Министерство образования Республики Беларусь

ПОЛОЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра технологий программирования

**Методические указания  
 к лабораторной работе № 2.4  
 по курсу «Основы алгоритмизации   
и программирования»**

«Структуры. Перечисления»

Преподаватель: Магеров Владимир Владимирович

Составитель: Войтехович   
Агния Витольдовна

Полоцк, 2017

# **ЦЕЛЬ РАБОТЫ**

Научиться работать с векторными данными языка С++: структурами и перечислениями. Изучить правила объявления структур и перечислений, обращение к их полям, построение вложенных векторных типов данных.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

# 1 Структуры

**Структура** — это совокупность переменных, объединенных под одним именем. С помощью структур удобно размещать в смежных полях связанные между собой элементы информации. Объявление структуры создает шаблон, который можно использовать для создания ее объектов (то есть экземпляров этой структуры). Переменные, из которых состоит структура, называются членами. (Члены структуры еще называются полями.)

Как правило, члены структуры связаны друг с другом по смыслу. Например, элемент списка рассылки, состоящий из имени и адреса логично представить в виде структуры. В следующем фрагменте кода показано, как объявить структуру, в которой определены поля имени и адреса. Ключевое слово **struct** сообщает компилятору, что объявляется (еще говорят, "декларируется") структура.

struct ADDR

{

char name[30];

char street[40];

char city[20];

char state[3];

int zip;

};

Обратите внимание, что объявление завершается точкой с запятой, потому что объявление структуры является оператором. Кроме того, тег структуры **ADDR** идентифицирует эту конкретную структуру данных и является спецификатором ее типа.

В данном случае на самом деле никакая переменная не создается. Всего лишь определяется вид данных. Когда вы объявляете структуру, то определяете агрегатный тип, а не переменную. И пока вы не объявите переменную этого типа, то существовать она не будет. Чтобы объявить переменную (то есть физический объект) типа **ADDR**, напишите следующее:

ADDR addr\_info;

В этом операторе объявлена переменная типа **ADDR**, которая называется **addr\_info**. Таким образом, **ADDR** описывает вид структуры (ее тип), a **addr\_info** является экземпляром (объектом) этой структуры.

Когда объявляется переменная-структура, компилятор автоматически выделяет количество памяти, достаточное, чтобы разместить все ее члены. На рисунке 1.1 показано, как **addr\_info** размещена в памяти; в данном случае предполагается, что целые переменные типа int занимают по 4 байта.

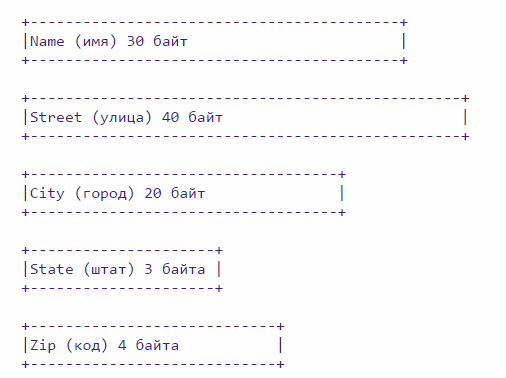


Рисунок 1.1 – Расположение в памяти структуры **addr\_info**

Одновременно с объявлением структуры можно объявить одну или несколько переменных. Например,

struct ADDR {

char name[30];

char street[40];

char city[20];

char state[3];

int zip;

} addr\_info, binfo, cinfo;

определяет тип структуры, называемый **ADDR**, и объявляет переменные этого типа **addr\_info, binfo** и **cinfo**. Важно понимать, что каждая переменная-структура содержит собственные копии членов структуры. Например, поле **zip** в **binfo** отличается от поля **zip** в **cinfo**. Изменения в **zip** из **binfo** не повлияют на содержимое поля **zip**, находящегося в **cinfo**.

Если нужна только одна переменная-структура, то тег структуры является лишним. В этом случае наш пример объявления можно переписать следующим образом:

struct {

char name[30];

char street[40];

char city[20];

char state[3];

unsigned long int zip;

} addr\_info;

В этом случае объявляется одна переменная с именем **addr\_info**, причем ее поля указаны в структуре, которая предшествует этому имени.

Общий вид объявления структуры такой:

struct тег {

тип имя-члена;

тип имя-члена;

тип имя-члена;

.

.

.

} переменные-структуры;

причем тег или переменные-структуры могут быть пропущены, но только не оба одновременно.

## 1.1 Доступ к членам структуры

Доступ к отдельным членам структуры осуществляется с помощью оператора **.** (который обычно называют оператором точка или оператором доступа к члену структуры). Например, в следующем выражении полю **zip** в уже объявленной переменной-структуре **addr\_info** присваивается значение ZIP-кода, равное ***12345***:

addr\_info.zip = 12345;

Этот отдельный член определяется именем объекта (в данном случае **addr\_info**), за которым следует точка, а затем именем самого этого члена (в данном случае **zip**). В общем виде использование оператора точка для доступа к члену структуры выглядит таким образом:

имя-объекта.имя-члена

Поэтому, чтобы вывести ZIP-код на экран, напишите следующее:

cout << addr\_info.zip;

Будет выведен ZIP-код, который находится в члене **zip** переменной-структуры **addr\_infо**.

Точно так же в вызове **gets()** можно использовать массив символов **addr\_infо.name**:

cin.gets(addr\_info.name);

Таким образом, в начало **name** передается указатель на символьную строку.

Так как **name** является массивом символов, то чтобы получить доступ к отдельным символам в массиве **addr\_info.name**, можно использовать индексы вместе с **name**. Например, с помощью следующего кода можно посимвольно вывести на экран содержимое **addr\_info.name**:

for(t=0; addr\_info.name[t]; ++t)

putchar(addr\_info.name[t]);

Обратите внимание, что индексируется именно **name** (а не **addr\_info**). Помните, что **addr\_info** — это имя всего объекта-структуры, a **name** — имя элемента этой структуры. Таким образом, если требуется индексировать элемент структуры, то индекс необходимо указывать после имени этого элемента.

## 1.2 Присваивание структур

Информация, которая находится в одной структуре, может быть присвоена другой структуре того же типа при помощи единственного оператора присваивания. Нет необходимости присваивать значения каждого члена в отдельности. Как выполняется присваивание структур, показывает следующая программа:

#include <iostream>

using namespace std;

int main(void)

{

struct {

int a;

int b;

} x, y;

x.a = 10;

y = x; /\* присваивание одной структуры другой \*/

cout << y.a;

return 0;

}

После присвоения в **y.a** будет храниться значение ***10***.

# 2 Массивы структур

Структуры часто образуют массивы. Чтобы объявить массив структур, вначале необходимо определить структуру (то есть определить агрегатный тип данных), а затем объявить переменную массива этого же типа. Например, чтобы объявить 100-элементный массив структур типа **ADDR**, который был определен ранее, напишите следующее:

ADDR addr\_list[100];

Это выражение создаст 100 наборов переменных, каждый из которых организован так, как определено в структуре **ADDR**.

Чтобы получить доступ к определенной структуре, указывайте имя массива с индексом. Например, чтобы вывести ZIP-код из третьей структуры, напишите следующее:

cout << addr\_list[2].zip;

Как и в других массивах переменных, в массивах структур индексирование начинается с 0.

**ДЛЯ СПРАВКИ:** чтобы указать определенную структуру, находящуюся в массиве структур, необходимо указать имя этого массива с определенным индексом. А если нужно указать индекс определенного элемента в структуре, то необходимо указать индекс этого элемента. Таким образом, в результате выполнения следующего выражения первому символу члена **name**, находящегося в третьей структуре из **addr\_list**, присваивается значение ***'X'***.

addr\_list[2].name[0] = 'X';

# 3 Передача структур функциям

## 3.1 Передача членов структур функциям

При передаче функции члена структуры передается его значение, притом не играет роли то, что значение берется из члена структуры. Проанализируйте, например, следующую структуру:

struct FRED

{

char x;

int y;

float z;

char s[10];

} mike;

Например, обратите внимание, каким образом каждый член этой структуры передается функции:

func(mike.x); /\* передается символьное значение x \*/

func2(mike.y); /\* передается целое значение y \*/

func3(mike.z); /\* передается значение с плавающей точкой z \*/

func4(mike.s); /\* передается адрес строки s \*/

func(mike.s[2]); /\* передается символьное значение s[2] \*/

В каждом из этих случаев функции передается значение определенного элемента, и здесь не имеет значения то, что этот элемент является частью какой-либо большей совокупности.

Если же нужно передать адрес отдельного члена структуры, то перед именем структуры должен находиться оператор **&**. Например, чтобы передать адреса членов структуры mike, можно написать следующее:

func(&mike.x); /\* передается адрес символа x \*/

func2(&mike.y); /\* передается адрес целого y \*/

func3(&mike.z); /\* передается адрес члена z с плавающей точкой \*/

func4(mike.s); /\* передается адрес строки s \*/

func(&mike.s[2]); /\* передается адрес символа в s[2] \*/

Обратите внимание, что оператор **&** стоит непосредственно перед именем структуры, а не перед именем отдельного члена. И еще заметьте, что **s** уже обозначает адрес, поэтому **&** не требуется.

## 2.2 Передача целых структур функциям

Когда в качестве аргумента функции используется структура, то для передачи целой структуры используется обычный способ вызова по значению. Это, конечно, означает, что любые изменения в содержимом параметра внутри функции не отразятся на той структуре, которая передана в качестве аргумента.

При использовании структуры в качестве аргумента надо помнить, что тип аргумента должен соответствовать типу параметра. Например, в следующей программе и аргумент **arg**, и параметр **parm** объявляются с одним и тем же типом структуры.

#include <iostream>

using namespace std;

/\* Определение типа структуры. \*/

struct struct\_type {

int a, b;

char ch;

} ;

void f1(struct\_type parm);

int main(void)

{

struct\_type arg;

arg.a = 1000;

f1(arg);

return 0;

}

void f1(struct\_type parm)

{

cout << parm.a;

}

Как видно из этой программы, при объявлении параметров, являющихся структурами, объявление типа структуры должно быть глобальным, чтобы структурный тип можно было использовать во всей программе. Например, если бы **struct\_type** был бы объявлен внутри **main()**, то этот тип не был бы виден в **f1()**.

Как уже говорилось, при передаче структуры тип аргумента должен совпадать с типом параметра. Для аргумента и параметра недостаточно просто быть физически похожими; должны совпадать даже имена их типов. Например, следующая версия предыдущей программы неправильная и компилироваться не будет. Дело в том, что имя типа для аргумента, используемого при вызове функции **f1()**, отличается от имени типа ее параметра.

/\* Эта программа неправильная и при компиляции будут

обнаружены ошибки. \*/

#include <iostream>

using namespace std;

/\* Определение типа структур. \*/

struct struct\_type {

int a, b;

char ch;

} ;

/\* Определение структуры, похожей на struct\_type,

но сдругими именами. \*/

struct struct\_type2 {

int a, b;

char ch;

} ;

void f1(struct\_type2 parm);

int main(void)

{

struct\_type arg;

arg.a = 1000;

f1(arg); /\* несовпадение типов \*/

return 0;

}

void f1(struct\_type2 parm)

{

printf("%d", parm.a);

}

# 4 Указатели на структуры

В языке С указатели на структуры также официально признаны, как и указатели на любой другой вид объектов. Однако указатели на структуры имеют некоторые особенности, о которых и пойдет речь.

## 4.1 Объявление указателя на структуру

Как и другие указатели, указатель на структуру объявляется с помощью звездочки **\***, которую помещают перед именем переменной структуры. Например, для ранее определенной структуры **ADDR** следующее выражение объявляет **addr\_pointer** указателем на данные этого типа (то есть на данные типа **ADDR**):

ADDR \*addr\_pointer;

## 4.2 Использование указателей на структуры

Указатели на структуры используются главным образом в двух случаях: когда структура передается функции с помощью вызова по ссылке, и когда создаются связанные друг с другом списки и другие структуры с динамическими данными, работающие на основе динамического размещения. В этой главе рассматривается первый случай.

У такого способа, как передача любых (кроме самых простых) структур функциям, имеется один большой недостаток: при выполнении вызова функции, чтобы поместить структуру в стек, необходимы существенные ресурсы. (Вспомните, что аргументы передаются функциям через стек.) Впрочем, для простых структур с несколькими членами эти ресурсы являются не такими уж большими. Но если в структуре имеется большое количество членов или некоторые члены сами являются массивами, то при передаче структур функциям производительность может упасть до недопустимо низкого уровня. Как же решить эту проблему? Надо передавать не саму структуру, а указатель на нее.

Когда функции передается указатель на структуру, то в стек попадает только адрес структуры. В результате вызовы функции выполняются очень быстро. В некоторых случаях этот способ имеет еще и второе преимущество: передача указателя позволяет функции модифицировать содержимое структуры, используемой в качестве аргумента.

Чтобы получить адрес переменной-структуры, необходимо перед ее именем поместить оператор **&**. Например, в следующем фрагменте кода

struct BAL {

float balance;

char name[80];

} person;

BAL \*p; /\* объявление указателя на структуру \*/

//адрес структуры person можно присвоить указателю p:

p = &person;

Чтобы с помощью указателя на структуру получить доступ к ее членам, необходимо использовать оператор стрелка **->**. Вот, например, как можно сослаться на поле **balance**:

p->balance

Оператор **->**, который обычно называют оператором стрелки, состоит из знака "минус", за которым следует знак "больше". Стрелка применяется вместо оператора точки тогда, когда для доступа к члену структуры используется указатель на структуру.

Чтобы увидеть, как можно использовать указатель на структуру, проанализируйте следующую простую программу, которая имитирует таймер, выводящий значения часов, минут и секунд:

/\* Программа-имитатор таймера. \*/

#include <iostream>

using namespace std;

#define DELAY 128000

struct MY\_TIME {

int hours;

int minutes;

int seconds;

} ;

void display(MY\_TIME \*t);

void update(MY\_TIME \*t);

void delay(void);

int main(void)

{

MY\_TIME systime;

systime.hours = 0;

systime.minutes = 0;

systime.seconds = 0;

for(;;) {

update(&systime);

display(&systime);

}

return 0;

}

void update(MY\_TIME \*t)

{

t->seconds++;

if(t->seconds==60) {

t->seconds = 0;

t->minutes++;

}

if(t->minutes==60) {

t->minutes = 0;

t->hours++;

}

if(t->hours==24) t->hours = 0;

delay();

}

void display(MY\_TIME \*t)

{

printf("%02d:", t->hours);

printf("%02d:", t->minutes);

printf("%02d\n", t->seconds);

}

void delay(void)

{

long int t;

/\* если надо, можно изменять константу DELAY (задержка) \*/

for(t=1; t<DELAY; ++t) ;

}

Эту программу можно настраивать, меняя определение DELAY.

В этой программе объявлена глобальная структура **my\_time**, но при этом не объявлены никакие другие переменные программы. Внутри же **main()** объявлена структура **systime** и она инициализируется значением ***00:00:00***. Это значит, что **systime** непосредственно видна только в функции **main().**

Функциям **update()** (которая изменяет значения времени) и **display()** (которая выводит эти значения) передается адрес структуры **systime**. Аргументы в обеих функциях объявляются как указатель на структуру **my\_time**.

Внутри **update()** и **display()** доступ к каждому члену **systime** осуществляется с помощью указателя. Так как функция **update()** принимает указатель на структуру **systime**, то она в состоянии обновлять значение этой структуры. Например, необходимо "в полночь", когда значение переменной, в которой хранится количество часов, станет равным 24, сбросить отсчет и снова сделать значение этой переменной равным 0. Для этого в **update()** имеется следующая строка:

if(t->hours==24) t->hours = 0;

Таким образом, компилятору дается указание взять адрес **t** (этот адрес указывает на переменную **systime** из **main()**) и сбросить значение **hours** в нуль.

Помните, что оператор точка используется для доступа к элементам структуры при работе с самой структурой. А когда используется указатель на структуру, то надо применять оператор стрелка.

# 5 Массивы и структуры внутри структур

Членом структуры может быть или простая переменная, например, типа **int** или **double**, или составной (не скалярный) тип. В языке С составными типами являются массивы и структуры. Один составной тип вы уже видели — это символьные массивы, которые использовались в **ADDR**.

Члены структуры, которые являются массивами, можно считать такими же членами структуры, как и те, что нам известны из предыдущих примеров. Например, проанализируйте следующую структуру:

struct X {

int a[10][10]; /\* массив 10 x 10 из целых значений \*/

float b;

} y;

Целый элемент с индексами 3, 7 из массива **a**, находящегося в структуре **y**, обозначается таким образом:

y.a[3][7]

Когда структура является членом другой структуры, то она называется вложенной. Например, в следующем примере структура **address** вложена в **EMP**:

struct **EMP** {

struct ADDR address; /\* вложенная структура \*/

float wage;

} worker;

Здесь структура была определена как имеющая два члена. Первым является структура типа **ADDR**, в которой находится адрес работника. Второй член — это **wage**, где находятся данные по его зарплате. В следующем фрагменте кода элементу **zip** из **address** присваивается значение ***93456***.

worker.address.zip = 93456;

Как вы видите, в каждой структуре любой член обозначают с помощью тех структур, в которые он вложен — начиная от самых общих и заканчивая той, непосредственно в которой он находится. В соответствии со стандартом С89 структуры могут быть вложенными вплоть до 15-го уровня. А стандарт С99 допускает уровень вложенности до 63-го включительно.

# 6 Перечисления

**Перечисление** — это набор именованных целых констант. Перечисления довольно часто встречаются в повседневной жизни. Вот, например, перечисление, в котором приведены названия монет, используемых в Соединенных Штатах:

penny (пенни, монета в один цент), nickel (никель, монета в пять центов), dime (монета в 10 центов), quarter (25 центов, четверть доллара), half-dollar (полдоллара), dollar (доллар)

Перечисления определяются во многом так же, как и структуры; началом объявления перечислимого типа (иногда используется термин перечисляемый тип) служит ключевое слово **enum**. Перечисление в общем виде выглядит так:

enum тег {список перечисления} список переменных;

Здесь тег и список переменных не являются обязательными. (Но хотя бы что-то одно из них должно присутствовать.) Следующий фрагмент кода определяет перечисление с именем **coin (монета)**:

enum coin { penny, nickel, dime, quarter,

half\_dollar, dollar};

Тег перечисления можно использовать для объявления переменных данного перечислимого типа. Вот код, в котором **money (деньги)** объявляется в качестве переменной типа **coin**:

enum coin money;

С учетом этих объявлений совершенно верными являются следующие операторы:

money = dime;

if(money==quarter)

cout << "Денег всего четверть доллара."<< endl;

Главное, что нужно знать для понимания перечислений — каждый их элемент (их также называют перечислителями и идентификаторами) представляет целое число. В таком виде элементы перечислений можно применять везде, где используются целые числа. Каждому элементу дается значение, на единицу большее, чем у его предшественника. Первый элемент перечисления имеет значение 0. Поэтому, при выполнении кода

cout << penny << " " << dime << endl;

на экран будет выведено 0 2.

Однако для одного или более элементов можно указать значение, используемое как инициализатор. Для этого после перечислителя надо поставить знак равенства, а затем — целое значение. Перечислителям, которые идут после инициализатора, присваиваются значения, большие предшествующего. Например, следующий код присваивает **quarter** значение 100:

enum coin { penny, nickel, dime, quarter=100,

half\_dollar, dollar};

вот какие значения появились у этих элементов:

penny 0

nickel 1

dime 2

quarter 100

half\_dollar 101

dollar 102

Относительно перечислений есть одно распространенное, но ошибочное мнение. Оно состоит в том, что их элементы можно непосредственно вводить и выводить. Это не так. Например, следующий фрагмент кода не будет выполняться так, как того ожидают многие неопытные программисты:

/\* этот код работать не будет \*/

money = dollar;

puts(money);

Здесь **dollar** — это имя для значения целого типа; это не строка. Таким образом, попытка вывести **money** в виде строки по существу обречена. По той же причине для достижения нужных результатов не годится и такой код:

/\* этот код не правильный \*/

strcpy(money, "dime");

То есть строка, содержащая имя элемента, автоматически в этот перечислитель не превратится.

На самом же деле, создавать код для ввода и вывода элементов перечислений — это довольно-таки скучное занятие (но его можно избежать лишь тогда, когда будет достаточно именно целых значений этих перечислителей). Например, чтобы выводить название монеты, вид которой находится в **money**, потребуется следующий код:

switch(money) {

case penny: cout << "пенни" << endl;

break;

case nickel: cout << "никель"<< endl;

break;

case dime: cout << "монета в 10 центов"<< endl;

break;

case quarter: cout << "четверть доллара"<< endl;

break;

case half\_dollar: cout << "полдоллара"<< endl;

break;

case dollar: cout << "доллар"<< endl;

}

Иногда можно объявить строчный массив и использовать значение перечисления как индекс при переводе этого значения в соответствующую строку. Например, следующий код также выводит нужную строку:

char name[][12]={

"пенни",

"никель",

"монета в 10 центов",

"четверть доллара",

"полдоллара",

"доллар"

};

puts (name[money]);

Конечно, он будет работать только тогда, когда не инициализирован ни один из элементов перечисления, так как строчный массив должен иметь индекс, который начинается с 0 и возрастает каждый раз на 1.

Так как при операциях ввода/вывода необходимо специально заботиться о преобразовании перечислений в их строчный эквивалент, который можно легко прочитать, то перечисления полезнее всего именно в тех процедурах, где такие преобразования не нужны. Например, перечисления часто применяются, чтобы определить таблицы соответствия символов в компиляторах.

ЗАДАНИЯ ДЛЯ ЛАБОРАТОРНОЙ РАБОТЫ №2\_4

**Задание 1.** Создать структуру для описания предметной области **вашей курсовой работы**.

**Задание 2.** В функции **main()** создать указатель на структуру из   
**задания 1.** Выделить динамическую память под эту структуру.

**Задание 3.** Написать функцию, которая будет считывать данные с клавиатуры и помещать их в структуру. Выделить дополнительную динамическую память в указателе на структуру под новую запись.

**Задание 4.** Написать функцию, которая будет печатать данные из структуры в формате «поле: значение\n».

**Задание 5.** Написать функцию, которая будет печатать данные из структуры в табличном формате.

**\*Дополнительное задание.** Организовать меню для вызова функций из **заданий 3-5**.