Курс «Алгоритмы и алгоритмические языки»
1 семестр 2015/2016

Лекция 12

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}});
```
```

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

Программа: вычисление площадей фигур

- Фигура определяется общей структурой, в которой объединены различные конкретные структуры для определенных фигур и поле тега фигуры
- Используется перечисление для задания возможных типов фигур
- Используется адресная арифметика для обхода массива считанных фигур
- Для обработки фигуры используется оператор выбора по тегу фигуры

Программа: типы данных

```
enum figure_tag {
  CIRCLE,
  RECTANGLE,
  TRIANGLE,
  LAST_FIGURE
};
struct point {
  double x, y;
};
struct fcircle {
  struct point center;
  double radius;
};
struct frectangle {
  struct point ll, ur;
};
```

Программа: типы данных

```
struct ftriangle {
  struct point a, b, c;
};
struct figure {
  enum figure_tag tag;
  union {
    struct fcircle fc;
    struct frectangle fr;
    struct ftriangle ft;
  } u;
};
struct farea {
  double min, max;
  int initialized : 1;
};
```

Программа: основная функция

```
int main (void) {
  enum { MAXFIG = 128 };
  struct figure figs[MAXFIG + 1];
  struct farea areas[LAST_FIGURE];
  int i = 0;

  while (i < MAXFIG)
    if (!read_figure (&figs[i]))
      break;
    else
      i++;

  if (i == 0)
    return 1;

  <... Обработка данных и вывод ...>
}
```

Программа: основная функция

```
int main (void) {
  enum { MAXFIG = 128 };
  struct figure figs[MAXFIG + 1];
  struct farea areas[LAST_FIGURE];
  int i = 0;

  <... Считывание данных ...>

  figs[i].tag = LAST_FIGURE;
  for (i = CIRCLE; i < LAST_FIGURE; i++) {
    areas[i].initialized = 0;
    calculate_minmax_areas (i, figs, &areas[i]);
    output_minmax_areas (i, &areas[i]);
  }
  return 0;
}
```

Программа: подсчет площади

```
static void calculate_minmax_areas (enum figure_tag tag,
  struct figure *f, struct farea *fa) {
  for (; f->tag != LAST_FIGURE; f++)
    if (f->tag == tag) {
      double a = calculate_area (f);
      if (!fa->initialized) {
        fa->min = fa->max = a;
        fa->initialized = 1;
      } else {
        if (a < fa->min)
          fa->min = a;
        if (a > fa->max)
          fa->max = a;
      }
    }
}
```

Программа: подсчет площади

```
static double calculate_area (struct figure *f) {
  double a = 0.0;

  switch (f->tag) {
    case CIRCLE: a = f->u.fc.radius * f->u.fc.radius *
      M_PI; break;
    case RECTANGLE: a = fabs ((f->u.fr.ll.x –
      f->u.fr.ur.x) * (f->u.fr.ll.y - f->u.fr.ur.y));
      break;
    case TRIANGLE: <...>
    default: assert (0);
  }
  return a;
}
```

Программа: подсчет площади

```
static double calculate_area (struct figure *f) {
  double a = 0.0;
  switch (f->tag) {
    <...>
    case TRIANGLE: {
      double x1, y1, x2, y2;
      x1 = f->u.ft.b.x - f->u.ft.a.x;
      y1 = f->u.ft.b.y - f->u.ft.a.y;
      x2 = f->u.ft.c.x - f->u.ft.a.x;
      y2 = f->u.ft.c.y - f->u.ft.a.y;
      a = fabs (x1 * y2 - x2 * y1) / 2;
      break;
    }
  <...>
  }
  return a;
}
```

Программа: ввод

```
static int read_figure (struct figure *f) {
  char name;

  if (scanf (" %c", &name) != 1)
    return 0;

  switch (toupper (name)) {
    case 'C': f->tag = CIRCLE; break;
    case 'R': f->tag = RECTANGLE; break;
    case 'T': f->tag = TRIANGLE; break;
    default: return 0;
  }
  <...>
  return 1;
}
```

Программа: ввод

```
static int read_figure (struct figure *f) {
  <...>
  switch (f->tag) {
    case CIRCLE: if (scanf ("%lf%lf%lf", &f->u.fc.center.x,
      &f->u.fc.center.y, &f->u.fc.radius) != 3)
                    return 0;
                  break;
    case RECTANGLE: if (scanf ("%lf%lf%lf%lf",
      &f->u.fr.ll.x, &f->u.fr.ll.y, &f->u.fr.ur.x,
      &f->u.fr.ur.y) != 4)
                      return 0;
                    break;
    case TRIANGLE:  if (scanf ("%lf%lf%lf%lf%lf%lf",
      &f->u.ft.a.x, &f->u.ft.a.y, &f->u.ft.b.x,
      &f->u.ft.b.y, &f->u.ft.c.x, &f->u.ft.c.y) != 6)
                      return 0;
                    break;
  }
```

Программа: вывод и заголовочные файлы

```
#include <stdio.h>
#include <ctype.h>
#include <math.h>
#include <assert.h>
static void output_minmax_areas (enum figure_tag ft,
  struct farea *fa) {
  static char *fnames[LAST_FIGURE] = {"CIRCLE",
  "RECTANGLE", "TRIANGLE"};
  if (!fa->initialized)
    printf ("No figures of type %s were met\n",
            fnames[ft]);
  else {
    printf ("Minimal area of %s(s): %.5f\n", fnames[ft],
            fa->min);
    printf ("Maximal area of %s(s): %.5f\n", fnames[ft],
            fa->max);
  }
}
```