Курс «Алгоритмы и алгоритмические языки»
1 семестр 2018/2019

Лекция 12

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*)

hackersdelight.org

```
x & 1      x | 1      x | (1 << 5)    x & (x – 1)
x ^= y, y ^= x, x ^= y    ~x + 1      x | (x + 1)
```

Структуры

- Структура – это совокупность нескольких переменных, часто разных типов, сгруппированных под одним именем для удобства
  - Переменные, перечисленные в объявлении структуры, называются ее *полями*, *элементами*, или *членами*.
- ***Объявление структуры***:

  struct *метка структуры* **{** *поля структуры* **};**

```
struct point
{
  int x;
  int y;
} f, g;
struct point h, center = {32, 32};
```

Структуры

- Поля структуры могут иметь любой тип, например, тип массива или тип другой структуры

```
struct rect
{
  struct point pt1;
  struct point pt2;
};
```

- ***Инициализация структуры***:

```
struct rect r = {.pt1 = {4, 4},
                 .pt2 = {7, 6}};
/* Остальные элементы – нулевые */
struct rect r2 = {.pt2.x = 5};
```

- Размер структуры в общем случае **не равен** сумме размеров ее элементов (**выравнивание**)

Структуры

- Доступ к полям структуры: операция точка "**.**"
  - **f.x, g.y, r.pt1.x**
- Присваивание структур целиком: **f = g;**
- Массивы структур

```
#define NRECT 15
/* Первый прямоугольник вокруг 0, 0 */
struct rect rectangles[NRECT]
= {{-1, -1, 1, 1}};
/* Последний прямоугольник – большой */
#define BOUND 1024
struct rect bounded_rectangles[NRECT]
= {[NRECT-1] = {-BOUND, -BOUND,
                 BOUND,  BOUND}};
```

Указатели на структуры

- ```
  struct rect r = {.pt1 = {4, 4},
                   .pt2 = {7, 6}};
  struct rect *pr = &r;
  ```
- Доступ к полям структуры через указатель:

  ```
  pr->pt1 (= (*pr).pt1), pr->pt2.x
  ```
- Адресная арифметика:

  ```
  struct rect *pr = &bounded_rectangles[0];
  while (pr->pt1.x != -BOUND)
      pr++;
  ```
  ```

Составные инициализаторы структур (C99)

- ```
  struct rect r;
  r = (struct rect) { {4, 4},
                      {7, 6} };
  ```
- Составной инициализатор генерирует **lvalue**!

  Т.е. можно передавать и указатель:

```
double area (struct rect *r) {
  return (r->pt1.x – r->pt2.x)
         * (r->pt1.y – r->pt2.y);
}
double da
= area (& (struct rect) {{4, 4}, {7, 6}});
```
```