4.8 Υλοποίηση ΑΤΔ Στοίβα και Ουρά με δείκτες

Στο τρίτο κεφάλαιο μελετήσαμε τις λίστες και είδαμε ότι η χρήση πινάκων ως τη βασική αποθηκευτική δομή δεν είναι πιστή υλοποίηση των λιστών, επειδή το σταθερό μέγεθος του πίνακα ορίζει σταθερό μέγεθος για τη λίστα. Το ίδιο είδαμε ότι ισχύει και για τις στοίβες και τις ουρές: η υλοποίησή τους με πίνακες θέτει ένα όριο στο μέγεθος της στοίβας ή της ουράς, ενώ θεωρητικά μπορούν να είναι απεριόριστες. Αφού, λοιπόν, είδαμε τις συνδεδεμένες λίστες, τώρα θα δούμε πώς υλοποιούνται οι στοίβες και οι ουρές ως συνδεδεμένες δομές.

Όπως είπαμε, μια στοίβα είναι μια λίστα που μπορεί να προσπελαστεί μόνο σε μια άκρη της, την *κορυφή*. Εφόσον, λοιπόν, σε μια συνδεδεμένη λίστα μόνο ο πρώτος κόμβος είναι άμεσα προσπελάσιμος (ο δείκτης List δείχνει πάντα στον πρώτο κόμβο), είναι πολύ λογικό να χρησιμοποιήσουμε μια συνδεδεμένη λίστα για να υλοποιήσουμε μια στοίβα (**linked stack**). Για μια τέτοια υλοποίηση στοίβας σε C μπορεί να κατασκευαστεί η παρακάτω διασύνδεση StackADT.h και η υλοποίηση της StackADT.c, όπου φαίνονται οι απαραίτητες δηλώσεις καθώς και οι διαδικασίες δημιουργίας κενής συνδεδεμένης στοίβας, ελέγχου αν μια συνδεδεμένη στοίβα είναι κενή, απώθησης και ώθησης στοιχείου στην συνδεδεμένη στοίβα. Οι συναρτήσεις δημιουργίας μιας κενής συνδεδεμένης στοίβας και ελέγχου αν μια συνδεδεμένη στοίβα είναι κενή είναι ίδιες με τις αντίστοιχες της συνδεδεμένης λίστας. Η λειτουργία της απώθησης για μια συνδεδεμένη στοίβα είναι στην ουσία η λειτουργία της διαγραφής του πρώτου στοιχείου μιας συνδεδεμένης λίστας, επομένως, η διαδικασία Pop είναι η απλοποιημένη διαδικασία Delete. Η διαδικασία της ώθησης σε μια στοίβα είναι παρόμοια με τη διαδικασία της εισαγωγής σε μια συνδεδεμένη λίστα, όταν η εισαγωγή γίνεται στην αρχή της.

/*Πακέτο για τον ΑΤΔ Συνδεδεμένη Στοίβα*/

```
//filename: 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
#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 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, αν η στοίβα δεν
                   είναι κενή, το στοιχείο Item.
  Επιστρέφει:      Την τροποποιημένη συνδεδεμένη στοίβα και το στοιχείο Item.
  Έξοδος:          Μήνυμα κενής στοίβας, αν η συνδεδεμένη στοίβα είναι κενή. */
 {
   StackPointer TempPtr;    /*προσωρινός δείκτης για τον κόμβο της κορυφής*/
    if (EmptyStack(*Stack))                    /*αν η στοίβα είναι άδεια*/
    {
        printf("Προσπαθείς να αφαιρέσεις στοιχείο από κενή στοίβα!");
    }
  else
  {
    TempPtr = *Stack;          /*θέσε στην προσωρινή μεταβλητή TempPtr την τιμή
                                 της Stack, δηλαδή τον κόμβο που βρίσκεται στην
                                 κορυφή της συνδεδεμένης στοίβας*/
    *Item = TempPtr-> Data;    /*θέσε στη μεταβλητή Item το στοιχείο που βρίσκεται
                                 στην κορυφή της στοίβας*/
    *Stack = TempPtr-> Next;   /*η μεταβλητή Stack δείχνει στον 2ο κόμβο από την
                                 κορυφή της συνδεδεμένης στοίβας*/
    free(TempPtr);             /*ο κόμβος που βρίσκεται στην κορυφή της
                                 συνδεδεμένης στοίβας και δεικτοδοτείται από τον
                                 δείκτη TempPtr επιστρέφεται στη δεξαμενή των
                                 διαθέσιμων κόμβων*/
  }
}
```

```
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;                                  /*δείκτης για τον κόμβο για  το
                                                         νέο στοιχείο*/
   TempPtr =
(StackPointer)malloc(sizeof(structStackNode));           /*πάρε ένα νέο κόμβο που
                                                         δεικτοδοτείται από τον δείκτη
                                                         TempPtr*/
   TempPtr->Data = Item;                                 /*θέσε στο πεδίο Data του κόμβου
                                                         που δεικτοδοτείται από τον δείκτη
                                                         TempPtr την τιμή της Item*/
   TempPtr->Next = *Stack;                               /*θέσε στο πεδίο Next του κόμβου
                                                         που δεικτοδοτείται από τον
                                                         TempPtr την τιμή της Stack,
                                                         δηλαδή ο νέος κόμβος δείχνει στον
                                                         κόμβο που βρισκόταν μέχρι τώρα
                                                         στην κορυφή της συνδεδεμένης
                                                         στοίβας*/
   *Stack = TempPtr;                                     /*θέσε στη μεταβλητή Stack την
                                                         τιμή της TempPtr έτσι ώστε η
                                                         Stack να δείχνει στο νέο κόμβο
                                                         που προστέθηκε στην κορυφή της
                                                         συνδεδεμένης στοίβας*/
}
```

Η διασύνδεση StackADT.h μπορεί να χρησιμοποιηθεί στο πρόγραμμα-πελάτης TestStack.c αντί για το StackADT.h, που είχαμε κατασκευάσει στο Κεφάλαιο 2, χωρίς καμία τροποποίηση στο πρόγραμμα. Αυτό σημαίνει ότι πρόκειται πράγματι για *αφηρημένο* τύπο δεδομένων, αφού, το πρόγραμμα που χρησιμοποιεί τη στοίβα ως δομή δεδομένων κατασκευάζεται ανεξάρτητα από το αν η στοίβα είναι υλοποιημένη με πίνακα ή με δείκτες.

Με παρόμοιο τρόπο μπορεί να υλοποιηθεί μια ουρά ως συνδεδεμένη λίστα. Η ουρά, όπως είπαμε, είναι μια λίστα, στην οποία αφαιρούνται στοιχεία μόνο από το ένα άκρο της, που λέγεται *εμπρός* ή *κεφάλι*, και εισάγονται στοιχεία μόνο στο άλλο άκρο της, το *πίσω* ή *ουρά*. Σε μια υλοποίηση ουράς ως συνδεδεμένη λίστα (**linked queue**), λοιπόν, μπορούμε να θεωρήσουμε το πρώτο στοιχείο της λίστας σαν το κεφάλι της ουράς, οπότε η διαδικασία της διαγραφής είναι ίδια με τη διαγραφή από στοίβα. Για την εισαγωγή, όμως, στοιχείου στη συνδεδεμένη ουρά θα πρέπει να γίνει διάσχιση της ουράς, ώστε να βρεθεί το τελευταίο στοιχείο της. Η διάσχιση μπορεί να αποφευχθεί αν διατηρούμε δύο δείκτες, έναν για το πρώτο στοιχείο, δηλαδή το κεφάλι, και έναν για το τελευταίο, δηλαδή την ουρά της συνδεδεμένης ουράς. Επομένως, η διασύνδεση QueueADT.h και το αρχείο υλοποίησης QueueADT.c που

μπορεί να κατασκευαστεί για την υλοποίηση της συνδεδεμένης ουράς είναι τα παρακάτω:

```
                    /*Πακέτο για τον ΑΤΔ Συνδεδεμένη Ουρά*/
// filename QueuePADT.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);

 // Filename QueuePADT.c

void CreateQ(QueueType *Queue)
/*Λειτουργία:               Δημιουργεί μια κενή συνδεδεμένη ουρά.
  Επιστρέφει:               Μια κενή συνδεδεμένη ουρά.*/
{
   Queue->Front = NULL;     /*θέσε στο πεδίο Front της εγγραφής Queue, στο οποίο
                            αποθηκεύεται η θέση της εμπρός άκρης της ουράς, την
                            τιμή NULL*/
   Queue->Rear = NULL;      /*θέσε στο πεδίο Rear της εγγραφής Queue, στο οποίο
                            αποθηκεύεται η θέση που της πίσω άκρης της ουράς, την
                            τιμή 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 RemoveQ(QueueType *Queue, QueueElementType *Item)
/*Δέχεται:          Μια συνδεδεμένη ουρά.
  Λειτουργία:       Αφαιρεί το στοιχείο Item από την  κορυφή της συνδεδεμένης
                    ουράς, αν δεν είναι  κενή.
  Επιστρέφει:       Το στοιχείο Item και την τροποποιημένη συνδεδεμένη ουρά.
  Έξοδος:           Μήνυμα κενής ουράς, αν η ουρά είναι  κενή.*/
 {
  QueuePointer
TempPtr;
 if (EmptyQ(*Queue))    /*αν η ουρά είναι κενή*/
 {
   printf("Προσπαθείς να διαγράψεις από κενή ουρά!\n");
 }
  else
  {
      TempPtr = Queue->Front;          /*θέσε στη μεταβλητή TempPtr την τιμή της
                                       Front, δηλαδή του κόμβου που βρίσκεται στην
                                       κορυφή της συνδεδεμένης ουράς*/
      *Item=TempPtr-> Data;            /*θέσε στη μεταβλητή Item την τιμή του
                                       πεδίου Data της εγγραφής Front, δηλαδή η
                                       Item παίρνει την τιμή του στοιχείου που
                                       βρίσκεται στην κορυφή της συνδεδεμένης
                                       ουράς*/
      Queue->Front = Queue->Front->Next;    /*θέσε στη μεταβλητή Front την τιμή
                                            του πεδίου Next της Front, δηλαδή η
                                            Front θα δείχνει στον κόμβο που
                                            βρισκόταν μέχρι τη διαγραφή του
                                            στοιχείου Item  πριν την κορυφή και
                                            τώρα βρίσκεται στην κορυφή της
                                            συνδεδεμένης ουράς*/
      free(TempPtr);                   /*επέστρεψε τον κόμβο που δεικτοδοτείται
                                       από τον TempPtr στη δεξαμενή των διαθέσιμων
                                       κόμβων*/
      if (Queue-> Front==NULL)
             Queue-> Rear=NULL;        /*αν η μεταβλητή Front έχει τιμή NULL,
                                       δηλαδή αν η ουρά είναι κενή, τότε θέσε και
                                       στη μεταβλητή Rear την τιμή 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));
                                        /*πάρε ένα νέο κόμβο που δεικτοδοτείται
                                        από τον TempPtr*/
    TempPtr->Data = Item;               /*θέσε στο πεδίο Data του TempPtr την τιμή
                                        της Item*/
    TempPtr->Next = NULL;               /*θέσε στο πεδίο Next του TempPtr την τιμή
                                        NULL, αφού το στοιχείο Item θα προστεθεί
                                        στο τέλος της συνδεδεμένης ουράς και δεν
                                        θα έχει επόμενο κόμβο*/
    if (Queue-> Front==NULL)            /*αν η συνδεδεμένη ουρά είναι κενή*/
         Queue-> Front=TempPtr;         /*θέσε στο πεδίο Front της Queue την τιμή
                                        του TempPtr, δηλαδή η μεταβλητή Front
                                        δείχνει στον κόμβο που προστέθηκε στη
                                        συνδεδεμένη ουρά και ο οποίος ως μοναδικός
                                        κόμβος της ουράς βρίσκεται στην κορυφή
                                        της*/
    else                                /*αν η συνδεδεμένη ουρά δεν είναι κενή*/
       Queue->Rear->Next = TempPtr;     /*ενημέρωσε το πεδίο Next της Rear  έτσι
                                        ώστε ο κόμβος που βρισκόταν μέχρι τώρα στο
                                        τέλος της συνδεδεμένης ουράς να δείχνει
                                        στο νέο κόμβο TempPtr που προστέθηκε στο
                                        τέλος της*/
    Queue-> Rear=TempPtr;               /*θέσε στο πεδίο Rear της Queue την τιμή
                                        του TempPtr, δηλαδή η μεταβλητή Rear
                                        δείχνει στον κόμβο που προστέθηκε στο
                                        τέλος της συνδεδεμένης ουράς*/
}
```