**Task 1**

**Question1.cpp**

#include"question 1.h"

//Question 1

//--------------------------------------------

//1. The Total Number of Relations

//2. Reflexive Relations

//3. Irreflexive Relations

//4. Symmetric Relations

//5. Anti - Symmetric Relations

//6. Asymmetric Relations

int main() {

int size1 = 0, size2 = 0, size3 = 0;

int\* set1 = nullptr; int\* set2 = nullptr; int\* set3 = nullptr;

char count = 'NULL';

do

{

cout << "------------------Menu--------------------" << endl;

cout << "Press 1 to create a sets" << endl;

cout << "Press 2 the Total Number of Relations" << endl;

cout << "Press 3 to check a Reflexive relation" << endl;

cout << "Press 4 to check an Irreflexive relation" << endl;

cout << "Press 5 to check a Symmetric relation" << endl;

cout << "Press 6 to check an Anti-Symmetric relation" << endl;

cout << "Press 7 to check an Asymmetric relation" << endl;

cout << "Press 8 to display cartesian" << endl;

cout << "Press 0 to exit" << endl;

cout << "\nEnter a value: ";

cin >> count;

cout << endl;

switch (count)

{

case '1':

cout << "Enter size of set 1: ";

cin >> size1;

cout << "Enter size of set 2: ";

cin >> size2;

cout << endl;

set1 = new int[size1];

set2 = new int[size2];

cout << "Enter the values for set 1" << endl;

inputSet(set1, size1);

cout << "Enter the values for set 2" << endl;

inputSet(set2, size2);

break;

case '2':

if (size1 <= 0 || size2 <= 0)

{

cout << "Relations not created. Please create relation first." << endl;

break;

}

for (int i = 0; i < size1; i++)

{

for (int j = 0; j < size2; j++) {

size3++; size3++;

}

}

cout << "The total number of relations are: " << size3 / 2 << endl;

break;

case '3':

if (size1 <= 0 || size2 <= 0)

{

cout << "Relations not created. Please create relation first." << endl;

break;

}

reflexive(set1, set2, size1, size2);

break;

case '4':

if (size1 <= 0 || size2 <= 0)

{

cout << "Relations not created. Please create relation first." << endl;

break;

}

Irreflexive(set1, set2, size1, size2);

break;

case '5':

if (size1 <= 0 || size2 <= 0)

{

cout << "Relations not created. Please create relation first." << endl;

break;

}

symmetric(set1, set2, size1, size2);

break;

case '6':

if (size1 <= 0 || size2 <= 0)

{

cout << "Relations not created. Please create relation first." << endl;

break;

}

antiSymmetric(set1, set2, size1, size2);

break;

case '7':

if (size1 <= 0 || size2 <= 0)

{

cout << "Relations not created. Please create relation first." << endl;

break;

}

asymmetric(set1, set2, size1, size2);

break;

case '8':

if (size1 <= 0 || size2 <= 0)

{

cout << "Relations not created. Please create relation first." << endl;

break;

}

cartesianDisplay(set1, set2, size1, size2);

break;

case '0':

break;

default:

cout << "Are you Mad, you entered a wrong value!!" << endl;

break;

}

} while (count != '0');

delete[] set1;

delete[] set2;

delete[] set3;

return 0;

}

**question 1.h**

#pragma once

#include<iostream>

using namespace std;

/\*============================================================================\*/

//Question 1

//--------------------------------------------

//1. The Total Number of Relations

//2. Reflexive Relations

//3. Irreflexive Relations

//4. Symmetric Relations

//5. Anti - Symmetric Relations

//6. Asymmetric Relations

/\*--------------------------------------------------- Inputing -------------------------------------------\*/

//input elements into a set

template<typename T>

void inputSet(T\* set, int size) {

if (size < 3) {

cout << "Size must be 3 or above." << endl;

return;

}

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

cout << "Enter element " << i + 1 << ": ";

cin >> set[i];

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

if (set[i] == set[j]) {

cout << "Duplicate value! Please enter a different value.\n";

--i;

break;

}

}

}

}

/\*------------------------------------------------------- end ---------------------------------------------\*/

/\*---------------------------------------------------- Reflexive ------------------------------------------\*/

//Reflexive

void reflexive(const int\* set1, const int\* set2, int size1, int size2)

{

int Reflexivecheck = 0;

for (int i = 0; i < size1; i++)

{

for (int j = 0; j < size2; j++)

{

if (set1[i] == set2[j])

{

Reflexivecheck = 1;

break;

}

else {

Reflexivecheck = 0;

}

}

}

if (Reflexivecheck)

{

cout << "Relation is Reflexive." << endl;

}

else

{

cout << "Relation is not Reflxive." << endl;

}

}

/\*-------------------------------------------------------- end --------------------------------------------\*/

/\*---------------------------------------------------- Irreflexive ----------------------------------------\*/

//Irreflexive

void Irreflexive(const int\* set1, const int\* set2, int size1, int size2) {

int irreflexivecheck = 1;

for (int i = 0; i < size1; i++)

{

for (int j = 0; j < size2; j++)

{

if (set1[i] == set2[j])

{

irreflexivecheck = 0;

break;

}

else {

irreflexivecheck = 0;

}

}

}

// checking reflexive relation

if (irreflexivecheck)

{

cout << "Relation is Irreflexive." << endl;

}

else

{

cout << "Relation is not irreflxive." << endl;

}

}

/\*------------------------------------------------------ end ---------------------------------------------\*/

/\*---------------------------------------------------- Symmetric ------------------------------------------\*/

//Symmetric

void symmetric(const int\* set1, const int\* set2, int size1, int size2)

{

int SymmetricCheck = 0;

for (int i = 0; i < size1; i++)

{

for (int j = 0; j < size2; j++)

{

if ((set1[i] == set2[j] && set1[i + 1] == set2[j + 1]) ||

(set1[i] == set2[j + 1] && set1[i + 1] == set2[j]))

{

SymmetricCheck = 1;

break;

}

else

{

SymmetricCheck = 0;

}

}

}

// checking symmetric relation

if (SymmetricCheck)

{

cout << "Relation is symmetric." << endl;

}

else

{

cout << "Relation is not symmetric." << endl;

}

}

/\*---------------------------------------------------- end ------------------------------------------------\*/

/\*----------------------------------------------- Anti-Symmetric ------------------------------------------\*/

//Anti-symmetric

void antiSymmetric(const int\* set1, const int\* set2, int size1, int size2)

{

int AntiSymmetricCheck = 1;

for (int i = 0; i < size1; i++)

{

for (int j = 0; j < size2; j++)

{

if ((set1[i] == set2[j] && set1[i + 1] == set2[j + 1]) ||

(set1[i] == set2[j + 1] && set1[i + 1] == set2[j]))

{

AntiSymmetricCheck = 0;

break;

}

}

}

// checking antisymmetric relation

if (AntiSymmetricCheck)

{

cout << "Relation is anti-symmetric." << endl;

}

else

{

cout << "Relation is not anti-symmetric." << endl;

}

}

/\*---------------------------------------------------- end ------------------------------------------\*/

/\*---------------------------------------------------- Asymmetric ----------------------------------------\*/

// Asymmetric

void asymmetric(const int\* set1, const int\* set2, int size1, int size2)

{

int asymmetricCheck = 1;

for (int i = 0; i < size1; i++)

{

for (int j = 0; j < size2; j++)

{

if (set1[i] == set2[j])

{

asymmetricCheck = 0;

break;

}

}

if (!asymmetricCheck)

{

break;

}

}

// Checking asymmetric relation

if (asymmetricCheck)

{

cout << "Relation is asymmetric." << endl;

}

else

{

cout << "Relation is not asymmetric." << endl;

}

}

/\*---------------------------------------------------- end ------------------------------------------\*/

/\*---------------------------------------------------- Display ------------------------------------------\*/

//Cartesian product Display

void cartesianDisplay(const int\* set1, const int\* set2, int size1, int size2) {

cout << "{";

for (int i = 0; i < size1; i++)

{

for (int j = 0; j < size2; j++)

{

cout << "(" << set1[i] << "," << set2[j] << ") ,";

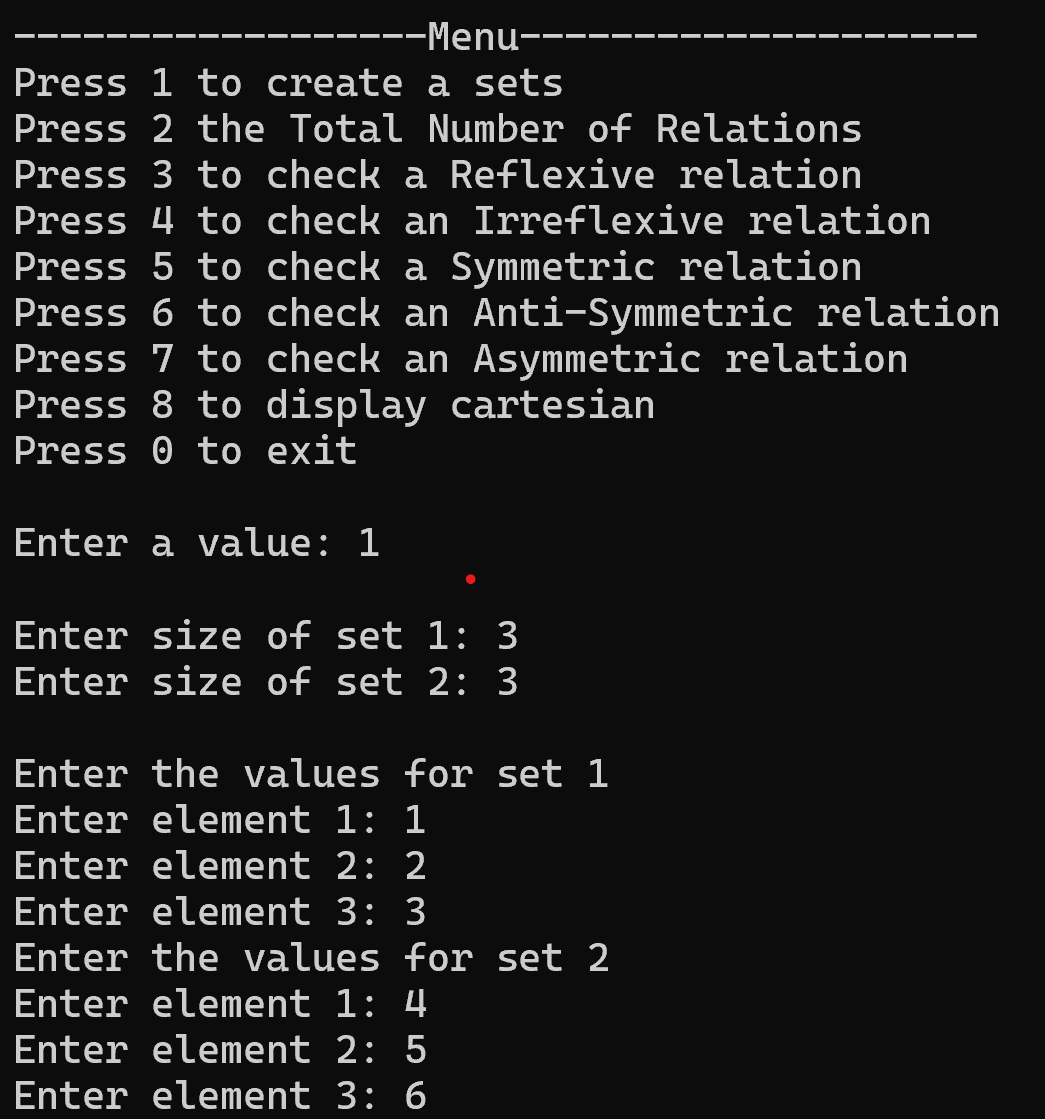
}

}

cout << "}" << endl;

}

**Output**

****

**A screenshot of a computer program

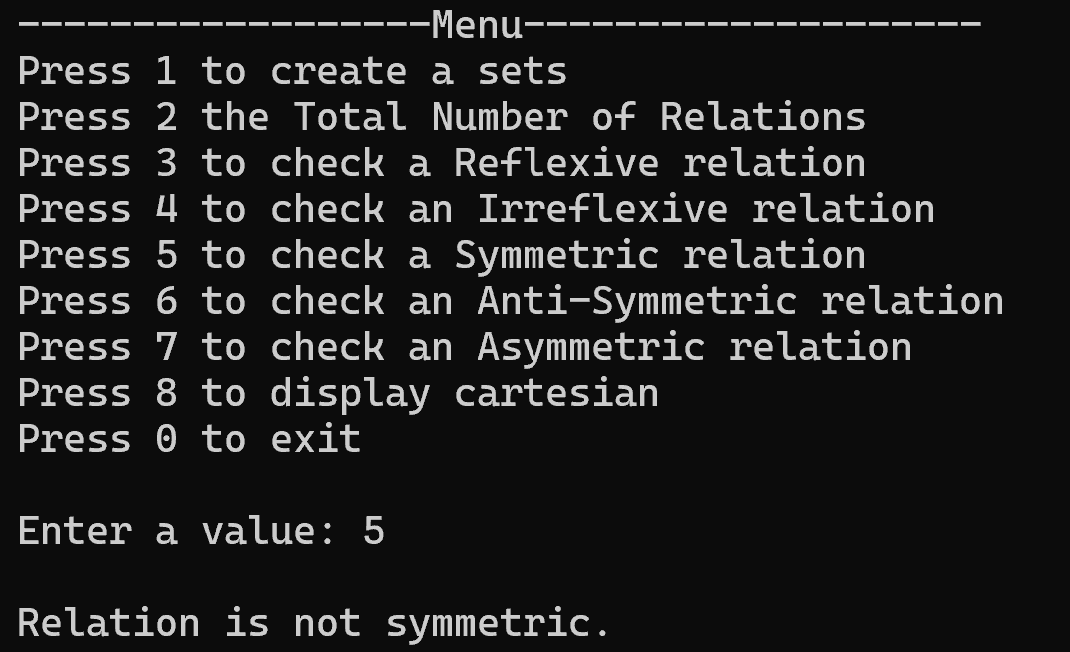
Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

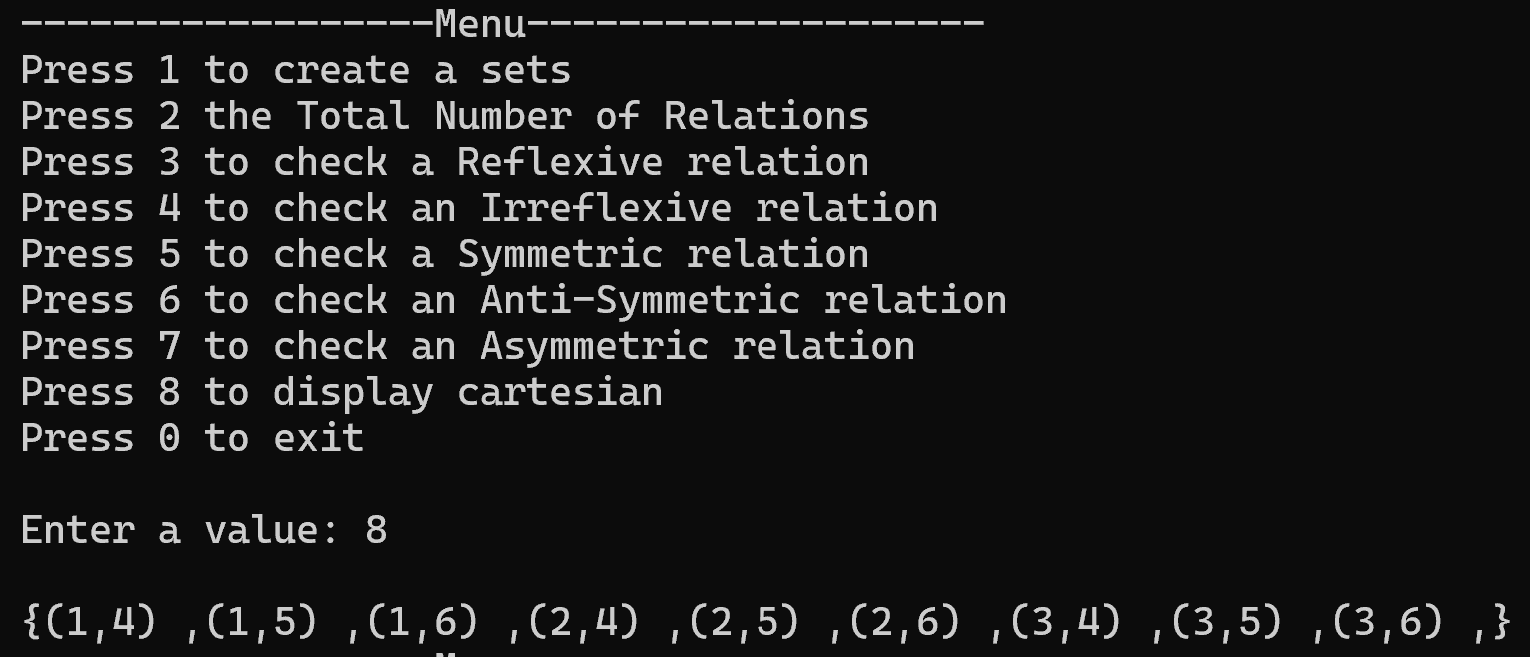
****

**A screenshot of a computer program

Description automatically generated**

**A screenshot of a computer program

Description automatically generated**

****

**A screenshot of a computer program

Description automatically generated**

**Task 2**

**Question2.cpp**

#include"question 2.h"

// Question 2

//--------------------------------------------

//1. Set Creation and Display

//2. Union and Intersection

//3. Subset and Superset

//4. Set Difference

//5. Power Set Generation

//6. Cardinality Calculation

//7. Set Equality

//8. Set Complement

// Prototype for saving data to a file

void saveData(const char\* set1, int size1, const char\* set2, int size2, const string& filename);

// Prototype for loading data from a file

void loadData(char\*& set1, int& size1, char\*& set2, int& size2, const string& filename);

int main() {

char\* set1 = nullptr;

char\* set2 = nullptr;

char\* set3 = nullptr;

char\* universalSet = nullptr;

int size1 = 0, size2 = 0, size3 = 0, universalSize = 0;

int count = 0;

string filename = "data.txt";

do

{

cout << "------------------Menu--------------------" << endl;

cout << "Press 1 to create a set and Display sets" << endl;

cout << "Press 2 to create and display Union of set or Intersection of set" << endl;

cout << "Press 3 to create and display Subset of set or Superset of set" << endl;

cout << "Press 4 to create and display Set Difference of set" << endl;

cout << "Press 5 to create and display Power Set Generation of set" << endl;

cout << "Press 6 to create and display Cardinality Calculation of set" << endl;

cout << "Press 7 to create and display Set Equality of set" << endl;

cout << "Press 8 to create and display Set Complement of set" << endl;

cout << "Press 9 to get data from file" << endl;

cout << "Press 0 to upload data in File" << endl;

cout << "Press -1 to exit" << endl;

cout << "\nEnter a value: ";

cin >> count;

cout << endl;

switch (count)

{

case 1:

cout << "Enter size of set 1: ";

cin >> size1;

cout << "Enter size of set 2: ";

cin >> size2;

cout << endl;

set1 = new char[size1];

set2 = new char[size2];

cout << "Enter the values for set 1" << endl;

inputSet(set1, size1);

cout << "Enter the values for set 2" << endl;

inputSet(set2, size2);

setDisplay(set1, set2, size1, size2);

break;

case 2:

if (size1 <= 0 || size2 <= 0)

{

cout << "Relations not created. Please create relation first." << endl;

break;

}

setUnion(set1, size1, set2, size2);

setIntersection(set1, size1, set2, size2);

fileUnion(set1, size1, set2, size2, "union.txt");

fileIntersection(set1, size1, set2, size2, "intersection.txt");

break;

case 3:

if (size1 <= 0 || size2 <= 0)

{

cout << "Relations not created. Please create relation first." << endl;

break;

}

if (subset(set2, size2, set1, size1))

cout << "Set 2 is a subset of Set 1." << endl;

else

cout << "Set 2 is not a subset of Set 1." << endl;

if (superset(set1, size1, set2, size2))

cout << "Set 1 is a superset of Set 2." << endl;

else

cout << "Set 1 is not a superset of Set 2." << endl;

break;

case 4:

if (size1 <= 0 || size2 <= 0)

{

cout << "Relations not created. Please create relation first." << endl;

break;

}

setDifference(set1, size1, set2, size2);

break;

case 5:

set3 = new char[size3];

cout << "Enter size of set 3: ";

cin >> size3;

cout << "Enter the values of set to find power set" << endl;

inputSet(set3, size3);

powerSet(set3, size3);

break;

case 6:

if (size1 <= 0 || size2 <= 0)

{

cout << "Relations not created. Please create relation first." << endl;

break;

}

cout << "Cardinality of the set 1: ";

cardinality(set1, size1);

cout << "Cardinality of the set 2: ";

cardinality(set2, size2);

break;

case 7:

if (size1 <= 0 || size2 <= 0)

{

cout << "Relations not created. Please create relation first." << endl;

break;

}

if (setEquality(set1, size1, set2, size2))

cout << "The sets are equal.\n";

else

cout << "The sets are not equal.\n";

break;

case 8:

if (size1 <= 0 || size2 <= 0)

{

cout << "Relations not created. Please create relation first." << endl;

break;

}

universalSet = new char[universalSize];

cout << "Enter size of universal set: ";

cin >> universalSize;

cout << "Enter the values of set to find Compliment of sets" << endl;

inputSet(universalSet, universalSize);

cout << endl;

cout << "Complement of set 1 is: ";

setComplement(universalSet, set1, universalSize, size1);

cout << endl;

cout << "Complement of set 2 is: ";

setComplement(universalSet, set2, universalSize, size2);

break;

case 9:

loadData(set1, size1, set2, size2, filename);

break;

case 0:

saveData(set1, size1, set2, size2, filename);

break;

case -1:

break;

default:

cout << "Are you Mad, you entered a wrong value!!" << endl;

break;

}

} while (count != -1);

delete[] set1;

delete[] set2;

delete[] set3;

delete[] universalSet;

return 0;

}

//Definition of saveData function

void saveData(const char\* set1, int size1, const char\* set2, int size2, const string& filename) {

ofstream outputFile(filename);

if (!outputFile.is\_open()) {

cout << "Error opening file: " << filename << endl;

return;

}

//Save the size of sets and then the elements

outputFile << size1 << endl;

for (int i = 0; i < size1; ++i)

outputFile << set1[i] << " ";

outputFile << endl;

outputFile << size2 << endl;

for (int i = 0; i < size2; ++i)

outputFile << set2[i] << " ";

outputFile.close();

cout << "Data saved successfully to " << filename << endl;

}

//Definition of loadData function

void loadData(char\*& set1, int& size1, char\*& set2, int& size2, const string& filename) {

ifstream inputFile(filename);

if (!inputFile.is\_open()) {

cout << "Error opening file: " << filename << endl;

return;

}

//Read and display the size of sets

inputFile >> size1;

cout << "Size of set 1: " << size1 << endl;

set1 = new char[size1];

cout << "Elements of set 1: ";

for (int i = 0; i < size1; ++i) {

inputFile >> set1[i];

cout << set1[i] << " ";

}

cout << endl;

inputFile >> size2;

cout << "Size of set 2: " << size2 << endl;

set2 = new char[size2];

cout << "Elements of set 2: ";

for (int i = 0; i < size2; ++i) {

inputFile >> set2[i];

cout << set2[i] << " ";

}

cout << endl;

inputFile.close();

cout << "Data taken successfully from " << filename << endl;

}

**question 2.h**

**#pragma once**

**#include<iostream>**

**#include <fstream>**

**using namespace std;**

**/\*============================================================================\*/**

**// Question 2**

**//--------------------------------------------**

**//1. Set Creation and Display**

**//2. Union and Intersection**

**//3. Subset and Superset**

**//4. Set Difference**

**//5. Power Set Generation**

**//6. Cardinality Calculation**

**//7. Set Equality**

**//8. Set Complement**

**/\*------------------------------------------ Inputing & displaying -------------------------------------\*/**

**//input elements into a set(array)**

**template<typename T>**

**void inputSet(T\* set, int size) {**

**if (size < 0) {**

**cout << "Size must be 3 or above." << endl;**

**return;**

**}**

**for (int i = 0; i < size; ++i) {**

**cout << "Enter element " << i + 1 << ": ";**

**cin >> set[i];**

**for (int j = 0; j < i; ++j) {**

**if (set[i] == set[j]) {**

**cout << "Duplicate value! Please enter a different value.\n";**

**--i;**

**break;**

**}**

**}**

**}**

**}**

**// storing sets in txt file**

**template<typename T>**

**void inputfile(const T\* set1, const T\* set2, int size1, int size2, const string& filename) {**

**ofstream outputFile(filename);**

**if (!outputFile.is\_open()) {**

**cout << "Error opening file: " << filename << endl;**

**return;**

**}**

**outputFile << "Set 1 is: {";**

**for (int i = 0; i < size1; i++)**

**{**

**outputFile << set1[i];**

**if (i != size1 - 1)**

**outputFile << ",";**

**}**

**outputFile << "}\n";**

**outputFile << "Set 2 is: {";**

**for (int i = 0; i < size2; i++)**

**{**

**outputFile << set2[i];**

**if (i != size2 - 1)**

**outputFile << ",";**

**}**

**outputFile << "}\n";**

**outputFile.close();**

**}**

**//Set Display without txt file**

**template<typename T>**

**void setDisplay(const T\* set1, const T\* set2, int size1, int size2) {**

**cout << "Set 1 is: ";**

**cout << "{";**

**for (int i = 0; i < size1; i++)**

**{**

**cout << set1[i] << ",";**

**}**

**cout << "}";**

**cout << endl;**

**cout << "Set 2 is: ";**

**cout << "{";**

**for (int i = 0; i < size2; i++)**

**{**

**cout << set2[i] << ",";**

**}**

**cout << "}" << endl;**

**}**

**/\*----------------------------------------------- End --------------------------------------------------\*/**

**/\*------------------------------------------- Union / Intersection -------------------------------------\*/**

**//union of two sets(array)**

**template<typename T>**

**void setUnion(const T\* set1, int size1, const T\* set2, int size2) {**

**T\* set3 = new T[size1 + size2];**

**for (int i = 0; i < size1; ++i)**

**set3[i] = set1[i];**

**int size3 = size1;**

**for (int i = 0; i < size2; ++i) {**

**bool found = false;**

**for (int j = 0; j < size1; ++j) {**

**if (set2[i] == set1[j]) {**

**found = true;**

**break;**

**}**

**}**

**if (!found)**

**set3[size3++] = set2[i];**

**}**

**cout << "Union: { ";**

**for (int i = 0; i < size3; ++i)**

**cout << set3[i] << " ";**

**cout << "}\n";**

**delete[] set3;**

**} //---------------end---------------------**

**//intersection of two sets(array)**

**template<typename T>**

**void setIntersection(const T\* set1, int size1, const T\* set2, int size2) {**

**T\* set3 = new T[size1];**

**int size3 = 0;**

**for (int i = 0; i < size1; ++i) {**

**for (int j = 0; j < size2; ++j) {**

**if (set1[i] == set2[j]) {**

**bool check = false;**

**for (int k = 0; k < size3; ++k) {**

**if (set1[i] == set3[k]) {**

**check = true;**

**break;**

**}**

**}**

**if (!check)**

**set3[size3++] = set1[i];**

**break;**

**}**

**}**

**}**

**cout << "Intersection: { ";**

**for (int i = 0; i < size3; ++i)**

**cout << set3[i] << " ";**

**cout << "}\n";**

**delete[] set3;**

**}//------------end-----------------**

**//union of two sets in txt file**

**template<typename T>**

**void fileUnion(const T\* set1, int size1, const T\* set2, int size2, const string& filename) {**

**ofstream outputFile(filename);**

**if (!outputFile.is\_open()) {**

**cout << "Error opening file: " << filename << endl;**

**return;**

**}**

**T\* set3 = new T[size1 + size2];**

**for (int i = 0; i < size1; ++i)**

**set3[i] = set1[i];**

**int size3 = size1;**

**for (int i = 0; i < size2; ++i) {**

**bool found = false;**

**for (int j = 0; j < size1; ++j) {**

**if (set2[i] == set1[j]) {**

**found = true;**

**break;**

**}**

**}**

**if (!found)**

**set3[size3++] = set2[i];**

**}**

**outputFile << "Union: { ";**

**for (int i = 0; i < size3; ++i)**

**outputFile << set3[i] << " ";**

**outputFile << "}\n";**

**delete[] set3;**

**outputFile.close();**

**}//-------------end-----------------**

**// interection of two sets in txt file**

**template<typename T>**

**void fileIntersection(const T\* set1, int size1, const T\* set2, int size2, const string& filename) {**

**ofstream outputFile(filename);**

**if (!outputFile.is\_open()) {**

**cout << "Error opening file: " << filename << endl;**

**return;**

**}**

**T\* set3 = new T[size1];**

**int size3 = 0;**

**for (int i = 0; i < size1; ++i) {**

**for (int j = 0; j < size2; ++j) {**

**if (set1[i] == set2[j]) {**

**bool check = false;**

**for (int k = 0; k < size3; ++k) {**

**if (set1[i] == set3[k]) {**

**check = true;**

**break;**

**}**

**}**

**if (!check)**

**set3[size3++] = set1[i];**

**break;**

**}**

**}**

**}**

**outputFile << "Intersection: { ";**

**for (int i = 0; i < size3; ++i)**

**outputFile << set3[i] << " ";**

**outputFile << "}\n";**

**delete[] set3;**

**outputFile.close();**

**}//--------------------end------------------------**

**/\*----------------------------------------------- End --------------------------------------------------\*/**

**/\*------------------------------------------- subset / superset ----------------------------------------\*/**

**//Set Subset**

**template<typename T>**

**bool subset(const T\* set1, int size1, const T\* set2, int size2) {**

**for (int i = 0; i < size1; ++i) {**

**bool found = false;**

**for (int j = 0; j < size2; ++j) {**

**if (set1[i] == set2[j]) {**

**found = true;**

**break;**

**}**

**}**

**if (!found)**

**return false;**

**}**

**return true;**

**}//------------------end--------------------**

**//Superset of sets(array)**

**template<typename T>**

**bool superset(const T\* set1, int size1, const T\* set2, int size2) {**

**for (int i = 0; i < size2; ++i) {**

**bool found = false;**

**for (int j = 0; j < size1; ++j) {**

**if (set2[i] == set1[j]) {**

**found = true;**

**break;**

**}**

**}**

**if (!found)**

**return false;**

**}**

**return true;**

**}//---------------------end--------------------------**

**/\*----------------------------------------------- End --------------------------------------------------\*/**

**/\*------------------------------------------- set difference -------------------------------------------\*/**

**//Set Difference**

**template<typename T>**

**void setDifference(const T\* set1, int size1, const T\* set2, int size2) {**

**cout << "Set Difference (Set1 - Set2): { ";**

**bool found;**

**for (int i = 0; i < size1; ++i) {**

**found = false;**

**for (int j = 0; j < size2; ++j) {**

**if (set1[i] == set2[j]) {**

**found = true;**

**break;**

**}**

**}**

**if (!found)**

**cout << set1[i] << " ";**

**}**

**cout << "}\n";**

**}**

**/\*----------------------------------------------- End --------------------------------------------------\*/**

**/\*--------------------------------------------- power set ----------------------------------------------\*/**

**//Power Set Generation**

**template<typename T>**

**void powerSet(const T\* set, int size) {**

**cout << "Power Set:\n";**

**for (int i = 0; i < (1 << size); ++i) {**

**cout << "{ ";**

**for (int j = 0; j < size; ++j) {**

**if (i & (1 << j))**

**cout << set[j] << " ";**

**}**

**cout << "}\n";**

**}**

**}**

**/\*----------------------------------------------- End --------------------------------------------------\*/**

**/\*------------------------------------------- Cardinality ----------------------------------------------\*/**

**//Cardinality Calculation**

**template<typename T>**

**void cardinality(const T\* set, int size) {**

**cout << size;**

**}**

**/\*----------------------------------------------- End --------------------------------------------------\*/**

**/\*--------------------------------------------- equality -----------------------------------------------\*/**

**//Set Equality**

**template<typename T>**

**bool setEquality(const T\* set1, int size1, const T\* set2, int size2) {**

**if (size1 != size2)**

**return false;**

**char\* temp1 = new char[size1];**

**char\* temp2 = new char[size2];**

**for (int i = 0; i < size1; i++)**

**{**

**for (int j = 0; j < size2; j++) {**

**temp1[i] = set1[i];**

**temp2[j] = set2[j];**

**}**

**}**

**for (int i = 0; i < size1 - 1; ++i) {**

**for (int j = 0; j < size1 - i - 1; ++j) {**

**if (temp1[j] > temp1[j + 1]) {**

**T temp = temp1[j];**

**temp1[j] = temp1[j + 1];**

**temp1[j + 1] = temp;**

**}**

**if (temp2[j] > temp2[j + 1]) {**

**T temp = temp2[j];**

**temp2[j] = temp2[j + 1];**

**temp2[j + 1] = temp;**

**}**

**}**

**}**

**for (int i = 0; i < size1; ++i) {**

**if (set1[i] != set2[i])**

**return false;**

**}**

**return true;**

**}**

**/\*----------------------------------------------- End --------------------------------------------------\*/**

**/\*--------------------------------------------- compliment ---------------------------------------------\*/**

**//Set Complement**

**template<typename T>**

**void setComplement(T\* univeralSet, T\* set2, int universalSize, int size2) {**

**cout << "{ ";**

**bool found;**

**for (int i = 0; i < universalSize; ++i) {**

**found = false;**

**for (int j = 0; j < size2; ++j) {**

**if (univeralSet[i] == set2[j]) {**

**found = true;**

**break;**

**}**

**}**

**if (!found)**

**cout << univeralSet[i] << " ";**

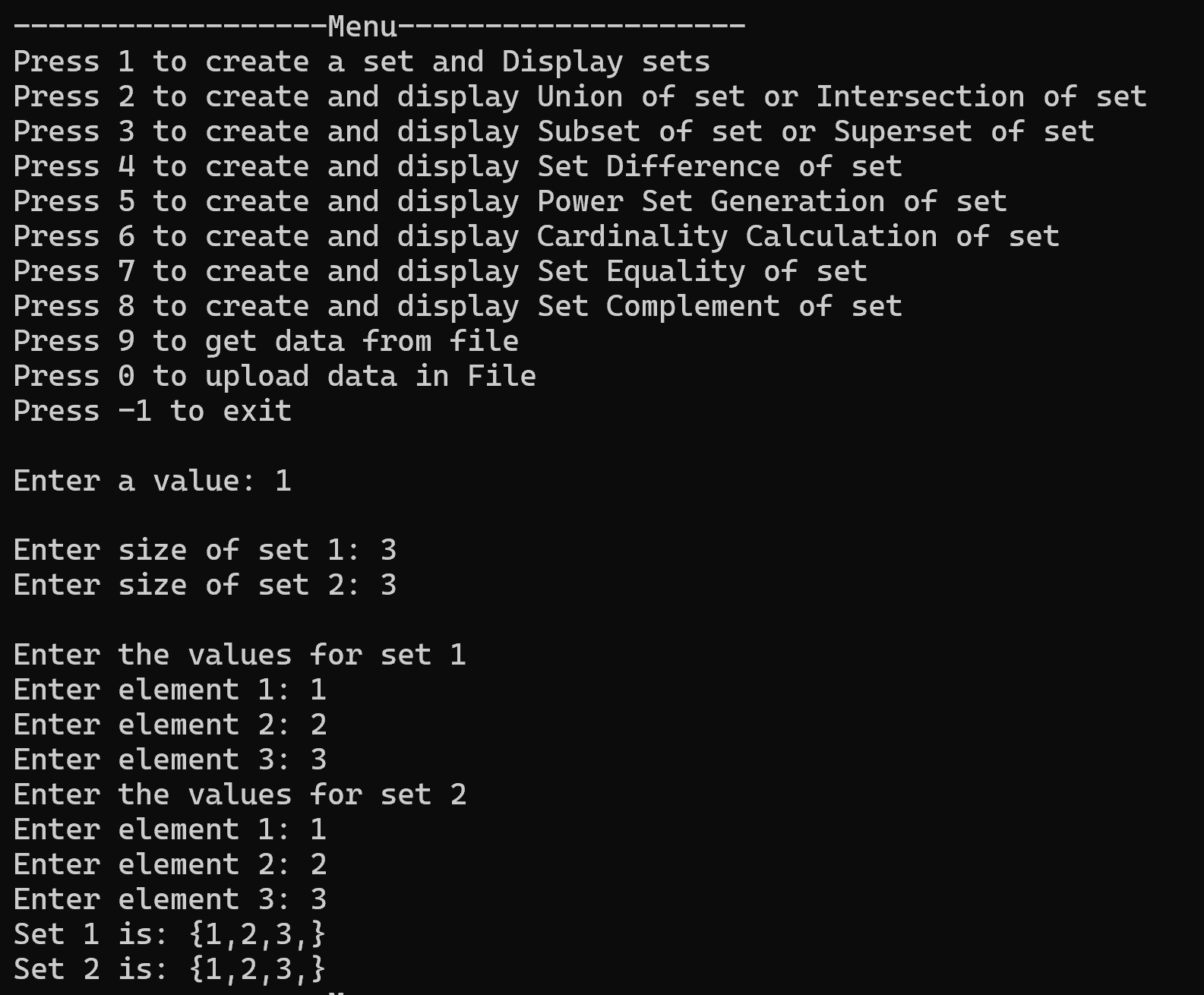
**}**

**cout << "}\n";**

**}**

**/\*----------------------------------------------- End --------------------------------------------------\*/**

**Output**

****

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

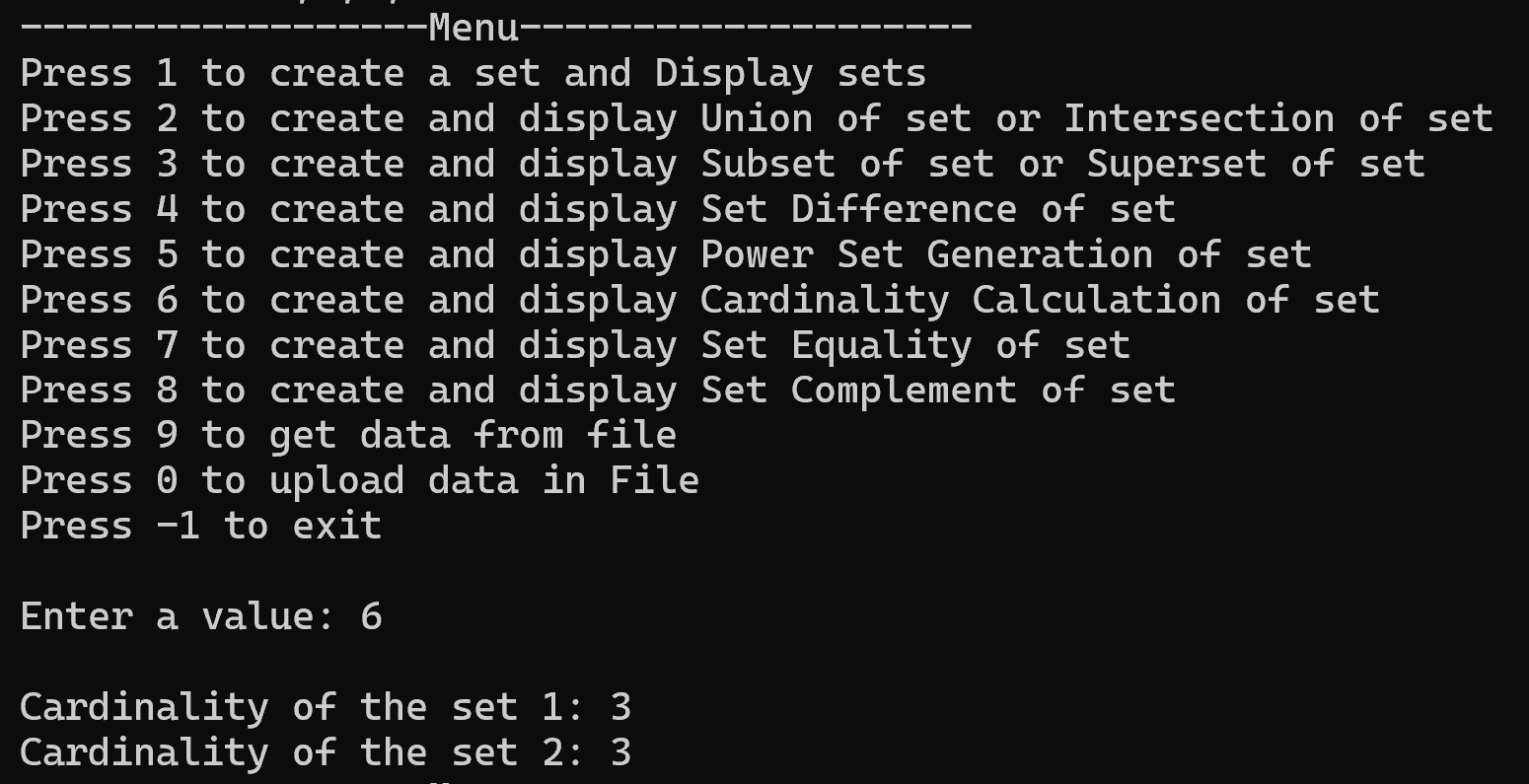
Description automatically generated**

**A screenshot of a computer

Description automatically generated**

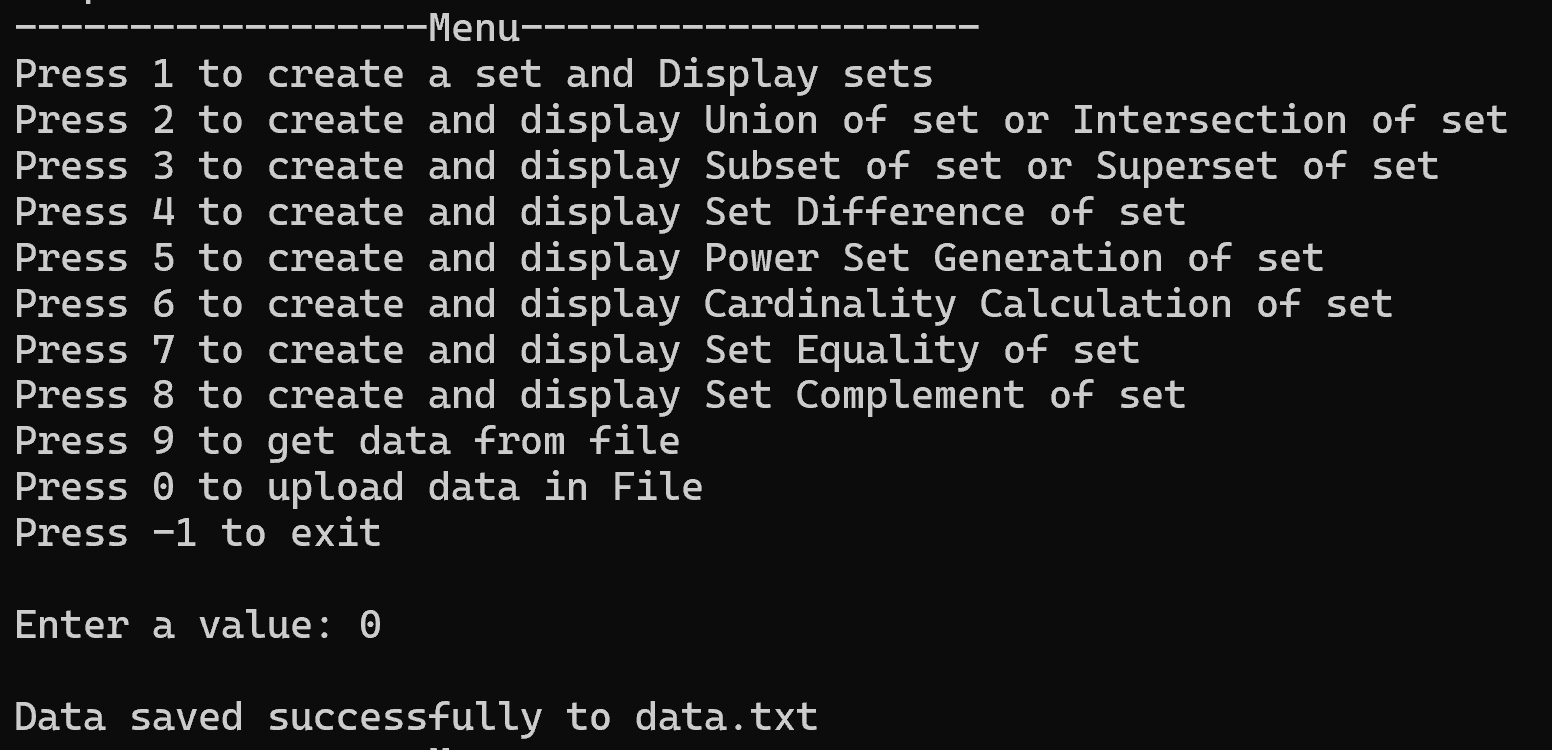
**A screenshot of a computer program

Description automatically generated**

****

**A screenshot of a computer program

Description automatically generated**

****

**A screenshot of a computer

Description automatically generated**