Курс «Алгоритмы и алгоритмические языки»

Лекция 7

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 является 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 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 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 программы, 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`**.

Результат выполнения функции

- Пример: перемножение матриц

```
void matr_prod (double *a, double *b, double *c,
                int m, int s, int n)
{
  int i, j, k;

  /* a: m x s, b: s x n, c: m x n */
  for (i = 0; i < m; i++)
    for (j = 0; j < n; j++) {
      c[i, j] = 0;
      for (k = 0; k < s; k++)
        c[i][j] += a[i][k]*b[k][j];
    }
}
```

- 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;
}
```

Рекурсия

- В языке Си функция может быть *рекурсивной*, т.е. вызывать саму себя:

```
int Fibrec (int n)
{
  if (n == 1 || n == 2)
    return 1;
  else
    return Fibrec (n – 2) + Fibrec (n – 1);
}
```

- Рекурсивные функции часто неэффективны по сравнению с их нерекурсивными вариантами:

```
int Fbn(int n)
{
  int k, g, h, fb;
  if (n == 1 || n == 2)
   return 1;
  else
   for (k = 2, g = h = 1; k < n; k++) {
     fb = g + h;
     h = g;
     g = fb;
   }
 return fb;
}
```

Ключевое слово `inline`: подставляемые функции (C99)

```
#include <stdio.h>
inline static int max (int a, int b)
{
  return a > b ? a : b;
}
int main (void)
{
  int x = 5, y = 17;
  printf ("Наибольшим из чисел %d и %d является %d\n", x,
           y, max(x, y));
  return 0;
}
```

◈ При обычной реализации **`inline`** приведенная программа эквивалентна:

```
#include <stdio.h>
inline static int max (int a, int b)
{
  return a > b ? a : b;
}
int main (void)
{
  int x = 5, y = 17;
  printf ("Наибольшим из чисел %d и %d является %d\n", x,
           y, (x > y ? x : y));
  return 0;
}
```

Операция `sizeof`

- Одноместная операция **`sizeof`** позволяет определить длину операнда в байтах.

  Операнды – типы, либо переменные.

  Результат имеет тип **`size_t`**

- Операция **`sizeof`** выполняется во время компиляции, ее результат представляет собой константу.
- Операция **`sizeof`** помогает улучшить переносимость программ.
- Операция **`sizeof`** позволяет определить, например, объем памяти в байтах, занимаемый двумерным массивом:

  `number_of_bytes = d1 × d2 × sizeof (element_type)`

  где **`d1`** – количество элементов по первому измерению,

  `d2` – количество элементов по второму измерению,
  `element_type` – тип элемента массива.

- Можно поступить и проще:

  `number_of_bytes = sizeof (имя_массива)`

***Операция* sizeof**

- Пример:

```
#include <stdio.h>
#include <string.h>
int main (int argc, char **argv)
{
      char buffer[10]; /* копирование 9 символов из
            argv[1] в buffer; sizeof(char)равно 1
            число элементов массива buffer равно его
            размеру в  байтах
      strncpy (buffer, argv[1], sizeof(buffer) -
               sizeof(char));
      buffer[sizeof(buffer) - 1] = '\0';
      return 0;
}
```

Операция `sizeof`

- **`sizeof`** можно применять только к «полностью» определенным типам.

  Для массивов это означает:

  - размерности массива должны присутствовать в его объявлении
  - тип элементов массива должен быть полностью определен.

- Пример. Если объявление массива имеет вид:

  ```
  extern int arr[];
  ```

  то операция **`sizeof (arr)`** ошибочна,
  так как у компилятора нет возможности узнать, сколько элементов содержит массив **`arr`**.