Uvod u programiranje

- predavanja -

listopad 2025.

3. Programski jezik C

Vodič „02 Prije trećeg predavanja”

- Očekuje se da ste savladali (na razini na kojoj je obrađeno u vodiču):
  - printf
  - scanf
  - Raspon cijelih i realnih brojeva
  - Preciznost realnih brojeva
  - Jednostavni oblik if-else

- Pitanja?

scanf – dodatni komentar

- Kako sve možemo unijeti vrijednosti koje se učitavaju putem scanf?
  - Tipkovnica:
    - Jedan po jedan
    - U grupama
    - Kombinacija
  - Redirekcijom
- Kako na unos utječu (višestruke) praznine?
- Probajmo:

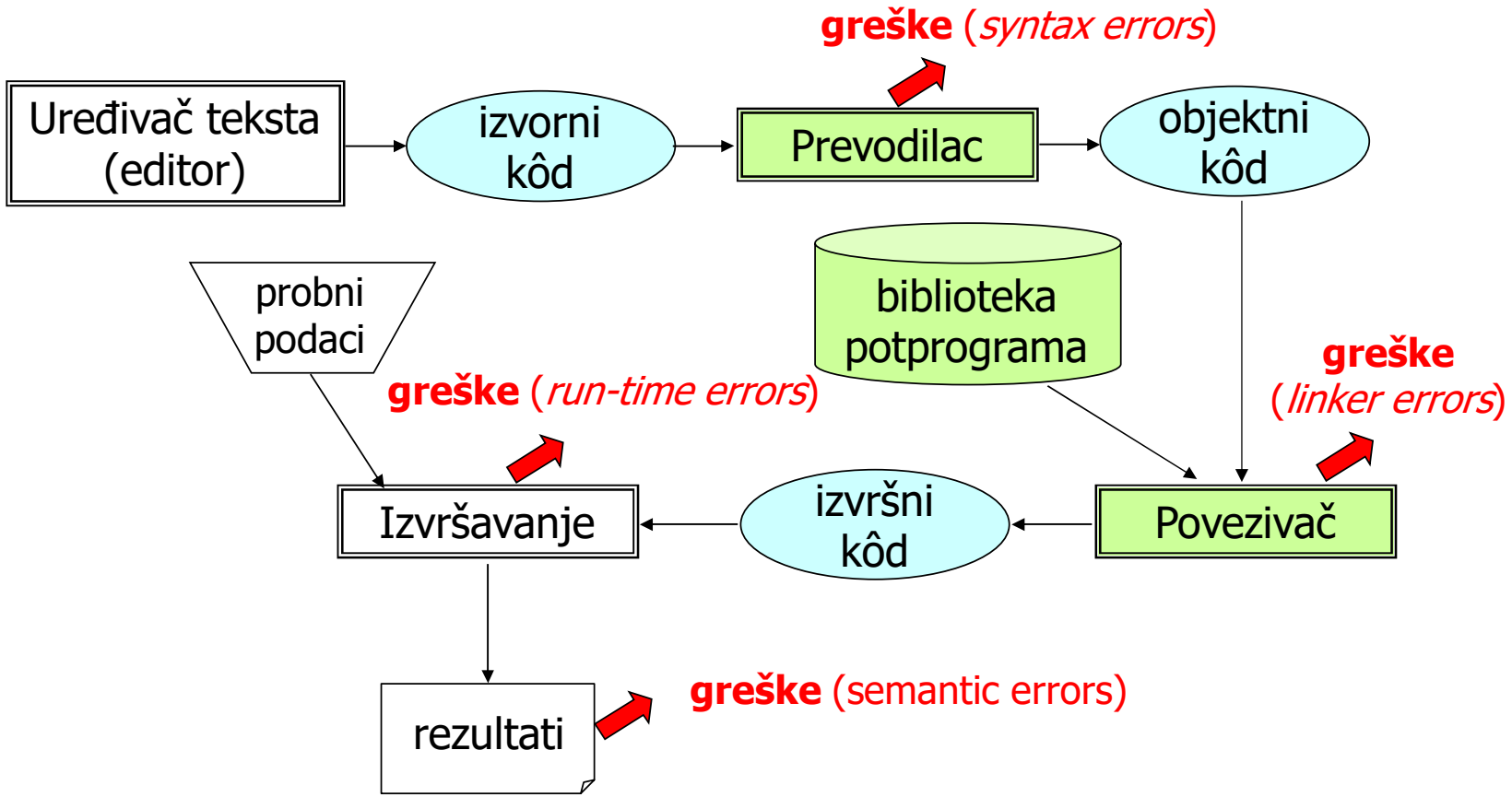
```
#include <stdio.h>
int main(void) {
    int n1, n2, n3, n4, n5;
    scanf("%d %d", &n1, &n2);
    scanf("%d%d", &n3, &n4);
    scanf("%d", &n5);
    printf("\nUcitani su: \t%d\t%d\t%d\t%d\t%d", n1, n2, n3, n4, n5);
    return 0;
}
```

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*)
   - povezivač (*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 povezivačem (*linker*)
    - povezuje se objektni kôd iz jednog ili više prevedenih modula i kôd iz programskih biblioteka
    - povezivač otkriva i dojavljuje greške povezivanja
    - programer ispravlja izvorni kôd i ponovo pokreće prevođenje i povezivanje
    - ponavlja se dok god povezivač 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

```
#inklude <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 #inklude
 #inklude <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

```
#inklude <stdio.h>

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

Primjer - greške povezivanja

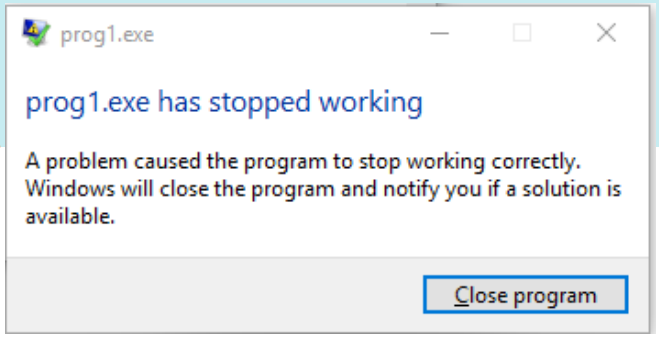
```
C:\upro>gcc -std=c11 -Wno-all -o prog1.exe prog1.c
C:\Users\user\AppData\Local\Temp\cc6jyeMA.o:prog1.c:(.text+0x20):
undefined reference to `scan'
C:\Users\user\AppData\Local\Temp\cc6jyeMA.o:prog1.c:(.text+0x41):
undefined reference to `print'
collect2.exe: error: ld returned 1 exit status
```

- 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 povezivač 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: -12
```

- 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
```

Zadatak

- Napišite program koji učitava cijeli broj koji predstavlja kalendarsku godinu
- Provjerite je li godina prijestupna ili nije te ispišite odgovarajuću poruku, npr.:
  - Godina 1900 NIJE prijestupna.
  - Godina 2000 JE prijestupna.
  - Godina 2004 JE prijestupna.

- Godina je prijestupna ako je djeljiva s 400 ili je djeljiva s 4 i nije djeljiva s 100.

Kontrola toka programa, kontrolne strukture

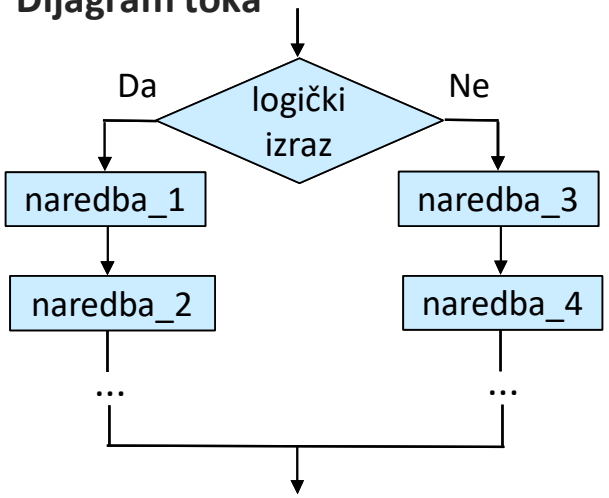
- jedan od najvažnijih aspekata programiranja jest mogućnost upravljanja redoslijedom izvršavanja naredbi (*flow control*): propisivanje koja naredba će se izvršiti u sljedećem koraku izvršavanja programa. U tu svrhu koriste se kontrolne programske strukture (*programming control structures*)
  - **selekcija**
    - jednostrana (*if ... then ...*)
    - dvostrana (*if ... then ... else ...*)
    - skretnica (*case ... then ..., case ... then ..., case ... then ..., ...*)
  - **petlja**
    - ponavljanje naredbi dok je zadovoljen uvjet (*condition-controlled*)
    - ponavljanje naredbi određeni broj puta (*count-controlled*)

Dvostrana selekcija

if else

Dvostrana selekcija - if - else

Dijagram toka



C program - sintaksa

```
if (logički_izraz)
   naredba_1;    jedna naredba!
else
   naredba_2;    jedna naredba!
```

Što ako treba izvršiti više od jedne naredbe?
Rješenje: koristiti **složenu naredbu** (sljedeći slajd)

Ne treba nam uvijek **dvostrana selekcija**, za neke probleme koristimo **jednostranu selekciju** i onda se `else` dio izostavlja

C program - primjer

```
if (ocjena > 1)
   printf("Ispit je polozen!");
else
   printf("Ispit nije polozen!");
```

Složena naredba (*compound statement*)

- **problem:** u naredbi if dopušteno je navođenje samo jedne naredbe koja će se izvršiti kada je rezultat logičkog izraza istina. Što učiniti ako je potrebno izvršiti više od jedne naredbe?
- **rješenje:** koristiti složenu naredbu (blok, blok naredbi)
  - jedna ili više naredbi omeđenih vitičastim zagradama
  - može se koristiti na mjestima gdje je prema sintaksi predviđena samo jedna naredba (u naredbama if, while, ...)
  - složena naredba se nikad ne terminira znakom ;

Sintaksa složene naredbe

```
{
   naredba_1;
   naredba_2;
   ...
}
```

Primjer korištenja složene naredbe

```
...
if (ocjena > 1) {
   printf("Ispit je polozen.");
   brojPol = brojPol + 1;
}
...
```

Kada pisati zagrade? primjer: C program

```c
#include <stdio.h>

int main(void) {
   float x;

   scanf("%f", &x);
   if (x != 0)
      printf("%f %f", x, 1/x);
   return 0;
}
```

```c
#include <stdio.h>

int main(void) {
   float x;

   scanf("%f", &x);
   if (x != 0) {
      printf("%f %f", x, 1/x);
   }
   return 0;
}
```

- obje varijante su ispravne, međutim
  - iako se radi o samo jednoj naredbi koju treba obaviti kada je uvjet zadovoljen, **preporuča se** koristiti oblik složene naredbe, tj. naredbu omeđiti vitičastim zagradama
    - smanjuje se mogućnost logičke pogreške koja može nastati nepažljivim prepravljanjem programa, kao što je prikazano u sljedećem primjeru

Kada pisati zagrade? primjer: C program

```c
#include <stdio.h>

int main(void) {
   float x, recip;

   scanf("%f", &x);

   if (x != 0)
      printf("%f", x);
      recip = 1 / x;
      printf(" %f", recip);
   return 0;
}
```

```c
#include <stdio.h>

int main(void) {
   float x, recip;

   scanf("%f", &x);

   if (x != 0) {
      printf("%f", x);
      recip = 1 / x;
      printf(" %f", recip);
   }
   return 0;
}
```

- program na lijevoj strani je neispravan! Prepravljajući lijevu varijantu programa iz prethodnog primjera programer je previdio da je trebalo dodati vitičaste zagrade i time napravio teško uočljivu logičku pogrešku

Kada pisati zagrade? Primjeri

- U nekim slučajevima, radi kompaktnosti i preglednosti programa, prikladno je za jednu naredbu ne koristiti vitičaste zagrade . Npr.

```
// varijablu a postavi na njenu apsolutnu vrijednost
if (a < 0) a = -1 * a;
```

- Oprez!

```
if (a < 0); a = -1 * a;
```

  - C prevodilac ovo smatra (sintaktički) ispravnim. "Ništa" terminirano znakom `;` naziva se nul-naredba (*null-statement*). Sasvim očekivano, nul-naredba ne obavlja nikakvu akciju
    - analizirati posljedice ove logičke greške

Primjer

- Programski zadatak
  - Na zaslon ispisati poruku `Upisite cijeli broj >`
  - Učitati cijeli broj s tipkovnice i ovisno o učitanoj vrijednosti ispisati jednu, obje ili niti jednu od sljedećih poruka:
    - Broj je pozitivan
    - Broj je paran
- Primjeri izvršavanja programa

```
Upisite cijeli broj > 12↵
Broj je pozitivan↵
Broj je paran↵
```

```
Upisite cijeli broj > 13↵
Broj je pozitivan↵
```

```
Upisite cijeli broj > -12↵
Broj je paran↵
```

```
Upisite cijeli broj > 0↵
Broj je paran↵
```

```
Upisite cijeli broj > -47↵
```

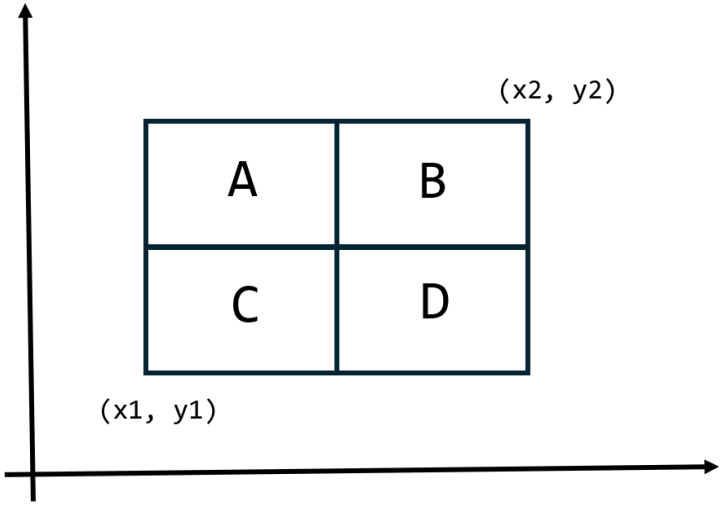
Koliko ima kombinacija ishoda? Hoćemo li ovdje koristiti dvostranu selekciju?

Rješenje

```c
#include <stdio.h>

int main(void) {
   int broj;

   printf("Upisite cijeli broj > ");
   scanf("%d", &broj);

   if (broj > 0) {
      printf("Broj je pozitivan\n");
   }

   if (broj % 2 == 0) {
      printf("Broj je paran\n");
   }
   return 0;
}
```

Zadatak

- Napisati program koji će za točku u ravnini napisati u kojem kvadrantu zadanog pravokutnika se nalazi.
- Potrebno je učitati:
  - koordinate donjeg lijevog i gornjeg desnog ugla pravokutnika
  - koordinate točke koja se ispituje
- Ovisno o tome gdje se točka nalazi ispisati:
  - Točka (…, …) se ne nalazi u pravokutniku
  - Točka (…, …) se nalazi u kvadrantu {A|B|C|D}



- Pretpostaviti da će koordinate pravokutnika biti ispravno zadane
- U slučaju da točka dodiruje više kvadranata (na granici je) ispisati bilo kojeg od njih

Prije sljedećeg predavanja

- Edgar:
  - Tutorial: **03. Prije četvrtog predavanja**
  - Public exams: **3. vježbe uz predavanja - osnove programskog jezika C**