Московский государственный университет им. М. В. Ломоносова
Факультет вычислительной математики и кибернетики

Алгоритмы и алгоритмические языки
Лекция 12

16 октября 2019 г.

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 её *полями*, *элементами*, или *членами*.

Объявление 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 её 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}});
```

Старшинство операций

Старшинство	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 typename { name[=value], ... };
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.