Курс «Алгоритмы и алгоритмические языки»
1 семестр 2017/2018

Лекция 10

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*)

Дома. Выпишите код для выделения отдельного бита, составления результата из двух «половинок» операндов (младших и старших бит), для подсчета единичных бит в числе

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**

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

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 NREСT 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}});
```
```

Приоритеты операций

Операции	Ассоциативность
`( ) [ ] -> .`	Слева направо
`! ~ ++ -- + - sizeof (type) * &`	Справа налево
`* / %`	Слева направо
`+ -`	Слева направо
`<< >>`	Слева направо
`< <= > >=`	Слева направо
`== !=`	Слева направо
`&`	Слева направо
`^`	Слева направо
`|`	Слева направо
`&&`	Слева направо
`||`	Слева направо
`?:`	Справа налево
`= += -= *= /= %= &= ^= |= <<= >>=`	Справа налево
`,`	Слева направо

Объединения

- Объединение – это объект, который может содержать значения различных типов (но не одновременно – только одно в каждый момент)

```
struct constant
{
  int ctype;
  union
  {
    int i;
    float f;
    char *s;
  } u;
} sc;
```

```
switch (sc.ctype)
{
  case CI:
    printf("%d",sc.u.i);
    break;
  case CF:
    printf("%f",sc.u.f);
    break;
  case CS: puts(sc.u.s);
}
```

- Размер объединения достаточно велик, чтобы содержать максимальный по размеру элемент
- Можно выполнять те же операции, что и со структурами

Анонимные объединения и структуры (C11)

- Для вложенных структур и объединений разрешено опускать тег для повышения читаемости

```
struct constant
{
  int ctype;
  union
  {
    int i;
    float f;
    char *s;
  } /* нет имени! */;
} sc;
```

```
switch (sc.ctype)
{
  case CI:
    printf("%d",sc.i);
    break;
  case CF:
    printf("%f",sc.f);
    break;
  case CS: puts(sc.s);
}
```

- Поля анонимной структуры считаются принадлежащими родительской структуре (если родительская также анонимна – то следующей родительской структуре и т.п.)

Битовые поля

- Для экономии памяти можно точно задать размер поля в битах (например, набор флагов)

```
struct tree_base {
  unsigned code : 16;
  unsigned side_effects_flag : 1;
  unsigned constant_flag : 1;
  <...>
  unsigned lang_flag_0 : 1;
  unsigned lang_flag_1 : 1;
  <...>
  unsigned spare : 12;
}
```

  - Адрес битового поля брать запрещено
  - Можно объявить анонимные поля (для выравнивания)
  - Можно объявить битовое поле ширины 0 (для перехода на следующий байт)

Перечисления

- Перечисления – целочисленные типы данных, определяемые программистом.
- Определение перечисления:

```
enum имя_типа { имена значений };
enum colors {red, orange, yellow, green,
azure, blue, violet};
```

- Значения перечисления нумеруются с 0, но можно присваивать свои значения

```
enum {red, orange = 23, yellow = 23,
green, cyan = 75, blue = 75, violet};
```

- Доступны операции над целочисленными типами и объявление указателей на переменные перечислимых типов
- Проверка корректности присваиваемых значений не производится