**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №1**

**по дисциплине «Программирование»**

Тема: Типы данных, определяемые пользователем. Структуры

|  |  |  |
| --- | --- | --- |
| Студент гр. 0324 |  | Виноградов П.Е. |
| 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. Чаще всего ограничиваются тем, как структуры представлены в языке С:

struct [имя\_типа] {

тип\_1 элемент\_1;

тип \_2 элемент\_2;

…

тип\_k элемент\_k;

} [ список\_описателей ];

Описание структуры начинается ключевым словом struct. 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.

Вся эта конструкция является инструкцией языка программирования, поэтому после нее всегда должен ставиться символ ‘;’.

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.

Для инициализации структуры значения ее элементов перечисляют в фигурных скобках в порядке их описания:

struct complex{

float real, im;

} data [2][2] = {

{{1,1}, {2,2}},

{{3,3}, {4,4}}

};

Все поля структурных переменных располагаются в непрерывной области памяти одно за другим. Общий объем памяти, занимаемый структурой, равен

сумме размеров всех полей структуры. Для определения размера структуры следует использовать инструкцию sizeof().

Для того чтобы записать данные в структурную переменную, необходимо каждому полю структуры присвоить определенное значение. Для этого необходимо использовать оператор ‘’ («точка»):

structStack { // Cтек

floatarr[100];

shorttopIndex;

};

…

Stackstack; // Объявляем переменную типа Stack

Stack.arr[0] = 1;

…

При доступе к определенному полю его следует рассматривать как обычную переменную, тип данных которой соответствует типу этого поля. Поля структур могут участвовать в качестве операндов любых выражений, допускающих использование операндов соответствующего типа данных.

Копирование данных из одной структурной переменной в другую осуществляется простой операцией присваивания, независимо от количества полей и размера структуры (это можно делать только в том случае, когда обе переменные одного и того же типа).

В программировании очень часто используются такие конструкции, как массивы структур. Например, сведения о студентах некоторой учебной группы можно хранить в массиве студентов:

t\_StudentGruppa\_N [30];

Был определен 30-элементный массив, каждый элемент которого предназначен для хранения данных одного студента. Получение доступа к данным некоторого студента из группы *N* осуществляется обычной индексацией переменной массива. Поскольку поля структуры могут быть любого типа данных, то они в свою очередь могут быть другой структурой или массивом других структур:

struct Stud

{

char FN[100];

shortlistNumber;

};

struct Group

{

intgroupNumber;

short students;

Stud stud[30];

};

Но в структуре поля нельзя использовать элемент, тип которого совпадает с типом самой структуры, так как рекурсивное использование структур запрещено.

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:

voidWriteStudent ( t\_Student S )

{

cout<< "Фамилия: " <<S.Fam<<endl;

cout<< "Имя: " <<S.Name<<endl;

cout<< "Год рождения: " <<S.Year<<endl;

if ( S.Sex )

cout<< "Пол: " << "М\n";

else

cout<< "Пол: " << "Ж\n";

cout<< "Среднийбалл: " <<S.Grade<<endl;

}

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:

voidWriteStudent ( t\_Student \*S )

{

cout<< "Фамилия: " << S -> Fam <<endl;

cout<< "Имя: " << S -> Name <<endl;

cout<< "Годрождения: " << S -> Year <<endl;

if ( S -> Sex )

cout<< "Пол: " << "М\n";

else

cout<< "Пол: " << "Ж\n";

cout<< "Среднийбалл: " << S -> Grade <<endl;

}

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).

**Постановка задачи.**

Необходимо создать массив структур, содержащий информацию о студентах: ФИО, пол, номер группы, номер в списке группы, оценки за прошедшую сессию (всего 3 экзамена и 5 дифференцированных зачетов), форма обучения, отметка времени о внесении или изменении данных. Ввод и изменение данных обо всех студентах должен осуществляться в файл students.

Написать функции, реализующие операции со структурами (ввод данных с клавиатуры):

1.   Создание новой записи о студенте.

2.   Внесение изменений в уже имеющуюся запись.

3.   Вывод всех данных о студентах.

4.   Вывод информации обо всех студентах группы N. N – инициализируется пользователем.

5.   Вывод топа самых успешных студентов с наивысшим по рейтингу средним баллом за прошедшую сессию.

6.   Вывод количества студентов мужского и женского пола.

7.   Определение количества студентов, которые будут получать стипендию (стипендия начисляется, если у студента нет троек и очная форма обучения).

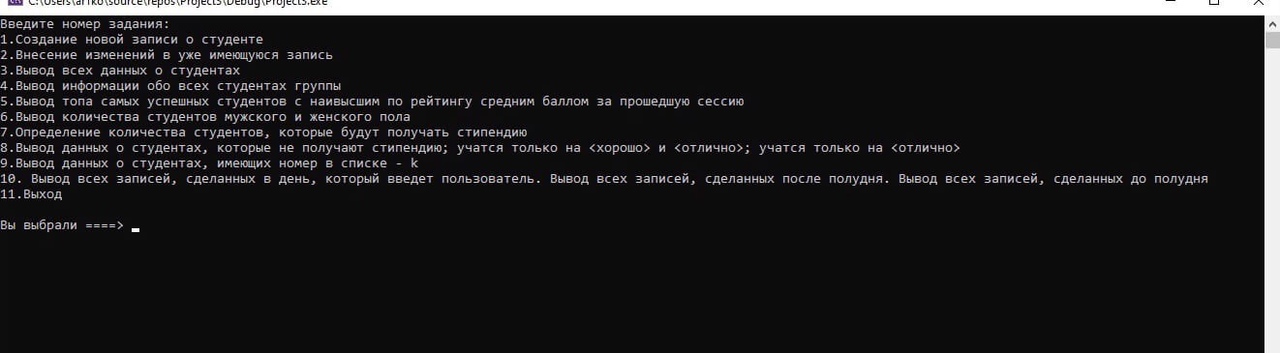
8.   Вывод данных о студентах, которые не получают стипендию; учатся только на «хорошо» и «отлично»; учатся только на «отлично»;

9.   Вывод данных о студентах, имеющих номер в списке – k.

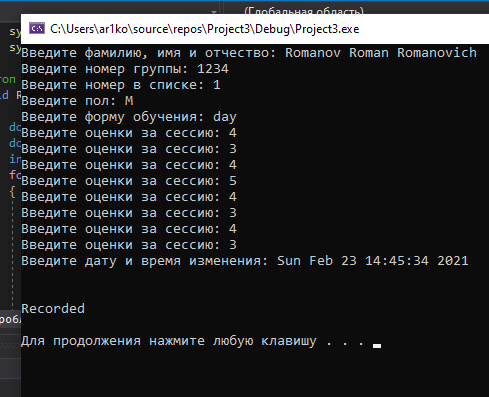
10.   Вывод всех записей, сделанных в день, который введет пользователь. Вывод всех записей, сделанных после полудня. Вывод всех записей, сделанных до полудня.

**Выполнение работы**

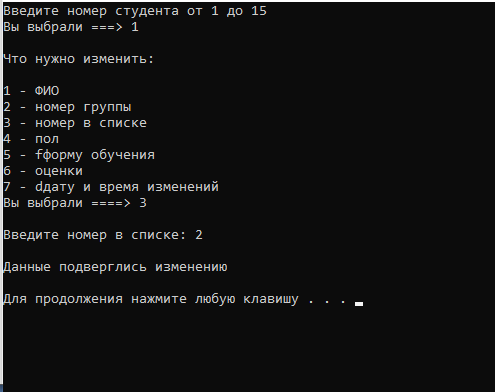
1)Визуал Меню



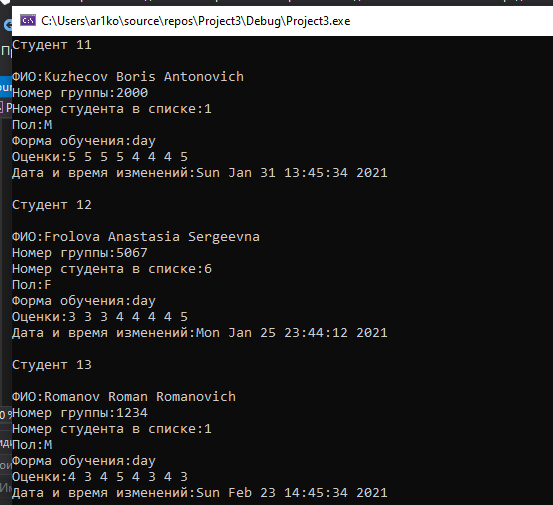
2)Пример работы первого задания.



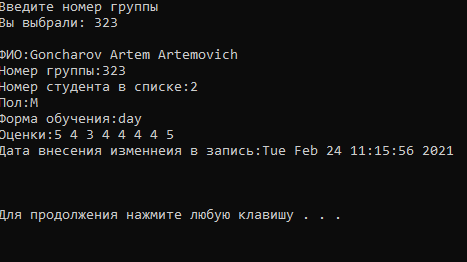
3) Пример работы второго задания

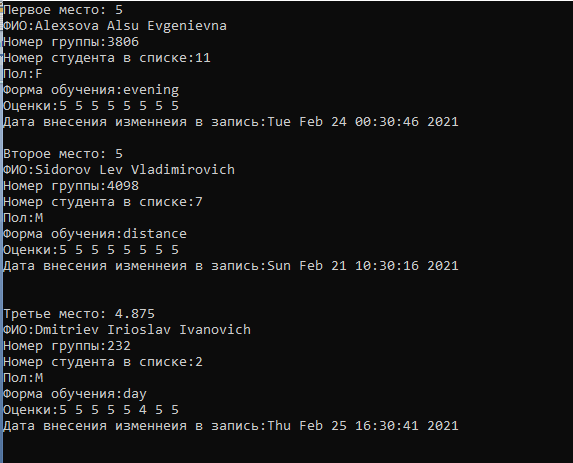
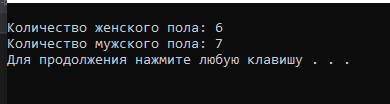


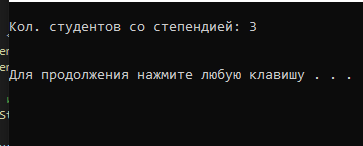
4)Пример работы 3 задания

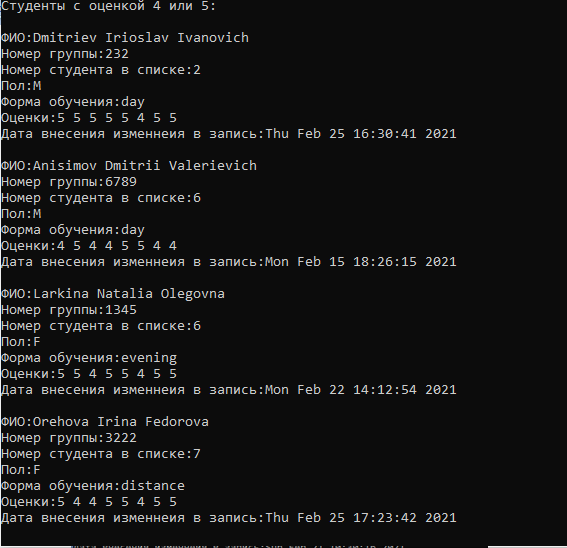
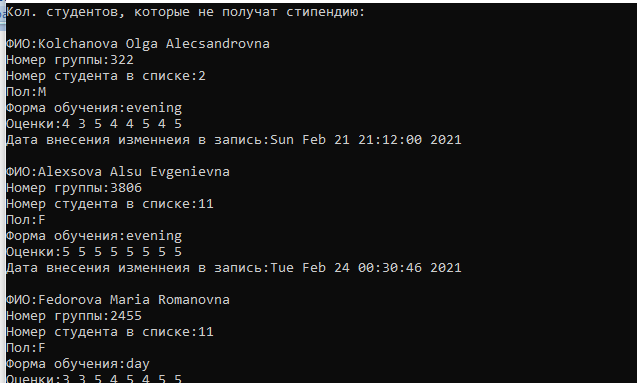
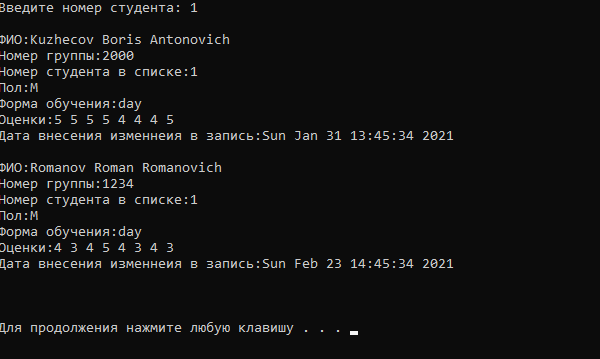
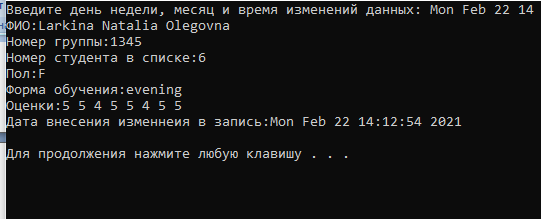
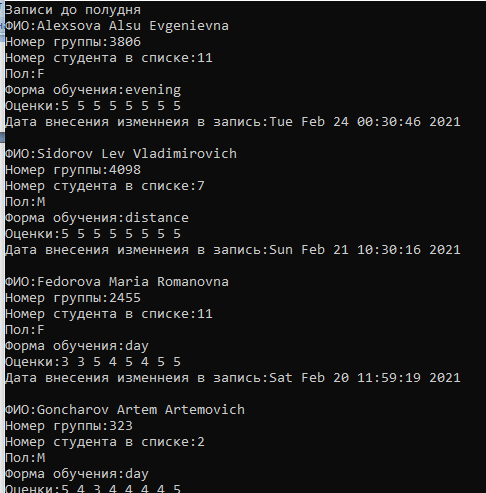
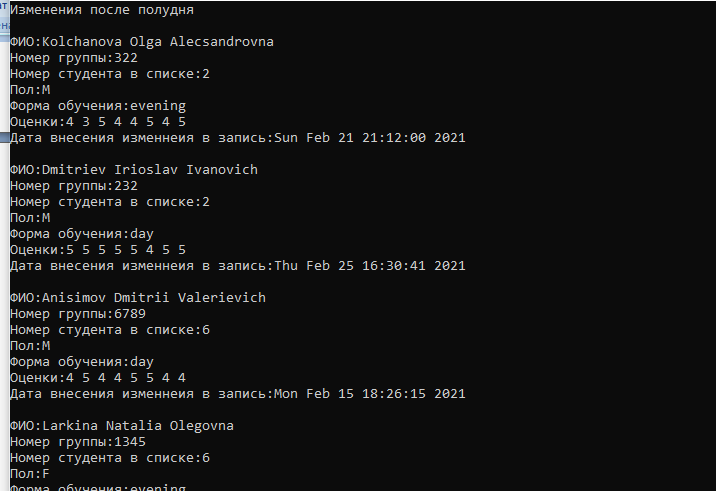


5)Пример решения четвертого задания



6)Пример работы пятого задания  
****7)Пример работы шестого задания  


8)Пример работы седьмого задания  


9)Пример работы восьмого задания  
****10)Пример работы 9 задания  
****11)Пример работы 10 задания  
****  
**  
Вывод**

Была проведена работа со структурами.

Структуры полезны, когда надо объединить несколько переменных с разными типами под одним именем. Это делает программу более компактной и более гибкой для внесения изменений. Также структуры хорошо подходят, когда необходимо сгруппировать некоторые данные, к примеру запись из базы данных.