

## Ενότητα 2: Στοιίβες – Ουρές - Λίστες

### Ασκήσεις και Λύσεις

#### Άσκηση 1

Έστω ότι μια βιβλιοθήκη σας παρέχει πρόσβαση σε στοιίβες ακεραίων. Η βιβλιοθήκη σας επιτρέπει να ορίσετε μια στοιίβα και να καλέσετε τις 5 βασικές λειτουργίες σε αυτή.

Για παράδειγμα, ο ορισμός μιας στοιίβας (ή μιας ουράς) S1 γίνεται με τη δήλωση:  
Stack S1,

ενώ υποστηρίζονται οι εξής λειτουργίες:

- void MakeEmptyStack(stack S)
- boolean IsEmptyStack(stack S)
- int Top(Stack S)
- int Pop(Stack S)
- void Push(Stack S, int x)

Έστω ότι θέλετε να δημιουργήσετε ένα πρόγραμμα το οποίο απαιτεί επιπρόσθετα των παραπάνω λειτουργιών και την εκτέλεση της λειτουργίας PrintStack(Stack S), η οποία εκτυπώνει όλα τα στοιχεία της στοιίβας S. Η εκτέλεση της PrintStack() δεν θα πρέπει να επηρεάζει τη μορφή της στοιίβας (δηλαδή η στοιίβα θα πρέπει να περιέχει τα ίδια στοιχεία και με την ίδια σειρά πριν και μετά την εκτέλεση της PrintStack()). Παρουσιάστε ψευδο-κώδικα που θα υλοποιεί την PrintStack().

**Υπόδειξη:** Επιτρέπεται να χρησιμοποιήσετε μια ή περισσότερες extra στοιίβες προκειμένου να υλοποιήσετε τις παραπάνω λειτουργίες.

#### Λύση

```
void PrintStack(Stack S) {  
    int element;  
    Stack tmpS;  
  
    MakeEmptyStack(tmpS); // βοηθητική στοιίβα  
  
    while(!IsEmptyStack(S)){ // Διατρέχουμε τη στοιίβα S, εκτελώντας επαναληπτικά την Pop.  
        // Τα στοιχεία που αφαιρούνται από την S τοποθετούνται στην tmpS  
        // για να μεταφερθούν και πάλι στην S μετά την περάτωση της εκτύπωσης  
        element = Pop(S); // αφαίρεσε το επόμενο στοιχείο από την S  
        Push(tmpS, element); // τοποθέτησέ το στην στοιίβα tmpS  
        print(element); // τύπωσε το  
    }  
  
    while (!IsEmptyStack(tmpS)) { // επαναφορά της S στην αρχική της μορφή  
        element = Pop(tmpS); // αφαίρεσε στοιχείο από την S  
        Push(S, element); // τοποθέτησέ το στην S, όπου και υπήρχε αρχικά  
    }  
}
```

**Άσκηση 2**

Δίνεται η ακόλουθη υλοποίηση για αραιούς δυσδιάστατους πίνακες. Ο αραιός δυσδιάστατος πίνακας  $A[n][m]$  υλοποιείται με έναν boolean δυσδιάστατο πίνακα  $B[n][m]$  (κάθε στοιχείο του  $B$  έχει την τιμή 0 ή 1) και μια δυναμική συνδεδεμένη λίστα που περιέχει τα μη-μηδενικά στοιχεία του πίνακα  $A$ . Για τον πίνακα  $B$  ισχύουν τα ακόλουθα. Για κάθε  $i, j$ , με  $0 \leq i \leq n$ ,  $1 \leq j \leq m$ ,  $B[i][j] = 1$  αν και μόνο αν  $A[i][j] \neq 0$ , και  $B[i][j] = 0$  αν και μόνο αν  $A[i][j] = 0$ .

Τα μη-μηδενικά στοιχεία στη λίστα αποθηκεύονται ως εξής: «τα μη-μηδενικά στοιχεία οποιασδήποτε γραμμής  $i$  είναι αποθηκευμένα στη λίστα πριν από τα μη-μηδενικά στοιχεία οποιασδήποτε γραμμής  $j > i$  και τα μη-μηδενικά στοιχεία οποιασδήποτε στήλης  $k$  απαντώνται στη λίστα πριν από τα μη-μηδενικά στοιχεία οποιασδήποτε στήλης  $m > k$ ». Για παράδειγμα, δίνεται ο αραιός πίνακας  $A$ :

|    |   |   |   |   |
|----|---|---|---|---|
| 0  | 7 | 0 | 0 | 0 |
| 1  | 2 | 0 | 0 | 3 |
| 0  | 0 | 4 | 0 | 0 |
| 12 | 0 | 0 | 0 | 0 |

Ο πίνακας  $B$  που αντιστοιχεί στον  $A$  είναι ο ακόλουθος:

|   |   |   |   |   |
|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |

και η λίστα έχει την εξής μορφή:

$7 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 12$

Παρουσιάστε αλγόριθμο που θα υλοποιεί τη λειτουργία:

`int Access(int i, int j)`

η οποία επιστρέφει το στοιχείο στη γραμμή  $i$  και στη στήλη  $j$  του  $A$ , δεδομένου ότι ο  $A$  υλοποιείται όπως περιγράφηκε παραπάνω.

**Λύση**

```
int Search(int i, int j){
    int k,l;
    struct node *p;
    int cnt = 0;

    if (i > n OR i < 0 OR j > m OR j < 0) {
        error();           // το στοιχείο προς πρόσβαση είναι εκτός ορίων πίνακα
        return;
    }
    if (B[i][j]==0)
        return 0;         // το στοιχείο προς πρόσβαση είναι 0
```

```

for (k = 0; k < i; k++) {    // διατρέχουμε τις γραμμές 0 ως i του B
    for (l = 0; l < m; l++) // διατρέχουμε όλα τα στοιχεία κάθε τέτοιας γραμμής

        if (B[k][l] == 1) cnt++; // μετράμε πόσα μη-μηδενικά στοιχεία υπάρχουν σε αυτές τις γραμμές
}

for (l = 0; l < j; l++){    // διατρέχουμε τη γραμμή i
    if (B[i][l]) cnt++;    // και συναθροίζουμε στη cnt τα μη μηδενικά στοιχεία
                          // που προηγούνται του στοιχείου j σε αυτή
}

p = L;
while (p != NULL) {        // διατρέχουμε τη λίστα L, στην οποία βρίσκονται τα μη-μηδενικά στοιχεία,
                          // μέχρι να φτάσουμε στο (cnt+1)-οστό στοιχείο της L,
                          // το οποίο αντιστοιχεί στο στοιχείο [i,j] που ζητείται
    if (cnt == 0) return p->data;
    p = p->next;
    cnt--;
}
}

```

### Άσκηση 3

Υλοποιήστε αλγόριθμο για την διαγραφή στοιχείου από μια ταξινομημένη λίστα.

**Λύση:**

```

Node * ListDelete(Node *L, int x) {
    Node *prev = NULL, *curr = L;    // βοηθητικοί δείκτες

    while(curr != NULL && curr->data < x){    // όσο δεν έχω διατρέξει όλη τη λίστα
                                              // και δεν έχω βρει το στοιχείο
        prev=curr;    // αποθηκεύω δείκτη στο προηγούμενο στοιχείο πριν προχωρήσω τον curr
        curr=curr->next;    // μετακινώ τον curr στο επόμενο στοιχείο της L
    }
    if (curr == NULL || curr->data > x) { // αν δεν υπάρχει το x στη λίστα
        printf("Element x does not exist in L");
        return L;
    }
    if (prev == NULL)    // το x είναι το πρώτο στοιχείο της L
        L = L->next;    // αφαιρέσέ το μετακινώντας τον L στο επόμενο στοιχείο
    else                // το x δεν είναι το πρώτο στοιχείο της L
        prev->next = curr->next;    // διαγραφή του x
    return L;    // επιστροφή (ενδεχόμενης) νέας τιμής του δείκτη L
}

```

**Άσκηση 4**

Να παρουσιαστεί υλοποίηση δύο στοιβών, τα στοιχεία των οποίων αποθηκεύονται στον ίδιο πίνακα  $A[N]$  (τα στοιχεία του πίνακα είναι τα  $A[0], \dots, A[N-1]$ ). Η υλοποίηση πρέπει να γίνει με τέτοιο τρόπο ώστε καμία στοιβή να μην υπερχειλίζει παρά μόνο αν το συνολικό πλήθος των στοιχείων και στις δύο στοιβές να ισούται με  $n$ . Όλες οι λειτουργίες στις στοιβές θα πρέπει να εκτελούνται σε χρόνο  $\Theta(1)$ .

**Λύση**

Έστω  $S0$  και  $S1$  οι δύο στοιβές. Έστω επίσης ότι τα στοιχεία  $Length[0]$  και  $Length[1]$  ενός πίνακα  $Length[]$  δύο θέσεων περιέχουν το μήκος των στοιβών  $S1$  και  $S2$ , αντίστοιχα.

Στην υλοποίηση που ακολουθεί, η  $S1$  αυξάνει από αριστερά προς τα δεξιά, ενώ η  $S2$  από δεξιά προς τα αριστερά.

Κάθε μια από τις λειτουργίες, παίρνει ως παράμετρο μια ακέραια μεταβλητή *which* που υποδηλώνει σε ποια στοιβή θα πρέπει να εφαρμοστεί η λειτουργία. Συγκεκριμένα, όταν η *which* έχει την τιμή 0 (1) η λειτουργία πρέπει να εφαρμοστεί στην  $S0$  ( $S1$ , αντίστοιχα).

**Απλή υλοποίηση (στην οποία οι πίνακες  $A[]$  και  $Length[]$  είναι καθολικές μεταβλητές).**

Type  $A[n]$ ;

int  $Length[2] = \{0,0\}$ ;

```
void MakeEmptyStack(boolean which){
    Length[which] = 0;
}
```

```
boolean IsEmptyStack(boolean which){
    if (Length[which] == 0) return 1;
    else return 0;
}
```

```
Type Top(boolean which){
    if (IsEmptyStack(which)) {
        print("Stack is empty");
        return;
    }
    else if (which==0)
        return A[Length[which] - 1]; // η S0 αναπτύσσεται από την αρχή του πίνακα προς το τέλος του
    else return A[N - Length[which]]; // η S1 αναπτύσσεται από το τέλος του πίνακα προς της αρχή του
    // άρα το κορυφαίο στοιχείο της βρίσκεται στη θέση N - Length[1] του A
}
```

```
Type Pop(boolean which) {
    Type x;

    if (Length[which] == 0) {
        print("Stack is empty");
        return;
    }
}
```

```

    }

    x=Top(which); // αποθηκεύω στο x το κορυφαίο στοιχείο της στοίβας και
    Length[which] = Length[which] - 1; // το αφαιρώ, μειώνοντας το μήκος της στοίβας κατά 1
    return x;
}

void Push(Type x, boolean which) {
    if (Length[0] + Length[1] == N) {
        print("Stack is full");
        return;
    }
    Length[which] = Length[which] + 1;
    if (which == 0) A[Length[which] - 1] = x;
    else A[N - Length[which]] = x;
}

```

**Πιο πολύπλοκη υλοποίηση (στην οποία οι πίνακες A[] και Length[] είναι πεδία ενός struct).**

```

typedef struct stack{
    int Length[2];
    Type A[N];
} STACK;

pointer MakeEmptyStack(boolean which) {
    pointer S; // βοηθητικός δείκτης
    int flag = 0;

    if (flag == 0) { // η ανάθεση μνήμης πρέπει να γίνει μόνο μια φορά
        S=newcell(STACK);
        flag = 1;
    }
    S->Length[which] = 0;
    return S;
}

boolean IsEmptyStack(pointer S, boolean which){
    if (S->Length[which] == 0) return 1;
    else return 0;
}

Type Top(pointer S, boolean which){
    if (IsEmptyStack(S, which)) {
        print("Stack is empty");
        return;
    }
    else if (which==0)
        return (S->A)[S->Length[which] - 1];
    else return (S->A)[N - S->Length[which]];
}

```

```

Type Pop(pointer S, boolean which){
    Type x;

    if (S->Length[which] == 0) {
        print("Stack is empty");
        return;
    }

    x = Top(S, which);
    S->Length[which] = S->Length[which] - 1;
    return x;
}

void Push(pointer S, Type x, boolean which){
    if (S->Length[0] + S->Length[1] == N) {
        print("Stack is full");
        return;
    }
    S->Length[which] = S->Length[which] + 1;
    if (which == 0) (S->A)[S->Length[which] - 1] = x;
    else (S->A)[N - S->Length[which]] = x;
}

```

### **Ασκηση 3**

Υλοποιήστε αναδρομικό αλγόριθμο για την εκτύπωση των στοιχείων μιας λίστας με ανάποδη σειρά.

**Λύση:**

```

void ListPrint(Node *L) {
    if (L == NULL) return;           // Αν φτάσαμε στο τέλος της λίστας
    ListPrint(L->next);               // Καλώ αναδρομικά μέχρι να βρώ το τελευταίο στοιχείο
    print(L->data)                     // Εκτυπώνω τα στοιχεία της λίστας
}

```

### **Ευχαριστίες**

Ευχαριστούμε την πρώην βοηθό του μαθήματος Κατερίνα Τζομπανάκη για την παραγωγή της ηλεκτρονικής έκδοσης του παραπάνω υλικού.