Курс «Алгоритмы и алгоритмические языки»

Лекция 8

Указатели

- **&** - операция адресации
  ***** - операция разыменования

Пример: фрагмент программы.

```
int a = 1;
int *p;
p = &a;
*p = 2;
printf ("Значение переменной a = %d\n", *p);
printf ("Адрес переменной a = %p\n", p);
```

В результате выполнения фрагмента будет напечатано:

```
Значение переменной a = 2
Адрес переменной a = 0xbffff7a4
```

Напечатанный адрес – 16-чное число:

$$b*16^7 + f*16^6 + f*16^5 + f*16^4 + f*16^3 + 7*16^2 + a*16 + 4 = 11*16^7 + 15*16^6 + 15*16^5 + 15*16^4 + 15*16^3 + 7*16^2 + 10*16 + 4 = 3221223332$$

Адресная арифметика

- В языке Си допустимы следующие операции над указателями:
  - сложение указателя с целым числом
  - вычитание целого числа из указателя
  - вычитание указателей
  - операции отношения и сравнения
- Пример. Пусть **`sizeof (int) == 4`**
  и пусть текущее значение **`int* p1`** равно **`2012`**.

  После операции **`p1++`** значение **`p1`** будет **`2016`** (а не **`2013`**), после операции **`p1 - 3`** – значение **`2000`**.
  - при увеличении (уменьшении) на целое число **`i`** указатель будет перемещаться на **`i`** ячеек соответствующего типа в сторону увеличения (уменьшения) их адресов.

Преобразование типа указателя

- Указатель можно преобразовать к другому типу, но такое преобразование типов обязательно должно быть явным.

  Иногда такое преобразование типов может вызвать непредсказуемое поведение программы.

- Пример.

```
#include <stdio.h>
int main (void)
{
  double x = 200.35, y;
  int *p;
  p = (int *)&x;  /* &x ссылается на double,
                     а p имеет тип *int */
  y = *p;       /* будет ли y присвоено
                   значение 200.35? */
  printf ("значение x равно %f\n", x);
  printf ("значение y равно %f\n", y);
  return 0;
}
```

Преобразование типа указателя

- Типичный вывод (GCC, Linux):

```
значение x равно 200.350000
значение y равно 858993459.000000
```

Выводимое число зависит от содержания области памяти, адресуемой **&y** длиной в 8 байт (присваивание **y = *p;** меняет только первые четыре байта области памяти с адресом **&y**)

- Таким образом, необходимо учитывать, что операции с указателями выполняются в соответствии с базовым типом указателей.

Преобразование типа указателя

- Разрешено также преобразование целого в указатель и наоборот. Однако пользоваться этим нужно очень осторожно, так как при этом легко получить непредсказуемое поведение программы (в частности, разыменование **`NULL`**).
- Допускается присваивание указателя типа **`void *`** указателю любого другого типа (и наоборот) без явного преобразования типа указателя. Это позволяет использовать указатель типа **`void *`**, когда тип объекта неизвестен.
- Использование типа **`void *`** в качестве параметра функции позволяет передавать в функцию указатель на объект любого типа.

Указатели и массивы

- Указатель на первый элемент массива можно создать, присвоив переменной типа “указатель на тип элемента массива” имя массива без индекса:

```
int array[15];
int *p, *q;
p = array;
q = &array[0];
```

- **p** и **q** указывают на начало массива **array[15]**
- Значение **array** изменить нельзя, а значение **p** – можно.

array не является *l*-значением, а **p** – является

array = p; array++ – писать нельзя (это ошибки)

p = array; p++ – писать можно

Указатели и массивы

- Индексирование указателей

```
int *p, a[10]; /* два способа присвоить 100 */
               /* 6-ому элементу массива a[10] */
p = a;
*(p + 5) = 100; /* адресная арифметика */
p[5] = 100;  /* индексирование указателя */
```

- Сравнение указателей

Если **p** и **q** являются указателями на элементы одного и того же массива и **p < q**, то **q – p + 1** равно количеству элементов массива от **p** до **q** включительно.

Можно написать:

```
if (p < q)
  printf ("p ссылается на меньший адрес, чем q");
```

Массивы указателей

- Указатели могут быть собраны в массив:

```
int *mu[27]; /* это массив из 27 указателей на int */
int (*um)[27]; /* это указатель на массив из 27 int */
```

- Пример

```
static void error (int errno)
{
  static char *errmsg[] = {
    "переменная уже существует",
    "нет такой переменной",
    <…>
    "нужно использовать переменную-указатель"
  };
  printf ("Ошибка: %s\n", errmsg[errno]);
}
```

- Имя массива указателей – пример многоуровневого указателя. Массив **errmsg** можно представить как **char **errmsg**

Функции

- ***Объявление функции***:

  тип_возвр_значения имя_функции(*тип параметр*, *тип параметр*, …, *тип параметр*);

  ```
  int atoi (char s[]);
  double atof (char s[]);
  void QuickSort (char *items, int count);
  ```

- Тип возвращаемого значения **`void`** означает, что функция не возвращает значения.
- ***Определение функции***:

  объявление_функции **{***тело_функции***}**

- ***Областью действия*** функции является *весь* программный файл, в котором она объявлена, начиная со строки, содержащей ее объявление.
- Определение функции должно присутствовать только в одном из программных файлов, в которых она объявлена, либо в одной из библиотек.

Вызов функции

- Если функция **`f()`** возвращает значение типа *тип*, то вызов этой функции может иметь вид: **`v = f();`**, где **`v`** – переменная типа *тип*.
- Если функция **`f(`***параметр***`)`** не возвращает значений, вызов этой функции имеет вид: **`f(`***аргумент***`);`**
- Если в программном файле вызывается какая-либо функция, она *обязательно должна быть объявлена в этом программном файле до ее вызова.*
- Директива препроцессора **`#include <`***имя_библиотеки***`.h>`** вставляет в программу объявления всех функций соответствующей библиотеки
- **В языке Си все аргументы передаются по значению** (т.е. передаются только значения аргументов, и эти значения копируются в память функции).
- Если аргументом является указатель, его значением может быть адрес объекта вызывающей функции, что обеспечивает вызываемой функции доступ к объекту.
- Массив всегда передается с помощью указателя на его первый элемент.

Функции

- Пример:

```
#include <ctype.h>
int atoi (char *s)
{
  int n, sign;
  for (; isspace (*s); s++)
    ;
  sign = (*s == '-') ? -1 : 1;
  if (*s == '+' || *s == '-')
    s++;
  for (n = 0; isdigit (*s); s++)
    n = 10 * (*s - '0');
  return sign * n;
}
```

- Стандартная библиотека **`ctype`** содержит такие функции, как **`isspace()`** , **`isdigit()`** и др.

Указатели и аргументы функций

- Используя аргументы-указатели, функция может обращаться к объектам вызвавшей ее функции.
- Использование указателей позволяет не дублировать массивы, передавая их функции:
  функции достаточно передать указатель на первый элемент массива.
- **Пример**. Функция **`void swap(int x, int y)`**
  меняет местами значения переменных **`x`** и **`y`**.

Неправильный вариант:

```
void swap (int x, int y)
{
  int tmp;
    tmp = x;
    x = y;
    y = tmp;
}
```

Правильный вариант:

```
void swap (int *px, int *py)
{
  int tmp;
   tmp = *px;
   *px = *py;
   *py = tmp;
}
```

Возврат из функции

- Возврат из функции в точку вызвавшей ее функции, следующей за точкой вызова функции, осуществляется:
  - либо при выполнении оператора **`return`**,
  - либо после выполнения последнего оператора функции, если она не содержит оператора **`return`**.

```
#include <string.h>
#include <stdio.h>
void print_str_reverse (char *s)
{
  register int i;
  for (i = strlen (s) – 1; i >= 0; i--)
    putchar (s[i]);
}
```

- Если тип функции не **`void`**, то в ее теле должен быть хотя бы один оператор **`return`** с возвращаемым значением
- Если у функции несколько операторов **`return`**, возврат осуществляется **немедленно** по тому из них, который будет выполнен первым.

Результат выполнения функции

- Все функции, кроме тех, которые относятся к типу **`void`**, возвращают значение, которое определяется выражением в операторе **`return`**.
- Помимо вычисления возвращаемого значения, функция может изменять значения переменных вызывающей функции (по указателю), а также изменять значения глобальных переменных.
- Результаты вызова функции, не связанные непосредственно с вычислением возвращаемых значений, составляют *побочный эффект* функции.
- В Си-программе можно использовать функции трех видов:
  (1) Функции, которые выполняют операции над своими аргументами с единственной целью – вычислить возвращаемое значение.
  (2) Функции, которые обрабатывают данные и возвращают значение, которое показывает, успешно ли была выполнена эта обработка.
  (3) Функции, не возвращающие значений. Все такие функции имеют тип **`void`**.
- Значения, возвращаемые функциями первого и второго вида, не обязательно должны быть использованы в программе

Результат выполнения функции

- Возвращаемым значением может быть указатель. Требуется, чтобы в объявлении такой функции тип возвращаемого указателя был объявлен точно: нельзя объявлять возвращаемый тип как **`int *`**, если функция возвращает указатель типа **`char *`**.
- Пример функции, возвращающей указатель (поиск первого вхождения символа **`c`** в строку **`s`**):

```
char *match (char c, char *s)
{
  while (c != *s && *s)
    s++;
  return s;
}
```