Alexandria University
Faculty of Engineering
Computer and Systems Engineering Department
CSE 214: Discrete Structures

Lab 1 – Bit Manipulations

Team Members

1. ID: 20011574 – محمد رفيق محمد السيد ابراهيم
2. ID: 20011923 – مصطفى خالد كمال احمد زايد

First: Problem Statement

The problem is divided into three parts:

- Part One: Implementing basic bit operations; getting, setting, clearing, and updating. Each of them takes two parameters; number, and bit position. (The update function takes and extra parameter that decides whether we set or clear the bit)

- Part Two: Implementing basic set operations; union, intersection, and complementing. The user is asked to enter a Universal Set and several subsets, then he chooses the desired operation and selects the subsets on which the operation is executed.

  This part must be implemented using bit manipulations, and in order to do so we convert the given subsets into binary arrays, and we apply bit manipulations on them.

- Part Three: Two applications are to be implemented:

  - Finding the number that appears only once in an array given that every other number is repeated twice.

  - Finding the number of '1' bits in an unsigned integer.

Second: Used Data Structures

Only basic arrays are used to implement the three parts of the assignment.

- Part one: No data structures are used in this part.

- Part two:

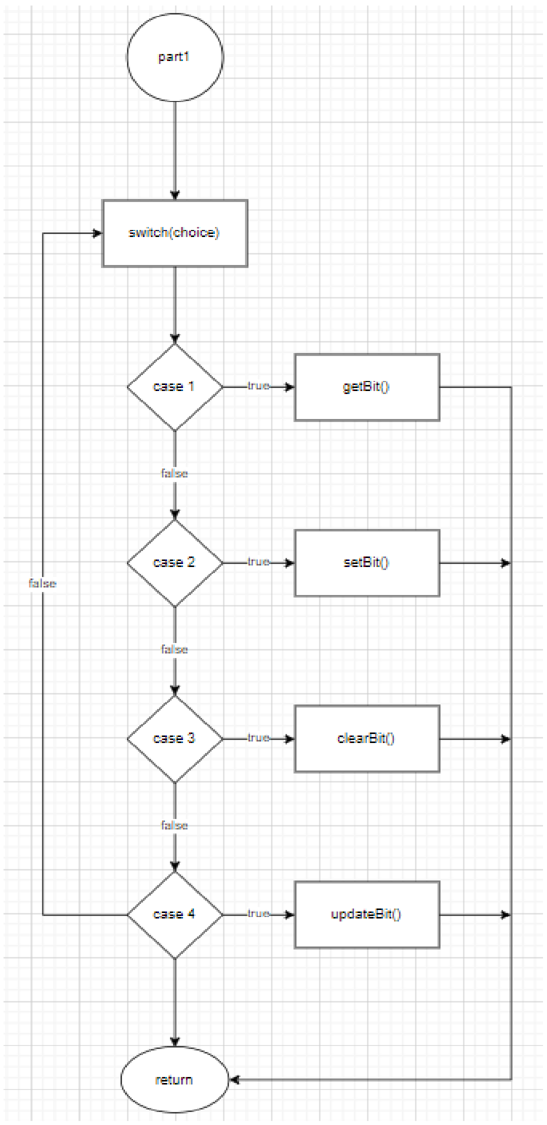
  - `string elements[10000]`: The universal set that the user inputs at the beginning of the program.
  - `int s1[len], s2[len]`: The resulting binary array after converting each original subset.
  - `static int ans[10000]`: The array that holds the answer to a specific operation, and it is declared as `static` as we return a pointer to its first element.
  - `string subs[num][len + 1]`: The array that holds the original subsets.

- Part Three:

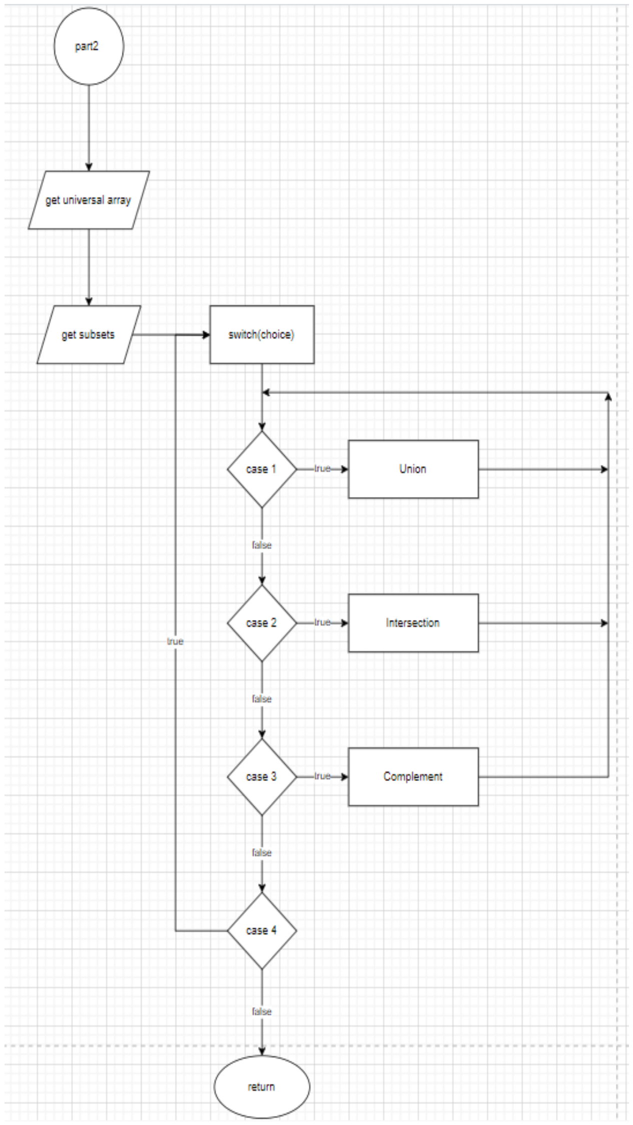
  - `long long arr[n]`: The array that holds the given numbers among which is the unique number to be found.

Third: Flow Charts

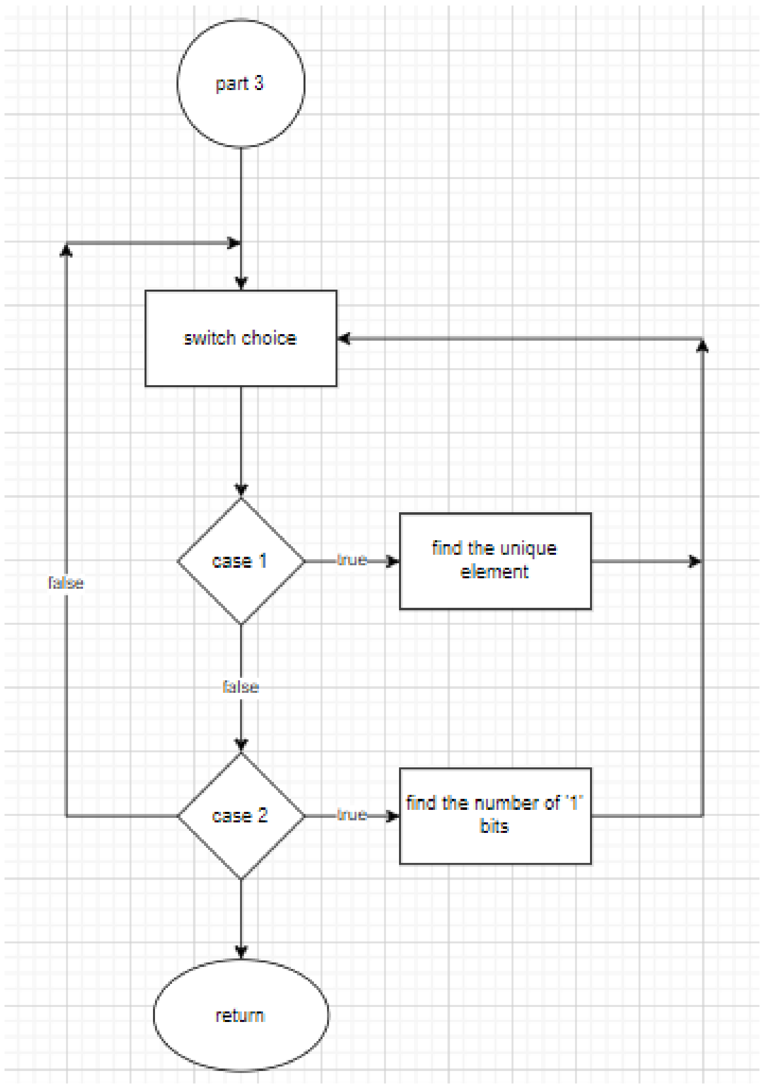
Part One



Part Two

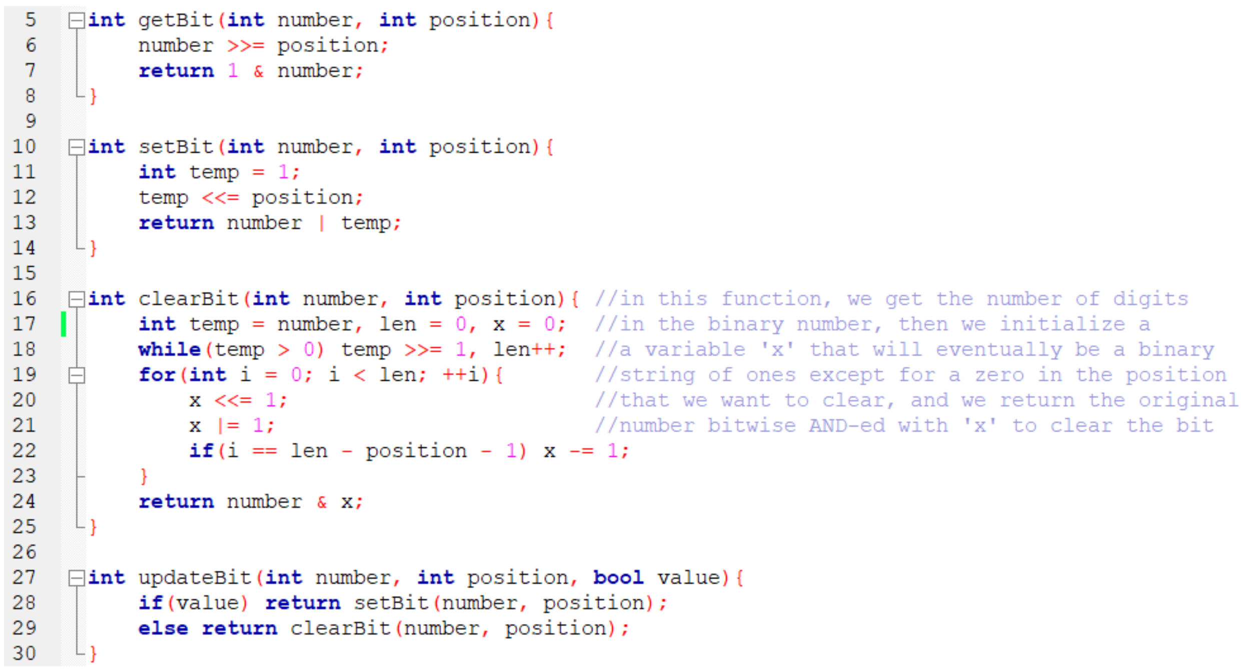


Part Three


part 3
switch choice
case 1
true
find the unique element
false
false
case 2
true
find the number of '1' bits
return

Fourth: Code Snippets

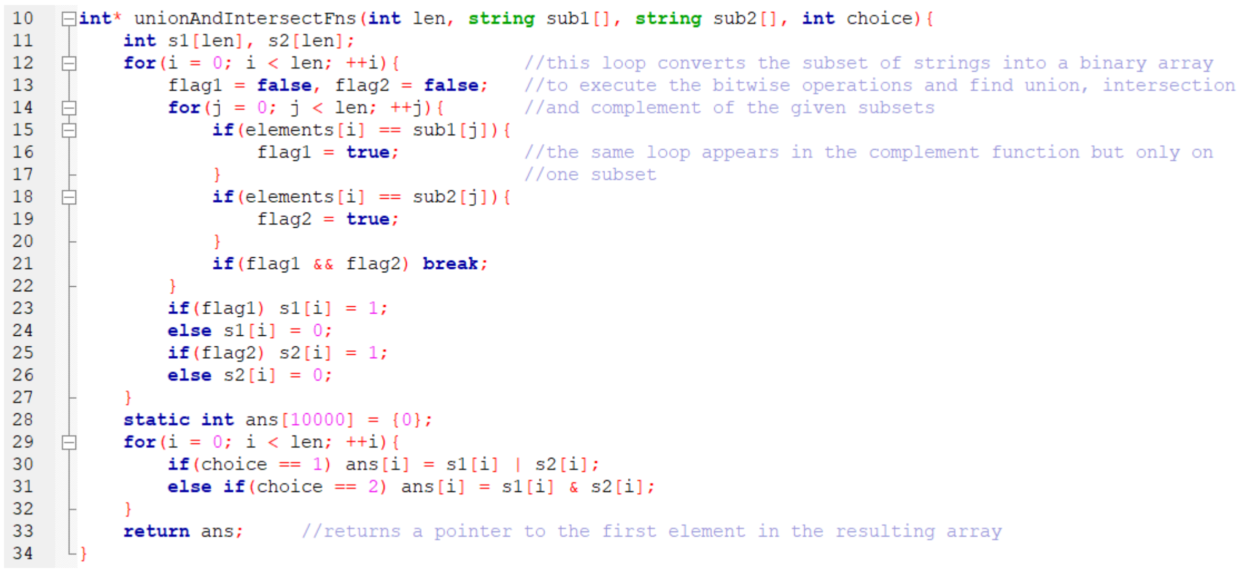
Part One:



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

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

int clearBit(int number, int position){ //in this function, we get the number of digits
    int temp = number, len = 0, x = 0;  //in the binary number, then we initialize a
    while(temp > 0) temp >>= 1, len++;  //a variable 'x' that will eventually be a binary
    for(int i = 0; i < len; ++i){       //string of ones except for a zero in the position
        x <<= 1;                        //that we want to clear, and we return the original
        x |= 1;                         //number bitwise AND-ed with 'x' to clear the bit
        if(i == len - position - 1) x -= 1;
    }
    return number & x;
}

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

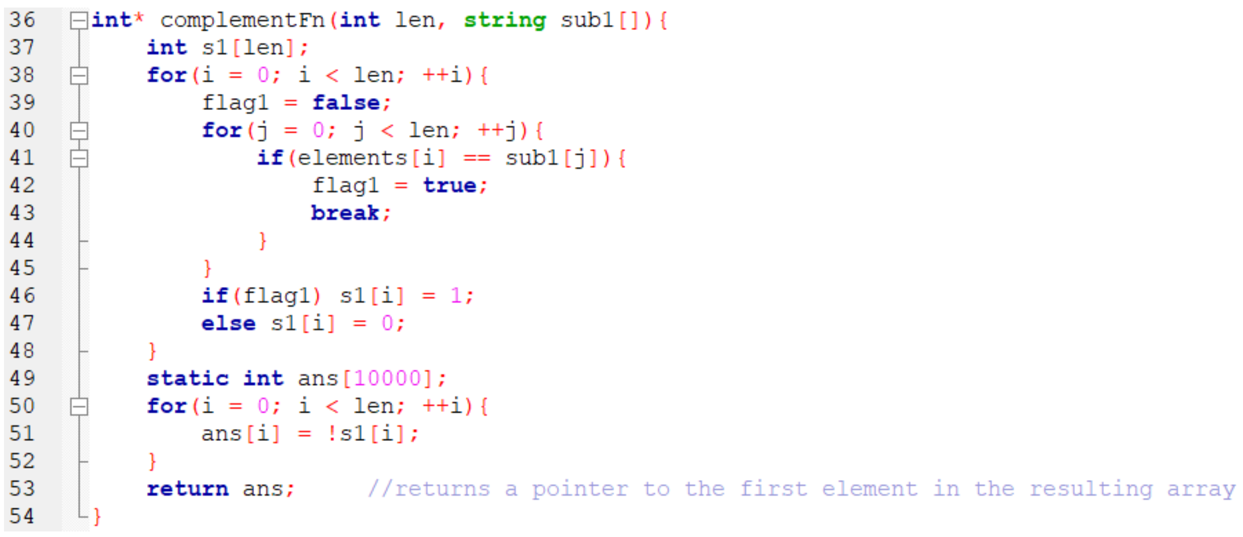
Basic Bit Operations

Part Two:



```cpp
int* unionAndIntersectFns(int len, string sub1[], string sub2[], int choice){
    int s1[len], s2[len];
    for(i = 0; i < len; ++i){           //this loop converts the subset of strings into a binary array
        flag1 = false, flag2 = false;   //to execute the bitwise operations and find union, intersection
        for(j = 0; j < len; ++j){       //and complement of the given subsets
            if(elements[i] == sub1[j]){
                flag1 = true;           //the same loop appears in the complement function but only on
            }                           //one subset
            if(elements[i] == sub2[j]){
                flag2 = true;
            }
            if(flag1 && flag2) break;
        }
        if(flag1) s1[i] = 1;
        else s1[i] = 0;
        if(flag2) s2[i] = 1;
        else s2[i] = 0;
    }
    static int ans[10000] = {0};
    for(i = 0; i < len; ++i){
        if(choice == 1) ans[i] = s1[i] | s2[i];
        else if(choice == 2) ans[i] = s1[i] & s2[i];
    }
    return ans;     //returns a pointer to the first element in the resulting array
}
```

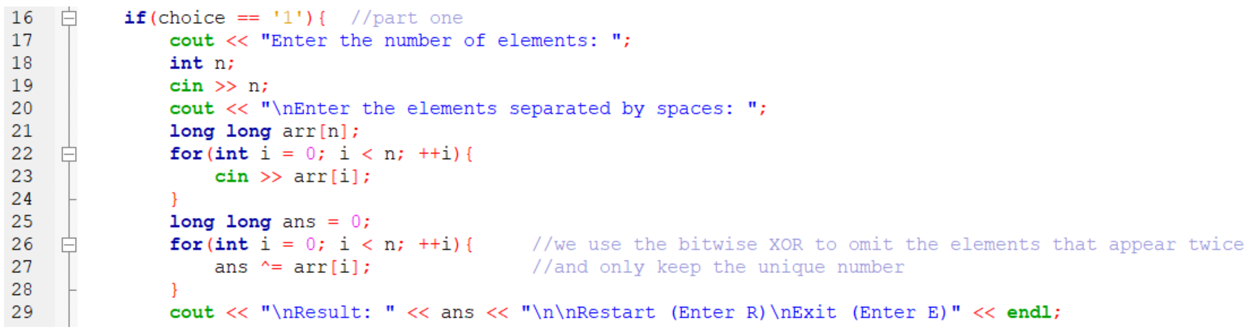
Union and Intersection



```cpp
int* complementFn(int len, string sub1[]){
    int s1[len];
    for(i = 0; i < len; ++i){
        flag1 = false;
        for(j = 0; j < len; ++j){
            if(elements[i] == sub1[j]){
                flag1 = true;
                break;
            }
        }
        if(flag1) s1[i] = 1;
        else s1[i] = 0;
    }
    static int ans[10000];
    for(i = 0; i < len; ++i){
        ans[i] = !s1[i];
    }
    return ans;     //returns a pointer to the first element in the resulting array
}
```

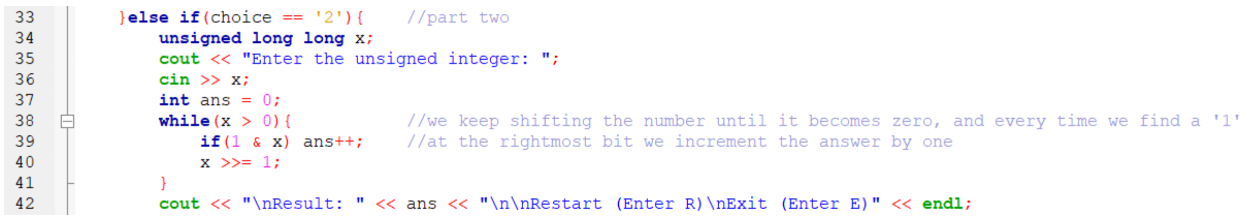
Complement

Part Three:



```cpp
    if(choice == '1'){  //part one
        cout << "Enter the number of elements: ";
        int n;
        cin >> n;
        cout << "\nEnter the elements separated by spaces: ";
        long long arr[n];
        for(int i = 0; i < n; ++i){
            cin >> arr[i];
        }
        long long ans = 0;
        for(int i = 0; i < n; ++i){     //we use the bitwise XOR to omit the elements that appear twice
            ans ^= arr[i];              //and only keep the unique number
        }
        cout << "\nResult: " << ans << "\n\nRestart (Enter R)\nExit (Enter E)" << endl;
```

Finding the unique number

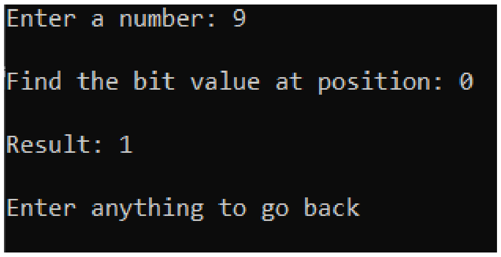


```cpp
    }else if(choice == '2'){    //part two
        unsigned long long x;
        cout << "Enter the unsigned integer: ";
        cin >> x;
        int ans = 0;
        while(x > 0){           //we keep shifting the number until it becomes zero, and every time we find a '1'
            if(1 & x) ans++;    //at the rightmost bit we increment the answer by one
            x >>= 1;
        }
        cout << "\nResult: " << ans << "\n\nRestart (Enter R)\nExit (Enter E)" << endl;
```

Counting the number of ‘1’ bits in a number

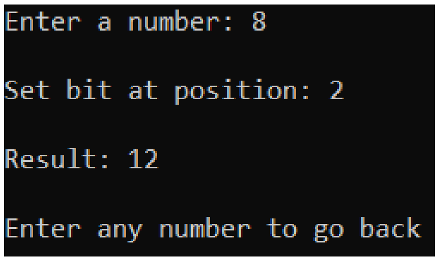
Fifth: Sample runs

Part One:



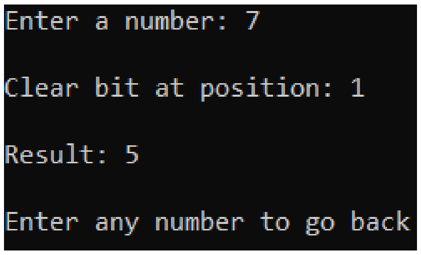
```
Enter a number: 9

Find the bit value at position: 0

Result: 1

Enter anything to go back
```

Getting a bit



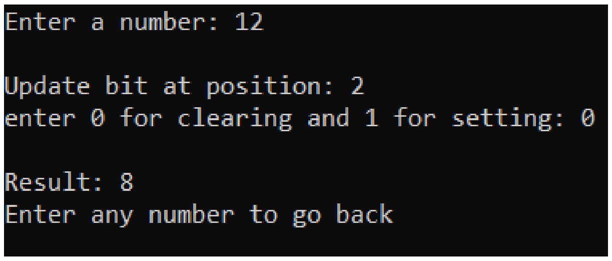
```
Enter a number: 8

Set bit at position: 2

Result: 12

Enter any number to go back
```

Setting a bit



```
Enter a number: 7

Clear bit at position: 1

Result: 5

Enter any number to go back
```

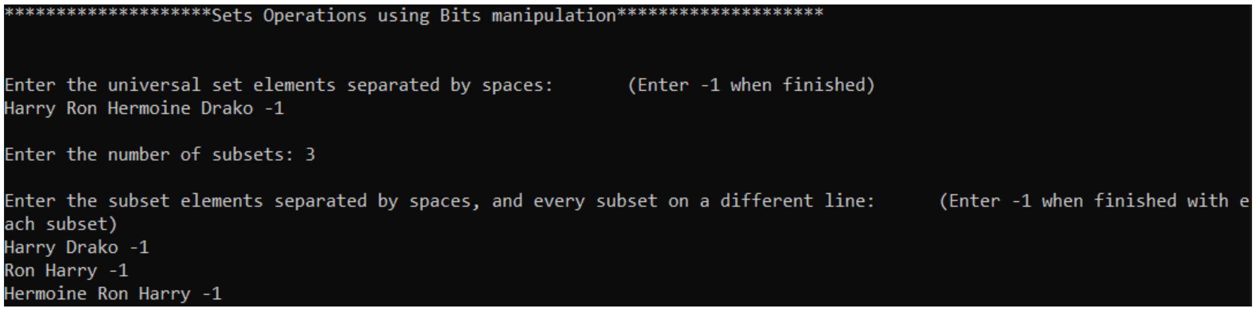
Clearing a bit



```
Enter a number: 12

Update bit at position: 2
enter 0 for clearing and 1 for setting: 0

Result: 8
Enter any number to go back
```

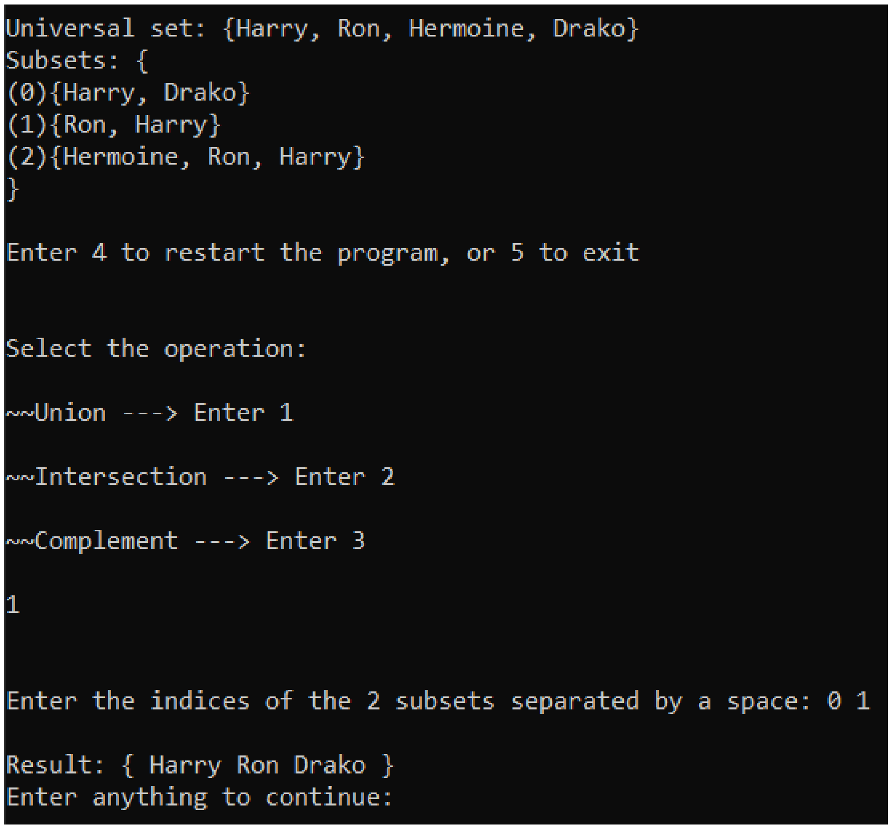
Updating a bit

Part two:



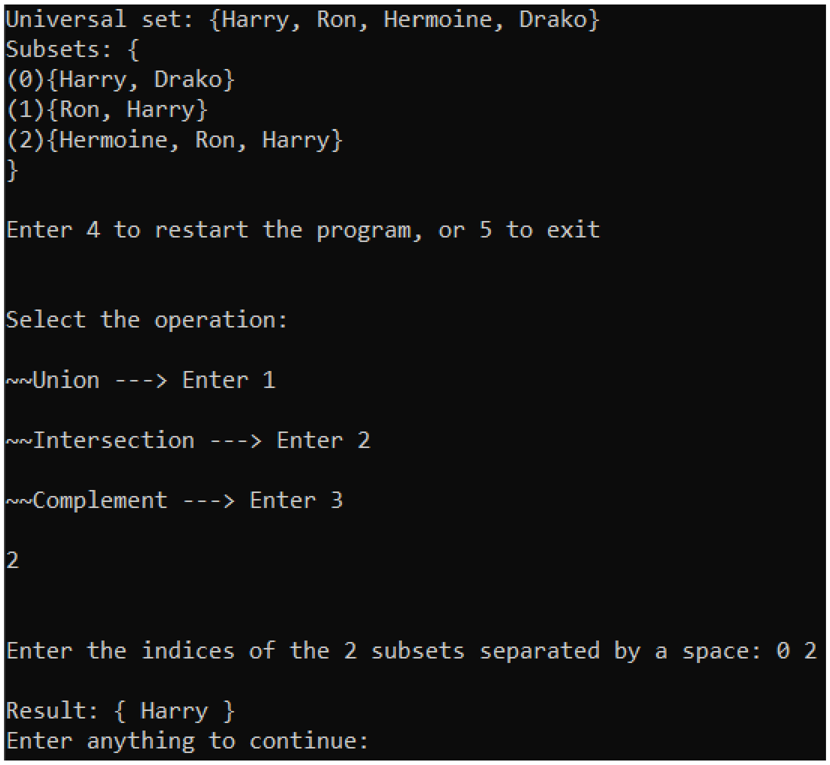
```
********************Sets Operations using Bits manipulation********************


Enter the universal set elements separated by spaces:       (Enter -1 when finished)
Harry Ron Hermoine Drako -1

Enter the number of subsets: 3

Enter the subset elements separated by spaces, and every subset on a different line:      (Enter -1 when finished with e
ach subset)
Harry Drako -1
Ron Harry -1
Hermoine Ron Harry -1
```

Entering the Universal Set and its subsets



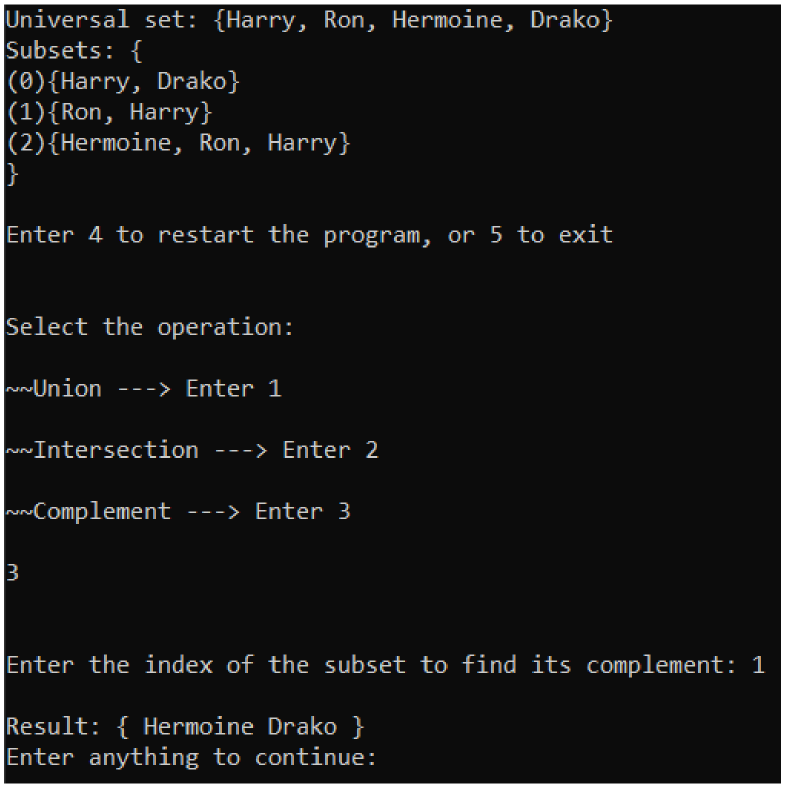
```
Universal set: {Harry, Ron, Hermoine, Drako}
Subsets: {
(0){Harry, Drako}
(1){Ron, Harry}
(2){Hermoine, Ron, Harry}
}

Enter 4 to restart the program, or 5 to exit


Select the operation:

~~Union ---> Enter 1

~~Intersection ---> Enter 2

~~Complement ---> Enter 3

1


Enter the indices of the 2 subsets separated by a space: 0 1

Result: { Harry Ron Drako }
Enter anything to continue:
```

Union



```
Universal set: {Harry, Ron, Hermoine, Drako}
Subsets: {
(0){Harry, Drako}
(1){Ron, Harry}
(2){Hermoine, Ron, Harry}
}

Enter 4 to restart the program, or 5 to exit


Select the operation:

~~Union ---> Enter 1

~~Intersection ---> Enter 2

~~Complement ---> Enter 3

2


Enter the indices of the 2 subsets separated by a space: 0 2

Result: { Harry }
Enter anything to continue:
```

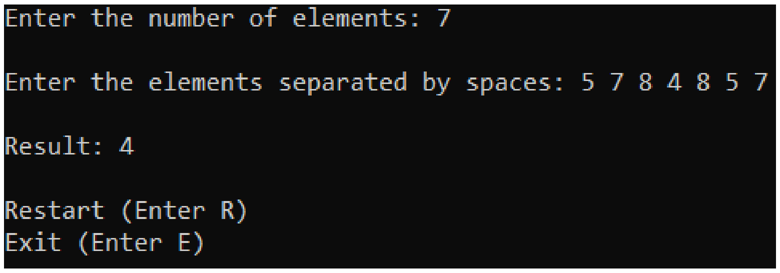
Intersection



```
Universal set: {Harry, Ron, Hermoine, Drako}
Subsets: {
(0){Harry, Drako}
(1){Ron, Harry}
(2){Hermoine, Ron, Harry}
}

Enter 4 to restart the program, or 5 to exit


Select the operation:

~~Union ---> Enter 1

~~Intersection ---> Enter 2

~~Complement ---> Enter 3

3


Enter the index of the subset to find its complement: 1

Result: { Hermoine Drako }
Enter anything to continue:
```

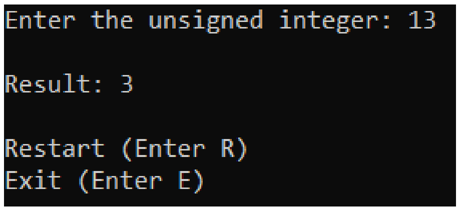
Complement

Part Three:



```
Enter the number of elements: 7

Enter the elements separated by spaces: 5 7 8 4 8 5 7

Result: 4

Restart (Enter R)
Exit (Enter E)
```

Finding the unique number



```
Enter the unsigned integer: 13

Result: 3

Restart (Enter R)
Exit (Enter E)
```

Finding the number of ‘1’ bits

Sixth: Important Assumptions and Details

All important code details are commented in the code snippets above.

We did cover most of the wrong input cases, however, some minor corner cases were not covered assuming the input correctness by the user.

In part one, with the setBit() function, the user cannot set a bit that is not in range of the original number, for example if we are given a number whose binary representation is 1000, the user can set bits 0, 1, 2, and 3, but nothing else.

In part two, we didn’t ask the user for the length of either the Universal Set or the subsets, instead, the user should enter ‘-1’ after finishing the set. (This is shown in the terminal window)

The maximum allowed number of elements in the Universal Set is 10000.