Operacija adresiranja & i pokazivačke promenljive

Operator & daje adresu promenljive. Operator * daje sadrzaj pokazivacke promenljive.

66. Sta je rezultat rada sledeceg C programa?

```
#include <stdio.h>
main()
{  int x = 8;   int* px;
   printf("Adresa promenljive x je : %p, a vrednost je %d\n", &x,x);
   px = &x;
   printf("Vrednost promenljive px je  : %p\n", px);
   printf("Vrednost promenljive na koju ukazuje px (tj. *px) je : %d\n", *px);
  *px = 12; printf("Vrednost promenljive na koju ukazuje px (tj. *px) je : %d\n", *px);
   printf("Vrednost promenljive x je : %d\n", x);
}
```

Izlaz (u konkretnom slucaju):

```
Adresa promenljive x je : 0012FF88, a vrednost je  8
Vrednost promenljive px je : 0012FF88
Vrednost promenljive na koju ukazuje px (tj. *px) je : 8
Vrednost promenljive na koju ukazuje px (tj. *px) je : 12
Vrednost promenljive x je : 12
```

OBJASNJENJA

1. int* px; px = &x; Adresu promenjive x zapamticemo u novoj promenljivoj, pokazivackoj promenljivoj px. Tip nove promenljive je pokazivac na int (int*)
2. *px = 12; Menjamo vrednost promenljive na koju ukazuje px
3. U poslednjem printf("Vrednost promenljive x je : %d\n", x); posto px sadrzi adresu promenljive x, ona ukazuje na x tako da se ispisuje promenjena vrednost promenljive x

Veza između funkcija korišćenjem pokazivača

67. Sta je rezultat rada sledeceg programa?

```
#include <stdio.h>
void trampa(int x, int y)
{  int pom;
   pom = x; x = y; y = pom;
}
main()
{ int x = 3, y = 5;  trampa(x, y);
  printf("x=%d y=%d\n", x,y);
}
```

68. Sta je rezultat rada sledeceg programa?

```
#include <stdio.h>
void trampa(int *x, int *y)
{  int pom;
   pom = *x; *x = *y; *y = pom;
}
main()
{ int x = 3, y = 5;  trampa(&x, &y);
  printf("x=%d y=%d\n", x,y);
}
```

U zadataku 67 zbog prenosa parametara x,y po vrednosti, funkcija trampa razmenjuje kopije promenljivih x,y iz main-a, a ne samih promenljivih.
Evo zasto. Pogledajmo sledeci primer.

```
#include <stdio.h>
void trampa(int x, int y)
{ int tmp;
  printf("F-ja trampa menja vrednosti promenljivim na adresama : \n");
  printf("x : %p\n", &x);  printf("y : %p\n", &y);
  tmp = x; x = y; y = tmp;
}
main()
{
int x = 3, y = 5;
printf("Adresa promenljive x je %p\n", &x);
printf("Vrednost promenljive x je %d\n", x);
printf("Adresa promenljive y je %p\n", &y);
printf("Vrednost promenljive y je %d\n", y);
```

Izlaz u konkretnom slucaju:

```
Adresa promenljive x je 0012FF88
Vrednost promenljive x je 3
Adresa promenljive y je 0012FF84
Vrednost promenljive y je 5
```

```
trampa(x, y);
printf("Posle  trampa:\n");
printf("Vrednost promenljive x je %d\n", x);
printf("Vrednost promenljive y je %d\n", y);
}
```

```
F-ja trampa menja vrednosti promenljivim na adresama :
x : 0012FF78
y : 0012FF7C
Posle trampa:
Vrednost promenljive x je 3
Vrednost promenljive y je 5
```

Dakle, nisu iste adrese promenljive x sa kojim radi funkcija trampa i sa kojima radi f-ja main

Pogledajmo sad prosiren primer sa ispisom adresa u tacnoj verziji funkcije trampa

```
#include <stdio.h>
void trampa(int* px, int* py)
{
int tmp;
printf("trampa : F-ja menja vrednosti promenljivim na adresama : \n");
printf("px = %p\n", px);
printf("py = %p\n", py);
tmp = *px;
*px = *py;
*py = tmp;
}
main()
{
int x = 3, y = 5;
printf("Adresa promenljive x je %p\n", &x);
printf("Vrednost promenljive x je %d\n", x);
printf("Adresa promenljive y je %p\n", &y);
printf("Vrednost promenljive y je %d\n", y);
trampa(&x, &y);
printf("Posle trampa:\n");
printf("Vrednost promenljive x je %d\n", x);
printf("Vrednost promenljive y je %d\n", y);
}
```

```
Izlaz u konkretnom slucaju:
Adresa promenljive x je 0012FF88
Vrednost promenljive x je 3
Adresa promenljive y je 0012FF84
Vrednost promenljive y je 5
trampa : F-ja menja vrednosti promenljivim na adresama :
px = 0012FF88
py = 0012FF84
Posle trampa:
Vrednost promenljive x je 5
Vrednost promenljive y je 3
```

Dakle, iste su adrese promenljive x sa kojim radi funkcija trampa i sa kojima radi f-ja main

69. NCP koji učitava dva prirodna broja i ispisuje njihov celobronji količnik i ostatak pri deljenju prvog drugim brojem. Izračunavanje količnika i ostatka izdvojiti u zasebnu funkciju.

```
#include <stdio.h>
void div_and_mod(unsigned x, unsigned y, int* div, int* mod)
{ *div = x / y; *mod = x % y; }
main()
{
unsigned a,b; int div, mod;  scanf(”%u%u”,&a,&b); div_and_mod(a, b, &div, &mod);
printf("\ndiv je %d, mod je %d \n", div,mod);
}
```

Objasnjenja:
1. Funkcija div_and_mod istovremeno vraca dve vrednosti - kolicnik i ostatak dva data broja. Ovo se postize tako sto se funkciji predaju vrednosti dva broja (x i y) i adrese dve promenljive na koje ce se smestiti rezultati. Zato su argumenti funkcije pokazivacke promenljive int *div, int *mod
2. U main-u pozivamo funkciju div_and_mod tako sto joj saljemo vrednosti dva broja (5 i 2) i adrese promenljvih div i mod na koje ce se postaviti rezultati.

70. NCP koji ucitava dva cela broja i ispisuje njihov maksimum i minimum. Izaracunavanje min i max izdvojiti u jednu zasebnu f-ju.

```
#include <stdio.h>
void maxmin(int x, int y, int* max, int* min)
{ if(x<y) {*max=y; *min=x; } else {*max=x; *min=y;} }
main()
{
int a,b, max, min;  scanf("%d%d",&a,&b); maxmin(a, b, &max, &min);
printf("\nmax je %d, min je %d \n", max,min);
}
```

Rekurzivne funkcije **recur (**Desiti se ponovo) *latinski:* ***re-*** *= nazad +* ***currere*** *= izvrsavati*

Mnoge matematicke funkcije se mogu definisati rekurzivno:

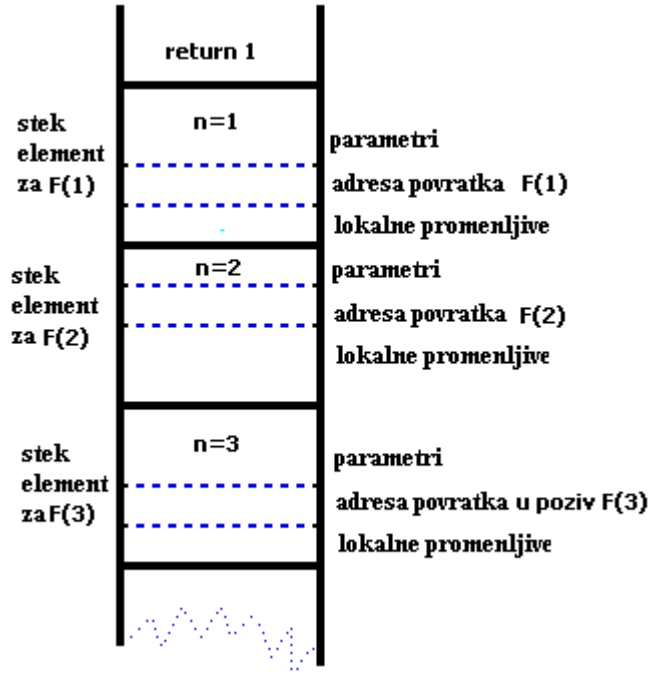
- faktorijel
- Fibonacijevi brojevi
- Euklidov NZD (najveci zajednicki delilac)
- aritmeticke operacije sabiranja, mnozenja

Jedan od najjednostavnijih primera rekurzivnih definicija je faktorijel funkcija i to je prirodan nacin za racunanje faktorijela, jer matematicki se faktorijel definise rekurzivno: F(0)=1, F(n)=F(n-1)*n, n>0

```
long F( int n )
    {   if ( n == 0 ) return 1;    else return n*F(n-1);
    }
```

Ova funkcija poziva samu sebe da izracuna sledeci clan. Na kraju ce doci do uslova prekida n==0 i izlaska iz rekurzije. Medjutim, pre nego sto dodje do uslova prekida, funkcija redom stavlja n poziva F(n-1), F(n-2), ..., F(2), F(1) na stek poziva.

Postojanje uslov prekida je neophodno u radu sa rekurzivnim funkcijama. Ako se izostavi, tada ce funkcija nastavi da poziva samu sebe sve dok program ne popuni ceo stek poziva i desice se neodgovarajuci rezultati! Skica rada za F(3):



ili krace rekurzivno resenje:

```
long F(int n)
{ return n == 0 ? 1 : n*F(n-1);}
```

71. a) Napisati rekurzivnu verziju funkcije float power(float f, int k) koja stepenuje realan broj f na celobrojni izlozilac k
b) Napisati iterativnu verziju funkcije float power(float f, int k) koja stepenuje realan broj f na celobrojni izlozilac k

```
float power(float f, int k){/* if (k == 0)    return 1;else return f*power(f, k-1);*/
/* Krace : */             return k==0 ? 1 : f*power(f, k-1);
}

/* Iterativna verzija prethodne funkcije */
float power(float x, int k)
{       int i;
              float s = 1;
              for (i = 0; i<k; i++)  s*=x;
              return s;
}
```

Koje resenje je bolje? Rekurzijom se obicno ne ˇstedi memorija, jer je potreban prostor (tzv. stek) za pamcenje podataka pri rekurzivnim pozivima funkcije. Rekurzijom se obicno ne povecava ni brzina izvrˇsavanja funkcije, ali je rekurzivni zapis kompaktniji, laksi za pisanje i razumevanje. Rekurzija ce nam biti posebno pogodna u obradi rekurzivno definisanih struktura podataka kakva su drveta, povezane liste,…

72. Napisati rekurzivnu verziju funkcije void printd(int n) koja stampa cifre celog broja n.
Ideja 1: Jedno resenje je racunanje cifara kao ostataka pri deljenju broja sa 10, od cifre najmanje do cifre najvece tezine, njihovo pamcenje u nizu, a zatim stampanje elemenata niza u obrnutom poretku. SKUPO!
Ideja 2: Druga mogucnost je rekurzivno resenje u kome funkcija prvo poziva samu sebe da obradi (odredi i odstampa) vodece cifre broja n, a zatim stampa poslednju cifru.

```
void printd(int n)
{ if(n<0) { putchar('-'); n = -n;}
  if(n/10)  printd(n/10);
  putchar(n%10 + '0');
}
```

73. Napisati rekurzivnu verziju funkcije int Z(int n) koja vraca zbir cifara broja n.

```
int Z(int n) { return n==0? 0: n%10+Z(n/10);}
```

74. Napisati rekurzivnu verziju funkcije unsigned NZD(unsigned x, unsigned y) koja vraca NZD prirodnih brojeva x,y.
Matematicka definicija f-je NZD(x,y) je rekurzivne prirode: NZD(x,y)=y ako x=0
NZD(x,y)=NZD(y%x,x) ako x!=0
Primer: NZD(12,18)=NZD(18%12,12)=NZD(6,12)=NZD(12%6,6)=NZD(0,6)=6
Sta ako x>y? NZD(18,12)=NZD(12%18, 18)= NZD(12,18)= i dalje kao u preth. Primeru

```
unsigned NZD(unsigned x, unsigned y)
{ return x==0? y:NZD(y%x,x); }
```

75. Napisati funkciju void Skrati(unsigned a, unsigned b, unsigned *c, unsigned *d) koja razlomak a/b, b!=0 dovodi do neskrativog razlomka c/d. Mozete koristi f-ju NZD iz prethodnog zadatka.

```
void Skrati(unsigned a, unsigned b, unsigned *c, unsigned *d)
{
  unsigned pom;
  pom=NZD(a,b);   *c=a/pom; *d=b/pom
}
```

KONTROLNI: funkcije i bitovi
Funkcije (knjiga cela glava 6), Bitovi (knjiga deo glave 3). Zadaci mogu, a i ne moraju biti iz knjige.