

Structuri de Date

Lucrarea de laborator nr. 5

17 martie 2020

ș.l.dr.ing. Corina Cîmpanu

Cuprins

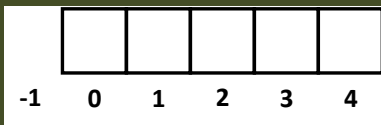
1. Stiva ordonată
2. Stiva înlănțuită
3. Forma poloneză și evaluarea unei expresii aritmetice
4. Probleme propuse

Stiva - Generalități

- **Mecanism de organizare**
 - LIFO** – Last In First Out
 - FILO** – First In Last Out
- **Operații de bază**
 - InitStack – inițializează stiva
 - IsEmpty – verifică dacă stiva este goală
 - IsFull – verifică dacă stiva este plină
 - Push – adaugă un element în stivă
 - Pop – extrage un element din stivă
 - Top – listează elementul din vârful stivei

1. Stiva ordonată

1.1. Structura



```

Main.cpp  Functii.cpp  Header.h  Stack_static
Stack_static
#pragma once

#include <iostream>
using namespace std;

#define DIMMAX 5

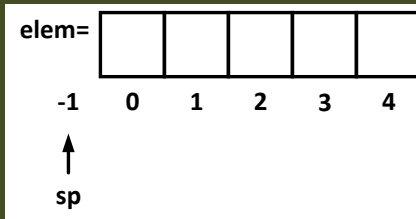
struct Stack {
    int sp;
    int* elem;
};

void InitStack(Stack& s);
bool IsEmpty(Stack s);
bool IsFull(Stack s);
bool Push(Stack& s, int val);
bool Pop(Stack& s, int& val);
bool Top(Stack s, int& val);

```

1. Stiva ordonată

1.2. Inițializarea



```
void InitStack(Stack& s)
{
    cout << "\n Initializarea stivei...";
    // initializam varful stivei sp cu -1

    //alocam memorie pentru elemente in elem cu maxim DIMMAX elemente
}
```

```
Main.cpp  Funcții.cpp  Header.h
Stack_static (Global Scope)
#include "Header.h"

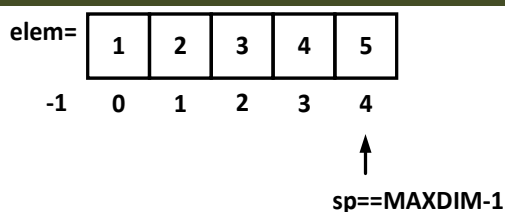
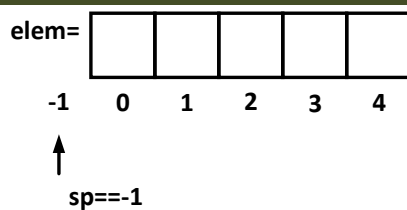
int main()
{
    Stack s;
    int val;
    bool ok;

    InitStack(s);
}
```

1. Stiva ordonată

1.3. Verificarea IsEmpty

1.4. Verificarea IsFull



```
Main.cpp  Funcții.cpp  Header.h
Stack_static (Global Scope)
[ ]

bool IsEmpty(Stack s)
{
    cout << "\n Verificare stiva goala...";
    // stiva este goala cand varful stivei sp este setat pe -1
    return true;
}

bool IsFull(Stack s)
{
    cout << "\n Verificare stiva plina ...";
    //stiva este plina cand varful stivei sp este setat pe DIMMAX-1
    return true;
}

cout << "\n IsEmpty(s): " << IsEmpty(s);
cout << "\n IsFull(s): " << IsFull(s);
```

1. Stiva ordonată

1.5. Push

```
bool Push(Stack& s, int val)
{
    cout << "\n Stiva sp = " << s.sp << " -> Push(" << val << ")...";
    //daca stiva este plina voi returna false

    //altfel incrementez varful stivei sp
    //pe pozitia indicata de sp in vectorul elem voi adauga valoarea val
    //returnez true - operatie efectuata cu succes
    return true;
}
```

```
cout << "\n Push: \n";
for (int i = 0; i < DIMMAX + 2; i++)
{
    ok = Push(s, i + 1);
    cout << "\n Pasul " << i << " : push status = " << ok;
}
```

1. Stiva ordonată

1.6. Pop

```
bool Pop(Stack& s, int& val)
{
    cout << "\n Stiva sp = " << s.sp << " -> Pop()...";
    //daca stiva este goala returnez false
    //acest false imi va indica faptul ca nu am nici o valoare utila in variabila val
    // atentie, transmit prin referinta val pentru a putea stoca valoarea extrasa

    //altfel, extrag valoarea din vectorul elem de pe pozitia indicata de sp si o stochez in val
    //decrementez sp
    //returnez true
    return true;
}
```

```
cout << "\n Pop: \n";
for (int i = 0; i < DIMMAX + 2; i++)
{
    ok = Pop(s, val);
    cout << "\n Pasul " << i << " : pop status = " << ok;
    if (ok)
        cout << " val = " << val;
}
```

1. Stiva ordonată

1.7. Top

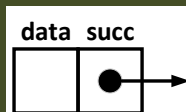
```
bool Top(Stack s, int& val)
{
    cout << "\n Stiva sp = " << s.sp << " -> Top()...";
    //daca stiva este goala returnez false
    //acest false imi va indica faptul ca nu am nici o valoare utila in variabila val
    // atentie, transmit prin referinta val pentru a putea stoca valoarea extrasa

    //altfel, extrag valoarea din vectorul elem de pe pozitia indicata de sp si o stochez in val
    //nu modific sp
    //returnez true
    return true;
}
```

```
cout << "\n Top: " << Top(s, val);
if (Top(s, val))
    cout << " " << val << endl;
```

2. Stiva înlănțuită

2.1. Structura



```
Main.cpp  Functii.cpp  Header.h  Elem
Stack_static

#pragma once

#include <iostream>
using namespace std;

/* ... */

struct Elem {
    int data;
    Elem* succ;
};

void InitStack(Elem*& sp);
void IsEmpty(Elem* sp);
bool Push(Elem*& sp, int val);
bool Pop(Elem*& sp, int& val);
bool Top(Elem* sp, int& val);
```

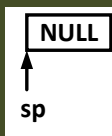
2. Stiva înlănțuită

2.2. Inițializarea

2.3. Verificarea IsEmpty

```
void InitStack(Elem*& sp)
{
    // initializeaza sp, varful stivei cu NULL
}
```

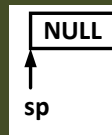
```
InitStack(sp);
```



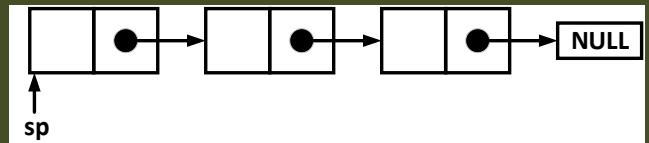
True

```
bool IsEmpty(Elem* sp)
{
    // verifica daca sp indica catre NULL
    return true;
}
```

```
cout << "\n IsEmpty(s): " << IsEmpty(sp);
```



False



2. Stiva înlănțuită

2.4. Push

```
bool Push(Elem*& sp, int val)
{
    // inserare in fata unei liste simplu inlantuite cu capul indicat de sp
    return true;
}
```

```
cout << "\n Push:";
for (int i = 0; i < 5; i++)
    cout << "\n Pasul " << i << " : Push(" << i + 1 << " ) : status = " << Push(sp, i + 1);
```

2. Stiva înlănțuită

2.5. Pop

```
bool Pop(Elem*& sp, int& val)
{
    //daca stiva este goala returneaza false si val ramane neinitializat cu ceva util

    //altfel va stoca in val valoarea din elementul indicat de sp
    // dupa care va sterge capul listei simplu inlantuite indicate de sp
    // si va returna true
    return true;
}
```

```
cout << "\n Pop:";
for (int i = 0; i < 7; i++)
{
    ok = Pop(sp, val);
    cout << "\n Pasul " << i << " : pop status = " << ok;
    if (ok)
        cout << " val = " << val;
}
```

2. Stiva înlănțuită

2.6. Top

```
bool Top(Elem* sp, int& val)
{
    //daca stiva este goala returneaza false si val ramane neinitializat cu ceva util

    //altfel va stoca in val valoarea din elementul indicat de sp
    // si va returna true
    return true;
}
```

```
cout << "\n Top: " << Top(sp, val);
if (Top(sp, val))
    cout << " " << val << endl;
```

2. Stiva înlănțuită

Main test

```

Main.cpp  Functii.cpp  Header.h
Stack_static  (Global Scope)  main()

*/
Elem* sp;
int val;
bool ok;

InitStack(sp);

cout << "\n IsEmpty(s): " << IsEmpty(sp);

cout << "\n Push:";
for (int i = 0; i < 5; i++)
    cout << "\n Pasul " << i << " : Push(" << i + 1 << " ) : status = " << Push(sp, i + 1);

cout << "\n IsEmpty(s): " << IsEmpty(sp);

cout << "\n Top: " << Top(sp, val);
if (Top(sp, val))
    cout << " " << val << endl;

cout << "\n Pop:";
for (int i = 0; i < 7; i++)
{
    ok = Pop(sp, val);
    cout << "\n Pasul " << i << " : pop status = " << ok;
    if (ok)
        cout << " val = " << val;
}

cout << "\n IsEmpty(s): " << IsEmpty(sp);

system("PAUSE");
return 0;
}

```

3. Forma poloneză a unei expresii aritmetice

Fie expresia aritmetică: $\text{expr} = a * b + c * d / (e - f)$

$\text{expr} = 1 * 3 + 2 * 9 / (5 - 2)$

Operatori: +, -, *, /, (,)

Operanzi: a, ..., f sau 1, 2, ..., 9, 0

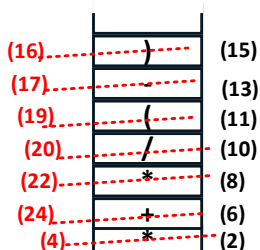
Pentru transformarea din forma infixată în forma postfixată vom folosi o stivă de operatori

Pentru evaluarea expresiei vom folosi o stivă de operanzi

3. Forma poloneză a unei expresii aritmetice

3.1. Transformarea în forma postfixată (1)

expr = a*b+c*d/(e-f) Stiva de operatori



(1)	(3)	(5)	(7)	(9)	(12)	(14)	(18)	(21)	(23)	(25)	(26)
a	b	*	c	d	e	f	-	/	*	+	\0
0	1	2	3	4	5	6	7	8	9	10	11

Cât timp am caractere valide în expr
sau caracterul de pe poziția 0 mai exact a
a este operand

(1) Pun a în forma postfixată

sau caracterul de pe poziția următoare *

* este operator

(2) Pun * în stivă

(3) Pun b în forma postfixată

+ este operator și are prioritate mai mică
decât elementul * din vârful stivei

(4) Scot * din stivă

(5) Pun + în forma postfixată

(6) Pun + în stivă

(7) Pun c în forma postfixată

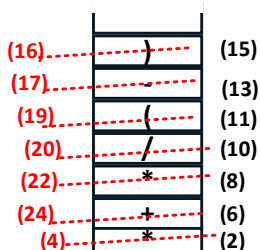
* este operator și are prioritate mai mare
decât elementul + din vârful stivei

(8) Pun * în stivă

3. Forma poloneză a unei expresii aritmetice

3.1. Transformarea în forma postfixată (2)

expr = a*b+c*d/(e-f)



(1)	(3)	(5)	(7)	(9)	(12)	(14)	(18)	(21)	(23)	(25)	(26)
a	b	*	c	d	e	f	-	/	*	+	\0
0	1	2	3	4	5	6	7	8	9	10	11

(9) Pun d în forma postfixată

/ este operator și are prioritate egală cu *

(10) Pun / în stivă

(11) Pun (în stivă

(12) Pun e în forma postfixată

(13) Pun - în stivă

(14) Pun f în forma postfixată

(15) Pun) în stivă

(16) Extrag) din stivă

(17) Extrag - din stivă

(18) Pun - în forma postfixată

(19) Extrag (din stivă

Nu mai am nici un caracter valid în
expresie (am atins terminatorul de șir '\0').

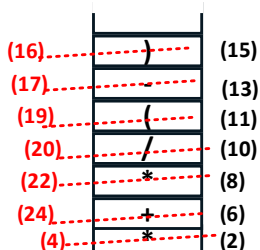
Scot tot ce am în stivă și adaug în forma
postfixată

(20) Extrag / din stivă

3. Forma poloneză a unei expresii aritmetice

3.1. Transformarea în forma postfixată (3)

expr = $a*b+c*d/(e-f)$



(21) Adaug / în forma postfixată
 (22) Extrag * din stivă
 (23) Adaug * în forma postfixată
 (24) Extrag + din stivă
 (25) Adaug + în forma postfixată
 Nu mai am elemente în stivă.
 Adaug terminatorul de șir '\0' în forma postfixată

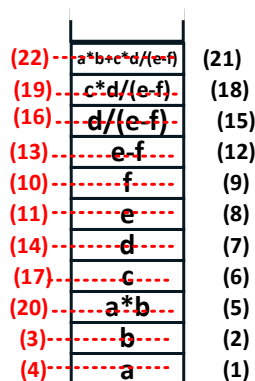
(1)	(3)	(5)	(7)	(9)	(12)	(14)	(18)	(21)	(23)	(25)	(26)
a	b	*	c	d	e	f	-	/	*	+	\0
0	1	2	3	4	5	6	7	8	9	10	11

3. Forma poloneză a unei expresii aritmetice

3.2. Evaluarea unei expresii în forma postfixată (1)

expr = $ab*cdef-/*+$ Stiva de operanzi

a	b	*	c	d	e	f	-	/	*	+	\0
0	1	2	3	4	5	6	7	8	9	10	11



Parcure expresia în forma postfixată până la întâlnirea terminatorului de sir. Adaug operandii în stivă. La întâlnirea unui operator extrag câte 2 operanzi din stivă, fac calculele și depun rezultatul pe stivă.
 a este operand
 (1) Adaug a în stivă
 (2) Adaug b în stivă
 * este operator
 (3) Extrag b
 (4) Extrag a
 $Op1=a, Op2=b, Rez=Op1*Op2=a*b$
 (5) Adaug $a*b$ în stivă
 (6)-(9) Adaug c,d,e,f în stivă
 Întâlnesc -
 (10)-(11) Extrag f, e

3. Forma poloneză a unei expresii aritmetice

3.2. Evaluarea unei expresii în forma postfixată (2)

expr = ab*cdef-//*+ Stiva de operanzi

a	b	*	c	d	e	f	-	/	*	+	\0
0	1	2	3	4	5	6	7	8	9	10	11

(22)	a*b+c*d/(e-f)	(21)
(19)	c*d/(e-f)	(18)
(16)	d/(e-f)	(15)
(13)	e-f	(12)
(10)	f	(9)
(11)	e	(8)
(14)	d	(7)
(17)	c	(6)
(20)	a*b	(5)
(3)	b	(2)
(4)	a	(1)

Calculez e-f

(12) Adaug e-f în stivă

Întâlnesc / în șir

(13), (14) Extrag e-f și d

Calculez d/(e-f)

(15) Adaug d/(e-f) în stivă

Întâlnesc * în șir

(16), (17) Extrag d/(e-f) și c

Calculez c*d/(e-f)

(18) Adaug c*d/(e-f) în stivă

Întâlnesc + în șir

(19), (20) Extrag c*d/(e-f) și a*b

Calculez suma lor

(21) Adaug a*b + c*d/(e-f) în stivă

Întâlnesc '\0' în șir

(22) Extrag a*b+c*d/(e-f) din stivă, ca fiind rezultatul expresiei inițiale

4. Probleme propuse

- Stiva ordonată – implementare
- Stiva înlănțuită – implementare
- Forma poloneză a unei expresii aritmetice
- Evaluarea unei expresii aritmetice plecând de la forma poloneză
- Transformarea unei expresii din forma postfixată în forma infixată și evaluarea ei