**Problem No.**: 01

**Problem Name:** Classes and Objects.

**Code:**

#include<iostream>

using namespace std;

class Rectangle{

private:

double w;

double l;

public:

Rectangle(){

w=1.0;

l=1.0;

};

public:

Rectangle(double width, double length){

w = width;

l = length;

}

public:

double area(){

return w\*l;

}

public:

double perimeter(){

return 2\*(w+l);

}

};

int main(){

double w1,l1, w2,l2, w3,l3;

cout<<"Enter width and length for the first rectangle: ";

cin>>w1>>l1;

cout<<"Enter width and length for the second rectangle: ";

cin>>w2>>l2;

cout<<"Enter width and length for the third rectangle: ";

cin>>w3>>l3;

Rectangle r[3] = {Rectangle(w1,l1), Rectangle(w2,l2), Rectangle(w3,l3)};

;

cout<<"Area of Rec1 = "<<r[0].area()<<endl;

cout<<"Area of Rec2 = "<<r[1].area()<<endl;

cout<<"Area of Rec3 = "<<r[2].area()<<endl;

cout<<"Perimeter of Rec1 = "<<r[0].perimeter()<<endl;

cout<<"Perimeter of Rec2 = "<<r[1].perimeter()<<endl;

cout<<"Perimeter of Rec3 = "<<r[2].perimeter()<<endl;

return 0;

}

**Output:**

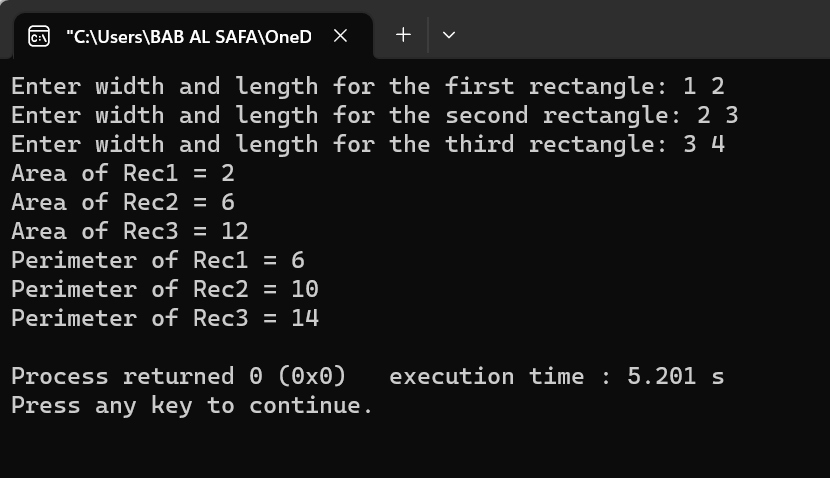


Fig 1.1: Output on console for case 1.

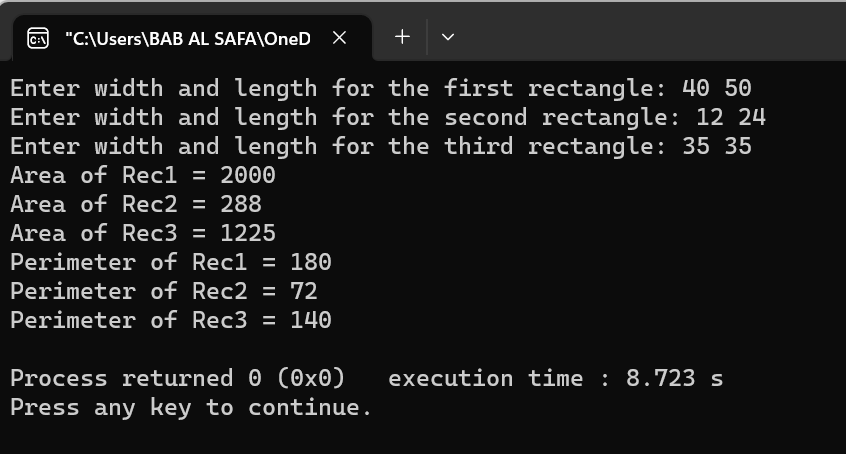


Fig 1.2: Output on console for case 2.

**Explanation:**

This is a C++ program that calculates the area and perimeter of three rectangles. The program uses a class named Rectangle that has two private variables w and l for width and length respectively. The class has four public methods: a default constructor that sets w and l to 1.0, a constructor that takes two arguments for w and l, an area method that returns the area of the rectangle, and a perimeter method that returns the perimeter of the rectangle. The program then creates three Rectangle objects using the two-argument constructor and stores them in an array. Finally, it prints out the area and perimeter of each rectangle.

**Problem No.**: 02

**Problem Name**: Class and Objects.

**Code:**

#include<iostream>

#include <ctime>

#include <chrono>

using namespace std;

class Date {

private:

std::chrono::time\_point<std::chrono::system\_clock> start = std::chrono::system\_clock::now();

std::chrono::time\_point<std::chrono::system\_clock> end = std::chrono::system\_clock::now();

std::time\_t end\_time = std::chrono::system\_clock::to\_time\_t(end);

public:

Date(){

cout << "finished computation at " << std::ctime(&end\_time)<<endl;

}

};

class Account{

private:

int ID;

double Balance;

double AnnualInterestRate;

Date date\_created;

public:

Account(int id, double balance, double annualInterestRate){

ID = id;

Balance = balance;

AnnualInterestRate = annualInterestRate;

}

bool withdraw(double amount){

if(Balance-amount<0)

return false;

else{

Balance-=amount;

return true;

}

}

void deposit(double amount){

Balance+=amount;

}

void getId(){

cout<<ID<<" ";

}

void getBalance(){

cout<<Balance<<" ";

}

void getAnnualInterestRate(){

cout<<AnnualInterestRate<<" ";

}

};

int main(){

Account testAccount(1122, 40000, 4.5);

if(testAccount.withdraw(2500)){

cout<<"Withdraw successfull"<<endl;

}

if(!testAccount.withdraw(2500)){

cout<<"Withdraw not successfull"<<endl;

}

testAccount.deposit(3000);

testAccount.getId();

testAccount.getBalance();

testAccount.getAnnualInterestRate();

cout<<endl;

return 0;

}

**Output:**

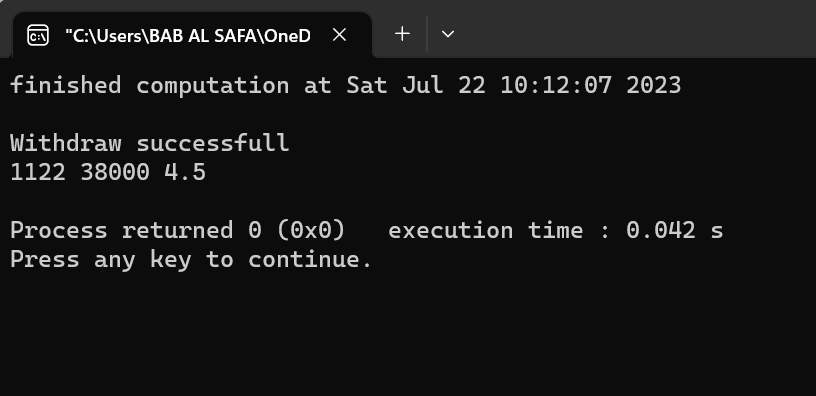


Fig 2.1: Output on console.

**Explanation:**

This code creates a class called `Account`. The class has four private data members: `ID`, `Balance`, `AnnualInterestRate`, and `date\_created`. The constructor initializes the data members with the values passed as arguments. The class has three public member functions: `withdraw()`, `deposit()`, and three accessor functions: `getId()`, `getBalance()`, and `getAnnualInterestRate()`. The function `withdraw()` checks if the amount to be withdrawn is greater than the balance. If it is, it returns false; otherwise, it subtracts the amount from the balance and returns true. The function `deposit()` adds the amount passed as an argument to the balance. The accessor functions return the values of their respective data members. The code also creates an instance of the class called `testAccount` with ID 1122, balance 40000, and annual interest rate 4.5. It then calls the member functions of this instance to test them. The code also creates an instance of the class called `Date` which prints out the time when the computation finished.

**Problem No.**: 03

**Problem Name**: Class and Objects.

**Code:**

#include<iostream>

#include <string>

#include <list>

using namespace std;

class Account{

private:

int ID;

double Balance;

double AnnualInterestRate;

public:

Account(int id, double balance, double annualInterestRate){

ID = id;

Balance = balance;

AnnualInterestRate = annualInterestRate;

}

bool withdraw(double amount){

if(Balance-amount<0)

return false;

else{

Balance-=amount;

return true;

}

}

void deposit(double amount){

Balance+=amount;

}

int getID(){

return ID;

}

string toString(){

return to\_string(ID) + " " + to\_string(Balance) + " " + to\_string(AnnualInterestRate);

}

};

class Client{

private:

int id;

std::string name;

std::string phone;

std::list<Account>accounts;

public:

Client(int ID, std::string Name, std::string Phone){

id = ID;

name = Name;

phone = Phone;

std::list<Account>accounts;

}

bool addAccount(Account account){

accounts.push\_back(account);

return true;

}

bool removeAccount(int accountId){

int flag=0;

for(auto it = accounts.begin(); it != accounts.end(); it++){

if(it->getID() == accountId){

flag=1;

accounts.erase(it);

break;

}

}

if(flag)

return true;

else

return false;

}

std::string toString(){

string s = id + " " + name + " " + phone + "\n" ;

for(auto it = accounts.begin(); it != accounts.end(); it++){

s += it->toString() + "\n";

}

return s;

}

int getId(){

return id;

}

void setId(int id){

id=id;

}

string getName(){

return name;

}

string setName(string name){

name=name;

}

string getPhone(){

return phone;

}

string setPhone(string phone){

phone=phone;

}

};

int main(){

Client c[2] = {Client(100, "Ali", "123123123"), Client(200, "Deli", "456654546")};

c[0].addAccount(Account(1, 1000, 4.5));

c[0].addAccount(Account(2, 2000, 2.5));

c[1].addAccount(Account(3, 6000, 1.5));

c[1].addAccount(Account(4, 4000, 3.5));

c[1].addAccount(Account(5, 3000, 2.5));

cout<<c[0].toString()<<endl;

cout<<c[1].toString()<<endl;

cout<<endl;

return 0;

}

**Output:**

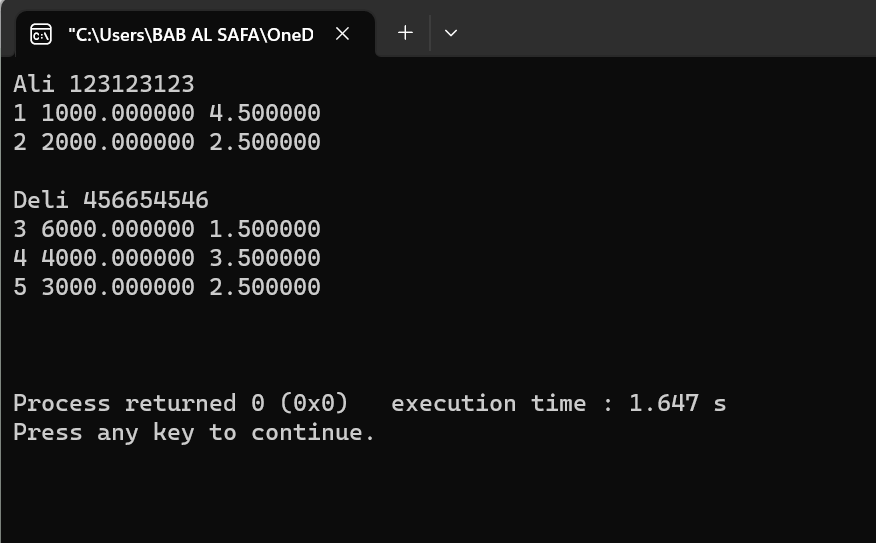


Fig 3.1: Output on console.

**Explanation:**

The above code Creates two class called Account and Client. The class Client creates a list of type Account and performs operations such as adding and removing accounts from the list.

Then inside the main function it creates an array of type Client and adds two accounts to for the first client and three accounts for the second client and prints the string representation of the accounts onto the console.

**Problem No.**: 04

**Problem Name**: Function and function prototype

**Code:**

#include<iostream>

using namespace std;

int sum(int, int);

int main(){

int a,b;

cin>>a>>b;

cout<<"sum = "<<sum(a,b)<<endl;

return 0;

}

int sum(int a, int b){

return a+b;

}

**Output:**

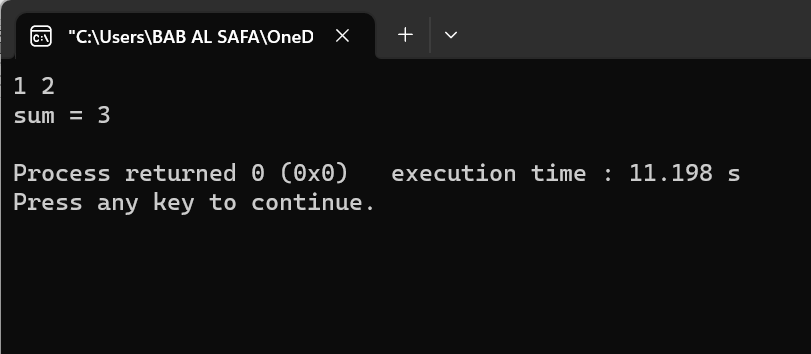


Fig 4.1: Output on console.

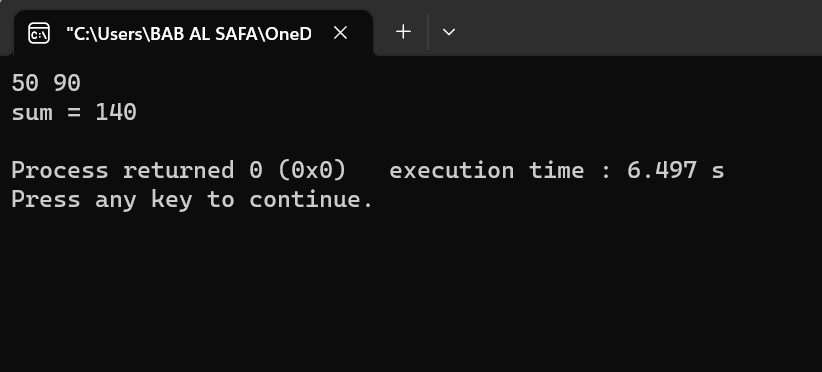


Fig 4.2: Output on console.

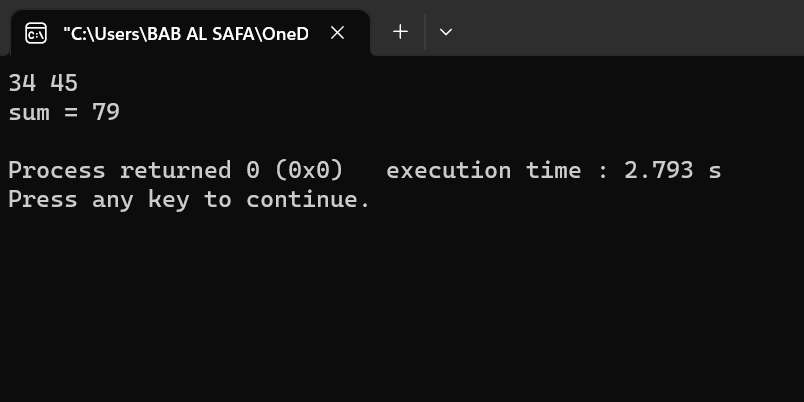


Fig 4.3: Output on console.

**Explanation:**

This code is a simple program that takes two integers as input and returns their sum. The function sum takes two integers as input and returns their sum.

Before the main function the function prototype for function sum is declared.

The main function is the entry point of the program. It takes two integers as input using the cin function and stores them in variables a and b. It then calls the sum function with these two variables as arguments and prints the result using the cout function.

**Problem No.**: 05

**Problem Name**: Program to find the size of the Union

**Code:**

#include <stdio.h>

union test1 {

int x;

int y;

} Test1;

union test2 {

int x;

char y;

} Test2;

union test3 {

int arr[10];

char y;

} Test3;

int main()

{

int size1 = sizeof(Test1);

int size2 = sizeof(Test2);

int size3 = sizeof(Test3);

printf("Sizeof test1: %d\n", size1);

printf("Sizeof test2: %d\n", size2);

printf("Sizeof test3: %d", size3);

return 0;

}

**Output:**

Sizeof test1: 4

Sizeof test2: 4

Sizeof test3: 40

**Explanation:**

The above code declares three unions named Test1, Test2 and Test3. The first union has two integer members named x and y. The second union has an integer member named x and a character member named y. The third union has an array of ten integers named arr and a character member named y. The driver code then uses the sizeof() operator to find the size of each union and prints it to the console.

**Problem No.**: 06

**Problem Name**: Program to check if its possible to store data in multiple Union

**Code:**

#include <stdio.h>

union test {

int x, y;

};

int main()

{

union test t;

t.x = 2;

printf("After making x = 2:\n x = %d, y = %d\n\n", t.x,

t.y);

t.y = 10;

printf("After making y = 10:\n x = %d, y = %d\n\n", t.x,

t.y);

return 0;

}

**Output:**

After making x = 2:

x = 2, y = 2

After making y = 10:

x = 10, y = 10

**Explanation:**

The code you provided declares a union named test with two integer members named x and y. The driver code then declares a union variable t and assigns the value 2 to its member x. Since x and y share the same memory location, y also gets the value 2. The program then prints the values of x and y. The driver code then assigns the value 10 to t’s member y. Since x and y share the same memory location, x is also updated to 10. The program then prints the values of x and y again.

**Problem No.**: 07

**Problem Name**: Program to illustrate differences between Structure and Union

**Code:**

#include <stdio.h>

#include <string.h>

struct struct\_example {

int integer;

float decimal;

char name[20];

};

union union\_example {

int integer;

float decimal;

char name[20];

};

void main()

{

struct struct\_example s = { 18, 38, "geeksforgeeks" };

union union\_example u = { 18, 38, "geeksforgeeks" };

printf("structure data:\n integer: %d\n"

"decimal: %.2f\n name: %s\n",

s.integer, s.decimal, s.name);

printf("\nunion data:\n integer: %d\n"

"decimal: %.2f\n name: %s\n",

u.integer, u.decimal, u.name);

printf("\nsizeof structure : %d\n", sizeof(s));

printf("sizeof union : %d\n", sizeof(u));

printf("\n Accessing all members at a time:");

s.integer = 183;

s.decimal = 90;

strcpy(s.name, "geeksforgeeks");

printf("structure data:\n integer: %d\n "

"decimal: %.2f\n name: %s\n",

s.integer, s.decimal, s.name);

u.integer = 183;

u.decimal = 90;

strcpy(u.name, "geeksforgeeks");

printf("\nunion data:\n integer: %d\n "

"decimal: %.2f\n name: %s\n",

u.integer, u.decimal, u.name);

printf("\n Accessing one member at time:");

printf("\nstructure data:");

s.integer = 240;

printf("\ninteger: %d", s.integer);

s.decimal = 120;

printf("\ndecimal: %f", s.decimal);

strcpy(s.name, "C programming");

printf("\nname: %s\n", s.name);

printf("\n union data:");

u.integer = 240;

printf("\ninteger: %d", u.integer);

u.decimal = 120;

printf("\ndecimal: %f", u.decimal);

strcpy(u.name, "C programming");

printf("\nname: %s\n", u.name);

printf("\nAltering a member value:\n");

s.integer = 1218;

printf("structure data:\n integer: %d\n "

" decimal: %.2f\n name: %s\n",

s.integer, s.decimal, s.name);

u.integer = 1218;

printf("union data:\n integer: %d\n"

" decimal: %.2f\n name: %s\n",

u.integer, u.decimal, u.name);

}

**Output:**

structure data:

integer: 18

decimal: 38.00

name: geeksforgeeks

union data:

integer: 18

decimal: 0.00

name: \_x0012\_

sizeof structure : 28

sizeof union : 20

Accessing all members at a time:structure data:

integer: 183

decimal: 90.00

name: geeksforgeeks

union data:

integer: 1801807207

decimal: 277322871721159507258114048.00

name: geeksforgeeks

Accessing one member at time:

structure data:

integer: 240

decimal: 120.000000

name: C programming

union data:

integer: 240

decimal: 120.000000

name: C programming

Altering a member value:

structure data:

integer: 1218

decimal: 120.00

name: C programming

union data:

integer: 1218

decimal: 0.00

name: �

**Explanation:**

This code defines two user-defined data types: `struct struct\_example` and `union union\_example`.

The structure `struct\_example` has three members: an integer named `integer`, a float named `decimal`, and a character array named `name`.

The union `union\_example` has three members: an integer named `integer`, a float named `decimal`, and a character array named `name`.

The code initializes both the structure and union with the same values. It then prints out the values of each member variable in both the structure and union. Finally, it changes the value of one member variable in each data type and prints out the new values.