Alexandria University

Faculty of Engineering

CSED 2025

Discrete Lab 1
Report

Submitted to

Eng. Omar Elsherif

Faculty of Engineering, Alex. University

Submitted by

Islam Yasser Mahmoud 20010312

Mkario Michel Azer 20011982

Faculty of Engineering, Alex. University

Oct. 2022

Table of Contents

Problem Statement: 3
- Part 1: (Basic Bit Operations) 3
- Part 2: (Sets Operations using Bits manipulation) 3
- Part 3: (Applications for bits manipulation) 3

Used data structures: 3
- Part 1: (Basic Bit Operations) 3
- Part 2: (Sets Operations using Bits manipulation) 3
- Part 3: (Applications for bits manipulation) 3

Algorithms used: 4
- Part 1: (Basic Bit Operations) 4
- Part 2: (Sets Operations using Bits manipulation) 4
- Part 3: (Applications for bits manipulation) 4

Code Snippets: 5
- Part 1: (Basic Bit Operations) 5
- Part 2: (Sets Operations using Bits manipulation) 6
- Part 3: (Applications for bits manipulation) 8

Sample runs: 8
- Part 1: (Basic Bit Operations) 8
- Part 2: (Sets Operations using Bits manipulation) 9
- Part 3: (Applications for bits manipulation) 11

Assumptions and necessary details to be known: 12
- Part 1: (Basic Bit Operations) 12
- Part 2: (Sets Operations using Bits manipulation) 12
- Part 3: (Applications for bits manipulation) 12

Problem Statement:

- **Part 1: (Basic Bit Operations)**

An implementation of 4-bit operations which are:

  - **getBit**: which returns a chosen bit value of a number from its binary representation.
  - **setBit**: which returns a number in decimal representation after setting a chosen bit value to 1 in its binary representation.
  - **clearBit**: which returns a number in decimal representation after setting a chosen bit value to 0 in its binary representation.
  - **updateBit**: which returns a number in decimal representation after updating a chosen bit value to whether 0 or 1 according to the user's choice in its binary representation.

- **Part 2: (Sets Operations using Bits manipulation)**

  Given a Universe set and a number of subsets then we should find Union or Intersection of two sets or Complement of a set based on the user's request

- **Part 3: (Applications for bits manipulation)**

  1. Finding the number not in pair using XOR

     Given a set of numbers which each appears twice except a single one.

  2. Finding number of '1' bits

     Given an unsigned integer and it is required to return the number of '1' bits in its binary representation.

Used data structures:

- **Part 1: (Basic Bit Operations)**

No need. Just basic bitwise operations and some functions.

- **Part 2: (Sets Operations using Bits manipulation)**
  - Vector of strings (U) which stores the elements of the Universe set
  - Map (m) which maps the elements of the Universe to their index
  - Vector (sets) where each index contains vector of strings to store the elements of each subset
  - Array(sets_id) which stores the binary number representation of each set but in decimal number
- **Part 3: (Applications for bits manipulation)**
  - An **array** of size n called nums.

Algorithms used:

- **Part 1: (Basic Bit Operations)**
  - **int getBit(int number, int position)**

  Right shifting number by position value number of times then performing bitwise AND with '1'

  - **int setBit(int number, int position)**

  Left shifting '1' by position value number of times then performing bitwise OR with number

  - **int clearBit(int number, int position)**

  Left shifting '1' by position value number of times then doing bitwise NOT on the previous result then performing bitwise AND with number

  - **int updateBit(int number, int position, int value)**

  if value = 1 then use same algorithm as setBit ,else use same algorithm as clearBit

- **Part 2: (Sets Operations using Bits manipulation)**
  - Read the data from user
  - Store the elements of the Universe set in a vector and map the elements to their index by using map data structure
  - Store the elements of each subset in a vector of strings
  - Use map data structure to get the index of any element and use setBit() function to get the binary representation of each set
  - Store the binary representation of each set in an array
  - Perform Union, Intersection or Complement operations on the subsets upon user's request where Union is done by bitwise OR, Intersection is done by bitwise AND, Complement is done by bitwise NOT
- **Part 3: (Applications for bits manipulation)**

  1) **Find single occurrence**

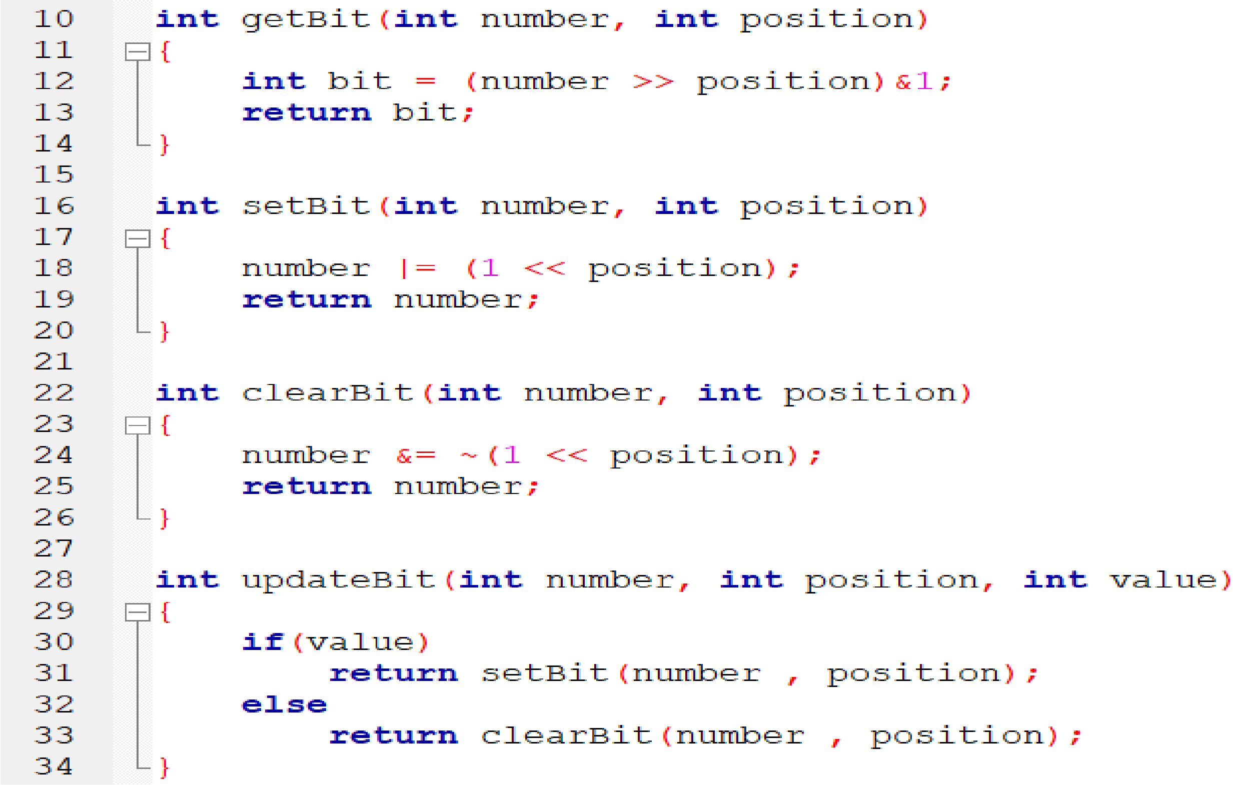
     - Read the elements from the user and store them in an array
     - Perform bitwise XOR between first and second element and store the result in a variable then iterate through array elements and perform bitwise XOR between last result and current element

  2) **number of '1' bits**

     - Read the number from the user and store it in a variable
     - Loop on the number until it becomes equal '0'
     - In every loop perform bitwise AND on the number and '1', if the result not equals '0' increase number of bits by one
     - At the end of every loop divide the number by 2

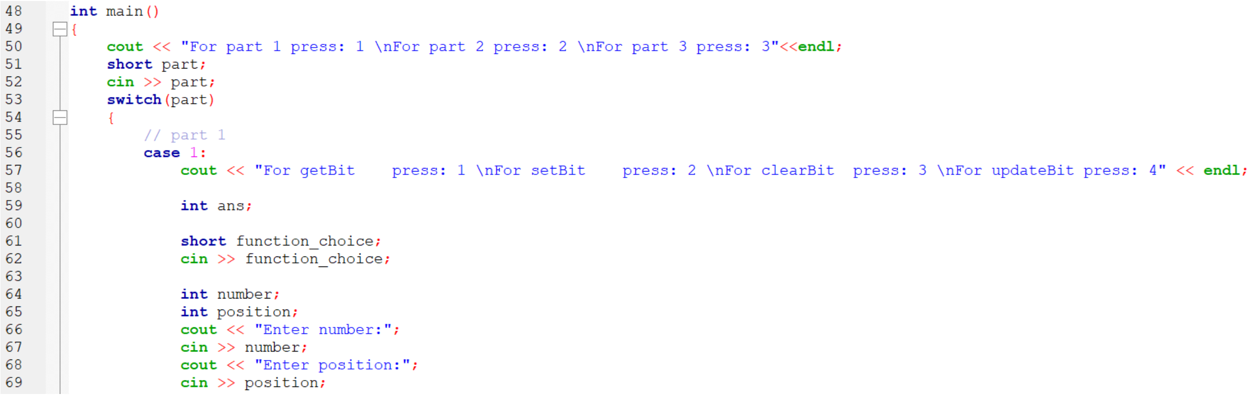
Code Snippets:

- **Part 1: (Basic Bit Operations)**



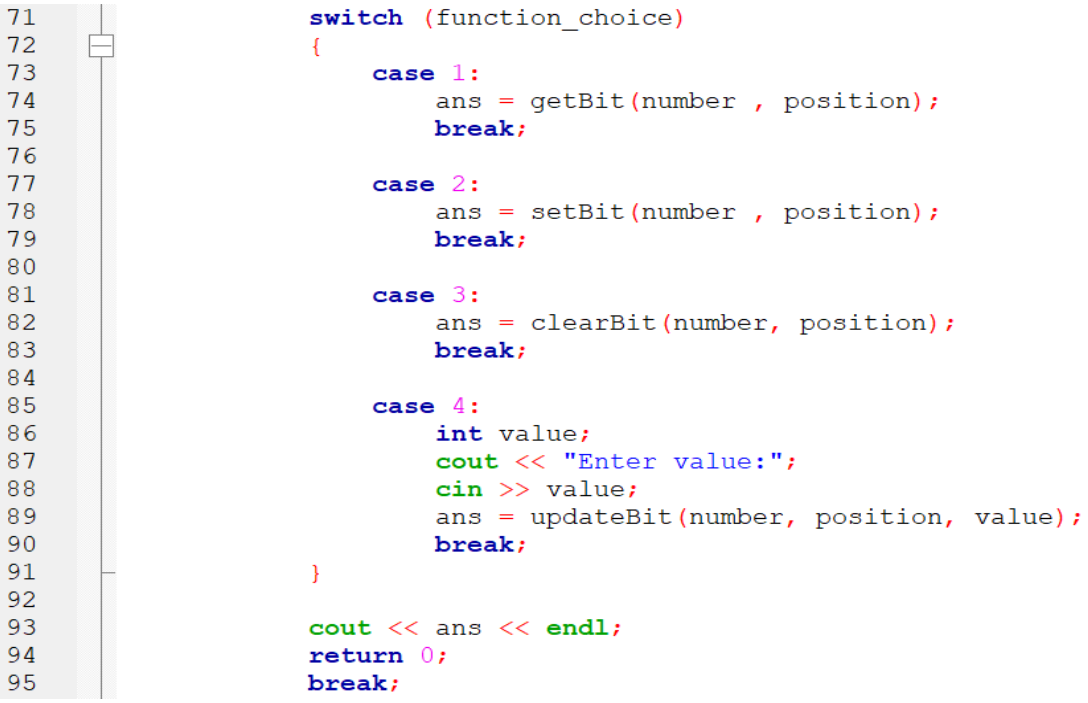
```cpp
int getBit(int number, int position)
{
    int bit = (number >> position)&1;
    return bit;
}

int setBit(int number, int position)
{
    number |= (1 << position);
    return number;
}

int clearBit(int number, int position)
{
    number &= ~(1 << position);
    return number;
}

int updateBit(int number, int position, int value)
{
    if(value)
        return setBit(number , position);
    else
        return clearBit(number , position);
}
```

Figure 0-0: The functions



```cpp
int main()
{
    cout << "For part 1 press: 1 \nFor part 2 press: 2 \nFor part 3 press: 3"<<endl;
    short part;
    cin >> part;
    switch(part)
    {
        // part 1
        case 1:
            cout << "For getBit    press: 1 \nFor setBit    press: 2 \nFor clearBit  press: 3 \nFor updateBit press: 4" << endl;

            int ans;

            short function_choice;
            cin >> function_choice;

            int number;
            int position;
            cout << "Enter number:";
            cin >> number;
            cout << "Enter position:";
            cin >> position;
```

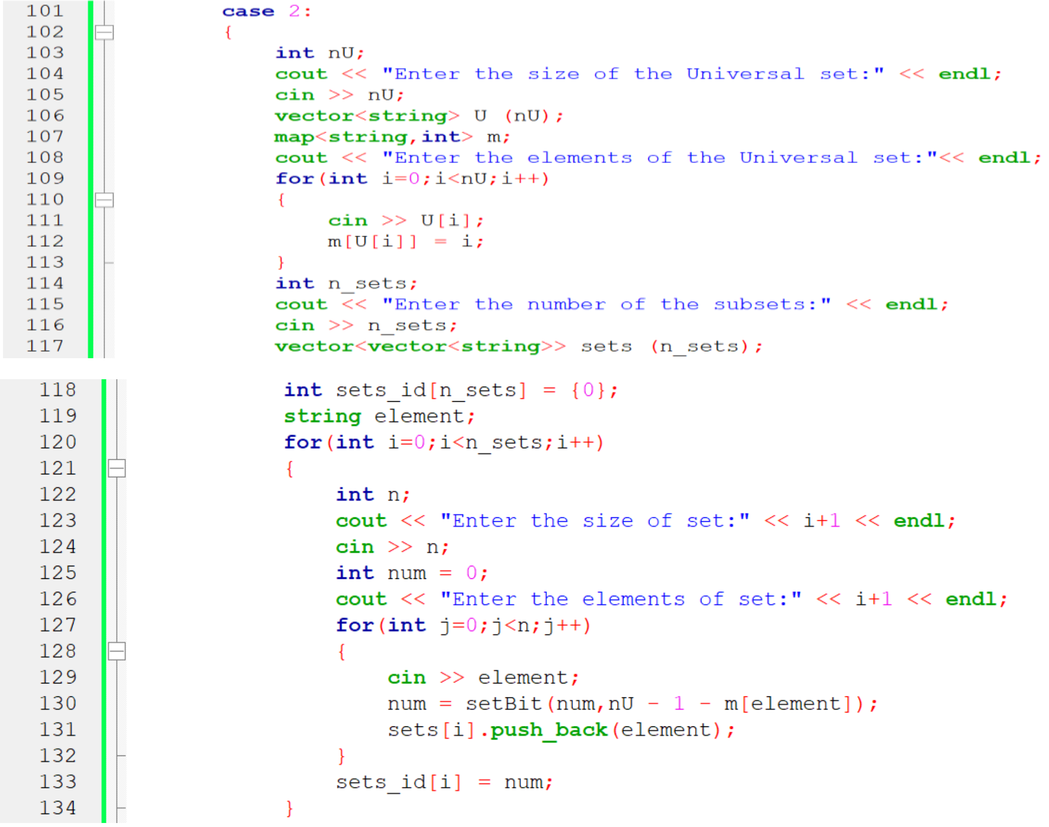
Figure 0-1: Initializing the needed data



```
switch (function_choice)
{
    case 1:
        ans = getBit(number , position);
        break;

    case 2:
        ans = setBit(number , position);
        break;

    case 3:
        ans = clearBit(number, position);
        break;

    case 4:
        int value;
        cout << "Enter value:";
        cin >> value;
        ans = updateBit(number, position, value);
        break;
}

cout << ans << endl;
return 0;
break;
```

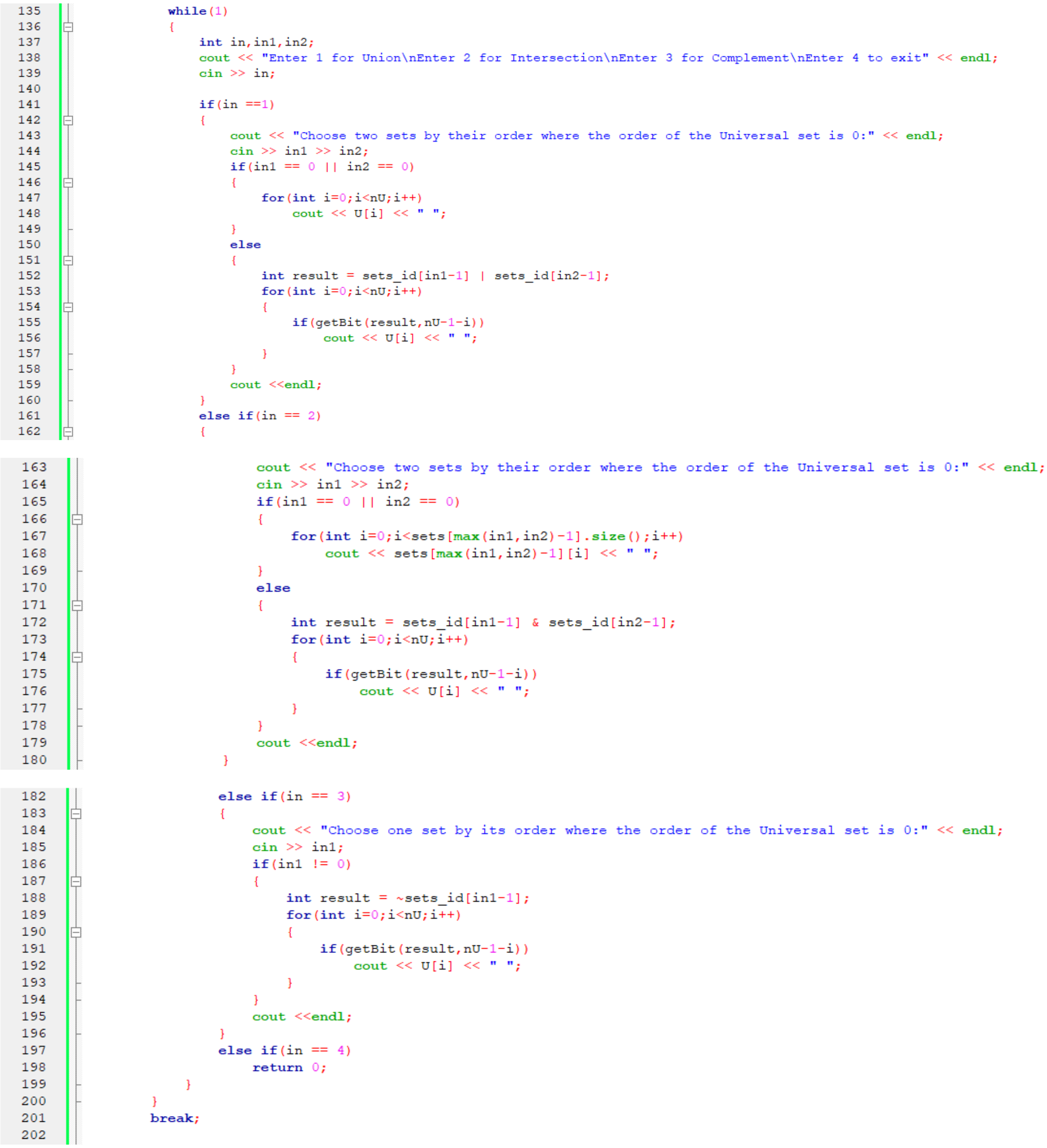
Figure 0-2: calling the functions and printing

- **Part 2: (Sets Operations using Bits manipulation)**



```
case 2:
{
    int nU;
    cout << "Enter the size of the Universal set:" << endl;
    cin >> nU;
    vector<string> U (nU);
    map<string,int> m;
    cout << "Enter the elements of the Universal set:"<< endl;
    for(int i=0;i<nU;i++)
    {
        cin >> U[i];
        m[U[i]] = i;
    }
    int n_sets;
    cout << "Enter the number of the subsets:" << endl;
    cin >> n_sets;
    vector<vector<string>> sets (n_sets);
    int sets_id[n_sets] = {0};
    string element;
    for(int i=0;i<n_sets;i++)
    {
        int n;
        cout << "Enter the size of set:" << i+1 << endl;
        cin >> n;
        int num = 0;
        cout << "Enter the elements of set:" << i+1 << endl;
        for(int j=0;j<n;j++)
        {
            cin >> element;
            num = setBit(num,nU - 1 - m[element]);
            sets[i].push_back(element);
        }
        sets_id[i] = num;
    }
```

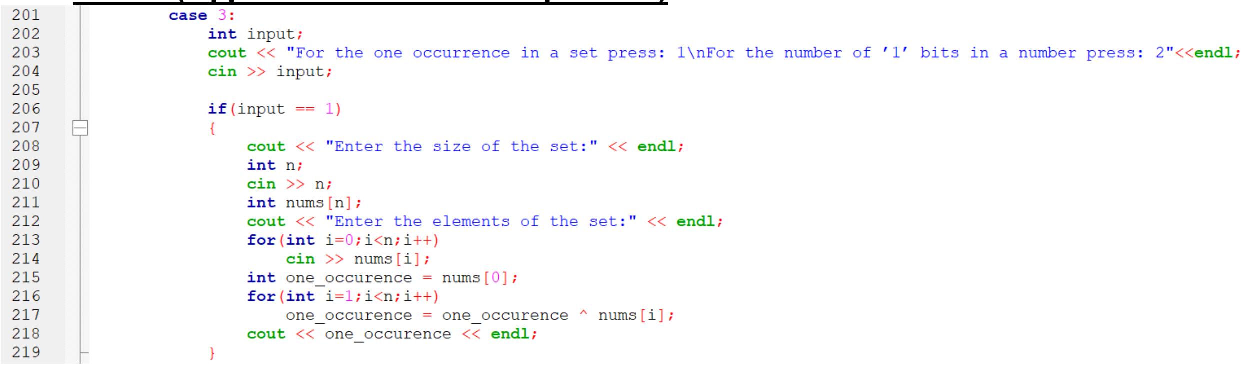
Figure 0-3, 0-4: Taking the data from the user



```cpp
        while(1)
        {
            int in,in1,in2;
            cout << "Enter 1 for Union\nEnter 2 for Intersection\nEnter 3 for Complement\nEnter 4 to exit" << endl;
            cin >> in;

            if(in ==1)
            {
                cout << "Choose two sets by their order where the order of the Universal set is 0:" << endl;
                cin >> in1 >> in2;
                if(in1 == 0 || in2 == 0)
                {
                    for(int i=0;i<nU;i++)
                        cout << U[i] << " ";
                }
                else
                {
                    int result = sets_id[in1-1] | sets_id[in2-1];
                    for(int i=0;i<nU;i++)
                    {
                        if(getBit(result,nU-1-i))
                            cout << U[i] << " ";
                    }
                }
                cout <<endl;
            }
            else if(in == 2)
            {
                cout << "Choose two sets by their order where the order of the Universal set is 0:" << endl;
                cin >> in1 >> in2;
                if(in1 == 0 || in2 == 0)
                {
                    for(int i=0;i<sets[max(in1,in2)-1].size();i++)
                        cout << sets[max(in1,in2)-1][i] << " ";
                }
                else
                {
                    int result = sets_id[in1-1] & sets_id[in2-1];
                    for(int i=0;i<nU;i++)
                    {
                        if(getBit(result,nU-1-i))
                            cout << U[i] << " ";
                    }
                }
                cout <<endl;
            }

            else if(in == 3)
            {
                cout << "Choose one set by its order where the order of the Universal set is 0:" << endl;
                cin >> in1;
                if(in1 != 0)
                {
                    int result = ~sets_id[in1-1];
                    for(int i=0;i<nU;i++)
                    {
                        if(getBit(result,nU-1-i))
                            cout << U[i] << " ";
                    }
                }
                cout <<endl;
            }
            else if(in == 4)
                return 0;
        }
    }
    break;
```

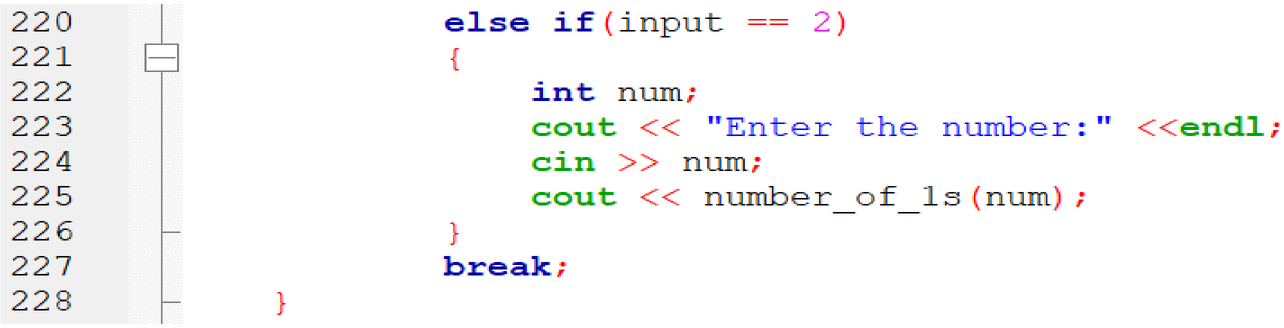
Figure 0-5, 0-6, 0-7 code implementation & Logic for part 2

- **Part 3: (Applications for bits manipulation)**



```
        case 3:
            int input;
            cout << "For the one occurrence in a set press: 1\nFor the number of ’1’ bits in a number press: 2"<<endl;
            cin >> input;

            if(input == 1)
            {
                cout << "Enter the size of the set:" << endl;
                int n;
                cin >> n;
                int nums[n];
                cout << "Enter the elements of the set:" << endl;
                for(int i=0;i<n;i++)
                    cin >> nums[i];
                int one_occurence = nums[0];
                for(int i=1;i<n;i++)
                    one_occurence = one_occurence ^ nums[i];
                cout << one_occurence << endl;
            }
```

Figure 0-8: Application 1



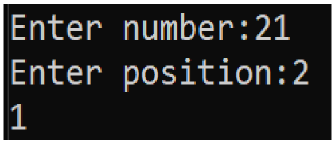
```
            else if(input == 2)
            {
                int num;
                cout << "Enter the number:" <<endl;
                cin >> num;
                cout << number_of_1s(num);
            }
            break;
        }
```

Figure 0-9: Application 2

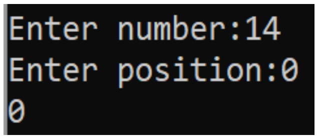
Sample runs:

- **Part 1: (Basic Bit Operations)**

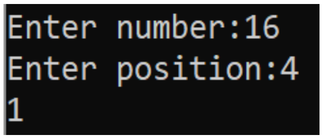
getBit



```
Enter number:21
Enter position:2
1
```

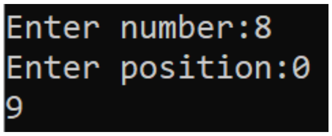


```
Enter number:14
Enter position:0
0
```

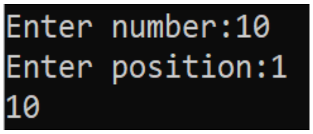


```
Enter number:16
Enter position:4
1
```

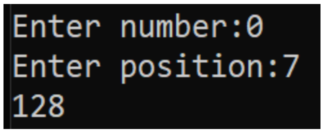
setBit



```
Enter number:8
Enter position:0
9
```

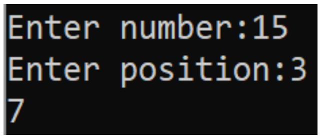


```
Enter number:10
Enter position:1
10
```

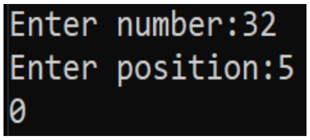


```
Enter number:0
Enter position:7
128
```

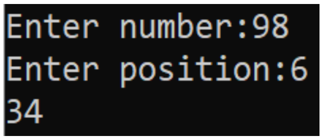
clearBit



```
Enter number:15
Enter position:3
7
```

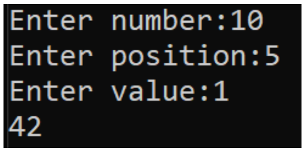


```
Enter number:32
Enter position:5
0
```

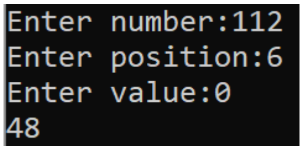


```
Enter number:98
Enter position:6
34
```

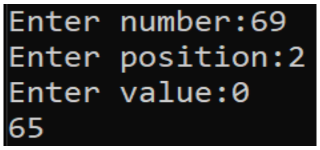
updateBit



```
Enter number:10
Enter position:5
Enter value:1
42
```



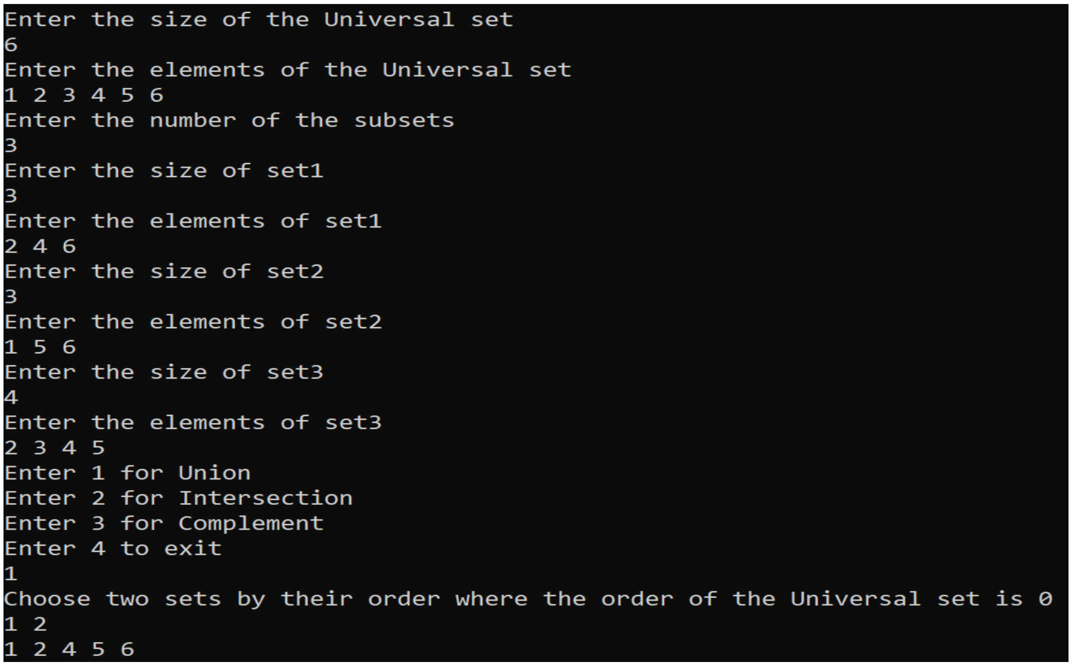
```
Enter number:112
Enter position:6
Enter value:0
48
```



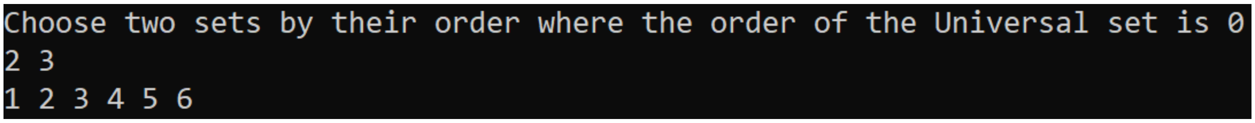
```
Enter number:69
Enter position:2
Enter value:0
65
```

- **<u>Part 2: (Sets Operations using Bits manipulation)</u>**

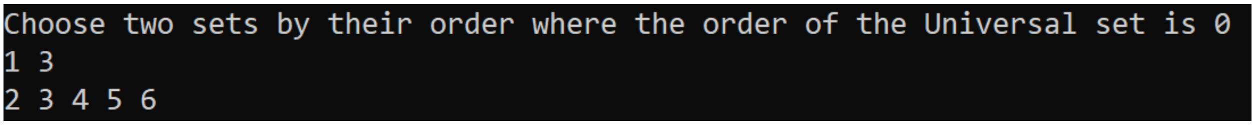
Union



```
Enter the size of the Universal set
6
Enter the elements of the Universal set
1 2 3 4 5 6
Enter the number of the subsets
3
Enter the size of set1
3
Enter the elements of set1
2 4 6
Enter the size of set2
3
Enter the elements of set2
1 5 6
Enter the size of set3
4
Enter the elements of set3
2 3 4 5
Enter 1 for Union
Enter 2 for Intersection
Enter 3 for Complement
Enter 4 to exit
1
Choose two sets by their order where the order of the Universal set is 0
1 2
1 2 4 5 6
```

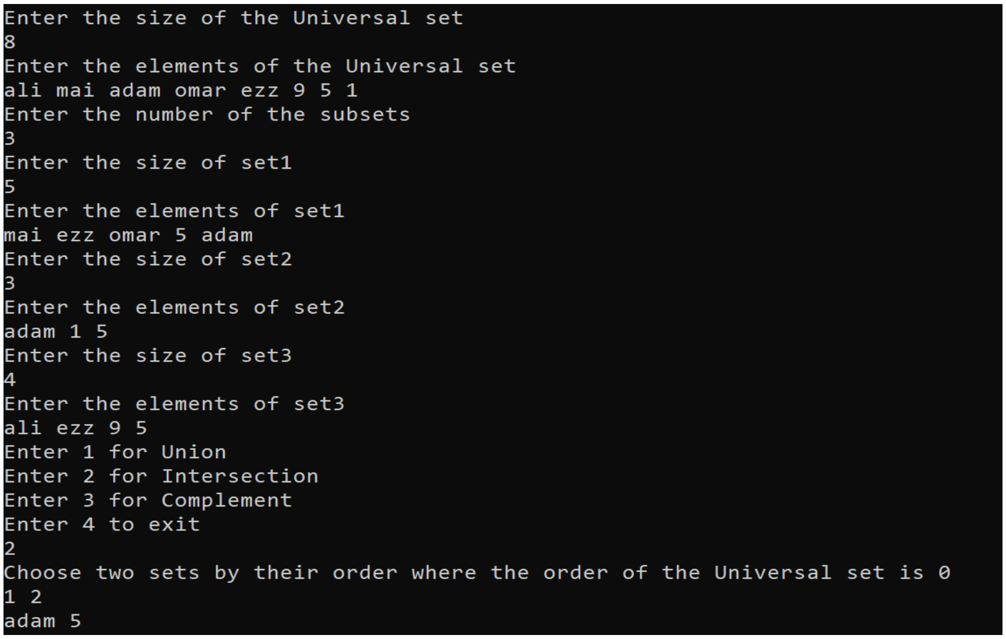


```
Choose two sets by their order where the order of the Universal set is 0
2 3
1 2 3 4 5 6
```

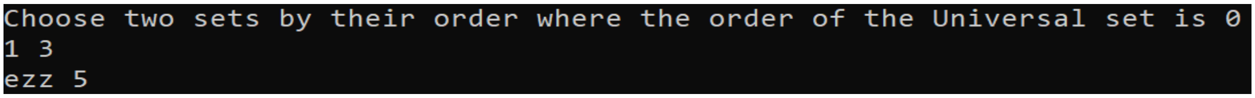


```
Choose two sets by their order where the order of the Universal set is 0
1 3
2 3 4 5 6
```

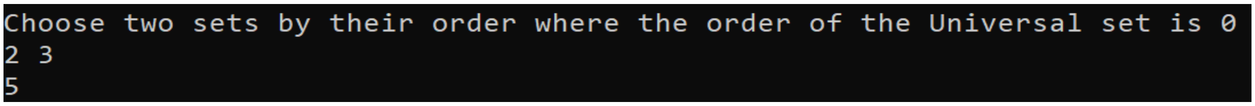
Intersection



```
Enter the size of the Universal set
8
Enter the elements of the Universal set
ali mai adam omar ezz 9 5 1
Enter the number of the subsets
3
Enter the size of set1
5
Enter the elements of set1
mai ezz omar 5 adam
Enter the size of set2
3
Enter the elements of set2
adam 1 5
Enter the size of set3
4
Enter the elements of set3
ali ezz 9 5
Enter 1 for Union
Enter 2 for Intersection
Enter 3 for Complement
Enter 4 to exit
2
Choose two sets by their order where the order of the Universal set is 0
1 2
adam 5
```

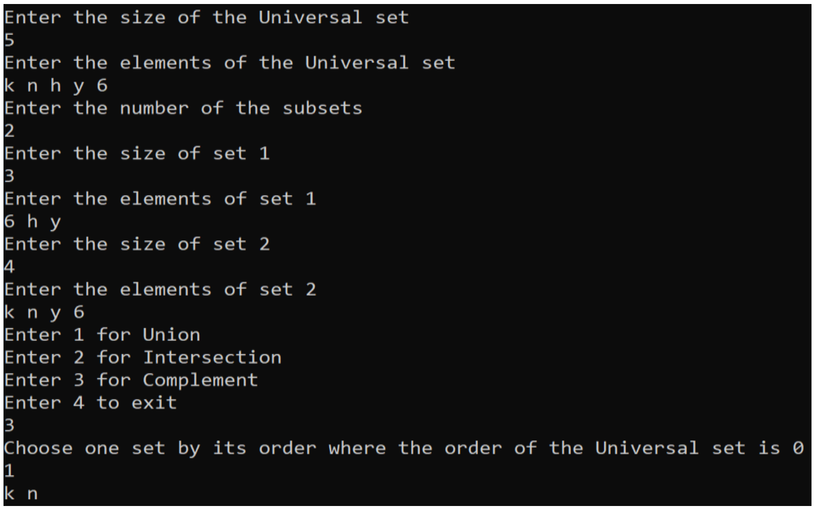


```
Choose two sets by their order where the order of the Universal set is 0
1 3
ezz 5
```

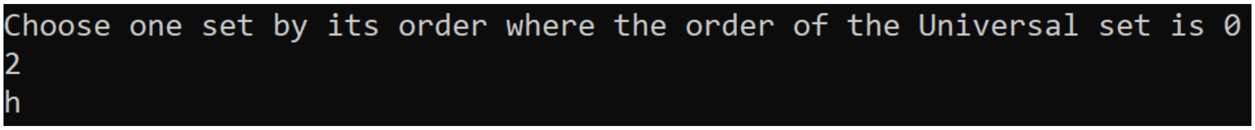


```
Choose two sets by their order where the order of the Universal set is 0
2 3
5
```

Complement



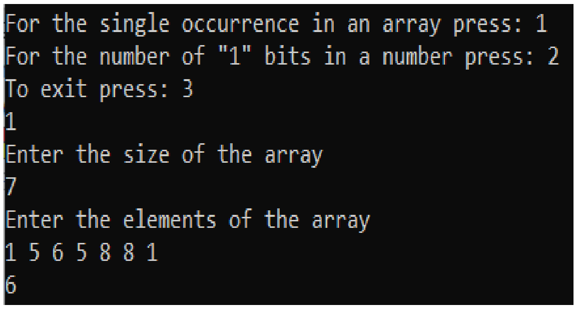
```
Enter the size of the Universal set
5
Enter the elements of the Universal set
k n h y 6
Enter the number of the subsets
2
Enter the size of set 1
3
Enter the elements of set 1
6 h y
Enter the size of set 2
4
Enter the elements of set 2
k n y 6
Enter 1 for Union
Enter 2 for Intersection
Enter 3 for Complement
Enter 4 to exit
3
Choose one set by its order where the order of the Universal set is 0
1
k n
```



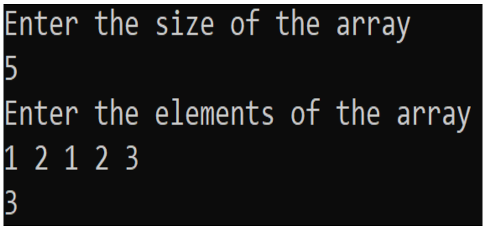
```
Choose one set by its order where the order of the Universal set is 0
2
h
```

- **Part 3: (Applications for bits manipulation)**

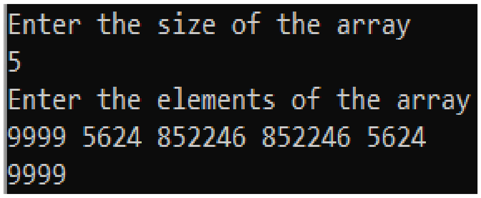
Find single occurrence



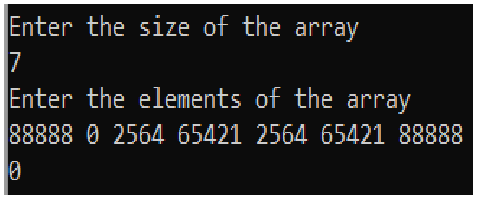
```
For the single occurrence in an array press: 1
For the number of "1" bits in a number press: 2
To exit press: 3
1
Enter the size of the array
7
Enter the elements of the array
1 5 6 5 8 8 1
6
```



```
Enter the size of the array
5
Enter the elements of the array
1 2 1 2 3
3
```

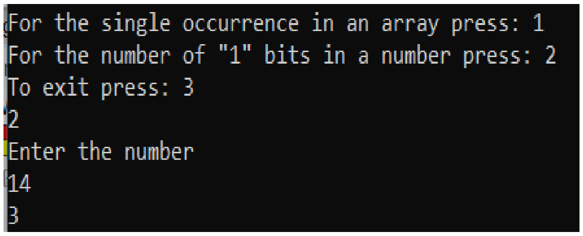


```
Enter the size of the array
5
Enter the elements of the array
9999 5624 852246 852246 5624
9999
```

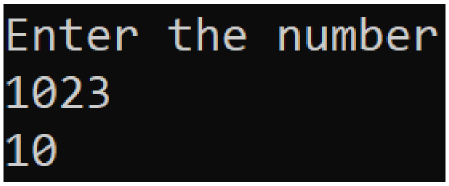


```
Enter the size of the array
7
Enter the elements of the array
88888 0 2564 65421 2564 65421 88888
0
```

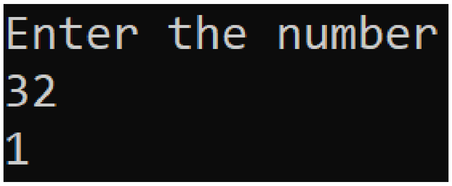
number of ’1’ bits



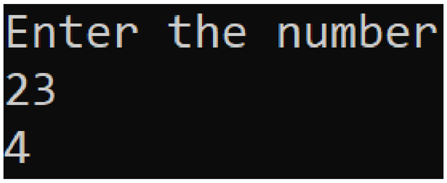
```
For the single occurrence in an array press: 1
For the number of "1" bits in a number press: 2
To exit press: 3
2
Enter the number
14
3
```



```
Enter the number
1023
10
```



```
Enter the number
32
1
```



```
Enter the number
23
4
```

Assumptions and necessary details to be known:

- **Part 1: (Basic Bit Operations)**
  - Bitwise operations (AND, OR, NOT, Right shift, Left shift) perform the four functions (getBit(), setBit(), clearBit(), updateBit()) efficiently, easily and in a short code implementation.

- **Part 2: (Sets Operations using Bits manipulation)**
  - using map data structure for mapping the elements of the universal set to their index is O(1) which facilitates search operation instead of using the built in function find() which is O(n).
  - we use an array of integers where the binary representation of each number expresses the elements that exists in each set .
  - each set is identified by its order of input and the order of the universe is 0.
  - Bitwise OR is used for Union operation, Bitwise AND is used for Intersection operation and Bitwise NOT is used for Complement operation.
- **Part 3: (Applications for bits manipulation)**

  Single occurrence
    - using Bitwise XOR facilitates much finding the number that exists once.