**Task 01:**  
  
**Code :**  
  
#include<bits/stdc++.h>

using namespace std;

class BankAccount{

double balance;

public:

BankAccount(double balance=0) : balance(balance){}

BankAccount(BankAccount& other) : balance(other.balance){}

void showBalance() const{cout<<"Balance : "<<balance<<"$\n";}

void deductBalance(double amount){balance = max(0.0, balance - amount);}

friend ostream& operator<<(ostream& out, BankAccount &b);

};

ostream& operator<<(ostream& out, BankAccount &b){

out<<b.balance;

return out;

}

int main(){

BankAccount account1, account2(1000);

cout<<"Account 1"<<endl;

account1.showBalance();

cout<<"Account 2"<<endl;

account2.showBalance();

BankAccount account3(account2);

cout<<"Account 3"<<endl;

account3.showBalance();

cout<<"Deducting 200$ from Account 3";

account3.deductBalance(200);

cout<<"Account 2 (after copying into Account 3 showing no change)"<<endl;

account2.showBalance();

int\* ptr = new int(5);

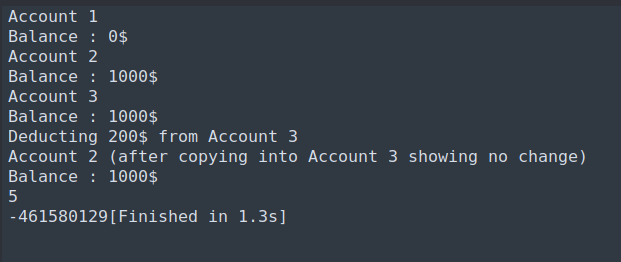
cout<<\*ptr<<endl;

delete ptr;

cout<<\*ptr;

return 0;

}



Output:

Task 02:

Code:

#include<bits/stdc++.h>

using namespace std;

class Exam{

string\* student\_name=nullptr;

string\* exam\_date=nullptr;

int\* score=nullptr;

public:

/\*

Since custom copy constructor is not built so the default one which performs shallow copy rather

than deep one because it will just copy the pointers of one object to another i.e pointing

to the same memory location instead of creating a new memory location for the second object

\*/

void setName(string s){

if(student\_name)

\*student\_name =s;

else

student\_name = new string(s);

}

void setExamDate(string s){

if(exam\_date)

\*exam\_date = s;

else

exam\_date = new string(s);

}

void setScore(int s){

if(score)

\*score = s;

else

score = new int(s);

}

void displayDetails() const{

cout<<"\nStudent Name : "<<\*student\_name;

cout<<"\nExam Date : "<<\*exam\_date;

cout<<"\nScore : "<<\*score;

}

/\* ~Exam(){

delete student\_name;

delete exam\_date;

delete score;

} \*/

};

int main(){

Exam s1;

s1.setName("AbdullahPrhaku");

s1.setScore(100);

s1.setExamDate("10/10/2005");

cout<<"\n Details of student 1";

s1.displayDetails();

Exam s2(s1);

cout<<"\n Details of student 2 (copied from student 1)";

s2.displayDetails();

s2.setName("AbdullahShantu");

s2.setExamDate("AbdullahShantu");

s2.setScore(101);

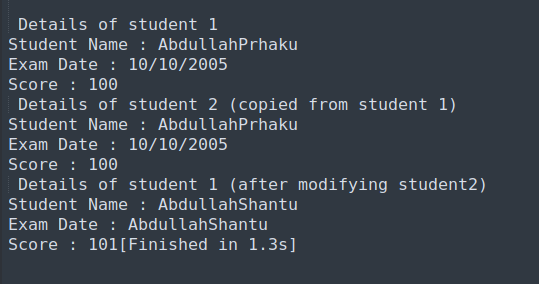
cout<<"\n Details of student 1 (after modifying student2)";

s1.displayDetails();

return 0;

}

Output:



Output only possible b/c of not including custom destructor or else same memory would have been tried to delete twice(due to shallow copy) resulting in segmentation fault.

Task 03:

Code:

#include<bits/stdc++.h>

using namespace std;

class Box{

int\* value;

public:

Box(int v){value = new int(v);}

// Deep Copy Constructor

Box(Box& o){

value = new int(\*(o.value));

}

// Shallow Copy Constructor

// Box(Box& o){

// value = o.value;

// }

~Box(){delete value;}

void operator=(Box& o){

value = new int(\*(o.value));

}

void modifyValue(int v){\*value = v;}

void printValue() const{cout<<"Value : "<<\*value<<endl;}

};

int main(){

Box b1(10);

Box b2(b1);

cout<<"Value of b1 :\n";

b1.printValue();

cout<<"Value of b2 :\n";

b2.printValue();

cout<<"Modifying b2 : \n";

b2.modifyValue(5);

b2.printValue();

cout<<"Value of b1 after modifying b2 :\n";

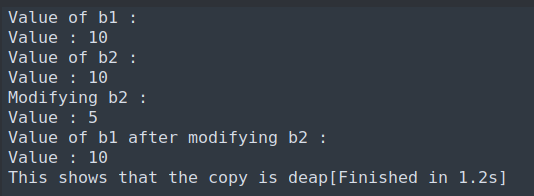
b1.printValue();

cout<<"This shows that the copy is deap";

return 0;

}

Output:



Task 04:

Code :

#include<bits/stdc++.h>

using namespace std;

class Product{

string name;

double price;

int stock;

public:

Product(){}

Product(string name, double price, int stock) : name(name), price(price), stock(stock){}

void displayDetails() const{

cout<<"\n Product Details:-";

cout<<"\n Name : "<<name;

cout<<"\n Price : "<<price;

cout<<"\n Stock : "<<stock;

}

};

class Bundle{

Product \*products=nullptr;

int productCount = 0;

double discount\_percentage=0;

public:

Bundle(Product \*list, int count, double discount){

products = new Product[count];

for(int i=0;i<count;i++) products[i] = list[i];

productCount = count;

discount\_percentage = discount;

}

// Deep copy

Bundle(Bundle& o){

products = new Product[o.productCount];

for(int i=0;i<o.productCount;i++) products[i] = o.products[i];

productCount = o.productCount;

discount\_percentage = o.discount\_percentage;

}

// Shallow Copy

// Bundle(Bundle& o){

// products=o.products;

// productCount = o.productCount;

// discount\_percentage = o.discount\_percentage;

// }

void applyDiscount(double d){discount\_percentage=d;}

void addProduct(Product p){

Product\* temp = new Product[productCount+1];

for(int i=0;i<productCount;i++) temp[i]=products[i];

temp[productCount++]=p;

delete[] products;

products=temp;

}

void display(){

cout<<"\n Discount : "<<discount\_percentage<<"%";

cout<<"\nDetails of Products in this bundle :-";

for(int i=0;i<productCount;i++){

cout<<"\n"<<i+1<<". ";

products[i].displayDetails();

}

}

~Bundle(){

delete[] products;

}

};

int main(){

Product\* p= new Product[3];

for(int i=1;i<=3;i++){

p[i-1] = Product("Product " + to\_string(i), i\*100, i\*5);

}

cout<<endl<<"Product List : ";

for(int i=0;i<3;i++) p[i].displayDetails();

Bundle b(p, 3, 0);

cout<<endl<<"Initital Bundle b";

b.display();

Bundle b2(b);

cout<<endl<<"Initital Bundle b2";

b2.display();

b2.addProduct(Product("Product 4", 400, 20));

cout<<endl<<"Bundle b2 after adding product in b2";

b2.display();

cout<<endl<<"Bundle b after adding product in b2";

b.display();

return 0;

}

Output:

