Курс «Алгоритмы и алгоритмические языки»
1 семестр 2016/2017

Лекция 9

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.

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 на каждом пути выполнения должен быть 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.
- Выделяют следующие виды функций:
  (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, возвращающие несколько значений (через указатели-аргументы и через возвращаемое значение).
  (4) Функции, не возвращающие значений. Все такие функции имеют тип **`void`**.

Результат выполнения функции

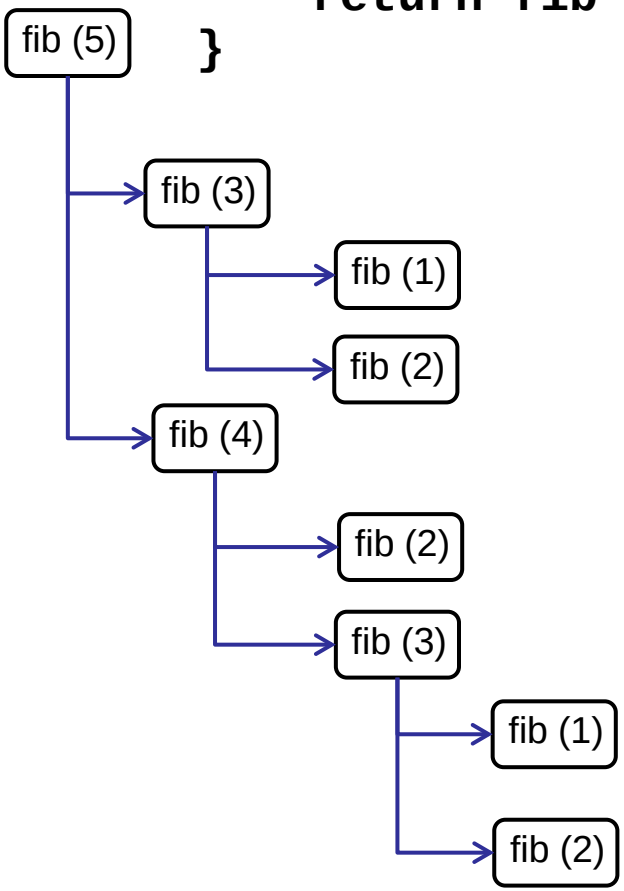
- Возвращаемым значением может быть указатель. Требуется, чтобы в объявлении такой функции тип возвращаемого указателя был объявлен точно: нельзя объявлять возвращаемый тип как **`int *`**, если функция возвращает указатель типа **`char *`**.
- Пример функции, возвращающей указатель (поиск первого вхождения 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 fib (int n)
{
  if (n == 1 || n == 2)
    return 1;
  else
    return fib (n – 2) + fib (n – 1);
}
```



Рекурсия

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

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

- Функция **`fib`** работает за экспоненциальное время и линейную память,
функция **`fibn`** – за линейное время и константную память.

*Хвостовая рекурсия**

- *Хвостовая рекурсия* (tail recursion) – рекурсивный вызов в самом конце функции. Как правило, этот вызов может быть оптимизирован компилятором в цикл.

```
int fact (int n) {
  if (n == 0)
    return 1;
  else
    return n*fact (n-1);
}
```

```
int fact (int n) {
  return tfact (n, 1);
}
int tfact (int n, int acc) {
  if (n == 0)
    return acc;
  return tfact (n-1, n*acc);
}
```

```
int fact (int n) {
  int t_n = n, t_acc = 1;

/* tfact встроена в fact и оптимизирована в цикл */
start:
  if (t_n == 0)
    return t_acc;
  t_acc = t_n * t_acc;
  t_n = t_n – 1;
  goto start;
}
```

Ключевое слово `inline`: встраиваемые 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;
}
```

Указатели на функцию

- Каждая функция располагается в памяти по определенному адресу. Адресом функции является ее точка входа (при вызове функции управление передается именно на эту точку).
- Присвоив значение адреса функции переменной типа указатель, получим указатель на функцию.
- Указатель функции можно использовать вместо ее имени при вызове этой функции. Указатель «лучше» имени тем, что его можно передавать другим функциям в качестве их аргумента.
- Имя функции **`f()`** без скобок и аргументов (**`f`**) по определению является указателем на функцию **`f()`** (аналогия с массивом).

```
int (*pf) (const char*, const char*);
char *s1, *s2;
int x = (*pf) (s1, s2);
int y = pf (s2, "string constant");
```

Указатели на функцию

- ***Пример***. Сравнение двух строк символов, введенных пользователем (функция **check()**).

```
#include <stdio.h>
#include <string.h>

static void check (char *a, char *b,
                   int (*pf) (const char*, const char*)) {
  printf ("Проверка на совпадение: ");
  if (! pf (a, b))
    printf ("равны\n");
  else
    printf ("не равны\n");
}

int main (void) {
  char s1[80], s2[80];

  printf ("Введите две строки \n");
  fgets (s1, sizeof (s1), stdin); s1[strlen (s1) - 1] = 0;
  fgets (s2, sizeof (s2), stdin); s2[strlen (s2) - 1] = 0;
  check (s1, s2, strcmp);
  return 0;
}
```

- Объявление **`int (*p)(const char *, const char *);`** сообщает компилятору, что **`p`** – указатель на функцию, имеющую два параметра типа **`const char *`** и возвращающую значение типа **`int`**.
- Скобки вокруг **`*p`** нужны, так как операция **`*`** имеет более низкий приоритет, чем **`()`**: если написать **`int *p(...)`**, получится, что объявлен не указатель на функцию, а функция **`p`**, которая возвращает указатель на целое.
- **`(*cmp)(a, b)`** эквивалентно **`cmp(a, b)`.**
- У функции **`check`** три параметра: два указателя на тип **`char`** и указатель на функцию **`pf`**. Указатель **`pf`** и функция **`strcmp`** имеют одинаковый формат, что позволяет использовать имя функции в качестве аргумента, соответствующего параметру **`pf`**.
- В данном случае использование указателя на функцию позволяет не менять программу сравнения, и тем самым получается более общий алгоритм.

```
int compvalues (const char *a, const char *b) {
  return atoi (a) != atoi (b);
}
```

- Массивы указателей на функцию: гибкая обработка событий

Поразрядные операции

- **&** (поразрядное И)
- **|** (поразрядное включающее ИЛИ)
- **^** (поразрядное исключающее ИЛИ)
- **<<** (сдвиг влево)
- **>>** (сдвиг вправо)
  - Беззнаковое число – заполнение нулями
  - Знаковое число – заполнение значением знакового разряда («арифметический сдвиг») или нулями («логический сдвиг»)
- **~** (дополнение до 1, или *инверсия*)

Тип «степень множества» (булеан)

- Пусть U – множество. Множество всех подмножеств множества U называется *степенью множества* U.
- Пусть B – степень конечного множества U. Тогда $|B| = 2^{|U|}$.
- *Характеристической функцией* χ_C подмножества C множества U называется функция, принимающая значение 1 на элементах U, входящих в состав C, и значение 0 на остальных элементах U.
- Множества удобно задавать через их характеристические функции. При этом в зависимости от количества элементов базового множества U его характеристическая функция, кодированная битами целого типа, может иметь тип

  `unsigned char` (если $|U| \le 8$)

  `unsigned int` (если $|U| \le 32$)

  `unsigned long long` (если $|U| \le 64$)

Тип «степень множества»

- **Пример**. Пусть $U = \{r, o, y, g, c, b, v, w\}$. Тогда его подмножества задаются переменными типа **`unsigned char`**: $|B| = 2^{|U|}$

  $\{\}$ задается значением **`00000000`**,

  $\{r, y, g\}$ – значением **`10110000`**

  $\{r, o, y, g, c, b, v, w\}$– значением **`11111111`**

буквы r, o, y, g, c, b, v, w являются первыми буквами семи цветов спектра и белого цвета:

$r - red$	$c - cyan$
$o - orange$	$b - blue$
$y - yellow$	$v - violet$
$g - green$	$w - white$

Реализация операций над множествами с помощью поразрядных операций

- **&** (поразрядное И) соответствует пересечению множеств:

  $B = \{r, y, g, b, w\}$, $C = \{r, o, y, g, v\}$

  χ_B **= 10110101,** χ_C **= 11110010**

  χ_B **&** χ_C **= 10110101 & 11110010 = 10110000**

  $B \cap C = \{r, y, g\}$

Реализация операций над множествами с помощью поразрядных операций

- **|** (поразрядное включающее ИЛИ) соответствует объединению множеств:

$B = \{r, y, g, b, w\}$, $C = \{r, o, y, g, v\}$

χ_B **= 10110101,** χ_C **= 11110010**

χ_B **|** χ_C **= 10110101 | 11110010 = 11110111**

$B \cup C = \{r, o, y, g, b, v, w\}$

Реализация операций над множествами с помощью поразрядных операций

- ~ (инверсия) соответствует дополнению до множества U:

  $B = \{r, y, g, b, w\}$

  χ_B **= 10110101**

  $\chi_{\sim B}$ **= 01001010**

  $\sim B = \{o, c, v\}$