Структуры

Часто реальный объект характеризуется величинами разного типа, для их описания используются структуры.

Структура – это совокупность данных разного типа, лежащих в непрерывной области памяти и объединенных общим именем.

Она позволяет строить новые типы данных языка Си. В других языках программирования структуры называют записями или кортежами.

Описание структуры выглядит следующим образом:

```
struct имя_структуры  {
    описание полей структуры
};
```

Структуры

struct имя_структуры {

описание полей структуры

}; ; обязательна

Здесь **имя_структуры** – идентификатор, соответствующий синтаксису языка Си,

описания полей структуры – любая последовательность описаний переменных, имена и типы этих переменных могут быть произвольными.

Структуры

Опишем вектор в трехмерном пространстве, который задается тремя вещественными координатами x, y, z:

```
struct R3Vector {
    double x;
    double y;
    double z;
};
```

При описании структуры память не выделяется, только лишь описывается новый тип.

Структуры

Таким образом, вводится новый тип "struct R3Vector"; объект этого типа содержит внутри себя три вещественных поля с именами x, y, z. После того как структура определена, можно описывать переменные такого типа, при этом в качестве имени типа следует использовать выражение struct R3Vector. Например, ниже описываются вещественные вектора в трехмерном пространстве с именами ***u, v*** и ***a, b***, а также выделяется память.

struct *R3Vector u, v*;

Объявлены переменные структурного типа R3Vector

Правила C++ разрешают опускать спецификатор **struct**

R3Vector a, b;

Объявлены переменные структурного типа R3Vector

Структуры

Рассмотрим пример структуры на примере набора сведений Почтовый адрес:

```
struct address {  // почтовый адрес
  char* name;       // имя
  int   number;     // номер дома
  char* street;     // улица
  char* town;
  long   zip;
} id1, id2;
```

struct – ключевое слово;

address – имя структурного типа, спецификатор типа (аналогично int, char и т.д.)

name, number, street, town, zip – элементы структуры (поля).

id1, id2 – объекты структурного типа address.

Структуры

Для **инициализации** переменных структурного типа можно использовать:

```
address jd = {
"Иванов", 15,
"ул. Амурская",
"Благовещенск",
675000
};
```

Порядок следования должен строго соответствовать порядку переменных-членов в определении структурного типа.

Замечание: инициализация данных при объявлении структуры запрещена.

Структуры

Причем, если количество инициализирующих значений превышает количество членов структуры – это ошибка.

Если же их меньше, чем членов структуры, то имеющиеся значения используются для инициализации переменных-членов в порядке их объявления.

Остальным переменным-членам, которым не хватило инициализирующих значений, присваиваются нулевые значения соответствующего типа.

Структуры

Переменные типа ***address*** могут описываться точно также, как другие переменные, а доступ к отдельным членам (полям) получается с помощью операции **.** (точка).

Например:

```
address jd;
jd.name = "Иванов";
jd.number = 15;
jd.street = "ул. Амурская";
jd.town = "Благовещенск";
jd.zip = 675000;
```

Структуры

Итак, имеется возможность работать с полями структуры.

Например, в следующем фрагменте в векторе ***w*** вычисляется векторное произведение векторов ***u*** и ***v*** трехмерного пространства:

w = u × v.

```
struct R3Vector u, v, w;
// Вычисляем векторное произведение w = u * v
w.x = u.y * v.z - u.z * v.y;
w.y = (-u.x) * v.z + u.z * v.x;
w.z = u.x * v.y - u.y * v.x;
```

Структуры

Определение структуры обычно располагается вне определений функций (подобно определению глобальных констант).

В этом случае тип структуры доступен всем функциям программы. После такого определения – имя структуры может использоваться аналогично базовым типам int, char и т.д.

Параметры структурного типа могут передаваться в функции как по значению, так и по ссылке.

Возвращаемое функцией значение также может иметь структурный тип.

Структуры

С объектами типа структура можно работать как с единым целым, например, для структур одного и того же типа допускается операция присваивания:

struct ***R3Vector u, v***;

. . .

u = v; //Копируем вектор как единое целое

копирование структур сводится к переписыванию области памяти.

Структуры

Операции сравнение (== и !=) не определены.

Однако пользователь может определить эти операции, используя прием перегрузки операций.

```
struct  R3Vector u, v;
. . .
if (u == v) {//Ошибка! Сравнивать структуры нельзя
    . . .
}
```

Структуры

В приведенных выше примерах все поля структуры ***R3Vector*** имеют один и тот же тип ***double***, однако это совершенно не обязательно.

Полями структуры в свою очередь могут быть структуры.

Пример: плоскость в трехмерном пространстве задается точкой и вектором нормали, ей соответствует структура ***R3Plane***. Точке трехмерного пространства соответствует структура ***R3Point***, которая определяется аналогично вектору.

Никаких ограничений на уровень вложенности структур нет.

Полное описание всех трех структур:

```
struct R3Vector {     // Вектор трехмерного пространства
    double x;
    double y;
    double z;
};
struct R3Point {   // Точка трехмерного пространства
    double x;
    double y;
    double z;
};
struct R3Plane {        // Плоскость в трехмерном пр-ве
    struct R3Point origin;     // точка в плоскости
    struct R3Vector normal;      // нормаль к плоскости
};
```

Структуры

Пусть ***plane*** — это объект типа плоскость.

struct R3Plane ***plane***;

Для того, чтобы получить координату ***x*** точки плоскости, надо два раза применить операцию доступа к полю структуры (операцию "точка")

plane.origin.x

Структуры

К структурным объектам типа указатель ***обращаются с*** использованием операции **-> (стрелка)**.

Например:

```
void print_addr (address* p)
{  cout << p->name << "\n" << p->number <<
 "\n" << p->street << "\n" << p->town << "\n" <<
    p->zip << "\n";
}
```

Вызов print_addr (*&jd*) ;

Имя типа становится доступным сразу после того, как оно встретилось, а не только после того, как полностью просмотрено все описание.

Например, возможно объявление:

```
struct link{
 // информационные поля
 link* previous;  // указатель (превьюз - предыдущий)
 link* successor;  // указатель (саксесор -удачный)
 };
```

Новые объекты структурного типа не могут быть описаны, пока все описание не просмотрено, так например, следующее объявление:

```
struct no_good
{      no_good member;  };
```

является ошибочным (компилятор не может установить размер no_good).

Структуры

Чтобы дать возможность двум (или более) структурным типам ссылаться друг на друга, можно предварительно описать имя структурного типа. Например:

```
struct list;        // должна быть определена позднее
struct link {
    link* pre;
    link* suc;
    link* member_of;
};
struct list
{     link* head;  }
```

Без первого описания list описание link вызвало бы синтаксическую ошибку.

Структуры

Рассмотрим проектирование символьной таблицы, в которой каждый элемент содержит **имя** и **значение**, причем значение может быть либо строкой, либо целым числом.

Тогда придется ввести дополнительное поле ***type*** по значению которого определяется тип вводимого значения. Значение храниться либо в поле ***string_value***, либо в поле ***int_value***.

```
struct entry {
   char* name;
   char  type;
   char* string_value;     // используется если type = = 's'
   int   int_value;        // используется если type = = 'i'
};
```

Структуры

Для рассмотренного типа ***entry*** приведем пример функции – печати значений:

```
void print_entry(entry* p)
 {   cout<<p->name;
     switch p->type
     {   case 's':     cout << p->string_value;    break;
     case 'i':     cout << p->int_value;          break;
     default:    cerr << "испорчен type\n"; break;
     }
 }
```

Объединения

Поскольку ***string_value*** и ***int_value*** никогда не могут использоваться одновременно, ясно, что пространство пропадает впустую.

Ситуацию можно исправить, указав, что оба поля структуры должны быть членами ***union*** (объединения); например, так:

```
struct entry {
        char* name;
        char  type;
            union {
      char* string_value; //используется если type== 's'
             int   int_value;     // используется если type == 'i'
            };
};
```

Объединения

При размещении **entry** поля структуры ***string_value*** и ***int_value*** будут храниться по одному и тому же адресу памяти.

Все члены объединения вместе занимают лишь столько памяти, сколько занимает наибольший член объединения.

Структуры

Многие компьютеры требуют, чтобы объекты определенных типов выравнивались в памяти только по некоторым зависящим от архитектуры границам: например: целое должно быть выровнено по границе машинного слова.

Поэтому размер объекта структурного типа нельзя вычислить просто как сумму размеров его членов.

Наиболее эффективно обрабатывают объекты, которые выровнены в машине.

Такой механизм приводит к "***дырам***" в структурах при их хранении в памяти.

С помощью операции ***sizeof*** можно определить размер памяти, которая соответствует идентификатору или типу. Операция sizeof имеет следующий формат:

sizeof(выражение)

Структуры

Когда ***sizeof*** применяется к имени типа структуры или объединения или к идентификатору имеющему тип структуры или объединения, то результатом является фактический размер структуры или объединения, который может включать участки памяти, используемые для выравнивания элементов структуры или объединения.

Таким образом, результат может не соответствовать размеру, получаемому путем сложения размеров элементов структуры.

Пример:

```
struct    {
   char    h;
   int      b;
   double f;
   } str;
     int  a1;      a1=sizeof(str);
```

1 байт
4 байта
8 байт

a1 получит значение 16, в то же время если ...жить длины всех используемых в ...уктуре типов, то получим, что длина ...уктуры ***str*** должна быть равна 13.

Структуры

```
#include<iostream.h>
#include <stdio.h>
struct  {
    char     h;
    int          b;
    double f;
    } str;
int main()
{   int  a1;
    a1=sizeof(str);
    cout<<a1<<"\n";
    system("pause");
}
```

D:\ковчег\Алутина\Программирование\К занятиям\Задания к з

```
16
Для продолжения нажмите любую клавишу . . .
```

```
#include<iostream.h>
#include <stdio.h>
struct  proba {
     char   h;
     double f;
     int    b;

   } str;
int main()
{  int  a1;
   a1=sizeof(str);
   cout<<a1<<"\n";
   system("pause");
}
```

D:\ковчег\Алутина\Програ

```
24
Для продолжения нажми
```

```
#include<iostream.h>
#include <stdio.h>
struct  proba {
     double f;
     char   h;
     int    b;
} str;
int main()
{  int  a1;
   a1=sizeof(str);
   cout<<a1<<"\n";
   system("pause");
}
```

D:\ковчег\Алутина\Программирование\К занятиям\Задания

```
16
Для продолжения нажмите любую клавишу . . .
```

Структуры

Пример:

```
#include <iostream>
#include <iomanip.h>
#include <string.h>


using namespace std;
struct library {
    int shifr;
    char author[20];
    char title[20];
    int year;
};
```

Пример программы - библиотека

```
void Print ()
{
cout<<"=========================================\n";
cout<<"||  Shifr book  ||   avtor  ||   NAME   ||    year издания    ||\n";
cout<<"=========================================\n";
} ;
void PrintData (library *ptr, int i)
{
cout<<"||"<<setw(10)<<ptr[i].shifr<<"||"<<setw(20)<<ptr[i].author;
                                cout<<"||"<<setw(20)<<ptr[i].title<<"||"<<
setw(15)<<ptr[i].year<<"||"<<endl;
} ;
void PrintEnd ()
{
cout<<"=========================================\n";
} ;
```

```
int main ()
{    int i, n, m, god, v=0, g=0;
     char avtor[20];

struct library ptr[50];

cout<<"Vvedite kol-vo books \n";     cin>>n;
for (i=0; i<n; i++)
  {cout<<"Information of"<<(i+1)<<"book :\n";
   cout<<"Shifr book \n";
   cin>>ptr[i].shifr;
   cout<<"FIO avtora  \n";
   cin>>ptr[i].author;
   cout<<"Name book \n";
   cin>>ptr[i].title;
   cout<<"Year издания  \n";
   cin>>ptr[i].year;
  };
```

```
do  { cout<<"Выберите действие :\n";
        cout<<"1 – список книг в алфавитном порядке\n";
        cout<<"2 – список книг необходимого года \n";
        cout<<"3 – список книг определенного автора \n";
        cout<<"4 - выход \n";
        cin>>m;
switch (m)
{ case 1:
      { Print();
         for (char w='A'; w<'Z'; w++)
          for (i=0; i<n; i++)
           if(ptr[i].title[0]==w)    PrintData(&ptr,i);
   PrintEnd();
  cout<<"Для выхода в меню нажмите любую клавишу ";
  getch();    break;
      };
```

```
case 2:
   {  cout<<"Vvedite нужный year : \n";
      cin>>god;
      for (i=0; i<n; i++)
       {
         if (ptr[i].year==god)
         { v+=1;
           if (v==1)  Print();
           PrintData(&ptr, i);
         };
       };
    if(v!=0) PrintEnd();
    else cout <<"No books необходимого year"<< endl;
    v=0;
    cout<<"For exit in menu push any клавишу ";
    getch();     break;
};
```

```
 case 3:
{  cout<<"Vvedite FIO avtora : \n";
   cin>>avtor;
   for (i=0; i<n; i++)
       {  if (strcmp(ptr[i].author, avtor)==0)
              {  g+=1;
                 if (g==1)   Print();
                  PrintData(&ptr,i);
              };
       };
   if(g!=0) PrintEnd();
   else cout <<"No books данного avtora"<< endl;
   g=0;
   cout<<"For exit in menu push any клавишу ";
   getch();   break;
};
```

```
case 4:
        break;
   }
}
      while (m!=4);
system("Pause");
 }
```