Структуры, объединения, перечисления

СТРУКТУРЫ

Рассмотрим новый тип данных - структуру. Он не только гибок для представления разнообразных данных, но и позволяет создавать новые(пользовательские) типы данных.

Пример использования - создание каталога книг. Каждая книга имеет следующие атрибуты: шифр, название, автора, издательство, год издания, число страниц, тираж, цену. Это несколько массивов. Очень сложно организовать одновременную работу с каталогом, если нужно их упорядочить по названиям, авторам, цене и так далее. Лучше иметь один массив, в котором каждый элемент содержит всю информацию о книге.

Структура – это объект, состоящий из последовательностей поименованных элементов. Каждый элемент имеет свой тип.

Для определения нового типа данных нужно его описать(задать шаблон структуры):

```
struct book {
  char title [81];
  char author[30];
  float value;
};
```

book – это имя нового типа данных.

Структурный шаблон является основной схемой, описывающей, как образуется новый тип. **struct** - ключевое слово, имя типа структуры book - необязателен, если сразу определить имя переменной, то его можно не вводить.

```
struct {
  char title [81];
  char author[30];
  float value;
}libry;
```

Каждый элемент структуры определяется своим собственным описанием. Это переменные и массивы стандартных типов данных.

Шаблон является схемой без содержания. Он сообщает компилятору, как сделать что-то, но ничего не делает в программе, а вот создание структурной переменной, это и есть смысл слова «структура». Согласно шаблону под эту переменную выделяется память, равная сумме всех элементов (81).

struct book играет ту же роль, что и int, float перед именем переменной.

struct book doyle, panshin;

Для доступа к элементам структурной переменной используется операция точка. Имя переменной, точка, имя элемента структуры.

```
void main(void) {
  struct book libry;          //описание перем-й типа book
  puts("Введите название книги");
  gets(libry.title);
  puts("Введите фамилию автора");
```

```
  gets(libry.author);
  puts("Введите цену книги");
  scanf("%f",&libry.value);
  printf("%s, %s, %p.2f",libry.title,libry.author,libry.value);
 }
```

Структурную переменную можно инициализировать:

struct book libry={"Руслан и Людмила", "А.С.Пушкин", 1.50};

Массивы структур

Если переменных типа структура много, то определяется массив структур.

```
void main(void){
  struct book libry[100];
  int i;
   for(i=0; i<100; i++){
    puts("Введите название книги");
    gets(libry[i].title);
    puts("Введите автора книги");
    gets(libry[i], author);
    puts("Введите цену книги");
    scanf("%f",&libry[i].value);
   }
 }
```

Индекс применяется к имени массива структур libry[i].

Если libry[2].title[3] – это 4-й элемент в title в 3-й структуре типа book.

Вложенные структуры

Если одна структура содержится или "вложена" в другую, то говорят, что это вложенные структуры.

```
struct names{ char name[20];
              char fio[20];};
struct worker{ struct names people;
              char job[20];
              float money;};
void main(void){
struct worker driver = {{"Иван", "Иванов"},
                                "водитель", 1234.1};
```

Для обращения к элементу вложенной структуры применяется две операции «точка».

```
  puts(driver .people.name);
```

Указатели на структуры

Указателями на структуры легче пользоваться, чем самими структурами. Структуру не имеет смысла передавать в качестве аргумента в функцию в целях экономии памяти и быстродействия в отличии от указателя.

```
 struct worker *pdrv;
 pdrv = &driver;
```

```
struct worker driver[2];      //массив структур
a) pdrv = driver;             // pdrv <=> &driver[0];
pdrv+1 <=> &driver[1].
```

Доступ к элементу структуры осуществляется через операцию ->.

```
pdrv->job -> driver[0].job ->(*prdv).job
```

б) *pdrv->people.name*

Операции над структурами

1) Операция получения элемента.

```
driver.money=1234;
```

2) Операция косвенного получения элемента.

```
pdrv->money=3456;
```

Передача структуры в функцию

1. Можно передавать элемент структуры в качестве параметра в функцию. Тогда функция не знает, что это структура.

```
struct funds{
      char* bank;
      float fonds;
      char* name;
      float savef;
      }stan={"ПРБ", 1023.87, «Иванов И.И.», 123,45};
  float sum (float, float);
  void main(void){
       printf ("У Иванова И.И.всего %.2f рубл.\n", sum(stan.fonds,stan.savef));
  }
  float sum(float x, float y){
    return(x+y);
  }
```

Функция sum() не знает, что ей передается элементы структуры, важно, что они имеют тип float.

2. Если нужно, чтобы она воздействовала на элемент структуры, то нужно передвать адрес элемента и далее работать через указатель определенного типа.

```
modify(&stan.savef);
```

3. Сообщение функции, что она имеет дело со структурой. Для этого нужно передать адрес структуры в качестве параметра.

```
struct funds {...} stan={...};
void main(void){
    float sum (struct funds*);
    printf ("У Иванова И.И. %.2f рублей\n", sum(&stan));
 }
 float sum (stuct funds* money){
   return(money->fonds+money->savef);
```

```
 }
```

Указатель money ссылается на структуру funds. В отличие от массива имя структуры не является её адресом, поэтому указываем адрес &stan.

4. Имеется массив структур. В этом случае имя массива является его адресом.

```
struct funds {...}stans[2]={{...},{...}};
void main(void){
   float sum (struct funds*);
   printf ("Всего капитала %.2f рублей\n", sum(stans));
  }
float sum(struct funds* money){
    float summ;
    int i;
    for (i=0, summ=0;i<2; i++, money++)
        summ+=money->fonds+money->savef;
    return (summ);
  }
```

money <=> &stan[0]; увеличивается money++, ссылаемся на stan[1].

Пример 1. Определить номер дня в году.

```
struct date { int day; int month;
          int year; int yearday;}d={25,3,1999};
int date_tab[2][13]={{ 0,31,28,31,30,31,30,31,31,30,31,30,31},
                       { 0,31,29,31,30,31,30,31,31,30,31,30,31}};
int day_of_year(struct date *pd) {
int i, day, l=0;
day = pd->day;
if(pd->year%4==0&&pd->year%100!=0||pd->year%400==0)  //год высок.
        l=1;
for(i=0; i< pd->month; i++)
        day+=date_tab[l][i];
return(day);
}
void main(void) {
d.yearday=day_of_year(&d); - вызов функции.
printf("%d", d.yearday);
}
```

ОБЪЕДИНЕНИЯ

Объединение – это группирование переменных, которые разделяют одну и ту же область памяти. В зависимости от интерпретации осуществляется обращение к той или другой переменной объединения. Все переменные, что включены в объединение начинаются с одной границы.

Объединение позволяет представить в компактном виде данные, которые могут изменяться. Одни и те же данные могут быть представлены разными способами с помощью объединений.

Точно также как и структуры, объединения требуют объявления типа (шаблона) и объявления переменной этого типа.

Форма объявления типа (шаблона) объединения? Ключевое слово union

Объявление объединения (типа объединения или шаблона объединения) начинается с ключевого слова union.

```
union имя_типа_объединения
{
   тип переменная1;
   тип переменная2;
   ...
   тип переменнаяN;
};
```

где

- *имя_типа_объединения* – непосредственно имя новосозданного типа;
- *переменная1*, *переменная2*, *переменнаяN* – переменные, которые есть полями объединения. Эти переменные могут быть разных типов;
- *тип* – тип переменной, который есть полем объединения.

Тип переменной может быть:

- базовым типом, принятым в языке C++/CLI;
- тип структура;
- тип объединения;
- тип класс.

Длина объединения.

Длина объединения – это размер памяти в байтах, которая выделяется для одной переменной этого типа объединения.

Длина объединения вычисляется как максимум из всех длин (размеров в байтах) отдельных полей шаблона. Следует напомнить, что одно поле – это объявление одной переменной в объединении.

Пример: Пусть задан тип объединения, которое содержит переменные типов с плавающей точкой

```
// объявление типа "объединение Floats"
union Floats
{
   float f; // рассматривается 4 байта
   double d; // рассматривается 8 байт
};
```

Тип объединения Floats содержит 2 переменные с именами *f* и *d*. Переменная *f* есть типа float, переменная *d* есть типа double. Для переменной *f* типа float рассматривается (принимается во внимание) 4 байта. Для переменной *d* типа double принимается во внимание 8 байт, так как компилятор выделяет для этого типа именно 8 байт.

Чтобы использовать объединение Floats в другом программном коде (методе, обработчике события и т.п.) нужно объявить переменную типа Floats как показано ниже

```
Floats Fl;
int d;
Fl.f = 20.5; // Fl.d не определено
Fl.d = -100.35; // теперь Fl.f не определено

d = sizeof(Fl); // d = 8
```

Поскольку размещение переменных в памяти условно начинается с одного адреса, то для переменной Fl типа Floats выделяется 8 байт памяти. Это связано с тем, что переменная типа double требует больше памяти для своего представления чем переменная типа float.

На рисунке 1 отображено размещение (интерпретация) переменных *f*, *d* из объединения Floats.

Рис. 1. Представление переменных *f*, *d* в объединении Floats

⇑

5. Как осуществляется доступ к полям объединения?

Доступ к полям объединения осуществляется так же, как и для структуры:

- с помощью символа **‘ . ‘**;
- с помощью последовательности символов **‘->’** в случае, когда объявлена переменная-указатель на объединение.

⇑

6. Пример объявления и использования указателя (*) на объединение

Работа объединений с неуправляемыми (*) указателями точно такая же, как и работа **структур с неуправляемыми указателями**.

В нижеследующем примере объявляется неуправляемый указатель на объединение типа Ints

```
// указатель на объединение
Ints *p; // неуправляемый указатель

// выделить память для объединения
p = new Ints;

// доступ к полям с помощью указателя
pI->a = 200;
pI->b = 3400;
```

7. Вложенные объединения (структуры, классы) в шаблоне объединения. Пример

Шаблон объединения может включать поля, что есть структурами, объединениями и классами.

В примере ниже объявляется шаблон объединения с именем Types, содержащий два вложенных объединения Floats и Ints, структуру ArrayOfChars и класс MyPoint.

Объявление структур и объединений имеет следующий вид

```
// объединение целочисленных типов
union Ints
{
   unsigned short int a;
   unsigned int b;
   unsigned long int c;
};

// структура, содержащая 2 строки
struct ArrayOfChars
{
   char A[10];
   char B[8];
};

// объявление типа "объединение Floats"
union Floats
{
   float f; // рассматривается 4 байта
   double d; // рассматривается 8 байт
};
```

Объявление типа объединение Types с вложенными сложными типами Ints, Floats, ArrayOfChars.

```
...

// объявление типа "объединение Types"
```

```
union Types
{
   Floats Fl; // объединение
   Ints I;   // объединение
   ArrayOfChars A; // структура
};
```

Использование объединения Types в некотором программном коде:

// объявить переменную типа "объединение Types"

Types T;

```
// изменить значения полей переменной T
T.Fl.f = (float)20.35; // объединение Floats
T.I.b = 230; // объединение Ints
T.A.A[2] = 'A'; // структура ArrayOfChars
```

Массив объединений.

```
// Пример объявления и использования массива объединений
Floats F[5]; // объявляется массив из 5 объединений типа Floats

// заполнение значений полей
for (int i=0; i<5; i++)
{
   F[i].d = i*0.2 + i*i;
}
```

Особенности применения операции sizeof() для объединений и структур?

В программах на C++ для определения размера переменной типа «структура» или «объединение» обязательно нужно использовать операцию sizeof. Определение размера «вручную» есть ошибочным поскольку:

- размеры некоторых встроенных типов (например тип int) могут быть разными для разных компьютеров. Например, на одних платформах для типа int будет выделено 2 байта, на других 4 байта;
- компилятор делает так называемое «выравнивание памяти» на границе слова (2 байта) или абзаца (16 байт). Например, если компилятор делает выравнивание на границе абзаца, то структура (объединение) типа ArraysOfChars:

```
// структура, содержащая 2 строки
struct ArrayOfChars
{
   char A[10];
   char B[8];
};
```

может занимать в памяти 24 байта. Так как для массива A выделяется 16 байт а не 10 байт. Компилятор дополнительно выделяет 6 байт чтобы реализовать выравнивание на границе абзаца.
Таким образом, использование операции sizeof() для определения типа структуры или объединения гарантирует переносность программного кода.

Синоним имени типа

Встречаются ситуации, когда удобно ввести синоним для имени некоторого типа.
Строится синоним имени с помощью ключевого слова typedef.

Примеры:

```
typedef int INT          //INT-синоним типа int
INT x, y;

typedef unsigned size_t;
size_t x, y;             //переменная типа unsigned

typedef char string[225];
string array;            //char array[225];
```

1. Функция typedef даёт имена типам данных.

2. Выполняется компилятором.

3. Более гибка, чем #define.

Испоьзование типа real вместо float:

```
typedef float real;
real x, y[5], *px;
```

если определение расположено внутри функции, то область действия локальна, вне функции глобальна.

```
typedef char* STRING     //STRING-идентификатор указателя на тип char.
STRING name, sign;       //char*name,*sign;
```

Определение именованных констант

Существуют 3 вида именованных констант:
- имя любого массива или функции;
- имена членов перечисления;
- любое имя любого типа, в определении которого присутствует модификатор **const.**

```
const i = 5;
const char *ip = &i;
```

Поскольку модификация такого объекта-константы запрещена, он должен быть инициализирован.

```
const int *ip;       //константой является объект, на который указывает указатель;
int* const ip;       //сам указатель является константой
```

```
const char *pc = "Это строка";
pc[2] = 'a';                    //ошибка
pc = "Это другая строка";       //верно
char* const pc = "Это строка";
pc[2] = 'a';                    //верно
pc = "Это другая строка";       //ошибка
```

Использование const предпочтительнее по сравнению с #define, так как использование константы контролирует компилятор.

Перечисления

Спецификатор **enum** позволяет программисту создавать собственные типы.

enum weekDays {Monday, Tuesday, Wensday, Thursday, Friday};

Идентификаторы перечисления представляют собой целочисленные переменные, которые по умолчанию имеют значения 0,1,..., если не указаны другие значения.

weekDays days;

Переменная days теперь может принимать одно из 5 значений.

days = Wensday;

Пример 2.

enum colors {Red=2, Green=3, Grey};

Если задано значение впереди стоящему члену перечисления, то Grey по умолчанию будет равен 4.

Пример 3.

```
enum VIDEO_BASE_ADDRES { VGA_EGA=0xA000000, CGA=0xB800000,
                         MONO=0xB000000};
```

Битовые поля

В некоторыя задачах для экономии памяти необходимо упаковывать несколько объектов в одно машинное слово. В Си для этого определяются поля и доступ к ним. Поле – это последовательность битов внутри одного целого значения.

```
struct {    unsigned a:8;
            unsigned b:6;
            unsigned c:2;}d;
```

Определяем структуру d, содержащую поле a – 8 битов, поле b – 6 битов, c – 2 бита. Поля описываются как unsigned, чтобы подчеркнуть, что это величины без знака. Отдельные поля теперь обозначаются как d.a, d.b, d.c. С полями можно выполнять различные операции.

d.*a= d.b=(d.c<<2)+6;*

Поля не могут переходить за границу слова в ЭВМ. Если же очередное поле не помещается в частично заполненное слово, то под него выделяется новое слово. Поля могут

быть безымянными. Используются как заполнители. Для принудительного перехода на новое слово используется специальный размер 0.

```
struct {unsigned a:8;
                :2;
        unsigned b:6;
                :0;
        unsigned c:12;} d;
```

Битовые поля и объединения можно применять для неявного преобразования типов.

Пример 1.

```
struct DOS_DATE { unsigned int day:5;
                  unsigned int month:4;
                  unsigned int year:7;};
union DATE_CONV { unsigned int packed_date;
                    struct DOS_DATE unpacked_date;};
typedef union DATE_CONV DATE
void main(void) {
struct ffblk ff;          //структура в которую читается информация о
//файле из каталога, описана в <dir.h>
        int done=findfirst("*.*", &ff,0);       //ищет первый файл в каталоге
        if(!done) {
                DATE d;
                d.packed_date=ff.ff_date;
printf(%2d/%2d/%4d", d.unpacked_date.day, d.unpacked_date.month, d.unpacked_ date
.year+1980);
        }
        }
```