Discrete Structure (12/07/2021)

Submitted to:
Professor Shalini Gupta

Submitted by:
Sufiyan Adam
88035

Ans 1:

```
#include <iostream>
#include <set>
#include <math.h>
using namespace std;
int isMember(multiset<int> A)
{
    cout << "\nEnter an element to find in the set\n";
    int c;
    cin >> c;
    if (A.count(c))
        cout << "The element is present\n";
    else
        cout << "There is no such element present\n";
}
int powerSet(multiset<int> A)
{
    cout<<"The Power set is: "<<endl;
    for (int counter = 0; counter < pow(2, A.size()); counter++)
    {
        for (int j = 0; j < A.size(); j++)
        {
            if (counter & (1 << j))
            {
                auto first = A.begin();
                std::advance(first, j);
                cout << *first;
            }
        }
        cout << endl;
    }
}
int dispSet(multiset<int> A)
{
    cout << "Entered set is: \n";
    for (auto it = A.begin(); it != A.end(); ++it)
        cout << " " << *it;
    cout << endl;
    cout << "Cardinality of entered set is:";
```

```
    cout << A.size();
    return 0;
}
int inputSet(multiset<int> A)
{
    int c = 1;
    cout << "Enter the elements in the set- \n";
    while (c)
    {
        int n;
        cin >> n;
        A.insert(n);
        cout << "Do you want to enter more?<0/1>\n";
        cin >> c;
    }
    dispSet(A);
    isMember(A);
    powerSet(A);
    return 0;
}
int main()
{
    multiset<int> A;
    inputSet(A);
    return 0;
}
```

Ans 2:

```
#include<iostream>
using namespace std;
class SET{
    private:
        int* setA;
        int* lengthOfSetA;
        int* setB;
        int* lengthOfSetB;
        int* intersectionSet;
        int* lengthOfIntersectionSet;
        int* unionSet;
        int* lengthOfUnionSet;
        int* universalSet;
        int* lengthOfUniversalSet;
        int* complementSet;
        int* lengthOfComplementSet;
        int* setA_SetB;
        int* lengthOfsetA_SetB;
        int* setB_SetA;
        int* lengthOfsetB_SetA;
        int* symmetricDifferenceSet;
```

```
        int* lengthOfsymmetricDifferenceSet;
        string** cartesianSet;

    public:
        SET(void);
        void line(void);
        void setData(int* &setA,int* &setB,int* &lengthOfSetA , int* &lengthOfSetB);
        void makeItSet();
        void display(int* &set ,int* &lengthOfSet);
        void display();
        void intersectionOfTwoSet();
        bool subSet();
        void unionOfTwoSet();
        void unionOfTwoSet(int* &setA,int* &setB,int* &lengthOfSetA , int* &lengthOfSetB);
        void complementOfSet(int* &set , int* &lengthOfSet);
        void cartesianProduct();
        void DiffSetA_SetB();
        void DiffSetB_SetA();
        void symmetricDifference();
        void executeCode(SET* &user);
        void mainMenu(void);
        void secondSubMenu(SET* &user);
        void thirdSubMenu(SET* &user);
        void fourthSubMenu(SET* &user);

};
SET::SET(){

    this->universalSet = new int[10];
    for (int i = 0; i < 10; i++)
    {
        universalSet[i] = i+1;
    }

    this->lengthOfUniversalSet=new int(10);
    //-----------------------------------


}
void SET::setData(int* &setA,int* &setB,int* &lengthOfSetA , int* &lengthOfSetB){
    this->setA = setA;
    this->setB = setB;
    this->lengthOfSetA=lengthOfSetA;
    this->lengthOfSetB=lengthOfSetB;
}
bool SET::subSet(){

    for (int i = 0; i < *lengthOfSetB; i++)
    {
        bool flag = false;
        for (int j = 0; j<*lengthOfSetA; j++)
        {
            if (setB[i]==setA[j])
            {
                flag=true;
```

```
            }
        }
        if(!flag){
            return false;
        }

    }
    return true;

}
void SET::unionOfTwoSet(){
    this->unionSet = new int[*lengthOfSetA+*lengthOfSetB];
    this->lengthOfUnionSet = new int(0);
    for (int i = 0; i < *lengthOfSetA; i++){
        unionSet[i] = setA[i];
        *lengthOfUnionSet+=1;
    }
    for (int i = 0; i < *lengthOfSetB; i++){
        bool flag = true;
        for(int j = 0; j < *lengthOfSetA; j++){
            if (setB[i]==setA[j])
            {
                flag=false;
            }

        }
        if(flag){
            unionSet[*lengthOfUnionSet] = setB[i];
            *lengthOfUnionSet+=1;
        }
    }
    display(unionSet,lengthOfUnionSet);


}
void SET::unionOfTwoSet(int* &setA,int* &setB,int* &lengthOfSetA , int* &lengthOfSetB){
    this->unionSet = new int[*lengthOfSetA+*lengthOfSetB];
    this->lengthOfUnionSet = new int(0);
    for (int i = 0; i < *lengthOfSetA; i++){
        unionSet[i] = setA[i];
        *lengthOfUnionSet+=1;
    }
    for (int i = 0; i < *lengthOfSetB; i++){
        bool flag = true;
        for(int j = 0; j < *lengthOfSetA; j++){
            if (setB[i]==setA[j])
            {
                flag=false;
            }

        }
        if(flag){
            unionSet[*lengthOfUnionSet] = setB[i];
            *lengthOfUnionSet+=1;
```

```
        }
    }
    display(unionSet,lengthOfUnionSet);


}
void SET::intersectionOfTwoSet(){
    this->lengthOfIntersectionSet=new int(0);
    this->intersectionSet = new int[*lengthOfSetA];
    //-----------------------------------------------
    for(int i=0 ; i<*lengthOfSetA ; i++){
        bool flag  = false;
        for(int j =0 ; j<*lengthOfSetB ; j++){
            if(this->setA[i] == this->setB[j]){
                flag=true;
                break;
            }
        }
        if(flag){
            this->intersectionSet[*lengthOfIntersectionSet] = this->setA[i];
            *(lengthOfIntersectionSet)+=1;
        }
    }
    display(intersectionSet,lengthOfIntersectionSet);
}
void SET::makeItSet(){
    int* lengthOfUniqueSetA=new int(0);
    int* lengthOfUniqueSetB=new int(0);
    int* uniqueSetA = new int[*lengthOfSetA];
    int* uniqueSetB = new int[*lengthOfSetB];
    //---------------------------------------
    for(int i=0 ; i<*lengthOfSetA ; i++){
        bool flag  = true;
        for(int j =0 ; j<*lengthOfSetA ; j++){
            if(setA[i]==uniqueSetA[j]){
                flag=false;
                break;
            }
        }
        if(flag){
            uniqueSetA[*(lengthOfUniqueSetA)] = setA[i];
            *(lengthOfUniqueSetA)+=1;
        }
    }
    //---------------------------------------
    for(int i=0 ; i<*lengthOfSetB ; i++){
        bool flag  = true;
        for(int j =0 ; j<*lengthOfSetB ; j++){
            if(setB[i]==uniqueSetB[j]){
                flag=false;
                break;
            }
        }
        if(flag){
```

```
            uniqueSetB[*(lengthOfUniqueSetB)] = setB[i];
            *(lengthOfUniqueSetB)+=1;
        }
    }
    //---------------------------------------
    this->lengthOfSetA = lengthOfUniqueSetA;
    this->lengthOfSetB = lengthOfUniqueSetB;
    this->setA         = uniqueSetA;
    this->setB         = uniqueSetB;
    line();
    cout<<"SET-A    : ";
    display(setA,lengthOfSetA);
    cout<<"SET-B    : ";
    display(setB,lengthOfSetB);
}
void SET::complementOfSet(int* &set , int* &lengthOfSet){
    this->lengthOfComplementSet = new int(0);
    this->complementSet=new int[*lengthOfUniversalSet];
    display(universalSet,lengthOfUniversalSet);
    for (int i = 0; i < *lengthOfUniversalSet; i++)
    {
        bool check = false;
        for(int j=0 ; j< *lengthOfSet ;j++){
            if(universalSet[i]==set[j]){
                check=true;
                break;
            }
        }
        if(check){
            continue;
        }
        complementSet[*lengthOfComplementSet] = universalSet[i];
        *lengthOfComplementSet+=1;
    }
    display(complementSet,lengthOfComplementSet);
}
void SET::cartesianProduct(){
    cartesianSet = new string*[*lengthOfSetA];
    for (int i = 0; i < *lengthOfSetA; i++)
    {
        cartesianSet[i] = new string[*lengthOfSetB];
    }

    for (int i = 0; i < *lengthOfSetA; i++)
    {
        for (int j = 0; j < *lengthOfSetB; j++)
        {
            cartesianSet[i][j]=to_string(setA[i])+","+to_string(setB[j]);
        }

    }
    display();
}
void SET::display(int* &set ,int* &lengthOfSet){
```

```
    int* pointerOfSet = set;
    cout<<"[";
    for(int i = 0 ; i<*lengthOfSet ; i++){
        cout<<*pointerOfSet++;
        if(i!=(*lengthOfSet-1)){
            cout<<" ,";
        }

    }
    cout<<"]"<<endl;
}
void SET::display(){
    cout<<"[ "<<endl;
    for (int i = 0; i < *lengthOfSetA; i++)
    {
        for (int j = 0; j < *lengthOfSetB; j++)
        {
            cout<< "(" << cartesianSet[i][j]<<")";
            if(i!= *lengthOfSetA-1 || j!= *lengthOfSetB-1 ){
                cout<<" ,";
            }
        }
        cout<<endl;

    }
    cout<<"]"<<endl;
}
void SET::DiffSetA_SetB(){
    setA_SetB          = new int[*lengthOfSetA];
    lengthOfsetA_SetB  = new int(0);
    for (int i = 0; i < *lengthOfSetA; i++)
    {
        bool notPresent = true;
        for (int j = 0; j < *lengthOfSetB; j++)
        {
            if(setA[i]==setB[j]){
                notPresent=false;
                break;
            }
        }
        if(notPresent){
            setA_SetB[*lengthOfsetA_SetB]=setA[i];
            *lengthOfsetA_SetB+=1;
        }

    }
    display(setA_SetB,lengthOfsetA_SetB);

}
void SET::DiffSetB_SetA(){
    setB_SetA          = new int[*lengthOfSetB];
    lengthOfsetB_SetA  = new int(0);
    for (int i = 0; i < *lengthOfSetB; i++)
    {
```

```
        bool notPresent = true;
        for (int j = 0; j < *lengthOfSetA; j++)
        {
            if(setB[i]==setA[j]){
                notPresent=false;
                break;
            }
        }
        if(notPresent){
            setB_SetA[*lengthOfsetB_SetA]=setB[i];
            *lengthOfsetB_SetA+=1;
        }

    }
    display(setB_SetA,lengthOfsetB_SetA);

}
void SET::symmetricDifference(){
    unionOfTwoSet(setA_SetB,setB_SetA,lengthOfsetA_SetB,lengthOfsetB_SetA);
}
void SET::executeCode(SET* &user){
    cout<<"---------------------------------------"<<endl;
    user->mainMenu();
    int choice ;
    cin.clear();
    // cin.ignore();
    cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"<<endl;
    cout<<"Please enter your choice : ";cin>>choice;
    cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"<<endl;
    cin.ignore();

    switch (choice) {
        case 1:
            if(user->subSet()){
                cout<<"Yes, Set(B) is subset of Set(A) "<<endl;
            }
            else{
                cout<<"No,Set(B) isn't a subset of Set(A) "<<endl;
            }
            executeCode(user);
            break;
        case 2:
            secondSubMenu(user);
            executeCode(user);
            break;
        case 3:
            thirdSubMenu(user);
            executeCode(user);
            break;
        case 4:
            fourthSubMenu(user);
            executeCode(user);
            break;
        case 5:
```

```
            user->cartesianProduct();
            executeCode(user);
            break;
        case 6:
            exit(0);
            break;
        default:
            cout<<"- - - - - - - - - "<<endl;
            cout<<"Invalid Choice!!"<<endl;
            cout<<"- - - - - - - - - "<<endl;
            break;
    }
    // cin.ignore();

    cin.clear();
    cin.ignore();
    // executeCode(user);



}
void SET::mainMenu(void){
    cout<<"1.Check whether SET-B is a subset of SET-A?"<<endl;
    cout<<"2.Union or Intersection of two Sets."<<endl;
    cout<<"3.Complement of a SET."<<endl;
    cout<<"4.Set difference and Symmetric difference of two Set."<<endl;
    cout<<"5.Cartesian product of Set."<<endl;
    cout<<"6.Exit."<<endl;
}
void SET::secondSubMenu(SET* &user){
    cout<<"1.Union  of two Sets."<<endl;
    cout<<"2.Intersection of two Sets."<<endl;
    int subChoice;
    // cin.ignore();
    cin.clear();
    cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"<<endl;
    cout<<"Please enter your choice : ";cin>>subChoice;
    cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"<<endl;
    if(subChoice==1){
        user->unionOfTwoSet();
    }
    else if(subChoice==2){
        user->intersectionOfTwoSet();
    }
    else{
        cout<<"- - - - - - - - - "<<endl;
        cout<<"Invalid Choice!!"<<endl;
        cout<<"- - - - - - - - - "<<endl;
        cin.clear();
        cin.ignore();
        secondSubMenu(user);
    }
}
void SET::thirdSubMenu(SET* &user){
```

```
    cout<<"1.Complement of Set(A)."<<endl;
    cout<<"2.Complement of Set(B)."<<endl;
    int subChoice;
    cin.clear();
    // cin.ignore();
    cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"<<endl;
    cout<<"Please enter your choice : ";cin>>subChoice;
    cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"<<endl;
    if(subChoice==1){
        user->complementOfSet(setA,lengthOfSetA);
    }
    else if(subChoice==2){
        user->complementOfSet(setB,lengthOfSetB);
    }
    else{
        cout<<"- - - - - - - - - "<<endl;
        cout<<"Invalid Choice!!"<<endl;
        cout<<"- - - - - - - - - "<<endl;
        cin.clear();
        cin.ignore();
        thirdSubMenu(user);
    }
}
void SET::fourthSubMenu(SET* &user){
    cout<<"1.Set(A) - Set(B)."<<endl;
    cout<<"2.Set(B) - Set(A)."<<endl;
    int subChoice;
    cin.clear();
    cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"<<endl;
    cout<<"Please enter your choice : ";cin>>subChoice;
    cout<<"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"<<endl;
    if(subChoice==1){
        user->DiffSetA_SetB();
    }
    else if(subChoice==2){
        user->DiffSetB_SetA();
    }
    else{
        cout<<"- - - - - - - - - "<<endl;
        cout<<"Invalid Choice!"<<endl;
        cout<<"- - - - - - - - - "<<endl;
        cin.clear();
        cin.ignore();
        fourthSubMenu(user);
    }
}
void SET::line(){
    cout<<"---------------------------------------"<<endl;
}
int main(){
    int arr1[] = {5,2,3,1};
    int arr2[] = {1,2,6,8,7};
    int* parr1 = arr1;
    int* parr2 = arr2;
```

```
    int* l1    = new int(4);
    int* l2    = new int(5);
    SET* user  = new SET();
    user->setData(parr1,parr2,l1,l2);
    user->makeItSet();
    user->executeCode(user);
    return 0;
}
```

Ans 3&4:

```
#include <iostream>
using namespace std;
class Relation
{
public:
    int f = 0;
    int rel[4][4] = {{1, 1, 1},
                     {1, 1, 1},
                     {1, 1, 1}};
    int checkReflexive()
    {
        for (int i = 0; i < 3; i++)
        {
            if (rel[i][0] != 1)
            {
                cout << "Not reflexive ";
                return 0;
            }
        }
        cout << "Reflexive ";
        f++;
        return 0;
    }
    int checkSymmetric()
    {
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                if (rel[i][j] == 1 && rel[j][i] != 1)
                {
                    cout << "Not symmetric ";
                    return 0;
                }
            }
        }
        cout << "Symmetric ";
        f++;
        return 0;
    }
```

```
int checkAntiSymmetric()
{
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            if (rel[i][j] == 1 && rel[j][i] != 1)
            {
                cout << "Antisymmetric ";
                f++;
                return 0;
            }
        }
    }
    cout << "Not Antisymmetric ";
    return 0;
}
int checkTransitive()
{
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
        {
            for (int k = 0; k < 3; k++)
            {
                if (rel[i][j] && rel[j][k] && !rel[i][k])
                {
                    cout << "Not Transitive ";
                    return 0;
                }
            }
        }
    }
    f++;
    cout << "Transitive ";
    return 0;
}
int checkEqui()
{
    checkReflexive();
    cout << ", ";
    checkSymmetric();
    cout << "and ";
    checkTransitive();
    if (f == 3)
    {
        cout << "so in conclusion the relation is an Equivalence relation";
    }
    else
    {
        cout << "so in conclusion the relation is not an Equivalence relation";
    }
}
int checkParOrder()
```

```
    {
        checkReflexive();
        cout << ", ";
        checkAntiSymmetric();
        cout << "and ";
        checkTransitive();
        if (f == 3)
        {
            cout << "so in conclusion the relation is a Partial order relation";
        }
        else
        {
            cout << "so in conclusion the relation is not a Partial order relation";
        }
    }
};
int main()
{
    Relation r;
    int n;
    while (n != 7)
    {
        cout << "\nWhat do you want to do with this relation?\n";
        cout << "1.Check Symmetry\n";
        cout << "2.Check Reflexivity\n";
        cout << "3.Check Antisymmetry\n";
        cout << "4.Check Transitivity\n";
        cout << "5.Check Equivalence\n";
        cout << "6.Check Partial Order\n";
        cout << "7.Exit\n";
        cin >> n;
        switch (n)
        {
        case (1):
            cout << "The relation is ";
            r.checkSymmetric();
            break;
        case (2):
            cout << "The relation is ";
            r.checkReflexive();
            break;
        case (3):
            cout << "The relation is ";
            r.checkAntiSymmetric();
            break;
        case (4):
            cout << "The relation is ";
            r.checkTransitive();
            break;
        case (5):
            cout << "The relation is ";
            r.checkEqui();
            break;
        case (6):
```

```
            cout << "The relation is ";
            r.checkParOrder();
            break;
        case (7):
            cout << "Thank You!";
            break;
        default:
            cout << "Please enter a number between 1-5\n";
        }
    }
}
```

Ans 13:

```
#include <iostream>
#include <set>
#include <stdio.h>
#include <iomanip>
using namespace std;
int exOR()
{
    cout << setw(5) << "x" << setw(5) << "|" << setw(5) << "y" << setw(5) << "|" << setw(1
0) << "x XOR y\n";
    cout << "   -------------------------\n";
    for (int i = 0; i <= 1; i++)
    {
        for (int j = 0; j <= 1; j++)
        {
            cout << setw(5) << i << setw(5) << "|" << setw(5) << j << setw(5) << "|";
            cout << "    ";
            printf("%d", i ^ j);
            cout << "\n";
        }
    }
    return 0;
}
int conj()
{
    cout << setw(5) << "x" << setw(5) << "|" << setw(5) << "y" << setw(5) << "|" << setw(1
0) << "x AND y\n";
    cout << "   -------------------------\n";
    for (int i = 0; i <= 1; i++)
    {
        for (int j = 0; j <= 1; j++)
        {
            cout << setw(5) << i << setw(5) << "|" << setw(5) << j << setw(5) << "|";
            cout << "    ";
            printf("%d", i && j);
            cout << "\n";
        }
    }
    return 0;
```

```
}
int disj()
{
    cout << setw(5) << "x" << setw(5) << "|" << setw(5) << "y" << setw(5) << "|" << setw(1
0) << "x OR y\n";
    cout << "   -------------------------\n";
    for (int i = 0; i <= 1; i++)
    {
        for (int j = 0; j <= 1; j++)
        {
            cout << setw(5) << i << setw(5) << "|" << setw(5) << j << setw(5) << "|";
            cout << "    ";
            printf("%d", i || j);
            cout << "\n";
        }
    }
    return 0;
}
int cond()
{
    cout << setw(5) << "x" << setw(5) << "|" << setw(5) << "y" << setw(5) << "|" << setw(1
0) << "x --> y\n";
    cout << "   -------------------------\n";
    for (int i = 0; i <= 1; i++)
    {
        for (int j = 0; j <= 1; j++)
        {
            cout << setw(5) << i << setw(5) << "|" << setw(5) << j << setw(5) << "|";
            cout << "    ";
            printf("%d", !i || j);
            cout << "\n";
        }
    }
    return 0;
}
int biCond()
{
    cout << setw(5) << "x" << setw(5) << "|" << setw(5) << "y" << setw(5) << "|" << setw(1
0) << "x <--> y\n";
    cout << "   -------------------------\n";
    for (int i = 0; i <= 1; i++)
    {
        for (int j = 0; j <= 1; j++)
        {
            cout << setw(5) << i << setw(5) << "|" << setw(5) << j << setw(5) << "|";
            cout << "    ";
            printf("%d", ((!i || j) && (!j || i)));
            cout << "\n";
        }
    }
    return 0;
}
int exNOR()
{
```

```
    cout << setw(5) << "x" << setw(5) << "|" << setw(5) << "y" << setw(5) << "|" << setw(1
0) << "x XOR y\n";
    cout << "   -------------------------\n";
    for (int i = 0; i <= 1; i++)
    {
        for (int j = 0; j <= 1; j++)
        {
            cout << setw(5) << i << setw(5) << "|" << setw(5) << j << setw(5) << "|";
            cout << "     ";
            printf("%d", !(i ^ j));
            cout << "\n";
        }
    }
    return 0;
}
int neg()
{
    cout << setw(5) << "x" << setw(5) << "|" << setw(7) << "x'\n";
    cout << "   -------------\n";
    for (int i = 0; i <= 1; i++)
    {
        cout << setw(5) << i << setw(5) << "|";
        cout << "     ";
        printf("%d", !i);
        cout << "\n";
    }
    return 0;
}
int nand()
{
    cout << setw(5) << "x" << setw(5) << "|" << setw(5) << "y" << setw(5) << "|" << setw(1
0) << "x NAND y\n";
    cout << "   -------------------------\n";
    for (int i = 0; i <= 1; i++)
    {
        for (int j = 0; j <= 1; j++)
        {
            cout << setw(5) << i << setw(5) << "|" << setw(5) << j << setw(5) << "|";
            cout << "     ";
            printf("%d", !(i && j));
            cout << "\n";
        }
    }
    return 0;
}
int nor()
{
    cout << setw(5) << "x" << setw(5) << "|" << setw(5) << "y" << setw(5) << "|" << setw(1
0) << "x NOR y\n";
    cout << "   -------------------------\n";
    for (int i = 0; i <= 1; i++)
    {
        for (int j = 0; j <= 1; j++)
        {
```

```
            cout << setw(5) << i << setw(5) << "|" << setw(5) << j << setw(5) << "|";
            cout << "    ";
            printf("%d", !(i || j));
            cout << "\n";
        }
    }
    return 0;
}
int main()
{
    conj();
    cout << "\n\n";
    disj();
    cout << "\n\n";
    exOR();
    cout << "\n\n";
    cond();
    cout << "\n\n";
    biCond();
    cout << "\n\n";
    exNOR();
    cout << "\n\n";
    neg();
    cout << "\n\n";
    nand();
    cout << "\n\n";
    nor();
}
```

Ans 14:

```
#include <iostream>

using namespace std;
int recurrenceOne(int n)
{
    if (n == 0)
        return 1;
    return recurrenceOne(n - 1) + n;
}
int recurrenceTwo(int n)
{
    if (n == 0)
        return 1;
    return recurrenceTwo(n - 1) + n * n;
}
int recurrenceThree(int n)
{
    if (n == 1)
        return 1;
```

```
    return 2 * recurrenceThree(n / 2) + n;
}

int main()
{
    int n, ch;
    cout << "Choose Recurrence Relation to Evaluate:\n"
         << "  (1) T(n) = T(n - 1) + n and T(0) = 1\n"
         << "  (2) T(n) = T(n - 1) + n^2 and T(0) = 1\n"
         << "  (3) T(n) = 2 * T(n / 2) + n and T(1) = 1\n";
    cout << "Enter Choice: ";
    cin >> ch;
    switch (ch)
    {
    case 1:
        cout << "\nEnter Value of n: ";
        cin >> n;
        cout << "\nValues for T(n) = T(n - 1) + n:\n";
        for (int i = n; i >= 0; i--)
        {
            if (i == 0)
                cout << "T(0) = " << recurrenceOne(i)
                     << endl;
            else
                cout << "T(" << i << ") = T(" << (i - 1)
                     << ") + " << i << " = "
                     << recurrenceOne(i)
                     << endl;
        }
        break;
    case 2:
        cout << "\nEnter Value of n: ";
        cin >> n;
        cout << "\nValues for T(n) = T(n - 1) + n^2:\n";
        for (int i = n; i >= 0; i--)
        {
            if (i == 0)
                cout << "T(0) = " << recurrenceTwo(i)
                     << endl;
            else
                cout << "T(" << i << ") = T(" << (i - 1)
                     << ") + " << i * i << " = "
                     << recurrenceTwo(i)
                     << endl;
        }
        break;
    case 3:
        cout << "\nEnter Value of n: ";
        cin >> n;
        cout << "\nValues for T(n) = 2 * T(n / 2) + n:\n";
        for (int i = n; i >= 1; i--)
        {
            if (i == 1)
                cout << "T(1) = " << recurrenceThree(i)
```

```
                    << endl;
            else
                cout << "T(" << i << ") = 2 * T(" << i
                    << " / 2) + " << i << " = "
                    << "2 * T(" << (i / 2)
                    << ") + " << i << " = "
                    << recurrenceThree(i)
                    << endl;
        }
        break;
    default:
        cout << "\nInvalid Choice!\n";
        break;
    }
    return 0;
}
```