Курс «Основы программирования»

Федорук Елена Владимировна ст. преподаватель каф РК-6 МГТУ им.Н.Э.Баумана

Лекция №13

Структуры

Структура — это набор элементов, которые могут иметь различные типы.

Структуру можно представить себе как запись, состоящую из нескольких полей или элементов. Структуры обеспечивают удобный способ структурной организации связанных по смыслу переменных.

Агрегатным типом данных называется тип, конструируемый из элементов независимых (возможно различных) типов. Структуры являются одновременно агрегатным и производным типом данных.

Задание шаблонов и объявление структур

Новый тип данных для структуры создается путем описания шаблона структуры.

Шаблон структуры представляет собой список объявлений переменных, описывающий элементы структуры.

```
struct [имя_шаблона] {список_элементов};
```

Область действия шаблона структуры определяется также, как и для переменной. Если шаблон объявляется внутри блока, то он известен только в пределах этого блока. Однако обычно шаблон структуры задается в начале файла или в заголовочном файле. Таким способом шаблон можно сделать доступным всем определенным в файле функциям. Как только шаблон описан, его имя может быть использовано в любом месте файла для объявления соответствующих переменных.

Пример 1

```
struct emp {
        char name[21];
        char id[8];
        double salary;          /*это шаблон, память не выделяется */
        };
int main ()
{
struct emp prgmr;               /*это объявление: выделяется память под переменную prgmr*/
int num;
...
```

```
}
int f(void)
{
struct emp supervisor;      /*это объявление: выделяется память под переменную supervisor*/
...
}
```

Элементы структуры хранятся в памяти в том порядке, в каком они появляются в списке элементов. При этом в структуру могут быть включены неиспользуемые байты, необходимые для соблюдения требований процессора к выравниванию данных. Так числовые типы данных обычно выравниваются на границу слова. На рис. 1 показана гипотетическая ситуация, так как фактический размер структуры машинно-зависим.

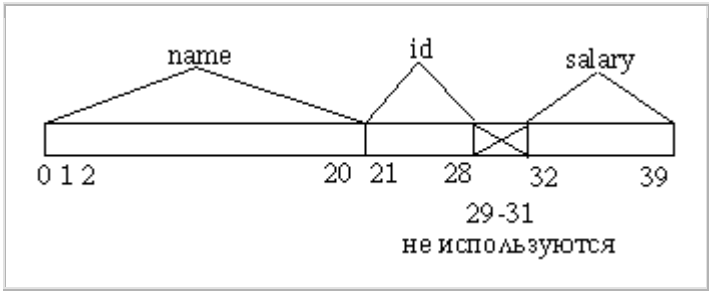

Рис. 1. Элементы структуры

Ниже рассматриваются альтернативные способы объявления структур.

Одновременное задание имени структуры и объявление структурных переменных

По желанию программиста, переменные типа структуры могут быть объявлены сразу после описания шаблона. Если шаблон и объявление переменной задаются вне функции, то переменная является внешней, в противном случае область ее видимости ограничивается блоком.

Пример 2

```
struct emp {
        char    name[21];
        char    id[8];
        double salary;
        } prgmr, employee[100], *p;
```

В прим. 2 объявляется структура **prgmr**, массив структур **employee** и указатель на структуру **p**.

Пропуск имени структуры

Другой способ состоит в том, чтобы опустить имя шаблона структуры. В этом случае все переменные данного типа должны быть перечислены после шаблона.

Пример 3

```
struct {
        char    name[21];
        char    id[8];
        double salary;
      } prgmr, employee[100], *p;
```

Использование объявления typedef

Третий способ состоит в том, чтобы использовать оператор typedef. Это избавит программиста от необходимости писать как ключевое слово **`struct`**, так и имя структуры при объявлении структурных переменных. Использование оператора **`typedef`** для задания однословного имени нового типа может также способствовать повышению удобочитаемости программы.

Пример 4

```
typedef struct emp {
                    char   name[21];
                    char   id[8];
                    double salary;
                   } EMPLOYEE;
EMPLOYEE prgmr, employee[100], *p;
```

Доступ к элементам структуры

Операция доступа к элементу структуры — операция "точка"(`.`), результат которой — значение элемента структуры.

`имя_структуры.имя_элемента_структуры`

Эта операция используется для доступа к элементу структуры с тем, чтобы присвоить ему значение, напечатать его, использовать его значение в арифметической операции и т.д. Эта операция является первичной и находится в самой верхней строке таблицы приоритетов операций языка Си.

Пример 1

```
#include <stdio.h>
#include <stdlib.h>
struct emp {
            char name[21];
            char id[8];
            double salary;      /*это шаблон, память не выделяется */
```

```
        };
int main()
{
  struct emp prgmr;          /*это объявление переменной - выделяется память */
  int num;
  ...
  gets(prgmr.name);
  gets(prgmr.id);
  gets(buf);
  prgmr.salary = atof(buf);
  printf("Name: %s\n", prgmr.name);
  printf("Id: %s\n", prgmr.id);
  printf("salary: %.2f\n", prgmr.salary);

  ...
  prgmr.salary *= 1.15;

  ...
}
```

Инициализация структуры

Инициализация структуры заключается в присваивании начальных значений элементам структуры. Структуры могут быть проинициализированы при их объявлении.

Инициализирующая запись - это заключенный в фигурные скобки список, элементы которого разделяются запятыми и являются константами. Любые неинициализированные элементы внешних или статических структур по умолчанию равны 0. Значения неинициализированных элементов автоматических структур неопределены.

Пример 1

```
struct course {
                char name[30];
                int  number;
                char nickname[30];
                } title = {"C for new programmers",
                           1633,
                          "TRR YYY"};
struct mailinfo {
                  char name[25];
                  char mailadr[30];
```

```
} proj_member [] = {
                        {"Ivanov Oleg", "rz3bb@imr"},
                        {"Klimov Alex", "rz3bb@ha"},
                        {"Petrova Elena", "rz3bb@wij"},
                        { "", ""}
                       };
```

```
int f (void)
{
static struct mailinfo admin = {"Administrator", "root"};
...
}
```

В качестве признака окончания массива часто используется "нулевая" запись, поэтому зачастую используют циклы подобные прим. 2. Цикл будет выполняться до тех пор, пока первый символ поля **`name`** структуры, на которую указывает указатель **`p`**, не равен **`'\0'`**.

Пример 2

```
for (p = proj_member; *p->name != '\0'; p++)
    printf ("%s %s \n", p->name, p->mail_addr);
```

Массивы структур

Массив структур — это массив, каждый элемент которого является структурой. В памяти элементы массива структур размещаются последовательно.

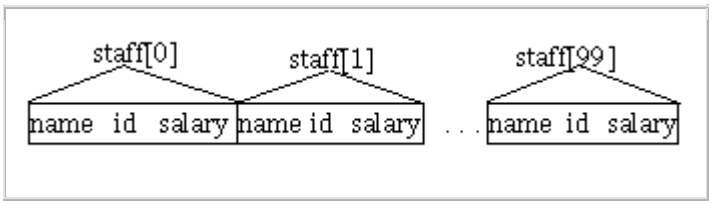

Рис. 1. Массив структур

Для обработки элементов массива структур используется операция доступа к элементу массива (квадратные скобки).

Массивы структур широко используются для структурной организации данных в прикладных программах и системном программном обеспечении.

Пример 1

```
/* Печать общей суммы зарплаты служащих */
#include <stdio.h>
```

```
#define NUM_EMPS 100
void fillarray(struct emp *, int);
struct emp {
            char name[21];
            char id[8];
            double salary;          /*это шаблон, память не выделяется */
            };


int main()
{
 struct emp staff[NUM_EMPS];       /*выделяется память под массив структур*/
 int num;
 double sal_tot = 0;
  fillarray(staff, NUM_EMPS);        /*заполняет массив структур*/
  for (i=0; i<NUM_EMPS; i++)
      sal_tot += staff[i].salary;
  printf ("total of salaries: %.2f\n", sal_tot);
}
```

Указатели на структуры

Указатель на структуру объявляется точно так же, как и указатель на данные простых типов: используется операция ***** и указывается тип данных. Тип данных структуры указывается заданием ключевого слова **`struct`** и имени шаблона этой структуры.

Пример 1

Объявление указателей:

```
int       *ip;
struct emp *sp;
```

Пример 2

```
#include <stdio.h>
#define NUM_EMPS 100
void fillarray(struct emp *, int);
struct emp {                              /*это шаблон, память не выделяется */
            char   name[21];
            char   id[8];
            double salary;
          };
int main()
{
```

```
struct emp staff[NUM_EMPS], *sp;    /*выделяется память под массив
                                     структур и под   указатель*/
double    sal_tot = 0;
fillarray(staff , NUM_EMPS);          /*заполняет массив структур данными */
for (sp = staff; sp != &staff[NUM_EMPS]; sp++)
    sal_tot += sp->salary;
printf("total of salaries: %.2f\n", sal_tot);
}
```

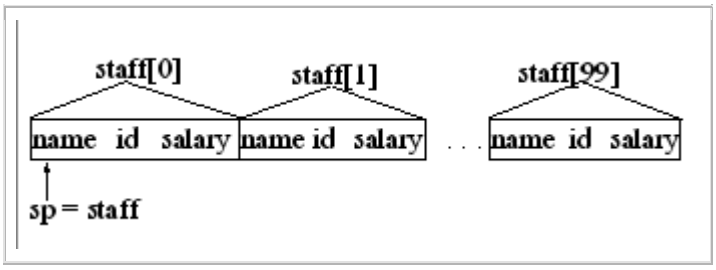

Рис. 1. Указатель на массив структур

В прим. 2 указатель **sp** после инициализации указывает на начало массива структур **staff**, т.е. содержит адрес первой структуры массива. Поскольку **sp** – указатель на структуру, то результат операции ***sp** является структурой, в данном случае **staff[0]**. Чтобы обратиться к полю **salary**структуры **staff[0]**, можно использовать следующие выражения:

- **staff[0].salary**
- **(*sp).salary** (круглые скобки требуются в этом контексте, чтобы учесть приоритет операций).

Операция доступа к элементу структуры через указатель

Ввиду того, что указатели на структуры используются очень часто, в языке Си есть специальная *операция доступа к элементу структуры через указатель* **(->)**, позволяющая сослаться на элемент структуры, заданный указателем на нее:

имя_указателя->имя_элемента

Эта операция является первичной и находится в самой верхней строке таблицы приоритетов операций языка Си.

После того, как указатель **sp** инициализирован и указывает на элемент массива структур **staff[0]**, следующие выражения эквиваленты:

- staff[0].salary /*имя_структуры.имя_элемента*/
- sp->salary /*имя_указателя->имя_элемента*/

Структура как параметр функции

Структура может быть передана функции как параметр. При этом на стеке в кадре вызываемой функции размещаются значения всех элементов структуры, а затем они ставятся в соответствие формальному параметру.

Функция может возвращать структурное значение.

В языке Си реализована *операция копирования структур* — присваивание значений элементов одной структуры элементам другой структуры того же типа при помощи операции присваивания. Первые компиляторы Си не обеспечивали этой возможности.

В прим. 1 показано, как в функцию передается структура и как функция возвращает структурное значение.

Пример 1

```
#include <stdio.h>
#include "emp.h"        /* содержит шаблон структуры emp */
int main()
{
  struct emp prgmr, raise(struct emp, double);
  ...
  printf("Old salary: %.2f\n", prgmr.salary);
  prgmr = raise(prgmr, .12);
  printf("New salary: %.2f\n", prgmr.salary);
}
struct emp raise(struct emp person, double increase)
{
 person.salary *= (1+ increase);
 return person;
}
```

Данная программа выполнялась бы более эффективно, если в функцию **`raise()`** передавался указатель на структуру. В этом случае при передаче управления в функцию передается лишь значение указателя, что делает доступными в функции элементы структуры. Этот вариант программы показан в прим. 2.

Пример 2

```
#include <stdio.h>
#include "emp.h" /* содержит шаблон структуры emp */
int main()
{
  struct emp prgmr;
  void raise(struct emp *, double);
  ...
```

```
  printf("Old salary: %.2f\n", prgmr.salary);
  raise(&prgmr, .12);
  printf("New salary: %.2f\n", prgmr.salary);
}
void raise(struct emp *person, double increase)
{
person->salary *= (1+ increase);
}
```

Самоссылающиеся структуры

Структура не может включать элементы того же типа, что и сама структура (рекурсивное определение). Однако несколько элементов структуры могут быть указателями на такие же структуры. Подобные *самоссылающиеся структуры* обычно используются для создания динамических структур данных таких, как, например, связные списки или деревья.

Пример 1

```
/* Связанный список */
struct info {
        int         num;
        float       sum;
        struct info *next
        };
```

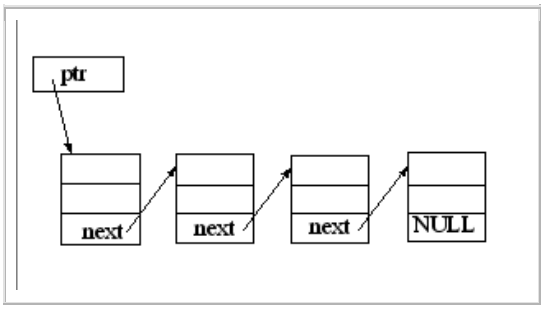

Рис. 1. Связный список

Пример 2

```
/* Дерево */
struct node {
        int         key;
        char        description [50];
        struct node *left, *right;
    };
```

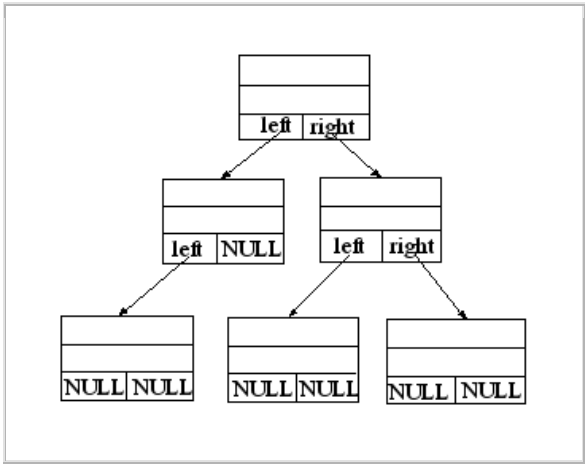

Рис. 2. Дерево

Объединения

Объединение — объект, который в каждый момент времени содержит один из нескольких элементов различных типов.

Все компоненты объявления структур, такие как шаблоны, имена типа, имена элементов и т.д. применимы и при объявлении объединений. Единственное отличие состоит в том, что при объявлении объединения вместо ключевого слова **`struct`** используется **`union`**.

Примечание 1

Объединение не может иметь битовые поля.

Операции доступа к элементу структуры (операция . и операция ->) могут применяться и к объединениям. Допустимы массивы объединений и указатели на объединения. Объединения могут передаваться функции как параметры и возвращаться функцией.

Пример 1

```
struct s_tag{
        char c;
        int i;
        doubl d;
        } s_item;
```

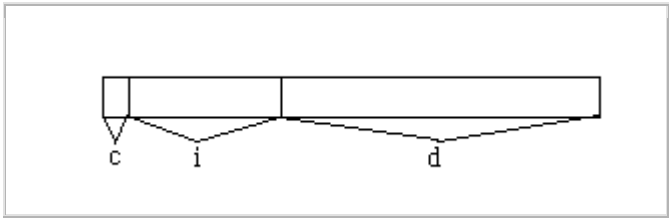

Рис. 1. Структура

Пример 2

```
union u_tag{
        char c;
        int i;
        doubl d;
        } u_item;
```

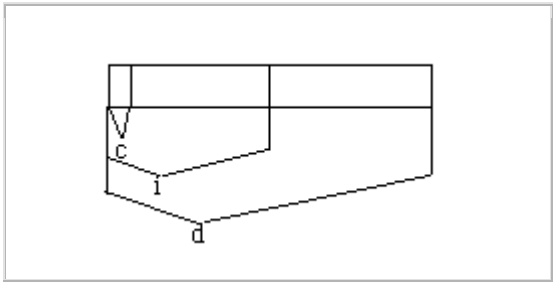

Рис. 2. Объединение

В отличие от структуры, объединение может в любой момент времени содержать только один из своих элементов. Объединение позволяет использовать одну область памяти для хранения различных видов данных в разные моменты времени. Фактически, объединение — это структура, в которой все поля начинаются со смещением 0, таким образом, поля накладываются друг на друга.

Пример 3

```
printf("размер структуры = %d\n", sizeof(s_item));
printf("размер объединения = %d\n", sizeof(u_item));
```

```
Вывод:
размер структуры = 16
размер объединения = 8
```

Объединения часто включаются в структуры, один из элементов которых является ключом, указывающим тип хранимого в памяти элемента объединения.

Пример 4

```
/* содержит информацию о работающих служащих и пенсионерах */
struct mail {
        char id;      /* a - active (), r - retired()*/
        union{
              struct{char name[30];
                     char dept[10];
                     char location[3];
                    } active;
               struct{char name[30];
                      char street[20];
                      char city_state[3];
```

```
            char zip[5];
          } retired;
      } info;
} preson;
```

Структура типа **`struct mail`** используется для хранения почтового адреса работающего служащего или пенсионера. При заполнении структуры этого типа информацией в нее заносится порция данных, соответствующая элементам **`active`** или **`retired`** объединения **`info`**. Поле **`id`** устанавливается равным **`'a'`** или **`'r'`** для указания фактически записанного в объединении элемента. При применении объединения используется меньше памяти, чем в случае применения структуры, которая имела бы идентичные поля, но некоторые из них не использовались бы.

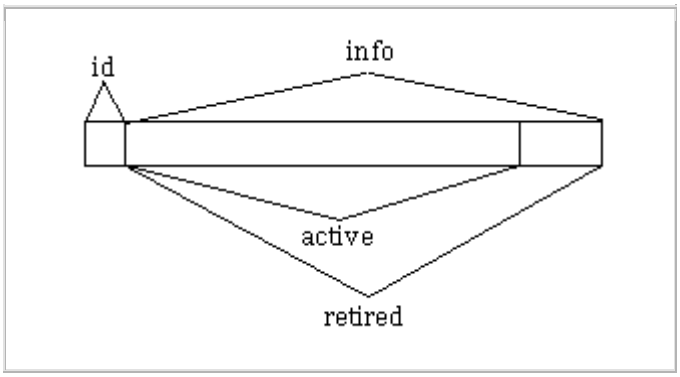

Рис. 3. Объединение как элемент структуры