**Računalstvo III1 27.04.2020 1h i 7h**

**Rekurzivna funkcija**

Rekurzivna funkcija je funkcija koja **poziva samu sebe**. U pravilu, **rekurzivna rješenja su kraća, ali je za njihovo izvođenje potrebno više vremena i memoriije.**

Svaka rekurzija mora imati uvjet zaustavljanja koji će omogućiti izlazak iz rekurzije. Inače će se rekurzivna funkcija izvoditi beskonačno. Također svaki poziv mora se približavati uvjetu zaustavljanja.

Svi podatci, njihove vrijednosti ili adrese, prenose se preko **STOGA** (eng. stack). **Stog je dinamički dio memorije koji se upotrebljava za pohranjivanje podataka koji se privremeno koriste u određenim dijelovima programa. Broj podataka na stogu je promjenjiv. Međutim, količina memorije za stog je ograničena.** Stog se puni i prazni prema principu „**zadnji unutra – prvi van“ (eng. LIFO – last in – first out).** Za bolje razumijevanjestoga često se koristi analogija s hrpom tanjura (npr. u restoranima): posljednji tanjir stavljen na hrpu se uzima prvi.

Na stogu se koriste dvije operacije: push i pop. Operacija push šalje podatak na stog dok operacija pop skida podatak sa stoga.

Za pohranjivanje rezultata i povratak iz rekurzije upotrebljava se STOG.

**Primjer izračunavanja vrijednosti funkcije ni s pomoću rekurzije:**

*/\*Ovaj problem već smo rješavali iterativno\*/*

0! = 1

1! =1

2! = 1\*2 = 2 $ \rightarrow $ 2! = 1!\*2

3! = 1\*2\*3 = 6 $\rightarrow $ 3! = 2!\*3

4! = 1\*2\*3\*4 = 24 $\rightarrow $ 4! = 3!\*4

5! = 1\*2\*3\*4\*5 = 120 $\rightarrow $ 5! = 4!\*5

itd.

n! = 1\*2\*3\*4\*5\*......(n-1)\*n $\rightarrow $ n! = (n-1)!\*n

Dakle n! Se može izračunati ako je poznat (n-1)!, a (n-1)! Može se izračunati ako je poznat (n-2)! itd.

Iz toga možemo zaključiti: ako računamo umnožak prvih n prirpdnih brojeva,

**uvjet zaustavljanja je 1! = 1**

**a opći oblik**  će biti n! = n\*(n-1)!.

**Rekurzivna funkcija**  bi glasila:

long  **int** faktorijel **(int** n**)**

{

**if** (n==1) /\*može biti i n==0\*/

**return** 1;

**else**

**return** faktorijel (n-1)\*n;

}

*Što se zapravo događa?*

**STOG** */\*na početkuje prazan\*/*

**1.korak**

Neka je **n = 4.** Pri prvom pozivu rekurzija funkcije na stog se spremi vrijednost n = 4.

|  |
| --- |
| 4 |

**2.korak**

S obzirom na to da 4 nije jednako 1 ponovno se poziva rekurzivna funkcija, ali sada je  **n = n-1 = 3** pa s na stog sprema vijednost 3.

|  |
| --- |
| 3 |
| 4 |

**3.korak**

S obzirom na to da opet nije ispunjen uvjet zaustavljanja, funkcija se ponovno poziva, ali za **n = 2.** Ta vrijednost se sprema na stog.

|  |
| --- |
| 2 |
| 3 |
| 4 |

**04.korak**

Funkcija se poziva ponovno s vrijednosti **n = 1.** Sada je uvjet zaustavljen zadovoljen.

|  |
| --- |
| 1 |
| 2 |
| 3 |
| 4 |

Kad rekurzivna funkcija dostigne uvjet zaustavljanja, stog se prazni po principu „prvi unutra – zadnji van“. Znači, prvo se izračunava vrijednost funkcije za n=1.

**1.korak**

faktorijel (1) = 1 *(uvjet zaustavljanja)*

|  |
| --- |
| 1 |
| 2 |
| 3 |
| 4 |

**2.korak**

Uzima se sljedeća vrijednost sa stoga (n=2) i izračunava

faktorijel(2) = 2 \* faktorijel(1) = 2\*1 = 2

|  |
| --- |
| 2 |
| 3 |
| 4 |

**3.korak**

Sada je n = 3 i

faktorijel(3) = 3 \* faktorijel(2) = 3\*2 = 6.

|  |
| --- |
| 3 |
| 4 |

**4.korak**

Napokon

faktorijel(4) = 4 \* faktorijel(3) = 4\*6 = 24.

|  |
| --- |
| 4 |

**Primjer:**

**Za nenegativni cijeli broj e sljedeća rekurzivna funkcija računa potenciju be , /\*b0=1\*/**

**long int** pontecija (**int** b, **int** e)

{

 **If** (e==0)

 **return=1;**

 **else**

 **return** b\*potencija(b, e-1);

}

**Zadatak:**

Napravite program kojim se, za uneseni prirodan broj n računa zbroj:



*/\*iskoristite prethodnu rekurzivnu funkciju za računanje faktorijela\*/*

Rješenje:

#include <stdio.h>

/\*rekurzivna funkcija za racunanje potencije+/

long int potencija(int b,int e)

{

 if (e==0)

 return 1;

 else

 return b\*potencija(b,e-1);

}

/\*rekurzivna funkcija za racunanje faktorijela\*/

long int faktorijel(int n)

{

 if (n==1)

 return 1;

 else

 return faktorijel(n-1)\*n;

}

void main(void)

{

 int x,i,y;

 float s;

 printf("Upisite prirodan broj x i n ");

 scanf("%d,%d",&x,&y);

 if (y<=0)

 printf("Broj mora biti prirodan");

 else

 {

 for (i=1;i<=y;i++)

 s+=(float)potencija(x,i)/faktorijel(i);

 printf("Zbroj je %.2f ",s);

 }

}

Za zadaću:

1. Napravite program kojim će te na zaslon ispisati svaki četvrti broj iz intervala od 200 do 3(for petlja).
2. Napišite program koji će izračunati 7! (while petlja).