Лабораторная работа 3. Структуры

Простая структура

Мы начнем рассмотрение со структуры, содержащей три поля, два из которых имеют целый тип и одно поле — вещественный тип. Эта структура предназначена для хранения информации о комплектующих деталях изделий, выпускаемых фабрикой. Компания производит несколько типов изделий, поэтому номер модели изделия включен в структуру как первое из полей. Номер самой детали представлен вторым полем, а ее стоимость — последним, третьим полем. В программе PARTS определяется структура part и переменная part1 типа part. Затем полям переменной part присваиваются значения и выводятся на экран.

```
// parts.cpp
// структура для хранения информации о деталях изделий
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////
struct part
      // объявление структуры
{
      int modelnumber; // номер модели изделия
      int partnumber; // номер детали
      float cost; // стоимость детали
};
////////////////////////////////////////////////////////
int main()
{
      part part1; // определение структурной переменной
      part1.modelnumber = 6244; // инициализация полей
      part1.partnumber = 373;   // переменной part1
      part1.cost = 217.55F;
      // вывод значений полей на экран
      cout << "Модель " << part1.modelnumber;
      cout << ", деталь " << part1.partnumber;
      cout << ", стоимость $" << part1.cost << endl;
      return 0;
}
```

Определение структуры

```
struct part // объявление структуры
{
      int modelnumber; // номер модели изделия
      int partnumber; // номер детали
      float cost; // стоимость детали
};
```

Определение структурной переменной

```
part part1;
```

Доступ к полям структуры

```
part1.modelnumber = 6244;
```

Инициализация полей структуры

```
// partinit.cpp
// инициализация структурных переменных
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////
struct part // объявление структуры

{
      int modelnumber; // номер модели изделия
      int partnumber; // номер детали
      float cost;  // стоимость детали
};
////////////////////////////////////////////////////////
int main()
{
      // инициализация переменной
      part part1 = {6244, 373, 217.55F};
      part part2;  // объявление переменной
      // вывод полей первой переменной
      cout << "Модель " << part1.modelnumber;
      cout << ", деталь " << part1.partnumber;
      cout << ", стоимость $" << part1.cost << endl;
      part2 = part1; // присваивание структурных переменных
 // вывод полей второй переменной
      cout << "Модель " << part2.modelnumber;
      cout << ", деталь " << part2.partnumber;
      cout << ", стоимость $" << part2.cost << endl;
      return 0;
}
```

В приведенной программе определены две переменные типа part: part1 и part2. После того как поля переменной part1 инициализированы, происходит вывод их значений на экран, затем значение переменной part1 присваивается переменной part2, и значения ее полей также выводятся на экран.

Результат работы программы выглядит следующим образом:

Модель 6244, деталь 373, стоимость $217.55

Пример применения структур

Рассмотрим теперь, каким образом можно применить способность структуры объединять данные различных типов. Возможно, вы знаете, что в английской системе мер основными единицами измерения длины служат фут и дюйм, причем один фут равен 12 дюймам. Расстояние, равное 15 футам и 8 дюймам, записывается как 15'-8". Дефис в данной записи не означает знак «минус», а служит для разделения значений футов и дюймов. Иллюстрация к английской системе мер приведена на рис. 4.5.

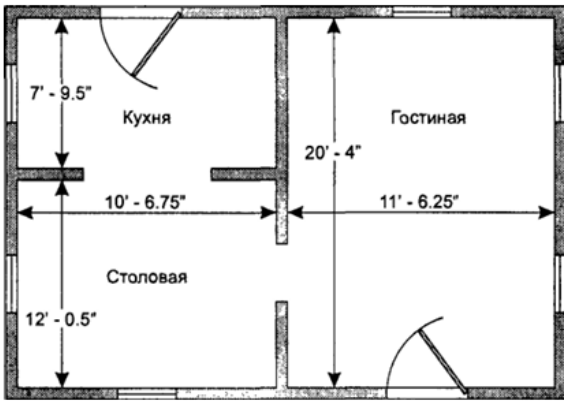


Рис. 4.5. Английская система мер

Предположим, вам необходимо создать инженерный проект с использованием английской системы мер. Было бы удобно представлять длины в виде двух чисел, одно из которых равнялось бы количеству футов, а другое — количеству дюймов. В следующем примере, ENGLSTRC, такая идея реализована с помощью структуры. Мы покажем, как можно при помощи типа Distance складывать две длины.

```
// englstrc.cpp
// английская система мер, реализованная с помощью структуры
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////////
struct Distance // длина в английской системе
{
	int feet;
	float inches;
};
////////////////////////////////////////////////////////////
int main()
{
	Distance d1, d3; // определения двух переменных
	Distance d2 = { 11, 6.25 }; // определение с инициализацией
	// ввод полей переменной d1
	cout << "\nВведите число футов: "; cin >> d1.feet;
	cout << "Введите число дюймов: "; cin >> d1.inches;
	// сложение длин d1 и d2 и присвоение результата d3
	d3.inches = d1.inches + d2.inches;  // сложение дюймов
	d3.feet = 0; // с возможным заемом
	if (d3.inches >= 12.0) // если сумма больше 12.0,
	{
		d3.inches -= 12.0; // то уменьшаем число дюймов на 12.0 и
		d3.feet++; // увеличиваем число футов на 1
	}
	d3.feet += d1.feet + d2.feet; // сложение футов
	// вывод всех длин на экран
	cout << d1.feet << "\'-" << d1.inches << "\" + ";
	cout << d2.feet << "\'-" << d2.inches << "\" = ";
	cout << d3.feet << "\'-" << d3.inches << "\"\n";
	return 0;
}
```

Вложенные структуры

Структуры допускают вложенность, то есть использование структурной переменной в качестве поля какой-либо другой структуры. Внесем дополнения в программу ENGLSTRC, которые позволят это продемонстрировать. Пусть нам необходимо хранить данные о размерах комнаты, то есть ее ширину и длину. Поскольку мы работаем с английской системой мер, нам придется хранить эти величины с использованием типа Distance:

```
// englarea.cpp
// использование вложенных структур
#include <iostream>
using namespace std;
////////////////////////////////////////////////////////
struct Distance // длина в английской системе
{
      int feet; // футы
      float inches; // дюймы
};
////////////////////////////////////////////////////////
struct Room // размеры прямоугольной комнаты
{
      Distance length; // длина
      Distance width; // ширина
};
////////////////////////////////////////////////////////
int main()
{
      Room dining; // переменная dining типа Room
      dining.length.feet = 13; // задание параметров комнаты
      dining.length.inches = 6.5;
      dining.width.feet = 10;
      dining.width.inches = 0.0;
      // преобразование длины и ширины в вещественный формат
      float l = dining.length.feet + dining.length.inches / 12;
      float w = dining.width.feet + dining.width.inches / 12;
      // вычисление площади комнаты и вывод на экран
      cout << "Площадь комнаты равна " << l * w
            << " квадратных футов\n";
      return 0;
}
```

Пример карточной игры

Давайте рассмотрим еще один пример. В нем с помощью структуры будет создана модель игральной карты. Эта программа имитирует действия уличного игрока. Игрок показывает вам три карты, перемешивает их и раскладывает. Если вам удастся угадать, в какой последовательности разложены карты, то вы выиграли. Все происходит у вас на виду, но игрок мешает карты настолько быстро и умело, что вы путаетесь в расположении карт и, как правило, проигрываете свою ставку.

Структура, хранящая информацию об игральной карте, выглядит так:

```
struct card

{

int number;

int suit;

};
```

Эта структура содержит два поля, number и suit, предназначенных для хранения соответственно достоинства карты и ее масти. Достоинство карты определяется числом от 2 до 14, где числа 11, 12, 13 и 14 означают соответственно валета, даму, короля и туза. Масть карты — число от 0 до 3, где 0 означает трефовую масть, 1 — бубновую, 2 — червовую, 3 — пиковую. Листинг программы CARDS выглядит следующим образом:

// cards.cpp

// представление игральных карт при помощи структуры

```
#include <iostream>
using namespace std;
const int clubs = 0;
const int diamonds = 1;
const int hearts = 2;
const int spades = 3;
const int jack = 11;
const int queen = 12;
const int king = 13;
const int ace = 14;
// масти
// достоинства карт
////////////////////////////////////////////////////////
struct card
{
	int number;
	int suit;
};
// достоинство
// масть
////////////////////////////////////////////////////////
int main()
{
	card temp, chosen, prize;
	int position;
	card card1 = { 7, clubs };
	cout << "Карта 1: 7 треф\n";
	// три карты
	// инициализация карты 1
	card card2 = { jack, hearts };  // инициализация карты 2
	cout << "карта 2: валет червей\n";
	card card3 = { ace, spades };
	cout << "Карта 3: туз пик\n ";
	prize = card3;
	// инициализация карты 3
	// запоминание карты 3
	cout << "Меняем местами карту 1 и карту 3...\n";
	temp = card3; card3 = card1; card1 = temp;
	cout << "Меняем местами карту 2 и карту 3...\n";
	temp = card3; card3 = card2; card2 = temp;
	cout << "Меняем местами карту 1 и карту 2...\n";
	temp = card2; card2 = card1; card1 = temp;
	cout << "На какой позиции (1, 2 или 3) теперь туз пик? ";
	cin >> position;
	switch (position)
	{
	case 1: chosen = card1; break;
	case 2: chosen = card2; break;
	case 3: chosen = card3; break;
	}
	if (chosen.number == prize.number && // сравнение карт
		chosen.suit == prize.suit)
		cout << "Правильно! Вы выиграли!\n";
	else
		cout << "Неверно. Вы проиграли.\n";
	return 0;
}
```

Вот пример возможного взаимодействия с программой:
Карта 1: 7 треф
Карта 2: валет червей
Карта 3: туз пик
Меняем местами карту 1 и карту 3
Меняем местами карту 2 и карту 3
Меняем местами карту 1 и карту 2
На какой позиции (1.2 или 3) теперь туз пик? 3
Неверно. Вы проиграли.

Перечисления

Дни недели

Перечисления используются в тех случаях, когда переменные создаваемого типа могут принимать заранее известное конечное (и, как правило, небольшое) множество значений. Приведем простой пример, DAYENUM, в котором перечисления используются для создания типа данных, хранящего дни недели.

```
// dayenum.cpp
// применение перечислений
#include <iostream>
using namespace std;
// объявление перечисляемого типа
enum days_of_week { Sun, Mon, Tue, Wed, Thu, Fri, Sat };
int main()
{
	days_of_week day1, day2;
	day1 = Mon;
	day2 = Thu;
	int diff = day2 - day1;
	// определения переменных,
	// хранящих дни недели
	// инициализация переменных
	// арифметическая операция
	cout << "Разница в днях: " << diff << endl;
	if (day1 < day2)
		// сравнение
		cout << "day1 наступит раньше, чем day2\n";
	return 0;
}
```

Перечисления и программа подсчета числа слов

Наша следующая программа подсчитывает количество слов в предложении, - вводимом с клавиатуры пользователем. Структура работы программы показана на рис. 4.10.



Рис. 4.10. Работа программы WDCOUNT

В отличие от примера CHCOUNT, здесь мы уже не будем сводить подсчет слов к подсчету введенных пробелов (хотя наша программа все еще не будет обрабатывать табуляции и прочие разделяющие символы). Мы используем перечисляемый тип данных с двумя значениями. Вот листинг программы WDCOUNT:

```
// wdcount.cpp
// подсчет числа слов в предложении с использованием перечислений
#include <iostream>
using namespace std;
#include <conio.h>              // для getche()
enum itsaWord { NO, YES };      // NO = 0, YES = 1
int main()
{
    itsaWord isWord = NO;        // isWord равно YES, когда
    // вводится слово, и NO, когда вводятся пробелы
    char ch = 'a';               // считывание символа с клавиатуры
    int wordcount = 0;           // число слов
    cout << "Введите предложение: \n";
    do {
        ch = _getche();             // ввод символа
        if (ch == ' ' || ch == '\r')// если введен пробел,
        {
            if (isWord == YES)        // а до этого вводилось слово,
            {                       // значит, слово закончилось
                wordcount++;            // учет слова
                isWord = NO;            // сброс флага
            }
        }                           // в противном случае
        else                          // ввод слова продолжается
            if (isWord == NO)         // если начался ввод слова,
                isWord = YES;           // то устанавливаем флаг
    } while (ch != '\r');         // выход по нажатию Enter
    cout << "\n---Число слов: " << wordcount << "---\n";
    return 0;
}
```

Примеры перечисляемых типов

```
enum months { Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec };
// месяцы года
enum switch { off, on };  // переключатель
enum meridian { am, pm }; // меридиан
enum chess { pawn, knight, bishop, rook, queen, king };
// шахматные фигуры
enum coins { penny, nickel, dime, quarter, half - dollar, dollar };
// монеты
```

Упражнения

1. Номер телефона, например (212) 767—8900, можно условно разделить на три части: код города (212), номер телефонной станции (767) и номер абонента (8900). Напишите программу с использованием структуры, позволяющую раздельно хранить эти три части телефонного номера. Назовите структуру phone. Создайте две структурные переменные типа phone. Инициализацию одной из них произведите сами, а значения для другой запросите с клавиатуры. Затем выведите содержимое обеих переменных на экран. Результат работы программы должен выглядеть приблизительно так:
   Введите код города, номер станции и номер абонента:

415 555 1212
Мой номер (212) 767-8900
Ваш номер (415) 555-1212

2. Расположение точки на плоскости можно задать с помощью двух координат: x и y. Например, если точка имеет координаты (4, 5), то это значит, что она отстоит на 4 единицы справа от вертикальной оси и на 5 единиц вверх от горизонтальной оси. Сумма двух точек определяется как точка, имеющая координаты, равные сумме соответствующих координат слагаемых. Напишите программу, использующую для интерпретации точки на плоскости структуру с названием point. Определите три переменные типа point, и две из них инициализируйте с помощью значений, вводимых с клавиатуры. Затем присвойте третьей переменной значение суммы первых двух переменных и выведите результат на экран.

   Результат работы программы может выглядеть следующим образом:
   Введите координаты точки p1: 3 4
   Введите координаты точки p2: 5 7
   Координаты точки p1 + p2 равны 8.11
3. Создайте структуру с именем Volume, содержащую три поля типа Distance из примера englstrc, для хранения трех измерений помещения. Определите переменную типа Volume, инициализируйте ее, вычислите объем, занимаемый помещением, и выведите результат на экран. Для подсчета объема переведите каждое из значений типа Distance в значение типа float, равное соответствующей длине в футах и хранимое в отдельной переменной. Затем для вычисления объема следует перемножить три полученные вещественные переменные.
4. Создайте структуру с именем employee, содержащую два поля: номер сотрудника типа int и величину его пособия в долларах типа float. Запросите с клавиатуры данные о трех сотрудниках, сохраните их в трех структурных переменных типа employee и выведите информацию о каждом из сотрудников на экран.
5. Создайте структуру типа date, содержащую три поля типа int: месяц, день и год. Попросите пользователя ввести день, месяц и год в формате 31/12/2002, сохраните введенное значение в структурной переменной, а затем извлеките данные из этой переменной и выведите их на экран в том же формате, в каком они вводились.
6. Как мы говорили, стандартные средства ввода/вывода C++ вместо значений перечисляемых типов данных выводят их внутреннее представление в виде целых чисел. Для того чтобы преодолеть это ограничение, вы можете использовать конструкцию switch, с помощью которой устанавливается соответствие между значением переменной перечисляемого типа и ее внутренним представлением. Пусть,

например, в программе определен перечисляемый тип данных etype, отражающий должность сотрудника:

```
enum etype { laborer, secretary, manager, accountant, executive, researcher };
```

Напишите программу, которая сначала по первой букве должности, введенной пользователем, определяет соответствующее значение переменной, помещает это значение в переменную типа etype, а затем выводит полностью название должности, первую букву которой ввел пользователь. Взаимодействие программы с пользователем может выглядеть следующим образом:

Введите первую букву должности (laborer, secretary, manager, accountant, executive, researcher); a
полное название должности: accountant

[Возможно, вам понадобится два ветвления switch: одно — для ввода значения, другое — для вывода].

7. Добавьте поля типа enum etype (см. упражнение 6) и struct date (см. упраж- нение 5) в структуру employee из упражнения 4. Организуйте программу таким образом, чтобы пользователь вводил 4 пункта данных о каждом из трех сотрудников: его номер, величину зарплаты, его должность и дату принятия на работу. Программа должна хранить введенные значения в трех переменных типа employee и выводить их содержимое на экран.
8. Вернитесь к упражнению 9 лабораторной работы 2 «Основы программирования на C++». В этом упражнении требуется написать программу, которая хранит значения двух дробей в виде числителя и знаменателя, а затем складывает эти дроби согласно арифметическому правилу. Измените эту программу так, чтобы значения дробей хранились в структуре fraction, состоящей из двух полей типа int, предназначенных для хранения числителя и знаменателя. Все значения дробей должны храниться в переменных типа fraction.
9. Создайте структуру с именем time. Три ее поля, имеющие тип int, будут называться hours, minutes и seconds. Напишите программу, которая просит пользователя ввести время в формате часы, минуты, секунды. Можно запрашивать на ввод как три значения сразу, так и выводить для каждой величины отдельное приглашение. Программа должна хранить время в структурной переменной типа time и выводить количество секунд в веденном времени, определяемое следующим образом:

long totalsecs = t1.hours * 3600 + t1.minutes * 60 + t1.seconds