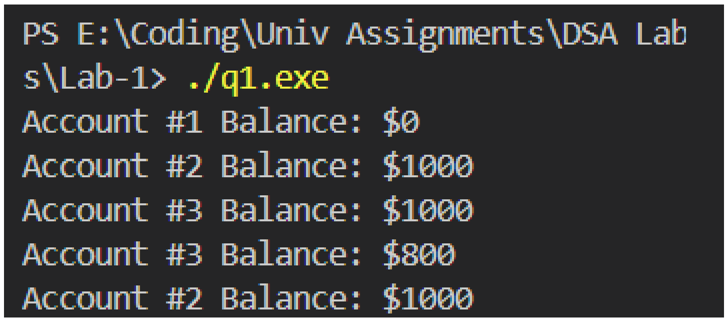
Q1:

```cpp
#include "iostream"
using namespace std;

class BankAccount // creating class
{
    double balance; // private var balance

public:
    BankAccount() : balance(0.0) {}                          // default
contructor
    BankAccount(double balance) { this->balance = balance; } // param constructor
    BankAccount(const BankAccount &other)                    // copy constructor
    {
        this->balance = other.balance;
    }
    void displayBalance() // method to display balance
    {
        cout << "Balance: $" << balance << endl;
    }
    double deposit(int cash) // method to deposite cash
    {
        balance += cash; // cash increment to balance
        return balance;
    }
    double withdraw(int cash) // method to withdraw cash
    {
        if (balance >= cash) // if balance is less than cash req thean a warning
instead of withdrawal
        {
            balance -= cash;
        }
        else
        {
            cout << "Insufficient Balance" << endl; // warning
        }
        return balance; // returns remaining balance
    }
};

int main()
```

```
{

    // demonstration of deep copy below

    BankAccount account1;

    cout << "Account #1 ";
    account1.displayBalance();

    BankAccount account2(1000);

    cout << "Account #2 ";
    account2.displayBalance();

    BankAccount account3 = account2;

    cout << "Account #3 ";
    account3.displayBalance();
    account3.withdraw(200);

    cout << "Account #3 ";
    account3.displayBalance();

    cout << "Account #2 ";
    account2.displayBalance();

    return 0;
}
```

Output:



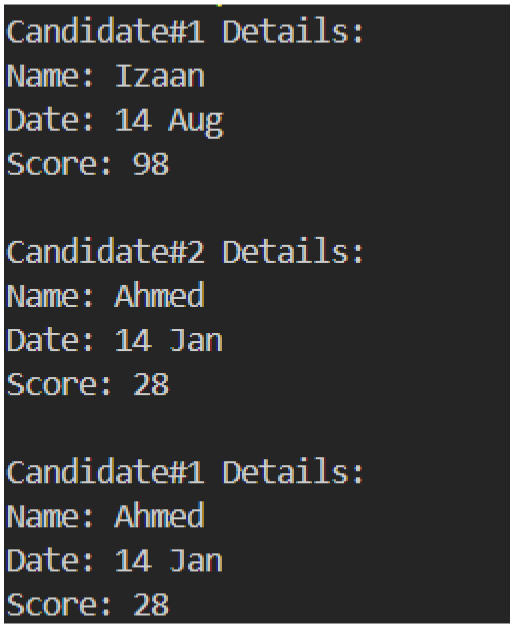
```
PS E:\Coding\Univ Assignments\DSA Lab
s\Lab-1> ./q1.exe
Account #1 Balance: $0
Account #2 Balance: $1000
Account #3 Balance: $1000
Account #3 Balance: $800
Account #2 Balance: $1000
```

Q2:

```
#include "iostream"
```

```cpp
#include "cstring"
using namespace std;

class Exam
{
    string *Name;
    string *Date;
    int *Score;

public:
    Exam() // default constructor that performs DMA
    {
        Score = new int;
        Name = new string;
        Date = new string;
    }
    //below setters for all pointers
    void setName(string name) { *Name = name; }
    void setDate(string date) { *Date = date; }
    void setScore(int score) { *Score = score; }

    void displayExam() // display function to display exam details
    {
        cout << "Name: " << *Name << endl;
        cout << "Date: " << *Date << endl;
        cout << "Score: " << *Score << endl;
    }

    ~Exam() // Destructor to free memory
    {
        delete Score;
        delete Name;
        delete Date;
    }
};

int main()
{

    //demonstration of shallow copy (result of no copy constructor)

    Exam candidate1;
    candidate1.setName("Izaan");
    candidate1.setDate("14 Aug");
    candidate1.setScore(98);
    cout << "Candidate#1 Details:" << endl;
    candidate1.displayExam();
    cout << endl;

    Exam candidate2 = candidate1;
    candidate2.setName("Ahmed");
    candidate2.setDate("14 Jan");
    candidate2.setScore(28);
    cout << "Candidate#2 Details:" << endl;
    candidate2.displayExam();
    cout << endl;
    cout << "Candidate#1 Details:" << endl;
    candidate1.displayExam();
    cout << endl;

    return 0;
}
```

Output:



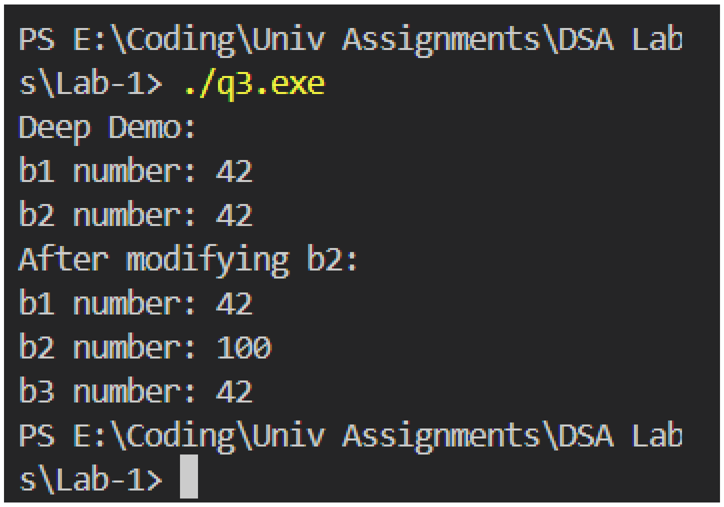
```
Candidate#1 Details:
Name: Izaan
Date: 14 Aug
Score: 98

Candidate#2 Details:
Name: Ahmed
Date: 14 Jan
Score: 28

Candidate#1 Details:
Name: Ahmed
Date: 14 Jan
Score: 28
```

Q3:

```cpp
#include <iostream>
using namespace std;
```

```cpp
class Box
{
    int *number;

public:
    Box() // default constructor performing DMA on the one pointer
    {
        number = new int;
    }
    // getter & setter below
    void setNumber(int value) { *number = value; }
    int getNumber() const { return *number; }

    ~Box() // destructor to free memory
    {
        delete number;
    }

    Box(const Box &other) // copy constructor to ensure deep copies
    {
        number = new int(*other.number);
    }

    Box &operator=(const Box &other) // move assignment operator to move one object to another completely
    {
        if (this == &other)
        {
            return *this;
        }

        delete number; // The memory of the pervious object occupation is freed
        number = new int(*other.number);
        return *this;
    }
};

int main()
{

    // demonstration of the deep copy below
    cout << "Deep Demo:" << endl;
    Box b1;
    b1.setNumber(42);

    Box b2 = b1;
    cout << "b1 number: " << b1.getNumber() << endl;
    cout << "b2 number: " << b2.getNumber() << endl;

    b2.setNumber(100);
    cout << "After modifying b2:" << endl;
    cout << "b1 number: " << b1.getNumber() << endl;
    cout << "b2 number: " << b2.getNumber() << endl;

    Box b3;
    b3 = b1;
    cout << "b3 number: " << b3.getNumber() << endl;

    return 0;
}
```

Output:



```
PS E:\Coding\Univ Assignments\DSA Lab
s\Lab-1> ./q3.exe
Deep Demo:
b1 number: 42
b2 number: 42
After modifying b2:
b1 number: 42
b2 number: 100
b3 number: 42
PS E:\Coding\Univ Assignments\DSA Lab
s\Lab-1> 
```