Практическое занятие 18

Тема: Структуры данных

Цель работы. Изучить принципы создания и использования структур данных.

Теоретическая часть

Объекты, представляемые в программе, обладают рядом разнообразных и разнотипных свойств. Для конструирования объектов на C++ предусматривается использование структур и объединений. Структура может быть представлена как некоторый набор разнотипных и/или однотипных данных, совокупность которых рассматривается как новый, пользовательский тип данных. Структура объявляется с помощью ключевого слова struct, за которым следует необязательное имя тега для создания нового типа данных и указываемый в фигурных скобках шаблон, по которому будут создаваться переменные структурного типа. Шаблон содержит указываемые через точку с запятой объявления полей или членов структуры. Объявление поля состоит из указания типа и имени переменной:

```
struct NewType
{
typel Namel;
type2 Name2;
….....................
typeN NameN;
};
```

Использование структурированных данных в программе возможно, если будет объявлен какой-нибудь объект вновь созданного типа. Например, для приведенного случая можно указать:

```
NewType Variable;
```

Будет создан структурированный объект Variable типа NewType. Объект Variable может создаваться непосредственно при объявлении структуры:

```
struct NewType
{
typel Namel;
type2 Name2;
….....................
typeN NameN;
} Variable;
```

Инициализация элементов структуры (присвоение им начальных данных) может быть произведена непосредственно при объявлении. Присваиваемые значения указываются через запятую в фигурных скобках, например:

```
struct MyStruct
{
int iVariable;
long lValue;
char Str[10];
} mystruct = {10, 300L, "Hello"};
```

Пусть в программе необходимо создать некоторую базу данных, содержащую информацию о жилых домах микрорайона. Известно, что потребуются данные о номере микрорайона, названии улицы, номере дома, количестве этажей, числе квартир, наличии

прилегающей стоянки. Пусть под номер микрорайона отводится беззнаковое короткое целое, название улицы может быть закодировано строкой из 50-ти символов, номер дома представим как строку из пяти символов (на случай, если номер дома дополнительно содержит букву), количество этажей и число квартир - беззнаковое короткое целое, а информация о наличии стоянки - логическая переменная. Ниже представлена структура, в которой учтено все сказанное:

```
struct HOUSE
{
unsigned short RegNum;
char Street[51];
char HouseNum[6];
unsigned short MaxFloorNum;
unsigned short MaxFlatNum;
bool Parking;
};
```

Для использования полученного типа HOUSE необходимо объявить соответствующую переменную House:

```
HOUSE House;
```

Чтобы записать или прочитать данные структуры, после имени объекта необходимо поставить символ точки (.), за которым следует имя члена структуры. Вся конструкция рассматривается как единая переменная. Например, заполним имеющийся объект House:

```
House.RegNum = 524;
strcpy (MyHouse. Street, " ул. Гоголя") ;
strcpy(MyHouse.HouseNum, "2-a") ;
House.MaxFloorNum = 7;
House.MaxFlatNum = 84;
House.Parking = true;
```

Присвоение значений целочисленным и логическим данным производится через знак «=», а заполнение строковых членов структуры осуществляется с помощью функции работы со строками strcpy ().

Для определения размера структурированного объекта в памяти к нему применяют оператор (или функцию) sizeof :

```
int i = sizeof(HOUSE) ;
```

Организация структуры может позволить сэкономить память благодаря использованию битовых полей. В этом случае объявление поля структуры имеет вид:

объявление__поля : константное_выражение;

где **объявление__поля** - объявление типа и имени поля структуры;
константное_выражение определяет длину поля в битах.

Тип поля должен быть целочисленным (int, long, unsigned, char). Объявление_поля может отсутствовать. В этом случае в шаблоне структуры пропускается указанное после двоеточия число битов. Таким образом, если разработчик знает наверняка, что элемент структуры может принимать значения 0 или 1, для него можно отвести один бит. Дальнейшая работа с таким элементом структуры ведется с использованием поразрядных логических операций. Реализация битовых полей тесно связана с аппаратной платформой,

на которой функционирует компилятор, поэтому детали использования битовых полей описываются в документации компилятора.

Пусть необходимо создать структуру, содержащую информацию о дате и времени некоторых событий. Это можно сделать следующим образом:

```
struct DATETIME {
unsigned short Year; // год
unsigned short Month; // месяц
unsigned short Date; // дата
unsigned short Hour; // часы
unsigned short Minute; // минуты
unsigned short Second; // секунды
}
```

Объект типа DATETIME в памяти будет занимать 6(элементов) × 2(байта) = 12 байт. В описании такой структуры присутствует значительная избыточность, так как год может принимать значения от 0000 до 2048 (11 бит), месяц - от 1 до 12 (4 бита), дата - от 1 до 31 (5 бит), часы 0-23 (5 бит), минуты и секунды - от 0 до 59 (по 6 бит на каждый элемент). Применяя битовые структуры, приведенная выше структура примет вид:

```
struct DATETIME2 {
unsigned Year : 11; // год 11 бит
unsigned Month : 4; // месяц 4 бита
unsigned Date : 5; // дата 5 бит
unsigned Hour : 5; // часы 5 бит
unsigned Minute : 6; // минуты 6 бит
unsigned Second : 6; // секунды 6 бит
}
```

Экземпляр модифицированного типа DATETIME2 будет занимать 11+4+5+5+6+6=37 бит. В 1 байте — 8 бит, поэтому для целого счета необходимо использование еще 3 бит. Очевидно, что расширяемым полем является Year, поэтому DATETIME2 примет вид:

```
struct DATETIME2 {
unsigned Year : 14; // год 14 бит
unsigned Month : 4; // месяц 4 бита
unsigned Date : 5; // дата 5 бит
unsigned Hour : 5; // часы 5 бит
unsigned Minute : 6; // минуты 6 бит
unsigned Second : 6; // секунды 6 бит
}
```

Общий объем структуры теперь будет не 12 байт, а только 5 байт.

Пример 1

```
#include <iostream>

using namespace std;

int main()
{

   struct DATETIME{
      unsigned short Year;// год
      unsigned short Month; // месяц
      unsigned short Date; // дата
      unsigned short Hour; // часы
```

```
    unsigned short Minute; // минуты
    unsigned short Second; // секунды
};

struct DATETIME2{
    unsigned Year:14;// год 14 бит
    unsigned Month:4; // месяц 4 бита
    unsigned Date:5; // дата 5 бит
    unsigned Hour:5; // часы 5 бит
    unsigned Minute:6; // минуты 6 бит
    unsigned Second:6; // секунды 6 бит
};

DATETIME d1={2013,9,21,12,22,43};
DATETIME2 d2={2014,4,2,11,30,59};

cout << "Размер структуры DATETIME = " << sizeof(d1) << endl;
cout << "год = " << d1.Year << endl;
cout << "месяц = " << d1.Month << endl;
cout << "день = " << d1.Date << endl;
cout << "час = " << d1.Hour << endl;
cout << "мин = " << d1.Minute << endl;
cout << "сек = " << d1.Second << endl << endl;

cout << "Размер структуры DATETIME2 = " << sizeof(d2) << endl;
cout << "год = " << d2.Year << endl;
cout << "месяц = " << d2.Month << endl;
cout << "день = " << d2.Date << endl;
cout << "час = " << d2.Hour << endl;
cout << "мин = " << d2.Minute << endl;
cout << "сек = " << d2.Second << endl;

return 0;
}
```

Структуры как аргументы функций

Если в тело функции необходимо передать информацию, структурированную по определенному принципу, то в качестве параметра в прототипе функции можно указать пользовательский тип данных, сформированный объявлением структуры. Например, если была объявлена структура

```
struct ALLNUMB
{
    int nVar;
    long lVar;
    short shVar;
    unsigned int uiVar;
    };
```

то прототип функции может иметь вид:

```
void Func (ALLNUMB) ;
```

Функция может также возвращать объект типа структуры:

```
ALLNUMB Func2(ALLNUMB);
```

Рассмотрим пример передачи в функцию описанной выше структуры House для вывода на экран названия улицы и номера дома, содержащихся в ней.

Пример 2

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

using namespace std;

struct HOUSE {
unsigned short RegNum;
char Street[51]; // с учетом '\0'
char HouseNum[6];
unsigned short MaxFloorNum;
unsigned short MaxFlatNum;
bool Parking;
};
void OutAddress(HOUSE) ;
int main()
{
HOUSE MyHouse;
MyHouse.RegNum = 524;
strcpy(MyHouse.Street, "ул. Гоголя") ;
strcpy(MyHouse.HouseNum, "2-а") ;
MyHouse.MaxFloorNum = 7;
MyHouse.MaxFlatNum = 84;
MyHouse.Parking = true;
OutAddress(MyHouse) ;
return 0;
}

void OutAddress(HOUSE house)
{
cout << house.Street << ", ";
cout << house.HouseNum << "\n";
}
```

Вызов функции OutAddress(MvHouse) передает в тело сформированную структуру, доступ к членам которой осуществляется в соответствии с описанным выше правилом, через символ «точка» (.).

Теперь рассмотрим функцию, возвращающую структуру:

```
struct point makepoint (int x,int y) //makepoint – формирует точку по компонентам x и y
{
   struct point temp;
   temp.x = x;
   temp.y = y;
   return temp;
}
```

Результат работы этой функции может быть сохранен в специальной переменной и выведен на экран:

```
struct point buf;
buf=makepoint(10,40);
cout << “X=” << buf.x << “  Y=” << buf.y;
```

После выполнения этого фрагмента на экран будут выведены два числа: 10 и 40.

Задания к работе

1. Изучите примеры, приведенные в теоретической части.
2. Напишите программу, которая демонстрирует использование объявления типа структура на примере задания времени Time. Структура должна включать такие данные: Hour – часы, Min — минуты, Sec - секунды. Из примеры должно быть ясно как задаются параметры времени Time и как извлекаются (вывод на экран). Дайте пояснения к ней.
3. Определите структуру для представления записи информации о сданных студентом экзаменах (имя и фамилия студента, число экзаменов, массив с названием экзаменационных дисциплин и массив полученных оценок). Определите функции для задания имени и фамилии студента, а также для задания результатов экзаменов. Напишите программу, которая демонстрирует использование разработанной структуры и функций.
4. Составьте отчет, в который включите результаты по пунктам 1-3. Приведите в отчет код программ и скриншоты результатов их работы.