```
/*
   ASSIGNMENT 4              QUESTION 1
Design the class(es) for the following. Each account has an account number and balance amount. A list of accounts is to be maintained where one can add and find accounts, display information of all accounts. While adding, the account number must be unique. Withdraw object has account number(must exist) and amount(will not exceed balance amount of corresponding account). Withdraw object will update the balance of the corresponding account in the list. Users will be able to search and view accounts, add accounts and withdraw money from the account. Implement your design. Use friend function wherever required and again modify your implementation to avoid friend function.

   NAME : Anirudh Modi          ROLL : 002310501031        DATE : 10/11/2024
*/

#include <iostream>
using namespace std;
class ACCLIST; // Forward declaration
class ACC
{
   int acc_no;
   double amt;
   // Private method to display account details
   void details()
      cout << "Account number: " << acc_no << " Amount: " << amt << endl;
   // ACCLIST can access private members of ACC
   friend ACCLIST;
};
class TRANS
{
   double amount;
   int acc_no;
public:
   TRANS(int ac, double am)
   {
      amount = am;
      acc_no = ac;
   }
   // ACCLIST can access private members of TRANS
   friend ACCLIST;
};
class ACCLIST
{
   ACC list[10000];
   int num_acc;
public:
   ACCLIST()
```

```
    num_acc = 0;
// Add a new account if the account number is unique
void add_acc(int num, double amt)
{
    for (int i = 0; i < num_acc; i++)
    {
        if (list[i].acc_no == num)
        {
            cout << "Account already exists." << endl;
            return;
        }
    }
    list[num_acc].acc_no = num;
    list[num_acc].amt = amt;
    num_acc++;
    cout << "Account added successfully." << endl;
}
// Display account details for a given account number
void display_acc_info(int num)
{
    for (int i = 0; i < num_acc; i++)
    {
        if (list[i].acc_no == num)
        {
            list[i].details();
            return;
        }
    }
    cout << "Account not found." << endl;
}
// Withdraw money from an account
void withdraw(TRANS &ref)
{
    for (int i = 0; i < num_acc; i++)
    {
        if (list[i].acc_no == ref.acc_no)
        {
            if (ref.amount < 0)
            {
                cout << "Amount cannot be negative." << endl;
                return;
            }
            if (list[i].amt < ref.amount)
            {
                cout << "Insufficient balance." << endl;
                return;
            }
```

```
            list[i].amt -= ref.amount;
            cout << "Withdrawal successful. Remaining balance: " << list[i].amt <<
endl;
            return;
        }
    }
    cout << "Account not found." << endl;
}
// Deposit money into an account
void deposit(TRANS &ref)
{
    for (int i = 0; i < num_acc; i++)
    {
        if (list[i].acc_no == ref.acc_no)
        {
            if (ref.amount < 0)
            {
                cout << "Amount cannot be negative." << endl;
                return;
            }
            list[i].amt += ref.amount;
            cout << "Deposit successful. New balance: " << list[i].amt << endl;
            return;
        }
    }
    cout << "Account not found." << endl;
}
};
int main()
{
    ACCLIST accList;
    int choice;
    int acc_no;
    double amt;
    while (true)
    {
        // Display menu options
        cout << "\nMenu:\n";
        cout << "1. Add Account\n";
        cout << "2. Display Account Info\n";
        cout << "3. Withdraw Money\n";
        cout << "4. Deposit Money\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice)
        {
```

```
        case 1: // Add Account
            cout << "Enter account number: ";
            cin >> acc_no;
            cout << "Enter initial balance: ";
            cin >> amt;
            accList.add_acc(acc_no, amt);
            break;
        case 2: // Display Account Info
            cout << "Enter account number to view details: ";
            cin >> acc_no;
            accList.display_acc_info(acc_no);
            break;
        case 3: // Withdraw Money
            cout << "Enter account number for withdrawal: ";
            cin >> acc_no;
            cout << "Enter amount to withdraw: ";
            cin >> amt;
            {
                TRANS withdraw(acc_no, amt);
                accList.withdraw(withdraw);
            }
            break;
        case 4: // Deposit Money
            cout << "Enter account number for deposit: ";
            cin >> acc_no;
            cout << "Enter amount to deposit: ";
            cin >> amt;
            {
                TRANS deposit(acc_no, amt);
                accList.deposit(deposit);
            }
            break;
        case 5: // Exit
            cout << "Exiting program...\n";
            return 0;
        default:
            cout << "Invalid choice, please try again.\n";
            break;
        }
    }
    return 0;
}
```

```
Menu:
1. Add Account
2. Display Account Info
3. Withdraw Money
4. Deposit Money
5. Exit
Enter your choice: 1
Enter account number: 1
Enter initial balance: 500
Account added successfully.

Menu:
1. Add Account
2. Display Account Info
3. Withdraw Money
4. Deposit Money
5. Exit
Enter your choice: 2
Enter account number to view details: 1
Account number: 1 Amount: 500

Menu:
1. Add Account
2. Display Account Info
3. Withdraw Money
4. Deposit Money
5. Exit
Enter your choice: 3
Enter account number for withdrawal: 1
Enter amount to withdraw: 600
Insufficient balance.
```

```
Menu:
1. Add Account
2. Display Account Info
3. Withdraw Money
4. Deposit Money
5. Exit
Enter your choice: 4
Enter account number for deposit: 2
Enter amount to deposit: 12
Account not found.

Menu:
1. Add Account
2. Display Account Info
3. Withdraw Money
4. Deposit Money
5. Exit
Enter your choice: 5
Exiting program...
```

```
/*
   ASSIGNMENT 4                QUESTION 2
Design a COMPLEX class, which will behave like normal integer with respect to
                    ● addition,
                    ● subtraction,
                    ● accepting the value and
                    ● Displaying the value.
   NAME : Anirudh Modi          ROLL : 002310501031        DATE : 10/11/2024
*/
#include<bits/stdc++.h>
using namespace std;
class Complex{
   int r, im;
   public:
      Complex(int real = 0, int imag = 0){
         r = real;     im = imag;
      }
      Complex operator+(Complex c){
         Complex t;
         t.r = r + c.r;      t.im = im + c.im;
         return t;
      }
      Complex operator-(Complex c){
          Complex t;
          t.r = r - c.r;      t.im = im - c.im;
          return t;
       }
       friend ostream& operator<<(ostream& out, const Complex& c){
         out << c.r << " + " << c.im << "i";
         return out;
       }
       friend istream& operator>>(istream& in, Complex& c) {
         cout << "Enter real and imaginary parts: ";
         in >> c.r >> c.im;
         return in;
      }
};
int main(){
   Complex c1, c2, c3, c4;       cin>>c1>>c2;
   c3 = c1 + c2;         c4 = c1 - c2;
   cout<<c3<<"\n"<<c4<<"\n";
}
```

```
Enter real and imaginary parts: 7 9
Enter real and imaginary parts: 3 5
10 + 14i
4 + 4i
```

```
/*
   ASSIGNMENT 4              QUESTION 3
Design an ARRAY class with the following features:
    a. Array objects may be declared for a specific size and a value for initialising
       all the elements. If this is to be assumed as a 0.
    b. An array object may be declared and initialised with another object.
    c. An array object may be declared and initialised with another array (not the
       object, standard array as in C language).
Let a and b are two objects:
I.   a + b will add corresponding elements.
II.  a = b will do the assignment.
III. a[I] will return the ith element of the object.
IV.  a*5 or 5*a will multiply the element with 5.
   NAME : Anirudh Modi          ROLL : 002310501031        DATE : 10/11/2024
*/

#include <iostream>
using namespace std;

class ARRAY {
   int* arr;
   int size;
public:
   // Constructor with default value
   ARRAY(int s = 0, int def = 0) {
      if(s == 0){
         arr = NULL;
      }
      else{
         size = s;
         arr = new int[size];
         for (int i = 0; i < size; i++) {
            arr[i] = def;
         }
      }
   }
   // Copy constructor
   ARRAY(const ARRAY& temp) {
      size = temp.size;
      arr = new int[size];
      for (int i = 0; i < size; i++) {
         arr[i] = temp.arr[i];
      }
   }
   // Copy from STL array
   ARRAY(int* stlArr, int s) {
      size = s;
      arr = new int[size];
```

```
    for (int i = 0; i < size; i++) {
        arr[i] = stlArr[i];
    }
}
// Assignment operator overloading
ARRAY& operator=(const ARRAY& temp) {
    if (this != &temp) {
        delete[] arr;
        size = temp.size;
        arr = new int[size];
        for (int i = 0; i < size; i++) {
            arr[i] = temp.arr[i];
        }
    }
    return *this;
}
// Overloading the + operator
ARRAY operator+(const ARRAY& temp) const {
    ARRAY res(size);
    if (size == temp.size) {
        for (int i = 0; i < size; i++) {
            res.arr[i] = arr[i] + temp.arr[i];
        }
    } else {
        cerr << "Array of different sizes, addition not possible\n";
    }
    return res;
}
// Overloading the [] operator
int& operator[](int index) {
    if (index >= 0 && index < size) {
        return arr[index];
    } else {
        cerr << "Index out of bounds!" << endl;
        static int dummy = -1;
        return dummy;
    }
}
// Overloading the * operator for scalar multiplication
ARRAY operator*(int scalar) const {
    ARRAY res(size);
    for (int i = 0; i < size; i++) {
        res.arr[i] = arr[i] * scalar;
    }
    return res;
}
// Overloading the * operator for scalar on the left
```

```
    friend ARRAY operator*(int scalar, const ARRAY& tmp) {
        ARRAY res(tmp.size);
        for (int i = 0; i < tmp.size; i++) {
            res.arr[i] = tmp.arr[i] * scalar;
        }
        return res;
    }
    // Destructor
    ~ARRAY() {
        delete[] arr;
    }
    // Display function
    void display() const {
        cout << "arr[" << size << "]: ";
        for (int i = 0; i < size; i++) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }
};
int main() {
    ARRAY a1(5, 7);
    a1.display();
    ARRAY a2(a1);
    a2.display();
    int arr[5] = {0, 1, 2, 5, 7};
    ARRAY a3(arr, 5);
    a3.display();
    ARRAY a4 = a2;
    a4.display();
    ARRAY a5 = a1 + a3;
    a5.display();
    ARRAY a6 = 5 * a5;
    a6.display();
    ARRAY a7 = a5 * 5;
    a7.display();
    cout<<a7[4];
    return 0;
}
```

```
arr[5]: 7 7 7 7 7
arr[5]: 7 7 7 7 7
arr[5]: 0 1 2 5 7
arr[5]: 7 7 7 7 7
arr[5]: 7 8 9 12 14
arr[5]: 35 40 45 60 70
arr[5]: 35 40 45 60 70
70
```

```
/*
   ASSIGNMENT 4                QUESTION 4
    1. Design a STRING class, which will have the initialization facility similar to array class.
       Provide support for
            • Assigning one object for another,
            • Two string can be concatenated using + operator,
            • Two strings can be compared using the relational operators.

   NAME : Anirudh Modi          ROLL : 002310501031        DATE : 10/11/2024
*/

#include <bits/stdc++.h>
using namespace std;

class STRING {
   char *st;
   int size;

public:
   STRING(int s = 0, char c = '/') {
      if (s == 0) {
         st = NULL;
         size = 0;
      } else {
         size = s;
         st = new char[size];
         for (int i = 0; i < size; i++) {
            st[i] = c;
         }
      }
   }

   // Destructor to free memory
   ~STRING() {
      delete[] st;
   }

   // Copy constructor
   STRING(const STRING &temp) {
      size = temp.size;
      if (temp.st == NULL) {
         st = NULL;
      } else {
         st = new char[size];
         for (int i = 0; i < size; i++) {
            st[i] = temp.st[i];
         }
```

```
    }
}

// Copy from another string
STRING(const char *str, int s) {
    size = s;
    st = new char[size];
    for (int i = 0; i < size; i++) {
        st[i] = str[i];
    }
}

STRING &operator=(const STRING &temp) {
    if (this != &temp) {
        delete[] st;
        if (temp.st == NULL) {
            st = NULL;
            size = 0;
        } else {
            size = temp.size;
            st = new char[size];
            for (int i = 0; i < size; i++)
                st[i] = temp.st[i];
        }
    }
    return *this;
}

STRING operator+(const STRING temp) {
    STRING res(size + temp.size, '/');
    for (int i = 0; i < size; i++) {
        res.st[i] = st[i];
    }
    for (int i = size; i < size + temp.size; i++) {
        res.st[i] = temp.st[i - size];
    }
    return res;
}

int operator>(const STRING temp) {
    if (size > temp.size)
        return 1;
    if (size < temp.size)
        return 0;
    for (int i = 0; i < size; i++) {
        if (st[i] < temp.st[i])
            return 0;
```

```
            else if (st[i] > temp.st[i])
                return 1;
        }
        return 2; // Equality
    }

    void display() {
        if (size == 0)
            cout << "nothing to display\n";
        for (int i = 0; i < size; i++) {
            cout << st[i];
        }
        cout << "\n";
    }
};

int main() {
    STRING s1(5, 'a');
    s1.display();

    STRING s2;
    s2 = s1;
    s2.display();

    STRING s3("anirudh", 7);
    s3.display();

    STRING s4;
    s4 = s2 + s3;
    s4.display();

    int i = s2 > s3;
    cout<<i;
}
```

```
aaaaa
aaaaa
anirudh
aaaaaanirudh
0
```

```
/*
   ASSIGNMENT 4                QUESTION 5
```
Modify the STRING class so that assigning/initialising a string by another will not copy it physically but will keep a reference count, which will be incremented. Reference value 0 means the space can be released

```
   NAME : Anirudh Modi          ROLL : 002310501031        DATE : 10/11/2024
*/

#include<iostream>
#include<math.h>

using namespace std;
class String{
    int size;
    char* str;
    int* pcnt;
    public:
        String(int s = 0,char val = '\0'){
            size = s;
            str = new char[s];
            pcnt = new int;
            for(int i = 0; i < size; i++){
                str[i] = val;
            }
        }
        String(const String& t){
            size = t.size;
            pcnt = t.pcnt;
            str = t.str;
            *pcnt = *pcnt + 1;
        }
        String(char astr[],int n){
            size = n;
            str = new char[size];
            for(int i = 0; i < size; i++){
                str[i] = astr[i];
            }
        }
        void operator=(String);
        void operator+(String);
        bool operator==(String);
        void display();
        ~String(){
            *pcnt = *pcnt - 1;
            if(*pcnt == 0){
                delete pcnt;
```

```
            delete[] str;
        }
    }
};
void String::operator+(String t){
    char temp[size];
    for(int i = 0; i < size; i++){
        temp[i] = str[i];
    }
    int end = size + t.size;
    str = new char[end];
    int i = 0;
    for(i = 0; i < size; i++){
        str[i] = temp[i];
    }
    for(int j = 0; j < t.size; j++){
        str[i] = t.str[j];
        i++;
    }
    size = end;
    return;
}
void String::operator=(String t){
    *pcnt = *pcnt - 1;
    if(*pcnt == 0){
        delete pcnt;
        delete[] str;
    }
    pcnt = t.pcnt;
    str = t.str;
    *pcnt = *pcnt + 1;
    return;
}
bool String::operator==(String t){
    if(size != t.size){
        return false;
    }
    for(int i = 0; i < size; i++){
        if(str[i] != t.str[i]){
            return false;
        }
    }
    return true;
}

void String::display(){
    cout<<"String is :- ";
```

```
    for(int i = 0; i < size; i++){
        cout<<str[i];
    }
    cout<<endl;
    return;
}
int main(){
    char arr[6] = {'a','b','c','d','e','f};
    String a(5),b(5,'c'),c(b),d(arr,6);
    a.display();
    b.display();
    c.display();
    d.display();
    cout<<(c==b)<<endl;
    b + d;
    b.display();
    a = d;
    a.display();
    cout<<(c==b)<<endl;
    return 0;
}
```

```
String is :- .....
String is :- ccccc
String is :- ccccc
String is :- abcdef
1
String is :- cccccabcdef
String is :- abcde
0
```