BitMap

1 Struct

```
struct set {
  int n;
  unsigned int members;
};
```

2 Headers

```
#define MAX_MEMBERS 32

typedef struct set Set;
Set* setCreate(void);
void setDestroy(Set *set);
void setShow(char *title, Set *set);
void setInsert(Set *set, int i);
void setRemove(Set *set, int i);
Set *setCopy(Set *set);
Set *setUnion(Set *set1, Set *set2);
Set *setIntersection( Set *set1, Set *set2);
Set *setDifference(Set *set1, Set *set2);
int setIsMember(Set *set, int i);
int setIsSubset( Set *set1, Set *set2);
int setIsEqual( Set *set1, Set *set2);
int setNumberOfElements(Set *set);
Set *setComplement(Set *set);
```

3 Implementation

3.1 Set Cria

```
Set *setCreate(void){
  Set *set;
  set = (Set *)malloc(sizeof(Set));
  if (set != NULL) {
    set->n = MAX_MEMBERS;
    set->members = 0;
  }
  return set;
}
```

3.2 Set Destroy

```
void setDestroy(Set *set) {
  if (set) free(set);
}
```

3.3 Set is Member

```
int setIsMember(Set *set, int i){
  if (set==NULL) return 0;
  if (!setMemberValid(i)) return 0;
  return ((1<<i) & (set->members));
}
```

3.4 Set Show

```
void setShow(char* title, Set *set){
  int i, first=1;
  printf("%s = {", title);
  for (i = 0; i < MAX_MEMBERS; i++) {
    if (setIsMember(set, i)) {
      if (first) {
        printf("%d", i);
        first = 0;
      }
      else
        printf(",%d", i);
    }
  }
  printf("}\n\n");
}
```

3.5 Set Copy

```
Set* setCopy(Set *set){
  if (set==NULL) return NULL;
  Set * temp = malloc(sizeof(Set));

  return temp;
}
```

3.6 Set is Equal

```
int setIsEqual(Set *set1, Set *set2) {
  if (set1==NULL || set2 == NULL) return 0;
  if(set1->members == set2->members) return 1;

  return 0;
}
```

3.7 Set Insert

```
void setInsert(Set *set, int i) {
  if (set == NULL) return;
  int a = 0x01;
  a = a<<i;
  set->members = set->members | a;
}
```

3.8 Set Remove

```
void setRemove(Set *set, int i){
  if (set == NULL) return;
  int a = 0x01;
```

```
  a = a<<i;
  a = ~a;
  set->members = set->members & a;
}
```

3.9 Set Complement

```
Set *setComplement(Set *set){
  if (set == NULL) return NULL;
  Set * temp = setCopy(set);
  temp->members = ~set->members;
  return temp;
}
```

3.10 Set Union

```
Set *setUnion(Set *set1, Set *set2){
  if (set1==NULL || set2 == NULL) return NULL;
  Set * temp = setCreate();
  temp->members = set1->members | set2->members;
  return temp;
}
```

3.11 Set Intersection

```
Set *setIntersection(Set *set1, Set *set2){
  if (set1==NULL || set2 == NULL) return NULL;
  Set * temp = setCreate();
  temp->members = set1->members & set2->members;
  return temp;
}
```

3.12 Set Difference

```
Set *setDifference(Set *set1, Set *set2){
  if (set1==NULL || set2 == NULL) return NULL;
  Set * temp = setCreate();
  temp->members = set1->members & ~set2->members;

  return temp;
}
```

3.13 Set is Subset

```
int setIsSubset(Set *set1, Set *set2) {
  if (set1==NULL || set2 == NULL) return 0;
  int a;
  a = (~set1->members)&set2->members;
  //if (b == (a&b));
  if(a == 0)
    return 1;
  else
    return 0;
}
```

3.14 Set Number of Elements

```
int setNumberOfElements(Set *set){
  if (set==NULL) return 0;

  int i, counter=0;
  for (i = 0; i < MAX_MEMBERS; i++) {
    if (setIsMember(set, i)) {
      counter++;
    }
  }
  return counter;
}
```