# C PROGRAMSKI JEZIK

Dragan dr Dinu

# C PROGRAMSKI JEZIK



- Dennis Ritchie, razvijen 1969. – 1973.
- Jedan od najupotrebljavanijih jezika do danas
- Jezik treće generacije sa mogućnostima druge generacije
- Obezbeđuje programeru maksimalnu procesnu moć uz veliku odgovornost
- Programski jezik operatora i izraza
- Inspiracija mnogim drugim programskim jezicima
- Često upotrebljavan alat za pravljenje sistemskog softvera
- Mali programski jezik (mali broj naradbi) i brz
- Fleksibilan i prenosiv

# SLUŽBENE REČI



- **Samo 32 reči**
- **CASE SENSITIVE**

| | | | |
|---|---|---|---|
| auto | break | case | char |
| const | continue | default | do |
| double | else | enum | extern |
| float | for | goto | if |
| int | long | register | return |
| short | signed | sizeof | static |
| struct | switch | typedef | union |
| unsigned | void | volatile | while |

# ALFABET



**~ ! # % & * ( ) - _ + = [ ]**

**{ } : ; ' " ? < > / , \ |**

- ali i kombinacije

**== -- ++ >> <<**

- plus **SLUŽBENE REČI**

# OSNOVNE SINTAKSNE KATEGORIJE



- **CIFRA: 0 1 2 ... 9**
- **CEO BROJ:**

  **+3 3 -3 0012 156**
- **BROJ:**
  - **decimalna tačka a ne decimalni zarez !**

  **+7.12 7.12 -7.12 0012.89564**
- **SLOVO: A B C ... Z a b c ... z**
- **ZNAK: SLOVO ili CIFRA ili ! ili ~ ili ... ili ?**
- **KARAKTER:**

  **'a' '\n'**
- **TEKST:**

  **"Jos danas" "Dan za danom..."**

# STRUKTURA C PROGRAMA

# OSNOVNA STRUKTURA C PROGRAMA



- Linearna u jednom nivou
- Top-down razvoj
- Prevođenje u jednom prolazu
- Prvo definiši pa tek onda koristi !

DIREKTIVE PRETPROCESORU

SEKCIJA GLOBALNIH PODATAKA

PROTOTIPOVI drugih FUNKCIJA

```
DEFINICIJA main FUNKCIJE
{
....
}
```

```
DEFINICIJE drugih FUNKCIJA
{
....
}
```

# PRETPROCESOR ...



- **Pretprocesira izvorni C kod:**
  1. **Eliminiše trigrafe**
     **??[ postaje { a ??- postaje ~ ....**
  2. **Svođenje više blanko na jedan i eliminiše komentare**
     ***/*  */  ,  //***
  3. **Zamenjuje iskejp sekvence sa ASCII**
     **\n u 10 13, \a u 07 ...**
  4. **Identifikovanje i izvršavanje pretprocesorskih direktiva**

```
#include <stdio.h>
#include "naziv.h”
#define MaksimumPosetilaca 350
#define cetvrtina(vrednost) ((vrednost)/4)
#undef
#if ... #elif ... #else ... #endif
```

# PRETPROCESOR U PRIMERU ...



- #include
- Uključivanje zaglavlja datoteke
- Čita se i kopira sadržaj datoteke
- < ... > traži u standardnom include folderu
- " ... " traži u folderu u kojem se nalazi izvorni kod
- Standardni ulaz/izlaz stdio.h
- Funkcije opšte namene stdlib.h (radilo bi i bez stdlib)

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

int main()
{
   printf("Hello world!\n");
   return 0;
}
```

# ... PRETPROCESOR U PRIMERU



```
#define cetvrtina(vrednost) ((vrednost)/4)
...
x = cetvrtina(36);
...
x = ((36)/4);
x = 9;
```

```
#define MaksimumPosetilaca 350
...
x = MaksimumPosetilaca;
...
x = 350;
```

# MAIN FUNKCIJA



- **Početna tačka svakog programa**
- Može biti sa parametrima ili bez
- Postoji samo jedna **main** funkcija

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

int main()
{
    printf("Hello world!\n");
   return 0;
}
```

# ISKAZI I PROGRAMSKI BLOK



- C programski blok čine iskazi zatvoreni unutar para vitičastih zagrada **{ }**
- Iskaz je sekvenca jednog C izraza i simbola tačka-zarez **;**
- Izraz predstavlja poziv funkcije, neka operacija i sl.

```
{
   int a, int b = 113, float pi=3.14;
   printf("Hello world!\n");
}
```

# IDENTIFIKATORI …



- Fundamentalna stvar u programiranju
- Služe za imenovanje raznih stvari u programu
- Jako je važno razumeti kako se daju imena
- Dobar programer koristi i deklariše identifikatore na konzistentan način (uvek dosledno)
- Identifikator je sekvenca karaktera, brojeva i donje crte (’_’)
- Identifikator započinje slovom ili donjom crtom

```
N   n   _rate   x15   quite_a_long_name   HelloWorld
```

# … IDENTIFIKATORI



- U identifikatoru nisu dozvoljena prazna mesta (space)

```
ovoJeValidanIdentifikator
ovi nije Validan identifikator
```

- U C-u se pravi razlika između malih i velikih slova (case-sensitive)

`HelloWorld`, `helloworld`, `HELLOWORLD` i `hElloWorLD`

su sve validni identifikatori, ali različiti

- Rezervisane reči ne mogu biti identifikatori (`class`, `public`, `if`, `else`, `while` i ostale)

- Identifikatori koji se sastoje od više reči mogu koristiti crticu za njihovo razdvajanje ili se svako početno slovo reči sem kod prve piše veliko: `helloWorld`

# PROMENLJIVE

# PROMENLJIVE ...



- Program transformiše ulazne podatke u izlazne (rezultat)
- Mogu se koristiti i međurezultati
- Podaci se čuvaju u promenljivama, slično opštem broju u matematici
- Predstavljaju i mesto u memoriji
- Dodela vrednosti (nije isto što i izjednačavanje u matematici)

```
a = 5;        b = a + 13;        a = a + 2;
```

# ... PROMENLJIVE ...



- Dodela vrednosti (nije isto što i izjednačavanje u matematici), jer je promenljiva i mesto u memoriji

- Prvo se pročita stara vrednost, te se ona uvede u izraz, nakon čega se rezultat izraza upiše umesto stare vrednosti

```
a = 3;
a = a + 2;
```

```
a + 2
3 + 2
5
```

```
a = 5;
```

# ... PROMENLJIVE ...



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

int main()
{
   int a = 5;
   int b;

   printf("Promenljiva a ima vrednost: %d\n", a);

   b = a + 13;
   a = a + 2;

   printf("\nPromenljiva b=a+13 ima vrednost: %d\n", b);
   printf("\nPromenljiva a=a+2 ima vrednost: %d\n", a);

   return 0;
}
```

**Posle svega saberite a i b, rezultat sačuvajte u b i prikažite b**

# ... PROMENLJIVE ...



- Za svaku promenljivu definiše se:
  - **Naziv**, identifikator, preko koje se identifikuje i koristi, mora imati deskriptivnu komponentu da bi se lakše identifikovala i koristila
    - **C razlikuje mala i velika slova!**
  - **Tip podatka** koji se čuva u promenljivoj (ceo broj, razlomljen broj, karakter, tekst, niz, logička promenljiva, niz od 40 brojeva)
    a koji određuje i koliko mesta u memoriji zauzima promenljiva
  - **Vrednost promenljive** (dobra praksa je uvek dodeljivati vrednost promenljive)

    **int a = 5;** **int b;** **kolika je početna vrednost b?**
  - Ograničenje na vrednost promenljive ( 5 < ocena < 10 )

# ... PROMENLJIVE



- Mesto deklarisanja promenljive nije ograničeno (može bilo gde u kodu)
- Preporučuje se da se promenljive deklarišu na jednom mestu (najbolje na početak programskog bloka), jer je lakše za održavanje programskog koda
- Ako nije drugačije označeno, promenljiva važi (vidljiva je) od mesta deklarisanja do kraja programskog bloka
- Programski blok (programska celina) određena **{ }**

# DOSEG PROMENLJIVE



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

int main()
{
   int a = 5;
   printf("Promenljiva a ima vrednost: %d\n", a);
   printf("\nPromenljiva b ima vrednost: %d\n", b);
   a = a + 2;
   printf("\nPromenljiva a=a+2 ima vrednost: %d\n", a);

   int b;
   printf("\nPromenljiva b ima vrednost: %d\n", b);
   b = a + 13;
   printf("\nPromenljiva b=a+13 ima vrednost: %d\n", b);

   return 0;
}
```

Da li ovo može ovde?

Da li ovo može ovde?

a

b

# TIPOVI PODATAKA

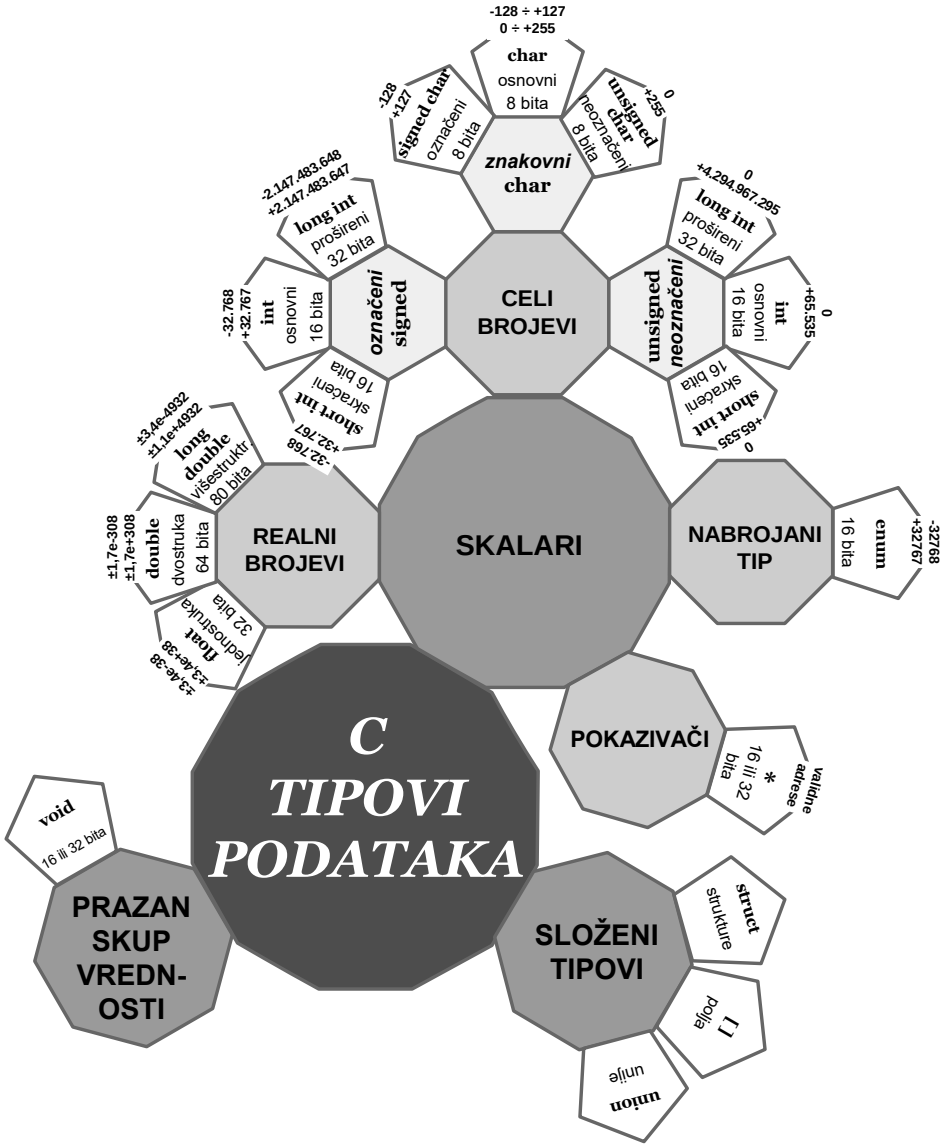
# TIPOVI PODATAKA



- Definišu dostupne tipove podataka u nekom programu
- Određuje:
  - **količinu memorije koju će zauzeti promenljiva**
  - **opseg mogućih vrednosti**
  - **dozvoljene operacije**
- Više tipova podataka koji opisuju brojeve:
  - int, unsigned, short, long (celi brojevi)
  - float, double, long double (realni brojevi)
  - char (karakter / može i kao broj)
  - **boolean (logički tip)?**
- Koliko koji tip zauzima memorije zavisi od konkretne implementacije C kompajlera

# TIPOVI PODATAKA PREMA C99





# CELI BROJEVI



- Celobrojne vrednosti:

  **3**, **7**, **-13**, **12000**

- Najčešće korišćen celobrojni tip je **int**

**unsigned/signed short/long int**

- **signed int** ili samo **int**
- **short int** ili samo **short**
- **long int** ili samo **long**
- **unsigned int** ili samo **unsigned**

- Ukoliko su celobrojne vrednosti dovoljno male, može se koristiti short

# REALNI BROJEVI



- Koriste se za predstavljanje realnih (razlomljenih brojeva):

  **12.22**, **234.0004444**, **-11.4444**

  – **decimalna tačka a ne decimalni zarez !**

- Najčeće korišćen je **double**

**float , double , long double**

- Kada se želi uštedeti na memorijskom prostoru, a raspon upotrebljenih vrednosti to dozvoljava, koristiti se **float**

- Kada se koriste izrazito velike vrednosti, ili je potrebno postoći veliku preciznost, koristi se **long double**

# CHAR TIP ...



- Karakteri:
  **'A', 'B', 'c', 'd', '1', '4', '#', '?', '\n'**

- Promenljive **char** tipa se koriste za čuvanje tekstualnih znakova:
  - slova
  - cifara
  - specijalnih znakova
  - neštampajućih (belih) znakova
  - ali može da se koristi i za čuvanje malih celobrojnih vrednosti (**od 0 do 255 ili od -128 d0 127**)

# ... CHAR TIP



- **CHAR** je samo broj!
- Svakom karakteru se pridružuje numerički kod
- Postoje različiti skupovi kodova:
  - ASCII (American Standard Code for Information Interchange) – najčešći
  - EBCDIC – zastareo, retko se koristi
  - Unicod – noviji, dobija dominantu ulogu
- Vežbaćemo sa ASCII kododovima

# DEKLARACIJA PROMENLJIVE



- Promenljiva se deklariše ali joj se ne dodeljuje inicijalna vrednost

**tip identifikator;**

```
int godina;
float pi;
short sat;
char znak;
```

- Iako vrednost nije esplicitno navedena,
  **C kompajler dodaje neku vrednost**
  **(**ako je sreće, to je **nula**)

# DEFINICIJA PROMENLJIVE



- Promenljiva se deklariše ali joj se dodeljuje i inicijalna vrednost

**tip identifikator = vrednost;**

```
int godina = 2011;
float pi = 3.14;
short sat = 23;
char znak =  'a';
```

- Dobra praksa je uvek inicijalizovati promenljivu na poznatu i željenu vrednost

# INICIJALIZACIJA PROMENLJIVE



- Dodela vrednosti naknadno već deklarisanoj ili inicijalizovanoj promenljivi

**identifikator = vrednost;**

```
godina = 2014;

sat = sat – 1;
```

- **Da bi se promenljiva koristila mora prethodno biti deklarisana !**

- Konkretna vrednost zove se **neposredni operand** jer joj se vrednost neposredno zadaje i ne može menjati

# MEMORIJSKA KLASA PROMENLJIVE



- Definiše u kojem delu memorije će se nalaziti promenljiva
- Opciono se može navesti ispred oznake tipa u deklaraciji promenljive, ako se ne navede podrazumeva se **auto**

  **<mem_klasa> tip identifikator;**

- Memorijske klase u C programskom jeziku su:
  - **auto** – kreira se na početku bloka u kojem je deklarisana (u nekoj funkciji), traje do kraja bloka, utiče da li je lokalna ili globalna promenljiva
  - **register** – sugeriše kompajleru da smesti promenljivu u registar, ali kompajler ne mora to da uradi tako
  - **static** – promenljiva se kreira na početku programa i traje do kraja izvršavanja programa (vrednost promenljive se prenosi iz jednog izvršavanja funkcije u drugo izvršavanje funkcije)
  - **extern** – označava da se deklaracija promenljive nalazi u nekoj eksternoj datoteci

# PRIMERI RADA SA PROMENLJIVAMA



**Koristiti %f kod printf umesto %d.**

1. Program koji obračunava transakciju dinara u evro po kursu 116.5, pretpostaviti da se menja 155000 dinara.
   */* je operator deljenja

2. Program koji računa prosečnu potrošnju goriva na pređenih 100 kilometara ako je pređeno 350 km a potrošeno 17 l goriva.
   ***** je operator množenja

3. Program koji računa površinu i obim kruga poluprečnika

$$P = r^2\pi \qquad O = 2r\pi$$

# KONSTANTE

# KONSTANTE



- Identifikator čija vrednost se ne menja u programu
- **Netipska**
  - vrednost zamenjuje identifikator pre kompajliranje
  - postaje neposredni operand

```
#define Punoletan 18
#define PorukaLozinka “Unesite lozinku”
```

- **Tipska**
  - čuva se u memorije ali joj se vrednost ne može menjati
  - tip se ne mora eksplicitno navoditi (zaključuje se iz vrednosti)
  - mora se dodeliti vrednost

```
const Punoletan = 18;

const short RadniDanSati = 8;

const double pi = 3.14159265358979323846264433;
```

# PRIMERI RADA SA KONSTANTAMA



- Šta su kandidati za konstante u prethodnim primerima?
- **kurs**
- **kmprosek**
- **pi**

# DEKLARACIJE TIPOVA

# DEKLARACIJE TIPOVA



- Korišćenjem ključne reči **typedef** postojećim tipovima podataka daju se nova imena (**ne nastaju novi objekti, niti novi tipovi!**)

**typedef tip novo_ime_tipa;**

- **novo_ime_tipa** postaje sinonim za **tip**
- Kasnije se u kodu koriste ravnopravno sa istim značenjem
- Svrha i korisnost deklaracije tipova se uočava tek kod složenih tipova:
  - omogućuje lakše razumevanje (čitanje) programskog koda
  - omogućuje lakše dokumentovanje programa

# PRIMERI DEKLARACIJE TIPOVA



```
typedef int Brzina;

...

Brzina prva, druga, treca;
```

```
typedef  int TDaniUMesecu[31];

...

TDaniUMesecu Januar, Mart;
```

# ENUMERACIJA

# NABROJANI TIP – ENUMERACIJA …



- Skup mogućih vrednosti je skup svih nabrojanih vrednosti
- U C programskom jeziku je skalar
- Ona služi da se napravi fiksna lista nekih vrednosti
- Svaku nabrojanu vrednost kodira 16bitnim brojem
- Deklariše se na sledeći način:

```
enum naziv_enum_tipa { lista_dozvoljenih_vrednosti }
```

# … NABROJANI TIP – ENUMERACIJA …



- Na primer:

```
enum {Ne, Da};
enum temp {min = -15, radna = 30, maks = 50};
enum VojnickiCin {Razvodnik, Desetar, MladjiVodnik};
typedef enum VojnickiCin TVojnickiCin;
enum meseci {Jan=1, Feb, Mart, Apr, Maj, Jun, Juli, Avg, Sept, Okt,
    Nov, Dec};
enum temp temperatura; //moze -15, 30 ili 50
enum VojnickiCin Vojnici;   // ali sa TVojnickiCin Vojnici;
Vojnici = Desetar;          // ali i sa Vojnici = 1;
Vojnici = Vojnici + 1;      //koju vrednost sada ima Vojnici?
```

- Ove vrednosti mogu na ulaz/izlaz ali samo kao int

# … NABROJANI TIP – ENUMERACIJA



- Svaki element enumeracije zauzima tačno određenu poziciju u enumeraciji
- Vrednosti mogu biti i eksplicitno zadate, pri čemu dalje nabrajanje nastavlja od definisane vrednosti
- Enumeracija omogućava veću čitljivost koda, jer često postoji potreba da se vrednost koju može da uzme neka promenljiva ograniči na jedan skup mogućih vrednosti
- Omogućuje da kompajler uoči ako se ne upotrebi neka od očekivanih vrednosti (jer su samo enum vrednosti legalne) čime se mogu smenjiti semantičke greške
- enum se deklariše van ***main()*** metode

# POKAZIVAČI

Dragan dr Dinu

# POKAZIVAČI ...



- Pokazivač nosi indirektni podatak:
  **adresu memorijske lokacije gde se nalazi stvarni podatak**

- Tretira se kao skalar jer je to zapravo broj (lokacija u memoriji)

- Pokazivačke promenljive pokazuju na druge promenljive ili na početak memorijskog bloka
  (pokazivači na promenljive tipe *int*, *float*, i sl.)

- Deklaracija pokazivača:

  **tip *pok;**

  pokazivač **pok** na promenljivu tipa **tip**

# ... POKAZIVAČI ...



- Definisanje pokazivača:

  **tip *pok = &prom;**

  pokazivač **pok** na promenljivu tipa **tip** koji čuva adresu promenljive **prom**

- Primenom unarnog operatora ***** posredno se pristupa nekom podatku pomoću adrese (**dereferenciranje**)

- Inicijalizacija pokazivača:

  **pok = &prom;**

  pokazivaču **pok** se dodeljuje adresa promenljive **prom**
  (**pok** pokazuje na **prom**)

- Unarni operator **&** daje adresu promenljive (**referenciranje**)

# ... POKAZIVAČI ...



- Pristup vrednosti na koju pokazuje pokazivač:

**prom2 = *pok;**

promenljiva **prom2** dobija vrednost sa lokacije na koju pokazuje pokazivač **pok**

***pok =prom3;**

u lokaciju na koju pokazuje pokazivač **pok** upisuje se vrednost promenljive **prom3**

**prom4 = pok;**

promenljiva **prom4** dobija adresu lokacije na koju pokazuje pokazivač **pok**

**pok =prom5;**

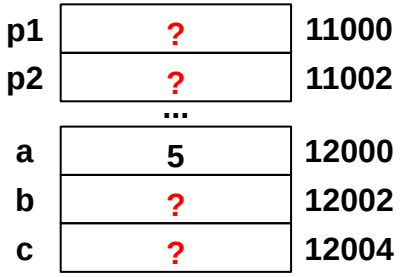
u pokazivač **pok** upisuje se vrednost promenljive **prom5**

# POKAZIVAČI – primeri



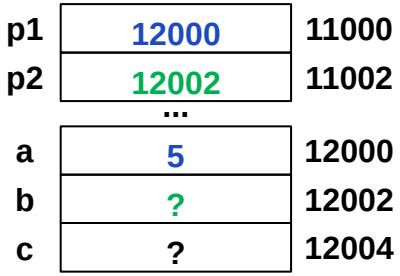
- Deklarisanje:

```
int a = 5;
int b, c;
int *p1, *p2;
```



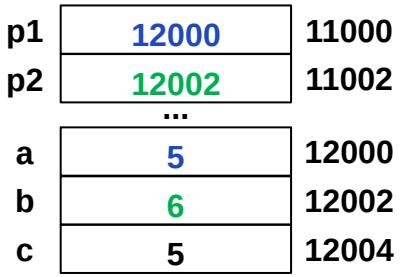

- Dodela vrednosti:

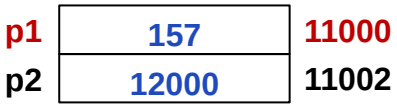
```
p1 = &a;
p2 = &b;
```




- Pristup lokaciji:

```
c = *p1;
*p2 = 6;
p2 = p1
p1 = 157;
```



# OPERACIJE SA POKAZIVAČIMA



- Dodela vrednosti dovodi do toga da obe pokazivača pokazuju na isto

```
int a = 5;
int *p1, *p2;
p1 = &a;
p2 = p1;          // p1 i p2 sada pokazuju na istu lokaciju
```

- Mogu se sabirati/oduzimati sa brojem

```
p1 = p1 + 10;     // ako je p1 bilo 12000, posle sabiranja biće 12020
p2++;             // ako je p2 bilo 12002, posle inkrementiranja biće 12004
                  // zbog int koji zauzima 2 bajta
```

- Mogu se porediti

```
if (p1 == p2)     // proverava se da li su iste adrese
```

# NULL KONSTANTA



- Nalazi se u fajlu zaglavlja **<stdio.h>**
- Ako pozivač ima vrednost **NULL**, onda ne pokazuje ni na šta

- Primer:

```
int *p;
p = NULL;
```

- **Neinicijalizovana promenljiva nema vrednost NULL !**
  (mora se eksplicitno inicijalizovati)

```
int *p;   ⇨   p != NULL;
```

# VOID TIP



- Generički tip pokazivača – **void**

  **void *pokVoid;**

  pokazivač **pokVoid** ne može do podatka jer on ne pokazuje na konkretan tip

- Može primiti vrednost drugih pokazivača:

  **int *pokInt;**

  **pokVoid = pokInt;**

- Sada **pokInt** i **pokVoid** pokazuju na istu memorijsku lokaciju

- Ipak, do podatka se može doći samo eksplicitnom konverzijom tipa:

  **int a = (int *) pokVoid + 12;**

# RAD SA POKAZIVAČIMA...



- Dinamička alokacija memorije:
  - **malloc** funkcija, alocira memoriju tražene veličine i vraća pokazivač na alociranu memoriju (ako je uspelo) ili NULL (ako alociranje nije uspelo)

    **int *pokInt = (int *)malloc(sizeof(int));**

    - **sizeof** je operator za određivanje memorije koju zauzima neka promenljiva ili tip

  - **calloc** funkcija, alocira blok kontinualne memorije za **N** elemenata određene veličine, i za razliku od **malloc** postavlja vrednosti svih lokacija na 0, vraća pokazivač na alociranu memoriju (ako je uspelo) ili NULL (ako alociranje nije uspelo)

    **double *pokNizDoble = (double *)calloc(10, sizeof(double));**

# ... RAD SA POKAZIVAČIMA



- **realloc** funkcija, pokušava da proširi alociranu memoriju tako da odgovara traženoj veličini, vraća pokazivač na alociranu memoriju (ako je uspelo) ili NULL (ako alociranje nije uspelo)

  **double *pokNizDoble = (double *)realloc(pokNizDoble , 15* sizeof(double));**

- Kada se završi sa radom, alociranu memoriju je potrebno osloboditi, C program to ne radi automatski
  - **free** funkcija, koristi se za oslobađanje (dealociranje) dinamički zauzete memorije (**malloc, calloc, realloc**), ne vraća nikakvu vrednost

    **free (pokInt);**

    **free(pokNizDouble);**

# ČESTE GREŠKE U RADU SA POKAZIVAČIMA



- Iako je moguće, da li je potrebno deklarisati pokazivač na konstantu?
  **const int *pok;**
- Isto važi i za konstantan pokazivač na promenljivu
  **int * const pok = &a;**
- Nemoguće je promeniti adresu promenljive (ne zavisi od nas)
- Sledeći izrazi su pogrešni:

```
p1 = &5;
p2 = &(i + 7);
p3 = &(x == y);
&x = &y;
&x = 150;
```

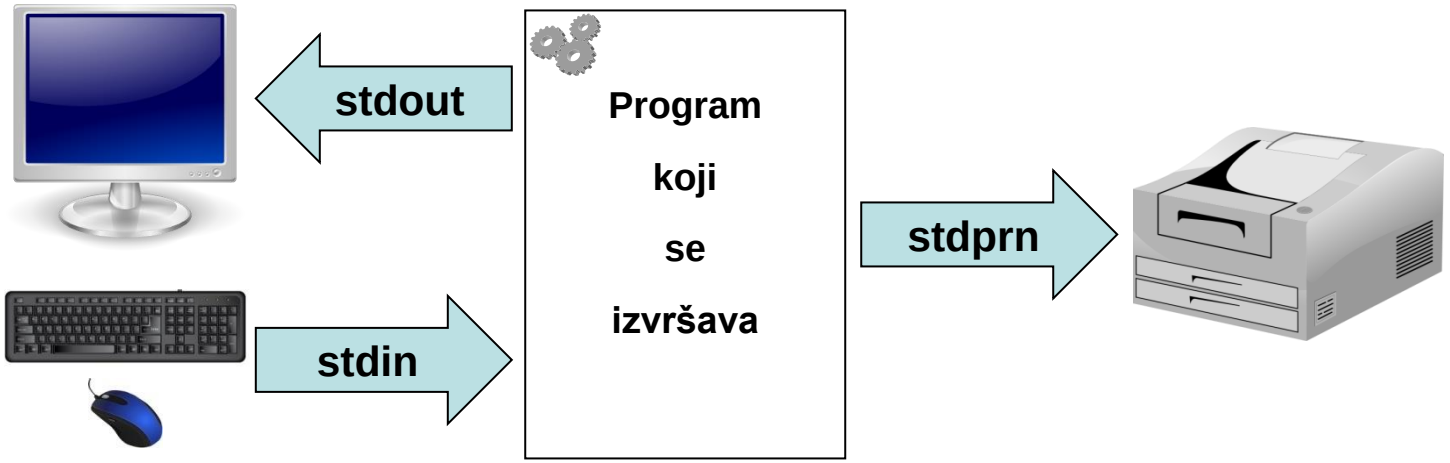
# KOMUNIKACIJA SA OKOLINOM

Dragan dr Dinu

# KOMUNIKACIJA SA OKOLINOM



- C sve uređaje tretira kao fajl
- Upisuje se i čita iz fajla



# OSNOVNO O FUNKCIJAMA



- Omogućuju **modularnost**, ponovno **korišćenje**
- Implementiraju **semantički zatvoren posao** tako da se mogu koristiti u drugom rešenju
- Identifikuju se **nazivom**
- Potrebne podatke za rad dobijaju putem **ulaznih** i/ili **ulazno/izlaznih** parametara
- Rezultate rada prosleđuju **nazivom** i/ili **izlaznim** parametrima
- Mogu biti ugrađene (deo pratećih C zaglavlja) ili ih program implementira i koristi

# PRINTF FUNKCIJA ...



- Koristi se za prikaz podataka korisniku
- Automatski konvertuje podatke iz promenljivih u izlazni tekst
- Može imati jedan (najmanje) ili više parametara
- Deo je standardnog fajla izlaza/ulaza, **<stdio.h>**

**printf("*Tekst*", promenljive...)**

- **Tekst** može sadržati:
  - običan string, npr. **FTN**
  - iskejp sekvence, npr. **\n**
  - format polja – definiše kako treba interpretirati promenljive, npr. **%d**

# ... PRINTF FUNKCIJA



- Izlaz se dodatno može formatirati pomoću iskejp sekvenci
- Broj promenljivih bi trebalo da odgovara broju format polja
  - ako ima više promenljivih, ništa se neće desiti
  - **ako ima više format polja, nešto će se ispisati**
- Funkcija printf svojim identifikatorom vraća broj konvertovanih (ispisanih) karaktera na monitor

```
char adresa[] = “Fruskogorska”;
unsigned broj = 11;
printf("Adresa: \" %s\n%u \"", adresa, broj);
```

**Na ekranu je prikazano:**

```
Adresa: "Fruskogorska
11"
```

  - **printf vraća vrednost 26, a ne 25, zašto?**
- **fflush(stdout)** – čisti fajl u koji ispisuje printf

# FORMAT POLJA PRINTF FUNKCIJE



Formira se na sledeći način:

**% [flags] [width] [.prec] form.ind.**

1. %
2. **flags: -** ili **+** ili **blank** ili **0**
3. **width**
4. **.prec**
5. **format identifikatori:**
   **d** ili **i** ili **o** ili **u** ili **x** ili **X** ili **f** ili **e** ili **g** ili **E** ili **G** ili **c** ili **s** ili **%**

2, 3, 4 moguće je preskočiti (zato **[ ]**)

**1 i 5 su obavezni**

**-**, poravnaj ulevo, inače udesno
**+**, forsiraj predznak
**blank**, prefiksovan praznim mestima,
**0**, prefiksovan nulama
**[width]** – min. mesta ekrana, moguća dopuna **blank** ili **0**,
**[.prec]** – specifikator preciznosti,
**d,i** – označeni dekadni ceo broj,
**hd** – short, **ld** – long
**o** – neoznačeni oktalni ceo broj,
**u** – neoznačeni dekadni broj,
**x** – neoznačeni heks. ceo broj,
**X** – neoznačeni heks. ceo broj,
**f** – vrednost u obliku [-]dddd.dddd,
**lf** – double, **Lf** – long double
**e** – vred. [-]d.dddd ili e[+/-]ddd,
**g** – e ili f oblik,
**E,G** – kao e,g, E umesto e,

# ISKEJP SEKVENCE (ESCAPE SEQUENCES)



| ZAPIS | ZNAČENJE SEKVENCE |
|---|---|
| **\n** | prelazak u novi red (NewLine) |
| **\t** | horizontalna tabulacija, horizontalni pomeraj (Horizontal Tabulation) |
| **\v** | vertikalna tabulacija, vertikalni pomeraj (Vertical Tabulation) |
| **\b** | povratak za jedno mesto ulevo (Backspace) |
| **\r** | povratak na početak tekućeg reda (Carriage return) |
| **\f** | prelazak na novu stranicu (Form feed) |
| **\a** | oglašavanje sistemskog zvona, sistemski bip (Alert/Bell) |
| **\'** | vrednost simbola jednostrukog navodnika |
| **\"** | vrednost simbola dvostrukog navodnika |
| **\?** | vrednost simbola znaka pitanja |
| **** | vrednost simbola obrnute kose crte (Backslash) |
| **\o□□□** | vrednost simbola čiji je ASCII kôd zapisan oktalno, za 'A' je '\o101' |
| **\x□□** | vrednost simbola čiji je ASCII kôd zapisan heksadecimalno, za 'A' je '\x41' |

# PRINTF FUNKCIJA PRIMERI



- U prethodnim primerima identifikovati format polja u printf funkcijama.
- Izmeniti gde je moguće %f u %d
- Formatirati prikaz tako da lepo izgleda
  (koristiti ----------, poravnavanje,
  širinu i preciznost prikazivanja rezultata)
- Kod prvog primera (kurs) probati %9.2, pa %09.2
- Probati i **fflush(stdout)**

# SCANF FUNKCIJA ...



- Koristi se za preuzimanje vrednosti od korisnika
- Automatski konvertuje ulazni tekst u odgovarajuće tipove podataka i upisuje podatke u promenljive
- Trebalo bi da ima minimum dva ili više parametara
- Deo je standardnog fajla izlaza/ulaza, **<stdio.h>**

**scanf("*Tekst*", adrese_promenljivih ...)**

- **Tekst** sadržati format polja (slično printf funkciji)

# ... SCANF FUNKCIJA



- **Mora biti adresa promenljive** a ne samo identifikator (osim u slučajevima kada je identifikator zapravo adresa ...)

  **scanf("%d", a);** – će izazvati grešku, možda baciti poruku o grešci, ali će program vrlo verovatno nastaviti sa radom

- Do adrese promenljive dolazi se operatorom **&**

  **scanf("%d", &a);**

- Funkcija scanf svojim identifikatorom **vraća broj uspešno obavljenih konverzija**

- Broj promenljivih bi trebalo da odgovara broju format polja
  - ako ima više promenljivih, ništa se neće desiti
  - **ako ima više format polja, preuzeće vrednosti za sva format polja ali ih nigde neće upisati**

# FORMAT POLJA SCANF FUNKCIJE



Formira se na sledeći način:

**% [flags] [width] [.prec] form.ind.**

1. %
2. *****
3. **width**
4. **format identifikatori:**
   **d** ili **i** ili **hd** ili **ld** ili **o** ili **u** ili **x** ili **X** ili **f** ili **lf** ili **Lf** ili **e** ili **g** ili **E** ili **G** ili **c** ili **s** ili **%**

Korake 2, 3 moguće je preskočiti

**Koraci 1 i 4 su obavezni**

**%** - mark konverzacionog polja,

**[*]** – preskoči 1 ul. polje datog tipa,

**[width]** – min. broj karaktera koji će se čitati, ali i manje ako se dostigne blank ili nekonvertabilni karakter,

**form.ind.** – obavezan, oznaka kraja konverzacionog polja, specifikuje način interpretacije učitanog,

# SCANF FUNKCIJA PRIMERI



- U prethodnim primerima identifikovati kojim promenljivama vrednost treba korisnik da definiše
- Odabrati odgovarajuća format identifikatore
- Uočiti da bez prethodne upotrebe printf funkcije koja objašnjava scanf akciju, korisnik neće znati šta treba da unese, npr.

```
printf("Unesite vrednost promenljive a:\n");
scanf("%d", &a);
```

- Šta se desi ako se izuzme \n iz printf funkcije?