```c
// * Filename: setADT.h

#define megisto_plithos 10          //μέγιστο πλήθος στοιχείων συνόλου

typedef enum {
    FALSE, TRUE
} boolean;

typedef boolean typos_synolou[megisto_plithos];
typedef int stoixeio_synolou;

void Dimiourgia(typos_synolou synolo);
void Katholiko(typos_synolou synolo);
void Eisagogi(stoixeio_synolou stoixeio, typos_synolou synolo);
void Diagrafi(stoixeio_synolou stoixeio, typos_synolou synolo);
boolean Melos(stoixeio_synolou stoixeio, typos_synolou synolo);
boolean KenoSynolo(typos_synolou synolo);
boolean IsaSynola(typos_synolou s1, typos_synolou s2);
boolean Yposynolo(typos_synolou s1, typos_synolou s2);
void EnosiSynolou(typos_synolou s1, typos_synolou s2, typos_synolou enosi);
void TomiSynolou(typos_synolou s1, typos_synolou s2, typos_synolou tomi);
void DiaforaSynolou(typos_synolou s1, typos_synolou s2, typos_synolou diafora);

// * Filename: setadt.c

#include <stdio.h>
#include <stdlib.h>
#include "SetADT.h"


void Dimiourgia(typos_synolou synolo)
/* Λειτουργία: Δημιουργεί ένα σύνολο χωρίς στοιχεία, δηλαδή το κενό σύνολο.
   Επιστρέφει: Το κενό σύνολο
*/
{
    stoixeio_synolou i;

    for (i = 0; i < megisto_plithos; i++)
        synolo[i] = FALSE;
}

void Katholiko(typos_synolou synolo)
/* Δέχεται:     Ένα σύνολο.
   Λειτουργία: Δημιουργεί ένα σύνολο με όλα τα στοιχεία παρόντα,
                έτσι όπως ορίστηκε στο τμήμα δηλώσεων του προγράμματος.
   Επιστρέφει: Το καθολικό σύνολο που δημιουργήθηκε
*/
{
    stoixeio_synolou i;

    for (i = 0; i < megisto_plithos; i++)
        synolo[i] = TRUE;
}

void Eisagogi(stoixeio_synolou stoixeio, typos_synolou synolo)
/* Δέχεται:    Ένα σύνολο και ένα στοιχείο.
   Λειτουργία: Εισάγει το στοιχείο στο σύνολο.
   Επιστρέφει: Το τροποποιημένο σύνολο
*/
{
    synolo[stoixeio] = TRUE;
}

```

```
64  void Diagrafi(stoixeio_synolou stoixeio, typos_synolou synolo)
65  /* Δέχεται:     Ένα σύνολο και ένα στοιχείο.
66     Λειτουργία:  Διαγράφει το στοιχείο από το σύνολο.
67     Επιστρέφει:  Το τροποποιημένο σύνολο
68  */
69  {
70      synolo[stoixeio] = FALSE;
71  }
72
73  boolean Melos(stoixeio_synolou stoixeio, typos_synolou synolo)
74  /* Δέχεται:    Ένα σύνολο και ένα στοιχείο.
75     Λειτουργία: Ελέγχει αν το στοιχείο είναι μέλος του συνόλου.
76     Επιστρέφει: Επιστρέφει TRUE αν το στοιχείο είναι μέλος του και FALSE διαφορετικά
77  */
78  {
79      return synolo[stoixeio];
80  }
81
82  boolean KenoSynolo(typos_synolou synolo)
83  /*
84  Δέχεται: Ένα σύνολο.
85   Λειτουργία: Ελέγχει αν το σύνολο είναι κενό.
86   Επιστρέφει: Επιστρέφει TRUE αν το σύνολο είναι κενό και FALSE διαφορετικά
87  */
88  {
89      stoixeio_synolou i;
90      boolean keno;
91
92      keno=TRUE;
93      i = 0;
94
95      while (i < megisto_plithos  && keno) {
96          if (Melos(i, synolo))
97              keno = FALSE;
98          else
99              i++;
100     }
101     return keno;
102 }
103
104 boolean IsaSynola(typos_synolou s1, typos_synolou s2)
105 /* Δέχεται:     Δύο σύνολα s1 και s2.
106    Λειτουργία: Ελέγχει αν τα δύο σύνολα είναι ίσα.
107    Επιστρέφει: Επιστρέφει TRUE αν τα δύο σύνολα έχουν τα ίδια στοιχεία και FALSE
       διαφορετικά
108 */
109 {
110     stoixeio_synolou i;
111     boolean isa;
112
113     isa = TRUE;
114     i=0;
115     while (i < megisto_plithos && isa)
116         if (Melos(i,s1) != Melos(i,s2))
117             isa = FALSE;
118         else
119             i++;
120     return isa;
121 }
122
123 boolean Yposynolo(typos_synolou s1, typos_synolou s2)
124 /* Δέχεται:     Δύο σύνολα s1 και s2.
125    Λειτουργία: Ελέγχει αν το σύνολο s1 είναι υποσύνολο του s2.
```

```
126       Επιστρέφει: Επιστρέφει true αν το σύνολο s1 είναι ένα υποσύνολο του s2,
127                   δηλαδή αν κάθε στοιχείο του s1 είναι και στοιχείο του s2
128   */
129   {
130       stoixeio_synolou i;
131       boolean yposyn;
132
133       yposyn = TRUE;
134       i=0;
135       while (i < megisto_plithos && yposyn)
136           if (Melos(i, s1) && !Melos(i, s2))
137               yposyn = FALSE;
138           else
139               i++;
140       return yposyn;
141   }
142
143   void EnosiSynolou(typos_synolou s1, typos_synolou s2, typos_synolou enosi)
144   /* Δέχεται:     Δύο σύνολα s1 και s2.
145      Λειτουργία: Δημιουργεί ένα νέο σύνολο με τα στοιχεία που ανήκουν ή στο s1 ή
146                   στο s2 ή και στα δύο σύνολα.
147      Επιστρέφει: Επιστρέφει το σύνολο enosi που προκύπτει από την ένωση των συνόλων s1 και
         s2
148   */
149   {
150       stoixeio_synolou i;
151
152       for (i = 0; i < megisto_plithos; i++)
153           enosi[i] = Melos(i, s1) || Melos(i, s2);
154   }
155
156   void TomiSynolou(typos_synolou s1, typos_synolou s2, typos_synolou tomi)
157   /* Δέχεται:     Δύο σύνολα s1 και s2.
158      Λειτουργία: Δημιουργεί ένα νέο σύνολο με τα στοιχεία που ανήκουν και στα δύο σύνολα s1
         και s2.
159      Επιστρέφει: Επιστρέφει το σύνολο tomi που προκύπτει από την τομή των συνόλων s1 και s2
160   */
161   {
162       stoixeio_synolou i;
163
164       for (i = 0; i < megisto_plithos; i++)
165           tomi[i] = Melos(i, s1) && Melos(i, s2);
166   }
167
168   void DiaforaSynolou(typos_synolou s1, typos_synolou s2, typos_synolou diafora)
169   /* Δέχεται: Δύο σύνολα s1 και s2.
170    Λειτουργία: Δημιουργεί ένα νέο σύνολο με τα στοιχεία που ανήκουν στο σύνολο s1 και δεν
       ανήκουν στο s2.
171    Επιστρέφει: Επιστρέφει το σύνολο diafora που προκύπτει από την διαφορά των συνόλων s1-s2.
172   */
173   {
174       stoixeio_synolou i;
175
176       for (i = 0; i < megisto_plithos; i++)
177           diafora[i] = Melos(i, s1) && (!Melos(i, s2));
178   }
179
180
```

```c
// FILENAME StackADT.h
                    /* ΥΛΟΠΟΙΗΣΗ ΣΤΟΙΒΑΣ ΜΕ ΠΙΝΑΚΑ *
              *ΤΑ ΣΤΟΙΧΕΙΑ ΤΗΣ ΣΤΟΙΒΑΣ ΕΙΝΑΙ ΤΥΠΟΥ int */

#define StackLimit 50   // το όριο μεγέθους της στοίβας, ενδεικτικά ίσο με 50


typedef int StackElementType;   // ο τύπος των στοιχείων  της στοίβας
                                //ενδεικτικά τύπος int
typedef struct  {
    int Top;
    StackElementType Element[StackLimit];
} StackType;

typedef enum {
    FALSE, TRUE
} boolean;

void CreateStack(StackType *Stack);
boolean EmptyStack(StackType Stack);
boolean FullStack(StackType Stack);
void Push(StackType *Stack, StackElementType Item);
void Pop(StackType *Stack, StackElementType *Item);

// FILENAME StackADT.c
                    /* ΥΛΟΠΟΙΗΣΗ ΣΤΟΙΒΑΣ ΜΕ ΠΙΝΑΚΑ *
              *ΤΑ ΣΤΟΙΧΕΙΑ ΤΗΣ ΣΤΟΙΒΑΣ ΕΙΝΑΙ ΤΥΠΟΥ int*/

#include <stdio.h>
#include "StackADT.h"

void CreateStack(StackType *Stack)
/* Λειτουργία: Δημιουργεί μια κενή στοίβα.
   Επιστρέφει: Κενή Στοίβα.*
*/
{
    Stack -> Top = -1;
   // (*Stack).Top = -1;
}

boolean EmptyStack(StackType Stack)
/* Δέχεται: Μια στοίβα Stack.
   Λειτουργία: Ελέγχει αν η στοίβα Stack είναι κενή.
   Επιστρέφει: True αν η Stack είναι κενή, False διαφορετικά
*/
{
    return (Stack.Top == -1);
}

boolean FullStack(StackType Stack)
/* Δέχεται: Μια στοίβα Stack.
   Λειτουργία: Ελέγχει αν η στοίβα Stack είναι γεμάτη.
   Επιστρέφει: True αν η Stack είναι γεμάτη, False διαφορετικά
*/
{
    return (Stack.Top == (StackLimit - 1));
}

void Push(StackType *Stack, StackElementType Item)
/* Δέχεται: Μια στοίβα Stack και ένα στοιχείο Item.
   Λειτουργία: Εισάγει το στοιχείο Item στην στοίβα Stack αν η Stack δεν είναι γεμάτη.
   Επιστρέφει: Την τροποποιημένη Stack.
   Έξοδος: Μήνυμα γεμάτης στοίβας, αν η στοίβα Stack είναι γεμάτη
```

```
64    */
65    {
66        if (!FullStack(*Stack)) {
67            Stack -> Top++;
68            Stack -> Element[Stack -> Top] = Item;
69        } else
70            printf("Full Stack...");
71    }
72
73    void Pop(StackType *Stack, StackElementType *Item)
74    /* Δέχεται: Μια στοίβα Stack.
75       Λειτουργία: Διαγράφει το στοιχείο Item από την κορυφή της Στοίβας αν η Στοίβα δεν είναι
       κενή.
76       Επιστρέφει: Το στοιχείο Item και την τροποποιημένη Stack.
77       Έξοδος:  Μήνυμα κενής στοίβας αν η Stack είναι κενή
78    */
79    {
80        if (!EmptyStack(*Stack)) {
81            *Item = Stack -> Element[Stack -> Top];
82            Stack -> Top--;
83        } else
84            printf("Empty Stack...");
85    }
86
87
88
```

```c
/*****************************************************************************
* Filename: QueueADT.h
*****************************************************************************/

/* Queue */
#define QueueLimit 20  //το όριο μεγέθους της ουράς

typedef int QueueElementType;   /* ο τύπος δεδομένων των στοιχείων της ουράς
                                   ενδεικτικά τύπος int */

typedef struct {
    int Front, Rear;
    QueueElementType Element[QueueLimit];
} QueueType;

typedef enum {FALSE, TRUE} boolean;

void CreateQ(QueueType *Queue);
boolean EmptyQ(QueueType Queue);
boolean FullQ(QueueType Queue);
void RemoveQ(QueueType *Queue, QueueElementType *Item);
void AddQ(QueueType *Queue, QueueElementType Item);

/*****************************************************************************
* Filename: QueueADT.c
*****************************************************************************/
#include <stdio.h>
#include "QueueADT.h"


void CreateQ(QueueType *Queue)
/*  Λειτουργία:  Δημιουργεί μια κενή ουρά.
    Επιστρέφει:  Κενή ουρά
*/
{
    Queue->Front = 0;
    Queue->Rear = 0;
}

boolean EmptyQ(QueueType Queue)
/* Δέχεται:    Μια ουρά.
   Λειτουργία: Ελέγχει αν η ουρά είναι κενή.
   Επιστρέφει: True αν η ουρά είναι κενή, False διαφορετικά
*/
{
    return (Queue.Front == Queue.Rear);
}

boolean FullQ(QueueType Queue)
/* Δέχεται:    Μια ουρά.
   Λειτουργία: Ελέγχει αν η ουρά είναι γεμάτη.
   Επιστρέφει: True αν η ουρά είναι γεμάτη, False διαφορετικά
*/
{
    return ((Queue.Front) == ((Queue.Rear +1) % QueueLimit));
}

void RemoveQ(QueueType *Queue, QueueElementType *Item)
/* Δέχεται:    Μια ουρά.
   Λειτουργία: Αφαιρεί το στοιχείο Item από την εμπρός άκρη της ουράς
                αν η ουρά δεν είναι κενή.
   Επιστρέφει: Το στοιχείο Item και την τροποποιημένη ουρά.
   Έξοδος:     Μήνυμα κενής ουρά αν η ουρά είναι κενή
```

```
64    */
65    {
66        if(!EmptyQ(*Queue))
67        {
68            *Item = Queue ->Element[Queue -> Front];
69            Queue ->Front  = (Queue ->Front + 1) % QueueLimit;
70        }
71        else
72            printf("Empty Queue\n");
73    }
74
75    void AddQ(QueueType *Queue, QueueElementType Item)
76    /* Δέχεται:    Μια ουρά Queue και ένα στοιχείο Item.
77       Λειτουργία: Προσθέτει το στοιχείο Item στην ουρά Queue
78                    αν η ουρά δεν είναι γεμάτη.
79       Επιστρέφει: Την τροποποιημένη ουρά.
80       Έξοδος:     Μήνυμα γεμάτης ουράς αν η ουρά είναι γεμάτη
81    */
82    {
83        if(!FullQ(*Queue))
84        {
85            Queue ->Element[Queue ->Rear] = Item;
86            Queue ->Rear = (Queue ->Rear + 1) % QueueLimit;
87        }
88        else
89            printf("Full Queue\n");
90    }
91
92
93
94
95
96
```

```c
// Filename L_ListADT.h

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

#define NumberOfNodes 50    /*μέγεθος της δεξαμενής κόμβων (λίστας), ενδεικτικά τέθηκε ίσο με 50*/

#define NilValue -1          // ειδική μεδενικη τιμη δείχνει το τέλος της Συνδ.λίστας

typedef int ListElementType; /*τύπος δεδομένων για τα στοιχεία της συνδεδεμένης λίστας,
                               ενδεικτικά επιλέχθηκε ο τύπος int */
typedef int ListPointer;

typedef struct {
    ListElementType Data;
    ListPointer  Next;
} NodeType;

typedef enum {
    FALSE, TRUE
} boolean;

void InitializeStoragePool(NodeType Node[], ListPointer *FreePtr);
void CreateList(ListPointer *List);
boolean EmptyList(ListPointer List);
boolean FullList(ListPointer FreePtr);
void GetNode(ListPointer *P, ListPointer *FreePtr, NodeType Node[]);
void ReleaseNode(NodeType Node[NumberOfNodes], ListPointer P, ListPointer *FreePtr);
void Insert(ListPointer *List, NodeType Node[],ListPointer *FreePtr, ListPointer PredPtr, ListElementType Item);
void Delete(ListPointer *List, NodeType Node[], ListPointer *FreePtr, ListPointer PredPtr);
void TraverseLinked(ListPointer List, NodeType Node[]);

// Filename L_ListADT.c
/*               ΥΛΟΠΟΙΗΣΗ ΣΥΝΔΕΔΕΜΈΝΗΣ ΛΊΣΤΑΣ ΜΕ ΠΙΝΑΚΑ
         ΤΑ ΣΤΟΙΧΕΙΑ ΤΗΣ ΣΥΝΔΕΔΕΜΕΝΗΣ ΛΙΣΤΑΣ ΕΙΝΑΙ ΤΥΠΟΥ int
*/

#include <stdio.h>
#include "L_ListADT.h"

void InitializeStoragePool(NodeType Node[], ListPointer *FreePtr)
/* Δέχεται:   Τον πίνακα Node και τον αριθμοδείκτη FreePtr που δείχνει στον
                πρώτο διαθέσιμο κόμβο.
  Λειτουργία: Αρχικοποιεί τον πίνακα Node ως συνδεδεμένη λίστα συνδέοντας μεταξύ
                τους διαδοχικές εγγραφές του πίνακα,
                και αρχικοποιεί τον αριθμοδείκτη FreePtr .
  Επιστρέφει: Τον τροποποιημένο πίνακα Node και τον
                αριθμοδείκτη FreePtr του πρώτου διαθέσιμου κόμβου
*/
{
   int i;

    for (i=0; i<NumberOfNodes-1;i++)
    {
        Node[i].Next=i+1;
        Node[i].Data=-1;    /* δεν είναι αναγκαίο η απόδοση αρχικής τιμής στο πεδίο των δεδομένων, βοηθάει στην εκτύπωση */
    }
    Node[NumberOfNodes-1].Next=NilValue;
    Node[NumberOfNodes-1].Data=-1;
```

```
60          *FreePtr=0;
61      }
62
63      void CreateList(ListPointer *List)
64      /* Λειτουργία: Δημιουργεί μια κενή συνδεδεμένη λίστα.
65         Επιστρέφει: Έναν (μηδενικό) αριθμοδείκτη που δείχνει σε κενή ΣΛ
66      */
67      {
68        *List=NilValue;
69      }
70
71      boolean EmptyList(ListPointer List)
72      /* Δέχεται:    Έναν αριθμοδείκτη List που δείχνει στο 1ο στοιχείο της ΣΛ.
73         Λειτουργία: Ελέγχει αν η συνδεδεμένη λίστα είναι κενή.
74         Επιστρέφει: True αν η συνδεδεμένη λίστα είναι κενή και false διαφορετικά
75      */
76      {
77        return (List==NilValue);
78      }
79
80      boolean FullList(ListPointer FreePtr)
81      /* Δέχεται:    Μια συνδεδεμένη λίστα.
82         Λειτουργία: Ελέγχει αν η συνδεδεμένη λίστα είναι γεμάτη.
83         Επιστρέφει: True αν η συνδεδεμένη λίστα είναι γεμάτη, false διαφορετικά
84      */
85      {
86        return (FreePtr == NilValue);
87      }
88
89      void GetNode(ListPointer *P, ListPointer *FreePtr, NodeType Node[])
90      /* Δέχεται:   Τον πίνακα Node με τα στοιχεία της ΣΛ και τους διαθέσιμους
91                    κόμβους και τον αριθμοδείκτη FreePtr..
92        Λειτουργία: Αποκτά τον 1ο "ελεύθερο" κόμβο
93        Επιστρέφει: Τον αριθμοδείκτη Ρπου δείχνει στο διαθέσιμο κόμβο και τον
94                    τροποποιημένο αριθμοδείκτη FreePtr που δεικτοδοτεί στο 1ο
95                    (νέο) διαθέσιμο κόμβο.
96      */
97      {
98        *P = *FreePtr;
99        if (!FullList(*FreePtr))
100         *FreePtr =Node[*FreePtr].Next;
101     }
102
103     void ReleaseNode(NodeType Node[], ListPointer P, ListPointer *FreePtr)
104     /* Δέχεται:    Τον πίνακα Node, που αναπαριστά τα στοιχεία της ΣΛ και τη
105                    δεξαμενή των διαθέσιμων κόμβων, και έναν αριθμοδείκτη Ρ.
106        Λειτουργία: Επιστρέφει στη δεξαμενή τον κόμβο στον οποίο δείχνει ο Ρ.
107        Επιστρέφει: Τον τροποποιημένο πίνακα Node και τον αριθμοδείκτη FreePtr
108     */
109     {
110       Node[P].Next =*FreePtr;
111       Node[P].Data = -1; /* Οχι αναγκαία εντολή, βοηθητική για να φαίνονται στην
112                             εκτύπωση οι διαγραμμένοι κόμβοι */
113       *FreePtr =P;
114     }
115
116     void Insert(ListPointer *List, NodeType Node[],ListPointer *FreePtr, ListPointer PredPtr,
        ListElementType Item)
117     /* Δέχεται:    Μια συνδεδεμένη λίστα, τον πίνακα Node, τον αριθμοδείκτη PredPtr και
118                    ένα στοιχείο Item.
119       Λειτουργία: Εισάγει στη συνδεδεμένη λίστα, αν δεν είναι γεμάτη, το στοιχείο
120                    Item μετά από τον κόμβο στον οποίο δείχνει ο αριθμοδείκτης PredPtr.
121       Επιστρέφει: Την τροποποιημένη συνδεδεμένη λίστα, τον τροποποιημένο πίνακα Node
```

```
122                     και τον αιρθμοδείκτη FreePtr.
123       Εξοδος:     Μήνυμα γεμάτης λίστας, αν η συνδεδεμένη λίστα είναι γεμάτη
124     */
125     {
126       ListPointer TempPtr;
127       GetNode(&TempPtr,FreePtr,Node);
128       if (!FullList(TempPtr)) {
129         if (PredPtr==NilValue)
130         {
131             Node[TempPtr].Data =Item;
132             Node[TempPtr].Next =*List;
133             *List =TempPtr;
134         }
135         else
136           {
137             Node[TempPtr].Data =Item;
138             Node[TempPtr].Next =Node[PredPtr].Next;
139             Node[PredPtr].Next =TempPtr;
140           }
141     }
142       else
143         printf("Full List ...\n");
144     }
145
146     void Delete(ListPointer *List, NodeType Node[], ListPointer *FreePtr, ListPointer PredPtr)
147     /* Δέχεται:   Μια συνδεδεμένη λίστα και τον αριθμοδείκτη PredPtr που δείχνει
148                     στον προηγούμενο κόμβο από αυτόν που θα διαγραφεί.
149       Λειτουργία: Διαγράφει από τη συνδεδεμένη λίστα, αν δεν είναι κενή,
150                     τον προηγούμενο κόμβο από αυτόν στον οποίο δείχνει ο PredPtr.
151       Επιστρέφει: Την τροποποιημένη λίστα και τον αριθμοδείκτη FreePtr.
152       Έξοδος:     Μήνυμα κενής λίστας, αν η συνδεδεμένη λίστα είναι κενή
153     */
154     {
155       ListPointer TempPtr ;
156
157       if (!EmptyList(*List)) {
158         if (PredPtr == NilValue)
159         {
160             TempPtr =*List;
161             *List =Node[TempPtr].Next;
162         }
163         else
164           {
165             TempPtr =Node[PredPtr].Next;
166             Node[PredPtr].Next =Node[TempPtr].Next;
167           }
168           ReleaseNode(Node,TempPtr,FreePtr);
169       }
170       else
171         printf("Empty List ...\n");
172     }
173
174     void TraverseLinked(ListPointer List, NodeType Node[])
175     /* Δέχεται:    Μια συνδεδεμένη λίστα.
176        Λειτουργία: Κάνει διάσχιση της συνδεδεμένης λίστας, αν δεν είναι κενή.
177        Έξοδος:     Εξαρτάται από την επεξεργασία
178     */
179     {
180       ListPointer CurrPtr;
181
182       if (!EmptyList(List))
183       {
184           CurrPtr =List;
```

```
185          while (CurrPtr != NilValue)
186          {
187              printf("(%d: %d->%d) ",CurrPtr,Node[CurrPtr].Data, Node[CurrPtr].Next);
188              CurrPtr=Node[CurrPtr].Next;
189           }
190           printf("\n");
191       }
192      else printf("Empty List ...\n");
193    }
194
```

```c
// Filename ListPADT.h

typedef int ListElementType;           /* ο τύπος των στοιχείων της συνδεδεμένης λίστας
                                        ενδεικτικά τύπου int */
typedef struct ListNode *ListPointer;   //ο τύπος των δεικτών για τους κόμβους
typedef struct ListNode
{
    ListElementType Data;
    ListPointer Next;
} ListNode;

typedef enum {
    FALSE, TRUE
} boolean;


void CreateList(ListPointer *List);
boolean EmptyList(ListPointer List);
void LinkedInsert(ListPointer *List, ListElementType Item, ListPointer PredPtr);
void LinkedDelete(ListPointer *List, ListPointer PredPtr);
void LinkedTraverse(ListPointer List);
void LinearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr, boolean
*Found);
void OrderedLinearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr,
boolean *Found);

// ListPADT.c
/*              ΥΛΟΠΟΙΗΣΗ ΣΥΝΔΕΔΕΜΕΝΗΣ ΛΙΣΤΑΣ ΔΥΝΑΜΙΚΑ - ΜΕ ΔΕΙΚΤΕΣ
                ΤΑ ΣΤΟΙΧΕΙΑ ΤΩΝ ΚΟΜΒΩΝ ΕΙΝΑΙ ΤΥΠΟΥ int */

#include <stdio.h>
#include <stdlib.h>
#include "ListPADT.h"

void CreateList(ListPointer *List)
/* Λειτουργία: Δημιουργεί μια κενή συνδεδεμένη λίστα.
  Επιστρέφει:  Τον μηδενικό δείκτη List
*/
{
    *List = NULL;
}

boolean EmptyList(ListPointer List)
/* Δέχεται:   Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο.
  Λειτουργία: Ελέγχει αν η συνδεδεμένη λίστα είναι κενή.
  Επιστρέφει: True αν η λίστα είναι κενή και false διαφορετικά
*/
{
    return (List==NULL);
}

void LinkedInsert(ListPointer *List, ListElementType Item, ListPointer PredPtr)
/* Δέχεται:    Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο,
                ένα στοιχείο δεδομένων Item και έναν δείκτη PredPtr.
   Λειτουργία: Εισάγει έναν κόμβο, που περιέχει το Item, στην συνδεδεμένη λίστα
                μετά από τον κόμβο που δεικτοδοτείται από τον PredPtr
                ή στην αρχή  της συνδεδεμένης λίστας,
                αν ο PredPtr είναι μηδενικός(NULL).
  Επιστρέφει:  Την τροποποιημένη συνδεδεμένη λίστα με τον πρώτο κόμβο της
                να δεικτοδοτείται από τον List.
*/
{
    ListPointer TempPtr;
```

```
62
63      TempPtr= (ListPointer)malloc(sizeof(struct ListNode));
64      TempPtr->Data = Item;
65       if (PredPtr==NULL) {
66           TempPtr->Next = *List;
67           *List = TempPtr;
68       }
69       else {
70           TempPtr->Next = PredPtr->Next;
71           PredPtr->Next = TempPtr;
72       }
73   }
74
75   void LinkedDelete(ListPointer *List, ListPointer PredPtr)
76   /* Δέχεται:    Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο της
77                    και έναν δείκτη PredPtr.
78      Λειτουργία: Διαγράφει από τη συνδεδεμένη λίστα τον κόμβο που έχει
79                   για προηγούμενό του αυτόν στον οποίο δείχνει ο PredPtr
80                   ή διαγράφει τον πρώτο κόμβο, αν ο PredPtr είναι μηδενικός,
81                   εκτός και αν η λίστα είναι κενή.
82      Επιστρέφει: Την τροποποιημένη συνδεδεμένη λίστα με τον πρώτο κόμβο
83                   να δεικτοδοτείται από τον List.
84      Έξοδος:     Ένα μήνυμα κενής λίστας αν η συνδεδεμένη λίστα ήταν κενή .
85   */
86   {
87       ListPointer TempPtr;
88
89       if (EmptyList(*List))
90           printf("EMPTY LIST\n");
91      else
92      {
93           if (PredPtr == NULL)
94           {
95                 TempPtr = *List;
96                 *List = TempPtr->Next;
97           }
98           else
99           {
100               TempPtr = PredPtr->Next;
101               PredPtr->Next = TempPtr->Next;
102          }
103          free(TempPtr);
104       }
105  }
106
107  void LinkedTraverse(ListPointer List)
108  /* Δέχεται:    Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο.
109     Λειτουργία: Διασχίζει τη συνδεδεμένη λίστα και
110                  επεξεργάζεται κάθε δεδομένο ακριβώς μια φορά.
111     Επιστρέφει: Εξαρτάται από το είδος της επεξεργασίας.
112  */
113  {
114       ListPointer CurrPtr;
115
116     if (EmptyList(List))
117          printf("EMPTY LIST\n");
118     else
119     {
120          CurrPtr = List;
121           while ( CurrPtr!=NULL )
122          {
123               printf("%p\t%d\t%p\n",CurrPtr,(*CurrPtr).Data, (*CurrPtr).Next);
124               CurrPtr = CurrPtr->Next;
```

```
125             }
126          }
127       }
128
129       void LinearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr, boolean
          *Found)
130       /* Δέχεται:   Μια συνδεδεμένη λίστα με τον List να δείχνει στον πρώτο κόμβο.
131         Λειτουργία: Εκτελεί μια γραμμική αναζήτηση στην μη ταξινομημένη συνδεδεμένη
132                     λίστα για έναν κόμβο που να περιέχει το στοιχείο Item.
133         Επιστρέφει: Αν η αναζήτηση είναι επιτυχής η Found είναι true, ο CurrPtr δείχνει
134                       στον κόμβο που περιέχει το Item και ο PredPtr στον προηγούμενό του
135                       ή είναι NULL αν δεν υπάρχει προηγούμενος.
136                       Αν η αναζήτηση δεν είναι επιτυχής η Found είναι false.
137       */
138       {
139          ListPointer CurrPtr;
140          boolean stop;
141
142          CurrPtr = List;
143           *PredPtr=NULL;
144          stop= FALSE;
145          while (!stop && CurrPtr!=NULL )
146           {
147                if (CurrPtr->Data==Item )
148                   stop = TRUE;
149                else
150                {
151                   *PredPtr = CurrPtr;
152                   CurrPtr = CurrPtr->Next;
153                }
154           }
155           *Found=stop;
156       }
157
158       void OrderedLinearSearch(ListPointer List, ListElementType Item, ListPointer *PredPtr,
          boolean *Found)
159       /* Δέχεται:    Ένα στοιχείο Item και μια ταξινομημένη συνδεδεμένη λίστα,
160                       που περιέχει στοιχεία δεδομένων σε αύξουσα διάταξη και στην οποία
161                       ο δείκτης List δείχνει στον πρώτο  κόμβο.
162          Λειτουργία: Εκτελεί γραμμική αναζήτηση της συνδεδεμένης ταξινομημένης λίστας
163                       για τον πρώτο κόμβο που περιέχει το στοιχείο Item ή για μια θέση
164                       για να εισάγει ένα νέο κόμβο που να περιέχει το στοιχείο Item.
165          Επιστρέφει: Αν η αναζήτηση είναι επιτυχής η Found είναι true,
166                       ο CurrPtr δείχνει στον κόμβο που περιέχει το Item και
167                       ο PredPtr στον προηγούμενό του ή είναι NULL αν δεν υπάρχει προηγούμενος.
168                       Αν η αναζήτηση δεν είναι επιτυχής η Found είναι false.
169       */
170       {
171          ListPointer CurrPtr;
172          boolean DoneSearching;
173
174          CurrPtr = List;
175          *PredPtr = NULL;
176          DoneSearching = FALSE;
177          *Found = FALSE;
178          while (!DoneSearching && CurrPtr!=NULL )
179           {
180                if (CurrPtr->Data>=Item )
181                {
182                   DoneSearching = TRUE;
183                   *Found = (CurrPtr->Data==Item);
184                }
185                else
```

```
186             {
187                *PredPtr = CurrPtr;
188                CurrPtr = CurrPtr->Next;
189             }
190         }
191     }
192
193
194
```

```c
// StackADT.h

typedef int StackElementType;           /*ο τύπος των στοιχείων της στοίβας
                                        ενδεικτικά τύπου int */
typedef struct StackNode *StackPointer;
typedef struct StackNode
{
    StackElementType Data;
    StackPointer Next;
} StackNode;

typedef enum {
    FALSE, TRUE
} boolean;

void CreateStack(StackPointer *Stack);
boolean EmptyStack(StackPointer Stack);
void Push(StackPointer *Stack, StackElementType Item);
void Pop(StackPointer *Stack, StackElementType *Item);

// Filename: StackADT.c

/* ΥΛΟΠΟΙΗΣΗ ΣΤΟΙΒΑΣ ΔΥΝΑΜΙΚΑ ΜΕ ΔΕΙΚΤΕΣ*/

#include <stdio.h>
#include <stdlib.h>
#include "StackADT.h"

void CreateStack(StackPointer *Stack)
/* Λειτουργία: Δημιουργεί μια κενή συνδεδεμένη στοίβα.
   Επιστρέφει: Μια κενή συνδεδεμένη στοίβα, Stack
*/
{
    *Stack = NULL;
}

boolean EmptyStack(StackPointer Stack)
/* Δέχεται:     Μια συνδεδεμένη στοίβα, Stack.
   Λειτουργία:  Ελέγχει αν η Stack είναι κενή.
   Επιστρέφει:  TRUE αν η στοίβα είναι κενή, FALSE διαφορετικά
*/
{
    return (Stack==NULL);
}

void Push(StackPointer *Stack, StackElementType Item)
/* Δέχεται:    Μια συνδεδεμένη στοίβα που η κορυφή της δεικτοδοτείται από τον
                δείκτη Stack και ένα στοιχείο Item.
   Λειτουργία: Εισάγει στην κορυφή της συνδεδεμένης στοίβας, το στοιχείο Item.
   Επιστρέφει: Την τροποποιημένη συνδεδεμένη στοίβα
*/
{
    StackPointer TempPtr;

    TempPtr= (StackPointer)malloc(sizeof(struct StackNode));
    TempPtr->Data = Item;
    TempPtr->Next = *Stack;
    *Stack = TempPtr;
}

void Pop(StackPointer *Stack, StackElementType *Item)
/* Δέχεται:    Μια συνδεδεμένη στοίβα που η κορυφή της δεικτοδοτείται από τον δείκτη Stack.
   Λειτουργία: Αφαιρεί από την κορυφή της συνδεδεμένης στοίβας,
```

```
64                      αν η στοίβα δεν είναι κενή, το στοιχείο Item.
65          Επιστρέφει: Την τροποποιημένη συνδεδεμένη στοίβα και το στοιχείο Item.
66          Έξοδος:     Μήνυμα κενής στοίβας, αν η συνδεδεμένη στοίβα είναι κενή
67      */
68      {
69          StackPointer TempPtr;
70
71          if (EmptyStack(*Stack)) {
72              printf("EMPTY Stack\n");
73          }
74         else
75         {
76              TempPtr = *Stack;
77              *Item=TempPtr->Data;
78              *Stack = TempPtr->Next;
79              free(TempPtr);
80          }
81      }
82
83
84
85
```

```c
// QueueADT.h

typedef int QueueElementType;           /*ο τύπος των στοιχείων της συνδεδεμένης ουράς
                                        ενδεικτικά τύπου int*/
typedef struct QueueNode *QueuePointer;

typedef struct QueueNode
{
    QueueElementType Data;
    QueuePointer Next;
} QueueNode;

typedef struct
{
    QueuePointer Front;
    QueuePointer Rear;
} QueueType;

typedef enum {
    FALSE, TRUE
} boolean;


void CreateQ(QueueType *Queue);
boolean EmptyQ(QueueType Queue);
void AddQ(QueueType *Queue, QueueElementType Item);
void RemoveQ(QueueType *Queue, QueueElementType *Item);

// QueuePADT.c
                        /*ΥΛΟΠΟΙΗΣΗ ΟΥΡΑΣ ΔΥΝΑΜΙΚΑ ΜΕ ΔΕΙΚΤΕΣ*/

#include <stdio.h>
#include <stdlib.h>
#include "QueueADT.h"

void CreateQ(QueueType *Queue)
/* Λειτουργία: Δημιουργεί μια κενή συνδεδεμένη ουρά.
   Επιστρέφει: Μια κενή συνδεδεμένη ουρά
*/
{
    (*Queue).Front = NULL;
    Queue->Rear = NULL;
}

boolean EmptyQ(QueueType Queue)
/* Δέχεται:    Μια συνδεδεμένη ουρά.
   Λειτουργία: Ελέγχει αν η συνδεδεμένη ουρά είναι κενή.
   Επιστρέφει: True αν η ουρά είναι κενή, false  διαφορετικά
*/
{
    return (Queue.Front==NULL);
}

void AddQ(QueueType *Queue, QueueElementType Item)
/* Δέχεται:    Μια συνδεδεμένη ουρά Queue και ένα  στοιχείο Item.
   Λειτουργία: Προσθέτει το στοιχείο Item στο τέλος της συνδεδεμένης ουράς Queue.
   Επιστρέφει: Την τροποποιημένη ουρά
*/
{
    QueuePointer TempPtr;

    TempPtr= (QueuePointer)malloc(sizeof(struct QueueNode));
    TempPtr->Data = Item;
```

```
64          TempPtr->Next = NULL;
65          if (Queue->Front==NULL)
66              Queue->Front=TempPtr;
67          else
68              Queue->Rear->Next = TempPtr;
69          Queue->Rear=TempPtr;
70      }
71
72      void RemoveQ(QueueType *Queue, QueueElementType *Item)
73      /* Δέχεται:    Μια συνδεδεμένη ουρά.
74         Λειτουργία: Αφαιρεί το στοιχείο Item από την  κορυφή της συνδεδεμένης ουράς,
75                      αν δεν είναι  κενή.
76         Επιστρέφει: Το στοιχείο Item και την τροποποιημένη συνδεδεμένη ουρά.
77         Έξοδος:     Μήνυμα κενής ουράς, αν η ουρά είναι  κενή
78      */
79      {
80          QueuePointer TempPtr;
81
82          if (EmptyQ(*Queue)) {
83              printf("EMPTY Queue\n");
84          }
85         else
86         {
87              TempPtr = Queue->Front;
88              *Item=TempPtr->Data;
89              Queue->Front = Queue->Front->Next;
90              free(TempPtr);
91              if (Queue->Front==NULL) Queue->Rear=NULL;
92          }
93      }
94
95
96
97
```

```c
// * Filename   BstADT.h

typedef int BinTreeElementType;        /*ο τύπος των στοιχείων του ΔΔΑ
                                        ενδεικτικά τύπου int */
typedef struct BinTreeNode *BinTreePointer;
typedef struct BinTreeNode {
    BinTreeElementType Data;
    BinTreePointer LChild, RChild;
} BinTreeNode;

typedef enum {
    FALSE, TRUE
} boolean;


void CreateBST(BinTreePointer *Root);
boolean EmptyBST(BinTreePointer Root);
void BSTInsert(BinTreePointer *Root, BinTreeElementType Item);
void BSTSearch(BinTreePointer Root, BinTreeElementType KeyValue, boolean *Found,
BinTreePointer *LocPtr);
void BSTSearch2(BinTreePointer Root, BinTreeElementType KeyValue, boolean *Found,
    BinTreePointer *LocPtr, BinTreePointer *Parent);
void BSTDelete(BinTreePointer *Root, BinTreeElementType KeyValue);
void InorderTraversal(BinTreePointer Root);

// Filename   BstADT.c
/*                ΥΛΟΠΟΙΗΣΗ ΔΔΑ ΔΥΝΑΜΙΚΑ ΜΕ ΔΕΙΚΤΕΣ
                   ΤΑ ΣΤΟΙΧΕΙΑ ΤΩΝ ΚΟΜΒΩΝ ΤΟΥ ΔΔΑ ΕΙΝΑΙ ΤΥΠΟΥ int*
*/

#include <stdio.h>
#include <stdlib.h>
#include "BstADT.h"

void CreateBST(BinTreePointer *Root)
/* Λειτουργία: Δημιουργεί ένα κενό ΔΔΑ.
   Επιστρέφει:  Τον μηδενικό δείκτη(NULL) Root
*/
{
    *Root = NULL;
}

boolean EmptyBST(BinTreePointer Root)
/* Δέχεται:   Ενα ΔΔα με το Root να δείχνει στη ρίζα του.
  Λειτουργία: Ελέγχει αν το ΔΔΑ είναι κενό.
  Επιστρέφει: TRUE αν το ΔΔΑ είναι κενό και FALSE διαφορετικά
*/
{   return (Root==NULL);
}

void BSTInsert(BinTreePointer *Root, BinTreeElementType Item)
/* Δέχεται:     Ένα ΔΔΑ με το δείκτη Root να δείχνει στη ρίζα του και ένα στοιχείο Item.
   Λειτουργία:  Εισάγει το στοιχείο Item στο ΔΔΑ.
   Επιστρέφει: Το τροποποιημένο ΔΔΑ με τον δείκτη Root να δείχνει στη ρίζα του
*/
{
    BinTreePointer LocPtr, Parent;
    boolean Found;

    LocPtr = *Root;
    Parent = NULL;
    Found = FALSE;
    while (!Found && LocPtr != NULL) {
```

```
63              Parent = LocPtr;
64              if (Item < LocPtr->Data)
65                  LocPtr = LocPtr ->LChild;
66              else if (Item > LocPtr ->Data)
67                  LocPtr = LocPtr ->RChild;
68              else
69                  Found = TRUE;
70          }
71          if (Found)
72              printf("TO STOIXEIO EINAI HDH STO DDA\n");
73          else {
74              LocPtr = (BinTreePointer)malloc(sizeof (struct BinTreeNode));
75              LocPtr ->Data = Item;
76              LocPtr ->LChild = NULL;
77              LocPtr ->RChild = NULL;
78              if (Parent == NULL)
79                  *Root = LocPtr;
80              else if (Item < Parent ->Data)
81                  Parent ->LChild = LocPtr;
82              else
83                  Parent ->RChild = LocPtr;
84          }
85      }
86
87      void BSTSearch(BinTreePointer Root, BinTreeElementType KeyValue, boolean *Found,
88                     BinTreePointer *LocPtr)
89      /* Δέχεται:    Ένα ΔΔΑ με το δείκτη Root να δείχνει στη ρίζα του και μια τιμή KeyValue.
90         Λειτουργία: Αναζητά στο ΔΔΑ έναν κόμβο με τιμή KeyValue στο πεδίο κλειδί του.
91         Επιστρέφει: Η Found έχει τιμή TRUE και ο δείκτης LocPtr δείχνει στον κόμβο που
92                     περιέχει την τιμή KeyValue, αν η αναζήτηση είναι επιτυχής.
93                     Διαφορετικά η Found έχει τιμή FALSE
94      */
95      {
96
97          (*LocPtr) = Root;
98          (*Found) = FALSE;
99          while (!(*Found) && (*LocPtr) != NULL)
100         {
101             if (KeyValue < (*LocPtr)->Data)
102                 (*LocPtr) = (*LocPtr)->LChild;
103             else
104                 if (KeyValue > (*LocPtr)->Data)
105                     (*LocPtr) = (*LocPtr)->RChild;
106                 else (*Found) = TRUE;
107         }
108     }
109
110     void BSTSearch2(BinTreePointer Root, BinTreeElementType KeyValue, boolean *Found,
111         BinTreePointer *LocPtr, BinTreePointer *Parent)
112     /* Δέχεται:    Ένα ΔΔΑ με το δείκτη Root να δείχνει στη ρίζα του και μια τιμή KeyValue.
113        Λειτουργία: Αναζητά στο ΔΔΑ έναν κόμβο με τιμή KeyValue στο πεδίο κλειδί του
114                     και τον πατέρα του κόμβου αυτού.
115        Επιστρέφει: Η Found έχει τιμή TRUE, ο δείκτης LocPtr δείχνει στον κόμβο που
116                     περιέχει την τιμή KeyValue και ο Parent δείχνει στον πατέρα
117                     αυτού του κόμβου, αν η αναζήτηση είναι επιτυχής.
118                     Διαφορετικά η Found έχει τιμή FALSE.
119     */
120     {
121         *LocPtr = Root;
122         *Parent=NULL;
123         *Found = FALSE;
124         while (!(*Found) && *LocPtr != NULL)
125         {
```

```
126             if (KeyValue < (*LocPtr)->Data) {
127                 *Parent=*LocPtr;
128                 *LocPtr = (*LocPtr)->LChild;
129             }
130             else
131                 if (KeyValue > (*LocPtr)->Data) {
132                     *Parent=*LocPtr;
133                     *LocPtr = (*LocPtr)->RChild;
134                 }
135                 else *Found = TRUE;
136         }
137
138     }
139
140     void BSTDelete(BinTreePointer *Root, BinTreeElementType KeyValue)
141     /* Δέχεται:  Ένα ΔΔΑ με το δείκτη Root να δείχνει στη ρίζα του και μια τιμή KeyValue.
142       Λειτουργία: Προσπαθεί να βρει έναν κόμβο στο ΔΔΑ που να περιέχει την τιμή
143                   KeyValue στο πεδίο κλειδί του τμήματος δεδομένων του και,
144                   αν τον βρει, τον διαγράφει από το ΔΔΑ.
145       Επιστρέφει: Το τροποποιημένο ΔΔΑ με τον δείκτη Root να δείχνει στη ρίζα του.
146     */
147     {
148
149        BinTreePointer
150        n,                    //δείχνει στον κόμβο που περιέχει την τιμή KeyValue *)
151        Parent,               // πατέρας του n ή του nNext
152        nNext,                // ενδοδιατεταγμένος επόμενος του n
153        SubTree;              // δείκτης προς υποδέντρο του n
154        boolean Found;        // TRUE AN ΤΟ ΣΤΟΙΧΕΙΟ KeyValue ΕΙΝΑΙ ΣΤΟΙΧΕΟ ΤΟΥ ΔΔΑ *)
155
156          BSTSearch2(*Root, KeyValue, &Found , &n, &Parent);
157          if (!Found)
158             printf("TO STOIXEIO DEN EINAI STO DDA\n");
159          else {
160               if (n->LChild != NULL && n->RChild != NULL)
161               {  // κόμβος προς διαγραφή με δύο παιδιά
162                      //Βρες τον ενδοδιατεταγμένο επόμενο και τον πατέρα του
163                      nNext = n->RChild;
164                      Parent = n;
165                      while (nNext->LChild !=NULL)  //* DIASXISH PROS TA ARISTERA *)
166                      {
167                           Parent = nNext;
168                           nNext = nNext->LChild;
169                      }
170                     /* Αντιγραφή των περιεχομένων του nNext στον n και
171                     αλλαγή του n ώστε να δείχνει στον επόμενο */
172                     n->Data = nNext->Data;
173                     n = nNext;
174               } //Συνεχίζουμε με την περίπτωση που ο κόμβος έχει το πολύ 1 παιδί
175               SubTree = n->LChild;
176               if (SubTree == NULL)
177                  SubTree = n->RChild;
178               if (Parent == NULL)                 //* ΘΑ DIAGRAFEI H RIZA *)
179                  *Root = SubTree;
180               else if (Parent->LChild == n)
181                       Parent->LChild = SubTree;
182                    else
183                        Parent->RChild = SubTree;
184               free(n);
185          }
186     }
187
188     void InorderTraversal(BinTreePointer Root)
```

```
189     /* Δέχεται:    Ένα δυαδικό δέντρο με το δείκτη Root να δείχνει στην ρίζα του.
190        Λειτουργία: Εκτελεί ενδοδιατεταγμένη διάσχιση του δυαδικού δέντρου και
191                     επεξεργάζεται κάθε κόμβο ακριβώς μια φορά.
192        Εμφανίζει: Το περιεχόμενο του κόμβου, και εξαρτάται από το είδος της επεξεργασίας
193     */
194     {
195         if (Root!=NULL) {
196             InorderTraversal(Root->LChild);
197             printf("%d ",Root->Data);
198             InorderTraversal(Root->RChild);
199         }
200     }
201
202
203
204
```

```c
/*
 * File:   BstRecADT.h
 */

typedef int BinTreeElementType;             /*ο τύπος των στοιχείων του ΔΔΑ
                                                ενδεικτικά τύπου int */
typedef struct BinTreeNode *BinTreePointer;
typedef struct BinTreeNode {
    BinTreeElementType Data;
    BinTreePointer LChild, RChild;
} BinTreeNode;

typedef enum {
    FALSE, TRUE
} boolean;


void CreateBST(BinTreePointer *Root);
boolean BSTEmpty(BinTreePointer Root);
void RecBSTInsert(BinTreePointer *Root, BinTreeElementType Item);
void RecBSTSearch(BinTreePointer Root, BinTreeElementType KeyValue, boolean *Found,
BinTreePointer *LocPtr);
void RecBSTDelete(BinTreePointer *Root, BinTreeElementType KeyValue);
void RecBSTInorder(BinTreePointer Root);
void RecBSTPreorder(BinTreePointer Root);
void RecBSTPostorder(BinTreePointer Root);

//File:   BstRecADT.c
/*                 ΥΛΟΠΟΙΗΣΗ ΔΔΑ ΔΥΝΑΜΙΚΑ ΜΕ ΔΕΙΚΤΕΣ
                    ΤΑ ΣΤΟΙΧΕΙΑ ΤΩΝ ΚΟΜΒΩΝ ΤΟΥ ΔΔΑ ΕΙΝΑΙ ΤΥΠΟΥ int*
*/

#include <stdio.h>
#include <stdlib.h>
#include "BstRecADT.h"

void CreateBST(BinTreePointer *Root)
/* Λειτουργία: Δημιουργεί ένα κενό ΔΔΑ.
   Επιστρέφει:  Τον μηδενικό δείκτη(NULL) Root
*/
{
    *Root = NULL;
}

boolean BSTEmpty(BinTreePointer Root)
/* Δέχεται:   Ενα ΔΔα με το Root να δείχνει στη ρίζα του.
  Λειτουργία: Ελέγχει αν το ΔΔΑ είναι κενό.
  Επιστρέφει: TRUE αν το ΔΔΑ είναι κενό και FALSE διαφορετικά
*/
{
    return (Root==NULL);
}

void RecBSTInsert(BinTreePointer *Root, BinTreeElementType Item)
/* Δέχεται:     Ένα ΔΔΑ με το δείκτη Root να δείχνει στη ρίζα του και ένα στοιχείο Item.
   Λειτουργία:  Εισάγει το στοιχείο Item στο ΔΔΑ.
   Επιστρέφει: Το τροποποιημένο ΔΔΑ με τον δείκτη Root να δείχνει στη ρίζα του
*/
{
    if (BSTEmpty(*Root)) {
        (*Root) = (BinTreePointer)malloc(sizeof (struct BinTreeNode));
        (*Root) ->Data = Item;
        (*Root) ->LChild = NULL;
```

```
63              (*Root) ->RChild = NULL;
64          }
65          else
66              if (Item < (*Root) ->Data)
67                  RecBSTInsert(&(*Root) ->LChild,Item);
68              else if (Item > (*Root) ->Data)
69                  RecBSTInsert(&(*Root) ->RChild,Item);
70              else
71                  printf("TO STOIXEIO EINAI HDH STO DDA\n");
72      }
73
74      void RecBSTSearch(BinTreePointer Root, BinTreeElementType KeyValue,
75                        boolean *Found, BinTreePointer *LocPtr)
76      /* Δέχεται:    Ένα ΔΔΑ με το δείκτη Root να δείχνει στη ρίζα του και μια τιμή KeyValue.
77         Λειτουργία: Αναζητά στο ΔΔΑ έναν κόμβο με τιμή KeyValue στο πεδίο κλειδί του.
78         Επιστρέφει: Η Found έχει τιμή TRUE και ο δείκτης LocPtr δείχνει στον κόμβο που
79                     περιέχει την τιμή KeyValue, αν η αναζήτηση είναι επιτυχής.
80                     Διαφορετικά η Found έχει τιμή FALSE
81      */
82      {
83          if (BSTEmpty(Root))
84              *Found=FALSE;
85          else
86              if (KeyValue < Root->Data)
87                  RecBSTSearch(Root->LChild, KeyValue, &(*Found), &(*LocPtr));
88              else
89                  if (KeyValue > Root->Data)
90                      RecBSTSearch(Root->RChild, KeyValue, &(*Found), &(*LocPtr));
91                  else
92                      {
93                          *Found = TRUE;
94                          *LocPtr=Root;
95                      }
96      }
97
98      void RecBSTDelete(BinTreePointer *Root, BinTreeElementType KeyValue)
99      /* Δέχεται:  Ένα ΔΔΑ με το δείκτη Root να δείχνει στη ρίζα του και μια τιμή KeyValue.
100        Λειτουργία: Προσπαθεί να βρει έναν κόμβο στο ΔΔΑ που να περιέχει την τιμή
101                    KeyValue στο πεδίο κλειδί του τμήματος δεδομένων του και,
102                    αν τον βρει, τον διαγράφει από το ΔΔΑ.
103       Επιστρέφει: Το τροποποιημένο ΔΔΑ με τον δείκτη Root να δείχνει στη ρίζα του.
104     */
105     {
106
107        BinTreePointer TempPtr;          //* true AN TO STOIXEIO KeyValue EINAI STOIXEIO TOY
          DDA *)
108
109         if (BSTEmpty(*Root))          //* ΑΔΕΙΟ ΔΕΝΔΡΟ ΤΟ KeyValue ΔΕ ΘΑ ΒΡΕΘΕΙ *)
110             printf("TO STOIXEIO DEN BRE8HKE STO DDA\n");
111          else
112              //* αναζήτησε αναδρομικά τον κόμβο που περιέχει την τιμή KeyValue και διάγραψέ
              τον
113               if (KeyValue < (*Root)->Data)
114                  RecBSTDelete(&((*Root)->LChild), KeyValue);       //* ΑΡΙΣΤΕΡΟ ΥΠΟΔΕΝΔΡΟ *
115               else
116                 if (KeyValue > (*Root)->Data)
117                       RecBSTDelete(&((*Root)->RChild), KeyValue);   //* ΔΕΞΙ  ΥΠΟΔΕΝΔΡΟ *
118                 else                                          //* ΤΟ KeyValue ΒΡΕΘΗΚΕ ΔΙΑΓΡΑΦΗ
                    ΤΟΥ ΚΟΜΒΟΥ *)
119                     if ((*Root)->LChild ==NULL)
120                      {
121                           TempPtr = *Root;
122                           *Root = (*Root)->RChild;      //* ΔΕΝ ΕΧΕΙ ΑΡΙΣΤΕΡΟ ΠΑΙΔΙ *)
```

```
123                         free(TempPtr);
124                     }
125                    else if ((*Root)->RChild == NULL)
126                       {
127                            TempPtr = *Root;
128                            *Root = (*Root)->LChild;      //* ΕΧΕΙ ΑΡΙΣΤΕΡΟ ΠΑΙΔΙ, ΑΛΛΑ ΟΧΙ
    ΔΕΞΙ *)
129                            free(TempPtr);
130                       }
131                       else                               //* ΕΧΕΙ 2 ΠΑΙΔΙΑ *)
132                       {
133                            //* ΕΥΡΕΣΗ ΤΟΥ INORDER ΑΠΟΓΟΝΟΥ ΤΟΥ *)
134                            TempPtr = (*Root)->RChild;
135                            while (TempPtr->LChild != NULL)
136                                  TempPtr = TempPtr->LChild;
137                            /* ΜΕΤΑΚΙΝΗΣΗ ΤΟΥ ΑΠΟΓΟΝΟΥ ΤΗς ΡΙΖΑΣ ΤΟΥ ΥΠΟΔΕΝΔΡΟΥ
138                            ΠΟΥ ΕΞΕΤΑΖΕΤΑΙ ΚΑΙ ΔΙΑΓΡΑΦΗ ΤΟΥ ΑΠΟΓΟΝΟΥ ΚΟΜΒΟΥ */
139                            (*Root)->Data = TempPtr->Data;
140                            RecBSTDelete(&((*Root)->RChild), (*Root)->Data);
141                       }
142    }
143
144    void RecBSTInorder(BinTreePointer Root)
145    /* Δέχεται:    Ένα δυαδικό δέντρο με το δείκτη Root να δείχνει στην ρίζα του.
146       Λειτουργία: Εκτελεί ενδοδιατεταγμένη διάσχιση του δυαδικού δέντρου και
147                    επεξεργάζεται κάθε κόμβο ακριβώς μια φορά.
148       Εμφανίζει: Το περιεχόμενο του κόμβου, και εξαρτάται από το είδος της επεξεργασίας
149    */
150    {
151        if (Root!=NULL) {
152        //    printf("L");
153            RecBSTInorder(Root->LChild);
154            printf("%d ",Root->Data);
155        //    printf("R");
156            RecBSTInorder(Root->RChild);
157        }
158       // printf("U");
159    }
160
161    void RecBSTPreorder(BinTreePointer Root)
162    /* Δέχεται:    Ένα δυαδικό δέντρο με το δείκτη Root να δείχνει στην ρίζα του.
163       Λειτουργία: Εκτελεί προδιατεταγμένη διάσχιση του δυαδικού δέντρου και
164                    επεξεργάζεται κάθε κόμβο ακριβώς μια φορά.
165       Εμφανίζει: Το περιεχόμενο του κόμβου, και εξαρτάται από το είδος της επεξεργασίας
166    */
167    {
168        if (Root!=NULL) {
169            printf("%d ",Root->Data);
170           // printf("L");
171            RecBSTPreorder(Root->LChild);
172          //  printf("R");
173            RecBSTPreorder(Root->RChild);
174        }
175       // printf("U");
176    }
177
178    void RecBSTPostorder(BinTreePointer Root)
179    /* Δέχεται:    Ένα δυαδικό δέντρο με το δείκτη Root να δείχνει στην ρίζα του.
180       Λειτουργία: Εκτελεί μεταδιατεταγμένη διάσχιση του δυαδικού δέντρου και
181                    επεξεργάζεται κάθε κόμβο ακριβώς μια φορά.
182       Εμφανίζει: Το περιεχόμενο του κόμβου, και εξαρτάται από το είδος της επεξεργασίας
183    */
184    {
```

```
185         if (Root!=NULL) {
186           //  printf("L");
187             RecBSTPostorder(Root->LChild);
188          //   printf("R");
189             RecBSTPostorder(Root->RChild);
190             printf("%d ",Root->Data);
191         }
192        // printf("U");
193     }
194
195
196
197
198
199
```

```c
//filename : HeapADT.h

#define  MaxElements 10          //το μέγιστο πλήθος των στοιχείων του σωρού

typedef int  HeapElementType;    //ο τύπος δεδομένων των στοιχείων του σωρού
typedef  struct {
     HeapElementType key;
    // int Data;                // οποισδήποτε τύπος για τα παρελκόμενα δεδομένα κάθε κόμβου
} HeapNode;


typedef struct {
        int Size;
        HeapNode Element[MaxElements+1];
} HeapType;

typedef enum {
    FALSE, TRUE
} boolean;

void CreateMaxHeap(HeapType *Heap);
boolean FullHeap(HeapType Heap);
void InsertMaxHeap(HeapType *Heap, HeapNode Item);
boolean EmptyHeap(HeapType Heap);
void DeleteMaxHeap(HeapType *Heap, HeapNode *Item);

//filename : HeapADT.c

#include <stdio.h>
#include "HeapADT.h"

void CreateMaxHeap(HeapType *Heap)
/* Λειτουργία: Δημιουργεί ένα κενό σωρό.
   Επιστρέφει: Ένα κενό σωρό
*/
{
  (*Heap).Size=0;
}

boolean EmptyHeap(HeapType Heap)
/* Δέχεται:    Ένα σωρό Heap.
   Λειτουργία: Ελέγχει αν ο σωρός είναι κενός.
   Επιστρέφει: TRUE αν ο σωρός είναι κενός, FALSE διαφορετικά
*/
{
  return (Heap.Size==0);
}

boolean FullHeap(HeapType Heap)
/* Δέχεται:    Ένα σωρό.
   Λειτουργία: Ελέγχει αν ο σωρός είναι γεμάτος.
   Επιστρέφει: TRUE αν ο σωρός είναι γεμάτος, FALSE διαφορετικά
*/
{
  return (Heap.Size==MaxElements);
}

void InsertMaxHeap(HeapType *Heap, HeapNode Item)
/*  Δέχεται:    Ένα σωρό Heap και ένα στοιχείο δεδομένου Item .
    Λειτουργία: Εισάγει το στοιχείο Item στο σωρό, αν ο σωρός δεν είναι γεμάτος.
    Επιστρέφει: Τον τροποποιημένο σωρό.
    Έξοδος:     Μήνυμα γεμάτου σωρού αν ο σωρός είναι γεμάτος
```

```
63    */
64    {
65       int hole;
66
67       if (!FullHeap(*Heap))
68       {
69              (*Heap).Size++;
70
71              hole=(*Heap).Size;
72              while (hole>1 && Item.key > Heap->Element[hole/2].key)
73              {
74                (*Heap).Element[hole]=(*Heap).Element[hole/2];
75                         hole=hole/2;
76              }
77              (*Heap).Element[hole]=Item;
78       }
79       else
80         printf("Full Heap...\n");
81    }
82
83    void DeleteMaxHeap(HeapType *Heap, HeapNode *Item)
84    /* Δέχεται:    Ένα σωρό Heap.
85       Λειτουργία: Ανακτά και διαγράφει το μεγαλύτερο στοιχείο του σωρού.
86       Επιστρέφει: Το μεγαλύτερο στοιχείο Item του σωρού και τον τροποποιημένο σωρό
87    */
88    {
89       int parent, child;
90       HeapNode last;
91       boolean done;
92
93       if (!EmptyHeap(*Heap))
94       {
95            done=FALSE;
96            *Item=(*Heap).Element[1];
97            last=(*Heap).Element[(*Heap).Size];
98            (*Heap).Size--;
99
100           parent=1; child=2;
101
102           while (child<=(*Heap).Size && !done)
103           {
104               if (child<(*Heap).Size)
105                   if ((*Heap).Element[child].key < (*Heap).Element[child+1].key)
106                       child++;
107               if (last.key >= (*Heap).Element[child].key)
108                   done=TRUE;
109               else
110               {
111                  (*Heap).Element[parent]=(*Heap).Element[child];
112                  parent=child;
113                  child=2*child;
114               }
115           }
116           (*Heap).Element[parent]=last;
117        }
118       else
119           printf("Empty heap...\n");
120    }
121
122
123
```

```c
//filename : HashList.h

#define HMax 5              /*     το μέγεθος του πίνακα HashTable
                                ενδεικτικά ίσο με 5 */
#define VMax 30             /*το μέγεθος της λίστας,
                                ενδεικτικά ίσο με 30 */
#define EndOfList -1        /* σημαία που σηματοδοτεί το τέλος της λίστας
                                και της κάθε υπολίστας συνωνύμων */

typedef int ListElementType;  /*τύπος δεδομένων για τα στοιχεία της λίστας
                                * ενδεικτικά τύπου int */
typedef int KeyType;

typedef struct {
    KeyType key;
    ListElementType Data;
    int Link;
} ListElm;

typedef struct  {
    int HashTable[HMax];   // πίνακας δεικτών προς τις υπολίστες συνωνύμων
    int Size;               // πλήθος εγγραφών της λίστας List
    int SubListPtr;         // Δδείκτης σε μια υπολίστα συνωνύμων
    int StackPtr;           // δείκτης προς την πρώτη ελεύθερη θέση της λίστας List
    ListElm List[VMax];
} HashListType;

typedef enum {
    FALSE, TRUE
} boolean;

void CreateHashList(HashListType *HList);
int HashKey(KeyType key);
boolean FullHashList(HashListType HList);
void SearchSynonymList(HashListType HList,KeyType KeyArg,int *Loc,int *Pred);
void SearchHashList(HashListType HList,KeyType KeyArg,int *Loc,int *Pred);
void AddRec(HashListType *HList,ListElm InRec);
void DeleteRec(HashListType *HList,KeyType DelKey);


//filename : HashList.c

#include <stdio.h>
#include "HashList.h"

int HashKey(KeyType key)
/* Δέχεται:    Την τιμή key ενός κλειδιού.
   Λειτουργία: Βρίσκει την τιμή κατακερματισμού HValue για το κλειδί Key.
   Επιστρέφει: Την τιμή κατακερματισμού HValue
*/
{
    /*σε περίπτωση που το KeyType δεν είναι ακέραιος
    θα πρέπει να μετατρέπεται κατάλληλα το κλειδί σε αριθμό*/
    return key%HMax;
}

void CreateHashList(HashListType *HList)
/* Λειτουργία: Δημιουργεί μια δομή HList.
   Επιστρέφει: Την δομή HList
*/
{
    int index;

```

```
64          HList->Size=0;           //ΔΗΜΙΟΥΡΓΕΙ ΜΙΑ ΚΕΝΗ ΛΙΣΤΑ
65          HList->StackPtr=0;       //ΔΕΙΚΤΗς ΣΤΗ ΚΟΡΥΦΗ ΤΗΣ ΣΤΟΙΒΑΣ ΤΩΝ ΕΛΕΥΘΕΡΩΝ ΘΕΣΕΩΝ
66
67          /*ΑΡΧΙΚΟΠΟΙΕΙ ΤΟΝ ΠΙΝΑΚΑ HashTable ΤΗΣ ΔΟΜΗΣ HList ΩΣΤΕ ΚΑΘΕ ΣΤΟΙΧΕΙΟΥ ΤΟΥ
68              ΝΑ ΕΧΕΙ ΤΗ ΤΙΜΗ EndOfList (-1)*/
69          for (index=0;index<HMax;index++)
70          {
71              HList->HashTable[index]=EndOfList;
72          }
73           //Δημιουργία της στοίβας των ελεύθερων θέσεων στη λίστα HList
74
75          for(index=0;index < VMax-1;index++)
76          {
77              HList->List[index].Link=index+1;
78          }
79          HList->List[index].Link=EndOfList;
80      }
81
82      boolean FullHashList(HashListType HList)
83      /* Δέχεται:    Μια δομή HList.
84         Λειτουργία: Ελέγχει αν η λίστα List της δομής HList είναι γεμάτη.
85         Επιστρέφει: TRUE αν η λίστα List είναι γεμάτη, FALSE διαφορετικά.
86      */
87      {
88          return(HList.Size==VMax);
89      }
90
91      void SearchSynonymList(HashListType HList,KeyType KeyArg,int *Loc,int *Pred)
92      /* Δέχεται:     Μια δομή HList και μια τιμή κλειδιού KeyArg.
93          Λειτουργία: Αναζητά μια εγγραφή με κλειδί KeyArg στην υπολίστα συνωνύμων.
94          Επιστρέφει: Τη θέση Loc της εγγραφής και τη θέση Pred της προηγούμενης
95                      εγγραφής στην υπολίστα
96      */
97      {
98          int Next;
99          Next=HList.SubListPtr;
100         *Loc=-1;
101         *Pred=-1;
102         while(Next!=EndOfList)
103         {
104             if (HList.List[Next].key==KeyArg)
105             {
106                 *Loc=Next;
107                 Next=EndOfList;
108             }
109             else
110             {
111                 *Pred=Next;
112                 Next=HList.List[Next].Link;
113             }
114         }
115     }
116     void SearchHashList(HashListType HList,KeyType KeyArg,int *Loc,int *Pred)
117     /* Δέχεται:     Μια δομή HList και μια τιμή κλειδιού KeyArg.
118         Λειτουργία: Αναζητά μια εγγραφή με κλειδί KeyArg στη δομή HList.
119         Επιστρέφει: Τη θέση Loc της εγγραφής και τη θέση Pred της
120                     προηγούμενης εγγραφής της υπολίστας στην οποία ανήκει.
121                     Αν δεν υπάρχει εγγραφή με κλειδί KeyArg τότε Loc=Pred=-1
122     */
123     {
124         int HVal;
125         HVal=HashKey(KeyArg);
126         if (HList.HashTable[HVal]==EndOfList)
```

```
127         {
128             *Pred=-1;
129             *Loc=-1;
130         }
131         else
132         {
133             HList.SubListPtr=HList.HashTable[HVal];
134             SearchSynonymList(HList,KeyArg,Loc,Pred);
135         }
136     }
137
138     void AddRec(HashListType *HList,ListElm InRec)
139     /* Δέχεται:    Μια δομή HList και μια εγγραφή InRec.
140        Λειτουργία: Εισάγει την εγγραφή InRec στη λίστα List, αν δεν είναι γεμάτη,
141                    και ενημερώνει τη δομή HList.
142        Επιστρέφει: Την τροποποιημένη δομή HList.
143        Έξοδος:     Μήνυμα γεμάτης λίστας, αν η List είναι γεμάτη, διαφορετικά,
144                    αν υπάρχει ήδη εγγραφή με το ίδιο κλειδί,
145                    εμφάνιση αντίστοιχου μηνύματος
146     */
147     {
148         int Loc, Pred, New, HVal;
149
150        // New=0;
151         if(!(FullHashList(*HList)))
152         {
153             Loc=-1;
154             Pred=-1;
155             SearchHashList(*HList,InRec.key,&Loc,&Pred);
156             if(Loc==-1)
157             {
158                 HList->Size=HList->Size +1;
159                 New=HList->StackPtr;
160                 HList->StackPtr=HList->List[New].Link;
161                 HList->List[New]=InRec;
162                 if (Pred==-1)
163                 {
164                     HVal=HashKey(InRec.key);
165                     HList->HashTable[HVal]=New;
166                     HList->List[New].Link=EndOfList;
167                 }
168                 else
169                 {
170                     HList->List[New].Link=HList->List[Pred].Link;
171                     HList->List[Pred].Link=New;
172                 }
173             }
174
175             else
176             {
177                 printf("YPARXEI HDH EGGRAFH ME TO IDIO KLEIDI \n");
178             }
179         }
180         else
181         {
182             printf("Full list...");
183         }
184     }
185     void DeleteRec(HashListType *HList,KeyType DelKey)
186     /* DEXETAI:    TH DOMH (HList) KAI To KLEIDI (DelKey) THS EGGRAFHS
187                    POY PROKEITAI NA DIAGRAFEI
188        LEITOYRGIA: DIAGRAFEI, THN EGGRAFH ME KLEIDI (DelKey) APO TH
189                    LISTA (List), AN YPARXEI ENHMERWNEI THN DOMH HList
```

```
190       EPISTREFEI: THN TROPOPOIHMENH DOMH (HList)
191       EJODOS:     "DEN YPARXEI EGGRAFH ME KLEIDI" MHNYMA
192    */
193    {
194        int Loc, Pred, HVal;
195
196        SearchHashList(*HList,DelKey,&Loc,&Pred);
197        if(Loc!=-1)
198        {
199            if(Pred!=-1)
200            {
201                HList->List[Pred].Link=HList->List[Loc].Link;
202            }
203            else
204            {
205                HVal=HashKey(DelKey);
206                HList->HashTable[HVal]=HList->List[Loc].Link;
207            }
208            HList->List[Loc].Link=HList->StackPtr;
209            HList->StackPtr=Loc;
210            HList->Size=HList->Size -1;
211        }
212        else
213        {
214            printf("DEN YPARXEI  EGGRAFH ME KLEIDI %d \n",DelKey);
215        }
216    }
217
218
```