SETS

Sets are a type of abstract data type that allows you to store a list of non-repeated values. Their name derives from the mathematical concept of finite sets.

Unlike an array, sets are ***unordered*** and ***unindexed***.

These are the two properties of a set: the data is ***unordered*** and it is not **duplicated**.

Sets have the most impact in mathematical set theory. These theories are used in many kinds of proofs, structures, and abstract algebra.

In computer science, set theory is useful if you need to collect data and do not care about their multiplcity or their order. In databases, especially for relational databases, sets are very useful. There are many commands that finds unions, intersections, and differences of different tables and sets of data.

Implementation

*Structures

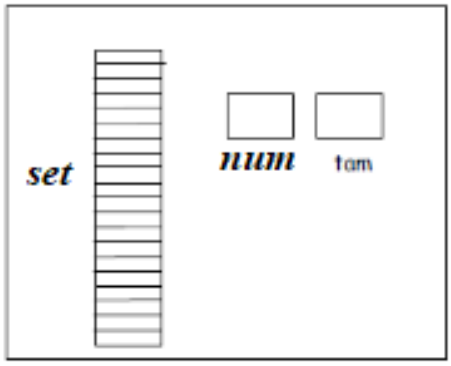
```
#define TAM noMaxElem

typedef struct
{
   <tipoDeDato> set[TAM];

    int num;

    int tam;
} SET;

SET miConjunto;
```



*Basic operations

Function	**Objetive**
iniSet(set)	Initialice the set
insert(set, i)	Adds *i* item to the set, *without duplicated*
remove(set, i)	Removes *i* item from the set
size(set)	Returns the size of the set
contains(set, i)	Returns whether or not the set contains *i*

```
int insert(SET *s, int dato)
{
 int res = 0, cont;

 if(s->num < s->tam)    /* ¿Hay lugar? */
 {
  cont=0;
  while(cont < s->num  && dato != s->set[cont])
     cont++;

  if( cont == s->num)            /* No repetido */
  {
    s->set[s->num] = dato;
    s->num++;
     res = 1;
   }
  }
  return(res);
}
```

```
int remove(SET *s, int dato)
{
 int res = 0, cont;

 cont=0;
 while(cont < s->num && dato != s->set[cont])
    cont++;

 if( cont < s->num)   /* ¿dato encontrado? */
 {
  s->set[cont]=s->set[s->num-1];
  s->num--;
  res = 1;
 }

 return(res);
}
```

```
int iniSet(SET *s, int t)
{
  int res = 0;

  if(t <= TAM)
  {
    s->tam = t;
    s->num = 0;
    res = 1;
  }

 return(res);
}
```

```
int sizeSet(SET s)
{
  return(s.tam);
}
```

```
int noElem(SET s)
{
   return(s.num);
}
```

```
int contains(SET s, int dato)
{
  int res = 0, cont;

  cont=0;
  while(cont < s.num && dato != s.set[cont])
    cont++;

  if(cont < s.num)
    res = 1;

 return(res);
}
```

Some operations are implemented that allow interactions between two sets:

Function	**Objetive**
union(set1, set2)	Returns the union of set *set1* and *set2*
intersection(set1, set2)	Returns the intersection of set *set1* and *set2*
difference(set1, set2)	Returns the difference of set *set1* and *set2*
subset(subSet, set)	Returns whether or not set sub*Set* is a subset of set *set*

```
int unionSet(SET s1, SET s2, SET *s3)
{
  int res, cont;

  res = iniSet(s3, s1.num + s2.num);
  if(res == 1)
  {
    for(cont=0; cont < s1.num; cont++)
        insert(s3, s1.set[cont]);
    for(cont=0; cont < s2.num; cont++)
        insert(s3, s2.set[cont]);
   }

  return(res);
}
```

```
int intersSet(SET s1, SET s2, SET *s3)
{
  int res, cont;

  res = iniSet(s3, s1.num < s2.num ? s1.num:s2.num);
  if(res == 1)
  {
    for(cont=0; cont < s1.num; cont++)
        if( contains(s2, s1.set[cont]) == 1)
              insert(s3, s1.set[cont]);
   }

  return(res);
}
```

```
int diffSet(SET s1, SET s2, SET *s3)
{
  int res, cont;

  res = iniSet(s3, s1.num);
  if(res == 1)
  {
    for(cont=0; cont < s1.num; cont++)
        if( contains(s2, s1.set[cont]) == 0)
              insert(s3, s1.set[cont]);
   }

  return(res);
}
```

```
int isSubSet(SET subSet, SET s)
{
  int cont, band = 0;

  if(subSet.num <= s.num)
  {
    band=1;
    for(cont=0; cont < subSet.num && band == 1; cont++)
          band = contains(s, subSet.set[cont]);
  }

  return(band);
}
```