# ALGORITMI I STRUKTURE PODATAKA

RAČUNSKE VEŽBE - TERMIN BR. 6 - 06.04.2020. - STEK

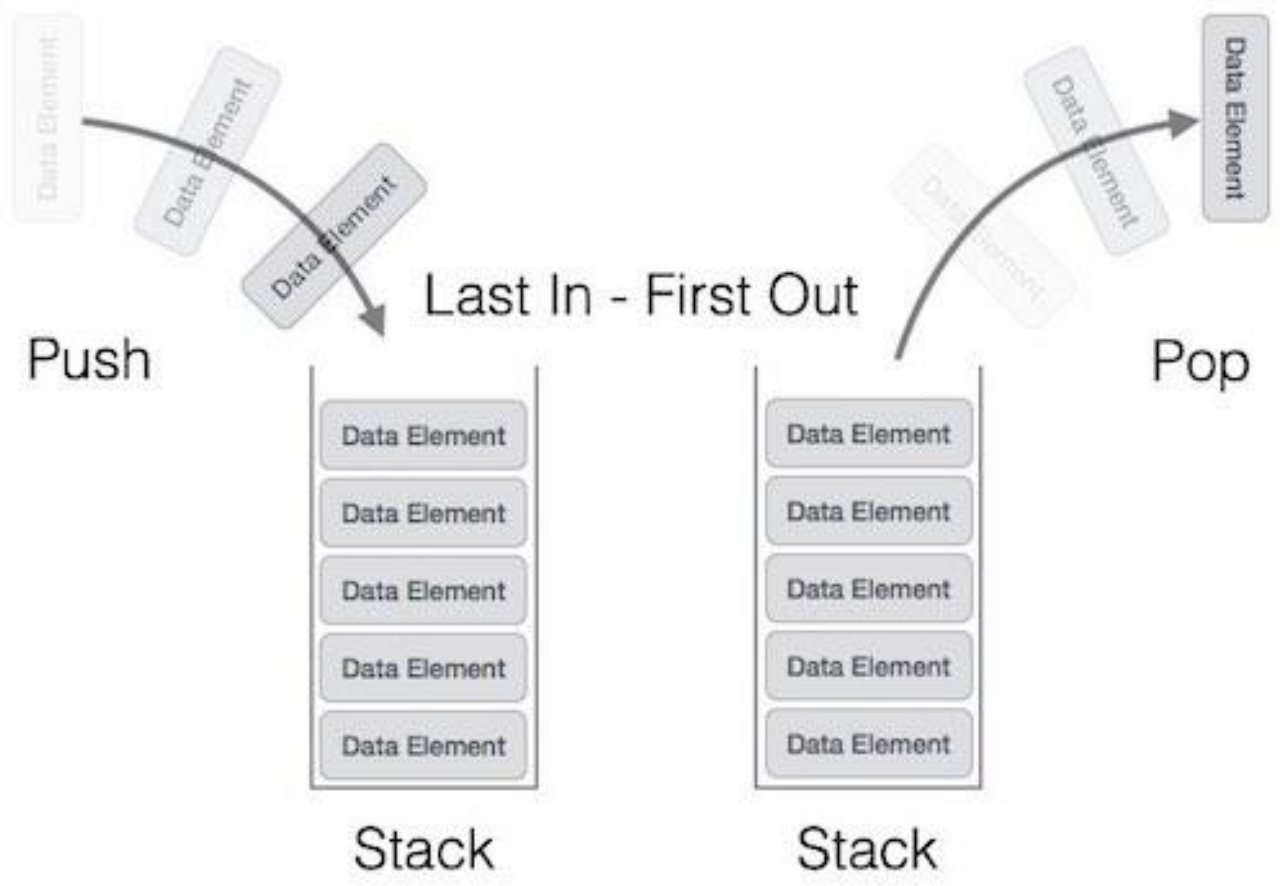
ALDINA AVDIĆ, DIPL. INŽ. - apljaskovic@np.ac.rs

RAČUNARSKA TEHNIKA; SOFTVERSKO INŽENJERSTVO; INFORMATIKA I MATEMATIKA

# Stek

- Stek je LIFO struktura podataka
- LIFO – Last In First Out
- Znači da element koji je poslednji dodat u strukturu prvi se briše
- Zamislite jednu kutiju u koju slažete knjige. Možete pristupiti samo knjizi sa vrha, tj. onoj koju ste poslednju ubacili u kutiju,
- Stoga stek ima glavni pokazivač za vrh steka TOP
- Ne možete, kao kod niza, pristupati elementima sa sredine
- Stek može da se implementira na dva načina:
  - Preko niza
  - Preko lančane liste

# Stek


Data Element
Data Element
Data Element
Last In - First Out
Push
Pop
Data Element
Data Element
Data Element
Data Element
Data Element
Stack
Data Element
Data Element
Data Element
Data Element
Data Element
Stack

# Operacije sa stekom

- Glavna osobina Steka je već rečena, pristupa se samo elementru sa vrha steka
- Stek karakterišu dve osnovne operacije
  - PUSH
  - POP
- Push služi za dodavanje elementa na vrh steka, ako stek nije pun
- Pop služi za brisanje elementa sa vrha steka, ako stek nije prazan

# Implementacija preko niza

## Ispitivanje da li je stek pun

```
#define SIZE 5                  /* Size of
Stack */
int s[SIZE],top=-1;             /* Global
declarations */


int Sfull()
{                               /* Function to
Check Stack Full */
    if(top == SIZE-1) return 1;
    return 0;
}
```

- Da bismo implementirali stek preko niza, potreban nam je niz i jedna promenljiva koja označava indeks elementa koji je na vrhu steka TOP
- Pošto Push može da se radi jedino ako stek nije pun, moramo napraviti funkciju koja proverava da li je TOP = SIZE - 1

# Implementacija preko niza

## Ispitivanje da li je stek prazan

```
 int Sempty()
{                              /*
Function to Check
Stack Empty */
    if(top == -1)
return 1;
    return 0;
}
```

- Pošto se iz steka može brisati jedino ako u njemu ima elemenata, treba nam jedna funkcija koja proverava da li je stek prazan, da li je TOP == -1

# Operacija Push

## Push

```
void push(int elem)
{                    /* Function for PUSH operation */
    if( Sfull()) printf("\n\n Overflow!!!!\n\n");
    else
    {
        ++top;
        s[top]=elem;
    }
}
```

- Operacija push ima jedan element, taj element koji treba da se upiše u stek
- Pre nego što upišemo element moramo proveriti da li u steku ima mesta
- Ako nema pišemo Prekoračenje
- Ako ima, inkrementiramo top
- I na novu poziciju upisujemo zadatati element

# Operacija Pop

- Operacijom Pop se briše element s vrha steka, tj. pokazivač top se pomera za jednu poziciju ispod, i kad bude vršeno novo dodavanje (Push) ono će ići preko tog elementa koji je obrisan
- Brisanje je moguće jedino ako imamo šta da obrišemo

## Pop

```
int pop()
{                              /* Function for POP
operation */
    int elem;

if(Sempty()){ printf("\n\nUnderflow!!!!\n\n");
    return(-1); }
    else
    {
        elem=s[top];
        top--;
        return(elem); }}
```

# Prikazivanje (štampanje) elemenata

```
display()
{                       /* Function
to display status of Stack */
    int i;
    if(Sempty()) printf(" \n
Empty Stack\n");
    else
    {
        for(i=0;i<=top;i++)
            printf("%d\n",s[i]);
        printf("^Top");}}
```

- Pošto je ovo implementacija preko niza, tačno znako koliko ima elemenata, pa možemo koristiti for petlju
- Prikazivanje elemenata takođe vršimo ako imamo šta da prikažemo, inače štampamo informaciju da je stek prazan

# Main funkcija

```
main()
{                              /* Main Program */
    int opn,elem;
    do
    {
        clrscr();
        printf("\n ### Stack Operations ### \n\n");
        printf("\n Press 1-Push, 2-Pop,3-Display,4-Exit\n");
        printf("\n Your option ? ");
        scanf("%d",&opn);
        switch(opn)
        {
        case 1: printf("\n\nRead the element to be pushed ?");
            scanf("%d",&elem);
            push(elem); break;
        case 2: elem=pop();
            if( elem != -1)
                printf("\n\nPopped Element is %d \n",elem);
```

# Implementacija preko lančane liste

```
#include <stdlib.h>
typedef struct node
{
    int data;
    struct node *link;
}NODE;

void Push(int);
int  pop();
void Display();
NODE *top=NULL;
```

- Stek se implementira kao jednostruka lančana lista
- Dovoljne su dve operacije Push i Pop, a Sfull i Sempty se mogu implementirati u okviru njih
- I naravno Display
- Top kada je stek prazan je NULL

# Operacija Push (dodavanje elementa)

```
void Push(int info)
{
    NODE *temp;
    temp=(NODE *)malloc(sizeof(NODE));
    if( temp == NULL)
        printf(" Out of Memory !! Overflow !!!");
else
    {
        temp->data=info;
        temp->link=top;
        top=temp;
        printf(" Node has been inserted at Top(Front) Successfully !!");
    }
}
```

- Kod operacije Push kod implementacije preko lančane liste, dinamički se kreira jedan element
- Ako to kreiranje nije uspelo znači da nema memorije, i to je u stvari kao da pitamo da li je stek pun
- Ako ima mesta u steku, onda u temp element upisujemo argument, i njegov sledbenim postaje trenutni top
- Zatim sam temp postaje vrh steka top
- I štampamo poruku da je insert dobro izvršen

# Operacija Pop (uklanjanje elementa)

```
int Pop()
{
    int info;
    NODE *t;
    if( top ==  NULL)
{
 printf(" Underflow!!!");
return -1;
 }
  else
    {
        t=top;
        info=top->data;
        top=top->link;
        t->link=NULL;
        free(t);
        return(info);
    }
}
```

- Operacija Pop nema argumente, jer ona izbacuje ono što je na vrhu steka
- Inicijalizujemo jedan pokazivač t da bismo mogli osloboditi memoriju
- Možemo brisati ako elemenata u steku ima
- Ako nema, štampamo informaciju o potkoračenju
- Ako ima šta da se obriše, onda element s vrha steka smeštamo u t, njegov data deo u info.
- Top postaje njegov sledbenik
- Raskida se veza između prethodnog topa i novog
- Oslobađa se čvor t u kom je smešten prethodni top
- Vraća se vrednost info da bismo mogli odštampati šta smo obrisali

# Prikaz (štampanje) elemenata

```
void Display()
{
    NODE *t;
    if( top == NULL) printf("Empty Stack\n");
    else
    {
        t=top;
        printf("Top->");

        while(t)
        {
            printf("[%d]->",t->data);
            t=t->link;
```

# Dva steka preko jednog niza

## SFull

```
#define SIZE 10
/* Size of Stack */

int s[SIZE],top[3]={0,-
1,SIZE};

int Sfull()

{                          /*
Function to Check Stack
Full */

    if(top[1] == top[2]-
1) return 1;

    return 0; }
```

- Možemo u jedan niz da smestimo dva steka, tako što će jedan da se puni s početka, drugi s kraja, i stek će biti pun ako top prvog i top drugog steka budu jedan do drugog
- Ovaj niz topova nek vas ne buni, top [0] ne služi ničemu, top[1] označava top prvog steka, a top[2] top drugog steka
- Sasvim je u redu i da programirate bez ovog niza a da imate promenljive top1 i top2

# Dva steka preko jednog niza

# Dva steka preko jednog niza

- Ovde za empty mora da se prosledi broj steka, pa prvi stek je prazan ako je njegov top -1, a drugi stek je prazan ako je njegov top jednak veličini niza

SEmpty

```
int Sempty(stno)
{
    /* Function to Check Stack Empty */
    switch(stno)
    {
    case 1: if(top[1] == -1) return 1; else
return 0;
    case 2: if(top[2] == SIZE) return 1;else
return 0;
    }}
```

# Dva steka preko jednog niza

## Push

```
push(int elem,int stno)
{ int pos;   if( Sfull())
printf("\n\n
Overflow!!!!\n\n");
    else {
        if(stno==1) pos=
++top[stno];
        else pos=--
top[stno];
        s[pos]=elem; }}
```

- Kod Push moramo da damo kao argument i u koji stek ubacujemo element
- U zavisnosti od toga ako je prvi stek, njegov top se povećava, jer ovaj stek „raste“ ka sredini niza
- Ako ubacujemo u drugi stek, njegov top se smanjuje

# Dva steka preko jednog niza

- I za brisanje prosleđujemo broj steka iz kog brišemo
- Brišemo ako ima šta da se obriše
- U zavisnosti šta smo prosledili kao argument, odgvarajući top se ažurira

## Pop

```
int pop(int stno)
{ int elem,pos;

if(Sempty(stno))
{ printf("\n\nUnderflow!!!!\n\n");
    return(-1); } else {
        pos=top[stno];
        elem=s[pos];
        if(stno == 1)top[stno]--;
        else top[stno]++;
        return(elem);}}
```

# Dva steka preko jednog niza

## Display

```
display(int stno)
{ int i;
    if(Sempty(stno))
printf(" \n Empty
Stack\n");
    else {
        if(stno == 1)
        {

for(i=0;i<=top[stno];i++)
```

## ...Main na moodle-u

```
printf("%d\n",s[i]);
            printf("^Top");
        }
        else
        {
            for(i=SIZE-1;i>=top[stno];i--)
                printf("%d\n",s[i]);
            printf("^Top"); }}}
```

# Infiksna u postfiksnu notaciju

- Za prevođenje izraza iz infiksne notacije u postfiksne može se koristiti stek
- Obrađuje se znak po znak po redu
- Ako se radi o operandu on ide na izlaz
- Ako se radi o operatoru on ide u stek
- I otvorena zagrada ide na stek, zatvorena je znak za pop sve do otvorene
- Na steku ne može da bude operator višeg prioriteta, a za njim nižeg. Znači ako je na steku +, a dođe *, onda puta ide u stek, ali da je bilo obrnuto, puta bi izašlo iz steka a došao bi plus

# Infiksna u postfiksnu notaciju

A trace of the algorithm that converts the infix expression *a* - (*b* + *c* * *d*)/*e* to postfix form

| ch | stack (bottom to top) | postfixExp | |
|---|---|---|---|
| a | | a | |
| – | – | a | |
| ( | – ( | a | |
| b | – ( | ab | |
| + | – ( + | ab | |
| c | – ( + | abc | |
| * | – ( + * | abc | |
| d | – ( + * | abcd | |
| ) | – ( + | abcd* | Move operators |
| | – ( | abcd*+ | from stack to |
| | – | abcd*+ | **postfixExp** until " ( " |
| / | –/ | abcd*+ | |
| e | –/ | abcd*+e | Copy operators from |
| | | abcd*+e/– | stack to **postfixExp** |

# Infiksna u postfiksnu notaciju

Suppose we want to convert 2*3/(2-1)+5*3 into Postfix form,

| Expression | Stack | Output |
|---|---|---|
| 2 | Empty | 2 |
| * | * | 2 |
| 3 | * | 23 |
| / | / | 23* |
| ( | /( | 23* |
| 2 | /( | 23*2 |
| - | /(- | 23*2 |
| 1 | /(- | 23*21 |
| ) | / | 23*21- |
| + | + | 23*21-/ |
| 5 | + | 23*21-/5 |
| * | +* | 23*21-/53 |
| 3 | +* | 23*21-/53 |
| | Empty | 23*21-/53*+ |

So, the Postfix Expression is 23*21-/53*+

# Zadaci za vežbanje

1) 10+3*5/(16-4)
2) A+B*(C-D)+(E/F)*G/H
3) a+b*c-d/e*f
4) (a+b*c-d)/(e*f)

# Zanimljive animacije

- https://www.cs.usfca.edu/~galles/visualization/Algorithms.html
- https://www.youtube.com/watch?v=vq-nUF0G4fI
- https://www.youtube.com/watch?v=IAxCAbcqQFA
- https://www.youtube.com/watch?v=08sY6rYwfzE

# Test

1. Preko kojih struktura podataka se može implementirati stek?
2. Koje operacije ima stek?
3. Šta je TOP?
4. Opisati korišćenje steka kod operacije UNDO?
5. Prevesti iz infiksne u postfiksnu notaciju sledeći izraz 10+3*5/(16-4).
6. Prevesti iz infiksne u postfiksnu notaciju sledeći izraz A+B*(C-D)+(E/F)*G/H.
7. Prevesti iz infiksne u postfiksnu notaciju sledeći izraz a+b*c-d/e*f.
8. Prevesti iz infiksne u postfiksnu notaciju sledeći izraz (a+b*c-d)/(e*f).
9. Napisati metodu PUSH kod steka implementiranog preko lančane liste.
10. Napisati metodu POP kod steka implementiranog preko niza.

# Test poslati na apljaskovic@np.ac.rs do 13.4. u 14h.

**Упутство за студенте**

- **Адекватан наслов: Одговори на тест провере знања (контролна питања) за наставну недељу „.....“.**

- **Текст меила**: Поштовани, шаљем одговоре на тест провере знања (контролна питања) за наставну недељу „.....“ . У наставку текста **редом дати одговоре** почевши од питања под бројем 1. па до краја.

-Уколико студент не одговори на питање, рачуна се да не зна одговор на дато питање.

-Уколико студент до следећег предавања у 14 сати не пошаље меил са одговорима на тест провере знања за дату наставну јединицу сматра се да је није савладао наставну јединицу.

Hvala na pažnji!