Ejemplos
Tipos abstractos de datos y objetos

Problema 1.

```
TAD Boolean(VALORES: True, False;
            OPERACIONES: Not, And, Or, Implica, SiysoloSi)

Sintaxis:

   *True                        → Boolean
   *False                       → Boolean
   Inic(Boolean)                → Boolean
   Not(Boolean)                 → Boolean
   And(Boolean, Boolean)        → Boolean
   Or(Boolean, Boolean)         → Boolean

Semántica:  para todo p de tipo Boolean

INIC(p) = p
NOT(TRUE) = FALSE
NOT(NOT(p)) = p
p OR NOT(p) = TRUE
p OR p = p
p AND NOT(p) = FALSE
p AND p = p
```

Problema 2

```
TAD Entero(VALORES: Números enteros;
           OPERACIONES: Cero, Sucesor, Antecesor, Suma, Resta Producto)

Sintaxis:

   *Cero                        → Entero
   *Sucesor(Entero)             → Entero
   *Antecesor(Entero)           → Entero
    Suma(Entero, Entero)        → Entero
    Diferencia(Entero, Entero)  → Entero
    Producto(Entero, Entero)    → Entero

Semántica:  para todo m, n de tipo entero
    Sucesor(Antecesor(n))       ⇒ n
    Antecesor(Sucesor(n))       ⇒ n
    Suma(n, Cero)               ⇒ n
    Suma(n, Sucesor(m))         ⇒ Sucesor(Suma(n, m))
    Suma(n, Antecesor(m))       ⇒ Antecesor(Suma(n, m))
    Diferencia(n, Cero)         ⇒ n
    Diferencia(n, Sucesor(m))   ⇒ Antecesor(Diferencia(n, m))
    Diferencia(n, Antecesor(m)) ⇒ Sucesor(Diferencia(n, m))
    Producto(n, Cero)           ⇒ Cero
    Producto(n, Sucesor(m))     ⇒ Suma(Producto(n, m), n)
    Producto(n, Antecesor(m))   ⇒ Diferencia(Producto(n, m), n)
```

Problema 3.

```
TAD Vector (VALORES: Elementos de un cierto tipo;
             OPERACIONES: Crear, Existe, Asignar y Valor)

Sintaxis:
   *Crear                                → Vector
   *Asignar(Vector, Entero, Elemento)    → Vector
    Existe(Vector, Entero)               → Boolean
```

```
   Valor(Vector, Entero)                → Elemento

Semántica: Para todo m, n de tipo entero; para todo e, e1, e2 elemento; para todo v de tipo
           vector.

   /*Se supone que m, n son enteros dentro del rango comienzo, final. La primera
   especificación es una ecuación que indica la semántica del constructor Asignar*/

   Asignar(Asignar(v, m, e1), n, e))     ⇒ si m = n entonces
                                             Asignar(v, m, e)
                                           si_no
                                             Asignar(Asignar(v, n, e), m, e1))
                                           fin_si
   Existe(Crear, m)                      ⇒ False
   Existe(Asignar(v, m, e), n)             ⇒ or(m = n, Existe(v, n))
   Valor(Crear, m)                       ⇒ Error
   Valor(Asignar(v, m, e), n)            ⇒ si m = n entonces
                                             e
                                           si_no
                                             Valor(v, n)
                                           fin_si
```

Problema 4.

```
TAD Conjunto(VALORES: Conjunto de elementos sin repetición;
             OPERACIONES: ConjuntoVacio, Anade, Borra, EsVacío, Pertenece, Unión,
                          Intersección, Diferencia, DiferenciaSimétrica, Igual,
                          Incluido, Cardinal)

Sintaxis:

   *ConjuntoVacio                            → Conjunto
   *Anade(Conjunto, elemento)                → Conjunto
   Borra(Conjunto, Elemento)                 → Conjunto
   Esvacío(Conjunto)                         → Bolean
   Pertenece(conjunto, elemento)             → Boolean
   Unión(Conjunto, Conjunto)                 → Conjunto
   Intersección(Conjunto, conjunto)          → Conjunto
   Diferencia(Conjunto, Conjunto             → Conjunto
   DiferenciaSimétrica(Conjunto, Conjunto)   → Conjunto
   Igual(Conjunto, Conjunto)                 → Boolean
   Incluido(Conjunto, Conjuto)               → Boolean
   Cardinal(Conjunto)                        → Natural

Semántica: Para todo e, e1 de tipo elemento y todo C de tipo conjunto

    Anade(Anade(C, e), e)                    ⇒ Anade(C, e)
    Anade(Anade(C, e), e1)                   ⇒ Anade(Anade(C, e1), e)
    Borra(ConjuntoVacío, e)                  ⇒ ConjuntoVacio
    Borra(Anade(C, e1), e)                   ⇒
                                           si Iguales(e, e1) entonces
                                                Borra(C, e)
                                           si_no
                                                Anade(Borra(C, e),e1)
                                           fin_si
    EsVacio(ConjuntoVacio)                   ⇒ True
    Esvacio(Anade(C, e))                     ⇒False
    Pertenece(ConjuntoVacío, e)              ⇒ false
    Pertenece(Anade(C,e1),e)                 ⇒
                                           si Iguales(e, e1) entonces
                                                True
                                           si_no
                                                Pertenece(C, e)
                                           fin_si

    Unión(ConjuntoVacio C)                   ⇒ C
```

```
Unión(anade(C, e), D)                    ⇒ Anade(unión(C, D), e)
Intersección(ConjuntoVacio, C)           ⇒ ConjuntoVacio
Intersección(Anade(C, e), D)             ⇒
                                         si Pertenece(D, e) entonces
                                            Anade(Intersección(C, D), e)
                                         si_no
                                            Intersección(C, D)
                                         fin_si
Diferencia(C, ConjuntoVacio)             ⇒ C
Diferencia(Anade(C, e), D)               ⇒
                                         si Pertenece(D, e) entonces
                                            Diferencia(C, D)
                                         si_no
                                            Anade(Diferencia(C, D), e)
                                         fin_si
DiferenciaSimétrica(C, D)                ⇒ Diferencia(Unión(C,D), Intersección(C,D))
Igual(C, D)                              ⇒
            si  Union(Diferencia(C, D), Diferencia(D, C)) es ConjuntoVacio entonces
               True
            si_no
               False
            fin_si
Incluido(C, D)                           ⇒
            si Diferencia(C, D) es ConjuntoVacio entonces
               True
            si_no
               False
            fin_si
Cardinal(ConjutoVacio)                   ⇒ Cero
Cardinal(Anade(C, e))                    ⇒
                                         si Pertenece(C, e) entonces
                                            Cardinal(C)
                                         si_no
                                            Sucesor(Cardinal(C))
                                         fin_si
```

Implementacion de TADConjunto en C++.

```
#include <stdio.h>
typedef struct
{
   int Numelementos;
   int L[512];          /* empieza en 0*/
} Conjunto;

int  Errorconjunto;
void ConjuntoVacio(Conjunto *c);
void Anade(Conjunto *c,int n);
int EsVacio(Conjunto c);
int Pertenece(Conjunto c, int n);
void Borra(Conjunto *c,int n);
int Cardinal(Conjunto c);
void Buscar(Conjunto c, int n, int *encontrado, int *p);

void ConjuntoVacio(Conjunto *c)
{
    Errorconjunto = 0;
    c->Numelementos = 0 ;
}

void Buscar(Conjunto c, int n, int *encontrado, int *p)
{
  // la búsqueda se realiza de forma binaria.
    int  primero, ultimo, central;
    primero = 0;
    ultimo = c.Numelementos - 1;
    encontrado = 0;
```

```
    while ((primero <= ultimo)&&(! encontrado))
   {
      central = (primero + ultimo)/ 2;
      if (c.L[central] == n)
         *encontrado = 1;
      else
         if (c.L[central] < n)
              primero = central + 1;
         else
              ultimo = central - 1 ;
    }
    if (!encontrado)
      *p = primero;
    else
      *p = central;
};

void Anade(Conjunto *c,int n)
{
    int encontrado, p, i;
    Buscar(*c, n, &encontrado, &p);
    if (!encontrado)
      if (c->Numelementos < 512)
      {
         for (i = c->Numelementos-1; i >= p; i--)
            c->L[i+1] = c->L[i];                  // desplaza datos a la derecha.
         c->L[p] = n;
         c->Numelementos++;
      }
      else
         Errorconjunto = 1;
}

int EsVacio(Conjunto c)
{
    return(c.Numelementos == 0);
}

int Cardinal(Conjunto c)
{
    return(c.Numelementos);
}

int Pertenece(Conjunto c, int n)
{
    int encontrado, p;
    Buscar(c, n, &encontrado, &p);
    return( encontrado);
}

void Borra(Conjunto *c, int n)
{
    int encontrado,p,i;
    Buscar(*c, n, &encontrado, &p);
    if (encontrado)
    {
          // desplaza datos hacia la izquierda.
      for (i = p; i < c->Numelementos - 1 ; i++)
         c->L[i] = c->L[i + 1];
      c->Numelementos --;
    }
}
```

Problema 7.13

```
void Union(Conjunto c1, Conjunto c2,Conjunto *c3);
void Interseccion(Conjunto c1, Conjunto c2,Conjunto *c3);
void Diferencia(Conjunto c1, Conjunto c2,Conjunto *c3);
void Diferenciasimetrica(Conjunto c1, Conjunto c2,Conjunto *c3);
int Igual(Conjunto c1, Conjunto c2);
int Incluido(Conjunto c1, Conjunto c2);
```

```
void Union(Conjunto c1, Conjunto c2,Conjunto *c3)
{
   int i = 0, j = 0, k = -1;
   while ((i <= Cardinal(c1) - 1) && (j <= Cardinal(c2)- 1) && (! Errorconjunto))
   {
      k ++;
      if (k > 511)
         Errorconjunto = 1;
      else
      {
         c3->Numelementos = k+1;
         if (c1.L[i] < c2.L[j])
         {
            c3->L[k] = c1.L[i];
            i ++;
         }
         else
           if(c1.L[i] > c2.L[j])
           {
              c3->L[k] = c2.L[j];
              j ++;
           }
           else
             if (c1.L[i]== c2.L[j])
             {
                c3->L[k] = c2.L[j];
                j ++;
                i ++;
             }
      }
   }

   while ((i <= Cardinal(c1)-1) && (! Errorconjunto))
   {
       k ++;
       if (k > 511)
         Errorconjunto =1;
       else
       {
          c3->Numelementos = k+1;
          c3->L[k] = c1.L[i];
          i++;
       }
   }
   while ((j <= Cardinal(c2)-1) && (! Errorconjunto))
   {
       k ++;
       if (k > 511)
          Errorconjunto = 1;
       else
       {
          c3->Numelementos = k;
          c3->L[k] = c2.L[j];
          j ++;
       }
   }
}

void Interseccion(Conjunto c1, Conjunto c2,Conjunto *c3)
{
   int i = 0, j = 0;
   c3->Numelementos = 0;
   while ((i <= Cardinal(c1)-1) && (j <= Cardinal(c2)-1))
     if (c1.L[i] < c2.L[j])
       i ++;
     else
       if (c1.L[i] > c2.L[j])
          j ++;
       else
          if (c1.L[i] == c2.L[j])
          {
             c3->Numelementos ++;
             c3->L[c3->Numelementos-1]= c2.L[j]; j ++ ;
```

```
            i ++ ;
         }
}

void Diferencia(Conjunto c1, Conjunto c2,Conjunto *c3)
{
   int  i = 0, j = 0, r1;
   c3->Numelementos = 0;
   while ((i <= Cardinal(c1)-1) && (j <= Cardinal(c2)-1))
     if (c1.L[i] < c2.L[j])
     {
        c3->Numelementos ++;
        c3->L[c3->Numelementos-1] = c1.L[i];
        i ++;
     }
     else
       if (c1.L[i] > c2.L[j])
         j ++;
       else
         if (c1.L[i] == c2.L[j])
         {
            j ++;
            i ++;
         };
   for (r1 = i;i< c1.Numelementos;r1++)
   {
       c3->L[c3->Numelementos]=c1.L[r1];
       c3->Numelementos++;
   }
}

void Diferenciasimetrica(Conjunto c1, Conjunto c2,Conjunto *c3)
{
    Conjunto  cu, ci;
    Union(c1, c2, &cu);
    Interseccion(c1, c2, &ci);
    Diferencia(cu, ci, c3);
}

int Incluido(Conjunto c1, Conjunto c2)
{
    Conjunto cd;
    Diferencia(c1, c2, &cd);
    return (EsVacio(cd));
}

int Igual(Conjunto c1, Conjunto c2)
{
    Conjunto cd1, cd2, cu;
    Diferencia(c1, c2, &cd1);
    Diferencia(c2, c1, &cd2);
    Union(cd1, cd2, &cu);
    return(EsVacio(cu));
}
```