

# Lekcija 5 – Funkcije

## Pregled

- 5.1 Uvod
- 5.2 Programski moduli u C-u
- 5.3 Matematičke funkcije
- 5.4 Funkije
- 5.5 Definicije funkcija
- 5.6 Protipovi funkcija
- 5.7 Header datoteke
- 5.8 Pozivanje funkcija: po vrijednosti i referenci
- 5.9 Generisanje slučajnih brojeva
- 5.10 Primjer: A Game of Chance
- 5.11 Memorijske klase (storage classes)
- 5.12 Opseg važenja (scope rules)
- 5.13 Rekurzija
- 5.14 Primjer rekurzije: Fibonačijevi brojevi
- 5.15 Rekurzija vs. Iteracija

## Ciljevi lekcije

- U ovoj lekciji:
  - Razumijećete kako treba konstruisati programe od modula (funkcija).
  - Upoznaćete se sa osnovnim matematičkim funkcijama iz standardne biblioteke.
  - Naučićete da kreirate nove funkcije.
  - Shvatićete mehanizme prenošenja parametara.
  - Upoznaćete se sa generisanjem slučajnih brojeva.
  - Shvatićete osnovne principe rekurzivnih funkcija.

## 5.1 Uvod

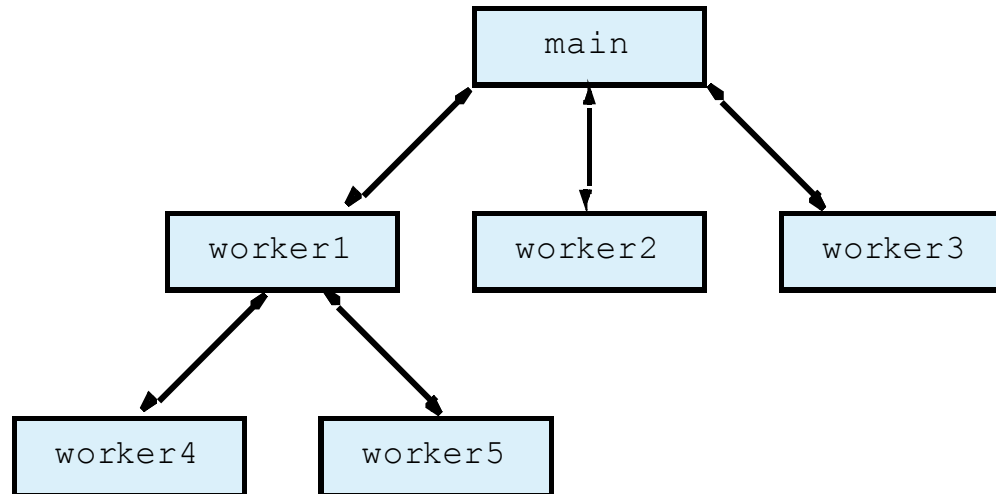
- Podijeli pa vladaj (divide and conquer)
  - Konstruisanje programa od manjih komponenti
    - Manje djelove koda (komponente) nazivamo modulima
  - Svaki komad koda (komponentu) je lakše obraditi nego originalni program

## 5.2 Programski moduli u C

- Funkcije
  - Moduli u C-u
  - Programi kombinuju korisnički definisane funkcije sa funkcijama iz biblioteka
    - C standardne biblioteke imaju veliki broj funkcija
- Pozivanje funkcija
  - Pozivanje (“invoking functions”)
    - Obezbijediti naziv funkcije i argumente
    - Funkcije obrađuju određene operacije
    - Funkcije (mogu da) vraćaju rezultat
  - Analogija pozivanja funkcija:
    - Šef zahtijeva od radnika da odradi određen posao
      - Radnik dobija informacije, odradi posao, vraća rezultat
      - Sakrivanje informacija: šef ne zna detalje posla

## 5.2 Programski moduli u C

Fig. 5.1 Hijerarhija funkcija šefa/funkcija radnika



## 5.3 Matematička biblioteka

- Matematička biblioteka
  - Standardna matematička izračunavanja
  - `#include <math.h>`
- Format poziva funkcija
  - `FunctionName( argument );`
    - Ako ima više argumenata, razdvajamo ih zarezima
  - `printf( "%.2f", sqrt( 900.0 ) );`
    - Pozivamo funkciju `sqrt`, koja vraća kvadratni korijen argumenta
    - Sve matematičke funkcije vraćaju tip `double`
  - Argumenti mogu biti konstante, promjenljive ili izrazi

## 5.3 Math Library Functions

Funkcija	Opje	Primjer
<code>sqrt( x )</code>	kvadratni korijen iz $x$	<code>sqrt( 900.0 )</code> je 30.0 <code>sqrt( 9.0 )</code> je 3.0
<code>exp( x )</code>	eksponencijalna $e^x$	<code>exp( 1.0 )</code> je 2.718282 <code>exp( 2.0 )</code> je 7.389056
<code>log( x )</code>	logaritam od $x$ (osnova $e$ )	<code>log( 2.718282 )</code> je 1.0 <code>log( 7.389056 )</code> je 2.0
<code>log10( x )</code>	logaritam od $x$ (osnova 10)	<code>log10( 1.0 )</code> je 0.0 <code>log10( 10.0 )</code> je 1.0 <code>log10( 100.0 )</code> je 2.0
<code>fabs( x )</code>	apsolutna vrijednost $x$	<code>fabs( 5.0 )</code> je 5.0 <code>fabs( 0.0 )</code> je 0.0 <code>fabs( -5.0 )</code> je 5.0
<code>ceil( x )</code>	najmanji cio broj ne manji od $x$	<code>ceil( 9.2 )</code> je 10.0 <code>ceil( -9.8 )</code> je -9.0
<code>floor( x )</code>	najveći cio broj ne veći od $x$	<code>floor( 9.2 )</code> je 9.0 <code>floor( -9.8 )</code> je -10.0
<code>pow( x, y )</code>	$x^y$	<code>pow( 2, 7 )</code> je 128.0 <code>pow( 9, .5 )</code> je 3.0
<code>fmod( x, y )</code>	ostatak pri dijeljenju	<code>fmod( 13.657, 2.333 )</code> je 1.992
<code>sin( x )</code>	$\sin x$ ( $x$ u radijanima)	<code>sin( 0.0 )</code> je 0.0
<code>cos( x )</code>	$\cos x$ ( $x$ u radijanima)	<code>cos( 0.0 )</code> je 1.0
<code>tan( x )</code>	$\tan x$ ( $x$ u radijanima)	<code>tan( 0.0 )</code> je 0.0

Fig. 5.2 Neke funkcije iz math.h.

## 5.4 Funkcije

- Funkcije
  - Modularizuju program
  - Sve promjenljive definisane unutar funkcije su lokalne
    - Poznate samo u funkciji u kojoj su definisane
  - Parametri
    - Komunikacija informacijama između funkcija
    - Ponašaju se kao lokalne promjenljive
- Prednosti funkcija
  - Podijeli pa vladaj (divide and conquer)
    - Lakši razvoj programa



## 5.4 Funkcije

- Prednosti funkcija (nastavak)
  - Ponovno korišćenje softvera (software reusability)
    - Koristiti postojeće funkcije kao gradivne blokove novih programa
    - Apstrakcija – sakrivanje detalja implementacije ( kao u bibliotekama)
  - Izbjegavamo ponavljanje koda

## 5.5 Definisanje funkcija

- Format za definisanje funkcija

```
return-value-type function-name( parameter-list )  
{  
    declarations and statements  
}
```

- Function-name: ispravan identifikator
- Return-value-type: tip rezultata (default `int`)
  - `void` – označava da funkcija ne vraća vrijednost
- Parameter-list: lista parametara razdvojenih zarezima, deklariše parametre
  - Tip mora biti eksplicitno naveden za svaki parametar, osim ako je tip `int`

## 5.5 Definisanje funkcija

- Format za definisanje funkcija (nastavak)

```
return-value-type function-name( parameter-list )  
{  
    declarations and statements  
}
```

- Declarations and statements: tijelo funkcije (blok)
  - Promjenljive mogu biti definisane unutar bloka (ugnježdavanje)
  - Funkcije ne mogu biti definisane unutar drugih funkcija
- Povratak iz funkcije
  - Ako ne vraća vrijednost
    - `return;`
    - Ili, kad naiđemo na desnu (zatvorenu) zagradu `}`
  - Ako vraća vrijednost
    - `return expression;`

**fig05\_03.c (Part 1  
of 2)**

```
1  /* Fig. 5.3: fig05_03.c
2      Creating and using a programmer-defined function */
3  #include <stdio.h>
4
5  int square( int y ); /* function prototype */
6
7  /* function main begins program execution */
8  int main()
9  {
10     int x; /* counter */
11
12     /* loop 10 times and calculate and output square of x each time */
13     for ( x = 1; x <= 10; x++ ) {
14         printf( "%d ", square( x ) ); /* function call */
15     } /* end for */
16
17     printf( "\n" );
18
19     return 0; /* indicates successful termination */
20
21 } /* end main */
22
```



## Outline



**fig05\_03.c (Part 2  
of 2)**

**Program Output**

```
23 /* square function definition returns square of an integer */
24 int square( int y ) /* y is a copy of argument to function */
25 {
26     return y * y; /* returns square of y as an int */
27
28 } /* end function square */
```

```
1  4  9 16 25 36 49 64 81 100
```

**fig05\_04.c (Part 1  
of 2)**

```
1  /* Fig. 5.4: fig05_04.c
2      Finding the maximum of three integers */
3  #include <stdio.h>
4
5  int maximum( int x, int y, int z ); /* function prototype */
6
7  /* function main begins program execution */
8  int main()
9  {
10     int number1; /* first integer */
11     int number2; /* second integer */
12     int number3; /* third integer */
13
14     printf( "Enter three integers: " );
15     scanf( "%d%d%d", &number1, &number2, &number3 );
16
17     /* number1, number2 and number3 are arguments
18        to the maximum function call */
19     printf( "Maximum is: %d\n", maximum( number1, number2, number3 ) );
20
21     return 0; /* indicates successful termination */
22
23 } /* end main */
24
```

**fig05\_04.c (Part 2  
of 2)**

```
25 /* Function maximum definition */
26 /* x, y and z are parameters */
27 int maximum( int x, int y, int z )
28 {
29     int max = x;      /* assume x is largest */
30
31     if ( y > max ) { /* if y is larger than max, assign y to max */
32         max = y;
33     } /* end if */
34
35     if ( z > max ) { /* if z is larger than max, assign z to max */
36         max = z;
37     } /* end if */
38
39     return max;      /* max is largest value */
40
41 } /* end function maximum */
```

```
Enter three integers: 22 85 17
Maximum is: 85
Enter three integers: 85 22 17
Maximum is: 85
Enter three integers: 22 17 85
Maximum is: 85
```

**Program Output**

## 5.6 Prototip funkcije

- Prototip funkcije
  - Naziv funkcije
  - Parametri – šta funkcija “uzima”
  - Tip rezultata – tip koji funkcija vraća (default `int`)
  - Prototip se koristi za validaciju
  - Prototip je neophodan samo ako se definicija funkcije pojavljuje poslije upotrebe funkcije u programu
  - Funkcija sa prototipom

```
int maximum( int x, int y, int z );
```

    - Ima 3 cijela broja kao argumente
    - Vraća `int`
- Pravilo promocije i konverzije
  - Konverzija u “niži” tip može dovesti do greške



## 5.6 Prototip funkcije

Tipovi podataka	printf konverzija specifikacija	scanf konverzija specifikacija
long double	%Lf	%Lf
double	%f	%lf
float	%f	%f
unsigned long int	%lu	%lu
long int	%ld	%ld
unsigned int	%u	%u
int	%d	%d
short	%hd	%hd
char	%c	%c

Fig. 5.5 Hijerarhija promocije za tipove podataka.

## 5.7 Header datoteke

- Header datoteke
  - Sadrže prototipove funkcija za bibliotečke funkcije
  - `<stdlib.h>` , `<math.h>` , itd.
  - Učitavamo ih sa `#include <filename>`  
`#include <math.h>`
- Korisničke header datoteke
  - Kreiramo datoteku sa funkcijom
  - Sačuvamo je kao `filename.h`
  - Uključujemo je u druge datoteke sa  
`#include "filename.h"`
  - Ponovna upotreba funkcija

## 5.7 Header datoteke

Standard library header	Objašnjenje
<code>&lt;assert.h&gt;</code>	Sadrži makroe i informacije koje dodaju dijagnostiku koja pomaže pri debugovanju.
<code>&lt;ctype.h&gt;</code>	Testiranje karaktera na određena svojstva i konverziju malih u velika slova i obratno.
<code>&lt;errno.h&gt;</code>	Makroi za prijavljivanje grešaka.
<code>&lt;float.h&gt;</code>	Granice za veličinu floating point brojeva na sistemu.
<code>&lt;limits.h&gt;</code>	Granice za veličinu cijelih brojeva na sistemu.
<code>&lt;locale.h&gt;</code>	Informacije za podešavanje programa na lokalne uslove kao što su format datuma, vremena, itd.
<code>&lt;math.h&gt;</code>	Matematičke funkcije.
<code>&lt;setjmp.h&gt;</code>	Mogućnost prevazilaženja uobičajenog načina pozivanja funkcija i povratka iz funkcije.

## 5.7 Header datoteke

Standard library header	Objašnjenje
<code>&lt;signal.h&gt;</code>	Obrada različitih uslova koji se mogu pojaviti pri izvršavanju programa.
<code>&lt;stdarg.h&gt;</code>	Makroi za obradu funkcija kod kojih nije poznat broj i tip argumenata..
<code>&lt;stddef.h&gt;</code>	Definicije tipove koje C koristi z određena izračunavanja.
<code>&lt;stdio.h&gt;</code>	Ulazno-izlazne funkcije i informacije o njima..
<code>&lt;stdlib.h&gt;</code>	Funkcije za konverziju brojeva u stringove i obratno, alokaciju memorije, slučajne brojeve, itd.
<code>&lt;string.h&gt;</code>	Funkcije za obradu stringova.
<code>&lt;time.h&gt;</code>	Funkcije za obradu datuma i vremena.
<b>Fig. 5.6</b> Neke standardne header datoteke.	

## 5.8 Predaja argumenata: po vrijednosti i referenci

- Po vrijednosti
  - Kopija argumenta predaje se funkciji
  - Promjena argumenta u funkciji ne utiče na original
  - Koristiti kada funkcija ne treba da modifikuje argument
    - Izbjegavaju se slučajne promjene argumenta
- Po referenci
  - Predaje se originalni argument
  - Promjena u funkciji utiče na original
  - Koristi se samo sa funkcijama kojima “vjerujemo”
- Za sada, samo po vrijednosti

## 5.9 Generisanje slučajnih brojeva

- rand funkcija
  - Učitati `<stdlib.h>`
  - Vraća “slučajan” (random) broj iz intervala 0 - RAND\_MAX (najmanje 32767)  
 $i = \text{rand}();$
  - Pseudoslučajan broj
    - Ranije postavi niz “slučajnih” brojeva
    - Isti niz za svaki poziv funkcije
- Skaliranje
  - Da bi dobili slučajan broj između 1 i n  
 $1 + ( \text{rand}() \% n )$ 
    - $\text{rand}() \% n$  vraća broj između 0 and  $n - 1$
    - Dodajemo 1 da bi ušli u zadati interval  
 $1 + ( \text{rand}() \% 6 )$ 
      - Broj između 1 i 6

## 5.9 Generisanje slučajnih brojeva

- `srand` funkcija
  - `<stdlib.h>`
  - Ima cjelobrojni argument (`seed`) i skače na tu lokaciju u nizu slučajnih brojeva
    - `srand( seed );`
  - `srand( time( NULL ) ); /*load <time.h> */`
    - `time( NULL )`
      - Vraća vrijeme za koje je program kompajliran u sekundama
      - “Randomizes” seed

**fig05\_07.c**

```
1  /* Fig. 5.7: fig05_07.c
2     Shifted, scaled integers produced by 1 + rand() % 6 */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  /* function main begins program execution */
7  int main()
8  {
9     int i; /* counter */
10
11     /* loop 20 times */
12     for ( i = 1; i <= 20; i++ ) {
13
14         /* pick random number from 1 to 6 and output it */
15         printf( "%10d", 1 + ( rand() % 6 ) );
16
17         /* if counter is divisible by 5, begin new line of output */
18         if ( i % 5 == 0 ) {
19             printf( "\n" );
20         } /* end if */
21
22     } /* end for */
23
24     return 0; /* indicates successful termination */
25
26 } /* end main */
```



[Outline](#)**Program Output**

6	6	5	5	6
5	1	1	5	3
6	6	2	4	2
6	2	3	4	1

**fig05\_08.c (Part 1 of 3)**

```
1  /* Fig. 5.8: fig05_08.c
2     Roll a six-sided die 6000 times */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  /* function main begins program execution */
7  int main()
8  {
9     int frequency1 = 0; /* rolled 1 counter */
10    int frequency2 = 0; /* rolled 2 counter */
11    int frequency3 = 0; /* rolled 3 counter */
12    int frequency4 = 0; /* rolled 4 counter */
13    int frequency5 = 0; /* rolled 5 counter */
14    int frequency6 = 0; /* rolled 6 counter */
15
16    int roll; /* roll counter */
17    int face; /* represents one roll of the die, value 1 to 6 */
18
19    /* loop 6000 times and summarize results */
20    for ( roll = 1; roll <= 6000; roll++ ) {
21        face = 1 + rand() % 6; /* random number from 1 to 6 */
22
```

**fig05\_08.c (Part 2  
of 3)**

```
23  /* determine face value and increment appropriate counter */
24  switch ( face ) {
25
26      case 1:          /* rolled 1 */
27          ++frequency1;
28          break;
29
30      case 2:          /* rolled 2 */
31          ++frequency2;
32          break;
33
34      case 3:          /* rolled 3 */
35          ++frequency3;
36          break;
37
38      case 4:          /* rolled 4 */
39          ++frequency4;
40          break;
41
42      case 5:          /* rolled 5 */
43          ++frequency5;
44          break;
45
```



## fig05\_08.c (Part 3 of 3)

```

45         case 6:          /* rolled 6 */
46             ++frequency6;
47             break;
48         } /* end switch */
49
50
51     } /* end for */
52
53     /* display results in tabular format */
54     printf( "%s%13s\n", "Face", "Frequency" );
55     printf( "    1%13d\n", frequency1 );
56     printf( "    2%13d\n", frequency2 );
57     printf( "    3%13d\n", frequency3 );
58     printf( "    4%13d\n", frequency4 );
59     printf( "    5%13d\n", frequency5 );
60     printf( "    6%13d\n", frequency6 );
61
62     return 0; /* indicates successful termination */
63
64 } /* end main */

```

Face	Frequency
1	1003
2	1017
3	983
4	994
5	1004
6	999

## Program Output

**fig05\_09.c (Part 1  
of 2)**

```
1  /* Fig. 5.9: fig05_09.c
2      Randomizing die-rolling program */
3  #include <stdlib.h>
4  #include <stdio.h>
5
6  /* function main begins program execution */
7  int main()
8  {
9      int i;          /* counter */
10     unsigned seed; /* number used to seed random number generator */
11
12     printf( "Enter seed: " );
13     scanf( "%u", &seed );
14
15     srand( seed ); /* seed random number generator */
16
17     /* loop 10 times */
18     for ( i = 1; i <= 10; i++ ) {
19
20         /* pick a random number from 1 to 6 and output it */
21         printf( "%10d", 1 + ( rand() % 6 ) );
22
```



## Outline



### fig05\_09.c (Part 2 of 2)

```

23      /* if counter is divisible by 5, begin a new line of output */
24      if ( i % 5 == 0 ) {
25          printf( "\n" );
26      } /* end if */
27
28  } /* end for */
29
30  return 0; /* indicates successful termination */
31
32 } /* end main */

```

```

Enter seed: 67
    6      1      4      6      2
    1      6      1      6      4
Enter seed: 867
    2      4      6      1      6
    1      1      3      6      2
Enter seed: 67
    6      1      4      6      2
    1      6      1      6      4

```

### Program Output

## 5.10 Primjer: Igra kockicama

- Simulator igre
- Pravila
  - Bacamo 2 kockice
    - 7 ili 11 u prvom bacanju, igrač pobjeđuje
    - 2, 3 ili 12 u prvom bacanju, igrač gubi
    - 4, 5, 6, 8, 9, 10 – bodovi igrača
  - Ako ima drugog (i ostalih) bacanja, igrač mora da osvoji isti broj bodova kao i u prvom bacanju, i tada pobjeđuje; ako dobije 7, tada igrač gubi.

**fig05\_10.c (Part 1  
of 4)**

```
1  /* Fig. 5.10: fig05_10.c
2     Craps */
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <time.h> /* contains prototype for function time */
6
7  /* enumeration constants represent game status */
8  enum Status { CONTINUE, WON, LOST };
9
10 int rollDice( void ); /* function prototype */
11
12 /* function main begins program execution */
13 int main()
14 {
15     int sum;          /* sum of rolled dice */
16     int myPoint;      /* point earned */
17
18     enum Status gameStatus; /* can contain CONTINUE, WON, or LOST */
19
20     /* randomize random number generator using current time */
21     srand( time( NULL ) );
22
23     sum = rollDice( ); /* first roll of the dice */
24
```



**fig05\_10.c (Part 2  
of 4)**

```
25  /* determine game status based on sum of dice */
26  switch( sum ) {
27
28      /* win on first roll */
29      case 7:
30      case 11:
31          gameStatus = WON;
32          break;
33
34      /* lose on first roll */
35      case 2:
36      case 3:
37      case 12:
38          gameStatus = LOST;
39          break;
40
41      /* remember point */
42      default:
43          gameStatus = CONTINUE;
44          myPoint = sum;
45          printf( "Point is %d\n", myPoint );
46          break; /* optional */
47  } /* end switch */
48
```

**fig05\_10.c (Part 3  
of 4)**

```
49  /* while game not complete */
50  while ( gameStatus == CONTINUE ) {
51      sum = rollDice( ); /* roll dice again */
52
53      /* determine game status */
54      if ( sum == myPoint ) { /* win by making point */
55          gameStatus = WON;
56      } /* end if */
57      else {
58
59          if ( sum == 7 ) { /* lose by rolling 7 */
60              gameStatus = LOST;
61          } /* end if */
62
63      } /* end else */
64
65  } /* end while */
66
67  /* display won or lost message */
68  if ( gameStatus == WON ) {
69      printf( "Player wins\n" );
70  } /* end if */
71  else {
72      printf( "Player loses\n" );
73  } /* end else */
74
```

**fig05\_10.c (Part 4  
of 4)**

```
75     return 0; /* indicates successful termination */
76
77 } /* end main */
78
79 /* roll dice, calculate sum and display results */
80 int rollDice( void )
81 {
82     int die1;    /* first die */
83     int die2;    /* second die */
84     int workSum; /* sum of dice */
85
86     die1 = 1 + ( rand() % 6 ); /* pick random die1 value */
87     die2 = 1 + ( rand() % 6 ); /* pick random die2 value */
88     workSum = die1 + die2;     /* sum die1 and die2 */
89
90     /* display results of this roll */
91     printf( "Player rolled %d + %d = %d\n", die1, die2, workSum );
92
93     return workSum; /* return sum of dice */
94
95 } /* end function rollDice */
```

**Program Output**

Player rolled  $5 + 6 = 11$   
Player wins

Player rolled  $4 + 1 = 5$   
Point is 5

Player rolled  $6 + 2 = 8$

Player rolled  $2 + 1 = 3$

Player rolled  $3 + 2 = 5$

Player wins

Player rolled  $1 + 1 = 2$   
Player loses

Player rolled  $1 + 4 = 5$   
Point is 5  
Player rolled  $3 + 4 = 7$   
Player loses

## 5.11 Memorijske klase (storage classes)

- Specifikacija memorijskih klasa
  - Vrijeme memorisanja – koliko dugo objekat postoji u memoriji
  - Opseg važenja (scope) – gdje objekat može biti referenciran u programu
  - Povezivanje (linkage) – specificira datoteke u kojim je identifikator poznat (može se koristiti)
- Automatske promjenljive
  - Objekat je kreiran i uništen unutar svog bloka
  - `auto`: default za lokalne promjenljive local variables  
`auto double x, y;`
  - `register`: pokušava da smjesti promjenljive u registre
    - Može se koristiti samo sa automatskim promjenljivim  
`register int counter = 1;`

## 5.11 Memorijske klase (storage classes)

- Statičke promjenljive
  - Promjenljive postoje u toku čitavog vremena izvršavanja programa
  - Default vrijednost je nula
  - `static`: lokalne promjenljive definisane u funkciji.
    - Čuvaju vrijednost po završetku funkcije
    - Mogu se koristiti samo u funkciji
  - `extern`: default za globalne promjenljive i funkcije
    - Mogu se koristiti u svim funkcijama

## 5.12 Opseg važenja

- Važenje u datoteci (file scope)
  - Identifikator definisan van funkcije može se koristiti u svim funkcijama
  - Koristi se za globalne promjenljive, definicije i prototipove funkcija
- Važenje u funkciji (function scope)
  - Može se koristiti samo unutar tijela funkcije
  - Koristi se za labele (`start:`, `case:`, etc.)

## 5.12 Opseg važenja

- Važenje u bloku (block scope)
  - Identifikator deklarisan unutar bloka
    - Blok počinje sa definicijom, završava zagradom
  - Koristi se za promjenljive i parametre funkcija (lokalne promjenljive funkcije)
  - Spoljašnji blokovi su sakriveni od unutrašnjih blokova, ako postoji promjenljiva sa istim imenom unutar unutrašnjeg bloka
- Važenje u prototipu (function prototype scope)
  - Koristi se za identifikatore u listi argumenata



**fig05\_12.c (Part 1  
of 3)**

```
1  /* Fig. 5.12: fig05_12.c
2     A scoping example */
3  #include <stdio.h>
4
5  void useLocal( void );      /* function prototype */
6  void useStaticLocal( void ); /* function prototype */
7  void useGlobal( void );     /* function prototype */
8
9  int x = 1; /* global variable */
10
11 /* function main begins program execution */
12 int main()
13 {
14     int x = 5; /* local variable to main */
15
16     printf("local x in outer scope of main is %d\n", x );
17
18     { /* start new scope */
19         int x = 7; /* local variable to new scope */
20
21         printf( "local x in inner scope of main is %d\n", x );
22     } /* end new scope */
23
24     printf( "local x in outer scope of main is %d\n", x );
25
```



## Fig05\_12.c (Part 2 of 3)

```
26 useLocal();          /* useLocal has automatic local x */
27 useStaticLocal();    /* useStaticLocal has static local x */
28 useGlobal();         /* useGlobal uses global x */
29 useLocal();          /* useLocal reinitializes automatic local x */
30 useStaticLocal();    /* static local x retains its prior value */
31 useGlobal();         /* global x also retains its value */
32
33 printf( "local x in main is %d\n", x );
34
35 return 0; /* indicates successful termination */
36
37 } /* end main */
38
39 /* useLocal reinitializes local variable x during each call */
40 void useLocal( void )
41 {
42     int x = 25; /* initialized each time useLocal is called */
43
44     printf( "\nlocal x in a is %d after entering a\n", x );
45     x++;
46     printf( "local x in a is %d before exiting a\n", x );
47 } /* end function useLocal */
48
```

**fig05\_12.c (Part 3  
of 3)**

```
49  /* useStaticLocal initializes static local variable x only the first time
50     the function is called; value of x is saved between calls to this
51     function */
52  void useStaticLocal( void )
53  {
54      /* initialized only first time useStaticLocal is called */
55      static int x = 50;
56
57      printf( "\nlocal static x is %d on entering b\n", x );
58      x++;
59      printf( "local static x is %d on exiting b\n", x );
60  } /* end function useStaticLocal */
61
62  /* function useGlobal modifies global variable x during each call */
63  void useGlobal( void )
64  {
65      printf( "\nglobal x is %d on entering c\n", x );
66      x *= 10;
67      printf( "global x is %d on exiting c\n", x );
68  } /* end function useGlobal */
```

**Program Output**

```
local x in outer scope of main is 5  
local x in inner scope of main is 7  
local x in outer scope of main is 5
```

```
local x in a is 25 after entering a  
local x in a is 26 before exiting a
```

```
local static x is 50 on entering b  
local static x is 51 on exiting b
```

```
global x is 1 on entering c  
global x is 10 on exiting c
```

```
local x in a is 25 after entering a  
local x in a is 26 before exiting a
```

```
local static x is 51 on entering b  
local static x is 52 on exiting b
```

```
global x is 10 on entering c  
global x is 100 on exiting c  
local x in main is 5
```

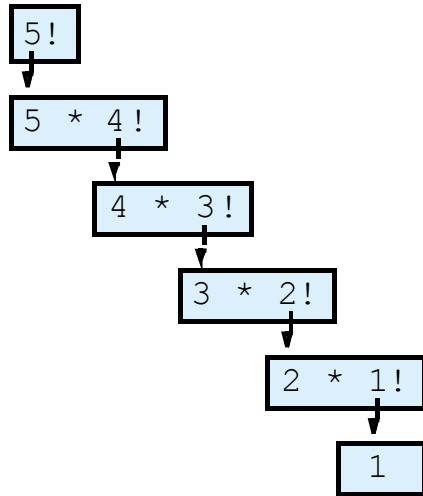
## 5.13 Rekurzija

- Rekurzivne funkcije
  - Funkcije mogu pozivati same sebe
  - Mogu riješiti samo bazni slučaj
  - Podijelimo problem na
    - Šta možemo uraditi
    - Šta ne možemo uraditi
      - Ako ovaj dio podsjeća na originalni problem
      - Funkcija startuje novu kopiju same sebe (rekurzivni korak - recursion step) da bi riješila problem
  - Ako je bazni slučaj riješen
    - Upotrebimo rješenje baznog slučaja, vraćamo se unazad (odmotavamo rekurziju) i time rješavamo polazni problem

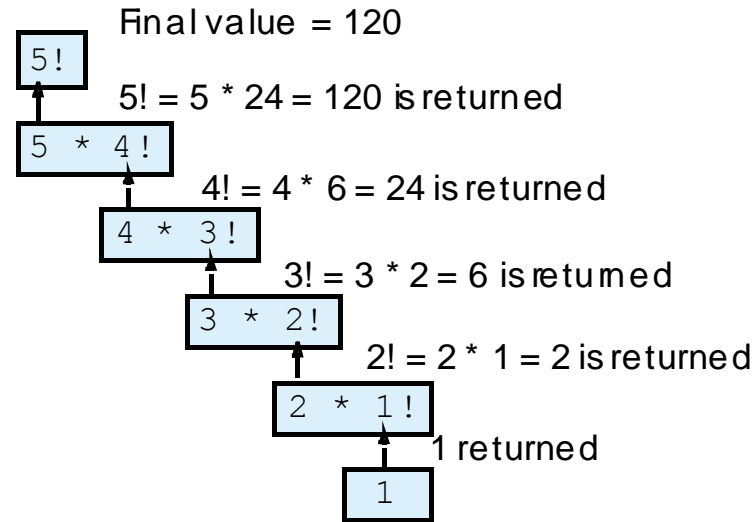
## 5.13 Rekurzija

- Primjer: faktorijel
  - $5! = 5 * 4 * 3 * 2 * 1$
  - Uočite da je
    - $5! = 5 * 4!$
    - $4! = 4 * 3! \dots$
  - Može se riješiti rekurzivno
  - Riješiti bazni slučaj ( $1! = 0! = 1$ ), pa onda odmotavamo
    - $2! = 2 * 1! = 2 * 1 = 2;$
    - $3! = 3 * 2! = 3 * 2 = 6;$

## 5.13 Recursion



(a) Sequence of recursive calls



(b) Values returned from each recursive call.

**fig05\_14.c (Part 1  
of 2)**

```
1  /* Fig. 5.14: fig05_14.c
2      Recursive factorial function */
3  #include <stdio.h>
4
5  long factorial( long number ); /* function prototype */
6
7  /* function main begins program execution */
8  int main()
9  {
10     int i; /* counter */
11
12     /* loop 10 times.  During each iteration, calculate
13         factorial( i ) and display result */
14     for ( i = 1; i <= 10; i++ ) {
15         printf( "%2d! = %ld\n", i, factorial( i ) );
16     } /* end for */
17
18     return 0; /* indicates successful termination */
19
20 } /* end main */
21
```



**fig05\_14.c (Part 2  
of 2)**

```
22 /* recursive definition of function factorial */
23 long factorial( long number )
24 {
25     /* base case */
26     if ( number <= 1 ) {
27         return 1;
28     } /* end if */
29     else { /* recursive step */
30         return ( number * factorial( number - 1 ) );
31     } /* end else */
32
33 } /* end function factorial */
```

```
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
```

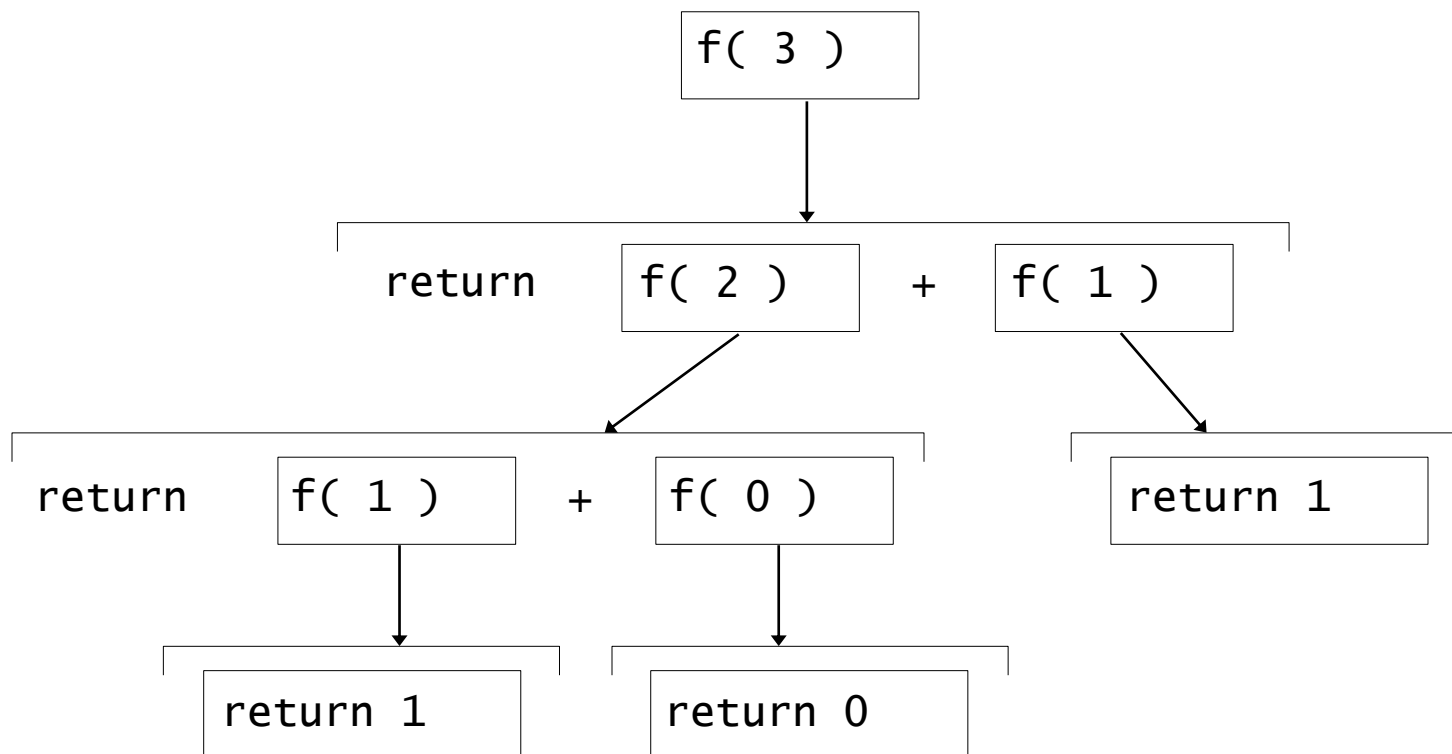
## 5.14 Primjer rekurzije: Fibonačijevi brojevi

- Fibonačijev niz: 0, 1, 1, 2, 3, 5, 8...
  - Svaki broj je suma prethodna dva
  - Rekurzivna definicija:
    - $\text{fib}(n) = \text{fib}(n - 1) + \text{fib}(n - 2)$
  - Kod za funkciju `fibonacci`

```
long fibonacci( long n )
{
    if (n == 0 || n == 1)  // base case
        return n;
    else
        return fibonacci( n - 1) +
            fibonacci( n - 2 );
}
```

## 5.14 Primjer rekurzije: Fibonačijevi brojevi

- Skup rekurzivnih poziva funkcije `fibonacci`



**fig05\_15.c (Part 1  
of 2)**

```
1  /* Fig. 5.15: fig05_15.c
2      Recursive fibonacci function */
3  #include <stdio.h>
4
5  long fibonacci( long n ); /* function prototype */
6
7  /* function main begins program execution */
8  int main()
9  {
10     long result; /* fibonacci value */
11     long number; /* number input by user */
12
13     /* obtain integer from user */
14     printf( "Enter an integer: " );
15     scanf( "%ld", &number );
16
17     /* calculate fibonacci value for number input by user */
18     result = fibonacci( number );
19
20     /* display result */
21     printf( "Fibonacci( %ld ) = %ld\n", number, result );
22
23     return 0; /* indicates successful termination */
24
25 } /* end main */
26
```

**fig05\_15.c (Part 2  
of 2)**

```
27 /* Recursive definition of function fibonacci */
28 long fibonacci( long n )
29 {
30     /* base case */
31     if ( n == 0 || n == 1 ) {
32         return n;
33     } /* end if */
34     else { /* recursive step */
35         return fibonacci( n - 1 ) + fibonacci( n - 2 );
36     } /* end else */
37
38 } /* end function fibonacci */
```

Enter an integer: 0  
Fibonacci( 0 ) = 0

Enter an integer: 1  
Fibonacci( 1 ) = 1

Enter an integer: 2  
Fibonacci( 2 ) = 1

Enter an integer: 3  
Fibonacci( 3 ) = 2

Enter an integer: 4  
Fibonacci( 4 ) = 3

**Program Output**

**Program Output  
(continued)**

Enter an integer: 5

Fibonacci( 5 ) = 5

Enter an integer: 6

Fibonacci( 6 ) = 8

Enter an integer: 10

Fibonacci( 10 ) = 55

Enter an integer: 20

Fibonacci( 20 ) = 6765

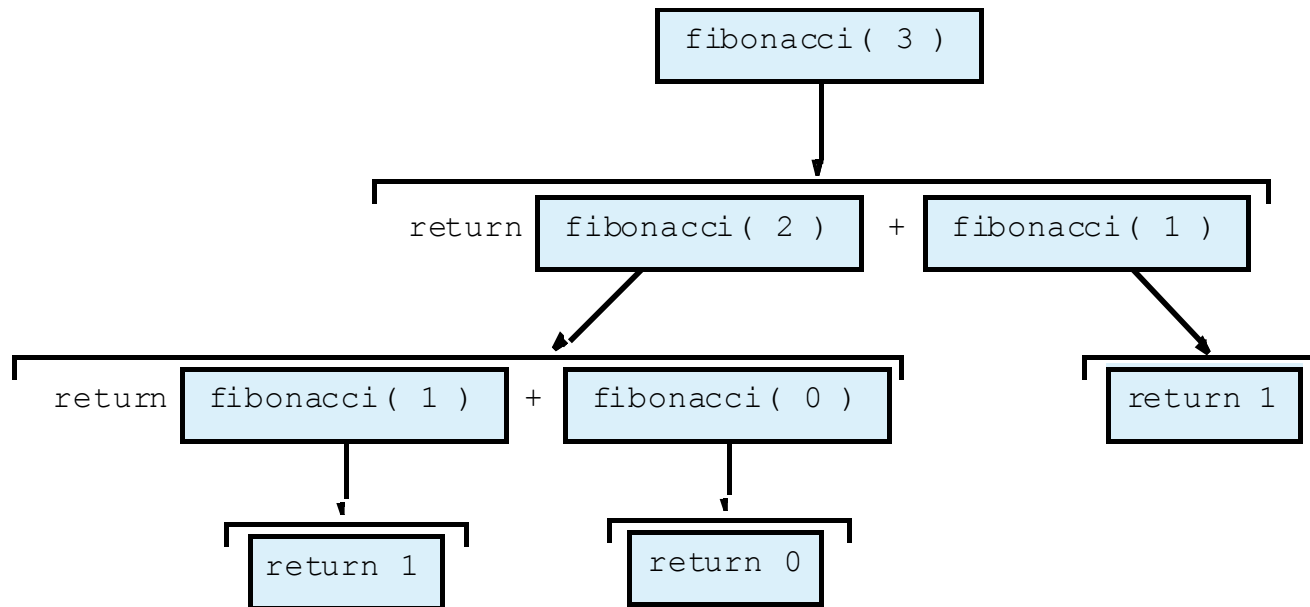
Enter an integer: 30

Fibonacci( 30 ) = 832040

Enter an integer: 35

Fibonacci( 35 ) = 9227465

## 5.14 Primjer rekurzije: Fibonačijevi brojevi



## 5.15 Rekurzija vs. Iteracija

- Repeticija
  - Iteracija: eksplicitna petlja (loop)
  - Rekurzija: ponavljanje poziva funkcije
- Kraj rada
  - Iteracija : uslov petlje je netačan
  - Rekurzija : bazni slučaj je prepoznat
- I jedan i drugi metod mogu imati beskonačne petlje
- Ravnoteža
  - Izaberite između performansi (iteracija) i elegancije i kratkoće (rekurzija)



## 5.15 Rekurzija vs. Iteracija

Lekcija	Primjeri rekurzije
<i>Lekcija 5</i>	Factorial function Fibonacci functions Greatest common divisor Sum of two integers Multiply two integers Raising an integer to an integer power Towers of Hanoi Recursive main Printing keyboard inputs in reverse Visualizing recursion
<i>Lekcija 6</i>	Sum the elements of an array Print an array Print an array backwards Print a string backwards Check if a string is a palindrome Minimum value in an array Selection sort Quicksort Linear search Binary search
<i>Lekcija 7</i>	Eight Queens Maze traversal
<i>Lekcija 8</i>	Printing a string input at the keyboard backwards
<i>Lekcija 12</i>	Linked list insert Linked list delete Search a linked list Print a linked list backwards Binary tree insert Preorder traversal of a binary tree Inorder traversal of a binary tree Postorder traversal of a binary tree
<b>Fig. 5.17</b> Pregled rekurzivnih programa u knjizi.	