**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Информационных систем**

отчет

**по лабораторной работе №1**

**по дисциплине «Программирование»**

Тема: Типы данных, определяемые пользователем

|  |  |  |
| --- | --- | --- |
| Студент гр. 0323 |  | Голубцов В.В. |
| Преподаватель |  | Глущенко А.Г. |

Санкт-Петербург

2021

**Цель работы.**

Изучение и организация структур; получение практических навыков работы со структурами; определение преимуществ и недостатков использования структур.

**Основные теоретические положения.**

Структуры представляют собой группы связанных между собой, как правило, разнотипных переменных, объединенных в единый объект, в отличие от массива, все элементы которого однотипны. В языке C++ структура является видом класса и обладает всеми его свойствами.  
Примечание  
Класс - в ООП представляет собой шаблон для создания объектов, обеспечивающий начальные значения состояний: инициализация полей-переменных и реализация поведения функций или методов. ООП в данном курсе не рассматривается, но иногда будут упоминаться понятия, связанные с ним.

Каждая входящая в структуру переменная называется членом (полем, элементом) структуры и описывается типом данных и именем. Поля структуры могут быть любого типа данных. Их количество не лимитировано.

После определения структуры можно её использовать. Для начала можно определить объект структуры - по сути обычную переменную, которая будет представлять описанную ранее структуру.

При описании структуры память для размещения данных не выделяется. Работать с описанной структурой можно только после того, как будет определена переменная (переменные) этого типа данных, только при этом компилятор выделит необходимую память.

В конце определения структуры после закрывающей фигурной скобки идет ее обозначение - в данном случае person. В дальнейшем можно использовать это обозначение для создания переменной структуры.

Еще один способ определить структуру представляет применение препроцессорной директивы **#define**:

В языке С++ отличие от С при определении переменной не нужно использовать слово **struct**.

Любая структурная переменная занимает в памяти определенное положение, характеризующееся конкретным адресом. Для работы с адресами структурных переменных (как и для простых переменных) можно использовать указатели. Указатели на структурные переменные определяются точно так же, как и для обычных переменных. Разыменование указателя (обращение к данным по адресу, хранящемуся в указателе) осуществляется также обычным образом.

Через указатели можно работать с отдельными полями структур. Для доступа к полю структуры через указатель используется оператор ‘->’ («стрелка»), а не «точка».

Структуры можно использовать в качестве параметров функций, как и обычные переменные. Для структур поддерживаются все три механизма передачи данных: по значению, через указатели и по ссылке.

Передачу структур в функции по значению необходимо использовать аккуратно:

Вызов такой функции сопровождается дополнительным расходом памяти для создания локальной переменной **S**и дополнительными затратами времени на физическое копирование данных из аргумента в параметр **S**. Учитывая то, что объем структур может быть очень большим, эти дополнительные затраты вычислительных ресурсов могут быть чрезмерными.

Предпочтительно использование передачи структуры по указателю или ссылке:

Фактической передачи данных в функцию не осуществляется. Дополнительные затраты памяти для создания локальной переменной небольшие – это адрес памяти (4 байта, независимо от размера самой структуры). Вызов такой функции будет происходить быстрее, а расход памяти будет существенно меньше, чем при передаче данных по значению.

Передача по ссылке по эффективности эквивалентна передаче данных через указатель. Однако, поскольку при передаче данных по ссылке все адресные преобразования берет на себя компилятор, существенно упрощается программирование действий со структурами. При использовании ссылочных параметров структурных типов доступ к членам структуры осуществляется обычным способом – с помощью оператора «точка».

Недостатком этих способов является то, что случайные изменения значений полей структуры внутри функции отразятся на значении аргумента после окончания работы функции. Если необходимо предотвратить изменения переданных по адресу аргументов, можно при определении соответствующего параметра объявить его константой (использовать спецификатор const).

Объявление **typedef** не создает типы. Оно создает синонимы для существующих типов или имена для типов, которые могут определяться другими способами. Если имя typedef используется как спецификатор типа, его можно использовать в сочетании с определенными спецификаторами типа (но нельзя использовать с другими спецификаторами). К допустимым модификаторам относятся **const** и **volatile** .

Спецификатор **volatile**  позволяет использовать переменную, значение которой может изменяться неявно в процессе выполнения программы. Например:

Так как переменная **status** явно не изменяется, компилятор в результате оптимизации прочитает её значение только один раз и больше не станет. Но переменная может принять иное значение в результате выполнения другого потока, либо изменить своё значение неявно в прерывании. Чтобы предотвратить подобное используется спецификатор **volatile** :

Имена **typedef** используют то же пространство имен, что и обычные идентификаторы. Поэтому в программе может присутствовать имя **typedef** и идентификатор с тем же именем в локальной области.   
При объявлении в локальной области идентификатора с тем же именем, что и имя **typedef**, или при объявлении члена структуры либо объединения в той же области или во внутренней области обязательно должен указываться спецификатор типа. Следующий пример иллюстрирует это ограничение:

Чтобы повторно использовать имя **FlagType**для идентификатора, члена структуры или члена объединения, необходимо указать тип:

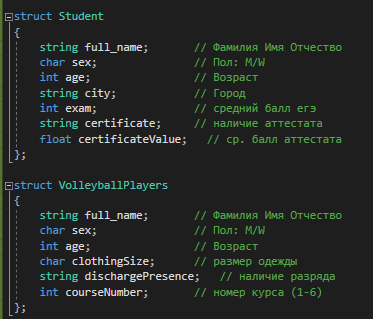
С помощью typedef можно объявить любой тип, включая типы указателей, функций и массивов. Имя typedef для типа указателя на структуру или объединение можно объявить до определения типа структуры или объединения, если только определение находится в той же области видимости, что и объявление.

Имена **typedef** можно использовать, чтобы сделать код более понятным. Все три следующих объявления **signal**задают один и тот же тип, причем в первом объявлении имена **typedef** не используются.

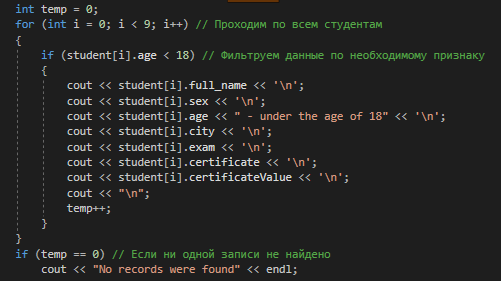
**Экспериментальные результаты.**

Программа разделена на две подпрограммы. В обоих из них алгоритм работы одинков, за исключением условий вывода данных на экран. Изначально данный берутся из текстового файла, преобразуются в структуру и дальше идет работа с ней.

Структуры выглядят следующим образом:

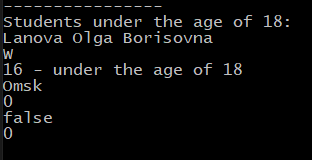


В первой подпрограмме мы должны вывести данные об абитуриентах: младше 18 лет; чей бал ЕГЭ больше 85; иногородних; у кого есть аттестат с отличием.

Принцип следующий: 

**Обработка результатов эксперимента.**

Программа выводит только нужные данные:



**Выводы.**

В ходе данной лабораторной работы я изучил работу с структурами и операциями над ними.

ПРИЛОЖЕНИЕ А

Листинг программы

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

struct Student

{

string full\_name; // Фамилия Имя Отчество

char sex; // Пол: M/W

int age; // Возраст

string city; // Город

int exam; // средний балл егэ

string certificate; // наличие аттестата

float certificateValue; // ср. балл аттестата

};

struct VolleyballPlayers

{

string full\_name; // Фамилия Имя Отчество

char sex; // Пол: M/W

int age; // Возраст

char clothingSize; // размер одежды

string dischargePresence; // наличие разряда

int courseNumber; // номер курса (1-6)

};

void ReadingFile() {

}

int Applicants2()

{

Student \*student = new Student[10];

ifstream database("students.txt");

if (!database.is\_open())

cout << "Error!";

else

{

for (int i = 0; i < 9; i++)

{

database.ignore();

getline(database, student[i].full\_name, '\n');

database >> student[i].sex;

database >> student[i].age;

database.ignore();

getline(database, student[i].city, '\n');

database >> student[i].exam;

database.ignore();

getline(database, student[i].certificate, '\n');

database >> student[i].certificateValue;

}

database.close(); // Закрываем файл

cout << "----------------\n" << "Students under the age of 18:" << endl;

int temp = 0;

for (int i = 0; i < 9; i++) // Проходим по всем студентам

{

if (student[i].age < 18) // Фильтруем данные по необходимому признаку

{

cout << student[i].full\_name << '\n';

cout << student[i].sex << '\n';

cout << student[i].age << " - under the age of 18" << '\n';

cout << student[i].city << '\n';

cout << student[i].exam << '\n';

cout << student[i].certificate << '\n';

cout << student[i].certificateValue << '\n';

cout << "\n";

temp++;

}

}

if (temp == 0) // Если ни одной записи не найдено

cout << "No records were found" << endl;

cout << "----------------\n" << "Students with a GPA greater than 85:" << endl;

temp = 0;

for (int i = 0; i < 9; i++) // Проходим по всем студентам

{

if (student[i].exam > 85) // Фильтруем данные по необходимому признаку

{

cout << student[i].full\_name << '\n';

cout << student[i].sex << '\n';

cout << student[i].age << '\n';

cout << student[i].city << '\n';

cout << student[i].exam << " - GPA greater than 85" << '\n';

cout << student[i].certificate << '\n';

cout << student[i].certificateValue << '\n';

cout << "\n";

temp++;

}

}

if (temp == 0) // Если ни одной записи не найдено

cout << "No records were found" << endl;

cout << "----------------\n" << "Nonresident students:" << endl;

temp = 0;

for (int i = 0; i < 9; i++) // Проходим по всем студентам

{

if (student[i].city != "Saint Petersburg") // Фильтруем данные по необходимому признаку

{

cout << student[i].full\_name << '\n';

cout << student[i].sex << '\n';

cout << student[i].age << '\n';

cout << student[i].city << " - Nonresident students" << '\n';

cout << student[i].exam << '\n';

cout << student[i].certificate << '\n';

cout << student[i].certificateValue << '\n';

cout << "\n";

temp++;

}

}

if (temp == 0) // Если ни одной записи не найдено

cout << "No records were found" << endl;

cout << "----------------\n" << "Students with honors:" << endl;

temp = 0;

for (int i = 0; i < 9; i++) // Проходим по всем студентам

{

if (student[i].certificateValue == 5) // Фильтруем данные по необходимому признаку

{

cout << student[i].full\_name << '\n';

cout << student[i].sex << '\n';

cout << student[i].age << '\n';

cout << student[i].city << '\n';

cout << student[i].exam << '\n';

cout << student[i].certificate << '\n';

cout << student[i].certificateValue << " - Students with honors" << '\n';

cout << "\n";

temp++;

}

}

if (temp == 0) // Если ни одной записи не найдено

cout << "No records were found" << endl;

delete[] student; // Освобождаем динамическую память

}

return 1;

}

int Volleyball5()

{

VolleyballPlayers \*volleyballPlayers = new VolleyballPlayers[10];

ifstream database("volleyballPlayers.txt");

if (!database.is\_open())

cout << "Error!";

else

{

for (int i = 0; i < 10; i++)

{

database.ignore();

getline(database, volleyballPlayers[i].full\_name, '\n');

database >> volleyballPlayers[i].sex;

database >> volleyballPlayers[i].age;

database >> volleyballPlayers[i].clothingSize;

database.ignore();

getline(database, volleyballPlayers[i].dischargePresence, '\n');

database >> volleyballPlayers[i].courseNumber;

}

database.close(); // Закрываем файл

cout << "----------------\n" << "Who has a volleyball category:" << endl;

int temp = 0;

for (int i = 0; i < 10; i++) // Проходим по всем студентам

{

if (volleyballPlayers[i].dischargePresence == "true") // Фильтруем данные по необходимому признаку

{

cout << volleyballPlayers[i].full\_name << '\n';

cout << volleyballPlayers[i].sex << '\n';

cout << volleyballPlayers[i].age << '\n';

cout << volleyballPlayers[i].clothingSize << '\n';

cout << volleyballPlayers[i].dischargePresence << " - volleyball category" << '\n';

cout << volleyballPlayers[i].courseNumber << '\n';

cout << "\n";

temp++;

}

}

if (temp == 0) // Если ни одной записи не найдено

cout << "No records were found" << endl;

cout << "----------------\n" << "Women's team:" << endl;

temp = 0;

for (int i = 0; i < 10; i++) // Проходим по всем студентам

{

if (volleyballPlayers[i].sex == 'W') // Фильтруем данные по необходимому признаку

{

cout << volleyballPlayers[i].full\_name << '\n';

cout << volleyballPlayers[i].sex << " - woman" << '\n';

cout << volleyballPlayers[i].age << '\n';

cout << volleyballPlayers[i].clothingSize << '\n';

cout << volleyballPlayers[i].dischargePresence << '\n';

cout << volleyballPlayers[i].courseNumber << '\n';

cout << "\n";

temp++;

}

}

if (temp == 0) // Если ни одной записи не найдено

cout << "No records were found" << endl;

int courseK = 0;

cout << "Enter course (1-6): ";

cin >> courseK;

cout << "----------------\n" << "Students K course:" << endl;

temp = 0;

for (int i = 0; i < 10; i++) // Проходим по всем студентам

{

if (volleyballPlayers[i].courseNumber == courseK) // Фильтруем данные по необходимому признаку

{

cout << volleyballPlayers[i].full\_name << '\n';

cout << volleyballPlayers[i].sex << '\n';

cout << volleyballPlayers[i].age << '\n';

cout << volleyballPlayers[i].clothingSize << '\n';

cout << volleyballPlayers[i].dischargePresence << '\n';

cout << volleyballPlayers[i].courseNumber << " - Students "<< courseK <<" course:" << '\n';

cout << "\n";

temp++;

}

}

if (temp == 0) // Если ни одной записи не найдено

cout << "No records were found" << endl;

cout << "----------------\n" << "Men with clothing sizes larger than M:" << endl;

temp = 0;

for (int i = 0; i < 10; i++) // Проходим по всем студентам

{

if (volleyballPlayers[i].sex == 'M' && volleyballPlayers[i].clothingSize != 'S' && volleyballPlayers[i].clothingSize != 'M') // Фильтруем данные по необходимому признаку

{

cout << volleyballPlayers[i].full\_name << '\n';

cout << volleyballPlayers[i].sex << '\n';

cout << volleyballPlayers[i].age << '\n';

cout << volleyballPlayers[i].clothingSize << " - clothing sizes larger than M" << '\n';

cout << volleyballPlayers[i].dischargePresence << '\n';

cout << volleyballPlayers[i].courseNumber << '\n';

cout << "\n";

temp++;

}

}

if (temp == 0) // Если ни одной записи не найдено

cout << "No records were found" << endl;

delete[] volleyballPlayers; // Освобождаем динамическую память

}

return 1;

}

int main()

{

int number = 0;

do {

cout << "####################" << endl;

cout << "Select task (1-2):";

cin >> number;

if (number == 1) {

Applicants2(); // задание №2

}

else if (number == 2) {

Volleyball5(); // задание №5

}

else {

cout << "Wrong number";

}

} while (number < 2);

system("pause");

return 0;

}