &a);//Хдесь тажа логика у &.
15   //Функции передаем не само значение переменной,
16   my_abs( &a );//а указатель на неё.
17   printf("abs ~ %f\n", a);
18   return 0;
19 }
```

Упражнение. Известно, что наибольший делитель d чисел a и b можно представить в виде их линейной комбинации: $d = va + ub$, где u, $v \in \mathbb{Z}$. Напишите программу, которой передаются два числа a и b, и которая возвращает соответствующие значения (т.е. u и v) через аргумент функции. Значение самой функции будет равно наибольшему делителю.

Возвращаемое значение В случае, если измененное значение передается как аргумент, то возвращаемому значению придается определенный смысл. А именно согласно принятым правилам, отрицательное значение возвращается в случае ошибки и является её кодом (идентификатором). Неотрицательные значения обозначают успех, где само значение может трактоваться по разному. Например, в функции `scanf` оно равнялось количеству успешно считанных переменных.

```
1  #include <stdio.h>
2  #include <math.h>
```

```
3
4  int my_sqrt(double *a)
5  {
6    if( (*a) < 0 )
7      return -1;
8    a = sqrt(a);
9    return 0;
10 }
11
12 int main()
13 {
14   double a;
15   scanf("%lf", &a);//Хдесь тажа логика у &.
16   //Функции передаем не само значение переменной,
17   if( my_sqrt( &a ) < 0 )//а указатель на неё.
18   {
19     printf("Ошибка вычисления\n");
20     return -1;
21   }
22   printf("square root ~ %f\n", a);
23   return 0;
24 }
```

Возвращаемое значение может быть и указателем. В такой случае, если он не равен нулю (NULL), то ошибки нет. Иначе считаем, что произошла ошибка.

1.3 Массив – индексная переменная

Иногда хочется чтобы была возможность обращаться к переменной по её номеру. Допустим нам нужно посчитать количество четных и нечетных чисел:

```
1  #include <stdio.h>
2
3  int main()
4  {
5    int even_cnt = 0, odd_cnt = 0;
6    int a;
7    while( scanf("%d", &a) == 1 )
8    {
9      if( (a%2) == 0 )
10       even_cnt++;
```

```
11      else
12        odd_cnt++;
13    }
14    printf("e %d, o %d\n", even_cnt, odd_cnt);
15    return 0;
16  }
```

Слишком мудрено. В данной программе объявлены две переменные, к которым идет обращение в теле цикла.

Для упрощения программы можно воспользоваться возможностью индексации переменных, а именно – создать *статический* (т.е. имеющий фиксированный размер, в момент компиляции) массив. Статический массив тоже является модификатором типа, т.е. может быть применен к любому существующему типу. Использую его программу можно модифицировать так:

```
1  #include <stdio.h>
2
3  int main()
4  {
5  //Объявляется (создается) массив следующим образом:
6  //<тип>   <имя>[размер];
7  //Он состоит из переменных:
8  //<имя>[0], <имя>[1], ... <имя>[размер-1].
9    //Объявляем массив из двух переменных:
10   int cnt[2];
11   //доступны переменные cnt[0] и cnt[1].
12   cnt[0] = 0;//Обнуляем первую переменную
13   cnt[1] = 0;//и вторую.
14   int a;
15   while( scanf("%d", &a) == 1 )
16     cnt[a%2]++;
17   printf("e %d, o %d\n", cnt[0], cnt[1]);
18   return 0;
19 }
```

Все стало намного лаконичнее. Так, массив позволил тело цикла сократить до одной строчки (16).

> Однажды ты спросишь меня, что для меня на первом месте: ты или программирование?
> И я отвечу тебе, что программирование.
> И ты уйдешь, так и не узнав, что ты для меня на нулевом месте.

Крайне важно, что нумерация переменных (элементов) в массиве начинается с 0, а не с 1.

Упражнение. Сколько целых чисел на отрезке $[a, b]$, где $a, b \in Z$?

Таким образом, массив начинается с переменной имеющей нулевой индекс, а завершается переменной с индексом на единицу меньше чем размер массива. В таком случае их общее количество как раз будет равно n.

Статический массив можно проинициализировать:

```
1 double k[3]={-5.5, 10.1, 4.5};
2 //В переменных k[0], k[1], k[2] соответственно хранятся
3 //числа -5.5, 10.1 и 4.5.
```

Более того, в последнем случае размер массива можно и не указывать:

```
1 double q[]={-5.5, 10.1, 4.5, -2.1};//q размера 4.
2 //q[0] ~ -5.5, q[1] ~ 10.1,q[2] ~ 4.5, q[3] ~ -2.1.
```

В последнем случае компилятор автоматически создал массив нужно размера. В случае, если размер указан, но он больше списка инициализации, то только первые элементы будут проинициализированы.

Операции на массивом К элементам массива можно получить доступ не только через индексацию. Так, имя массива само по себе является переменной и как к переменной к ней можно применять арифметическую операция сложения и разыменование:

```
1  int cnt[10];
2  //переменная cnt сама по себе является указателем на
3  //первый элемент (cnt[0]) массива cnt[10]:
4  *cnt=10;//Присвоили значение первому элементу.
5  //cnt[0] ~ 10.
6  *(cnt+2)=5;//cnt+2 указывает на переменную с индексом 2.
7  //cnt[2] ~ 5.
8  //В этом смысле, cnt+0 ~ cnt, поэтому:
9  *(cnt+0)=7;//cnt+0 указывает на переменную с индексом 0,
10 //т.е. на первый элемент.
```

```
11 //cnt[0] ~ 7.
12 *(cnt+10) = 3;//Ошибка! Вышли за конец массива.
```

Переменных за последним элементом не существуют. Выход за последний элемент массива запрещен. При запуске приведет к ошибке программы и принудительному завершению.

Двумерные массивы будут рассказаны в отдельной заметке.

1.4 Динамический массив

Массивы можно создавать динамически в процессе работы программы:

```
1  #include <stdio.h>
2  #include <stdlib.h>//Необходима для malloc и free.
3
4  int main()
5  {
6    int n;
7    scanf("%d", &n);
8    double *a;//Объявили указатель для нужного типа.
9    //Выделяем память под наш массив:
10   a = (double *)malloc( n * sizeof(double) );
11   //Элементами массива являются n переменных:
12   //a[0], ..., a[n-1].
13   int i;
14   for( i=0; i < n; i++)//Записываем значения в
15     scanf("%d", &a[i]);//элементы массива.
16   for( i=0; i < n/2; i++)//Отразим массив.
17   {
18     //Сохраняем значение элемента массива:
19     double tmp = a[i];
20     //Копируем значение с конца в начало:
21     a[i] = a[n - 1 - i];
22     //Присваиваем сохраненное значение:
23     a[n - 1 - i] = tmp;
24   }
25   //<n, так как последний индекс имеет номер n-1.
26   for( i=0; i < n; i++)
27     printf("%d ", a[i]);
28   free( a );//Освобождаем ранее выделенную память.
29   return 0;
```

```
30 }
```

Для этого доступна вспомогательная (см. стр. 2) функция `malloc`, которая *выделяет* массив запрашиваемого (например, `n * sizeof(double)`) размера и возвращает указатель на его начало (см. стр. 10). Память является ресурсом системы, поэтому, при завершении работы с массивом соответствующую ранее выделенную память необходимо освободить (см. стр. 28) посредством функции `free`.

Упражнение. Как сделать так, чтобы числа печатались через запятую, а последним была точка?

Операции Над переменной указатель также доступны арифметические операции сложение и вычитания (умножения и деления нет!).