**HIGHER CERTIFICATE IN SOFTWARE ENGINEERING**

**ACADEMIC YEAR 2020/2021**



Department of Electronic Engineering

Report title:

Software Project(Gym Database)

Student name: Godspower Olasupo

Student ID: x00150107

Submission date: 27/04/2021

Supervisor: Andrew Donnellan

**Declaration**

I hereby certify that the material, which I submit is entirely my own work and has not been taken from the work of others except to the extent that such work has been cited and acknowledged within the text of my own work.

Signature: \_\_\_\_Pelumi\_\_\_\_ Date: \_\_\_27/04/2021\_\_\_

Introduction

Below are the requirements of the project. My project is about a gym data base. The menu Displays exactly what can be see when you enter the Database. Entering a number from 1-8 will display different aspect of the Database.

[1] ] A basic class, with access member functions, Constructors, Static member data.

(a) class Deliveries

{

public:

Deliveries(int x = 0) : NumOfDeliveries(x) { count++; total += NumOfDeliveries; }//the number of deliveries daily is recorded

~Deliveries() { total -= NumOfDeliveries; count--; }

int CalculateNumOfWeeklyDeliveries() { return total; }

static int count; //public static data member

private:

static int total;

int NumOfDeliveries;

}; int Deliveries::count = 0; // must be initialised once at file scope

int Deliveries::total = 0; // can be initialised even if it is private

In Main:

if (menu == 8) {

Deliveries Mon(2), Tue(3), Wed(2), Thu(3), Fri(1);

cout << "Deliveries came in on " << Mon.count << " different days this week. "

<<"\nThe total number of deliveries is " << Tue.CalculateNumOfWeeklyDeliveries() << endl;

return 0;

}

(b)The deliveries class is a basic class with a access member function also called deliveries. The access member functions purpose is to record the number of deliveries and add it to get a total value. The count variable is also used to store the total number of individual deliveries, meaning the number of days that deliveries come in, which in this case is 5. The count is initialised as a static int variable. Static means no matter how many objects of the class are created, there is only one copy of the static member. The code prints out the number of days deliveries come in and the combined total deliveries through those days.

(c)There is a Delivery Constructor and Destructor. The constructor is a special method that is called when the object of a class is created.Destructor is a member function which deletes an object. This example displays the Gym record of Deliveries.

[2]/[3] Inheritance and polymorphism/ Abstract base classes and concrete derived classes

(a) // Base class

class Weights {

public:

virtual void print()

{

cout << "base class" << endl;

}// declaration of member print

void show() { cout << "Dumbells are derived from the Weights class" << endl; }

void setMass(int m) {//declare setMass

mass = m;

}

void setMass1(int m1) {//declare setMass1

mass1 = m1;

}

void setGravity(int g) {//declare setGravity

gravity = g;

}

protected://variable set in protected, cant be accessed by other classes

int mass;

int mass1;

int gravity;

};

// Derived class

class Kettebells : public Weights {//Kettlebells declaration class which is a derived class of weights

public:

int CalculateWeight() {//CalculateWeight function

return (mass \* gravity);

}

int CalculateWeight1() {//CalculateWeight1 function

return (mass1 \* gravity);

}

};

class Dumbells : public Weights {//dumbell decalaration class which is a derived class of weights

public:

void print() //print () is already virtual function in derived class, we could also declared as virtual void print () explicitly

{

cout << "In total there are 678 dumbells in the entire gym" << endl;

}

void show()//outputs string within cout

{

cout << "derived class" << endl;

}

};

In Main

if (menu == 2) {

Kettebells weight;

weight.setMass(5);

weight.setMass1(2);

weight.setGravity(9.8);

// Print the weight of the object.

cout << "There are 20 kettlebells that weigh: " << weight.CalculateWeight() << "Nw and" << endl;

cout << "There are 20 kettlebells that weigh: " << weight.CalculateWeight1() << "Nw\n" << endl;

Weights\* w;

Dumbells d;

w = &d;

//virtual function, binded at runtime (Runtime polymorphism)

w->print();

// Non-virtual function, binded at compile time

w->show();

return 0;

}

(b)This code contains an Abstract base class and concrete derived class. The Weights class is the base class, and the Dumbbells and Kettlebells are derived class. One or more methods are declared in the base class but are not defined. The derived class is where the virtual methods are overridden for the class to be concrete.2 different weights are calculated using functions from the Kettlebell class. The variable for gravity was declared also and the weights of 2 different types of kettlebells were calculated and displayed. There is also a virtual print function that prints of the number of dumbbells in the gym. There is a virtual and non-virtual function. The non-virtual function just prints to the console that the Dumbbell class is a derived class. This code shows the capability of a class to derive properties from another class, which is known as inheritance. Inheritance is known as a “is a ” relationship, meaning the kettlebell class is a derived class of a base class. Polymorphism means to have many forms. There are 2 types of polymorphism Compile time polymorphism and runtime polymorphism. The example in this code is known as runtime polymorphism because of the use of virtual functions such as print() and show(). Function overriding is occurring while the derived class provides a definition for a member function in the base class.

(C) The Gym contains different types of weights and here the user can see the number of dumbbells in the gym and the weights of the kettlebells. The gym keeps a record of its weights inventory.

4] Queue link list

(a)

In Main

char TempFName[15], TempSName[20];

float Tage;

float Theight;

Member Member1, MemberA;

queue q1;

int no, temp;

ifstream ip("C:\\Users\\x00150107\\Documents\\finalyear\\software\\software\\lab1 link list\\input.txt");//opens file

// modify for your location

ip >> no; /\* read in the number of Member \*/

for (temp = 0; temp < no; temp++)

{

/\* read in the Member data in file format \*/

ip >> TempFName;

Member1.assignFName(TempFName);

ip >> TempSName;

Member1.assignSName(TempSName);

ip >> Tage;

Member1.assignAge(Tage);

ip >> Theight;

Member1.assignheight(Theight);

q1.push(Member1);

}

ip.close();//close file

queue q2(q1);//placing q1 inside q2,sort of as a copy of q1

q2.push(MemberA);//StaffMemberA gets pushed into queue2

q1.pop();//oldest element in q1 gets popped off

q1.push(MemberA);//StaffMemberA gets pushed into queue2

cout << "Details for cleaning staff:\n " << endl;//print out message inbetween quotations

/\* for loop that pops out all the elements of the Members the same way it came in. it gets looped

as soon as it moves onto the next StaffMember.\*/

for (temp = 0; temp < no + 1; temp++)

{

Member1 = q2.pop();

Member1.print();

}

cout << "There are " << Member1.getNum() << " details documented about each Staff Member";//print out message inbetween quotations

return 0;

(b)The Queue link list code is an accumulation of different files, that are used to run the code just above. two pointers are used, frontand rear*.* The front points the first item of queue and rearpoints to last item on the queue. The push and pop function are they most important. Nodes are essential when it comes to keeping track of the current member in the queue and the next member. Different pages in the header and cpp files contain functions that allow the values of the cleaning staff to be passed through. The code takes in their first and second names, their age, and their height. Before the code can access this information, it needs to open an input file and read this information in from the file.

(c) This part of the Code allows the gym to keep personal information of their cleaning staff on records

[5] Operator Overloading, as a member function / [6]Operator Overloading, as a non member function

(a) class GymSession {

friend ostream& operator<<(ostream& ost2, const GymSession& d);

public:

GymSession(const GymSession& s);

int intensity;

friend int operator>(int, const GymSession& e);//non member

bool operator<(const GymSession& m);//member

GymSession(string a = "type", int b = 0, int c = 0) : type(a), numOfParticipants(b), intensity(c) { }//default constructor with default values for Employee class

private:

string type;

int numOfParticipants;

//int intensity;

};

GymSession::GymSession(const GymSession& s) {//overload function

type = s.type;

numOfParticipants = s.numOfParticipants;

intensity = s.intensity;

}

ostream& operator<<(ostream& ost2, const GymSession& d) {// output stream operater for GymSession

ost2 << d.numOfParticipants << " participants are taking part in a " << d.type << " session today, with an intesity of " << d.intensity;

return ost2;

}

int operator>(int intensity1, const GymSession& e) {//non member operator

//checks if intensity 1 is greater than the intensity interger its being compared to, it return an integer 1 or 0 depending

if (intensity1 > e.intensity) {

return 1;

}

else {

return 0;

}

}

bool GymSession::operator<(const GymSession& m) {//member operator

//GymSession numOfParticipants;

//checks if numOfParticipants is less than the numOfParticipants its being compared to, it return an boolean of true or false

if (numOfParticipants < m.numOfParticipants) {

return true;

}

else {

return false;

}

}

if (menu == 3) {

GymSession s1("hypertrophy focus", 25, 10);

GymSession s2("hypertrophy focus", 30, 5);

cout << s1;// operator << (cout,s1)

if (10 > s2) //calls greater than operater

{

cout << "\nthe intensity of s2 is less than 10 ";

}

else

{

cout << "\nthe intensity of s2 is greater than 10 ";

}

if (s2 < s1) {//calls less than operater

cout << "\nthe numOfParticipants in s2 is less than s1";

}

else

{

cout << "\nthe numOfParticipants in s2 is greater than s1";

}

return 0;

}

(b)There is a Gym session class. This code contains a non-member and member operator. The non-member operator is a function, that checks if the intensity1 is greater that the intensity its being compared to. If so, it will output the designated integer value. The member function is the less than operation. It checks which Gym session has fewer participants and return the Boolean value accordingly. There are two Gym sessions of hypertrophy in this code and the member and non-member functions use them to get a certain result and print the corresponding message to the console.

(c) The Gym need records of the classes they hold and which is for beginners or advanced members.

[7] Composition

(a)

class Qualification//Qualification class

{

friend ostream& operator<<(ostream& ost, const Qualification& a);

public:

Qualification(string title = "test", string institution = "test", int year = 2000) : t(title), i(institution), y(year) { }//default constructor with default values for Qualification class

~Qualification() {}; //deconstructor

private://initilizing variables

string t, i;//t for title, i for institution

int y;// y = year

};

class PersonalTrainers

{

//Friend allows it to be used by another class

friend ostream& operator<<(ostream& ost2, const PersonalTrainers& d);

public:

PersonalTrainers(const PersonalTrainers& s);

PersonalTrainers(string name = "name", string x = "N/A", string i = "institution", int year = 2020, int salary = 30000) : n(name), Q(x, i, year), S(salary) { }//default constructor with default values for PersonalTrainers class

private:

Qualification Q;//qualification class inside PersonalTrainers class

string n;//n for name

int S;//S for salary

};

PersonalTrainers::PersonalTrainers(const PersonalTrainers& s) {//overload function

n = s.n;

Q = s.Q;

S = s.S;

}

ostream& operator<<(ostream& ost, const Qualification& a) {// output stream operater for Qualification

ost << "with the highest qualification of " << a.t << " awarded in " << a.y << "\nby " << a.i; return ost;

}

ostream& operator<<(ostream& ost2, const PersonalTrainers& d) {// output stream operater for PersonalTrainers

ost2 << "The PersonalTrainers details are: Name is " << d.n << "\n" << d.Q << " and they have a salary of " << d.S;

return ost2;

}

if (menu == 4) {

PersonalTrainers PersonalTrainers1("Joe", "SportsScience", "UCD", 2020, 35000);//creating an PersonalTrainers object with (name, qualification details,institution, year, salary)

cout << PersonalTrainers1 << endl;//outputting PersonalTrainers object

cout << "There are 12 Personal trainers in the gym";

return 0;

}

(b) Composition is known as a “has a” relationship. The Personal trainer class has a composition relationship with the qualification class. The qualification has the same lifecycle as the employee and cannot exist independently of the employee object. The personal trainer class has a function that creates variables for name qualification and salary. The qualification class has a qualification function that declares variables for the title, institution, and the year of graduation. The code displays a certain personal trainers’ information on the console and the total number of personal trainers that there are.

(c) All the personal trainers need qualifications and the Gym requires records of their employees.

[8] Basic Association

(a)

class Membership

{

public:

Membership(string a = "12 months") : membership(a) {}

friend ostream& operator<<(ostream& ostr, Membership& a);

string typeMem();

private:

string membership;

};

class Person

{

public:

Person(string c = "name", string d = "membercode") : Name(c), memCode(d) { }

friend ostream& operator<<(ostream& ostr, Person& a);

string GetName();

private:

string Name;

string memCode;

};

ostream& operator<<(ostream& ostr, Person& a)

{

ostr << "The Person's name is " << a.Name << " and their membership code is " << a.memCode;

return ostr;

}

ostream& operator<<(ostream& ostr, Membership& a)

{

ostr << "Their Type of membership is : " << a.membership;

return ostr;

}

string Person::GetName()

{

return Name;

}

string Membership::typeMem()

{

return membership;

}

if (menu == 5) {

Person p1("Tom", "76897576");

Person p2("Jerry", "45765879");

Membership m1("12 months");

Membership m2("6 months");

cout << " p1: " << p1 << "\n" << m1 << endl;

cout << " p2: " << p2 << "\n" << m2 << endl;

return 0;

}

(b)The multiplicity in this is 1 to 1. One person can have only one membership. The person owns a membership. It is a unidirectional association. Associations are connections between classes. The person class is connected to the membership class. In the code there are functions that output the persons details such as the name and member code. The membership class outputs the type of membership the person has. 2 members and their membership types are outputted to the console.

(c) It is important for the gym to have a record of the kind of membership each member has.

[9] Qualifier / Qualified Association

class Orders

{

public:

Orders(string a = "No product", double b = 0) : description(a), price(b) {}

friend ostream& operator<<(ostream& os, Orders& c);

private:

string description;

double price;

};

class Orderlist

{

public:

Orderlist(string c = "noname") : Name(c) { }

void AddOrdersToOrderList(Orders\* d, string Ordernumber) {

itemlist[Ordernumber] = d;

}

Orders\* getOrderCode(string OrderCode) {

return itemlist[OrderCode];

}

private:

string Name;

std::map < string, Orders\* > itemlist;

};

ostream& operator<<(ostream& os, Orders& c) {// output stream operater for Employee

os << "OrderList Details\n" << "Order name:" << c.description;

os << "\tat a Price of $" << c.price;

return os;

}

if (menu == 6) {

Orderlist Orderlist1("types of benches");

Orders Order1("4 incline benchs", 500);

Orderlist1.AddOrdersToOrderList(&Order1, "X09897");

Orders\* c = Orderlist1.getOrderCode("X09897");

cout << \*c;

return 0;

}

(b)A Qualified association has a qualifier that is used to select an object out of a list. It allows it to select a specified item from a list/catalogue. In this case the qualifier is the order description and price. In the code there is an order list named types of benches, the code is looking for a certain order with the description 4 incline benches and a price of 500. It highlighted a specific order out of all the orders in the order list. This order has an order number attached to it which, separates it from all the other orders too. The code outputs the name of the order list and the specified description and price of a certain object in the list.

(c) The gym needs new equipment and the order needed is logging in the order list

[11] Dependency

(a)

class Volume

{

private:

int L, W, D;

public:

Volume(int l = 0, int w = 0, int d = 0) : L(l), W(w), D(d)

{

}

friend std::ostream& operator<< (std::ostream& out, const Volume& volume);

};

std::ostream& operator<< (std::ostream& out, const Volume& volume)

{

// Since operator<< is a friend of the Point class, we can access Point's members directly.

out << "Variables(" << volume.L << ", " << volume.W << ", " << volume.D << ")";

return out;

}

int CalculateVolume(int, int, int);

int CalculateVolume(int length, int width, int depth)

{

return(length \* width \* depth \* 7.5);

}

if (menu == 1)

{

{

int length = 50, width = 25, depth = 2;

Volume volume1(50, 25, 2);

std::cout << "\n" << volume1;

std::cout << "\nThe Volume of the swimming pool is " << CalculateVolume(length, width, depth) << "cm^3";

return 0;

}

}

(b)Dependency is like it says, when one class depends on another class. Volume isn’t directly related to std::cout, although Volume has a dependency on std::cout. The operator<< uses std::cout to print the Volume of the swimming pool. The function CalculateVolume takes the three variables of length width and depth, multiplies them and the result of it is outputted to the console.

(c)This ties into the topic of Gym because this specific gym has a swimming pool, and it needs the measurements and volume of the pool in case of future modifications.

[12] Detail best thing that you did i.e what was most novel, interesting or challenging piece of code, integration issue, etc that you solved

Snippets of Running code.

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generatedText

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Full Source code. Source.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <math.h>

#include <string.h>

#include <iostream>

#include <map>

#include <fstream>

#include <string>

#include "Member.h"

#include "Queue.h"

**using** **namespace** std**;**

int sel**,** menu**;**

/\*1. swimingpool Volume\*/

class Volume

**{**

private**:**

int L**,** W**,** D**;**

public**:**

Volume**(**int l **=** 0**,** int w **=** 0**,** int d **=** 0**)** **:** L**(**l**),** W**(**w**),** D**(**d**)**

**{**

**}**

friend std**::**ostream**&** **operator<<** **(**std**::**ostream**&** out**,** const Volume**&** volume**);**

**};**

std**::**ostream**&** **operator<<** **(**std**::**ostream**&** out**,** const Volume**&** volume**)**

**{**

// Since operator<< is a friend of the Point class, we can access Point's members directly.

out **<<** "Variables(" **<<** volume**.**L **<<** ", " **<<** volume**.**W **<<** ", " **<<** volume**.**D **<<** ")"**;**

**return** out**;**

**}**

int CalculateVolume**(**int**,** int**,** int**);**

int CalculateVolume**(**int length**,** int width**,** int depth**)**

**{**

**return(**length **\*** width **\*** depth **\*** 7.5**);**

**}**

//function to calculate membership price

inline float MembershipCalculator**(**int numMonths **=** 1**)** **{**

**return(**numMonths **\*** 35.00**);**

**}**

/\*2. Weights\*/

// Base class

class Weights **{**

public**:**

virtual void print**()**

**{**

cout **<<** "base class" **<<** endl**;**

**}**// declaration of member print

void show**()** **{** cout **<<** "Dumbells are derived from the Weights class" **<<** endl**;** **}**

void setMass**(**int m**)** **{**//declare setMass

mass **=** m**;**

**}**

void setMass1**(**int m1**)** **{**//declare setMass1

mass1 **=** m1**;**

**}**

void setGravity**(**int g**)** **{**//declare setGravity

gravity **=** g**;**

**}**

protected**:**//variable set in protected, cant be accessed by other classes

int mass**;**

int mass1**;**

int gravity**;**

**};**

// Derived class

class Kettebells **:** public Weights **{**//Kettlebells declaration class which is a derived class of weights

public**:**

int CalculateWeight**()** **{**//CalculateWeight function

**return** **(**mass **\*** gravity**);**

**}**

int CalculateWeight1**()** **{**//CalculateWeight1 function

**return** **(**mass1 **\*** gravity**);**

**}**

**};**

class Dumbells **:** public Weights **{**//dumbell decalaration class which is a derived class of weights

public**:**

void print**()** //print () is already virtual function in derived class, we could also declared as virtual void print () explicitly

**{**

cout **<<** "In total there are 678 dumbells in the entire gym" **<<** endl**;**

**}**

void show**()**//outputs string within cout

**{**

cout **<<** "derived class" **<<** endl**;**

**}**

**};**

/\*3. Participants of hypertrophy class\*/

class GymSession **{**

friend ostream**&** **operator<<(**ostream**&** ost2**,** const GymSession**&** d**);**

public**:**

GymSession**(**const GymSession**&** s**);**

int intensity**;**

friend int **operator>(**int**,** const GymSession**&** e**);**//non member

bool **operator<(**const GymSession**&** m**);**//member

GymSession**(**string a **=** "type"**,** int b **=** 0**,** int c **=** 0**)** **:** type**(**a**),** numOfParticipants**(**b**),** intensity**(**c**)** **{** **}**//default constructor with default values for Employee class

private**:**

string type**;**

int numOfParticipants**;**

//int intensity;

**};**

GymSession**::**GymSession**(**const GymSession**&** s**)** **{**//overload function

type **=** s**.**type**;**

numOfParticipants **=** s**.**numOfParticipants**;**

intensity **=** s**.**intensity**;**

**}**

ostream**&** **operator<<(**ostream**&** ost2**,** const GymSession**&** d**)** **{**// output stream operater for GymSession

ost2 **<<** d**.**numOfParticipants **<<** " participants are taking part in a " **<<** d**.**type **<<** " session today, with an intesity of " **<<** d**.**intensity**;**

**return** ost2**;**

**}**

int **operator>(**int intensity1**,** const GymSession**&** e**)** **{**//non member operator

//checks if intensity 1 is greater than the intensity interger its being compared to, it return an integer 1 or 0 depending

**if** **(**intensity1 **>** e**.**intensity**)** **{**

**return** 1**;**

**}**

**else** **{**

**return** 0**;**

**}**

**}**

bool GymSession**::operator<(**const GymSession**&** m**)** **{**//member operator

//GymSession numOfParticipants;

//checks if numOfParticipants is less than the numOfParticipants its being compared to, it return an boolean of true or false

**if** **(**numOfParticipants **<** m**.**numOfParticipants**)** **{**

**return** **true;**

**}**

**else** **{**

**return** **false;**

**}**

**}**

/\*4. Details of PersonalTrainer\*/

class Qualification//Qualification class

**{**

friend ostream**&** **operator<<(**ostream**&** ost**,** const Qualification**&** a**);**

public**:**

Qualification**(**string title **=** "test"**,** string institution **=** "test"**,** int year **=** 2000**)** **:** t**(**title**),** i**(**institution**),** y**(**year**)** **{** **}**//default constructor with default values for Qualification class

**~**Qualification**()** **{};** //deconstructor

private**:**//initilizing variables

string t**,** i**;**//t for title, i for institution

int y**;**// y = year

**};**

class PersonalTrainers

**{**

//Friend allows it to be used by another class

friend ostream**&** **operator<<(**ostream**&** ost2**,** const PersonalTrainers**&** d**);**

public**:**

PersonalTrainers**(**const PersonalTrainers**&** s**);**

PersonalTrainers**(**string name **=** "name"**,** string x **=** "N/A"**,** string i **=** "institution"**,** int year **=** 2020**,** int salary **=** 30000**)** **:** n**(**name**),** Q**(**x**,** i**,** year**),** S**(**salary**)** **{** **}**//default constructor with default values for PersonalTrainers class

private**:**

Qualification Q**;**//qualification class inside PersonalTrainers class

string n**;**//n for name

int S**;**//S for salary

**};**

PersonalTrainers**::**PersonalTrainers**(**const PersonalTrainers**&** s**)** **{**//overload function

n **=** s**.**n**;**

Q **=** s**.**Q**;**

S **=** s**.**S**;**

**}**

ostream**&** **operator<<(**ostream**&** ost**,** const Qualification**&** a**)** **{**// output stream operater for Qualification

ost **<<** "with the highest qualification of " **<<** a**.**t **<<** " awarded in " **<<** a**.**y **<<** "\nby " **<<** a**.**i**;** **return** ost**;**

**}**

ostream**&** **operator<<(**ostream**&** ost2**,** const PersonalTrainers**&** d**)** **{**// output stream operater for PersonalTrainers

ost2 **<<** "The PersonalTrainers details are: Name is " **<<** d**.**n **<<** "\n" **<<** d**.**Q **<<** " and they have a salary of " **<<** d**.**S**;**

**return** ost2**;**

**}**

/\*5. Details of Gym Members\*/

class Membership

**{**

public**:**

Membership**(**string a **=** "12 months"**)** **:** membership**(**a**)** **{}**

friend ostream**&** **operator<<(**ostream**&** ostr**,** Membership**&** a**);**

string typeMem**();**

private**:**

string membership**;**

**};**

class Person

**{**

public**:**

Person**(**string c **=** "name"**,** string d **=** "membercode"**)** **:** Name**(**c**),** memCode**(**d**)** **{** **}**

friend ostream**&** **operator<<(**ostream**&** ostr**,** Person**&** a**);**

string GetName**();**

private**:**

string Name**;**

string memCode**;**

**};**

ostream**&** **operator<<(**ostream**&** ostr**,** Person**&** a**)**

**{**

ostr **<<** "The Person's name is " **<<** a**.**Name **<<** " and their membership code is " **<<** a**.**memCode**;**

**return** ostr**;**

**}**

ostream**&** **operator<<(**ostream**&** ostr**,** Membership**&** a**)**

**{**

ostr **<<** "Their Type of membership is : " **<<** a**.**membership**;**

**return** ostr**;**

**}**

string Person**::**GetName**()**

**{**

**return** Name**;**

**}**

string Membership**::**typeMem**()**

**{**

**return** membership**;**

**}**

/\*6. Orders\*/

class Orders

**{**

public**:**

Orders**(**string a **=** "No product"**,** double b **=** 0**)** **:** description**(**a**),** price**(**b**)** **{}**

friend ostream**&** **operator<<(**ostream**&** os**,** Orders**&** c**);**

private**:**

string description**;**

double price**;**

**};**

class Orderlist

**{**

public**:**

Orderlist**(**string c **=** "noname"**)** **:** Name**(**c**)** **{** **}**

void AddOrdersToOrderList**(**Orders**\*** d**,** string Ordernumber**)** **{**

itemlist**[**Ordernumber**]** **=** d**;**

**}**

Orders**\*** getOrderCode**(**string OrderCode**)** **{**

**return** itemlist**[**OrderCode**];**

**}**

private**:**

string Name**;**

std**::**map **<** string**,** Orders**\*** **>** itemlist**;**

**};**

ostream**&** **operator<<(**ostream**&** os**,** Orders**&** c**)** **{**// output stream operater for Employee

os **<<** "OrderList Details\n" **<<** "Order name:" **<<** c**.**description**;**

os **<<** "\tat a Price of $" **<<** c**.**price**;**

**return** os**;**

**}**

/\*7.Details for cleaning staff\*/

//Link list

/\*8. Deliveries\*/

class Deliveries

**{**

public**:**

Deliveries**(**int x **=** 0**)** **:** NumOfDeliveries**(**x**)** **{** count**++;** total **+=** NumOfDeliveries**;** **}**//the number of deliveries daily is recorded and recorded

**~**Deliveries**()** **{** total **-=** NumOfDeliveries**;** count**--;** **}**

int CalculateNumOfWeeklyDeliveries**()** **{** **return** total**;** **}**

static int count**;** //public static data member

private**:**

static int total**;**

int NumOfDeliveries**;**

**};** int Deliveries**::**count **=** 0**;** // must be initialised once at file scope

int Deliveries**::**total **=** 0**;** // can be initialised even if it is private

int main**(**void**)**

**{**

printf**(**"\n\t\t\t\t ---GYM System-- "**);**

printf**(**"\n\t1- SwimingPool Volume "**);**

printf**(**"\n\t2- Weights"**);**

printf**(**"\n\t3- Participants of hypertrophy class"**);**

printf**(**"\n\t4- Details of PersonalTrainers"**);**

printf**(**"\n\t5- Details of Gym Members"**);**

printf**(**"\n\t6- Orders"**);**

printf**(**"\n\t7- Details for cleaning staff"**);**

printf**(**"\n\t8- Deliveries"**);**

printf**(**"\n\t9- Quit"**);**

printf**(**"\n\tSelect function by it's corresponding number: "**);**

scanf**(**"%d"**,** **&**menu**);**

**do** **{**

**if** **(**menu **==** 1**)**

**{**

**{**

int length **=** 50**,** width **=** 25**,** depth **=** 2**;**

Volume volume1**(**50**,** 25**,** 2**);**

std**::**cout **<<** "\n" **<<** volume1**;**

std**::**cout **<<** "\nThe Volume of the swimming pool is " **<<** CalculateVolume**(**length**,** width**,** depth**)** **<<** "cm^3"**;**

**return** 0**;**

**}**

**}**

**if** **(**menu **==** 2**)** **{**

Kettebells weight**;**

weight**.**setMass**(**5**);**

weight**.**setMass1**(**2**);**

weight**.**setGravity**(**9.8**);**

// Print the weight of the object.

cout **<<** "There are 20 kettlebells that weigh: " **<<** weight**.**CalculateWeight**()** **<<** "Nw and" **<<** endl**;**

cout **<<** "There are 20 kettlebells that weigh: " **<<** weight**.**CalculateWeight1**()** **<<** "Nw\n" **<<** endl**;**

Weights**\*** w**;**

Dumbells d**;**

w **=** **&**d**;**

//virtual function, binded at runtime (Runtime polymorphism)

w**->**print**();**

// Non-virtual function, binded at compile time

w**->**show**();**

**return** 0**;**

**}**

**if** **(**menu **==** 3**)** **{**

GymSession s1**(**"hypertrophy focus"**,** 25**,** 10**);**

GymSession s2**(**"hypertrophy focus"**,** 30**,** 5**);**

cout **<<** s1**;**// operator << (cout,s1)

**if** **(**10 **>** s2**)**

**{**

cout **<<** "\nthe intensity of s2 is less than 10 "**;**

**}**

**else**

**{**

cout **<<** "\nthe intensity of s2 is greater than 10 "**;**

**}**

**if** **(**s2 **<** s1**)** **{**

cout **<<** "\nthe numOfParticipants in s2 is less than s1"**;**

**}**

**else**

**{**

cout **<<** "\nthe numOfParticipants in s2 is greater than s1"**;**

**}**

**return** 0**;**

**}**

**if** **(**menu **==** 4**)** **{**

PersonalTrainers PersonalTrainers1**(**"Joe"**,** "Sports Science"**,** "UCD"**,** 2020**,** 35000**);**//creating an PersonalTrainers object with (name, qualification details,institution, year, salary)

cout **<<** PersonalTrainers1 **<<** endl**;**//outputting PersonalTrainers object

cout **<<** "There are 12 Personal trainers in the gym"**;**

**return** 0**;**

**}**

**if** **(**menu **==** 5**)** **{**

Person p1**(**"Tom"**,** "76897576"**);**

Person p2**(**"Jerry"**,** "45765879"**);**

Membership m1**(**"12 months"**);**

Membership m2**(**"6 months"**);**

cout **<<** " p1: " **<<** p1 **<<** "\n" **<<** m1 **<<** endl**;**

cout **<<** " p2: " **<<** p2 **<<** "\n" **<<** m2 **<<** endl**;**

**return** 0**;**

**}**

**if** **(**menu **==** 6**)** **{**

Orderlist Orderlist1**(**"types of benchs"**);**

Orders Order1**(**"4 incline benchs"**,** 500**);**

Orderlist1**.**AddOrdersToOrderList**(&**Order1**,** "X09897"**);**

Orders**\*** c **=** Orderlist1**.**getOrderCode**(**"X09897"**);**

cout **<<** **\***c**;**

**return** 0**;**

**}**

**if** **(**menu **==** 7**)** **{**

char TempFName**[**15**],** TempSName**[**20**];**

float Tage**;**

float Theight**;**

Member Member1**,** MemberA**;**

queue q1**;**

int no**,** temp**;**

ifstream ip**(**"C:\\Users\\x00150107\\Documents\\finalyear\\software\\software\\lab1 link list\\input.txt"**);**//opens file

// modify for your location

ip **>>** no**;** /\* read in the number of Member \*/

**for** **(**temp **=** 0**;** temp **<** no**;** temp**++)**

**{**

/\* read in the Member data in file format \*/

ip **>>** TempFName**;**

Member1**.**assignFName**(**TempFName**);**

ip **>>** TempSName**;**

Member1**.**assignSName**(**TempSName**);**

ip **>>** Tage**;**

Member1**.**assignAge**(**Tage**);**

ip **>>** Theight**;**

Member1**.**assignheight**(**Theight**);**

q1**.**push**(**Member1**);**

**}**

ip**.**close**();**//close file

queue q2**(**q1**);**//placing q1 inside q2,sort of as a copy of q1

q2**.**push**(**MemberA**);**//StaffMemberA gets pushed into queue2

q1**.**pop**();**//oldest element in q1 gets popped off

q1**.**push**(**MemberA**);**//StaffMemberA gets pushed into queue2

cout **<<** "Details for cleaning staff:\n " **<<** endl**;**//print out message inbetween quotations

/\* for loop that pops out all the elements of the Members the same way it came in. it gets looped

as soon as it moves onto the next StaffMember.\*/

**for** **(**temp **=** 0**;** temp **<** no **+** 1**;** temp**++)**

**{**

Member1 **=** q2**.**pop**();**

Member1**.**print**();**

**}**

cout **<<** "There are " **<<** Member1**.**getNum**()** **<<** " details documented about each Staff Member"**;**//print out message inbetween quotations

**return** 0**;**

**}**

**if** **(**menu **==** 8**)** **{**

Deliveries Mon**(**2**),** Tue**(**3**),** Wed**(**2**),** Thu**(**3**),** Fri**(**1**);**

cout **<<** "Deliveries came in on " **<<** Mon**.**count **<<** " different days this week. "

**<<**"\nThe total number of deliveries is " **<<** Tue**.**CalculateNumOfWeeklyDeliveries**()** **<<** endl**;**

**return** 0**;**

**}**

**if** **(**menu **==** 9**)** **{**

printf**(**"QUITTING PROGRAM..."**);**

**return(**0**);**

**}**

**else** **{**

printf**(**"Error: invalid Input.\nInput must be between a number between 1-9\n\n"**);**

**}**

**}** **while** **(**menu **!=** 9**);** //while quit is not selected

**}**

Member.h

#pragma once

//Module.h

#ifndef STD\_H

#define STD\_H

#include <string>

using namespace std;

class Member

{

public:

// just include one constructor definition here to show how it is written if do inline

Member(string fn = "Firstname", string sn = "Surname", int a = 0, float h = 0)

:firstname(fn), surname(sn), age(a), height(h) {

count++;//increments count

}

Member(const Member& s);//constuctor

~Member();//deconstructor

void assignFName(string tempFName);

void assignSName(string tempSName);

void assignheight(float Theight);

void assignAge(int Tage);

string getFName() { return firstname; } // just include one method here to show how it is written if do inline

string getSName();

int getAge();

float getheight();

void print() const;

static int getNum();

private:

string firstname, surname;

int age;

float height;

static int count;

};

#endif

Node.h

//pragma once

#include <iostream>

#include <string>

#include <fstream>

#include "Member.h"

using namespace std;

class node

{

//since queue is a friend class we can access all the member function of class node inside class queue

friend class queue;

public:

//constructor to initialize the data to d and next to NUll

node();

node(const Member& d) : data(d), next(NULL) {};

private:

Member data;

node\* next;

};

Queue.h

#pragma once

class Member;

class node;

class queue {

public:

queue();//constructor

~queue();//deconstructor

queue(const queue&);

int push(Member value);

Member pop(void);

private:

node\* listhead, \* listtail;//initialized by pointer

};

Member.cpp

#include <iostream> // need for cout, etc

#include "Member.h"

using namespace std;

//function of all the elements of a single Member,and it increases count by one to shift onto the next element

Member::Member(const Member& s)

{

firstname = s.firstname;

surname = s.surname;

age = s.age;

height = s.height;

count++;

}

Member::~Member() { count--; }//desconstructor of Member and decrements count

void Member::assignFName(string tempFName)

{

firstname = tempFName;

}

void Member::assignSName(string tempSName) { surname = tempSName; }

void Member::assignheight(float Theight) { height = Theight; }

void Member::assignAge(int Tage) { age = Tage; }

string Member::getSName() { return surname; }

int Member::getAge() { return age; }

float Member::getheight() { return height; }

int Member::getNum() { return count -6; }

//function to print out all the elements of a single Member

void Member::print() const

{

cout << firstname << " " << surname << " " << age << " " << height << " " << "\n" << endl; // prints firstname, surname, age and height.

}

int Member::count = 0;//allows the variable name count to be used but it has a different scope

Node.cpp

#include <iostream>

#include <string>

#include <fstream>

#include "Member.h"

using namespace std;

class node

{

//since queue is a friend class we can access all the member function of class node inside class queue

friend class queue;

public:

//constructor to initialize the data to d and next to NUll

node();

node(const Member& d) : data(d), next(NULL) {};

private:

Member data;

node\* next;

};

node::node() :next(NULL) {}////allows the variable next to be used but it has a different scope

Queue.cpp

#include "Member.h"

#include "Node.h"

#include "Queue.h"

queue::queue()

{

//node\* temp = new node;

listhead = NULL;

listtail = NULL;

}

queue::~queue()

{

node\* temp;

while (listhead != NULL)

{

// get the address of next node

temp = listhead;

// move to the next node on queue

listhead = listhead->next;

delete temp;

}

}

queue::queue(const queue& r)

{

node\* hold = r.listhead;

node\* temp, \* oldtemp;

//if the queue is empty ,both listhead and listtail equal null

if (hold == NULL)

{

listhead = NULL;

listtail = NULL;

}

//else listhead gets set as temp and the temp value goes through all the elements in the list untill it reaches listtail

else

{

temp = new node;

listhead = temp;

while (hold != NULL)

{

// get the value at the node looking at on queue 'c' list

temp->data = hold->data;

// move to the next entry on 'c' list

hold = hold->next;

oldtemp = temp;

if (hold != NULL)

{

temp = new node;

oldtemp->next = temp;

}

}

listtail = temp;

}

}

int queue::push(Member value)

{

int error = 1;

// get a new node

node\* temp = new node(value);

if (temp == NULL) //queue full

{

cout << "queue overflow" << endl;//prints this if queue is full

error = -32000;

}

else

{ /\*temp gets shift through all the objects of the list from the variable that entered to list first to the one that entered

the list last\*/

if (listhead == NULL)

{

listhead = temp;

}

else

{

listtail->next = temp;

}

listtail = temp;

}

return error;

}

Member queue::pop(void)

{

node\* temp;

Member value;

if (listhead == NULL) //queue empty

{

cout << "queue underflow" << endl;//prints this if queue is empty

}

else

{

// get the value at the front of the queue

value = listhead->data;

// takes top entry from the queue

temp = listhead;

listhead = listhead->next;

delete temp;

//Check if Queue empty

if (listhead == NULL)

listtail = NULL;

}

return value;// return the value

}