ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ

ΛΙΣΤΕΣ
4.8 Υλοποίηση ΑΤΔ Στοίβα και Ουρά με δείκτες

Υλοποίηση στοίβας και ουράς 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.

Υλοποίηση στοίβας με δείκτες

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**).

Υλοποίηση στοίβας με δείκτες

Για μια τέτοια υλοποίηση στοίβας σε C μπορεί να κατασκευαστεί η παρακάτω μονάδα 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 διαδικασίες:

- 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

```
// 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;
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Στοίβα

```
void CreateStack(StackPointer *Stack);

boolean EmptyStack(StackPointer Stack);

void Push(StackPointer *Stack, StackElementType Item);

void Pop(StackPointer *Stack, StackElementType *Item);
```

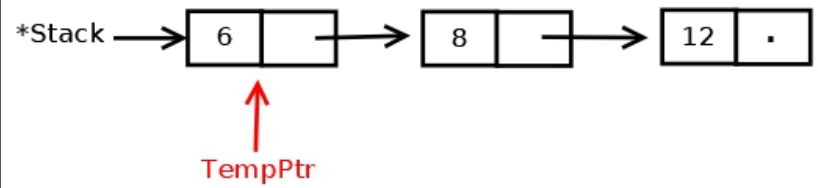
Πακέτο για τον ΑΤΔ Συνδεδεμένη Στοίβα

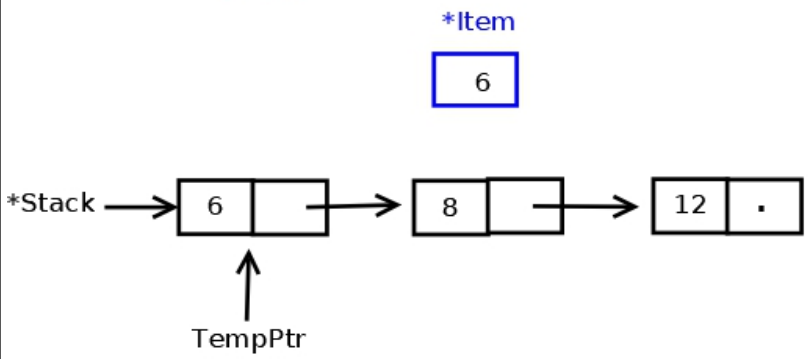
```
// StackADT.c

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);
}
```

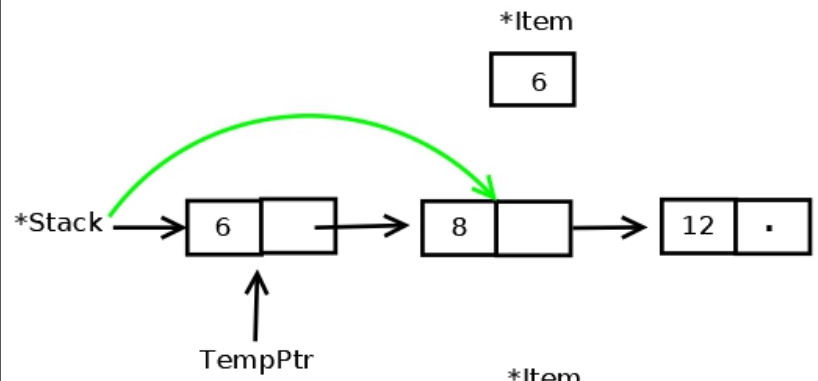
Πακέτο για τον ΑΤΔ Συνδεδεμένη Στοίβα

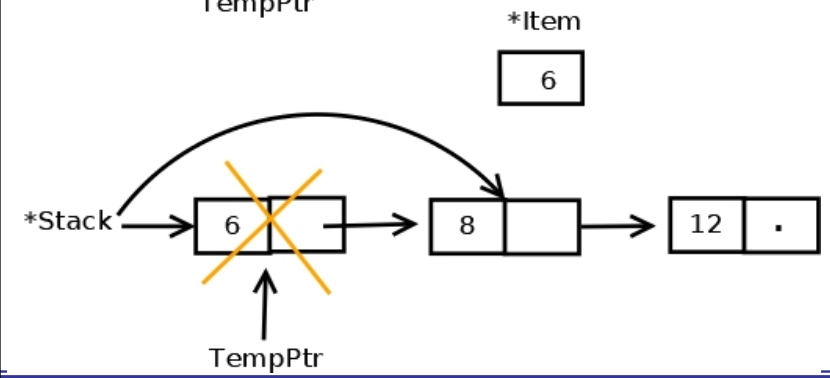
```
void Pop(StackPointer *Stack, StackElementType *Item)
/*Δέχεται:        Μια συνδεδεμένη στοίβα που η κορυφή της δεικτοδοτείται από
                  τον δείκτη Stack.
  Λειτουργία:     Αφαιρεί από την κορυφή της συνδεδεμένης στοίβας, αν η στοίβα
                  δεν είναι κενή, το στοιχείο Item.
  Επιστρέφει:     Την τροποποιημένη συνδεδεμένη στοίβα και το στοιχείο Item.
  Έξοδος:         Μήνυμα κενής στοίβας, αν η συνδεδεμένη στοίβα είναι κενή.*/
{
    StackPointer TempPtr;

    if (EmptyStack(*Stack))
            printf("EMPTY Stack\n");
   else
   {
        TempPtr = *Stack;
        *Item = TempPtr->Data;
        *Stack = TempPtr->Next;
        free(TempPtr);
    }
}
```

void Pop(StackPointer *Stack, StackElementType *Item)







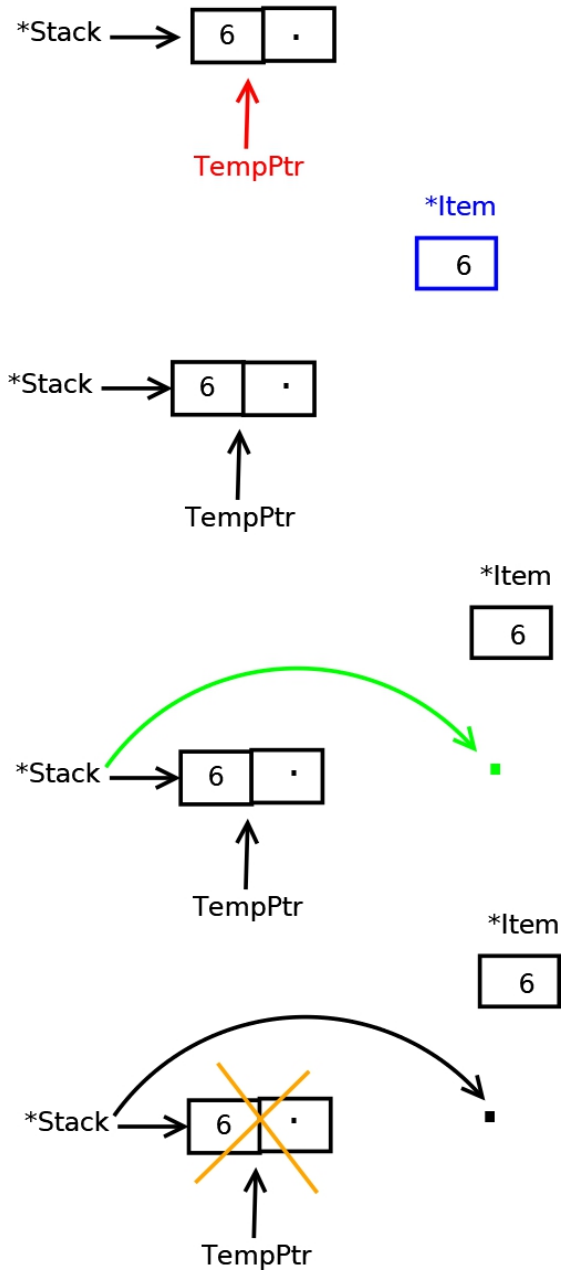



```
StackPointer TempPtr;

if (EmptyStack(*Stack))
        printf("EMPTY Stack\n");
else
{
    TempPtr = *Stack;
    *Item = TempPtr->Data;
    *Stack = TempPtr->Next;
    free(TempPtr);
}
```

κλήση από main

```
{
  StackPointer AStack;
  StackElementType AnItem;
....
  Pop(&AStack, &AnItem);
....
}
```

void Pop(StackPointer *Stack, StackElementType *Item)




```
StackPointer TempPtr;

if (EmptyStack(*Stack))
        printf("EMPTY Stack\n");
else
{
    TempPtr = *Stack;
    *Item = TempPtr->Data;
    *Stack = TempPtr->Next;
    free(TempPtr);
}
```

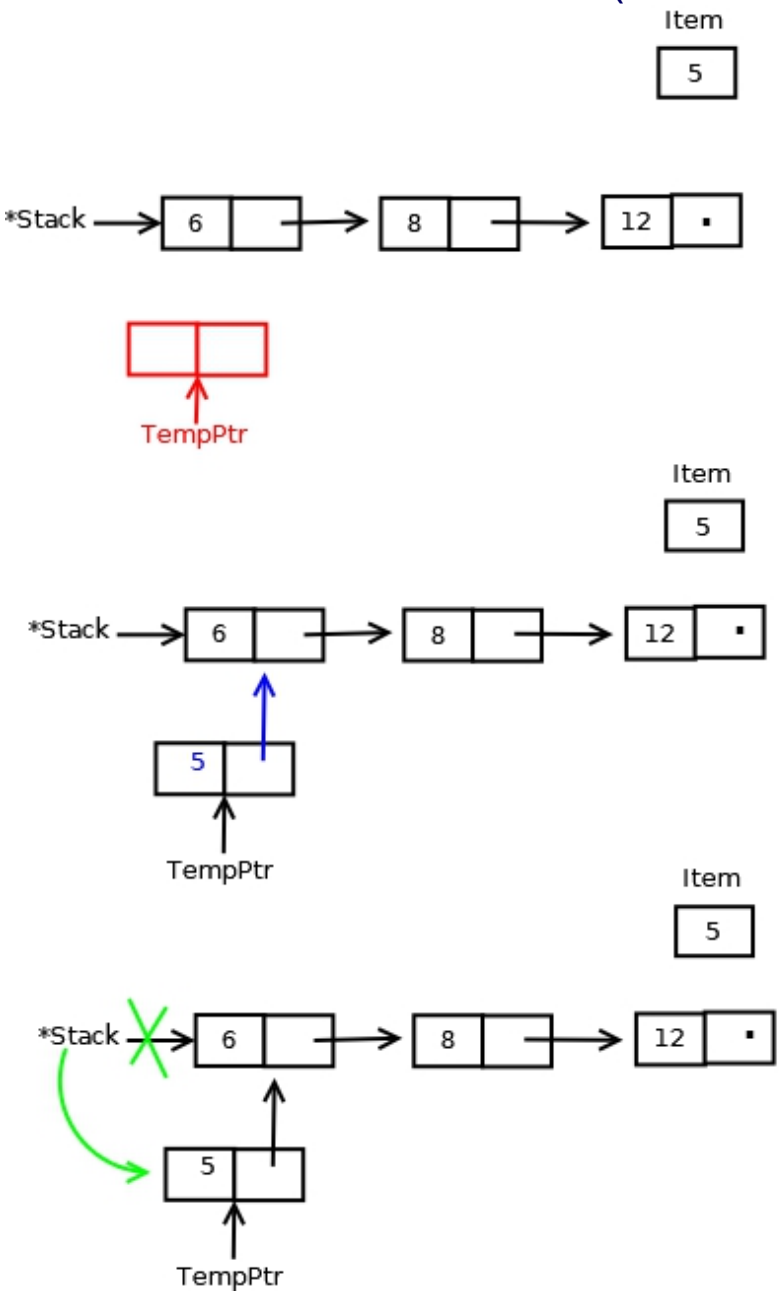
κλήση από main

```
{
  StackPointer AStack;
  StackElementType AnItem;
....
  Pop(&AStack, &AnItem);
....
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Στοίβα

```
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 Push(StackPointer *Stack, StackElementType Item)

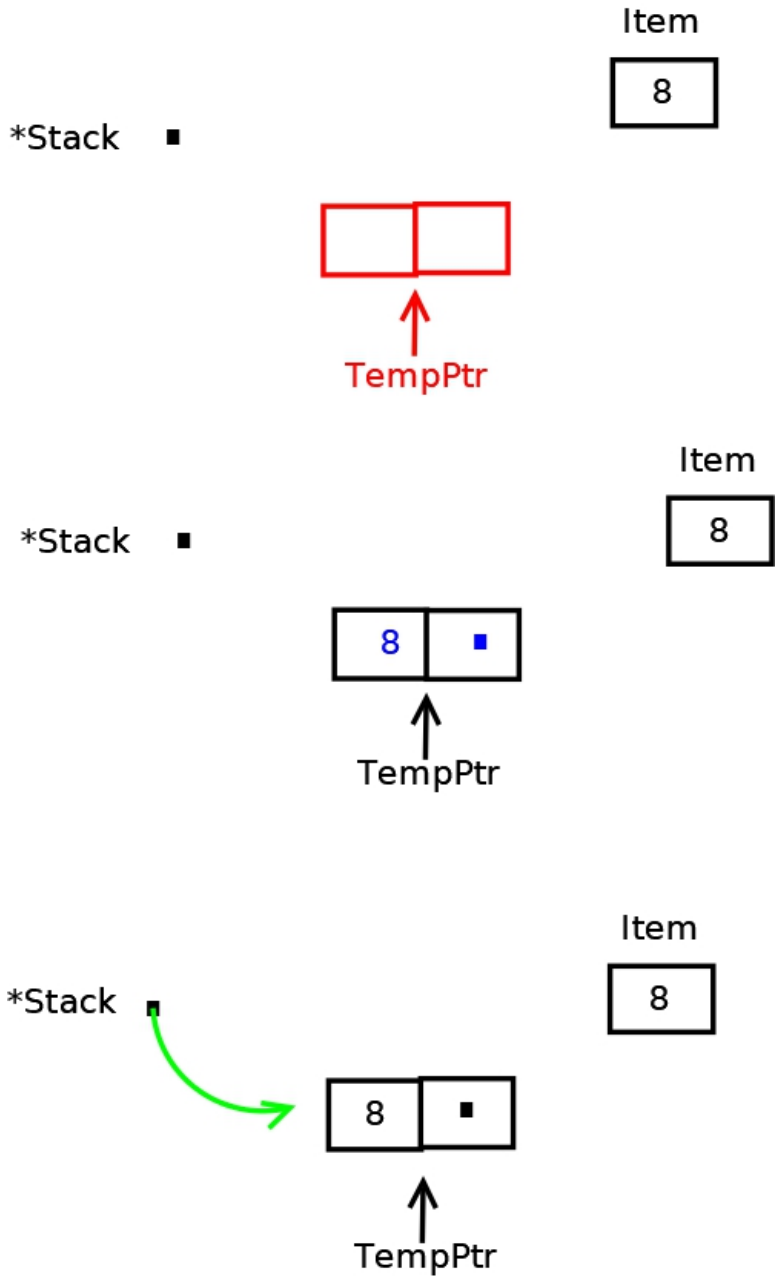


StackPointer TempPtr;

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

```
   κλήση από main
{
   StackPointer AStack;
   StackElementType AnItem;
….
   scanf("%d",&AnItem);
    Push(&AStack,AnItem);….
}
```

void Push(StackPointer *Stack, StackElementType Item)



StackPointer TempPtr;

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

```
   κλήση από main
{
   StackPointer AStack;
   StackElementType AnItem;
….
   scanf("%d",&AnItem);
    Push(&AStack,AnItem);….
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Ουρά

Με παρόμοιο τρόπο μπορεί να υλοποιηθεί μια ουρά ως συνδεδεμένη λίστα.

Η ουρά, 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.

Πακέτο για τον ΑΤΔ Συνδεδεμένη Ουρά

```
// QueueADT.h
typedef int QueueElementType;
typedef struct QueueNode *QueuePointer;
typedef struct QueueNode
{
        QueueElementType Data;
        QueuePointer Next;
} QueueNode;

typedef struct
{
        QueuePointer Front;
        QueuePointer Rear;
} QueueType;
typedef enum {
                FALSE, TRUE
} boolean;
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Ουρά

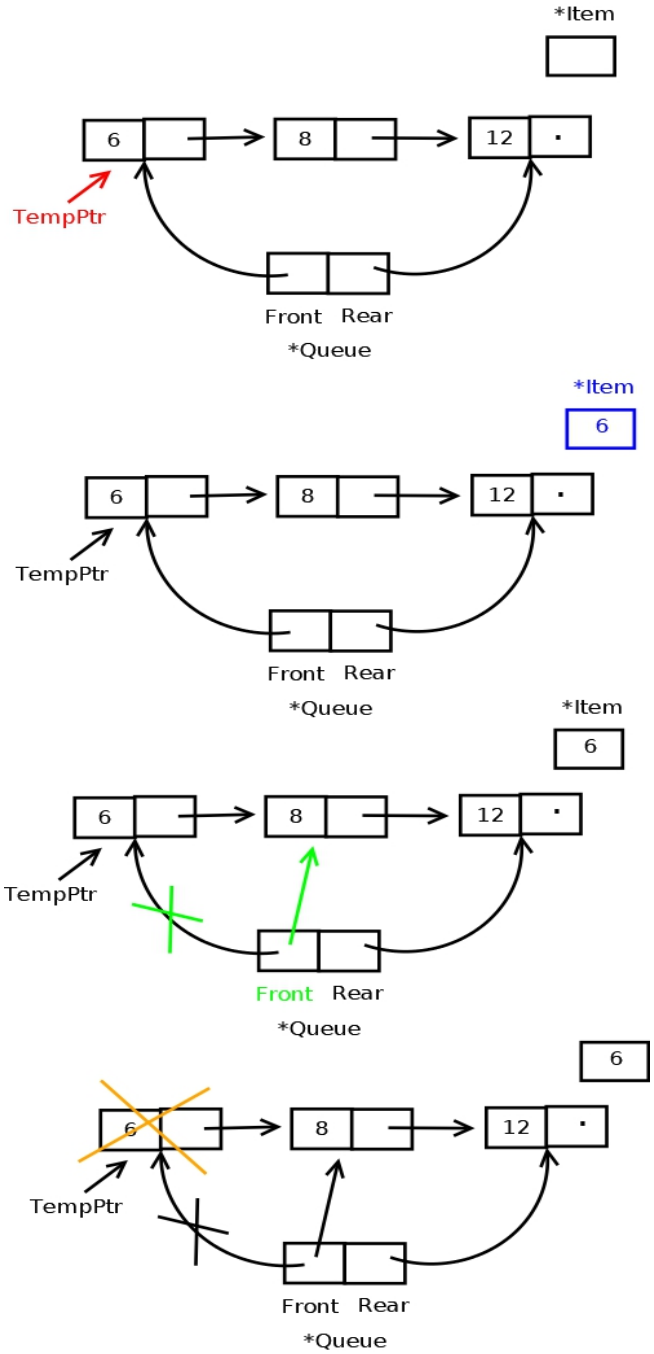
```
void CreateQ(QueueType *Queue);

boolean EmptyQ(QueueType Queue);

void AddQ(QueueType *Queue, QueueElementType Item);

void RemoveQ(QueueType *Queue, QueueElementType *Item);
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Ουρά

```
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);
}
```

Πακέτο για τον ΑΤΔ Συνδεδεμένη Ουρά

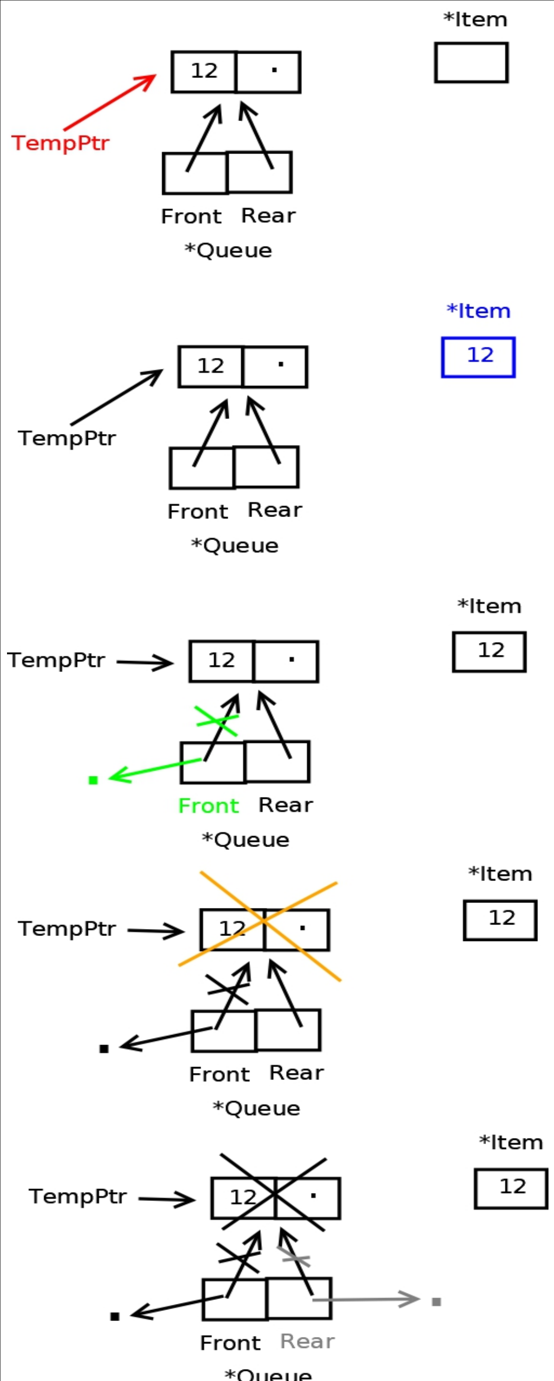
```
void RemoveQ(QueueType *Queue, QueueElementType *Item)
/*Δέχεται:        Μια συνδεδεμένη ουρά.
  Λειτουργία:     Αφαιρεί το στοιχείο Item από την  κορυφή της συνδεδεμένης
                  ουράς, αν δεν είναι  κενή.
  Επιστρέφει:     Το στοιχείο Item και την τροποποιημένη συνδεδεμένη ουρά.
  Έξοδος: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

    if (EmptyQ(*Queue))
            printf("EMPTY Queue\n");

   else
   {
       TempPtr = Queue->Front;
       *Item = TempPtr->Data;
       Queue->Front = Queue->Front->Next;
       free(TempPtr);
       if (Queue->Front == NULL)
                   Queue->Rear = NULL;
    }
}
```



void RemoveQ(QueueType *Queue, QueueElementType *Item)

```
QueuePointer TempPtr;

if (EmptyQ(*Queue))
        printf("EMPTY Queue\n");

else
{
        TempPtr = Queue->Front;
        *Item = TempPtr->Data;
        Queue->Front = Queue->Front->Next;
        free(TempPtr);
        if (Queue->Front == NULL)
                Queue->Rear = NULL;
}
```

```
main()
{
   QueueType AQueue;
   QueueElementType AnItem;

   RemoveQ(&AQueue, &AnItem);
    printf("KORYFAIO STOIXEIO %d\n",AnItem);
....
}
```



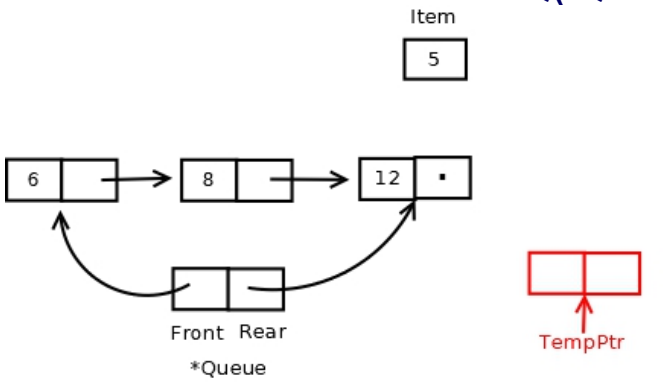

```
void RemoveQ(QueueType *Queue, QueueElementType *Item)

    QueuePointer TempPtr;

    if (EmptyQ(*Queue))
                printf("EMPTY Queue\n");

   else
   {
            TempPtr = Queue->Front;
            *Item = TempPtr->Data;
            Queue->Front = Queue->Front->Next;
            free(TempPtr);
            if (Queue->Front == NULL)
                    Queue->Rear = NULL;
   }
```

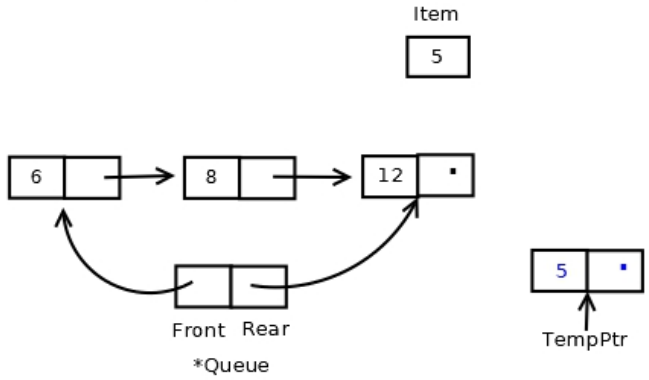
```
main()
{
   QueueType AQueue;
   QueueElementType AnItem;

   RemoveQ(&AQueue, &AnItem);
    printf("KORYFAIO STOIXEIO %d\n",AnItem);
….
}
```

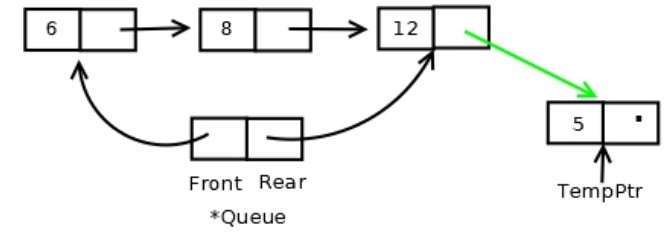
Πακέτο για τον ΑΤΔ Συνδεδεμένη Ουρά

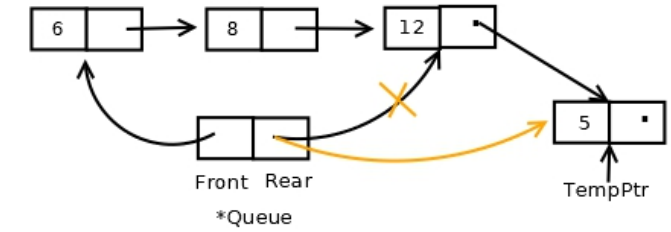
```
void AddQ(QueueType *Queue, QueueElementType Item)
/*Δέχεται:      Μια συνδεδεμένη ουρά Queue και ένα  στοιχείο Item.
  Λειτουργία:   Προσθέτει το στοιχείο Item στο τέλος της συνδεδεμένης ουράς
                Queue.
  Επιστρέφει:   Την τροποποιημένη ουρά.*/
{
        QueuePointer TempPtr;

        TempPtr = (QueuePointer)malloc(sizeof(struct QueueNode));
        TempPtr->Data = Item;
        TempPtr->Next = NULL;
        if (Queue->Front == NULL)
                Queue->Front = TempPtr;
        else
                Queue->Rear->Next = TempPtr;
        Queue->Rear=TempPtr;
}
```

void AddQ(QueueType *Queue, QueueElementType Item)





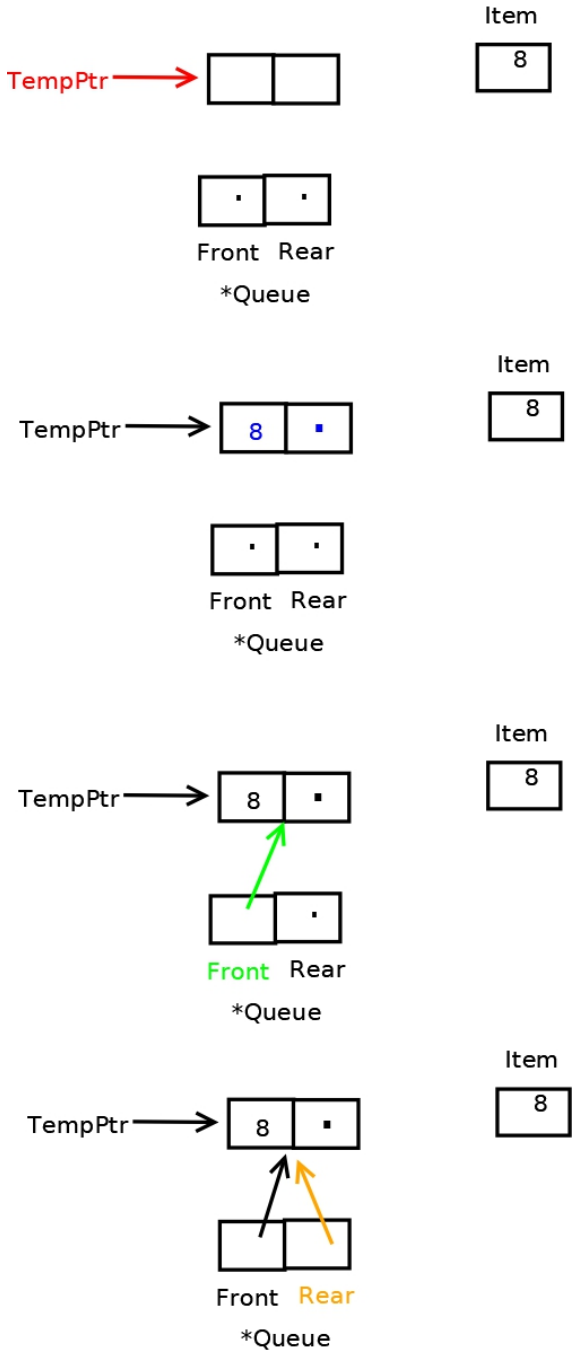





```
QueuePointer TempPtr;
TempPtr = (QueuePointer)malloc(sizeof(struct QueueNode));

TempPtr->Data = Item;
TempPtr->Next = NULL;
 if (Queue->Front == NULL)
        Queue->Front = TempPtr;
 else
        Queue->Rear->Next = TempPtr;
Queue->Rear=TempPtr;
```

```
main()
{
   QueueType AQueue;
   QueueElementType AnItem;

   scanf("%d",&AnItem);
   AddQ(&AQueue,AnItem);
   ….
}
```

void AddQ(QueueType *Queue, QueueElementType Item)



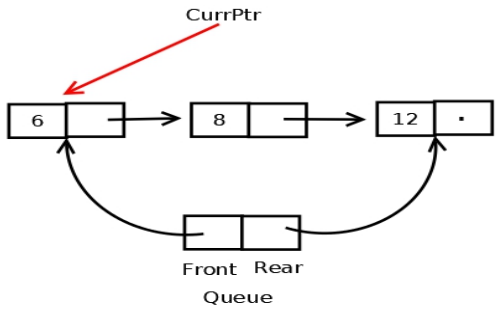

```
QueuePointer TempPtr;
TempPtr = (QueuePointer)malloc(sizeof(struct
QueueNode));

TempPtr->Data = Item;
TempPtr->Next = NULL;
 if (Queue->Front == NULL)
        Queue->Front = TempPtr;
 else
        Queue->Rear->Next = TempPtr;
 Queue->Rear=TempPtr;
```

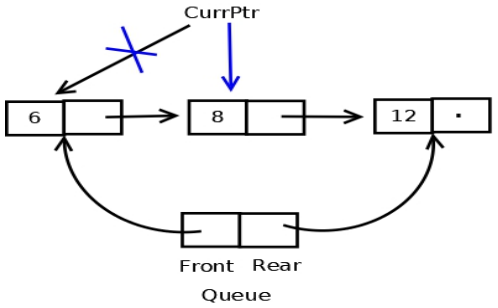
```
main()
{
   QueueType AQueue;
   QueueElementType AnItem;

   scanf("%d",&AnItem);
   AddQ(&AQueue,AnItem);
   ….
}
```

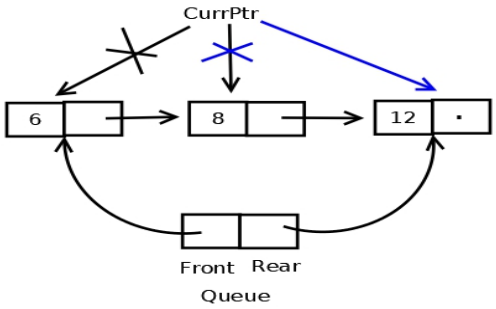
Πακέτο για τον ΑΤΔ Συνδεδεμένη Ουρά

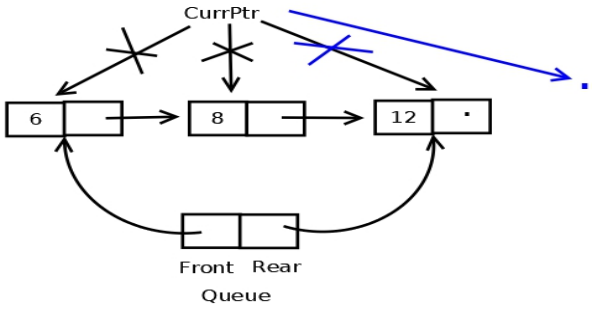
```
void TraverseQ(QueueType Queue)

/*Δέχεται:Μια συνδεδεμένη ουρά.
  Λειτουργία:       Διασχίζει τη συνδεδεμένη ουρά, αν δεν είναι κενή.
  Επιστρέφει:       Εξαρτάται από το είδος της  επεξεργασίας.*/
{
        QueuePointer CurrPtr;

        if (EmptyQ(Queue))
                printf("EMPTY Queue\n");
        else
        {
                CurrPtr = Queue.Front;
                while ( CurrPtr!=NULL )
                {
                        printf("%d\n", CurrPtr->Data);
                                CurrPtr = CurrPtr->Next;
                }
        }
}
```

void TraverseQ(QueueType Queue)










```
QueuePointer CurrPtr;

if (EmptyQ(Queue))
        printf("EMPTY Queue\n");
else{
        CurrPtr = Queue.Front;
        while ( CurrPtr!=NULL ){
                printf("%d\n", CurrPtr->Data);
                CurrPtr = CurrPtr->Next;
        }
}
```

```
main()
{
   QueueType AQueue;

   TraverseQ(AQueue);
   ….
}
```