

# Uvod u programiranje

- predavanja -

listopad 2019.

---

## 2. Programski jezik C

# Temeljni elementi jezika C

## Struktura C programa

# Struktura C programa

- C program se sastoji od deklaracija i definicija funkcija (imenovanih blokova), deklaracija i definicija varijabli i direktiva pretprocesoru
  - razlika između pojmova *deklaracija* i *definicija* bit će objašnjena kasnije. Za sada će se koristiti samo pojam *definicija*.
- složena naredba ili blok (imenovani ili neimenovani) može obuhvaćati deklaracije i definicije varijabli, ostale naredbe (*statement*) i neimenovane blokove
- svaka naredba mora završavati znakom ;
  - terminator: oznaka da na tom mjestu naredba završava (i može se, ako treba, početi pisati sljedeća)
  - blok NE završava znakom ; tj. iza znaka } ne stavlja se ;

# Primjer

```
#include <stdio.h>
```

direktiva pretprocesoru

```
int main(void) {
```

imenovani blok (funkcija)

```
    int n, rez;
```

definicija varijabli

```
    scanf("%d", &n);
```

naredba (*statement*)

```
    // izracunaj apsolutnu vrijednost
```

```
    if (n < 0) {
```

početak neimenovanog bloka

```
        rez = -1 * n;
```

```
    }
```

kraj neimenovanog bloka

```
    else {
```

početak neimenovanog bloka

```
        rez = n;
```

```
    }
```

kraj neimenovanog bloka

```
    printf("Ulaz: %d Rezultat: %d", n, rez);
```

```
    return 0;
```

```
}
```

# C je jezik slobodnog formata

- standard ne propisuje stil pisanja
  - mjesto početka naredbe u retku je proizvoljno, umetnute praznine nemaju specijalno značenje
  - dopušteno je napisati više naredbi u istom retku ili jednu naredbu u više redaka

```
...  
int n, rez; scanf("%d", &n);  
...  
printf("Ulaz: %d Rezultat: %d"  
      , n  
      , rez  
      );  
...
```

- međutim, poželjno je uredno pisanje, odnosno umetanje praznina i praznih redova na odgovarajućim mjestima

# Primjer

- Što nije u redu s ovim programom?

```
#include <stdio.h>
int main(

void
) { int n
, rez
; scanf(
"%d", &n); // izracunaj apsolutnu vrijednost
                                if ( n < 0 )
{rez = -1 * n
; }
else                                {rez =
n; }printf("Ulaz: %d Rezultat: %d", n
                                , rez); return 0; }
```

# Temeljni elementi jezika C

Ključne riječi

Uporaba velikih i malih slova

# Ključne riječi

- ključne riječi su predefinirani identifikatori koji za prevodioca imaju posebno značenje. ISO/IEC 9899:2011 (C11) propisuje sljedeće 44 ključne riječi:

auto	extern	short	while
break	float	signed	_Alignas
case	for	sizeof	_Alignof
char	goto	static	_Atomic
const	if	struct	_Bool
continue	inline	switch	_Complex
default	int	typedef	_Generic
do	long	union	_Imaginary
double	register	unsigned	_Noreturn
else	restrict	void	_Static_assert
enum	return	volatile	_Thread_local



# Uporaba velikih i malih slova

- C prevodilac razlikuje velika i mala slova
  - za imena varijabli, ključne riječi i ostale identifikatore mora se koristiti propisani oblik slova (veliko/malo)

```
#Include <STDIO.h>
INT Main(Void) {
    Return 0;
}
```

- svaka riječ u prethodnom programu je napisana neispravno. Prevodilac u tom programu neće moći prepoznati:
  - pretprocesorsku naredbu include
  - datoteku stdio.h
  - ključne riječi int, void, return
  - funkciju main

# Temeljni elementi jezika C

Komentari

# Komentari

- komentari nemaju utjecaj na izvršavanje programa. Mogu se ugraditi na dva načina, na bilo kojem mjestu u izvornom kôdu:
  - komentar koji započinje s dvije kose crte proteže se do kraja retka

```
// podaci o studentu  
int godRod;    // godina rođenja  
int godUpis;   // godina upisa na FER
```

- komentar koji započinje dvoznakom /\* i završava dvoznakom \*/ može se protezati i kroz više redaka
- komentari ovog oblika ne smiju se ugnježdjavati

```
/* funkcija izracunava najveći zajednički djelitelj za  
   zadane cijele pozitivne brojeve m i n  
*/  
int najvećiDjelitelj(int m, int n) {  
    ...  
}
```

# Temeljni elementi jezika C

Funkcija *main*

# Glavna funkcija (*main*)

```
int main(void) {  
    ...  
    return 0;  
}
```

- glavna funkcija predstavlja mjesto na kojem počinje izvršavanje C programa
  - svaki program mora sadržavati točno jednu funkciju *main*
  - *int* ispred *main* znači da funkcija u pozivajući program (u ovom slučaju operacijskom sustavu) vraća cijeli broj (*integer*). S time povezana naredba *return* u pozivajući program vraća cijeli broj
    - za sada: operacijskom sustavu uvijek vratiti cijeli broj nula, kao što je prikazano u primjeru
  - *void* znači da funkcija *main* ne prima niti jedan argument
  - početak i kraj bloka naredbi, koji predstavlja tijelo funkcije, označeni su vitičastim zagradama `{ i }`

# Temeljni elementi jezika C

Varijable i konstante

# Variable

- općenito: promjenljiv podatak (lat. *variabilis* - promjenljiv)
- u programiranju: prostor u memoriji računala, unaprijed zadane i nepromjenjive veličine, kojem je pri definiciji dodijeljeno ime i tip i čiji se sadržaj može mijenjati, npr. naredbom pridruživanja ili učitavanjem vrijednosti s tipkovnice.

```
...  
int main(void) {  
    int m, n;  
    ...  
    ...  
    n = 128;  
    scanf("%d", &m);  
    ...  
    m = 1000;  
    scanf("%d", &n);  
    ...  
}
```

sadržaj varijabli ovog trenutka nije siguran, varijable sadrže "garbage value", "smeće". Kažemo: varijable još nisu inicijalizirane (*uninitialized*)

sadržaj varijable n postavljen je pridruživanjem  
sadržaj varijable m postavljen je učitavanjem vrijednosti s tipkovnice

sadržaj varijabli može se ponovo promijeniti

# Definicija varijable

```
int n, rez;
```

- prethodnom naredbom definirane su dvije cjelobrojne varijable u koje je moguće pohranjivati isključivo cijele brojeve. Kažemo: varijable su tipa int

```
float x, y, z;  
float v;
```

- prethodnim naredbama definirane su realne varijable x, y, z i v u koje je moguće pohranjivati isključivo realne brojeve. Kažemo: varijable su tipa float (naziv je izveden iz pojma *floating point*)
- za sada će se koristiti samo tipovi int i float. Ostali podržani tipovi podataka bit će objašnjeni kasnije
- varijabla se može definirati na bilo kojem mjestu u bloku, ali obavezno prije nego se prvi puta koristi



# Definicija varijable uz inicijalizaciju

```
int k, m, n;  
m = 3;  
n = 3;
```

k, m, n ovog trenutka sadrže "smeće"  
m sadrži 3  
n sadrži 3  
k trenutčno sadrži "smeće"

- Naredba za definiciju varijable može sadržavati tzv. *inicijalizator* kojim se već pri definiciji postavlja početna vrijednost varijable
  - inicijalna vrijednost mora se navesti pojedinačno za svaku varijablu koju se želi inicijalizirati

```
...  
int main(void) {  
    int k, m = 3, n = 5;  
    ...
```

m sadrži 3; n sadrži 5  
k trenutčno sadrži "smeće"

# Imena varijabli

- imena varijabli (i svi drugi identifikatori, npr. imena funkcija) sastavljena su od slova, znamenki i znakova podcrtavanja \_
  - ime ne smije započeti:
    - znakom
    - s dva znaka podcrtavanja
    - znakom podcrtavanja i velikim slovom
  - ime ne smije biti jednako niti jednoj ključnoj riječi
  - prema konvenciji, za tvorbu imena varijabli prvenstveno se koriste mala slova, uz eventualni dodatak nekoliko velikih slova (vidjeti kasnije tzv. *camelCase*)
  - primjeri neispravnih imena varijabli

novi+datum	x1/1	x\$	rezultat!
float	int	return	void
__suma	_Produkt	1.suma	1produkt

# Imena varijabli

- duljina imena je proizvoljna, ali treba voditi računa o sljedećem:
  - kod nekih prevodioca moguća su ograničenja u broju značajnih znakova. Standard zahtijeva samo to da najmanje 31 prvih znakova imena bude značajno. Stoga je moguće da neki prevodioci neće moći međusobno razlikovati sljedeća imena varijabli:
    - `prosjecna_ocjena_na_predmetu_upro_2018_godine`
    - `prosjecna_ocjena_na_predmetu_upro_2019_godine`
  - preduga ili prekratka imena smanjuju preglednost ili otežavaju pisanje programa. Korištenje imena koje odražava značenje varijable bitno unapređuju jasnoću programa
    - umjesto predugih (gore) ili prekratkih imena (npr. `p1`, `p2`) bolje je:  
`upro_prosj_2018`, `upro_prosj_2019` (*snake\_case* oblik) ili  
`uproProsja2018`, `uproProsja2019` (*camelCase* oblik)

# Konstante

- slično varijablama, konstante također imaju svoje tipove. Tip konstante ovisi o formi u kojoj je napisana

```
float x, y;
```

```
int m, n;
```

```
x = 1.f;
```

```
y = 2;
```

```
m = 3;
```

```
n = 3.5f;
```

vrijednost realne konstante pridružuju se realnoj varijabli

vrijednost cjelobrojne konstante pridružuju se realnoj varijabli  
(prije pridruživanja obavlja se konverzija)

vrijednost cjelobrojne konstante pridružuje se cjelobrojnoj varijabli

vrijednost realne konstante pridružuju se cjelobrojnoj varijabli  
(prije pridruživanja obavlja se konverzija)

- zašto se realnim konstantama na kraj dodaje slovo f bit će objašnjeno u predavanjima o tipovima i načinima pohrane podataka

# Temeljni elementi jezika C

Direktive pretprocesoru

# Direktive pretprocesoru - *#include*

```
#include <stdio.h>
```

- uputa (direktiva) pretprocesoru: u program prije prevođenja uključiti sadržaj datoteke <stdio.h>
  - <stdio.h> sadrži deklaracije i definicije koje su potrebne da bi program na ispravan način mogao koristiti, između ostalog, funkcije printf i scanf (funkcije za čitanje i pisanje)
  - zaključak: na početak svakog programa koji će koristiti funkcije scanf ili printf treba ugraditi direktivu #include <stdio.h>

# Primjer

- `gcc -E prog1.c > prog1.i`

izvorni kôd

prog1.c

```
#include <stdio.h>

int main(void) {
    int n, rez;
    scanf("%d", &n);

    // izracunaj apsolutnu vrijednost
    if (n < 0) {
        rez = -1 * n;
    } else {
        rez = n;
    }

    printf("Ulaz: %d Rezultat: %d", n, rez);
    ...
}
```

pretprocesirani izvorni kôd

prog1.i

```
...
__attribute__((__cdecl__))
__attribute__((__nothrow__))
int printf (const char *, ...);
...
__attribute__((__cdecl__))
__attribute__((__nothrow__))
int scanf (const char *, ...);
...

int main(void) {
    int n, rez;
    scanf("%d", &n);

    if (n < 0) {
        rez = -1 * n;
    } else {
        ...
    }
}
```

# Direktive pretprocesoru - *#define*

```
#define PI 3.14159f
```

- uputa (direktiva) pretprocesoru: tijekom faze pretprocesiranja, svaku pojavu riječi PI u izvornom kodu zamijeniti s 3.14159f
  - time je definirana *simbolička konstanta*
  - prema konvenciji, imena konstanti pišu se velikim slovima

```
#include <stdio.h>
#define PI 3.14159f

int main(void) {
    float r;    // polumjer kruga
    scanf("%f", &r);
    printf("Povrsina kruga: %f\n",
           r * r * PI);
    printf("Opseg kruga: %f",
           2 * r * PI);
    return 0;
}
```

"ispis novog reda", skok u novi red na zaslonu

Ne ovako!  
float pi;  
pi = 3.14159f;



# Primjer

- gcc -E prog2.c > prog2.i

izvorni kôd

prog2.c

```
#include <stdio.h>
#define PI 3.14159f

int main(void) {
    float r;    // polumjer kruga
    scanf("%f", &r);
    printf("Povrsina kruga: %f\n",
           r * r * PI);
    printf("Opseg kruga: %f",
           2 * r * PI);

    return 0;
}
```

Ne ovako!

```
#define DVA 2
...
printf("Opseg kruga: %f", DVA * r * PI);
...
```

pretprocesirani izvorni kôd

prog2.i

```
...
__attribute__((__cdecl__))
__attribute__((__nothrow__))
int printf (const char *, ...);
...

int main(void) {
    float r;
    scanf("%f", &r);
    printf("Povrsina kruga: %f\n",
           r * r * 3.14159f);
    printf("Opseg kruga: %f",
           2 * r * 3.14159f);

    return 0;
}
```

# Primjer izvršavanja prethodnog programa

5↵

Povrsina kruga: 78.539801

Opseg kruga: 31.415920

- crvenom bojom prikazuju se znakovi koje je tipkovnicom upisao korisnik
- oznaka ↵ će se koristiti uvijek kada će u primjerima trebati naglasiti da se na nekom mjestu "ispisuje skok u novi red" ili da je pritisnuta tipka Enter (odnosno Return)

# Temeljni elementi jezika C

Izrazi

# Izrazi

- Izraz (*expression*) je kombinacija operatora, operanada (konstante, varijable, pozivi funkcija, izrazi, ...) i zagrada, koja po evaluaciji daje rezultat
  - izraz pridruživanja
  - aritmetički izraz
  - relacijski izraz
  - logički izraz
- Izrazi se mogu ugrađivati u druge, složenije izraze, koristiti kao argumenti funkcija i dijelovi nekih naredbi

# Izrazi

- Primjeri izraza (uz pretpostavku da su a i b cjelobrojne varijable):

256	aritmetički izraz
a	aritmetički izraz
a + 11	aritmetički izraz
(a + 1) * (b - 1)	aritmetički izraz
a = 20	izraz pridruživanja
b = a + 3	aritmetički izraz unutar izraza pridruživanja
a <= 10	relacijski izraz
(a <= 10) && (b == 0)	dva relacijska izraza unutar logičkog izraza

- Primjeri korištenja izraza na mjestu argumenta funkcije i dijela naredbe (u ovom slučaju naredbe return)

```
printf("%d", (a + 1) * (b - 1));  
return a - b;
```

# Operator i izraz pridruživanja

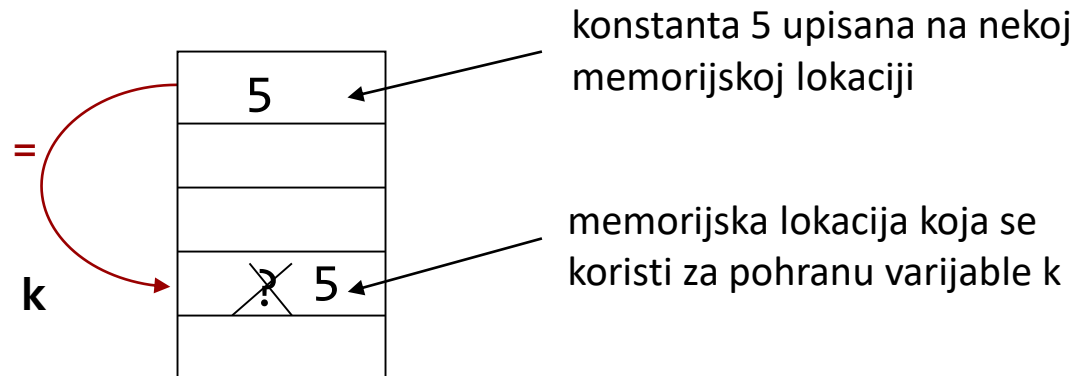
- Operator pridruživanja se koristi za pridruživanje vrijednosti

- simbol u pseudo-kodu      $:=$

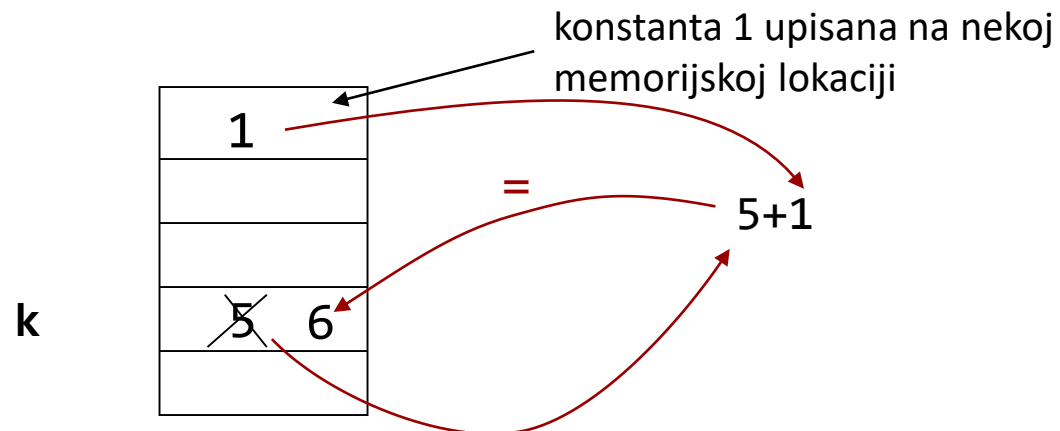
- u C-u      $=$

- Primjeri:

- `int k;`  
`k = 5;`



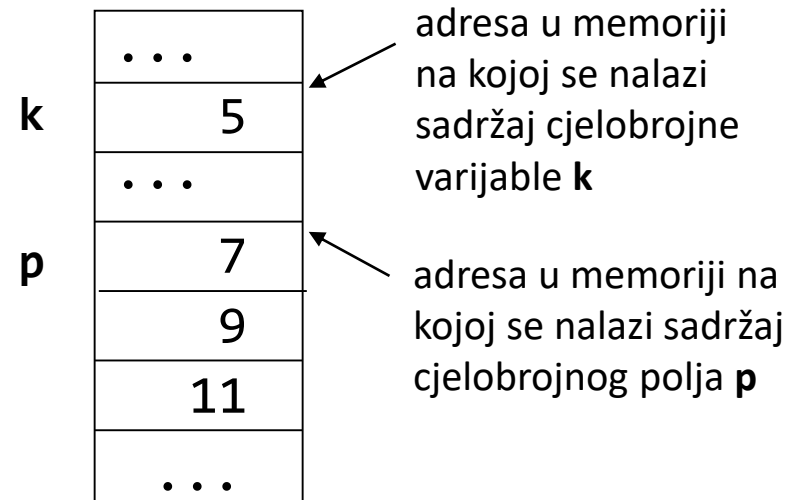
- `k = k + 1;`



# Locator value, lvalue

- *Locator value* (*Lvalue*, *L-value*, *lvalue*) je izraz koji predstavlja (određuje, *designates*) objekt koji se u memoriji nalazi na određenoj adresi (ili "ima određenu adresu")
  - npr. imena varijabli **k** i **p** su *lvalue* jer predstavljaju objekte (u ovom slučaju cjelobrojnu varijablu, odnosno cjelobrojno polje) čije su vrijednosti (*value*) pohranjene na određenim adresama u memoriji

```
...  
int k = 5;  
int p[3] = {7, 9, 11};  
...
```



# Modifiable lvalue, non-modifiable lvalue

- *lvalue* koji predstavlja objekt čija se vrijednost može promijeniti naziva se izmjenljivi *lvalue* (*modifiable lvalue*)
  - npr. ime varijable (jednostavnog tipa) je *modifiable lvalue*

```
...  
int k = 5;  
...  
k = 10;
```

vrijednost objekta može se promijeniti pomoću *lvalue* k

- npr. ime varijable tipa polje je *non-modifiable lvalue*. Sadržaj polja nije moguće promijeniti korištenjem (samo) imena varijable p1

```
...  
int p1[3] = {7, 9, 11}, p2[3] = {6, 8, 10};  
...  
p1 = p2;
```

**neispravno:** p1 je *non-modifiable lvalue*



# Lijeva strana u izrazu pridruživanja

- Lijeva strana (lijevi operand) izraza pridruživanja mora biti *modifiable locator value (modifiable lvalue)*.

```
int m, n, k;  
int p[3] = {7, 9, 11};  
n = 15 + 3;  
m = m + n + 1;  
k + 1 = m + 1;  
7 = m;  
p = 10;
```

*n je modifiable lvalue*

*m je modifiable lvalue*

**neispravno:** *k + 1 nije lvalue*

**neispravno:** *konstanta 7 nije lvalue*

**neispravno:** *p nije modifiable lvalue*

Prema (danas zastarjelom) tumačenju iz *Kernighan and Ritchie: The C Programming Language*, pojam *Lvalue (left-value)* odnosi se na vrstu izraza koji je dopušteno koristiti na lijevoj strani izraza pridruživanja.

# Desna strana u izrazu pridruživanja

- Na desnoj strani izraza pridruživanja može se nalaziti bilo koji izraz (može, ali ne mora biti *lvalue*). Npr. varijabla, konstanta, aritmetički izraz, poziv funkcije, itd.
  - vrijednost izraza (*value of expression*) na desnoj strani izračunava se (evaluira) i postavlja kao nova vrijednost objekta kojeg predstavlja *lvalue* na lijevoj strani izraza pridruživanja

```
int m, n;  
n = 15 * 3 - 100;  
m = abs(n) / 2;
```

Prema danas zastarjelom, ali u literaturi često korištenom tumačenju, pojam *Rvalue* (*right-value*) odnosi se na vrstu izraza koji je dopušteno koristiti na desnoj strani izraza pridruživanja.

# Rezultat izraza pridruživanja

- izraz pridruživanja se prvenstveno koristi za pridruživanje vrijednosti, ali kao i svaki drugi izraz, "po evaluaciji" daje rezultat.
  - taj rezultat najčešće se ne koristi, kao u sljedećem primjeru: aritmetički izraz daje rezultat 30, izrazom pridruživanja 30 se pridružuje varijabli m, a konačni rezultat izraza pridruživanja jest opet vrijednost 30 (vrijednost koja je upravo pridružena). Vrijednost izraza pridruživanja ostala je neiskorištena.

```
int m;  
m = 15 * 2;
```

- u sljedećem primjeru, rezultat izraza pridruživanja će se iskoristiti. Konstanta 5 pridružuje se varijabli m, konačni rezultat izraza pridruživanja je 5. Ta vrijednost se ispisuje na zaslon.

```
int m;  
printf("%d", m = 5);
```

# Višestruko pridruživanje

- činjenica da izraz pridruživanja "po evaluaciji" daje rezultat može se lijepo iskoristiti kod višestrukog pridruživanja

```
int a, b, c;
```

```
...
```

```
c = 3;
```

```
...
```

```
a = b = c * 5;
```

Redoslijed obavljanja:  $a = (b = (c * 5));$

- izračunata je vrijednost  $c * 5$  (u primjeru 15) i pridružena varijabli b.
  - rezultat izraza pridruživanja (15) pridružuje se varijabli a
  - rezultat izraza pridruživanja (15) se nema za što iskoristiti, pa se odbacuje
- Je li sljedeći izraz s višestrukim pridruživanjem ispravan?

```
a = b + 2 = c;
```

Neispravno jer  $b + 2$  nije lvalue

# Prioritet i asocijativnost operatora

- Redoslijed obavljanja operacija u izrazima ovisi o
  - **prioritetu operatora**, ako se radi o operatorima različitog prioriteta
$$a + b * c \qquad (a + b) * c \text{ ili } a + (b * c)$$
prioritet operatora određuje da se prvo obavlja operacija  $b * c$
  - **asocijativnosti operatora**, ako se radi o operatorima jednakog prioriteta
$$a / b * c \qquad (a / b) * c \text{ ili } a / (b * c)$$
asocijativnost operatora određuje da se prvo obavlja operacija  $a / b$

Prioritet ↑ Viši ↓ Niži	Operator	Asocijativnost operatora
	* /	$L \rightarrow D$
	binarne operacije + -	$L \rightarrow D$
	=	$D \rightarrow L$


# Primjer

- Tijekom evaluiranja izraza

$$a = b = c * 5$$

treba obaviti nekoliko operacija. Kojim redoslijedom će se operacije obaviti?

- Operator množenja ima viši prioritet od operatora pridruživanja
  - prvo će se obaviti operacija množenja  $a = b = (c * 5)$
- Operatori pridruživanja imaju jednaki prioritet. Redoslijed obavljanja određen je asocijativnošću operatora
  - asocijativnost operatora  $D \rightarrow L$  znači da se operandi i operacije grupiraju od desna prema lijevo, dakle


$$(a = (b = c * 5))$$

za operaciju  $c * 5$  je već prije odlučeno da se obavlja prva, stoga se ovdje ne razmatra.

# Primjer

- Cjelobrojne varijable *a* i *b* inicijalizirati na vrijednosti 14 i -9. Program treba ispisati njihove vrijednosti, zamijeniti vrijednosti u varijablama te ponovo ispisati vrijednosti varijabli na zaslon. Ispis na zaslonu treba izgledati ovako:

a=14, b=-9  
a=-9, b=14

```
#include <stdio.h>
int main(void) {
    int a = 14, b = -9, pom;
    printf("a=%d, b=%d\n", a, b);
    pom = a;
    a = b;
    b = pom;
    printf("a=%d, b=%d", a, b);
    return 0;
}
```

a	b	pom
14	-9	?
14	-9	?
14	-9	14
-9	-9	14
-9	14	14
-9	14	14

# Aritmetički operatori i izrazi

- Ovdje su navedeni samo osnovni aritmetički operatori

Operator	Značenje	Operandi
+	zbrajanje	int, float
-	oduzimanje	int, float
*	množenje	int, float
/	dijeljenje	int, float
%	ostatak cjelobrojnog dijeljenja	int

- aritmetički operator i pridruženi operandi čine aritmetički izraz
- operandi mogu biti varijable, konstante i složeniji aritmetički izrazi



# Djelovanje operatora na cjelobrojne operande

- uočiti: ako su oba operanda cjelobrojna
  - rezultat je cjelobrojan
  - operacija se obavlja u cjelobrojnoj domeni (naročito važno za operaciju dijeljenja)

```
int a = 11, b = 2;
```

aritmetički izraz	rezultat
$a + b$	13
$a - b$	9
$a * b$	22
$a / b$	5
$a \% b$	1

# Djelovanje operatora na realne operande

- uočiti: ako je barem jedan operand realnog tipa
  - rezultat je realan
  - operacija se obavlja u realnoj domeni (naročito važno za operaciju dijeljenja)
  - operator *modulo* ne smije se koristiti

```
int a = 11;  
float b = 2.f;
```

**aritmetički izraz**

**rezultat**

`a + b`

13.0

`a - b`

9.0

`a * b`

22.0

`a / b`

5.5

`a % b`

prevodilac odbija prevesti program

# Prioritet aritmetičkih operatora

- operatori množenja, dijeljenja i ostatka cjelobrojnog dijeljenja imaju veći prioritet od operatora zbrajanja i oduzimanja

$$a + b * c \equiv a + (b * c)$$

$$b * c + a \equiv (b * c) + a$$

- ako aritmetički operatori imaju jednak prioritet (npr. množenje, dijeljenje i ostatak cjelobrojnog dijeljenja), tada se operacije obavljaju s lijeva na desno

$$a / b * c \equiv (a / b) * c$$

$$x / a + b * c + d * e \equiv ((x / a) + (b * c)) + (d * e)$$

- ako pretpostavljeni redoslijed obavljanja operacija treba promijeniti, koristiti okrugle zagrade, npr.

$$(a + b) * c$$

$$x / ((a + b) * (c + d) * e)$$

# Primjer

- Odrediti veličinu i tip rezultata sljedećih aritmetičkih izraza

```
int m = 11;
float x = 2.f;

m / x           5.5, float
m / 2           5, int
m / 2 * x       10.0, float
x * m / 2       11.0, float
x * (m / 2)     10.0, float
m + 1 / x       11.5, float
(m + 1) / x     6.0, float
```

# Relacijski operatori i izrazi

- relacijskim operatorima testira se odnos među operandima
  - relacijski operator i pridruženi operandi čine relacijski izraz
  - operandi u relacijskim izrazima mogu biti i realni i cjelobrojni (također i drugih tipova). Operandi mogu biti varijable, konstante i složeniji aritmetički izrazi
  - relacijski izraz je jednostavan logički izraz, atomni sud, koji se evaluira kao istinit ili lažan. Npr.

```
...  
if (n < 0)  
...
```

- relacijski izraz  $n < 0$  evaluirat će se kao istinit ako je vrijednost varijable  $n$  manja od nule, inače, izraz se evaluira kao lažan

# Relacijski operatori

- navedeni su svi relacijski operatori

Operator	Značenje
>	veće
<	manje
>=	veće ili jednako
<=	manje ili jednako
==	jednako
!=	različito

- naročito paziti na sljedeće: operatori = i == imaju posve različito značenje. Uporabu neispravne vrste operacije prevodilac neće moći utvrditi: program će se prevesti i "raditi", ali neispravno.

# Logički operatori i izrazi

- logičkim operatorima grade se složeni sudovi
  - logički operator i pridruženi operandi čine logički izraz
  - kao operandi u logičkim izrazima mogu se koristiti relacijski izrazi i složeni logički izrazi. Npr.

```
...  
if (n >= 10 && n <= 20)  
...
```

- logički izraz `n >= 10 && n <= 20` evaluirat će se kao istinit ako i samo ako je rezultat relacijskog izraza `n >= 10` istinit i rezultat relacijskog izraza `n <= 20` je istinit

# Logički operatori

- navedeni su svi logički operatori

Operator	Značenje
!	logičko NE
&&	logički I
	logički ILI



# Prioritet operatora

- obratiti pažnju na prioritet aritmetičkih, relacijskih, logičkih operatora i operatora pridruživanja. Koristiti zagrade gdje je potrebno

Prioritet operatora
!
* / %
+ -
< <= > >=
== !=
&&
=

- Primjer:
  - `!x > 20` će se evaluirati kao `(!x) > 20`
  - ispravno je `!(x > 20)`

# Primjeri logičkih izraza

- Rezultat logičkog izraza je istina
  - ako je vrijednost realne varijable  $x$  unutar intervala  $[3, 5]$  ili unutar intervala  $[7, 9]$ 

$$(x \geq 3.f \ \&\& \ x \leq 5.f) \ || \ (x \geq 7.f \ \&\& \ x \leq 9.f)$$
  - ako je vrijednost realne varijable  $x$  unutar intervala  $[3, 5]$  i vrijednost varijable  $y$  nije unutar intervala  $[7, 9]$ 

$$x \geq 3.f \ \&\& \ x \leq 5.f \ \&\& \ !(y \geq 7.f \ \&\& \ y \leq 9.f) \ \text{ili}$$
$$x \geq 3.f \ \&\& \ x \leq 5.f \ \&\& \ (y < 7.f \ || \ y > 9.f)$$

Transformacija prvog u drugi izraz i obrnuto: De Morganova pravila
  - ako je vrijednost realne varijable  $x$  pozitivna ili barem za 10 veća i od vrijednosti varijable  $y$  i od vrijednosti varijable  $z$ 

$$x > 0.f \ || \ x \geq y + 10.f \ \&\& \ x \geq z + 10.f$$
  - za vježbu: provjerite nedostaju li negdje zagrade ili se još neke zagrade smiju ukloniti

# Temeljni elementi jezika C

Naredbe za promjenu programskog slijeda  
Jednostavni oblik selekcije

# Jednostavni oblik selekcije

- jedna od naredbi za kontrolu toka programa

```
if (n < 0) {  
    rez = -1 * n;  
} else {  
    rez = n;  
}
```

izvršavanje programa nastavlja se ovdje, bez obzira na rezultat logičkog izraza

- ako se logički izraz (u zagradama iza ključne riječi if) izračuna kao istina, obavljaju se sve naredbe unutar prvog para vitičastih zagrada
- inače se obavljaju sve naredbe unutar drugog para vitičastih zagrada (iza ključne riječi else)
- dio naredbe koji započinje ključnom riječi else ne mora se uvijek navesti

# Temeljni elementi jezika C

Funkcije za čitanje iz standardnog ulaza i  
pisanje na standardni izlaz

# Učitavanje vrijednosti iz standardnog ulaza

```
scanf("%d", &n);
```

- poziv funkcije za učitavanje vrijednosti s tipkovnice
  - prvi argument je format - niz znakova unutar dvostrukih navodnika koji sadrži jednu ili više tzv. *konverzijskih specifikacija*
    - "%d" je format koji omogućuje čitanje jedne cjelobrojne vrijednosti
    - "%f" je format koji omogućuje čitanje jedne realne vrijednosti
    - "%d %d %f %d" je format koji omogućuje čitanje (redom) dvije cjelobrojne, jedne realne i još jedne cjelobrojne vrijednosti
  - jedan ili više sljedećih argumenata predstavljaju adrese varijabli u koje funkcija treba upisati vrijednosti pročitane s tipkovnice. Adresa varijable dobije se tako da se ispred imena varijable napiše znak &. Tipovi varijabli moraju po broju i tipu odgovarati konverzijskim specifikacijama navedenim u prvom argumentu poziva funkcije.

# Primjer

```
int i, j, k;  
float x;  
scanf("%d %d %f %d", &i, &j, &x, &k);
```

- ako se tipkovnicom upišu vrijednosti: `37 5 3.14 2↵`
- nakon izvršavanja funkcije scanf, varijable i, j, x, k će redom sadržavati vrijednosti 37, 5, 3.14 i 2

oznakom ↵ se naglašava da se je na tom mjestu na zaslon "ispisan skok u novi red" ili da je tijekom unosa podataka preko tipkovnice pritisnuta tipka Enter, odnosno Return

# Ispis vrijednosti na standardni izlaz

```
printf("Ulaz: %d Rezultat: %d", n, rez);
```

- poziv funkcije za ispis vrijednosti na zaslon
  - prvi argument je format - niz znakova unutar dvostrukih navodnika koji sadrži kombinaciju znakova koji će se ispisati i/ili konverzijskih specifikacija
    - %d je konverzijska specifikacija za ispis cjelobrojne vrijednosti
    - %f za ispis realne vrijednosti
    - %e za ispis realne vrijednosti u znanstvenoj notaciji
  - daljnji argumenti (ako su navedeni) predstavljaju vrijednosti koje će se ispisati na mjestima na kojima su u formatu navedene konverzijske specifikacije.
    - vrijednosti mogu biti varijable, konstante, ali i složeniji izrazi
    - vrijednosti moraju po broju i tipu odgovarati konverzijskim specifikacijama navedenim u prvom argumentu poziva funkcije.



# Prilagodba širine ispisa cijelog broja

- konverzijskom specifikacijom moguće je utjecati na širinu ispisa cijelog broja
  - ako se širina ne navede ili je navedena širina manja od potrebnog broja znamenki, ispisat će se minimalni broj znamenki potreban za ispravan prikaz broja
  - ako je navedena širina veća od minimalno potrebnog broja znamenki, ispred znamenki će se ispisati odgovarajući broj praznina

```
printf("%d,%5d,%2d", 128, -12, 256);
```

```
128, -12,256
```

# Prilagodba širine i preciznosti ispisa realnog broja

- kod ispisa realnog broja moguće je prilagoditi ukupnu širinu ispisa i broj ispisanih znamenki (preciznost) iza decimalne točke

```
printf("%f,%f,%f\n", 4.5f, 1280.0f, -3.4555555f);
```

```
4.500000,1280.000000,-3.455556
```

%f          ukupna širina prema potrebi, 6 znamenki iza dec. točke, zaokružiti prema potrebi

```
printf("%5.2f,%12.4f,%.4f", 5.128f, -256.128f, 12.4555555f);
```

```
5.13,      -256.1280,12.4556
```

%5.2f      min. ukupna širina 5 znakova, 2 znamenke iza dec. točke, zaokružiti prema potrebi

%12.4f     min. ukupna širina 12 znakova, 4 znamenke iza dec. točke, zaokružiti prema potrebi

%.4f       4 znamenke iza decimalne točke, a ispred minimalno koliko je potrebno

# Ispis realnog broja u znanstvenoj notaciji

- specifikacija %e vrlo je slična specifikaciji %f. Jedina razlika je u tome što se broj ispisuje u obliku koji sadrži eksponent broja 10

```
printf("%e,%e,%e\n", 4.5f, 1280.0f, -3.4555555f);
```

```
4.500000e+000,1.280000e+003,-3.455556e+000
```

%e          ukupna širina prema potrebi, 6 znamenki iza dec. točke, zaokružiti prema potrebi

```
printf("%12.2e,%15.4e,%.4e", 5.128f, -256.128f, 12.4555555f);
```

```
5.13e+000,      -2.5613e+002,1.2456e+001
```

%12.2e    min. ukupna širina 12 znakova, 2 znamenke iza dec. točke, zaokružiti prema potrebi

%15.4e    min. ukupna širina 15 znakova, 4 znamenke iza dec. točke, zaokružiti prema potrebi

%.4e      4 znamenke iza decimalne točke, a ispred minimalno koliko je potrebno

# Raspon i preciznost

Cjelobrojni i realni tip

# Raspon cjelobrojnih varijabli i konstanti

- zbog načina pohrane cijelih brojeva u računalu, moguće je prikazati tek ograničeni skup cijelih brojeva
- detaljnije će se pohrana cijelih brojeva razmatrati kasnije, za sada je dovoljno znati sljedeće:
  - cjelobrojni tip podatka (varijabla i konstanta tipa int) može se koristiti za prikaz cijelih brojeva u intervalu  $[-2\ 147\ 483\ 648, 2\ 147\ 483\ 647]$ . Pokušaj korištenja cijelih brojeva izvan tog intervala može dovesti do neočekivanih rezultata, npr.

```
int min = -2147483648, max = 2147483647, rez1, rez2;  
rez1 = min - 1;  
rez2 = max + 1;  
printf("-2147483648 - 1 > %d\n", rez1);  
printf("2147483647 + 1 > %d", rez2);
```

```
-2147483648 - 1 > 2147483647  
2147483647 + 1 > -2147483648
```

# Raspon realnih varijabli i konstanti

- zbog načina pohrane realnih brojeva u računalu, moguće je prikazati samo sljedeće realne brojeve
  - brojeve iz intervala  $[-3.402823 \cdot 10^{38}, -1.401298 \cdot 10^{-45}]$  ili
  - brojeve iz intervala  $[1.401298 \cdot 10^{-45}, 3.402823 \cdot 10^{38}]$  ili
  - realni broj 0.0
- detaljnije će se pohrana realnih brojeva razmatrati kasnije
- pokušaj korištenja realnih brojeva izvan tog intervala može dovesti do neočekivanih rezultata

```
float x = 3.4e38f * 1.1f;  
float y = 1.401298e-45f / 2.f;  
printf("%f\n%e\n", x, y);
```

```
inf  
0.000000e+000
```

# Preciznost realnih varijabli i konstanti

- zbog načina pohrane realnih brojeva, nije moguće potpuno točno pohraniti sve realne brojeve iz prethodno navedenih intervala
  - zapravo, beskonačno mnogo brojeva iz navedenih intervala nije moguće prikazati potpuno točno
  - detaljnije će se preciznost pohrane realnih brojeva razmatrati kasnije

```
float x = 0.0625f;  
float y = 0.0624f;  
printf("%20.18f\n%20.18f", x, y);
```

```
0.062500000000000000  
0.062399998307228088
```

# Preciznost realnih varijabli i konstanti

- ponekad se samo zbog zaokruživanja pri ispisu (ako se navede dovoljno malen broj znamenki koje treba ispisati iza decimalne točke) čini da je broj pohranjen posve točno

```
float x = 0.01f;  
printf("%f\n", x);  
printf("a zapravo je pohranjeno:\n", x);  
printf("%20.18f", x);
```

0.010000

a zapravo je pohranjeno:

0.009999999776482582



# Korisne matematičke funkcije

# Korisne matematičke funkcije

- navedene funkcije se mogu upotrijebiti na mjestu operanda u aritmetičkom ili relacijskom izrazu
- argumenti ovih funkcija mogu biti varijable, konstante ili aritmetički izrazi
- argumenti i rezultati sljedećih funkcija su realni brojevi *dvostruke preciznosti*, međutim za sada se to može zanemariti.

Funkcija	Značenje
<code>sqrt(x)</code>	$\sqrt{x}$
<code>pow(x, y)</code>	$x^y$
<code>sin(x)</code>	sinus (rad)
<code>cos(x)</code>	kosinus (rad)
<code>tan(x)</code>	tangens (rad)
<code>log(x)</code>	$\ln x$
<code>log10(x)</code>	$\log_{10} x$

# Primjer

```
#include <stdio.h>
#include <math.h>      // Učitati ovaj include!

int main(void) {
    printf("sqrt(20.25) = %f\n", sqrt(20.25f));
    printf("pow(6.25, -0.5) = %f\n", pow(6.25f, -0.5f));
    printf("sin(3.1415926/2) = %f\n", sin(3.1415926f/2));
    printf("ln(2.7182818) = %f\n", log(2.7182818f));
    printf("log(1000.0) = %f\n", log10(1000.f));

    return 0;
}
```

```
sqrt(20.25) = 4.500000
pow(6.25, -0.5) = 0.400000
sin(3.1415926/2) = 1.000000
ln(2.7182818) = 1.000000
log(1000.0) = 3.000000
```

# Primjer

```
#include <stdio.h>
#include <math.h>


int main(void) {
    float a, b, c;
    printf("Upisite katete > ");
    scanf("%f %f", &a, &b);

    c = sqrt(pow(a, 2.f) + pow(b, 2.f));
    printf("Hipotenuza = %f\n", c);
    return 0;
}
```

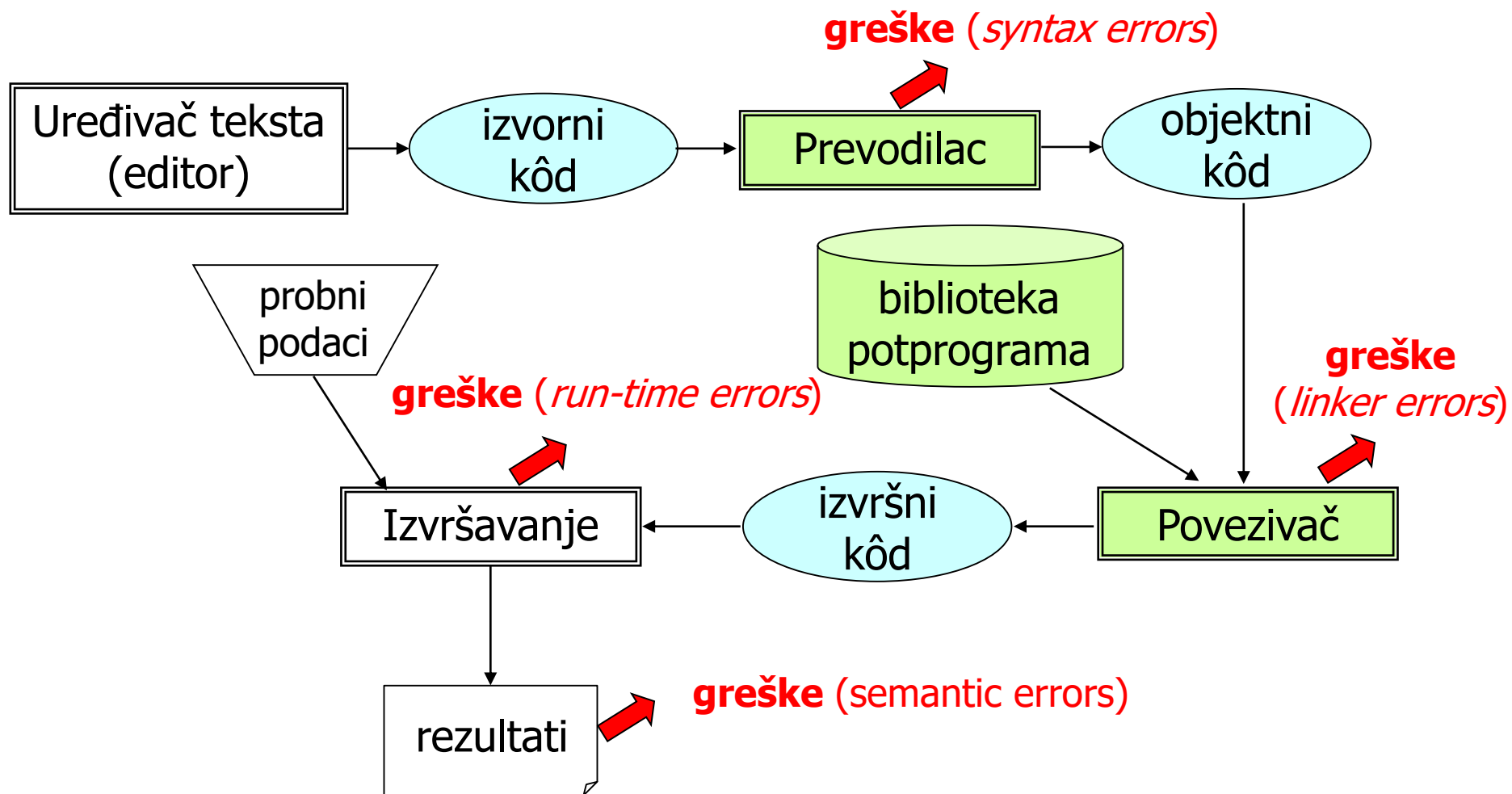
```
Upisite katete > 6 8↵
Hipotenuza = 10.000000↵
```

# Programiranje, prevodilac, vrste grešaka

# Postupci izrade manjih programa

1. Razvoj algoritma (npr. pomoću pseudo-koda)
  2. Implementacija algoritma u programskom jeziku (kodiranje)
  3. Prevođenje programa u objektni kôd (kompilacija)
    - prevodilac dojavljuje formalne greške
  4. Ispravljanje formalnih grešaka
  5. Povezivanje objektnog kôda (*linking*)
    - poveziivač (*linker*) dojavljuje greške povezivanja
  6. Ispravljanje grešaka povezivanja
  7. Izvršavanje programa uz primjenu testnih podataka
    - uočavaju se greške izvršavanja i semantičke greške
  8. Ispravljanje grešaka izvršavanja i semantičkih grešaka
- 

# Postupci izrade manjih programa



# Postupci izrade manjih programa

- Implementacija algoritma u programskom jeziku (kodiranje)
  - obavlja se upisivanjem izvornog kôda u datoteku pomoću uređivača teksta (*editor*). Npr. notepad, vi, ...
  - ili pomoću uređivača teksta ugrađenog u radnu okolinu programera, npr. VSCode, Eclipse, MS Visual Studio, ...
- Prevođenje izvornog programskog koda u objektni program
  - obavlja se prevodiocem (*compiler*)
    - prevodilac (i pretprocesor) otkrivaju i dojavljuju sintaktičke (pravopisne, formalne) greške
    - programer ispravlja izvorni kôd i ponovo pokreće prevođenje
    - postupak se ponavlja dok god prevodilac dojavljuju formalne greške



# Postupci izrade manjih programa

- Povezivanje (*linking*) prevedenog objektnog kôda u izvršni (apsolutni) programski kôd
  - obavlja se poveziivačem (*linker*)
    - povezuje se objektni kôd iz jednog ili više prevedenih modula i kôd iz programskih biblioteka
    - poveziivač otkriva i dojavljuje greške povezivanja
    - programer ispravlja izvorni kôd i ponovo pokreće prevođenje i povezivanje
    - ponavlja se dok god poveziivač dojavljuje greške povezivanja

# Postupci izrade manjih programa

- Testiranje programa
  - obavlja se definiranjem skupova ulaznih podataka i očekivanih rezultata, a zatim izvršavanjem programa, npr. pokretanjem programa iz ljuske operacijskog sustava
  - pri tome se uočavaju
    - greške koje uzrokuju abnormalni prekid programa (*run-time errors*) i
    - neispravni rezultati koji upućuju na postojanje semantičkih grešaka (*semantic errors*)
    - programer ispravlja izvorni kôd i ponovo pokreće prevođenje, povezivanje i testiranje
    - ponavlja se dok god se tijekom izvršavanja programa događaju abnormalni prekidi rada programa ili se dobivaju neispravni rezultati

# Primjer - razvoj malog programa

```
#include <stdio.h>

int main(void) {
    int n; rez;
    scan("%d", -n);

    // izracunaj apsolutnu vrijednost
    if (n < 0) {
        rez = --n;
    } else {
        rez = n;
    }

    print("Ulaz: %d Rezultat: %d", n, rez);
    return 0;
}
```

# Primjer - formalne greške

```
C:\upro>gcc -std=c11 -Wno-all -o prog1.exe prog1.c
prog1.c:1:2: error: invalid preprocessing directive #include
#include <stdio.h>
  ^~~~~~
prog1.c: In function 'main':
prog1.c:4:11: error: 'rez' undeclared (first use in this function)
    int n; rez;
           ^~
prog1.c:4:11: note: each undeclared identifier is reported only once for
each function it appears in
```

- ispraviti greške u izvornom kodu i ponoviti prevođenje

```
#include <stdio.h>

int main(void) {
    int n; rez;
    scan("%d", -n);
    ...
}
```

# Primjer - greške povezivanja

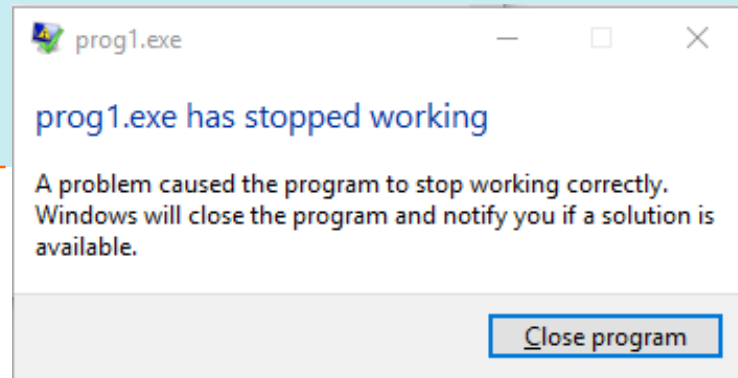
```
C:\upro>gcc -std=c11 -Wno-all -o prog1.exe prog1.c
C:\Users\user\AppData\Local\Temp\cc6jyeMA.o:prog1.c:(.text+0x20):
undefined reference to `scan'
C:\Users\user\AppData\Local\Temp\cc6jyeMA.o:prog1.c:(.text+0x41):
undefined reference to `print'
collect2.exe: error: ld returned 1 exit status
```

- ispraviti greške u izvornom kodu i ponoviti prevođenje i povezivanje

```
...
    int n = 0, rez;
    scan("%d", -n);
...
    print("Ulaz: %d Rezultat: %d", n, rez);
...
```

# Primjer - greške tijekom izvršavanja

```
C:\upro>gcc -std=c11 -Wno-all -o prog1.exe prog1.c  
C:\upro>prog1.exe  
-11  
  
C:\upro>
```



- kako otkriti greške koje prevodilac i povezičavač nisu uspjeli prepoznati?

# Traženje grešaka

- Opcije za traženje grešaka tijekom izvršavanja i semantičkih grešaka
  - čitati program i razmišljati
  - dodavati pomoćne ispise na strateška mjesta u programu

```
#include <stdio.h>
int main(void) {
    int n; rez;
    scanf("%d", &n);
    printf("Procitan je n=%d\n", n);

    // izracunaj apsolutnu vrijednost
    if (n < 0) {
        printf("n je manji od nule");
        rez = -n;
        printf("izracunat je rez=%d", rez);
    } else {
        printf("n je veci od nule");
        rez = n;
    }
    printf("Ulaz: %d Rezultat: %d", n, rez);
    return 0;
}
```

# Traženje grešaka

- Bolje: koristiti specijalizirane programe za traženje grešaka – (*debuggers*)
  - u naredbenoj liniji (ljusci operacijskog sustava)
  - integrirano u programerskoj okolini (VSCode, Eclipse, ...)
  - u nastavku je prikazan samo mali skup mogućnosti programa gdb koji se koristi u kombinaciji s prevodiocem gcc



# Debugger - traženje mjesta prekida programa

```
C:\upro>gcc -std=c11 -ggdb -o prog1.exe prog1.c
C:\upro>gdb prog1.exe
(gdb) start
Temporary breakpoint 1 at 0x401497: file prog1.c, line 5.
Starting program: C:\upro/prog1.exe
Temporary breakpoint 1, main () at prog1.c:5
5          scanf("%d", -n);
(gdb) frame                koja naredba će se izvršiti kao sljedeća?
#0  main () at prog1.c:5
5          scanf("%d", -n);
(gdb) next                izvrši dotičnu sljedeću naredbu
-11                        upis vrijednosti za n

Program received signal SIGSEGV, Segmentation fault.
0x7593ff0b in ungetwc () from C:\WINDOWS\SysWOW64\msvcrt.dll
```

- očito, program je prekinut nakon što je pozvana funkcija scanf
- sada više nije teško uočiti grešku. Ispraviti i ponoviti testiranje

```
scanf("%d", -n); → scanf("%d", &n);
```

# Debugger - traženje semantičkih grešaka

```
C:\upro>gcc -std=c11 -ggdb -o prog1.exe prog1.c
C:\upro>prog1.exe
-11
Ulaz: -11 Rezultat: 4194432
```

- Program nije prekinut, ali rezultat je neispravan. Gdje bi mogla biti (semantička) greška?

```
scanf("%d", &n);           je li vrijednost varijable n sada ispravno učitana?

// izracunaj apsolutnu vrijednost
if (n < 0) {                je li ispravno evaluiran ovaj relacijski izraz? Po čemu ćemo znati?
    rez = --n;              je li u varijablu rez upisan ispravan rezultat?
} else {
    rez = n;
}
```

# Debugger - traženje semantičkih grešaka

```
C:\upro>gcc -std=c11 -ggdb -o prog1.exe prog1.c
C:\upro>gdb prog1.exe
(gdb) start                pokreni program i zaustavi se već na prvoj naredbi
...
(gdb) watch rez            nadgledaj varijablu rez - kad god se promijeni, zaustavi program
Hardware watchpoint 2: rez
(gdb) continue            nastavi s izvršavanjem
Continuing.
-11                        upis vrijednosti za n
Hardware watchpoint 2: rez  promijenila se varijabla rez. U nastavku piše gdje i kako:
Old value = 4194432
New value = -12
0x004014c6 in main () at prog1.c:9
9          rez = --n; Aha! Tu si! Mora biti da nešto nije u redu s rez = --n;
```

# Slijedi: testiranje, testiranje, testiranje, ...

- Ispraviti grešku, ponovo prevesti, povezati i testirati s različitim ulaznim podacima

```
C:\upro>gcc -std=c11 -Wall -pedantic-errors -o prog1.exe prog1.c
C:\upro>prog1.exe
-11
Ulaz: -11 Rezultat: 11
C:\upro>prog1.exe
11
Ulaz: 11 Rezultat: 11
C:\upro>prog1.exe
0
Ulaz: 0 Rezultat: 0
```