Objective:

- To get a grip on Array Structure manipulation on heap
- Gets your hands dirty with pointers ☺
- Dynamic Memory Allocation

Task: ***Sets***

Write the following functions to support the set operations.

The struct names as 'Set' used for storing set information will be based on following members:

- int * data
  - It will point to an array of integers on heap treated as set of integers
- int capacity
  - it will store size of array pointed by '*int* *' treated as size of set/array
- int noOfElements
  - it will store number of Elements stored in set

Functions:

1. void createSet (Set &, int n);
2. bool addElement (Set &, int element);
3. bool removeElement (Set &, int element);
4. bool searchElement (Set, int element);
5. int searchElementPosition (Set, int element);
6. bool isEmpty(Set);
7. bool isFull(Set);
8. void displaySet (Set);
9. Set intersection (Set, Set);
10. Set calcUnion (Set, Set);
11. Set difference (Set, Set);
12. int isSubset (Set, Set);
    - 12.1. return 1 if proper subset
    - 12.2. return 2 if improper subset
    - 12.3. return 0 if not a subset
13. void reSize(Set &, int newSize);
14. void displayPowerSet (Set);
15. Set creatClone (Set & source);
16. void deallocateSet (Set &);

Sample Run:

```
struct Set
{
    int * data;
    int capacity;
    int noOfElements;
};
void createSet ( Set &, int n );
bool addElement ( Set &, int element );
bool removeElement ( Set &, int element
);
bool searchElement ( Set, int element );
int searchElementPosition ( Set, int
element );
bool isEmpty( Set );
bool isFull( Set );
void displaySet ( Set );
Set intersection ( Set, Set );
Set calcUnion ( Set, Set );
Set difference ( Set, Set );
int isSubset ( Set, Set );
void reSize( Set &, int newSize );
void displayPowerSet ( Set );
void creatClone ( Set & source, Set &
Target );
void deallocateSet ( Set & );
```

```
int main()
{
    Set setA, setB;

    createSet(setA, 10);
    createSet(setB, 7);

    addElement(setA, 5);
    addElement(setA, 15);
    addElement(setA, 9);
    addElement(setA, 10);

    cout << "Set A Elements : ";
    displaySet(setA);

    addElement(setA, 9);
    addElement(setA, 17);
    addElement(setA, 95);

    cout << "Set B Elements : ";
    displaySet(setB);

    Set setC = intersection(setA, setB);

    cout << "Set C Elements : ";
    displaySet(setC);


    cout<<"\nPower Set of B : ";
    displayPowerSet ( setB );

    return 1;
}
```

Console

```
Set A Elements : { 5, 15, 9, 10 }
Set B Elements : { 9, 17, 95 }
Set C Elements : { 9 }

Power Set of B : { {}, { 9 }, { 17 }, { 95 }, { 9, 17 }. { 9, 95 }, { 17, 95 }, { 9, 17, 95 } }
```