Laboratorul 4

Stive şi Cozi

STIVA

O stivă este o instanță a unui tip de date abstract ce formalizează conceptul de colecţie cu acces restricționat. Restricția respectă regula LIFO (Last In, First Out). Accesul la elementele stivei se face doar prin vârful acesteia (head).

```
typedef struct cel
{
   Item elem;
   struct cel* next;
}StackCel, *TStack;
```

Operații:

Push – adaugă un element (entitate) în stivă. Adăugarea se poate face doar la vârful stivei.

Pop – șterge un element din stivă. Ștergerea se poate face doar la vârful stivei.

Top – consultă (întoarce) elementul din vârful stivei fără a efectua nicio modificare asupra acesteia.

IsEmpty – întoarce 1 dacă stiva este goală; 0 dacă are cel puțin un element

COADA

O coadă este o structură de date ce modelează un buffer de tip FIFO (First In, First Out). Astfel, primul element introdus în coadă va fi și primul care va fi scos din coadă.

```
typedef struct cel
{
   Item elem;
   struct cel* next;
}QueueCel, *TQueue;

typedef struct {TQueue front, rear;} AQ;
```

Operații:

Enqueue – adaugă un element (entitate) în coadă. Adăugarea se poate face doar la sfârșitul cozii.

Dequeue – șterge un element din coadă. Ștergerea se poate face doar la începutul cozii.

Front – consultă (întoarce) elementul din capul cozii fără a efectua nicio modificare asupra acesteia.

IsEmpty – întoarce 1 dacă coada este goală; 0 dacă are cel puțin un element

CERINŢE

1. Să se scrie definiţia completă a următoarelor funcţi de lucru cu stiva în fişierul **stack.h**:

TStack InitStack()– iniţializează stiva vidă

int IsEmptyStack (TStack s)– verifică dacă stiva este vidă sau nu

TStack Push(TStack s, Item el)– adaugă un nou element în stivă

*TStack Pop(TStack s,Item *el)*– extrage elementul de la vârful stivei

Item Top (TStack s)– returnează elementul de la vârful stivei

void PrintStack(TStack s)– afişează conţinutul stivei **(0.5 x 6) puncte**

2. Să se scrie definiţia completă a următoarelor funcţii de lucru cu coada în fişierul **queue.h**:

AQ InitQueue()*– iniţializează coada vida

*int IsEmptyQueue(AQ *q)*– verifică dacă coada este vidă sau nu

*void Enqueue(AQ *q, Item el)*– adaugă un nou element în coadă

*Item Dequeue(AQ *q)*– elimină un element din coadă

*Item Front(AQ *q)*– returnează primul element din coadă

*void PrintQueue(AQ *q)*– afişează conţinutul cozii **(0.5 x 6) puncte**

3. În fişierul **Parantheses.c**, să se completeze funcţia *int isBalanced(char s[100])*, care verifică dacă un şir de paranteze (citit în şirul s) se închide perfect.

Exemplu:
Pentru s="((()))" se va afişa mesajul **Balanced**
Pentru s="(()()(" se va afişa mesajul **Not balanced**

(1.5 puncte)

4. Radix Sort

Radix Sort este un algoritm de sortare care ţine cont de cifrele individuale ale elementelor sortate. Aceste elemente pot fi nu doar numere, ci orice altceva ce se poate reprezenta prin întregi. Majoritatea calculatoarelor digitale reprezintă datele în memorie sub formă de numere binare, astfel că procesarea cifrelor din această reprezentare se dovedeşte a fi cea mai convenabilă. Există două tipuri de astfel de sortare: LSD (Least Significant Digit) şi MSD (Most Significant Digit). LSD procesează reprezentările dinspre cea mai puţin semnificativă cifră spre cea mai semnificativă, iar MSD invers.

O versiune simplă a radix sort este cea care foloseşte 10 cozi (câte una pentru fiecare cifră de la 0 la 9). Aceste cozi vor reţine la fiecare pas numerele care au cifra corespunzătoare rangului curent. După această împărţire, elementele se scot din cozi în ordinea crescătoare a indicelui cozii (de la 0 la 9), şi se reţin într-un vector (care devine noua secvenţă de sortat).

Exemplu:

Secvenţa iniţială:

170, 45, 75, 90, 2, 24, 802, 66

Numere sunt introduse în 10 cozi (într-un vector de 10 cozi), în funcţie de cifrele de la dreapta la stânga fiecărui număr.

Cozile pentru prima iteraţie vor fi:

* 0: 170, 090
* 1: nimic
* 2: 002, 802
* 3: nimic
* 4: 024
* 5: 045, 075
* 6: 066
* 7 - 9: nimic

a. Se face dequeue pe toate cozile, în ordinea crescătoare a indexului cozii, şi se pun numerele într-un vector, în ordinea astfel obţinută:

Noua secvenţă de sortat:

170, 090, 002, 802, 024, 045, 075, 066

b. A doua iteraţie:

Cozi:

* 0: 002, 802
* 1: nimic
* 2: 024
* 3: nimic
* 4: 045
* 5: nimic
* 6: 066
* 7: 170, 075
* 8: nimic
* 9: 090

Noua secvenţă:

002, 802, 024, 045, 066, 170, 075, 090

c. A treia iteraţie:

Cozi:

* 0: 002, 024, 045, 066, 075, 090
* 1: 170
* 2 - 7: nimic
* 8: 802
* 9: nimic

Noua secvenţă:

002, 024, 045, 066, 075, 090, 170, 802 (sortată)

Se cere să se completeze funcţia *void RadixSort(int a[MAX_DIM], int n)* din fişierul **RadixSort.c**, care să realizeze sortarea tabloului *a* cu *n* elemente cu ajutorul algoritmului **RadixSort.**

(1.5 puncte)