Rekurzija – n!

```
#include <stdio.h>

long fact(long n)
{
     if (n == 1) return 1;
     else return n * fact(n-1);
}

main()
{
      printf("%d\n",fact(3));
}
```

Rekurzija – n!

```
#include <stdio.h>

long fact(long n)
{
     if (n == 1) return 1;
     else return n * fact(n-1);
}

main()
{
      printf("%d\n",fact(3));
}
```

Pozivi na steku
fact, n = 3

Rekurzija – n!

```
#include <stdio.h>

long fact(long n)
{
     if (n == 1) return 1;
     else return n * fact(n-1);
}

main()
{
      printf("%d\n",fact(3));
}
```

n = 3

Pozivi na steku
fact, n = 3

Rekurzija – n!

```
#include <stdio.h>

long fact(long n)
{
     if (n == 1) return 1;
     else return n * fact(n-1);
}

main()
{
      printf("%d\n",fact(3));
}
```

n = 3

Pozivi na steku
fact, n = 2
fact, n = 3

Rekurzija – n!

```
#include <stdio.h>

long fact(long n)
{
     if (n == 1) return 1;
     else return n * fact(n-1);
}

main()
{
      printf("%d\n",fact(3));
}
```

n = 2

Pozivi na steku
fact, n = 2
fact, n = 3

Rekurzija – n!

```
#include <stdio.h>

long fact(long n)
{
     if (n == 1) return 1;
     else return n * fact(n-1);
}

main()
{
      printf("%d\n",fact(3));
}
```

n = 2

Pozivi na steku
fact, n = 1
fact, n = 2
fact, n = 3

Rekurzija – n!

```
#include <stdio.h>

long fact(long n)
{
     if (n == 1) return 1;
     else return n * fact(n-1);
}

main()
{
      printf("%d\n",fact(3));
}
```

n = 1

Pozivi na steku
× fact, n = 1 → 1
fact, n = 2
fact, n = 3

Rekurzija – n!

```
#include <stdio.h>

long fact(long n)
{
     if (n == 1) return 1;
     else return n * fact(n-1);
}

main()
{
      printf("%d\n",fact(3));
}
```

n = 2

1

Pozivi na steku
× fact, n = 2 → 2
fact, n = 3

Rekurzija – n!

```
#include <stdio.h>

long fact(long n)
{
     if (n == 1) return 1;
     else return n * fact(n-1);
}

main()
{
      printf("%d\n",fact(3));
}
```

n = 3

2

Pozivi na steku
× fact, n = 3 → 6

Rekurzija – n!

```
#include <stdio.h>

long fact(long n)
{
     if (n == 1) return 1;
     else return n * fact(n-1);
}

main()
{
      printf("%d\n",fact(3));
}
```

6

Pozivi na steku

Rekurzija

- **Primer**: Prikazivanje cifara nekog broja u pravilnom redosledu

```
/* printd: print n in decimal */
void printd(int n)
{
   if (n < 0)
   {
      putchar('-');
      n = -n;
   }
   if (n / 10)
      printd(n / 10);

   putchar(n % 10 + '0');
}
```

printd(-123)
n<0 ? → '-', n = 123
n/10 ? printd(12)
n<0 ?
n/10 ? printd(1)
n<0 ?
n/10 ?
→ '1'
→ '2'
→ '3'

- Rekurzija ne obezbeđuje uštedu memorijskog prostora niti je postupak izvršavanja brži.
- Kod je kompaktniji, jednostavniji za pisanje i razumevanje, pogodan za rad sa rekurzivno definisanim strukturama podataka

Eksterne promenljive

- Interne promenljive se definišu unutar funkcija
- Externe promenljive se definišu van funkcija i samim tim su dostupne svim funkcijama
- Funkcije su uvek eksterne zato što C ne dozvoljava definisanje funkcije unutar druge funkcije
- Eksterne promenljive su globalno dostupne i mogu se koristiti umesto argumenata ili vrednosti koja se vraća
- Ukoliko funkcije razmenjuju veliki broj parametara, nekada je pogodnije da to čine preko eksternih promenljivih. Ipak, to nije preporučljivo jer narušava strukturu programa i može dovesti do neželjenih efekata

Eksterne promenljive

Životni vek automatskih i spoljašnjih promenljivih

- Automatske promenljive su unutrašnje za funkciju. One nastaju kada otpočinje izvršavanje funkcije i nestaju kada se ono završava.
- Spoljašnje promenljive su permanentne i stoga zadržavaju svoje vrednosti između poziva funkcija. Dakle, životni ciklus ovih promenljivih je duži.

Kalkulator sa inverznom poljskom notacijom

- `(1 - 2) * (4 + 5)` - infiksni zapis
- `1 2 - 4 5 + *` - postfiksni zapis
- Implementacija
  - Svaki operand se stavlja na stek
  - Kada se naiđe na operator, sa steka se uzimaju operandi
  - Primenjuje se operator, a rezultat se ponovo stavlja na stek
- Primer: `1 2 - 4 5 + *`
  - 1 i 2 se stavljaju na stek
  - Zatim se skidaju sa steka i zamenjuju njihovom razlikom -1
  - 4 i 5 stavljaju na stek
  - Zatim se skidaju sa steka i zamenjuju njihovim zbirom 9
  - Zatim se -1 i 9 skidaju sa steka i zamenjuju njihovim proizvodom -9
  - Kada se stigne do kraja izraza, sa vrha steka se skida rezultat

Pseudo kod

```
while (sledeći operator ili operand nije znak za kraj fajla)
  if (broj)
    stavi ga na stek
  else if (operator)
    skini operande sa steka
    uradi operaciju
    stavi rezultat na stek
  else if (newline)
    skini i stampaj vrh steka
  else
    greška
```

- A gde je STEK?

Struktura programa

```
#include ...
#define ...

deklaracija funkcija za main

main()
{
   ...
}

eksterne promenljive za push i pop

void push( double f)
{
   ...
}

double pop(void)
{
   ...
}

int getop(char s[])
{
   ...
}

funkcije koje zove getop
```

```c
#include <stdio.h>
#include <stdlib.h> /* za atof() */

#define MAXOP 100 /* maksimalna velicina operanda ili operatora */
#define NUMBER '0' /* signal da je pronadjen broj */

int getop(char []);
void push(double);
double pop(void);

/* reverzni Poljski kalkulator */
main()
{
  int type;
  double op2;
  char s[MAXOP];

  while ((type = getop(s)) != EOF)
  {
    switch (type)
    {
      case NUMBER:
        push(atof(s));
        break;
      case '+':
        push(pop() + pop());
        break;
```

```
      case '*':
        push(pop() * pop());
        break;
      case '-':
        op2 = pop();
        push(pop() - op2);
        break;
      case '/':
        op2 = pop();
        if (op2 != 0.0)
          push(pop() / op2);
        else
          printf("Greska: deljenje nulom\n");
        break;
      case '\n':
        printf("\t%.8f\n", pop());
        break;
      default:
        printf("Greska: nepoznata komanda %s\n", s);
        break;
    }
  }

  return 0;
}
```

```c
#define MAXVAL 100 /* maksimalna visina steka */

int sp = 0; /* sledeca slobodna pozicija na steku */
double val[MAXVAL]; /* vrednosti na steku */

/* push: stavlja f na stek */
void push(double f)
{
  if (sp < MAXVAL)
    val[sp++] = f;
  else
    printf("Greska: stek je pun, nije moguce staviti %f\n", f);
}

/* pop: skida i vraca vrednost sa vrha steka */
double pop(void)
{
  if (sp > 0)
    return val[--sp];
  else
  {
    printf("Greska: stek je prazan\n");
    return 0.0;
  }
}
```

```c
#include <ctype.h>

int getch(void);
void ungetch(int);

/* getop: get next character or numeric operand */
int getop(char s[])
{
  int i, c;

  while ((s[0] = c = getch()) == ' ' || c == '\t') ;

  s[1] = '\0';

  if (!isdigit(c) && c != '.')
    return c; /* nije broj */

  i = 0;
  if (isdigit(c)) /* procitaj celobrojni deo */
    while (isdigit(s[++i] = c = getch())) ;

  if (c == '.') /* procitaj decimalni deo */
    while (isdigit(s[++i] = c = getch())) ;

  s[i] = '\0';

  if (c != EOF)   ungetch(c);

  return NUMBER;
}
```

Šta su getch() i ungetch()?

```
#define BUFSIZE 100

char buf[BUFSIZE]; /* bafer za ungetch */
int bufp = 0; /* sledeca slobodna pozicija u baferu */

int getch(void) /* uzima karakter iz bafera ili sa ulaza */
{
  return (bufp > 0) ? buf[--bufp] : getchar();
}

void ungetch(int c) /* vraca karakter u bafer */
{
  if (bufp >= BUFSIZE)
    printf("ungetch: prekoracena velicina bafera\n");
  else
    buf[bufp++] = c;
}
```

Opseg vidljivosti

- Domet (*scope*) nekog imena je deo programa unutar kojeg se to ime može koristiti.
- Automatske promenljive deklarisana na početku funkcije – cela funkcija. Lokalne promenljive i parametri funkcije
- Domet spoljašnje/eksterne promenljive - vidljive su od tačke u kojoj su deklarisane

```
main() { ... }

int sp = 0;
double val[MAXVAL];

void push(double f) { ... }
double pop(void) { ... }
```

Opseg vidljivosti

- Deklaracija promenljive objavljuje svojstva te promenljive.
- Definicija promenljive dovodi do rezervisanja memorije.
- Ako neka promenljiva treba da se koristi pre nego što se definiše, ili ako je definisana u različitoj izvornoj datoteci od one u kojoj se koristi tada je obavezna deklaracija **extern**
- Definicija (samo na jednom mestu)

```
int sp;
double val[MAXVAL];
```

- Deklaracija
  (svuda gde se koriste promenljive)

```
extern int sp;
extern double val[];
```

u fajlu1:

```
extern int sp;
extern double val[];

void push(double f) { ... }
double pop(void) { ... }
```

u fajlu2:

```
int sp = 0;
double val[MAXVAL];
```

Header fajlovi

calc.h

```
#define NUMBER '0'
void push(double);
double pop(void);
int getop(char []);
int getch(void);
void ungetch(int);
```

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "calc.h"
#define MAXOP 100
main() {
    ...
}
```

getop.c

```
#include <stdio.h>
#include <ctype.h>
#include "calc.h"
getop() {
    ...
}
```

stack.c

```
#include <stdio.h>
#include "calc.h"
#define MAXVAL 100
int sp = 0;
double val[MAXVAL];
void push(double) {
    ...
}
double pop(void) {
    ...
}
```

getch.c

```
#include <stdio.h>
#define BUFSIZE 100
char buf[BUFSIZE];
int bufp = 0;
int getch(void) {
    ...
}
void ungetch(int) {
    ...
}
```

gcc -o calc main.c getop.c stack.c getch.c

Statičke promenljive

- Promenljive `sp` i `val` u **`stack.c`** i `buf` i `bufp` u **`getch.c`** su namenjene samo za te datoteke
- Deklaracija **`static`** ograničava domet promenljive ili funkcije samo na fajl u kome su deklarisane

```
static char buf[BUFSIZE]; /* bafer za ungetch */
static int bufp = 0; /* sledeca slobodna pozicija u baferu */

int getch(void) { ... }
void ungetch(int c) { ... }
```

- Samo ove dve funkcije će videti promenljive `buf` i `bufp`
- Funkcije u drugim fajlovima ne vide ove promenljive
- U drugim fajlovima moguće je deklarisati promenljive sa istim imenima
- **`static`** se može primeniti na interne (lokalne) promenljive. Tada njihov vek trajanja nije ograničen samo na telo funkcije. Iako im je opseg vidljivosti samo unutar funkcije, one postoje sve vreme izvršenja programa.

Registarske promenljive

- Deklaracija **`register`** obaveštava kompajler da će promenljiva koja sledi biti veoma često korišćena
- Namera je da registarske promenljive budu smeštene u mašinskim registrima, što vodi do manjih i bržih programa
- Kompajleri su slobodni da odluče da li će da proglase ove promenljive za registarske
- Mogu se promeniti samo na automatske promenljive i formalne parametre funkcije

```
register int x;
register char c;

f(register unsigned m, register long n)
{
     register int i;
               ...
}
```

Blokovska struktura

- Promenljive se mogu deklarisati i unutar blokova
- Promenljive deklarisane unutar bloka su vidljive samo u tom bloku
- Njihov opseg vidljivosti i životni vek su samo unutar tog bloka
- Životni vek statičkih promenljivih deklarisanih unutar bloka traje sve vreme izvršenja programa
- Promenljive deklarisane unutar bloka zaklanjaju istoimene promenljive deklarisane van bloka (treba izbegavati)

```
int i;

if (n > 0)
{
   int i; /* deklaracija novog i */
   for (i = 0; i < n; i++)
      ...
}
```

- Automatske promenljive se inicijalizuju prilikom svakog ulaska u blok, dok se statičke inicijalizuju samo pri prvom ulasku u blok

Inicijalizacija

- Ne treba se oslanjati na to da će kompajler automatski inicijalizovati promenljive

```
int x = 1;
char squota = '\'';
long day = 1000L * 60L * 60L * 24L; /* milliseconds/day */
```

- Za eksterne i statičke promenljive inicijalizator mora biti konstantan izraz
- Za automatske i registarske promenljive inicijalizator može biti bilo koji izraz u kome učestvuju prethodno definisane vrednosti
- Nizovi se inicijalizuju navođenjem vrednosti u velikim zagradama

```
int days[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
```

- Znakovni nizovi (stringovi) se mogu inicijalizovati pomoću navodnika

```
char pattern[] = "ould";
```

što je skraćeni zapis od

```
char pattern[] = { 'o', 'u', 'l', 'd', '\0' };
```