5.7 Ο ΑΤΔ Μέγιστος Σωρός

Ο ΑΤΔ του μέγιστου σωρού (max heap) περιλαμβάνει τις ακόλουθες βασικές λειτουργίες:

- **δημιουργία** κενού σωρού
- **έλεγχος αν ο σωρός είναι άδειος**
- **έλεγχος αν ο σωρός είναι γεμάτος**
- **εισαγωγή στοιχείου** στο σωρό
- **διαγραφή του μεγαλύτερου στοιχείου** από το σωρό

Ένας τυπικός ορισμός για τον **Αφηρημένο Τύπο Δεδομένων του Μέγιστου Σωρού** είναι ο ακόλουθος:

ΑΤΔ ΜΕΓΙΣΤΟΣ ΣΩΡΟΣ

Συλλογή στοιχείων δεδομένων:

Ένα πλήρες δυαδικό δέντρο με n>0 στοιχεία οργανωμένα έτσι ώστε η τιμή σε κάθε κόμβο να είναι τουλάχιστο τόσο μεγάλη όσο εκείνη των παιδιών της.

Βασικές λειτουργίες:

Δημιουργία κενού σωρού - CreateMaxHeap:

Λειτουργία: Δημιουργεί ένα κενό σωρό.

Επιστρέφει: Ένα κενό σωρό.

Έλεγχος αν ο σωρός είναι γεμάτος - FullHeap:

Δέχεται: Ένα σωρό.

Λειτουργία: Ελέγχει αν ο σωρός είναι γεμάτος.

Επιστρέφει: TRUE, αν ο σωρός είναι γεμάτος, FALSE διαφορετικά.

Εισαγωγή στοιχείου στο σωρό - InsertMaxHeap:

Δέχεται: Ένα σωρό και ένα στοιχείο δεδομένων.

Λειτουργία: Εισάγει το στοιχείο στο σωρό, αν ο σωρός δεν είναι γεμάτος.

Επιστρέφει: Τον τροποποιημένο σωρό.

Έλεγχος αν ο σωρός είναι άδειος - EmptyHeap:

Δέχεται: Ένα σωρό.

Λειτουργία: Ελέγχει αν ο σωρός είναι άδειος.

Επιστρέφει: TRUE, αν ο σωρός είναι άδειος, FALSE διαφορετικά.

Διαγραφή του μεγαλύτερου στοιχείου από το σωρό - DeleteMaxHeap:

Δέχεται: Ένα σωρό.

Λειτουργία: Ανακτά και διαγράφει το μεγαλύτερο στοιχείο του σωρού.

Επιστρέφει: Το μεγαλύτερο στοιχείο του σωρού και τον τροποποιημένο σωρό.

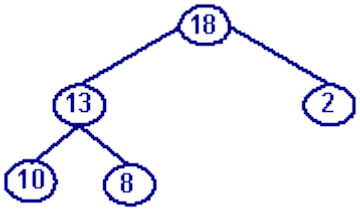
Μια τέτοια δομή μπορεί να υλοποιηθεί με μια εγγραφή (record), όπως φαίνεται παρακάτω:

```
#define MaxElements 10          /*μέγιστο πλήθος στοιχείων του σωρού, ενδεικτικά
                                10*/
typedef int HeapElementType;     /*ο τύπος των στοιχείων του σωρού, ενδεικτικά
                                 τύπου int*/
typedef struct {
        int key;
        HeapElementType Data;
} HeapNode;
typedef struct {
        int Size;
        HeapNode Element[MaxElements+1];
} HeapType;
```

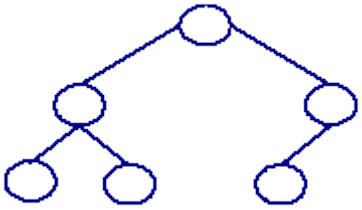
Εισαγωγή στοιχείου στο σωρό

Στη συνέχεια θα παρουσιάσουμε τη λειτουργία της εισαγωγής στοιχείου σε μέγιστο σωρό. Στο Σχήμα 5.7.1 φαίνεται ένας μέγιστος σωρός με 5 στοιχεία. Αν στο σωρό αυτό εισάγουμε ένα στοιχείο, τότε ο σωρός θα έχει τη δομή που φαίνεται στο Σχήμα 5.7.2, αφού ο μέγιστος σωρός είναι ένα πλήρες δυαδικό δέντρο.

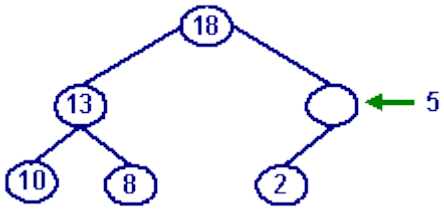
- Εισαγωγή στοιχείου με **τιμή κλειδιού 1**: το στοιχείο μπορεί να εισαχθεί στο μέγιστο σωρό ως αριστερό παιδί του κόμβου με τιμή κλειδιού 2, οπότε θα προκύψει ο σωρός με τη δομή που φαίνεται στο Σχήμα 5.7.1.
- Εισαγωγή στοιχείου με **τιμή κλειδιού 5**: στην περίπτωση αυτή το στοιχείο δεν μπορεί να εισαχθεί ως αριστερό παιδί του 2, γιατί τότε το πλήρες δυαδικό δέντρο που θα προκύψει δεν θα είναι ταυτόχρονα και μέγιστο δέντρο. Η επόμενη κίνησή μας είναι να μετακινήσουμε το 2 προς τα κάτω και συγκεκριμένα στη θέση του αριστερού παιδιού του (όπως φαίνεται στο Σχήμα 5.7.3) και να ελέγξουμε αν η εισαγωγή του 5 στην παλιά θέση του 2 οδηγεί ή όχι σε μέγιστο σωρό. Εφόσον, η τιμή κλειδιού του γονέα, η οποία είναι το 18, είναι τουλάχιστον τόσο μεγάλη όσο η τιμή του κλειδιού (5) του στοιχείου που εισάγεται, το στοιχείο μπορεί να εισαχθεί στη συγκεκριμένη θέση.
- Εισαγωγή στοιχείου με **τιμή κλειδιού 20**: αρχικά το 2 μετακινείται στη θέση του αριστερού παιδιού του, όπως φαίνεται στο Σχήμα 5.7.4. Στη συνέχεια ελέγχουμε αν το 20 μπορεί να εισαχθεί στην παλιά θέση του 2. Είναι προφανές ότι το 20 δεν μπορεί να εισαχθεί στη συγκεκριμένη θέση, αφού ο γονέας του κόμβου που βρίσκεται στη θέση αυτή έχει τιμή 18, η οποία είναι μικρότερη του 20. Συνεπώς, το 18 μετακινείται στη θέση του δεξιού παιδιού του και το νέο στοιχείο με τιμή κλειδιού 20 εισάγεται στη ρίζα του σωρού.



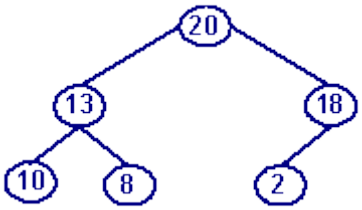
Σχήμα 5.7.1



Σχήμα 5.7.2



Σχήμα 5.7.3



Σχήμα 5.7.4

ΑΛΓΟΡΙΘΜΟΣ ΕΙΣΑΓΩΓΗΣ ΣΤΟΙΧΕΙΟΥ ΣΕ ΜΕΓΙΣΤΟ ΣΩΡΟ

/*Δέχεται: Μια δομή για το σωρό *MaxHeap,* η οποία περιλαμβάνει:

μια μεταβλητή *Size,* η οποία δείχνει κάθε φορά το πλήθος των στοιχείων του σωρού

έναν πίνακα *Heap*, στον οποίο αποθηκεύονται τα στοιχεία του σωρού

και ένα στοιχείο *Item* .

Λειτουργία: Εισάγει το στοιχείο *Item* στον σωρό, αν ο σωρός δεν είναι γεμάτος.

Επιστρέφει: Τον τροποποιημένο σωρό *MaxHeap*

Έξοδος: Μήνυμα γεμάτου σωρού αν ο σωρός *MaxHeap* είναι γεμάτος.*/

/*Καλείται η συνάρτηση *FullHeap*, η οποία ελέγχει αν ο σωρός είναι γεμάτος ή όχι και εκχωρεί την τιμή TRUE ή FALSE αντίστοιχα στη μεταβλητή *HeapError*/

Αν FullHeap(Heap) = FALSE **τότε**

/*Αν ο σωρός δεν είναι γεμάτος*/

Heap.Size ← Heap.Size + 1

/*Αύξησε το πεδίο *Size* της εγγραφής *Heap* κατά 1, δηλαδή αύξησε το πλήθος των στοιχείων του σωρού κατά 1*/

hole ← Heap.Size

/*Θέσε στη μεταβλητή *hole* την τιμή του πεδίου *Size* της εγγραφής *Heap*, δηλαδή το πλήθος των στοιχείων του σωρού μετά και την εισαγωγή του νέου στοιχείου*/

Όσο hole > 1 και Item.key > Heap.Element[hole / 2].key **επανάλαβε**

Heap.Element[hole] ← Heap.Element[hole / 2]

/*Θέσε στο στοιχείο που βρίσκεται στη θέση *hole* του πίνακα *Element* την τιμή του στοιχείου που βρίσκεται στη θέση *hole / 2* του πίνακα *Element*, δηλαδή ο πατέρας του νέου κόμβου μετακινείται προς τα κάτω και συγκεκριμένα παίρνει τη θέση του *hole*-ου κόμβου του σωρού ή αλλιώς του τελευταίου κόμβου που προστέθηκε στο σωρό*/

hole ← hole / 2

/*Θέσε στη μεταβλητή *iole* την τιμή *ihole/ 2*, έτσι ώστε στην επόμενη επανάληψη να ελεγχθεί αν το νέο στοιχείο μπορεί να πάρει τη θέση του κόμβου που ελευθερώθηκε με την μετακίνηση του κόμβου που έγινε με την προηγούμενη εντολή*/

Τέλος_επανάληψης

Heap.Element[holr] ← Item

/*Θέσε στο στοιχείο που βρίσκεται στη θέση *hole* του πίνακα *Element* το νέο στοιχείο, δηλαδή το νέο στοιχείο τοποθετείται στον θέση *i* που εντοπίστηκε μετά από μια σειρά ελέγχων και μετακινήσεων κόμβων που έγιναν στην παραπάνω επανάληψη*/

Αλλιώς

Γράψε 'Προσπαθείς να εισάγεις σε γεμάτο σωρό!'

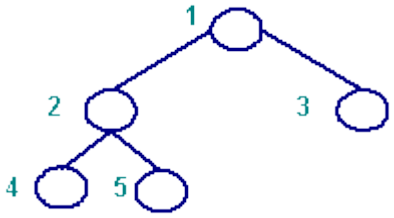
Τέλος _αν

Διαγραφή του μεγαλύτερου στοιχείου από το σωρό

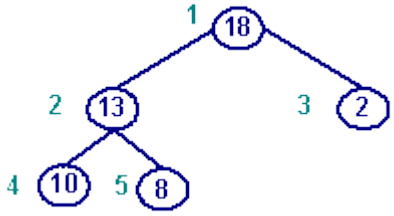
Στη συνέχεια θα παρουσιάσουμε τη λειτουργία της διαγραφής στοιχείου από μέγιστο σωρό. Στο Σχήμα 5.7.5 φαίνεται ένας μέγιστος σωρός με 5 στοιχεία. Αν από το σωρό αυτό διαγράψουμε το μεγαλύτερο στοιχείο, τότε ο σωρός θα έχει τη δομή που φαίνεται στο Σχήμα 5.7.7, αφού ο μέγιστος σωρός είναι ένα πλήρες δυαδικό δέντρο.

- Διαγραφή του μεγαλύτερου στοιχείου με **τιμή κλειδιού 18**: αν διαγράψουμε από τον μέγιστο σωρό που φαίνεται στο Σχήμα 5.7.6 το μεγαλύτερο στοιχείο με τιμή κλειδιού 18 τότε ο σωρός που θα προκύψει θα πρέπει να έχει τη δομή που φαίνεται στο Σχήμα 5.7.7, όπως ήδη αναφέραμε. Η πρώτη μας κίνηση λοιπόν είναι να διαγράψουμε τον κόμβο (με τιμή κλειδιού 8) που βρίσκεται στη θέση 5 του σωρού και να τον εισάγουμε στη ρίζα, η οποία είναι άδεια μετά τη διαγραφή του κόμβου με τιμή κλειδιού 18. Στη συνέχεια ελέγχουμε αν το πλήρες δυαδικό δέντρο που προκύπτει είναι ταυτόχρονα και μέγιστο δέντρο. Στο συγκεκριμένο παράδειγμα αυτό δεν συμβαίνει, αφού το στοιχείο της ρίζας με τιμή κλειδιού 8 δεν είναι μεγαλύτερο από τα παιδιά της που έχουν τιμές κλειδιού 13 και 2. Άρα, το στοιχείο με τιμή κλειδιού 13 που βρίσκεται στη θέση 2 μετακινείται προς τα πάνω και συγκεκριμένα στη ρίζα, ενώ το στοιχείο της ρίζας εισάγεται στον κενό πλέον κόμβο της 2ης θέσης. Η επόμενη κίνησή μας είναι να ελέγξουμε αν το πλήρες δυαδικό δέντρο που προκύπτει είναι ταυτόχρονα και μέγιστο δέντρο. Αυτό δεν συμβαίνει, αφού το στοιχείο στη θέση 2 με τιμή κλειδιού 8 δεν είναι μεγαλύτερο από το παιδί του που έχει τιμή κλειδιού 10. Άρα, το στοιχείο με τιμή κλειδιού 10 που βρίσκεται στη θέση 4 μετακινείται προς τα πάνω και

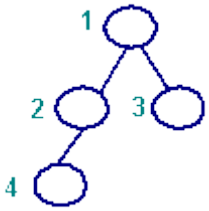
συγκεκριμένα στη θέση 2, ενώ το στοιχείο 8 της θέση 2 μετακινείται στον κόμβο της 4ης θέσης. Το πλήρες δυαδικό δένδρο που προκύπτει είναι και μέγιστο.



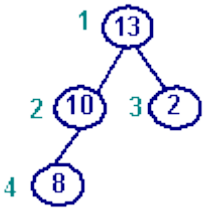
Σχήμα 5.7.5



Σχήμα 5.7.6



Σχήμα 5.7.7



Σχήμα 5.7.8

ΑΛΓΟΡΙΘΜΟΣ ΔΙΑΓΡΑΦΗΣ ΣΤΟΙΧΕΙΟΥ ΑΠΟ ΜΕΓΙΣΤΟ ΣΩΡΟ

/*Δέχεται: Ένα σωρό *Heap*.

Λειτουργία: Ανακτά και διαγράφει το μεγαλύτερο στοιχείο *Item* του σωρού.

Επιστρέφει: Το μεγαλύτερο στοιχείο *Item* του σωρού και τον τροποποιημένο σωρό.*/

Αν EmptyHeap(Heap)= FALSE **τότε**

/*Αν ο σωρός δεν είναι άδειος*/

dome ← FALSE

/*Θέσε στη μεταβλητή *done*, η οποία δείχνει αν η αναδιάταξη του δέντρου μετά τη διαγραφή του νέου στοιχείου έχει ολοκληρωθεί ή όχι, την τιμή FALSE*/

Item ← Heap.Element[1]

/*Θέσε στη μεταβλητή *Item* την τιμή του στοιχείου της 1ης θέσης του πίνακα *Element,* δηλαδή στην μεταβλητή *Item* θέτουμε την τιμή της ρίζας του σωρού*/

last ← Heap.Element[Heap.Size]

/*Θέσε στη μεταβλητή *last* την τιμή του στοιχείου της θέσης *Size* του πίνακα *Element,* δηλαδή στην μεταβλητή *Item* θέτουμε την τιμή του τελευταίου κόμβου του σωρού*/

Heap.Size ← Heap.Size - 1

/*Μείωσε τη μεταβλητή *Size* κατά 1, δηλαδή μείωσε το πλήθος των στοιχείων του σωρού κατά 1*/

parent ← 1

/*Θέσε στη μεταβλητή *parent* την τιμή 1*/

child ← 2

/*Θέσε στη μεταβλητή *child*, η οποία δείχνει τη θέση του κόμβου του σωρού που ελέγχεται κάθε φορά, την τιμή 2*/

Όσο (j<=Heap.Size) **και** (done = FALSE) **επανάλαβε**

/*Όσο η μεταβλητή *last* έχει τιμή μικρότερη ή ίση από τη μεταβλητή *Size* και η μεταβλητή *done* έχει τιμή FALSE, δηλαδή όσο η μεταβλητή *child* έχει τιμή μικρότερη ή ίση από το πλήθος των στοιχείων του σωρού (μετά τη διαγραφή) και η αναδιάταξη του σωρού δεν έχει ολοκληρωθεί*/

Αν child <Heap.Size **τότε**

/*Αν η μεταβλητή *child* έχει τιμή μικρότερη ή ίση από τη μεταβλητή *Size*, δηλαδή αν η μεταβλητή *child* έχει τιμή μικρότερη ή ίση από το πλήθος των στοιχείων του σωρού (μετά τη διαγραφή)*/

Αν Heap.Element[child].key < Heap.Element[child+1].key **τότε**

/*Αν η τιμή του πεδίου *key* του στοιχείου που βρίσκεται στη θέση *child* του πίνακα *Element* είναι μικρότερη από την τιμή του πεδίου *key* του στοιχείου που βρίσκεται στη θέση *child+1* του πίνακα *Element*, δηλαδή αν ο κόμβος *child+1*έχει μεγαλύτερη τιμή κλειδιού από τον κόμβο *child**/

child ← child + 1

/*Αύξησε τη μεταβλητή *child* κατά 1, έτσι ώστε να δείχνει στον κόμβο - από τα αδέρφια που εξετάζονται - με μεγαλύτερη τιμή κλειδιού*/

Τέλος_ αν

Τέλος_ αν

Αν last.key >= Heap.Element[child].key **τότε**

/*Αν η τιμή του πεδίου *key* της μεταβλητής *last* έχει τιμή μεγαλύτερη ή ίση από την τιμή του πεδίου *key* που που βρίσκεται στη θέση *child* του πίνακα *Element*, δηλαδή αν ο κόμβος που βρίσκεται στην τελευταία θέση του σωρού έχει τιμή κλειδιού μεγαλύτερη ή ίση από την τιμή κλειδιού του *child* ου κόμβου*/

done ← TRUE

/*Θέσε στη μεταβλητή *done* την τιμή TRUE, εφόσον ολοκληρώθηκε η αναδιάταξη του δέντρου*/

Αλλιώς

Heap.Element[parent] ← Heap.Element[child]

/*Θέσε στο στοιχείο που βρίσκεται στη θέση *i* του πίνακα *Element* την τιμή του στοιχείου που βρίσκεται στη θέση *j* του πίνακα *Element*, δηλαδή ο κόμβος *j* μετακινείται προς τα πάνω*/

parent ← child

/*Θέσε στη μεταβλητή *parent* την τιμή της μεταβλητής *child*, έτσι ώστε η

μεταβλητή *parent* να δείχνει στον κόμβο που μόλις εξετάστηκε*/

child ← 2*child

/*Θέσε στη μεταβλητή *child* την διπλάσια τιμή (2**jchild* από την τρέχουσα, έτσι ώστε η μεταβλητή *child* να δείχνει στο αριστερό παιδί του κόμβου *child* που μόλις ελέγχθηκε*/

Τέλος_ αν

Τέλος_επανάληψης

Heap.Element[parent] ← last

/*Θέσε στο στοιχείο που βρίσκεται στη θέση *parent* του πίνακα *Element* την τιμή του στοιχείου *last*, δηλαδή ο κόμβος που βρισκόταν στην τελευταία θέση του σωρού μετακινείται στη *parent* κόμβο έτσι ώστε μετά τη διαγραφή της ρίζας το δέντρο που προκύπτει να εξακολουθήσει να είναι σωρός*/

Αλλιώς

Γράψε 'Προσπαθείς να διαγράψεις από κενό σωρό!'

Τέλος_ αν

```
                    /*Πακέτο για τον ΑΤΔ Σωρός*/
// Filename HeapADT.h

interface
#define MaxElements 10       /*το μέγιστο πλήθος των στοιχείων του σωρού*/
typedef int HeapElementType;    /*ο τύπος δεδομένων των στοιχείων του σωρού*/

typedef struct {
        int key;
        HeapElementType Data;
} 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

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.*/
{
  int hole;
  if (!FullHeap(*Heap))
  {
     (*Heap).Size ++;
     hole = (*Heap).Size;
     done = FALSE;
     while (hole>1 && Item.key > Heap->Element[hole/2].key)
     {
         (*Heap).Element[i] = (*Heap).Element[i/2];
         hole = hole /2;
     }
     (*Heap).Element[hole] = Item;
  }
  else
   printf("Full Heap...\n");
}
```

```
void DeleteMaxHeap(HeapType *Heap, HeapNode *Item)
/*Δέχεται:        Ένα σωρό Heap.
 Λειτουργία:     Ανακτά και διαγράφει το μεγαλύτερο στοιχείο του σωρού.
 Επιστρέφει:     Το μεγαλύτερο στοιχείο Item του σωρού και τον τροποποιημένο
                 σωρό.*/
{
  int parent, child;
  HeapNode last;
  boolean notDone;
  if (!EmptyHeap(*Heap))
  {
    done = FALSE;
    (*Item) = (*Heap).Element[1];
    last = (*Heap).Element[(*Heap).Size];
    (*Heap).Size = (*Heap).Size-1;
    parent = 1; child = 2;
    while (child <= (*Heap).Size && !done)
    {
      if (child < (*Heap).Size)
        if ((*Heap).Element[child].key < (*Heap).Element[child +1].key)
                          child=child+1;
      if (last.key > = (*Heap).Element[child].key)
        done = TRUE;
      else
      {
        (*Heap).Element[parent] = (*Heap).Element[child];
        parent=child;
        child =2*child;
      }
    }
    (*Heap).Element[parent] = last;
  }
  else
    printf("Empty heap...\n");
}
```

Όλες οι παραπάνω συναρτήσεις περιλαμβάνονται στο HeapADT.c, που υλοποιεί τη διασύνδεση HeapADT.h για τον ΑΤΔ Μέγιστος Σωρός και μπορεί να χρησιμοποιηθεί σε ένα πρόγραμμα C με την εντολή

#include "HeapADT.h";

Παρακάτω φαίνεται ένα πρόγραμμα πελάτης, το TestHeap.c, που χρησιμοποιεί τη διασύνδεση HeapADT.h και την υλοποίηση HeapADT.c για την κατασκευή μέγιστου σωρού και εξετάζει αν η διασύνδεση και η υλοποίηση της που κατασκευάστηκε λειτουργεί σωστά. Το TestHeap.c επιτρέπει στο χρήστη να εκτελέσει όλες τις βασικές λειτουργίες που σχετίζονται

με τους σωρούς, δηλαδή να δημιουργήσει ένα κενό σωρό, να εισάγει στοιχεία σ' αυτόν, να διαγράφει στοιχεία από αυτόν και επιπλέον να εμφανίσει τα στοιχεία του σωρού.

```
// filename TestHeap.c

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

void menu(int *choice);
void PrintHeap(HeapType Heap);
int main()
{
  int choice ;
  char ch;
  HeapType AHeap;
  HeapNode AnItem;
  CreateMaxHeap(&AHeap);
  do
  {
        menu(&choice);
   switch(choice)
   {
          case 1: do
        {
              printf("Enter a number : ");
         scanf("%d", &AnItem.key);
         InsertMaxHeap(&AHeap, AnItem);
         printf("\nContinue Y/N: ");
         do
         {
               scanf("%c", &ch);
         } while (toupper(ch)!= 'N' && toupper(ch)!= 'Y');
        } while (toupper(ch)!='N');
        break;
    case 2: if (EmptyHeap(AHeap))
           printf("Empty Heap\n");
        else
           printf("Not an Empty Heap\n");
         break;
    case 3: if (EmptyHeap(AHeap))
          printf("Empty Heap\n");
        else
        {
          DeleteMaxHeap(&AHeap, &AnItem);
                printf("Dleted Item is %d \n", AnItem.key);
        }
         break;
    case 4: if (EmptyHeap(AHeap))
```

```
          printf("Empty Heap\n");
        else PrintHeap(AHeap);
        break;
   }
  } while (choice!=5);

 return 0;
}
void menu(int *choice)
{
  printf("         MENOY         \n");
  printf("-----------------------------------------------\n");
  printf("1. Εισαγωγή στοιχείου στο Heap\n");
  printf("2. Έλεγχος αν το Heap είναι άδειο\n");
  printf("3. Διαγραφή στοιχείου από το Heap\n");
  printf("4. Εμφάνιση του Heap (Βοηθητική)\n");
  printf("5. Τέλος\n");;
  printf("\n Choice: ");
  do
  {
        scanf("%d", choice);
  } while (*choice<1 && *choice>5);
}
void PrintHeap(HeapType Heap)
{
   int i,l,r,n;
   printf("%d \n", Heap.Size);
   printf("N  L  R\n");
   n=Heap.Size/2;
   for (i=1; i<=n; i++)
   {
     l=2*i;
     r=2*i+1;
     printf("%d, ",Heap.Element[i].key);
     if (l<=Heap.Size)
      printf("%d, ",Heap.Element[l].key);
     else printf(",  ");
     if (r<=Heap.Size)
      printf("%d \n", Heap.Element[r].key);
     else printf(" \n");
   }
}
```