Stive şi Cozi

1. Obiectivele lucrării

În această lucrare vom prezenta două structuri de date, mai abstracte, stiva şi coada, implementate cu vectori. Acestea reprezintă structuri de date foarte des utilizate în programare, majoritatea sistemelor de calcul punând la dispoziţia programatorului instrucţiuni implementate direct în hardware pentru manipularea stivelor şi cozilor.

2. Breviar teoretic

O ***stivă (stack)*** este o listă liniară ce are proprietatea că operaţiile de **inserare/extragere** a datelor pot fi efectuate **numai la unul din capete**, numit şi **vârful stivei**. Ultimul element inserat va fi primul extras. Din cauza ordinii în care se face introducerea, respectiv extragerea unui element dintr-o stivă aceasta se mai numeşte şi **strucură LIFO** (**L**ast **I**n, **F**irst **O**ut) sau liste *pushdown*. Într-adevăr, articolele sunt extrase dintr-o stivă în ordinea inversă celei ăn care au fost introduse. Deci, într-o stivă, spre deosebire de vectori, avem acces la un moment dat, la un singur element: cel care a fost inserat ultimul.

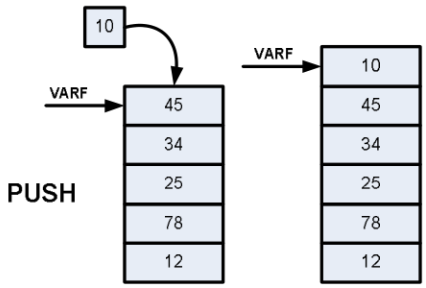
Operaţiile fundamentale asociate acestui tip abstract de date sunt:

initS – crează o stivă vidă;

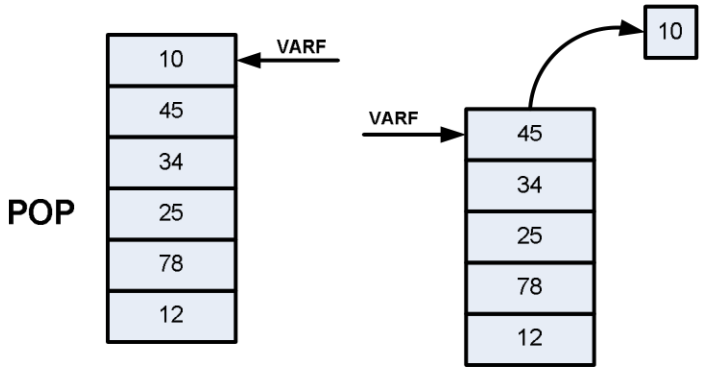
esteVida – testează daca o stiva nu conţine nici un element şi, în acest caz, întoarce adevărat;

estePlina – testează daca întreaga capacitate de memorare a stivei a fost utilizată;

pushS – adaugă un element în vârful stivei;



popS – extrage un element din vârful stivei şi întoarce valoarea acelui element.



Cel mai natural mod de reprezentare pentru o stivă este implementarea secvenţială într-un tablou S[1 … n], unde n este numărul maxim de noduri (elemente). Primul nod va fi memorat în S[1], al doilea în S[2], iar ultimul în S[vf], unde vf este o variabilă care conţine adresa (indicele) ultimului element inserat. Iniţial, când stiva este vidă, avem vf=0.

Algoritmii de inserare şi de extragere (ştergere) a unui element sunt prezentaţi în cele ce urmează.

```
Functie push(x, S[1…n])
{adauga nodul x in stiva}
If vf ≥ n then return "stiva plina"
vf ← vf+1
S[vf] ← x
return "succes"

Functie pop(x, S[1…n])
{sterge ultimul nod inserat din stiva si il
returneaza}
If vf ≤ 0 then return "stiva vida"
x ← S[vf]
vf ← vf-1
return x
```

Cei doi algoritmi necesită timp constant, deci nu depind de mărimea stivei.

O ***coadă (queue)*** este o listă liniară ce are proprietatea că operaţiile de inserare a nodurilor se fac pe la un cap, iar extragerile pe

la celălalt cap. Primul nod inserat va fi primul nod extras. De aceea cozile se mai numesc şi structuri **FIFO** (**F**irst **I**n, **F**irst **O**ut), elementul care este extras din coadă este întotdeauna cel care a stat cel mai mult în coadă. Structura de coadă modelează un rând de aşteptare pentru bilete la un spectacol. Prima persoană din rând, este prima care ajunge să cumpere bilete.

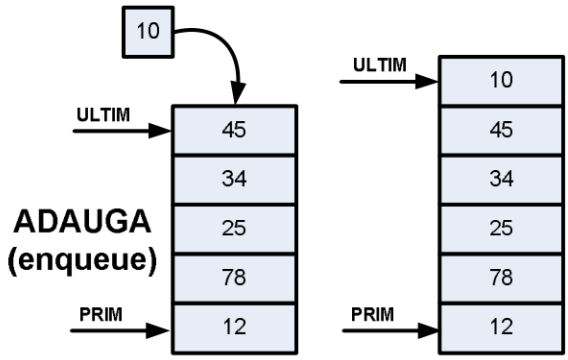
Operaţiile fundamentale asociate tipului abstract de date coadă sunt:

initC – crează o coadă vidă;

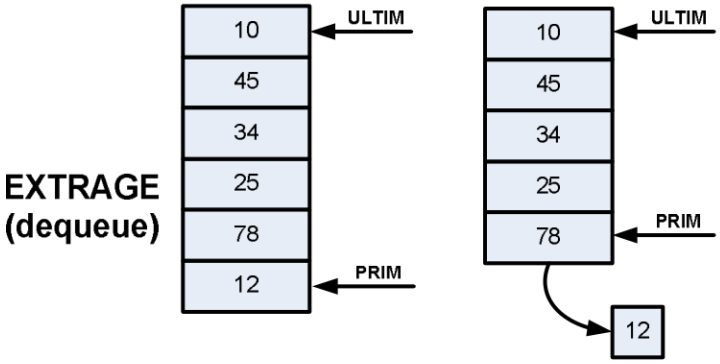
esteVida – testează daca o coadă este vidă sau nu;

estePlina – testează daca o coadă este plină (întreaga capacitate de memorare a cozii a fost utilizată) şi, în acest caz, întoarce adevărat;

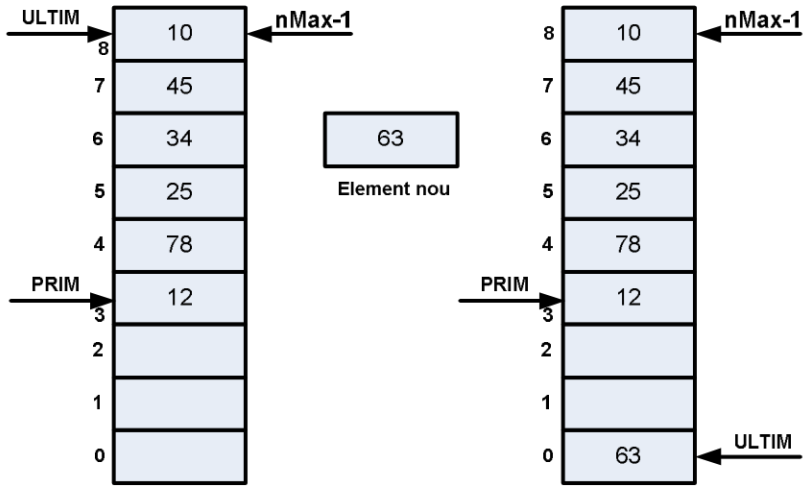
adaugaC – adaugă un element la o coadă. Uneori această operaţie este denumită şi ***enqueue*** (abreviere de la ***enter in queue***);



extrageC – extrage un element din coadă şi întoarce valoarea acelui element. Uneori această operaţie este denumită şi ***dequeue*** (abreviere de la ***delete from queue***).



Un caz particular de cozi sunt cozile circulare ce se implementează folosind un vector C de dimensiune n, pe care îl tratăm ca şi cum ar fi circular: după locaţia C[n-1] urmează locaţia 0.



3. Probleme rezolvate

Se vor edita şi apoi executa programele descrise în continuare.

1. Implementarea cu variabile globale a structurii de date de tip stivă. Nu se verifică în funcţiile specifice stivei (în funcţia push(), respectiv în funcţia pop()), dacă aceasta este plină (nu se mai pot adăuga elemente), sau dacă aceasta este vidă (nu se mai pot extrage elemente). Aceste verificări sunt lăsate în responsabilitatea programului principal, ce foloseşte funcţiile stivei.

Sursa programului:

```
#include <stdio.h>
#include <conio.h>
//prototipuri functii:
void init(int nrEl);
void push(int n);//funcţia de inserare în stivă
int pop();//funcţia de extragere element din vârful
//stivei
int esteVida();
int estePlina();
int peek();//citirea elementului din vârful stivei
void afisare();//afişarea tuturor elementelor din
//stivă
```

```
//Variabile globale prin care este descrisa stiva:
int s[1000];//o dimensiune acoperitoare
int vf;//indexul ultimui element introdus
int nMax;//numar maxim de elemente
```

```
void main()
{
      int nr;
      clrscr();
      init(100);//maxim 100 de elemente
      push(4);
      push(3);
      push(5);
      nr=pop();
      printf("S-a extras elementul: %d",nr);
      //afisare toata stiva:
      afisare();
      getch();
}

void init(int nrEl)
{
      nMax=nrEl;
      vf=-1;
}

void push(int n)
{
      vf=vf+1;
      s[vf]=n;
}

int pop()
{
      int nr=s[vf];
      vf=vf-1;
      return(nr);
}

int esteVida()
{
      if (vf==-1) return 1;
      else return 0;
}
```

```
int estePlina()
{
      if (vf==nMax) return 1;
      else return 0;
}

int peek()
{
      return(s[vf]);
}

void afisare()
{
      for(;;)
      {
            if (esteVida()) break;
            printf("\n%d",pop());
      }
}
```

2. Implementare stivă, fără variabile globale.

Sursa programului:

```
#include <stdio.h>
#include <conio.h>
//prototipuri functii:
void init(int& nMax,int& vf, int nrEl);
void push(int s[], int& vf, int n);
int pop(int s[], int& vf);
int esteVida(int vf);
int estePlina(int nMax, int vf);

void main()
{
      //variabilele prin care este descrisa stiva:
      int s[1000];//o dimensiune acoperitoare
      int vf;
      int nMax;//numar maxim de elemente
      //alte variabile din program:
      int nr;
      clrscr();
      init(nMax,vf,100);//maxim 100 de elemente
      push(s,vf,4);
      push(s,vf,3);
      push(s,vf,5);
      nr=pop(s,vf);
      printf("S-a extras elementul: %d",nr);
```

```
	getch();
}
```

```
void init(int& nMax,int& vf, int nrEl)
{
	nMax=nrEl;
	vf=-1;
}

void push(int s[], int& vf, int n)
{
	vf=vf+1;
	s[vf]=n;
}

int pop(int s[], int& vf)
{
	int nr=s[vf];
	vf=vf-1;
	return(nr);
}

int esteVida(int vf)
{
	if (vf==-1) return 1;
	else return 0;
}

int estePlina(int nMax, int vf)
{
	if (vf==nMax) return 1;
	else return 0;
}
```

3. Implementarea cu variabile globale a structurii de coadă (liniară).

Observaţie:

Adăugari şi ştergeri repetate în coadă deplasează conţinutul cozii la dreapta, faţă de începutul vectorului. Pentru a evita acest lucru ar trebui ca funcţia *extrage()* să deplaseze la stânga conţinutul cozii cu o poziţie. Primul element care urmeaza să fie scos va fi întotdeauna în prima pozitie, indicatorul *prim* pierzându-şi utilitatea.

Sursa programului:

```
#include <stdio.h>
#include <conio.h>
#include<iostream.h>
```

```
//prototipuri:
void init(int dim); //functia de initializare a cozii
void adauga(int nr);
int extrage();
int estePlina();
int esteVida();
void afisare();
//variabile globale:
int C[1000];
int nMax;//nr. maxim de elemente
int prim;//index prim element ce se va scoate
int ultim;//index ultim element asezat la coada
int nrElem;//nr. elemente aflate in coada

void main()
{
	int x;
	clrscr();
	init(5);
	if(estePlina())
		cout<<"Coada este plina";
	else
	{
		cout<<"x=";
		cin>>x;
		adauga(x);
	}
	if(esteVida())
		cout<<"Coada este vida";
	else
		cout<<"Am extras: " <<extrage();
	afisare();
	getch();
}

void init(int dim)
{
	nMax=dim;
	prim=0;
	ultim=-1;
	nrElem=0;
}

void adauga(int nr)
{
	ultim++;
	C[ultim]=nr;
	nrElem++;
}
```

```
int extrage()
{
	int nr=C[prim];
	prim++;
	nrElem--;
	return nr;
}

int estePlina()
{
	if(nrElem==nMax)
		return 1;
	return 0;
}

int esteVida()
{
	if(nrElem==0)
		return 1;
	return 0;
}

void afisare()
{
	int i;
	if(nrElem==0)
	{
		printf("\nCoada este vida.");
		return;
	}
	for(i=prim;i<=ultim;i++)
		printf("\n%d",C[i]);
}
```

4. Implementarea cu variabile globale a structurii de coadă circulară.

Sursa programului:

```
#include <stdio.h>
#include <conio.h>
//prototipuri:
void init(int dim); //functia de initializare a cozii
void adauga(int nr);
int extrage();
int estePlina();
int esteVida();
void afisare();
//variabile globale:
int C[1000];
```

```
int nMax;//nr. maxim de elemente
int prim;//index prim element ce se va scoate
int ultim;//index ultim element asezat la coada
int nrElem;//nr. elemente aflate in coada

void main()
{
	int i;
	init(10);
	//adauga 5 elemente:
	//adauga 5 elemente:
	for(i=1;i<=5;i++)adauga(i);
	//scoate pe primele 3:
	printf("Primele 3 numere extrase din coada:\n");
	for(i=1;i<=3;i++)
		printf("%d\n",extrage());
	printf("Au mai ramas in coada:");
	afisare();
	getch();
}

void init(int dim)
{
	nMax=dim;
	prim=0;
	ultim=-1;
	nrElem=0;
}

//Functia adauga(). Nu verifica daca este plina deja!
//Aceasta verificare se face inaintea apelarii metodei.
void adauga(int nr)
{
	ultim++;
	if(ultim==nMax)ultim=0;
	C[ultim]=nr;
	nrElem++;
}

//Functia extrage(). Nu verifica daca are ce extrage!
int extrage()
{
	int nr=C[prim];
	prim++;
	if(prim==nMax)prim=0;
	nrElem--;
	return nr;
}
```

```
int estePlina()
{
	if(nrElem==nMax)return 1;
	else return 0;
}

int esteVida()
{
	if(nrElem==0)return 1;
	else return 0;
}

void afisare()
{
	int i;
	if(nrElem==0){
		printf("\nEste vida.");
		return;}
	if(ultim>=prim)
		for(i=prim;i<=ultim;i++)printf("\n%d",C[i]);
	else{
		for(i=prim;i<nMax;i++)printf("\n%d",C[i]);
		for(i=0;i<=ultim;i++)printf("\n%d",C[i]);}
}
```

4. Probleme propuse

1. *Folosind structura de stivă, să se construiască un vector B, obţinut prin copierea în ordine inversă a unui vector A, dat.*
2. *Să se implementeze structura de coadă liniară, fără folosirea variabilelor globale şi cu deplasarea, la fiecare extragere,* conţinutului cozii cu o poziţie la stânga.
3. *Să se implementeze structura de coadă circulară, fără folosirea variabilelor globale.*