מבני נתונים בסיסיים+ADT

<u>במצגת זו נכיר מבני נתונים בסיסיים הבאים:</u>

1. מחסנית
2. תור

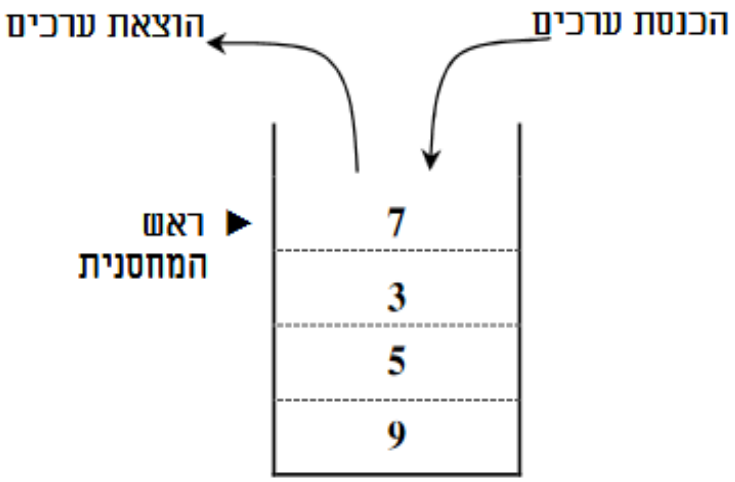
מחסנית

מחסנית (Stack) היא סוג של אוסף התומך בפעולות הכנסה והוצאה, כך שמתקיימת התכונה הבאה: הוצאת ערך ממחסנית אפשרית רק כאשר היא אינה ריקה והיא מחזירה תמיד את הערך שהוכנס אחרון, מבין הערכים הקיימים במחסנית. ניתן לחשוב על מחסנית כסדרה, שהפעולות עליה מתבצעות רק בקצה אחד של הסדרה, הנקרא **ראש המחסנית** (כמתואר באיור). הכנסת ערך למחסנית מוסיפה ערך חדש בראש המחסנית. הוצאת ערך מהמחסנית, מסירה את הערך המצוי בראש המחסנית וחושפת את הערך הבא בסדרה. נהוג לכנות את פעולת ההכנסה למחסנית בשם דחיפה, ואת פעולת ההוצאה בשם: שליפה.

קל לראות מתיאור זה שהערך הנשלף מהמחסנית הוא תמיד הערך האחרון שנדחף אליה. הפרוטוקול המגדיר את דרך הגישה לערכים נקרא **LIFO** – ראשי התיבות של המילים: **L**ast **I**n **F**irst **O**ut.

האיור שלפניכם הוא "תצלום" מצב רגעי של מחסנית לאחר שהוכנסו לתוכה הערכים:

9, 5, 3, 7, בזה אחר זה (מימין לשמאל).

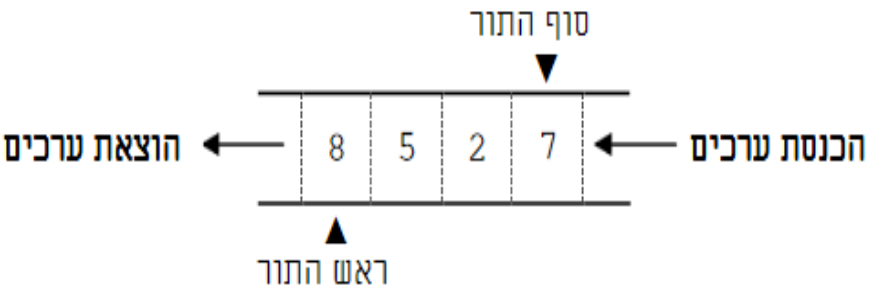


ניתן לראות שהערך 7, שהוכנס אחרון, נמצא בראש המחסנית.

תור

תור (Queue) הוא סוג של אוסף. הערכים בתור מאורגנים כסדרה, והפרוטוקול עבור פעולות ההכנסה וההוצאה הוא זה: הוצאת ערכים בתור מותרת רק מצד אחד, הקרוי **ראש התור**, והכנסת ערכים מותרת רק לצד האחר, הקרוי **סוף התור** (או 'זנב התור').

גישה כזו לטיפול בערכים נקראת **FIFO** – ראשי התיבות של המילים: **First In First Out**: האיבר הראשון שנכנס לתור הוא הראשון לצאת ממנו.



מימוש של תור ומחסנית

מקובל לממש תור ומחסנית בעזרת מערך או רשימה מקושרת.

לכל אחד מהצעות מימוש הבאות, יש להחליט האם אכן מדובר במימוש תור/מחסנית ואם כן, האם המימוש יעיל.

<u>מחסנית:</u>

1. נממש מחסנית בעזרת רשימה מקושרת חד-כיוונית כאשר ההוספה למחסנית מתבצעת לראש הרשימה והוצאה ממחסנית מתבצעת גם מראש הרשימה.
2. נממש מחסנית בעזרת רשימה מקושרת חד-כיוונית כאשר ההוספה למחסנית מתבצעת לסוף הרשימה והוצאה ממחסנית מתבצעת גם מסוף הרשימה.
3. נממש מחסנית בעזרת רשימה מקושרת חד-כיוונית כאשר ההוספה למחסנית מתבצעת לסוף הרשימה והוצאה ממחסנית מתבצעת מראש הרשימה.
4. נממש מחסנית בעזרת מערך כאשר ההוספה למחסנית מתבצעת לסוף המערך והוצאה ממחסנית מתבצעת מסוף המערך ע"י הקטנת משתנה שמכיל כמות הנתונים במערך (size--).
5. נממש מחסנית בעזרת מערך כאשר ההוספה למחסנית מתבצעת לתחילת המערך(בשביל זה יש להזיז את כל הנתונים ימינה) והוצאה ממחסנית מתבצעת מראש המערך(בשביל זה יש להזיז את כל הנתונים שמאלה).

מימוש של תור ומחסנית

<u>תור:</u>

1. נממש תור בעזרת רשימה מקושרת חד-כיוונית כאשר ההוספה לתור מתבצעת לסוף הרשימה והוצאה מתור מתבצעת מראש הרשימה.

2.נממש תור בעזרת רשימה מקושרת חד-כיוונית כאשר ההוספה לתור מתבצעת לראש הרשימה והוצאה מתור מתבצעת מסוף הרשימה.

3. נממש תור בעזרת רשימה מקושרת חד-כיוונית כאשר ההוספה לתור מתבצעת לסוף הרשימה והוצאה מתור מתבצעת גם מסוף הרשימה.

4. נממש תור בעזרת מערך כאשר ההוספה לתור מתבצעת לסוף המערך והוצאה מתור מתבצעת מתחילת המערך (בשביל זה יש להזיז את כל הנתונים שמאלה).

5. נממש תור בעזרת מערך כאשר ההוספה לתור מתבצעת לתחילת המערך (בשביל זה יש להזיז את כל הנתונים ימינה) והוצאה מתור מתבצעת מסוף המערך ע" הקטנת משתנה שמכיל כמות הנתונים במערך (size--).

מחסנית ותור

1.פעולת הוספה למחסנית נקראת push ופעולת ההוצאה – pop.

2. פעולת הוספה לתור נקראת enqueue ופעולת ההוצאה – dequeue.

3.לפונקציות האחראיות על הוצאת נתון ממנחסית/תור אסור לקבל כפרמטר את הנתון להוצאה!! ניהול של מבני נתונים תור/מחסנית אמור להיות כזה שמאפשר בכל רגע לדעת מי המועמד לצאת!

4. בדרך כלל מקובל שפונקציה האחראית על הוצאת נתון ממחסנית/תור גם תחזיר את הנתון שהוצא!

5. מחסנית ותור – טיפוס נתונים מופשט (ADT)

<u>טיפוס נתונים אבסטרקטי (Abstract Data Type = ADT):</u>

אוסף של פעולות על נתונים כלשהם.

- המשתמש ב-ADT מכיר את הפעולות והשפעתן על הנתונים.
- אינו נדרש להכיר את פרטי המימוש.

מחסנית ותור

לדוגמה:

מחסנית (Stack)

מוגדרת ע"י הפעולות הבאות:

- Push(S, x) – הכנסת איבר לראש המחסנית (כולל בדיקת overflow)
- Pop(S) – הוצאת האיבר שבראש המחסנית (כולל בדיקת underflow)
- StackEmpty(S) ו- StackFull(S) – בדיקה אם המחסנית ריקה/מלאה

מבנה נתונים (data structure):

שיטת אחסון וארגון של נתונים שמאפשרת ביצוע פעולות מסוימות.

מבנה נתונים הוא מימוש ל- ADT.

מבני נתונים אפשריים למימוש מחסנית:
1. מערך
2. רשימה מקושרת

ADT - Abstract Data Type

הכרנו את הטיפוסים הבסיסיים הקיימים בשפת C : שלם(int) ,ממשי(float) וכו'.
לעיתים הטיפוסים הבסיסיים לא מספיקים ואנחנו מגדירים טיפוסים חדשים. לדוגמא, יצרנו טיפוס חדש עבור צומת ברשימה מקושרת.

טיפוסי נתונים מופשטים (ADT, Abstract Data Types) הם טיפוסים המורכבים מטיפוסים בסיסיים יותר וכוללים גם פונקציות לטיפול בנתונים.
כשאומרים ADT – מתכוונים לסט נתונים וסט פעולות שניתן לבצע על הנתונים, כאשר לא מדובר על מימוש ספציפי כלשהו ולא על טיפוס נתונים ספציפי כלשהו.

דוגמא – מימוש Stack – עדיין לא ADT

```
/* Header file of stack using array implementation  */

#ifndef _STACK_H
#define _STACK_H

#include <stdio.h>
#include <ctype.h>
#include <conio.h>
#include <string.h>

typedef char Titem;

#define MAXN 5
typedef enum {NOT_OK, OK } Tboolean;

typedef struct {
        Titem array[MAXN];     /*Stack of chars*/
         int top;
     } Tstack;
```

דוגמא – מימוש Stack – עדיין לא ADT

```
/*Function Declarations*/

void initialize_stack (Tstack *Pstack);
Tboolean push( Tstack *Pstack, Titem item);
Tboolean pop( Tstack *Pstack, Titem *Pitem);
void print_stack (Tstack *Pstack);
}

 # endif
```

דוגמא – מימוש Stack – עדיין לא ADT

```
/* Array implementation of stack. */

#include “Stack.h"

void initialize_stack ( Tstack *Pstack)
{
    Pstack->top = -1;
}

Tboolean push( Tstack *Pstack, Titem item)
{
    if (Pstack->top >= MAXN - 1)
        return NOT_OK;
    else {
        (Pstack->top)++;
         Pstack->array[Pstack->top] = item;
         return OK;
    }
}
```

דוגמא – מימוש Stack – עדיין לא ADT

```
Tboolean pop( Tstack *Pstack, Titem *Pitem)
{
    if (Pstack->top == - 1)
     return NOT_OK;
    else {
        *Pitem = Pstack->array[Pstack->top];
         (Pstack->top)--;
         return OK;
    }
}


void print_stack (Tstack *Pstack) {
  int i;
  printf( "\n Stack is : ");
  for (i = Pstack->top; i >= 0 ; i-- )
      printf("\n %c ", Pstack->array[i]);
  printf("\n\n");
}
```

דוגמא – מימוש Stack – עדיין לא ADT

```
/* Application file */

#include “Stack.h"

void main(void) {
    Tstack stack;
    Tboolean succeeded;
    char chr;

    initialize_stack(&stack);

    printf("\n Enter letter to push onto stack");
    printf("\n Enter 1 to pop a letter from stack");
    printf("\n Enter 2  to end the program\n");
```

דוגמא – מימוש Stack – עדיין לא ADT

```
chr = _getche();
while (chr != '2')
{
   if (isalpha(chr))  {                    /*if it's letter*/
     succeeded=push(&stack, chr);
     if (!succeeded)
       printf("\n Push operation failed\n");
     print_stack(&stack);
  }
   if (chr == '1') {                       /*pop operation*/
      succeeded = pop(&stack, &chr);
      if  (succeeded) {
        printf("\nLetter popped from stack is %c ", chr);
        print_stack(&stack);
      }
      else
         printf("\nPop operation failed\n ");
   }
  chr = _getche();
 }
}
```

מימוש ADT Stack

```
/*Stack_General.h file - header file*/

#ifndef _STACK_GENERAL
#define _STACK_GENERAL

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

#define MAXN 5
typedef enum {NOT_OK, OK } Tboolean;
typedef void* Titem;        /*the general type*/

typedef struct  {
        Titem array[MAXN];
        int top;
    } Tstack;
```

שימוש נוסף ב-typedef – שם משמעותי לטיפוס

מבנה כללי – כל טיפול בו נעשה בפונקציות כלליות המוגדרות בקובץ המימוש

מימוש ADT Stack

```
/*General functions declarations*/

    void initialize_stack (Tstack *Pstack);
    Tboolean push( Tstack *Pstack, Titem item);
    Tboolean pop( Tstack *Pstack, Titem *Pitem);
    void print_stack (Tstack *Pstack,
       void(*Print)(Titem )) ;
    void free_stack (Tstack *Pstack,
       void(*FrElm)(Titem));


#endif
```

מימוש ADT Stack

```
/*Stack_General.c - Implementations of operation
functions*/

#include "Stack_General.h"

void initialize_stack ( Tstack *Pstack)  {
    Pstack->top = -1;
}

Tboolean push( Tstack *Pstack, Titem item)
{
    if (Pstack->top >= MAXN - 1)
        return NOT_OK;
    else {
        (Pstack->top)++;
        Pstack->array[Pstack->top] = item;
        return OK;
    }
}
```

מימוש ADT Stack

```
Tboolean pop( Tstack *Pstack, Titem *Pitem) {
    if (Pstack->top == - 1)
        return NOT_OK;
    else {
        *Pitem = Pstack->array[Pstack->top];
        (Pstack->top)--;
        return OK;
    }
}

void print_stack (Tstack *Pstack,
void(*Print)(Titem)) {
      int i;
      printf("\nStack is : ");
      for (i = Pstack->top; i >= 0 ; i-- )
          Print(Pstack->array[i]); /*call to special
function*/
     printf("\n\n");
}
```

מימוש ADT Stack

```
void free_stack(Tstack* Pstack,void (*FrElm)(Titem)
{
  int i;
   for(i=0;i<=Pstack->top; i++)     /*free memory */
     FrElm(Pstack->array[i]);
}
```

מימוש ADT Stack

```
/* Stack_General_Main.c file*/

#include "Stack_General.h"

        /*specific function – for chars only */
void Print_Char(Titem s) {
  printf("\n %c ",*(char*)s);
}
       /*specific function – for deleting chars only */
 void Free_Char(Titem s) {
      free(s);
 }

int main(void) {
    Tstack stack;
    Tboolean succeeded;
    char chr;
    char *ch;
```

מימוש ADT Stack

```
initialize_stack(&stack);

printf("\n Enter a letter to push onto stack");
printf("\n Enter 1 to pop a letter from stack");
printf("\n Enter 2 to end the program\n");

chr = getche();
while (chr != '2')
{
    if ( isalpha(chr) )    {            /*if it's letter*/
        ch = (char*)malloc( sizeof(char) );
        if(ch == NULL)
        {
            printf("Memory allocation error");
            free_stack(&stack, Free_Char);
            return 1;
         }
        *ch = chr;
        succeeded = push(&stack, ch);
        if (!succeeded)
            printf("\n Push operation failed\n");
       print_stack(&stack, Print_Char);
 }
```

מימוש ADT Stack

```
if (chr == '1')        /*pop operation*/
{
    succeeded = pop(&stack, &ch);
    if  (succeeded)
     {
        printf("\n Letter popped from stack is %c ", *ch);
        free(ch);
        print_stack(&stack, Print_Char);
     }
    else
         printf("\nPop operation failed\n ");
 }
chr = getche();
}

free_stack(&stack, Free_Char); /*free memory */
return 0;
}
```

שינו קובץ Stack_General_Main.c אם הנתונים הם מבנים ספציפיים ולא משתנה מסוג char

```
/* Stack_General_Main.c file*/

#include "Stack_General.h"

/*spesific structure*/
typedef struct person
{
  int age;
  char *name;
}person;

/*specific function – for struct person only */
void Print_person(Titem s)
{
  printf("\n The age is %d and the name is %s
",((person*)s)->age,((person*)s)->name);
}

/*specific function – for deleting struct person only */
void Free_person(Titem s)
{
  free(((person*)s)->name);
  free(s);
}
```

```
/*specific function – for struct person only */
person* Build_person()
{
    person* p;
    char temp_name[200];
    p = (person*)malloc(sizeof(person));
    if (p == NULL)
        return 0;
    printf("\n Enter an age");
    scanf("%d", &(p->age));
    printf("\n Enter a name");
    scanf("%s", temp_name);
    p->name = (char*)malloc(strlen(temp_name) + 1);
    if (p->name == NULL)
    {
        free(p);
        return 0;
    }
    strcpy(p->name, temp_name);
    return p;
}
```

```
int main(void)
 {
    Tstack stack;
    Tboolean succeeded;
    person *pers;
    char chr;
    initialize_stack(&stack);

    printf("\n Enter 0 to push onto stack");
    printf("\n Enter 1 to pop a person from stack");
    printf("\n Enter 2 to end the program\n");

    chr = _getche();            /*get the option number*/
    while (chr != '2')
    {
        if ( chr == '0' )       /*push*/
        {
            pers = (person*)malloc(sizeof(person));
            if(pers == NULL)
            {
                printf("Memory allocation error");
                free_stack(&stack, Free_person);
                return 1;
            }
```

```
pers = Build_person();
if (pers == 0)
 {
       printf("Memory allocation error");
       free_stack(&stack, Free_person);
       return 1;
  }
succeeded = push(&stack, pers);
 if (succeeded == 0)
       printf("\n Push operation failed\n");
 print_stack(&stack, Print_person);}
if (chr == '1')        /*pop operation*/
{
       succeeded = pop(&stack, &pers);
       if  (succeeded == 1)
       {
        printf("\n %s is popped from stack", pers->name);
         Free_person(pers);
         print_stack(&stack, Print_person);}
        else
                    printf("\nPop operation failed\n "); }
chr = _getche();
}

free_stack(&stack, Free_person);     /*free memory */
return 0;  }
```