Tareas

Díaz Urbina Eduardo
Boleta: 2012630487
Grupo: 1CV13

Tarea (búsqueda)

a) Escribir un programa que realice una búsqueda secuencial en una lista lineal vinculada en la que se puedan ingresar y eliminar datos. Debe incluir las mejoras de mover al frente y transposición, si un elemento se ha buscado más de tres veces, se aplica mover al frente, de lo contrario se aplica transposición.

Busqueda/TareaA.c

```c
#include <stdio.h>
#include <stdlib.h>

typedef char ListEntry;
typedef ListEntry Entry;

#include "DynamicList.h"

Node* createNode(Entry e){
   Node *n = (Node*)malloc(sizeof(Node));
      n->entry = e;
      n->t = 0;
      n->next = NULL;
   return n;
}

int equalNode(Node *a,Node *b){
   if(a->entry == b->entry)
      return 1;
   return 0;
}

/*Busca secuencialmente en una lista lineal vinculada,
**si un elemento se ha buscado más de 3 veces se mueve
**al frente, de lo contrario se transpone, retorna la
**posición donde lo encontró*/
int busq(Entry entry,List *l){
   if(empty(l))
      return -1;

   Node *aux = l->start,*aux2;

   /*Si está en la 1ra o 2da posición,
   **aquí, mover al frente es igual a transponer*/
   if(aux->entry == entry){
      aux->t++;
      return 1;
   }else if(aux->next != NULL && aux->next->entry == entry){
      aux2 = aux->next;
      aux->next = aux2->next;
      aux2->next = aux;
      l->start = aux2;
```

```
43          aux2->t++;
44          return 1;
45      }else if(aux->next == NULL)
46          return -1;
47
48      int pos = 3;
49      /*Si está en i-ésima posición,i > 2, sigue buscando*/
50      while(aux->next->next != NULL && aux->next->next->entry !=
            entry){
51          aux = aux->next;
52          pos++;
53      }
54
55      /*Si no lo encontró*/
56      if(aux->next->next == NULL)
57          return -1;
58
59      /*Si lo encontró, se registra otra búsqueda más para ese nodo*/
60      aux->next->next->t++;
61      /*Se mueve al frente/transpone*/
62      if(aux->next->next->t >= 3){
63          aux2 = aux->next->next;
64          aux->next->next = aux->next->next->next;
65          aux2->next = l->start;
66          l->start = aux2;
67          pos = 1;
68      }else{
69          aux2 = aux->next->next;
70          aux->next->next = aux->next->next->next;
71          aux2->next = aux->next;
72          aux->next = aux2;
73          pos--;
74      }
75
76      return pos;
77  }
78
79  void printList(List *l){
80      Node *aux = l->start;
81
82      puts("");
83      while(aux != NULL){
84          printf("(%c,%d)",aux->entry,aux->t);
85          aux = aux->next;
86      }
87      puts("");
88  }
89
90  int main(){
91      List l = createList();
92
93      int i,j;
94      for(i = 0;i < 12;i++)
95          insert(createNode('a' + i),i,&l);
96
97      printList(&l);
```

```
98
99      char c[7] = {'d','j','j','l','j','a','d'};
100
101     for(j = 0; j < 7; j++){
102        i = busq(c[j],&l);
103        printList(&l);
104        printf("%50c+++|%c en %d",' ',c[j],i);
105     }
106
107     return 0;
108 }
```

Capturas de Pantalla de Busqueda/TareaA.c

```
(b,0)(c,0)(d,0)(e,0)(f,0)(g,0)(h,0)(i,0)(j,0)(k,0)(l,0)

(b,0)(d,1)(c,0)(e,0)(f,0)(g,0)(h,0)(i,0)(j,0)(k,0)(l,0)
                                                  +++|d en 2
(b,0)(d,1)(c,0)(e,0)(f,0)(g,0)(h,0)(j,1)(i,0)(k,0)(l,0)
                                                  +++|j en 8
(b,0)(d,1)(c,0)(e,0)(f,0)(g,0)(j,2)(h,0)(i,0)(k,0)(l,0)
                                                  +++|j en 7
(b,0)(d,1)(c,0)(e,0)(f,0)(g,0)(j,2)(h,0)(i,0)(l,1)(k,0)
                                                  +++|l en 10
(j,3)(b,0)(d,1)(c,0)(e,0)(f,0)(g,0)(h,0)(i,0)(l,1)(k,0)
                                                  +++|j en 1
(j,3)(b,0)(d,1)(c,0)(e,0)(f,0)(g,0)(h,0)(i,0)(l,1)(k,0)
                                                  +++|a en -1
(j,3)(d,2)(b,0)(c,0)(e,0)(f,0)(g,0)(h,0)(i,0)(l,1)(k,0)
                                                  +++|d en 2
```

b) Hacer un programa para encontrar un elemento dentro de un arreglo usando búsqueda binaria, cada elemento del arreglo contiene la siguiente información: nombre, apellido paterno, apellido materno, boleta y promedio; la búsqueda se puede hacer tomando como llave cualquiera de estos elementos.

Busqueda/TareaB.c

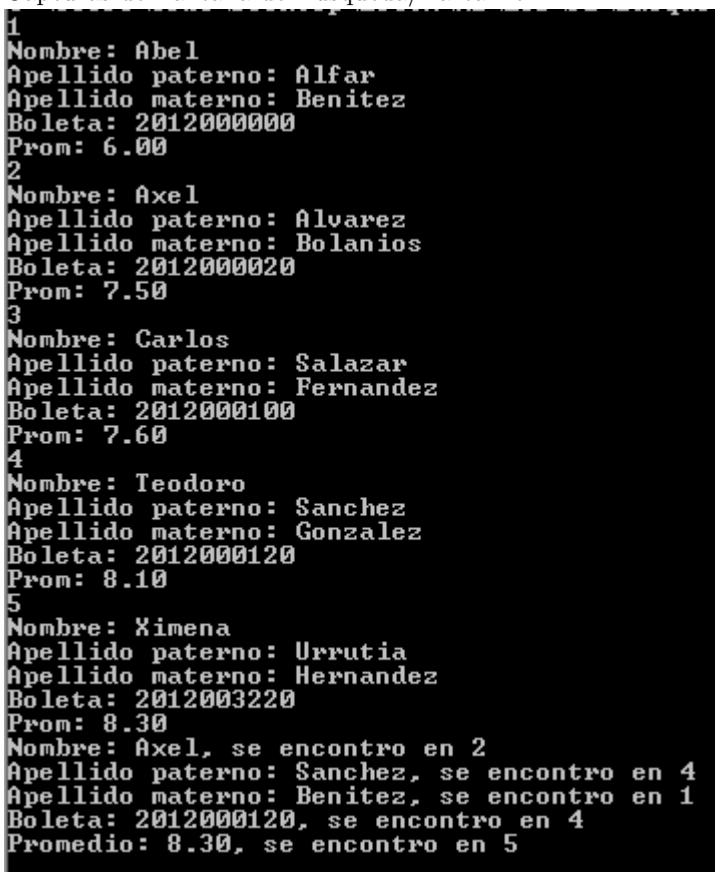
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define maxSize 20

typedef struct ListEntry{
   char nombre[21];
   char apeMat[21];
   char apePat[21];
   char boleta[11];
   double prom;
}ListEntry;

typedef union Key{
   char nombre[20];
   char apeMat[20];
   char apePat[20];
   char boleta[10];
   double prom;
}Key;

#include "StaticList.h"

Node createNode(char *nombre,char *apePat,char *apeMat,
                char *boleta,double prom){
   Node n;
      strncpy(n.entry.nombre,nombre,sizeof(n.entry.nombre));
      strncpy(n.entry.apeMat,apeMat,sizeof(n.entry.apeMat));
      strncpy(n.entry.apePat,apePat,sizeof(n.entry.apePat));
      strncpy(n.entry.boleta,boleta,sizeof(n.entry.boleta));
      n.entry.prom = prom;

   return n;
}

int equalNode(Node *a,Node *b){
   if(strncmp(a->entry.nombre,b->entry.nombre,
         sizeof(a->entry.nombre)) == 0 &&
      strncmp(a->entry.apeMat,b->entry.apeMat,
         sizeof(a->entry.apeMat)) == 0 &&
      strncmp(a->entry.apePat,b->entry.apePat,
         sizeof(a->entry.apePat)) == 0 &&
      strncmp(a->entry.boleta,b->entry.boleta,
         sizeof(a->entry.boleta)) == 0 &&
      a->entry.prom == b->entry.prom)
      return 1;
   return 0;
}

int keycmpNOMBRE(Key *k,Node *n){
```

```
52    return strncmp( k->nombre,n->entry.nombre,
53                    sizeof( n->entry.nombre ) );
54 }
55
56 int keycmpAPEPAT(Key *k,Node *n){
57    return strncmp( k->apePat,n->entry.apePat,
58                    sizeof( n->entry.apePat ) );
59 }
60
61 int keycmpAPEMAT(Key *k,Node *n){
62    return strncmp( k->apeMat,n->entry.apeMat,
63                    sizeof( n->entry.apeMat ) );
64 }
65
66 int keycmpBOLETA(Key *k,Node *n){
67    return strncmp( k->boleta,n->entry.boleta,
68                    sizeof( n->entry.boleta ) );
69 }
70
71 int keycmpPROM(Key *k,Node *n){
72    if( k->prom == n->entry.prom )
73       return 0;
74    else if( k->prom < n->entry.prom )
75       return -1;
76    return 1;
77 }
78
79 #define BUSQ_NOMBRE 32248
80 #define BUSQ_APEPAT 3223
81 #define BUSQ_APEMAT 32231
82 #define BUSQ_BOLETA 3242
83 #define BUSQ_PROM 323
84 /*Busqueda binaria, retorna -1 si no encuentra algo*/
85 /*Para que funcione, la llave de los elementos ya
86 debe estar ordenada*/
87 int busqBin( int busq , Key *k , List *l ){
88    int min = 0, max = l->size - 1;
89    int middle , aux = 0;
90
91    int (*keyCmp) ( Key* , Node* );
92
93    switch( busq ){
94       case BUSQ_NOMBRE:
95          keyCmp = &keycmpNOMBRE;
96          break;
97       case BUSQ_APEPAT:
98          keyCmp = &keycmpAPEPAT;
99          break;
100      case BUSQ_APEMAT:
101         keyCmp = &keycmpAPEMAT;
102         break;
103      case BUSQ_BOLETA:
104         keyCmp = &keycmpBOLETA;
105         break;
106      case BUSQ_PROM:
107         keyCmp = &keycmpPROM;
```

```
108             break;
109         }
110
111         while(min <= max){
112             middle = (min + max) / 2;
113             aux = keyCmp( k , &l->nodes[middle] );
114             if( aux == 0 )
115                 return middle;
116             else if( aux < 0 )
117                 max = middle - 1;
118             else if( 0 < aux )
119                 min = middle + 1;
120         }
121
122         return -1;
123     }
124
125     void print( List *l ){
126         int i;
127         for(i = 0; i < l->size; i++){
128             printf("%d\n",i+1);
129             printf("Nombre: ");
130             puts(l->nodes[i].entry.nombre);
131             printf("Apellido paterno: ");
132             puts(l->nodes[i].entry.apePat);
133             printf("Apellido materno: ");
134             puts(l->nodes[i].entry.apeMat);
135             printf("Boleta: ");
136             puts(l->nodes[i].entry.boleta);
137             printf("Prom: %.2f\n",l->nodes[i].entry.prom);
138         }
139     }
140
141     int main(){
142         List l = createList();
143         Key k;
144         int i;
145
146         insert( createNode( "Abel" , "Alfar" , "Benitez" ,
147                             "2012000000" , 6.0 ) , 1 , &l );
148         insert( createNode( "Axel" , "Alvarez" , "Bolanios" ,
149                             "2012000020" , 7.5 ) , 2 , &l );
150         insert( createNode( "Carlos" , "Salazar" , "Fernandez" ,
151                             "2012000100" , 7.6 ) , 3 , &l );
152         insert( createNode( "Teodoro" , "Sanchez" , "Gonzalez" ,
153                             "2012000120" , 8.1 ) , 4 , &l );
154         insert( createNode( "Ximena" , "Urrutia" , "Hernandez" ,
155                             "2012003220" , 8.3 ) , 5 , &l );
156
157         print( &l );
158
159         strncpy( k.nombre , "Axel" , 20 );
160         i = busqBin( BUSQ_NOMBRE , &k , &l );
161         printf( "Nombre: %s, se encontro en %d\n" ,
162                  k.nombre , i+1 );
163
```

```
164     strncpy( k.apePat , "Sanchez" , 20 );
165     i = busqBin( BUSQ_APEPAT , &k , &l );
166     printf( "Apellido paterno: %s, se encontro en %d\n" ,
167             k.apePat , i+1 );
168
169     strncpy( k.apeMat , "Benitez" , 20 );
170     i = busqBin( BUSQ_APEMAT , &k , &l );
171     printf( "Apellido materno: %s, se encontro en %d\n" ,
172             k.apeMat , i+1 );
173
174     strncpy( k.boleta , "2012000120" , 10 );
175     i = busqBin( BUSQ_BOLETA , &k , &l );
176     printf( "Boleta: %s, se encontro en %d\n" ,
177             k.boleta , i+1 );
178
179     k.prom = 8.3;
180     i = busqBin( BUSQ_PROM , &k , &l );
181     printf( "Promedio: %.2f, se encontro en %d\n" ,
182             k.prom , i+1 );
183
184     return 0;
185 }
```

Capturas de Pantalla de Busqueda/TareaB.c



c) Programar la búsqueda por interpolación para una lista de trabajadores que contiene los siguientes datos: número de identificación (del 000 al 999), nombre, puesto y sueldo. La llave en este caso es el número de identificación.

Busqueda/TareaC.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define maxSize 20

typedef struct ListEntry{
   int id;
   char nombre[21];
   char puesto[21];
   double sueldo;
}ListEntry;

#include "StaticList.h"

Node createNode(int id , char *nombre,
                char *puesto , double sueldo){
   Node n;
      n.entry.id = id%1000;
      strncpy(n.entry.nombre,nombre,sizeof(n.entry.nombre));
      strncpy(n.entry.puesto,puesto,sizeof(n.entry.puesto));
      n.entry.sueldo = sueldo;

   return n;
}

int equalNode(Node *a,Node *b){
   if(a->entry.id == b->entry.id &&
      strncmp(a->entry.nombre,b->entry.nombre,
         sizeof(a->entry.nombre)) == 0 &&
      strncmp(a->entry.puesto,b->entry.puesto,
         sizeof(a->entry.puesto)) == 0 &&
      a->entry.sueldo == b->entry.sueldo)
      return 1;
   return 0;
}

int interpol( int id , List *l ){
   int min = 0, max = l->size - 1;
   int i;

   while(min <= max){
      i = min;
      if( min != max )
      i += (max-min)/
           (l->nodes[max].entry.id -
            l->nodes[min].entry.id)*
           (id - min);

      if( l->nodes[i].entry.id == id)
         return i;
      else if( id < l->nodes[i].entry.id )
```

```
53          max = i - 1;
54       else if( l->nodes[i].entry.id < id )
55          min = i + 1;
56
57    }
58
59    return -1;
60 }
61
62 void print( List *l ){
63    int i;
64    for(i = 0; i < l->size; i++){
65       printf("%d\n",i);
66       printf("ID: %d\n",l->nodes[i].entry.id);
67       printf("Nombre: ");
68       puts(l->nodes[i].entry.nombre);
69       printf("Puesto: ");
70       puts(l->nodes[i].entry.puesto);
71       printf("Sueldo: %.2f\n",l->nodes[i].entry.sueldo);
72    }
73 }
74
75 int main(){
76    List l = createList();
77    int i,j;
78
79    insert( createNode( 234 , "Alberto" ,
80                        "Puesto_1" , 4232.0 ) , 1 , &l );
81    insert( createNode( 780 , "Juan" ,
82                        "Puesto_4" , 3209.0 ) , 2 , &l );
83    insert( createNode( 781 , "Pablo" ,
84                        "Puesto_7" , 6210.0 ) , 3 , &l );
85    insert( createNode( 890 , "Hector" ,
86                        "Puesto_8" , 7242.0 ) , 4 , &l );
87    insert( createNode( 999 , "Jose" ,
88                        "Puesto_3" , 4032.0 ) , 5 , &l );
89
90    int a[9] = {230,781,978,999,234,780,1000,-3,1001};
91
92    print( &l );
93    for(j = 0; j < 9; j++){
94       i = interpol( a[j] , &l );
95       printf("-------ID: %d encontrado en %d\n",a[j],i);
96    }
97
98    return 0;
99 }
```

Capturas de Pantalla de Busqueda/TareaC.c


0
ID: 234
Nombre: Alberto
Puesto: Puesto_1
Sueldo: 4232.00
1
ID: 780
Nombre: Juan
Puesto: Puesto_4
Sueldo: 3209.00
2
ID: 781
Nombre: Pablo
Puesto: Puesto_7
Sueldo: 6210.00
3
ID: 890
Nombre: Hector
Puesto: Puesto_8
Sueldo: 7242.00
4
ID: 999
Nombre: Jose
Puesto: Puesto_3
Sueldo: 4032.00
-------ID: 230 encontrado en -1
-------ID: 781 encontrado en 2
-------ID: 978 encontrado en -1
-------ID: 999 encontrado en 4
-------ID: 234 encontrado en 0
-------ID: 780 encontrado en 1
-------ID: 1000 encontrado en -1
-------ID: -3 encontrado en -1
-------ID: 1001 encontrado en -1

d) Hacer un programa de tablas hash abiertas. Las llaves en este caso son número enteros largos. El programa debe mostrar la manera en que queda la tabla después de insertar o eliminar los elementos. Proponer la función hash a utilizar y justificarla como comentario en el programa.

e) Hacer un programa de tablas hash cerradas. Las llaves en este caso son número enteros largos. El programa debe mostrar la manera en que queda la tabla después de insertar o eliminar los elementos. Proponer las funciones hash a utilizar y justificarla como comentario en el programa.

f) Se dispone de una aplicación de radares de tráfico que permite llevar la contabilidad del número de veces que ha pasado un determinado coche por dicho radar superando el límite de velocidad. Para ello se consulta una lista implementada mediante una Tabla Hash. Sabiendo que el formato de una matrícula consta de una serie de 3 números seguidos de 3 letras, hacer un programa que genere dicha lista con el fin de contabilizar el número de veces que ha pasado un coche; si es necesario, se debe actualizar la lista de la aplicación. Si la matrícula no está en la lista entonces el coche ha sido visto por primera vez. Si la matrícula ya estaba en la lista entonces hay que guardar en lista que se ha visto ese coche una vez más.

Tarea (recursión)

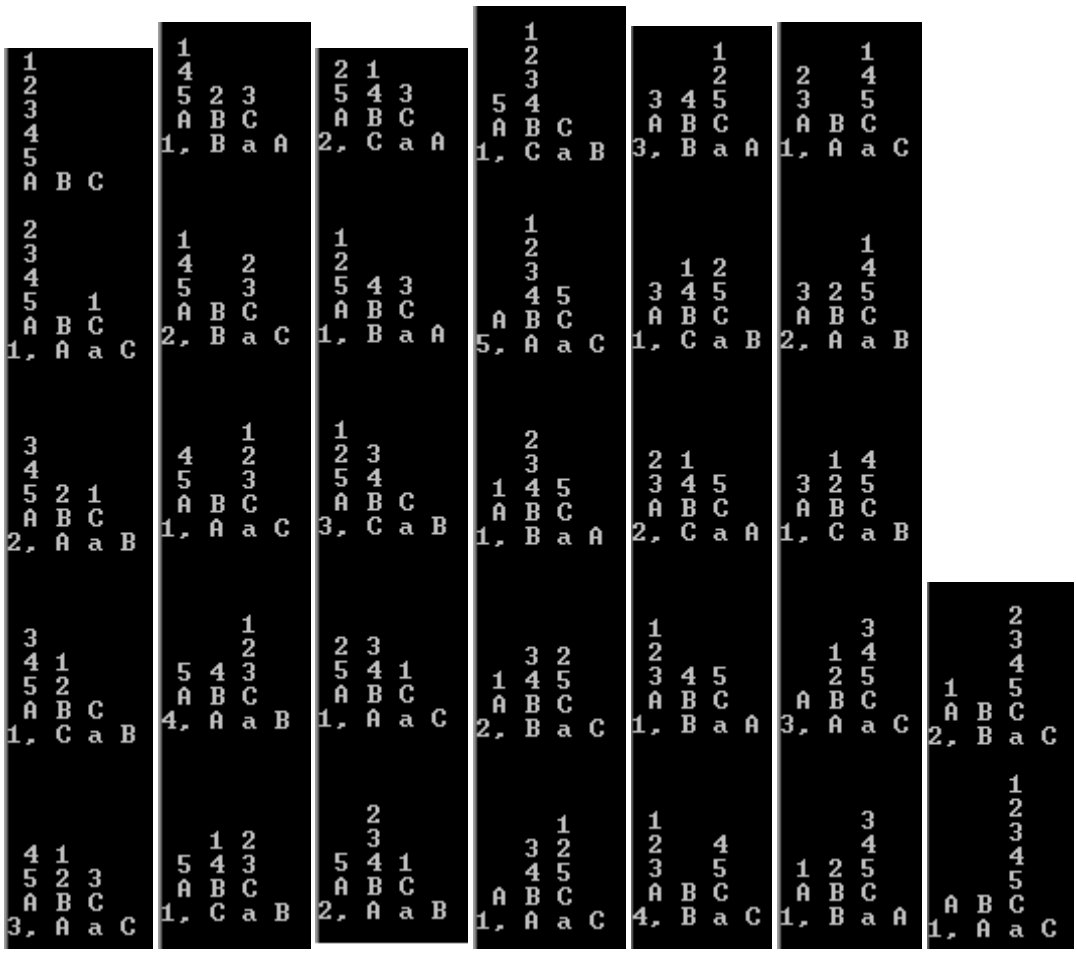
a) Resolver el problema de Las Torres de Hanoi para cinco discos, mostrar la solución en forma gráfica.

Recursion/TareaA.c

```
#include <stdio.h>

#define maxSize 5

typedef int Node;

typedef struct Stack{
   int size;
   char c;
   Node nodes[maxSize];
}Stack;

#define createStack(s,ch) do{ \
   (s).size = 0; \
   (s).c = ch; \
}while(0)

#define push(x,s) do{ \
   (s).nodes[(s).size++] = x; \
}while(0)

#define top(s) (s).nodes[(s).size - 1]

#define pop(s) do{ (s).size--; }while(0)

#define size(s) (s).size

void display();

void hanoi( int , Stack* , Stack* , Stack* );

Stack t[3];

int main(){
   int i;
      createStack( t[0] , 'A' );
      createStack( t[1] , 'B' );
      createStack( t[2] , 'C' );

   for(i = maxSize; i > 0; i--)
      push( i , t[0] );

   display();
   hanoi( size( t[0] ) , t , t+1 , t+2 );

   return 0;
}

void display(){
   int i,j;
   for(i = maxSize; i > 0; i--){
```

```
52        for(j = 0; j < 3; j++){
53           if( i <= size(t[j]) )
54              printf("%2d",t[j].nodes[i - 1]);
55           else
56              printf("%2s"," ");
57        }
58        puts("");
59     }
60     printf("%2s%2s%2s\n","A","B","C");
61  }
62
63  void hanoi( int d , Stack *a ,
64                      Stack *b ,
65                      Stack *c ){
66     if( d > 0 ){
67        hanoi( d - 1 , a , c , b );
68
69        push( top( *a ) , *c );
70        pop( *a );
71        display();
72        printf( "%d, %c a %c\n\n" ,
73                d , a->c , c->c );
74
75        hanoi( d - 1 , b , a , c );
76     }
77  }
```

Capturas de Pantalla Recursion/TareaA.c



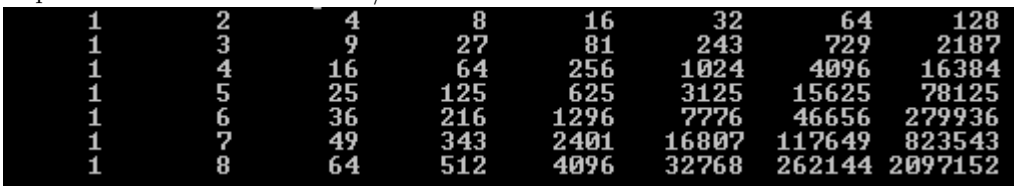
b) Hacer un programa que, dados dos números, a (número entero) y b (número natural mayor o igual que cero), determine recursivamente a^b (a elevado a la b).

Recursion/TareaB.c

```c
#include <stdio.h>

int pow2(int a,int b);

int main(){
   int a = 3,b;

   node_d();

   for(a = 2;a < 9;a++){
      for(b = 0;b < 8;b++)
         printf("%7d ",pow2(a,b));
      puts("");
   }
   return 0;
}

/*Si b es par, se calcula la raíz cuadrada de a^b,
y se devuelve el cuadrado de esta última; si b es
impar, se calcula la raiz cuadrada de a^b y se devuelve
el cuadrado, pero multiplicado por a*/
int pow2( int a , int b ){
   /*Caso base*/
   if( b == 0 )
      return 1;
   /*Recursión*/
   if( b % 2 ){
      b = pow2( a , b/2 );
      return a*b*b;
   }else{
      a = pow2( a , b/2 );
      return a*a;
   }
}
/*
NO LEER SI SE CREE EN EL FUNCIONAMIENTO DEL ALGORITMO

Demostración de que funciona el algoritmo recursivo:

--Inducción sobre b
+Casos bases:
b = 0:
   pow2(a,0) = 1 = a^0
b = 1:
   pow2(a,1) = a*pow2(a,0)*pow2(a,0) = a*1*1 = a = a^1
b = 2:
   pow2(a,2) = pow2(a,1)*pow2(a,1) = a*a = a^2

+Inducción (completa):
-Con cierto b > 1,
pow2(a,0), pow2(a,1), pow2(a,2),..., pow2(a,b)
funcionan y son correctos para cualquier a

```

```
-Con b+1,

Si b+1 es impar:
   pow2(a,b+1) = a*pow2(a,(b+1)/2)*pow2(a,(b+1)/2),
      ya que (b+1)/2 = b/2 en división entera,
      se puede reescribir como:
   pow2(a,b+1) = a*pow2(a,b/2)*pow2(a,b/2),
      como b/2 < b, pow2(a,b/2) funciona, es correcto
      e igual a a^(b/2),
   pow2(a,b+1) =
      a*a^(b/2)*a^(b/2) = a^(1+b/2+b/2) = a^(b+1)
Si b+1 es par:
   pow2(a,b+1) = pow2(a,(b+1)/2)*pow2(a,(b+1)/2),
      como (b+1)/2 < b, pow2(a,(b+1)/2) funciona,
      es correcto e igual a a^((b+1)/2)
   pow2(a,b+1) = a^((b+1)/2)*a^((b+1)/2) =
                 a^((b+1)/2+(b+1)/2) = a^(b+1)

Por lo que pow2(a,b+1) funciona y es correcto
para cualquier a, por lo tanto,

+Para todo b >= 0, pow2(a,b) funciona y es
correcto para cualquier a
*/
```

Capturas de Pantalla Recursion/TareaB.c



1	2	4	8	16	32	64	128
1	3	9	27	81	243	729	2187
1	4	16	64	256	1024	4096	16384
1	5	25	125	625	3125	15625	78125
1	6	36	216	1296	7776	46656	279936
1	7	49	343	2401	16807	117649	823543
1	8	64	512	4096	32768	262144	2097152

c) Hacer un programa que, dados dos arreglos de números enteros A y B de longitud n y m respectivamente, siendo $n >= m$, determine resursivamente si B está contenido en A. Por ejemplo, si $A = \{2, 3, 4, 5, 8, 9, -2\}$. Si $B = \{9, -2\}$ está contenido, si $B = \{4, 8\}$ no está contenido, si $B = \{5, 4\}$ no está contenido.

d) Suponga que $com(n, k)$ representa la cantidad de diferentes comités de k personas que pueden formarse, dadas n personas entre las cuales elegir. Por ejemplo, $com(4, 3) = 4$, porque dadas cuatro personas A,B, C y D hay cuatro comités de tres personas posibles: ABC, ABD, ACD y BCD. Se puede comprobar la identidad:

$com(n, k) = com(n - 1, k) + com(n - 1, k - 1)$

Escribir y probar un programa recursivo para calcular $com(n, k)$ para $n, k >= 1$.

Recursion/TareaD.c

```c
#include <stdio.h>

int com(int n,int k);

int main(){
   int n,k;

   for(n = 1;n < 13;n++){
      for(k = 1;k <= n + 1;k++)
         printf("%3d ",com(n,k));
      puts("");
   }

   return 0;
}

int com(int n,int k){
   /*Hay 0 formas de elegir más de n personas*/
   if(n < k)
      return 0;
   /*Hay n formas de elegir a 1 persona de n personas*/
   else if(k == 1)
      return n;
   else
      return com(n - 1,k) + com(n - 1,k - 1);
}
/*
NO LEER SI SE CREE EN EL FUNCIONAMIENTO DEL ALGORITMO

Demostración de que funciona el algoritmo recursivo,

+com(n,k) funciona, es correcto e igual a
0 cuando n < k, para cuando n >= k:

--Inducción sobre n
+Casos bases:
n = 1:
   k = 1:
   com(1,1) = 1
n = 2:
   k = 1:
   com(2,1) = 2
   k = 2:
   com(2,2) = com(1,2) + com(1,1) = 0 + 1 = 1

```

```
+Inducción (completa):
-Con cierto n >= 1,
com(0,k), com(1,k), com(2,k),..., com(n,k) funcionan y
son correctos desde k = 1 hasta k = n

-Con n+1,

com(n+1,k) = com(n,k) + com(n,k-1),
com(n,k) y com(n,k-1) funcionan y son correctos
desde k = 1 hasta k = n, por lo tanto com(n+1,k)
funciona y es correcto desde k = 1 hasta k = n,

falta revisar k = n+1
com(n+1,k) = com(n+1,n+1) = com(n,n+1) + com(n,n) =
0 + com(n,n) = com(n,n)
com(n,n) funciona y es correcto, por lo que,
com(n+1,k) funciona y es correcto desde k = 1 hasta k = n+1;
por lo tanto:

+Para todo n,k >= 1, com(n,k) funciona y es correcto
*/
```

Capturas de Pantalla de Recursion/TareaD.c

```
 1   0
 2   1   0
 3   3   1   0
 4   6   4   1   0
 5  10  10   5   1   0
 6  15  20  15   6   1   0
 7  21  35  35  21   7   1   0
 8  28  56  70  56  28   8   1   0
 9  36  84 126 126  84  36   9   1   0
10  45 120 210 252 210 120  45  10   1   0
11  55 165 330 462 462 330 165  55  11   1   0
12  66 220 495 792 924 792 495 220  66  12   1   0
```

e) Escribir un programa con una función recursiva que acepte una expresión prefija que conste de operadores binarios y operandos enteros de un solo dígito y retorne el valor de la expresión.

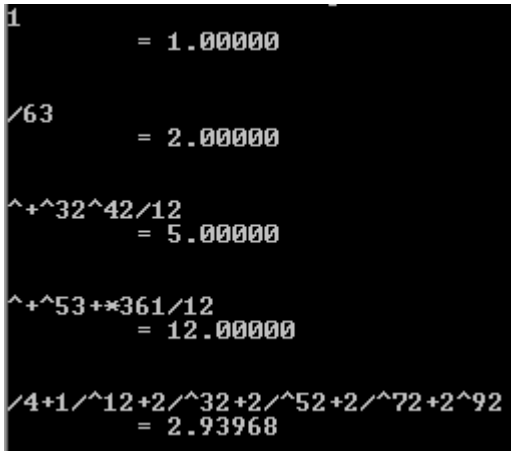
Recursion/TareaE.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>

float charToFloat(char c);

/*Hace la operación binaria indicada por op*/
float operate( char op, float a , float b );

/*Evalúa una expresión infija correctamente formada*/
/*  *expr apuntará al final de la expresión*/
float prefxSlv( char **expr );

int main(){
   char *t[5] = {"1","/63","^+^32^42/12",
                 "^+^53+*361/12",
                 "/4+1/^12+2/^32+2/^52+2/^72+2^92"};

   int i;
   for(i = 0; i < 5; i++){
      puts(t[i]);
      printf( "\t= %.5f\n\n\n" , prefxSlv( &t[i] ) );
   }

   return 0;
}

float charToFloat(char c){
   char s[2];
   s[0] = c;   s[1] = '\0';

   return atof(s);
}

float operate( char op , float a , float b ){
   switch( op ){
      case '^':
         return pow( a , b );
      case '*':
         return a * b;
      case '/':
         return a / b;
      case '+':
         return a + b;
      case '-':
         return a - b;
      case '%':
         return (float)(((int)a) % ((int)b));
      default:
         return 0.0f;
   }
```

```
53 }
54
55 /*Justo de después de evaluar el caso
56 base, *expr apuntará a la posición
57 sig. al fin de la expresión*/
58 float prefxSlv( char **expr ){
59    /*Caso base, un dígito*/
60    if( isdigit( *( (*expr)++ ) ) ){
61       return charToFloat( *( *expr - 1 ) );
62    }
63
64    /*Recursión, operador y dos operandos*/
65    int op = *( (*expr) - 1 );
66    float A = prefxSlv( expr );
67    float B = prefxSlv( expr );
68    return operate( op , A , B );
69 }
70 /*
71 NO LEER SI SE CREE EN EL FUNCIONAMIENTO DEL ALGORITMO
72
73 Nota: Originalmente se propuso
74 'return operate( **expr, prefxSlv( expr ),
75                           prefxSlv( expr ) );'
76 en vez de las líneas 64 a 68 pero ya que en C
77 no hay garantía de que el orden de evaluación
78 de parámetros sea de izq. a der., esta evaluación
79 no funcionaría, en una expresión = +AB, podría
80 evaluar por ejemplo a 'operate( \0 , B , A )' que
81 no tendría sentido, por eso se cambió a calcular
82 antes los operandos; la demostración se basa en el
83 algoritmo original suponiendo que sí se evalúan los
84 parámetros de izq. a der. pero es igual de válida para
85 el algoritmo implementado.
86
87 Demostración de que funciona el algoritmo recursivo:
88 $,# representan operadores binarios cualquiera
89 A,B,C dígitos cualquiera
90
91 --Inducción sobre el número de operadores (n)
92
93 +Casos bases:
94 n = 0:
95    *expr -> {'A'}
96    prefxSlv( &expr ) = A, *expr -> \0
97 n = 1:
98    *expr -> {'$','A','B'}
99    prefxSlv( &expr ) = operate( '$' , prefxSlv( &expr ),
100                                       prefxSlv( &expr ) )
101       ya que *expr -> {'A','B'} antes del 1er prefxSlv
102       1er prefxSlv( &expr ) = A, *expr -> {'B'},
103       2do prefxSlv( &expr ) = B, *expr -> \0
104    prefxSlv( &expr ) = operate( '$' , A , B ) = $AB,
105       *expr -> \0
106 n = 2:
107    *expr -> {'$','A','#','B','C'}
108    prefxSlv( &expr ) = operate( '$' , prefxSlv( &expr ),
```

```
                                        prefxSlv( &expr ) )
      ya que *expr -> {'A','#','B','C'} antes del 1er prefxSlv
         1er prefxSlv( &expr ) = A, *expr -> {'#','B','C'}
         2do prefxSlv( &expr ) = #BC, *expr -> \0 (ver n = 1)
   prefxSlv( &expr ) = operate( '$' , A , #BC ) = $A#BC

   *expr -> {'$','#','A','B','C'}
   prefxSlv( &expr ) = operate( '$' , prefxSlv( &expr ),
                                        prefxSlv( &expr ) )
      ya que *expr -> {'#','A','B','C'} antes del 1er prefxSlv
         1er prefxSlv( &expr ) = #AB, *expr -> {'C'} (ver n = 1)
         2do prefxSlv( &expr ) = C, *expr -> \0
   prefxSlv( &expr ) = operate( '$' , #AB , C ) = $#ABC

+Inducción (completa):
-Con cierto n > 2,
prefxSlv( &expr ) funciona, es correcto e igual
a la expresión que comienza en *expr si *expr
tiene de 0 a n operadores; además, deja *expr
apuntando a la posición sig. del final de la
expresión.

-Con n+1 operadores,
*expr debe ser de la forma $AB donde, A y B son
subexpresiones que tienen de 0 a n operadores
cada uno (si no fuera así, $AB tendría más de
n+1 operadores), con esto, *expr -> $AB
   prefxSlv( &expr ) = operate( '$' , prefxSlv( &expr ),
                                        prefxSlv( &expr ) )
      ya que *expr -> A antes del 1er prefxSlv
         1er prefxSlv( &expr ) = A, *expr -> B
         2do prefxSlv( &expr ) = B, *expr -> \0
   prefxSlv( &expr ) = operate( '$' , A , B ) = $AB,
      *expr = \0.
Así, con n+1 operadores, prefxSlv( &expr ) también es
igual a la expresión que comienza en *expr y deja *expr
apuntando a la posición sig. del final de expresión.

Por lo tanto,

+Para una expresión *expr de cualquier número de
operadores n >= 0, prefxSlv( expr ) funciona,
es correcto e igual a *expr y deja a *expr apuntando
a la posición sig. del final de *expr
*/
```

Capturas de Pantalla de Recursion/TareaE.c


1
= 1.00000
/63
= 2.00000
^+^32^42/12
= 5.00000
^+^53+*361/12
= 12.00000
/4+1/^12+2/^32+2/^52+2/^72+2^92
= 2.93968

Tarea (árboles)

a) Hacer un programa para crear y manipular (hacer todas las operaciones básicas) un árbol binario, haciendo una implementación dinámica.

Arboles/TareaA.c

```
/*VER Arboles/TareaC.c PARA CAPTURAS DE PANTALLAS*/
/*
data por default es int
debe ser especificado antes de un include
*/
#ifndef data
   #define data int
#endif

typedef struct node_d{
   data e;

   struct node_d *left;
   struct node_d *right;
   struct node_d *father;
}node_d;

typedef node_d* tree_d;

#ifndef REL_FUNCTION
   #define REL_FUNCTION 1234567

/*DEBE SER ESPECIFICADA*/
/*Devuelve valor < 0 si a debe ir a la
izq. de b, valor > 0 en caso contrario*/
int dataRelation( data a , data b );

#endif

/*Operaciones*/

/*Crea nodo*/
node_d* create_node_d( data );

/*Agrega nodo*/
node_d* insert_d( tree_d , data );

/*Borra arbol*/
void drop_node_d( tree_d );

data info_d( node_d* );

node_d* left_d( node_d* );
node_d* right_d( node_d* );
node_d* father_d( node_d* );
node_d* brother_d( node_d* );

int isLeft_d( node_d* );
int isRight_d( node_d* );
int isRoot_d( node_d* );

```

```
/*Implementación dinámica*/
node_d* create_node_d( data e ){
   node_d *n = (node_d*)malloc(sizeof(node_d));
      n->e = e;
      n->left = NULL;
      n->right = NULL;
      n->father = NULL;
   return n;
}

/*Para arbol t vacío, se debe usar t = insert_d( t , e )*/
node_d* insert_d( node_d *t , data e ){
   if( t == NULL ){
      t = create_node_d( e );
   }else{
      if( dataRelation( e , t->e ) < 0 ){
         t->left = insert_d( t->left , e );
         t->left->father = t;
      }else{
         t->right = insert_d( t->right , e );
         t->right->father = t;
      }
   }

   /*Sirve para asignar al hijo de t, por eso
   la recursión t->hijo = insert_d( t->hijo , e ),
   útil cuando t->hijo es NULL, trivial cuando no,
   ya que t->hijo permanece inalterado y se devuelve
   el mismo valor (t->hijo = t->hijo)*/
   return t;
}

#define insert_d( t , e ) do{ \
   if( t == NULL ) \
      t = insert_d( t , e ); \
   else \
      insert_d( t , e ); \
}while(0)

void drop_node_d( node_d *n ){
   if( n != NULL){
      drop_node_d( left_d( n ) );
      drop_node_d( right_d( n ) );
      free( n );
   }
}

data info_d( node_d *n ){ return n->e; }

node_d* left_d( node_d *n ){ return n->left; }
node_d* right_d( node_d *n ){ return n->right; }
node_d* father_d( node_d *n ){ return n->father; }
node_d* brother_d( node_d *n ){
   if( isRoot_d( n ) )   return NULL;
   if( n->father->left == n )
      return n->father->right;
```

```
108     return n->father->left;
109 }
110 
111 int isLeft_d( node_d *n ){
112     return !isRoot_d( n ) && n == n->father->left;
113 }
114 
115 int isRight_d( node_d *n ){
116     return !isRoot_d( n ) && n == n->father->right;
117 }
118 
119 int isRoot_d( node_d *n ){ return n->father == NULL; }
```

b) Hacer un programa para crear y manipular (hacer todas las operaciones básicas con) un árbol binario casi completo utilizando un arreglo. Si el árbol no es casi completo, debe hacerse y considerar nodos vacíos.

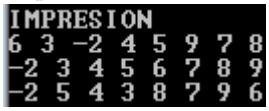
c) Hacer funciones para desplegar los árboles creados en los incisos anteriores.

Arboles/TareaC.c

```c
#include <stdio.h>
#include <stdlib.h>

#include "TareaA.c"
#include "TareaB.c"

int dataRelation( data a , data b ){
   if( a <= b)
      return -1;
   return 1;
}

void preOrderPrint_d( tree_d t );

void inOrderPrint_d( tree_d t );

void postOrderPrint_d( tree_d t );

int main(){
   int a[8] = {6,3,4,-2,9,7,8,5};
   int i;

   tree_d t1 = NULL;
   for(i = 0; i < sizeof( a ) / sizeof( a[0] ); i++)
      insert_d( t1 , a[i] );

   puts("IMPRESION");
   preOrderPrint_d( t1 );puts("");
   inOrderPrint_d( t1 );puts("");
   postOrderPrint_d( t1 );

   return 0;
}

void preOrderPrint_d( tree_d t ){
   if( t != NULL ){
      printf( "%d " , t->e );
      preOrderPrint_d( t->left );
      preOrderPrint_d( t->right );
   }
}

void inOrderPrint_d( tree_d t ){
   if( t != NULL ){
      inOrderPrint_d( t->left );
      printf( "%d " , t->e );
      inOrderPrint_d( t->right );
   }
}

void postOrderPrint_d( tree_d t ){
```

```
52    if( t != NULL ){
53       postOrderPrint_d( t->left );
54       postOrderPrint_d( t->right );
55       printf( "%d " , t->e );
56    }
57 }
```

Capturas de Pantalla de Arboles/TareaC.c



d) Escribir funciones que permitan determinar:
   a. La cantidad de nodos de un árbol binario.
   b. La suma del contenido de todos los nodos en un árbol binario.
   c. La profundidad de un árbol binario.
   d. El número de ocurrencias de un elemento en un árbol binario.

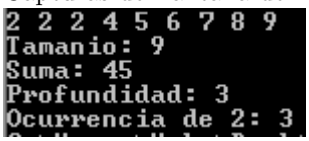
Arboles/TareaD.c

```c
#include <stdio.h>
#include <stdlib.h>

#include "TareaA.c"

int dataRelation( data a , data b ){
   if( a <= b)
      return -1;
   return 1;
}

void preOrderPrint_d( tree_d t );

void inOrderPrint_d( tree_d t );

void postOrderPrint_d( tree_d t );

/*a. # de nodos*/
int tree_size_d( node_d *t );

/*b. Suma de nodos*/
int tree_sum_d( node_d *t );

/*c. Profundidad de árbol*/
int tree_depth_d( node_d *t );

/*d. Ocurrencias en árbol*/
int times_in_tree_d( node_d *t , data value );

int main(){
   int a[9] = {6,2,2,9,4,7,2,5,8};
   int i;

   tree_d t1 = NULL;
   for(i = 0; i < sizeof( a ) / sizeof( a[0] ); i++)
      insert_d( t1 , a[i] );

   puts("InOrden");
   inOrderPrint_d( t1 );
   printf("\nTamanio: %d", tree_size_d( t1 ) );
   printf("\nSuma: %d", tree_sum_d( t1 ) );
   printf("\nProfundidad: %d", tree_depth_d( t1 ) );
   printf("\nOcurrencia de 2: %d",
          times_in_tree_d( t1 , 2 ) );

   return 0;
}

void preOrderPrint_d( tree_d t ){
   if( t != NULL ){
```

```
51        printf( "%d " , t->e );
52        preOrderPrint_d( t->left );
53        preOrderPrint_d( t->right );
54     }
55  }
56
57  void inOrderPrint_d( tree_d t ){
58     if( t != NULL ){
59        inOrderPrint_d( t->left );
60        printf( "%d " , t->e );
61        inOrderPrint_d( t->right );
62     }
63  }
64
65  void postOrderPrint_d( tree_d t ){
66     if( t != NULL ){
67        postOrderPrint_d( t->left );
68        postOrderPrint_d( t->right );
69        printf( "%d " , t->e );
70     }
71  }
72
73  /*a.*/
74  int tree_size_d( node_d *t ){
75     if( t == NULL )
76        return 0;
77     return tree_size_d( t->left ) + 1 +
78            tree_size_d( t->right );
79  }
80
81  /*b.*/
82  int tree_sum_d( node_d *t ){
83     if( t == NULL )
84        return 0;
85     return tree_sum_d( t->left ) + t->e +
86            tree_sum_d( t->right );
87  }
88
89  /*c.*/
90  int tree_depth_d( node_d *t ){
91     if( t == NULL )
92        return -1;
93     if( t->left == NULL && t->right == NULL )
94        return 0;
95
96     int depth_l = 0, depth_r = 0;
97
98     if( t->left != NULL )
99        depth_l = tree_depth_d( t->left );
100    if( t->right != NULL )
101       depth_r = tree_depth_d( t->right );
102
103    if( depth_l >= depth_r )
104       return 1 + depth_l;
105    return 1 + depth_r;
106 }
```

```
107
108 /*d.*/
109 int times_in_tree_d( node_d *t , data value ){
110    if( t == NULL )
111       return 0;
112    return times_in_tree_d( t->left , value ) +
113              (t->e == value) +
114           times_in_tree_d( t->right , value );
115 }
```

Capturas de Pantalla de Arboles/TareaD.c



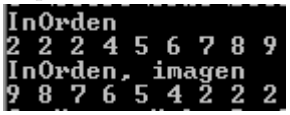
e) Escribir un programa que acepte un apuntador a un árbol binario y retorne un apuntador a un nuevo árbol binario que sea la imagen reflejo del primero, es decir, que todos los subárboles izquierdos sean ahora subárboles derechos y viceversa.

Arboles/TareaE.c

```c
#include <stdio.h>
#include <stdlib.h>

#include "TareaA.c"

int dataRelation( data a , data b ){
   if( a <= b)
      return -1;
   return 1;
}

void preOrderPrint_d( tree_d t );

void inOrderPrint_d( tree_d t );

void postOrderPrint_d( tree_d t );

/*Devuelve imagen reflejo de árbol*/
tree_d swap_d( tree_d );

int main(){
   int a[9] = {6,2,2,9,4,7,2,5,8};
   int i;

   tree_d t1 = NULL, t2 = NULL;
   for(i = 0; i < sizeof( a ) / sizeof( a[0] ); i++)
      insert_d( t1 , a[i] );

   t2 = swap_d( t1 );

   /*Si en verdad t2 es la imagen de t1,
   se imprimirá en forma descendente*/
   printf("InOrden\n");
   inOrderPrint_d( t1 );
   printf("\nInOrden, imagen\n");
   inOrderPrint_d( t2 );

   return 0;
}

void preOrderPrint_d( tree_d t ){
   if( t != NULL ){
      printf( "%d " , t->e );
      preOrderPrint_d( t->left );
      preOrderPrint_d( t->right );
   }
}

void inOrderPrint_d( tree_d t ){
   if( t != NULL ){
      inOrderPrint_d( t->left );
      printf( "%d " , t->e );
```

```
53          inOrderPrint_d( t->right );
54       }
55    }
56
57    void postOrderPrint_d( tree_d t ){
58       if( t != NULL ){
59          postOrderPrint_d( t->left );
60          postOrderPrint_d( t->right );
61          printf( "%d " , t->e );
62       }
63    }
64
65    tree_d swap_d( node_d *t ){
66       if( t == NULL )
67          return t;
68
69       if( t->left == NULL && t->right == NULL )
70          return create_node_d( t->e );
71
72       tree_d aux = create_node_d( t->e );
73       aux->left = swap_d( t->right );
74          if( aux->left != NULL )
75             aux->left->father = aux;
76       aux->right = swap_d( t->left );
77          if( aux->right != NULL )
78             aux->right->father = aux;
79
80       return aux;
81    }
```

Capturas de Pantalla de Arboles/TareaE.c



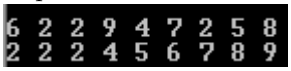
f) Escribir un programa que realice el ordenamiento de una serie de números, creando un árbol binario de búsqueda y después el recorrido correspondiente.

Arboles/TareaF.c

```c
#include <stdio.h>
#include <stdlib.h>

#include "TareaA.c"

int dataRelation( data a , data b ){
   if( a <= b)
      return -1;
   return 1;
}

void preOrderPrint_d( tree_d t );

void inOrderPrint_d( tree_d t );

void postOrderPrint_d( tree_d t );

/*Llena *int en inorden*/
void fill_inOrder_d( int** , tree_d );

/*Ordena b y lo pone en a */
void order_d( int *a , int *b , int n );

int main(){
   int a[9] = {6,2,2,9,4,7,2,5,8};
   int b[9];

   int i, j = sizeof( a ) / sizeof( a[0] );

   puts("");
   for(i = 0; i < j; i++)
      printf("%d ", a[i] );

   order_d( b , a , j );

   puts("");
   for(i = 0; i < j; i++)
      printf("%d ", b[i] );

   return 0;
}

void preOrderPrint_d( tree_d t ){
   if( t != NULL ){
      printf( "%d " , t->e );
      preOrderPrint_d( t->left );
      preOrderPrint_d( t->right );
   }
}

void inOrderPrint_d( tree_d t ){
   if( t != NULL ){
      inOrderPrint_d( t->left );
```

```
54          printf( "%d " , t->e );
55          inOrderPrint_d( t->right );
56       }
57    }
58
59    void postOrderPrint_d( tree_d t ){
60       if( t != NULL ){
61          postOrderPrint_d( t->left );
62          postOrderPrint_d( t->right );
63          printf( "%d " , t->e );
64       }
65    }
66
67    void fill_inOrder_d( int **A , node_d *t ){
68       if( t != NULL ){
69          fill_inOrder_d( A , t->left );
70          **A = t->e;
71          (*A)++;
72          fill_inOrder_d( A , t->right );
73       }
74    }
75
76    /*  B es un arreglo y n su tamaño*/
77    void order_d( int *A , int *B , int n ){
78       tree_d t = NULL;
79
80       int aux = n;
81       while( aux-- ){
82          insert_d( t , *(B++) );
83       }
84
85       fill_inOrder_d( &A , t );
86
87       return;
88    }
```

Capturas de Pantalla de Arboles/TareaF.c



g) Escribir un programa que construya un árbol de expresión a partir de una cadena posfija.

h) Hacer un programa que convierta una expresión infija a posfija mediante árboles.