Komunikacija C programa sa okruženjem (ulaz i izlaz)

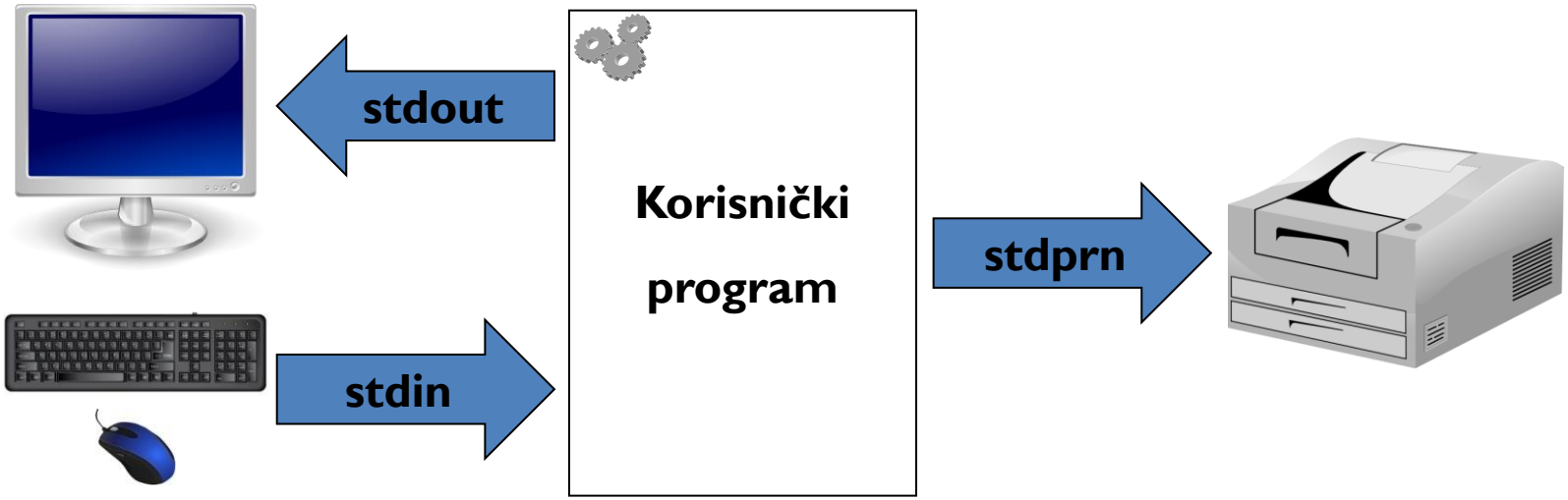
Komunikacija C programa sa okruženjem

U jeziku C, svi uređaji posmatraju kroz tokove podataka (engl. *streams*)

Tokovi podataka su logički interfejsi ka fajlovima

Upisuje se u i čita se iz toka podataka

Standardni tokovi podataka: *stdin*, *stdout*, *stderr*



Komunikacija putem funkcija

Omogućuju **modularnost**, ponovno **korišćenje koda**

Implementiraju **semantički zatvoren zadatak** tako da se mogu koristiti u različitim programima

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 implementirane u korisničkom programu

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 standardne biblioteke izlaza/ulaza **<stdio.h>**

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

Tekst može sadržati:

- običan string, npr. **FTN**
- specijalni karakteri - iskejp sekvence, npr. **\n**
- format specifikatori (polja) – definišu 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? ASCII vrednost 0 se dodaje automatski kao znak za kraj stringa

Format polja *printf* funkcije

Formira se na sledeći način:

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

1. %
2. **flags: -** ili **+** ili **blank** ili **0**
3. **širina**
4. **preciznost**
5. **modifikator dužine**
6. **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, 5 moguće je preskočiti (zato **[ ]**)

Delovi 1 i 6 su obavezni

- – poravnanje ulevo, inače udesno
+ – forsiranje prikaza predznaka
blank – blanko ispred pozitivnih brojeva
0 – dodavanje prefiksa u vidu nula
[width] – min. mesta ekrana, moguća dopuna **blank** ili **0**
[.prec] – specifikator preciznosti
d,i – označeni dekadni ceo broj
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
e – vred. [-]d.dddd ili e[+/-]ddd
g – e ili f oblik
E,G – kao e,g, E umesto e

Specijalni karakteri - *esc*ape sekvence

zapis	značenje specijalnog karaktera
\n	prelazak u novi red (engl. *new line*)
\t	horizontalna tabulacija, horizontalni pomeraj (engl. *horizontal tabulation*)
\v	vertikalna tabulacija, vertikalni pomeraj (engl. *vertical tabulation*)
\b	povratak za jedno mesto ulevo (engl. *backspace*)
\r	povratak na početak tekućeg reda (engl. *carriage return*)
\f	prelazak na novu stranicu (engl. *form feed*)
\a	oglašavanje sistemskog zvona, sistemski bip (engl. *alert/bell*)
\'	vrednost simbola jednostrukog navodnika
\"	vrednost simbola dvostrukog navodnika
\?	vrednost simbola znaka pitanja
****	vrednost simbola obrnute kose crte (engl. *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 standardne biblioteke izlaza/ulaza **<stdio.h>**

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

Tekst sadrži format polja (slično *printf* funkciji)

scanf funkcija

Mora se navesti 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:

% [*] [width] [lengthmod] form.ind.

1. %
2. *
3. **širina**
4. **modifikator dužine**
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 %

Delove 2, 3 i 4 moguće je preskočiti

Delovi 1 i 5 su obavezni

% - oznaka konverzionog polja

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

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

form.ind. – obavezan, oznaka kraja konverzionog polja, specificira način interpretacije učitanog

scanf funkcija – primeri

U prethodnim primerima identifikovati kojim promenljivama korisnik treba da definiše vrednost

Odabrati odgovarajuće 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 će se desiti ako se izuzme \n iz *printf* funkcije?

Komentari

Komentari su namenjeni čoveku, a ne računaru ili kompajleru!

Eliminiše ih pretprocesor

Dva osnovna tipa:

- /* */ – umeće se bilo gde u kodu, može da se prenese u više redova
- // – važi od mesta definisanja do kraja programske linije

```
/* Program koji
   radi…          */
int main ()
{
```

```
int main()
{
   int i = 1; //primer komentara
```

Komentari

Komentarisanje je važan deo pisanja kvalitetnih programa

Komentari olakšavaju razumevanje, održavanje i dalji razvoj programskog koda

Loš stil komentarisanja – komentari samo rečima opisuju sadržaj naredbe:

```
o = 2*r*pi;  // o se dobija kao proizvod 2 i r i pi
```

Dobar stil komentarisanja - komentari sadrže informaciju koja nije zapisana u samom kodu:

```
o = 2*r*pi;  // racunamo obim kruga
```

Operatori

C operatori

Određuju aktivnost nad operandom/operandima

Izvršavanje operatora rezultuje nekom vrednošću

Vrednost izvršavanja operatora može poslužiti kao operand za drugi operand

U programskom jeziku C **sve ima vrednost** sa kojom se može manipulisati

Operatori programskog jezika C:

- aritmetički operatori
- relaciono-logički operatori
- operatori nad bitovima (engl. *bitwise*)
- operatori dodele vrednosti
- ternarni operator
- ostali operatori (adresni, *sizeof*...)

Aritmetički operatori

Unarni operatori		
+	operator zadržavanja predznaka operanda	**+ operand**
-	operator promene predznaka operanda	**- operand**
++	prefiksni operator inkrementiranja	**++ operand**
++	sufiksni operator inkrementiranja	**operand ++**
--	prefiksni operator dekrementiranja	**-- operand**
--	sufiksni operator dekrementiranja	**operand --**

Aritmetički operatori

Primeri unarnih aritmetičkih operatora

x++;	x= x + 1
++x;	x = x + 1
y= x--;	y = x, x = x – 1
y = --x;	x = x – 1, y = x
(y + x)--;	nije dozvoljeno
++(y + x);	nije dozvoljeno
+ -3.14	-3.14

Aritmetički operatori

Prioritet je isti kao u matematici!

Binarni operatori		
+	operator sabiranja	**operand + operand**
–	operator oduzimanja	**operand - operand**
*	operator množenja	**operand * operand**
/	operator deljenja, tip operanada određuje vrstu deljenja, ako su oba celobrajna, rezultat je celobrajan bez ostatka ako je barem jedan realan, rezultat je realan broj	**operand / operand**
%	operator ostatka celobrojnog deljenja, oba operanda moraju biti celobrojni, predznak ostatka je isti kao predznak prvog operanda	**operand % operand**

Aritmetički operatori

Primeri binarnih aritmetičkih operatora

x + y – z	**(x + y) – z**
x - y * z	**x – (y * z)**
9 / 2 * 2	**8**
9. / 2 * 2	**9**
-8 % -3	**-2**

Vežba:
Implementirati program koji prihvata od korisnika dve celobrojne vrednosti i realizuje aritmetičke binarne operacije (+, -, *, /, %) nad njima. Rezultat prikazati korisniku.

Relacioni operatori

Ispituju relacije (odnose) između dva operanda

Rezultat je:

1 - ako relacija važi

0 - ako relacija ne važi

Binarni operatori		
<	operator manje	**operand < operand**
<=	operator manje ili jednako	**operand <= operand**
>	operator veće	**operand > operand**
>=	operator veće ili jednako	**operand => operand**
==	operator jednako	**operand == operand**
!=	operator nije jednako	**operand != operand**

Relacioni operatori

Grade relacione izraze

x < y < z, isto što i **(x < y) < z**

Ako se želi promeniti redosled operacija, koji zavisi od njihovog prioriteta, koristi se zagrada **( )**

x < (y < z)

Tako će:

x == y >= z, zbog prioriteta operacija biti **x == (y >= z)**

Operatori **<, <=, >, >=** su većeg prioriteta od **==, !=**

Logički operatori

Formiraju logičke izraze

Rezultat je:

- **1 za istinu (true)**
- **0 za neistinu (false)**

Rade na nivou čitavog operanda

Logički operatori		
!	prefiksni unarni operator logičke negacije	**! operand**
&&	binarni operator logičke I operacije	**operand && operand**
||	binarni operator logičke ILI operacije	**operand || operand**

Logički operatori

Primeri logičkih operatora:

!0 == 1	!1 == 0	!99 == 0	
0 && 0 == 0	0 && 1 == 0	17 && 0 == 0	(-14) && 5 == 1
0 || 0 == 0	0 || 1 == 1	-12 || 0 == 1	33 || 39 == 1

Koriste se najčešće kao veznici u relacionim izrazima

C kompajler skraćuje izvršavanje ako je rezultat jasan već u nekom koraku

Logički operatori

Primeri upotrebe logičkih operatora:

Izraz	Ponašanje kompajlera
x && y || z && k;	(x && y) || (z && k)
a == !b && c < d ;	(a == (!b)) && (c < d)
7 + 2 || ++x;	skraćeno do istine
10 % 5 && x != y;	skraćeno do neistine

Operatori nad bitovima

Ne mešati ih sa logičkim operatorima!

Rade na nivou pojedinačnih bitova (engl. *bitwise*)

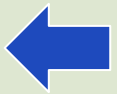
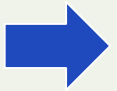
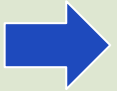
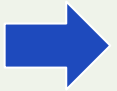
Operatori nad bitovima		
~	prefiksni unarni operator logičke negacije	**! operand**
&	*bitwise* operator logičke I operacije	**operand & operand**
|	*bitwise* operator logičke ILI operacije	**operand | operand**
^	*bitwise* operator logičke ekskluzivno ILI (EILI) operacije	**operand ^ operand**
<<	*bitwise* operator pomeranja ulevo	**operand << br_pomeranja**
>>	*bitwise* operator pomeranja udesno	**operand >> br_pomeranja**

Operatori nad bitovima

Primeri upotrebe *bitwise* operatora

Primer:	unsigned char uc1 = 110, uc2 = 3; signed char sc1 = 16, sc2 = -32;	
NE	~uc1 = ~(110) = ~01101110 = 10010001 = 145	
I	uc1 & uc2 = 110 & 3 = 2	01101110 & 00000011 00000010
ILI	uc1 | uc2 = 110 | 3 = 111	01101110 | 00000011 01101111
EILI	uc1 ^ uc2 = 110 ^ 3 = 109	01101110 ^ 00000011 01101101

Operatori nad bitovima

uc1 << uc2 = 110 << 3 = 112	01101110 11011100 10111000 01110000 
uc1 >> uc2 = 110 >> 3 = 13	01101110 00110111 00011011 00001101 
sc1 >> uc2 = 16 >> 3 = 2	00010000 00001000 00000100 00000010 
sc2 >> uc2 = -32 >> 3 = -4	11100000 11110000 11111000 11111100 

Operatori dodele vrednosti

Dodela vrednosti je jedna od osnovnih aktivnosti računara, sama po sebi vraća vrednost

x = 12;	promenljiva x dobija vrednost 12
y = x = 3;	promenljiva x i promenljiva y dobijaju vrednost 3
k = (z = x * 10) * y;	promenljiva z dobija vrednost 30 promenljiva k dobija vrednost 90

Obratiti pažnju na razliku između == i =, iako jedan proverava jednakost, a drugi dodeljuje vrednost, oba vraćaju vrednosti, pa se mogu koristiti i u logičkim izrazima

Operatori dodele vrednosti

Binarni operatori		
=	"standardno" dodeljivanje vrednosti	x = y
+=	sabiranje i dodeljivanje vrednosti	x += y , x = x + y
-=	oduzimanje i dodeljivanje vrednosti	x -= y , x = x - y
*=	množenje i dodeljivanje vrednosti	x *= y , x = x * y
/=	deljenje i dodeljivanje vrednosti	x /= y , x = x / y
%=	ostatak celobrojnog deljenja i dodeljivanje vrednosti	x %= y , x = x % y

Ternarni (uslovni) operator

Ternarni operator:

? :

Deo je uslovnog izraza:

```
izraz1 ? izraz2 : izraz3;
```

Skoro pa ekvivalentno:

```
if (izraz1)
    izraz2;
else
    izraz3;
```

Ternarni operator – primeri

```
int a = 13, b = 5;
int veci, manji;
...
veci = ((a > b) ? a : b);
// veci = ((13 > 5) ? 13 : 5) = ((1) ? 13 : 5) = 13;
...
manji = ((a < b) ? a : b);
// manji = ((13 < 5) ? 13 : 5) = ((0) ? 13 : 5) = 5;
...
printf( "Manja od dve vrednosti (%d, %d) je %d\n", a, b, manji);
```

Operator konverzije tipova

Konverzija omogućava prevođenje vrednosti promenljive iz jednog u drugi tip, može biti implicitna i eksplicitna (kastovanje – engl. *cast*)

Primena **operatora konverzije**: (ime tipa) izraz

Na primer:

```
int a;
long b;
float c = 2;       //implicitna konverzija
c = (float) a;     //eksplicitna konverzija - cast
a = (int) b;       //eksplicitna konverzija - cast
```

Primer za kraću notaciju kod neposrednih operanada:

5/2 = 2 (celobrojno deljenje)

5/2.0f = 2.5 ili kraće: 5/2.=2.5 (deljenje)

Prioritet izvršavanja operatora

Čest izvor grešaka – treba voditi računa!

Može se regulisati pomoću zagrada ( )

- zagrade se mogu ugnježdavati do proizvoljne dubine
- unutar zagrada, operatori se izvršavaju prema redosledu prioriteta
- ako više operatora u izrazu ima isti prioritet, izvršiće se sleva na desno ili sdesna na levo, u zavisnosti od smera grupisanja operatora na tom nivou prioriteta

C ima čak **15 nivoa operatora**

Prioritet izvršavanja C operatora

Prioritet	Broj operanada	Operatori	Smer grupisanja
15	2	`[] () . ->`	➡
14	1	`! ~ ++ -- + - * & (tip) sizeof`	⬅
13	2	`* / %`	➡
12	2	`+ -`	➡
11	2	`<< >>`	➡
10	2	`< <= > >=`	➡
9	2	`== !=`	➡
8	2	`&`	➡
7	2	`^`	➡
6	2	`|`	➡
5	2	`&&`	➡
4	2	`||`	➡
3	3	`?:`	➡
2	2	`= += -= *= /= %= &= ^= |= <<= >>=`	⬅
1	2	`,`	➡

Kontrola toka

Kontrola toka

Programske upravljačke strukture su **mehanizmi za kontrolu toka - definisanje redosleda izvođenja naredbi u programu**

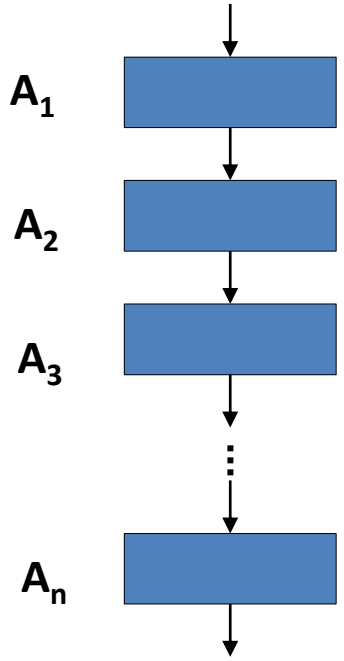
Rešenje bilo kog problema, koje se može realizovati putem algoritma, a time i pomoću računara, može se izraziti kao superpozicija osnovnih algoritamskih struktura:

- **sekvence**
- **selekcije (grananja)** i
- **iteracije (ciklusa, petlje)**

Sekvenca

Omogućava da se naredbe (tj. koraci u algoritmu) izvršavaju jedan za drugim

Linijska struktura koja se dobija kaskadnim povezivanjem blokova obrade



- Algoritamski koraci se izvršavaju sekvencijalno, jedan za drugim
- Algoritamski korak A_i, i=2,...,n ne može da otpočne sa izvršenjem dok se korak A_{i-1} ne završi

Selekcija (grananje)

Omogućava da se određene naredbe (tj. koraci u algoritmu) izvrše uslovno u zavisnosti od vrednosti određenog izraza

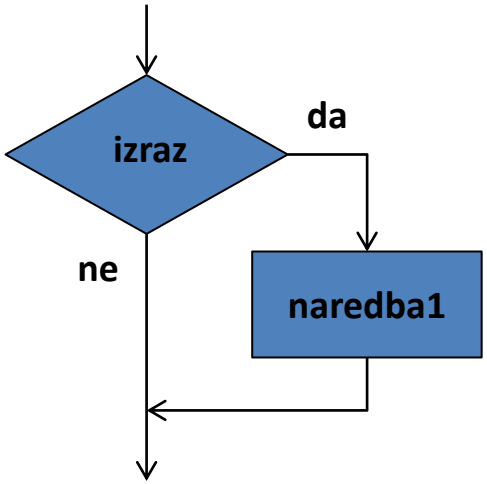
Rezervisane reči u jeziku C za realizaciju selekcije:

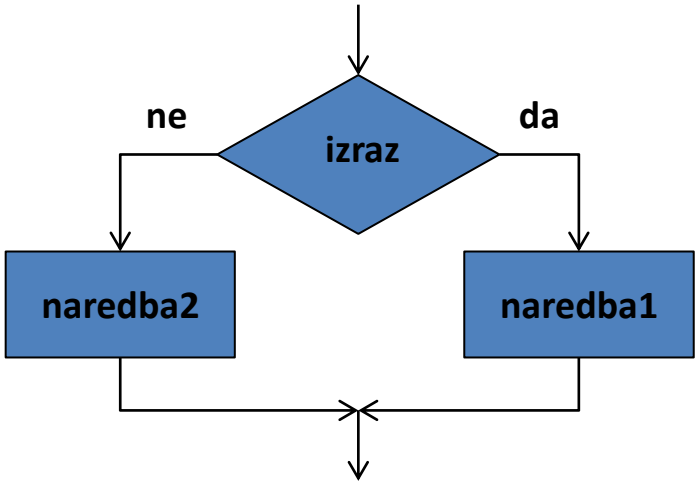
- **if** opciono praćen sa **else**
- **switch** praćen **case** (višestruko grananje)

Selekcija (grananje)

Koju granu (od moguće dve) izvršiti?

Nepotpuna if selekcija	**Potpuna if selekcija**
Ako je **izraz** tačan **naredba1** se izvršava	Ako je **izraz** tačan **naredba1** se izvršava u suprotnom, ako je **izraz** netačan **naredba2** se izvršava





Selekcija (grananje)

Implementacija u jeziku C – blok od jedne naredbe:

Nepotpuna if selekcija	Potpuna if selekcija

`if (izraz)`
`  naredba1;`
	`if (izraz)`
	`  naredba1;`
	`else`
	`  naredba2;`

Selekcija (grananje)

Nepotpuna if selekcija

```
if (izraz)
{
    naredba1 1;
    ...
    naredba1n;
}
```

Potpuna if selekcija

```
if (izraz)
{
    naredba1 1;
    ...
    naredba1n;
}
else
{
    naredba2 1;
    ...
    naredba2m;
}
```

Ugnježđavanje selekcije

Primer:
Ako je **x > 3** prikaži "gore", a ako je **x < 0** prikaži "dole"

```
int x = 0;
scanf("%d", &x);
if (x >= 0)
    if (x > 3)
        printf("gore");
    else
        printf("dole");
```

```
int x = 0;
scanf("%d", &x);
if (x >= 0)
{
    if (x > 3)
        printf("gore");
}
else
    printf("dole");
```

```
int x = 0;
scanf("%d", &x);
if (x > 3)
    printf("gore");
else
    if (x < 0)
        printf("dole");
```

else se uvek vezuje za poslednji **if**

Selekcija – primeri

U ranijim primerima (kurs, potrošnja, krug) dodati logičku proveru unetih vrednosti (da li su možda manje od nule i šta raditi u tom slučaju?)

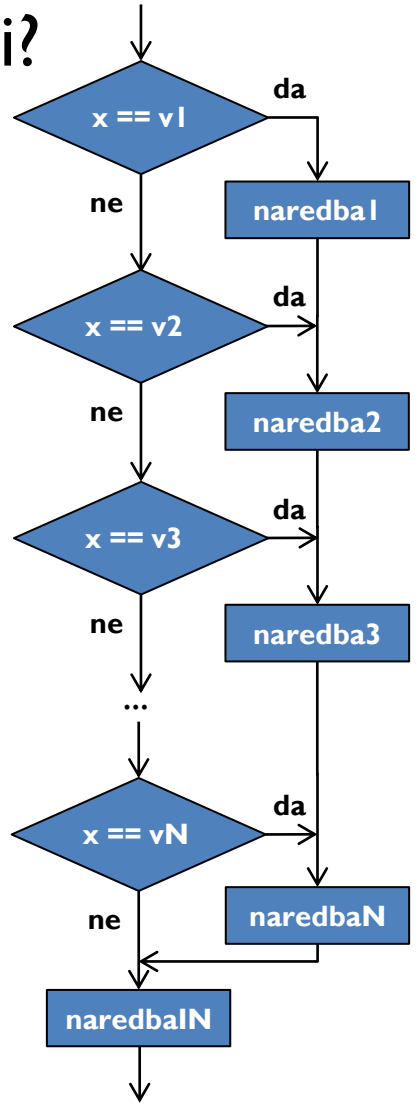
Zadatak za vežbu: Implementirati C program koji ispisuje koji je od dva broja uneta od strane korisnika veći/manji (odrediti tipove podataka, potrebnu proveru, ...)

Višestruko grananje

Koju granu tj. blok naredbi od više mogućih izvršiti?

Izbor se vrši na osnovu celobrojne vrednosti

```
switch (x)
{
    case v1: naredba1;
    case v2: naredba2;
    case v3: naredba3;
    ...
    case vn: naredban;
    default: naredban1;
};
```



Višestruko grananje

Program za ispis uspeha na osnovu unete ocene

```
int ocena;

printf("\n Program za ispis uspeha na osnovu unete ocene!\n\n");
printf("Unesite ocenu: ");
scanf("%d", &ocena);

switch (ocena)
{
    case 5: printf("\nOdlican!\n");

    case 4: printf("\nVrlo dobar!\n");

    case 3: printf("\nDobar!\n");

    case 2: printf("\nDovoljan!\n");

    case 1: printf("\nNedovoljan!\n");
}
```

Višestruko grananje – primer

Kako da se izvrši samo jedan slučaj?

```
int ocena;

printf("\n Program za ispis uspeha na osnovu unete ocene!\n\n");
printf("Unesite ocenu: ");
scanf("%d", &ocena);

switch (ocena)
{
    case 5: printf("\nOdlican!\n");
            break;
    case 4: printf("\nVrlo dobar!\n");
            break;
    case 3: printf("\nDobar!\n");
            break;
    case 2: printf("\nDovoljan!\n");
            break;
    case 1: printf("\nNedovoljan!\n");
}
```

Višestruko grananje – primer

Šta je **break**?

- Jedna od tri naredbe bezuslovnog skoka u jeziku C (pored **continue** i **goto**)
- Kada se naiđe na **break**, blok naredbi se napušta bez obzira na trenutnu vrednost
- Program nastavlja sa izvršavanjem prve naredbe iza bloka
- Ukoliko se pozove u ugnježdenim blokovima, **break** izlazi samo iz unutrašnjeg bloka
- Koristiti vrlo obazrivo i samo u krajnjoj nuždi!

Višestruko grananje – primer

Šta ako uneti broj nije iz opsega 1 - 5?

```
int ocena;

printf("\n Program za ispis uspeha na osnovu unete ocene!\n\n");
printf("Unesite ocenu: ");
scanf("%d", &ocena);

switch (ocena)
{
   case 5:  printf("\nOdlican!\n");
            break;
   case 4:  printf("\nVrlo dobar!\n");
            break;
   case 3:  printf("\nDobar!\n");
            break;
   case 2:  printf("\nDovoljan!\n");
            break;
   case 1:  printf("\nNedovoljan!\n");
            break;
   default: printf("\nOcena mora biti između 1 i 5.\n");
}
```

Zadatak – selekcija

Implementirati C program za računanje vrednosti korena kvadratne jednačine:

$$a \cdot x^2 + b \cdot x + c = 0$$

http://sh.wikipedia.org/wiki/Kvadratna_jednačina

Promenljive:

a, b, c, d, x1, x2, y1, y2

Šta ako je **a == 0** ili **b == 0** ?

Šta ako je **d == 0** ili **d < 0** ili **d > 0** ?

U zavisnosti od vrste korena (proveriti diskriminantu) prikazati rezultat i odgovarajući tekst (**switch?**)

Iteracija (petlja, ciklus)

Iskazi iteracije nazivaju se još i **programskim petljama** jer "vrte" naredbe u krug (petlju)

Omogućavaju ponavljanje određenih naredbi ili blokova više puta (telo iteracije)

C programski jezik omogućava **fleksibilne načine** za određivanje broja ponavljanja petlji ili donošenje odluke o izlasku iz petlje

Iteracija (petlja, ciklus)

Koliko puta će se iteracija izvršavati određuje se na osnovu toga da li je određeni izraz (**uslov izvršavanja iteracije**) tačan ili netačan

U zavisnosti od toga da li se uslov izvršavanja iteracije proverava na početku ili na kraju, razlikuju se **iteracije sa izlaskom na vrhu i na dnu**

Iskazi iteracije u C programskom jeziku:

- **do-while** iskaz (izlazak na dnu)
- **while** iskaz (izlazak na vrhu)
- **for** iskaz (izlazak na vrhu)

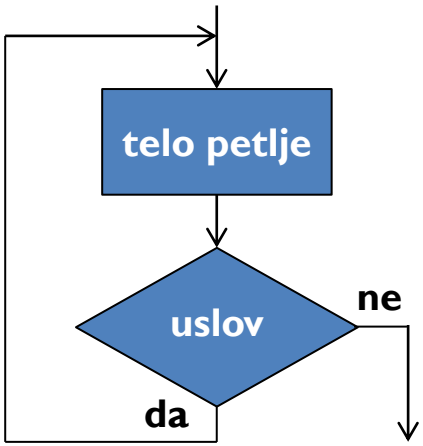
DO WHILE iteracija

Iteracija sa izlaskom na dnu, uslov iteracije se izračunava nakon tela petlje

Telo iteracije se izvršava **najmanje** jednom, čak i kada je uslov izvršavanja iteracije odmah netačan (jednak nuli)

```
do
  naredba;
while (uslov)
```

```
do
{
  naredba1;
  ...
  naredbaN;
}
while (uslov);
```



DO WHILE iteracija – primer

Proširiti primer za ispis uspeha na osnovu unete ocene tako da se korisniku dozvoli samo unos brojeva iz opsega 1 - 5

```
int ocena;

printf("\n Program za ispis uspeha na osnovu unete ocene (1 -5)!\n\n");
printf("Unesite ocenu: ");

do
 {
    scanf("%d", &ocena);
    if ((ocena < 1) || (ocena > 5))
        printf("\n\nOcena mora biti iz intervala 1-5. Unesite ponovo ocenu: ");
} while ((ocena < 1) || (ocena > 5));

switch (ocena)
{
    case 5: printf("\nOdlican!\n"); break;
    case 4: printf("\nVrlo dobar!\n"); break;
    case 3: printf("\nDobar!\n"); break;
    case 2: printf("\nDovoljan!\n"); break;
    case 1: printf("\nNedovoljan!\n"); break;
}
```

WHILE iteracija

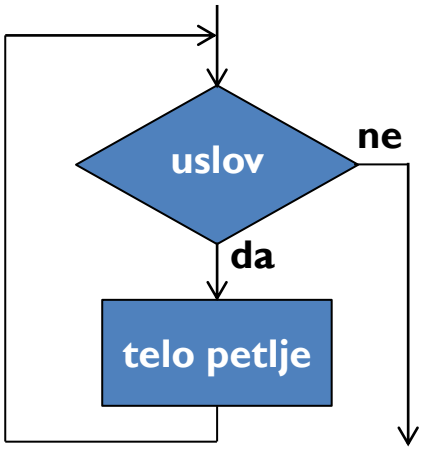
Iteracija sa izlaskom na vrhu, uslov se izvršava pre tela petlje

Telo petlje se izvršava sve dok je uslov izvršavanja iteracije ispunjen (jednak jedinici), kada to nije više slučaj izlazi se iz iteracije

Telo petlje se izvršava nula, jednom ili više puta

```
while (uslov)
    naredba;
```

```
while (uslov)
{
    naredba1;
    ...
    naredbaN;
}
```



WHILE iteracija – primer

Napisati program za računanje faktorijela celog broja koga definiše korisnik

```
int broj = 1, i = 1, fakt = 1;
printf("\nProgram za racunanje faktorijela unetog broja.\n");
printf("\nUnesite broj: ");
do
 {
    scanf("%d", &broj);
    if (broj < 0)
        printf("\n\nBroj mora biti veći od ili jednak nuli. Unesite ponovo broj: ");
} while (broj < 0);
if (broj != 0)
{
   while (i <= broj)
   {
        fakt *= i;
        i++;
   }
}
printf("\n\nFaktorijel broja %d je: %d. \n", broj, fakt);
return 0;
```

FOR iteracija

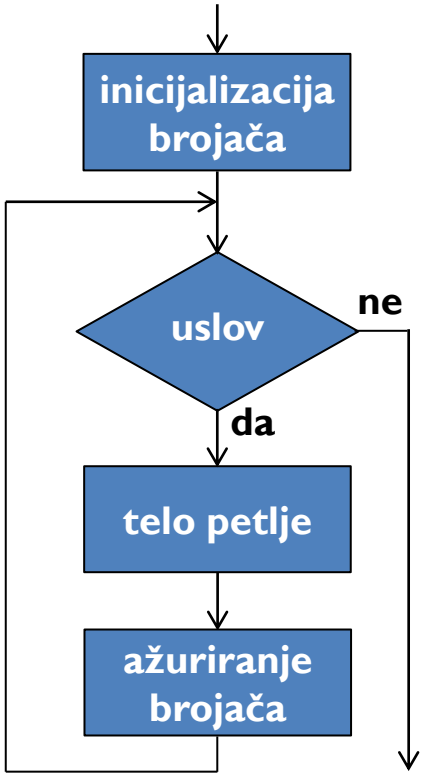
Iteracija sa izlaskom na vrhu, uslov se izvršava pre tela petlje

Telo petlje se izvršava nijednom, jednom ili više puta

Slično WHILE iteraciji, ali se izvršavanje kontroliše brojačkom promenljivom

```
for (inic_br; uslov; ažur_br)
    naredba;
```

```
for (inic.br; uslov; ažur_br)
{
    naredba1;
    ...
    naredbaN;
}
```



FOR iteracija – primer

Kod za prikazivanje prvih 100 prirodnih brojeva:

```
int i;
for (i = 1; i <= 100; i++)
{
    printf("\n%d\n", i);
}
```

Brojač je promenljiva **i**

Brojač se inkrementira (ili dekrementira) nakon svake iteracije

Može i ovako po C99 standardu:

```
for (int i = 1; i <= 100; i++)
{
    printf("\n%d\n", i);
}
```

Šta su prednosti i mane ovog pristupa?

FOR iteracija – primer

Napisati kod za računanje sume prvih *N* prirodnih brojeva, pri čemu se *N* zadaje na početku programa od strane korisnika (problem rešiti pomoću **for** iteracije)

```
int n = 1;
int i, suma = 0;

printf("\n Program za racunanje sume prvih N prirodnih brojeva.\n\n");
printf("Unesite N: ");

do
{
    scanf("%d", &n);
    if (n < 1)
    printf("\n\nN mora biti veći od 0. Unesite ponovo N: ");
} while (n < 1);

for (i = 1; i <= n ; i++)
    suma += i;

printf("\n\nSuma prvih %d prirodnih brojeva je: %d. \n", n, suma);

return 0;
```

FOR i WHILE iteracije

Postoji ekvivalencija između FOR i WHILE

U C programskom jeziku FOR se može konvertovati u WHILE i obrnuto

U zavisnosti od prirode problema, zavisi koja iteracija će se upotrebiti

Ako nešto treba obaviti određen unapred poznat broj puta, onda se koristi FOR

Ako se čeka da se neki uslov desi (a to nije unapred određeno brojem ponavljanja), bolje je koristiti WHILE

FOR i WHILE iteracije – primer

Napisati kod za računanje sume prvih *N* prirodnih brojeva, pri čemu se *N* zadaje na početku programa od strane korisnika (problem rešiti pomoću **WHILE** iteracije)

```
int n = 1, i, suma = 0;
printf("\n Program za racunanje sume prvih N prirodnih brojeva.\n\n");
printf("Unesite N: ");
do
{
    scanf("%d", &n);
    if (n < 1)
    printf("\n\nN mora biti veći od 0. Unesite ponovo N: ");
} while (n < 1);
i = 1;
while (i <= n)
{
    suma += i;
    i++;
};
printf("\n\nSuma prvih %d prirodnih brojeva je: %d. \n", n, suma);
return 0;
```

Naredbe bezuslovnog skoka

continue naredba nastavlja izvršavanje iteracije sa narednim prolazom, uz preskakanje svih naredbi tela tekuće iteracije (**while**, **for** i **do while**)

```
for(i = 0; i < n; i++)
{
    naredba1; naredba2;
    if (x[i] % 2)
        continue;
    naredba3; naredba4;
}
```

pocetak tela for iteracije;

i-ta vrednost nije deljiva sa 2

na sledeci prolaz ako i < N

Program nastavlja izvršavanje ulaskom u naredni ciklus
Isti efekat se postiže korišćenjem **IF-ELSE** selekcije

Naredba **goto** usmerava programski tok na datu labelu, nestrukturirano programiranje, **ZABRANJENO!**

E. Dijkstra, "Go-to statement considered harmful" *Comm. ACM*, vol. 11 (March1968), no. 3: 147–148.