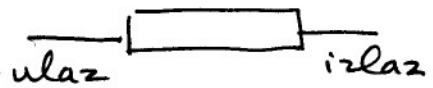
OSNOVNE NAREDBE U PROGRAMSKIM JEZICIMA

"Radni" dio programa (odn. cjeline) = niz naredbi.

Kakvih sve naredbi ima i kako izgledaju (tj. kako se pišu)?

Srećom, nema ih baš jako puno i zapravo odgovaraju osnovnim operacijama koje možemo napraviti u računalu.

Prvo što nam treba su ulaz i izlaz!
(sjetite se slikice: 

)

Počnimo s izlazom ("svaki algoritam ima neki izlaz").

C - poziv funkcije printf

printf (format-string , arg1, arg2, ...)

(ovog ne mora biti!)

format-string služi za kontrolu ispisa.
(string = niz znakova!)
- sadrži obične znakove - koji se doslovno prepisuju na izlaznu jedinicu
- i specifikacije pretvorbi (konverzije). Svaka od njih služi pretvorbi i pisanju sljedećeg po redu argumenta (jedan za drugim).

Klasični primjer:

```
printf ("␣a␣=␣%d\n", a);
```

ispisuje int a (vrijednost cjelobrojne varijable a) u obliku decimalnog broja (%d → decimal).

Izlaz:

␣a␣=␣35

(ovo je vrijednost (na pr.))

(\n = newline znak → završi red i "odi" u novi red)

Pasal Poziv procedure write (writeln):

```
writeln( " a = ", a );
```

FORTRAN Naredbe PRINT, odn. WRITE

```
PRINT *, ' a = ', a          (jednostruki navodnici!)
WRITE (*,*) ' a = ', a
```

značenje ovih *:
- u PRINT i druga u WRITE → "bez formata". Inače tu dođe broj FORMAT naredbe koja sadrži format ispisa
- prva u WRITE: tu dođe broj izlazne jedinice (datoteke) u koju se piše (6 ili * → stdout)

Ulaz ili čitanje podataka.

C Poziv funkcije scanf

```
scanf (format-string, arg1, arg2, ... )
```

Ovo čita znakove sa standardnog ulaza, interpretira ih prema specifikacijama u format-stringu i sprema rezultate kroz ostale argumente (arg1, ...)

U C-u, zbog prijenosa parametara po vrijednosti (v. Prog-C):

arg1 moraju biti pointeri - adrese varijabli kojima želimo dodijeliti vrijednost

(tj. adresa na koju se sprema rezultat čitanja)

Na primjer, ako učitavamo vrijednost cjelobrojne varijable n (int n), onda to radi ovako:

```
printf ("Upisi vrijednost cijelog broja n\n");
```

(uvijek si napišite poruku - da znate što treba upisati!) i onda:

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

"%d" — čitaj cijeli broj decimalno

&n — operator & (ispred varijable) daje adresu te varijable (pointer na nju)

- Na vježbama to pišemo pojednostavljeno:

```
scanf (n)
```

(što odgovara Pascalu i FORTRAN-u, ali NE RADI u C-u)

Pascal: Poziv procedure read

```
read (n)
```

FORTRAN: Naredba READ

```
READ *, n
```

ili

```
READ (*, *) n
```

U oba ova jezika pišem baš IME objekta (VARIJABLE) čiju vrijednost hoću učitati:

U C-u - moram napisati ADRESU tog objekta!

- Moj "pseudo-hrvatski" jezik:

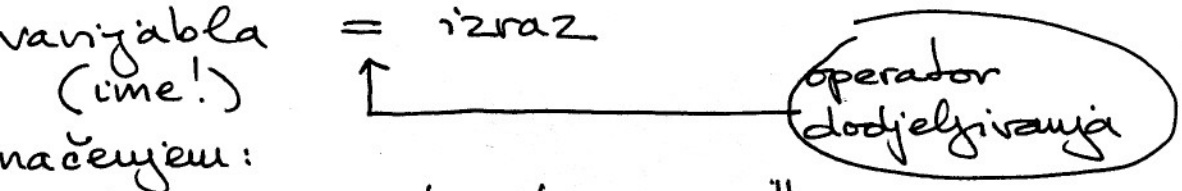
```
učitaj n
napiši n
```

(što je, valjda, dorolino jasno - bar za učenje algoritama).

Sad dolazimo na "prave" naredbe (u onoj kutiji iz slike —□—)

Osnovna naredba je tzv. naredba dodjeljivanja.

"Pseudo-jezični" oblik



sa značenjem:
- "izračunaj vrijednost izraza"
- "spremi tu vrijednost na mjesto rezervirano za varijablu"

(pa kažemo da varijabla dobiva tu vrijednost,)
tj. varijabli smo dodijelili vrijednost izraza.

C: varijabla = izraz

Pascal: varijabla := izraz

FORTRAN: varijabla = izraz

Napomena: operator = (odn. := u Pascalu) je OPERATOR DODJELJIVANJA vrijednosti (engl. ASSIGNMENT OPERATOR), tj. NIJE logički operator (preciznije – relacijski) za testiranje jednakosti (u C-u: ==, u FORTRANu: .EQ., a u Pascalu je = zaista relacijski operator)

- U pseudo-jeziku se katkad piše i

  varijabla ← izraz

(ideja je očita)

- Dvije napomene za C:
  - piše se IME (a ne adresa!) varijable, iako se sprema na tu adresu (razlog – slaganje tipova lijeve i desne strane)
  - Cijela ova naredba je opet IZRAZ, a vrijednost tog izraza je upravo ona ista vrijednost desne strane koja se sprema (tj. dodjeljuje varijabli).

Zašto u C-u smijemo pisati:

$m = n = 3$

(Što to radi i koja su značenja pojedinih komada ove naredbe)

- Ključna stvar – jezički i stvarno operativno:
  struktura zvana "izraz" je najkompliciranija stvar u jeziku (posebno za provjeru i prevođenje)

Zašto? Tu je skriveno bogatstvo jezika
= što sve možemo napraviti (operacije, funkcije) s razum objektima!

Pojednostavljeno:
- izraz sadrži konstante, varijable, operacije (tj. operatore – jer ima i unarnih i binarnih), pozive funkcija, ...
- objekti mogu imati različite, ali "kompatibilne" tipove.
  Na pr. zbrajam int i long int
  ili int i unsigned.
  Kojeg tipa je rezultat?
- Na kraju, cijela mijeduost izraza zna neki tip i taj mora odgovarati tipu varijable, inače još moramo napraviti "pretvaranje" tipova.

Klasični primjer u C-u:

"int" = (int) "double izraz"

sadrži zaokruživanje odbacivanjem – prema $\emptyset$.

"Pola" učenja nekog programskog jezika odlazi zapravo na strukturu izraz.

Kratko o standardnim operacijama (operatorima):

aritmetički: +, −, * (cijeli i realni brojevi)

dijeljenje: / → (C:) (F:) za cijele brojeve daje cjelobrojni kvocijent (EUKLID)

→ (P:) samo za realni rezultat (2/3 = 0.66....)

Pascal - div je cjelobrojno dijeljenje.

Ostatak kod cjelobrojnog dijeljenja:

C:	%	n % m
F:	MOD funkcija	MOD(n, m)
P:	mod operator	n mod m.

Potenciranje:

C:	funkcija pow	a^b = pow(a, b)
F:	operator **	a ** b
P:	nema!!	

mogu koristiti $a^b = (e^{\ln a})^b$, pa to pisati ovako exp(b * ln(a))

ali oprez - zbog zaokruživanja (ima slučajeva kad je oro LOŠE).

Primjeri:

C, P, F: x = 3.14 (i još ; za C, P) x = 3.14; x := 3.14 x = 3.14

Ako trebate π - NE OVAKO!!

- ili na punu točnost, ili recimo 4 · arctg(1)

3.14159 26535 89793 23846 26433....

Primjer: $y = \sin x - x^3$

C: `y = sin(x) - pow(x,3)`
P: `y = sin(x) - x*x*x` ($x^2 \rightarrow$ sqr(x))
F: `y = sin(x) - x**3`

— · —

relacijski operatori – uspoređivanje (rezultat logičkog tipa)

jednako, različito:

C: `==` , `!=` (! je unarna negacija)
P: `=` , `<>`
F: `.EQ.` , `.NE.`

veće, manje, ..., ...:

C: `>` , `<` , `>=` , `<=`
P: `>` , `<` , `>=` , `<=`
F: `.GT.` , `.LT.` , `.GE.` , `.LE.`

— · —

logički operatori: ne, i, ili (djeluju na logičkim mjeodnosima)

C: `!` , `&&` , `||`
P: not , and , or
F: `.NOT.` , `.AND.` , `.OR.`

- Tablica prioriteta operatora
- Redoslijed izvršavanja izraza (operacije, f-e, ...)
- Zagrade služe "nasilnom" mijenjanju prioriteta (podizraz u zagradi – prije ostalog!)

- Dakle, izraz je zaista kompliciran struktura (jezički – pravila pisanja, značenje – tj. pravila računanja)

- Ulaz, izlaz, dodjeljivanje → tzv. jednostavne naredbe ("rad s podacima").

- Složene naredbe – "rad s postupcima", tj. upravljanje naredbama ili blokovima naredbi

- Prvo – u C, P mogu niz naredbi "zatvoriti u zagrade" i proglasiti jednom tzv. složenom naredbom: (ili blok)

C:
```
{
⋮
}
```

P:
```
begin
⋮
end
```

(u F: toga nema direktno, ali sbarno ima, samo se "zagrade" pišu kao dijelovi naredbi)

- Osnovna podjela složenih naredbi:
  - uvjetne (if, case) – izbor alternativa
  - petlje (while, for) – ponavljanje !!
  - ... ima još

- Za pisanje osnovnih algoritama dovoljno mi je: if, while, for.

Uvjetne naredbe – kratko – služe grananju!

① tzv. if – then (bez else) radi ovo:

ako je uvjet ispunjen (tj. istinit) onda izvrši naredbu (ili blok naredbi)

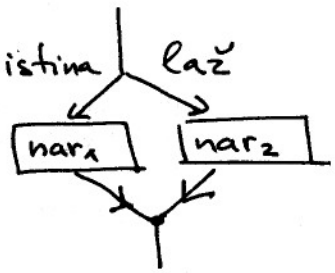
Podrazumijeva se i to da ako uvjet nije ispunjen (lažan), onda ne izvršavam tu naredbu, nego nastavljam izvršavanje na prvoj sljedećoj naredbi iza ove (ako...onda).

Što je uvjet? → izraz koji ima logičku vrijednost (laž, istina)

(Već rekosmo – u C-u: laž ⇔ 0, istina ⇔ ≠ 0).

② tzv. if-then-else:

ako je uvjet ispunjen onda
    naredba$_1$
inače
    naredba$_2$



Zapiši:

C:
```
if (uvjet)
    naredba;
```

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

PAS:
```
if uvjet then
    naredba
```

```
if uvjet then
    naredba1
else
    naredba2
```

FOR:
```
IF (UVJET) THEN
⋮ } naredbe
ENDIF
```

```
IF (UVJET) THEN
⋮ } naredbe1
ELSE
⋮ } naredbe2
ENDIF
```

(Tu "pišem" previše – izvršit ću nešto – možda da, možda ne
ili samo jedan od 2 bloka!)

– Ova druga naredba (case) = izbor između VIŠE alternativa.

– Petlje – pišem "malo", ali to ponavljam puno puta.

Osnovna petlja - tzv. while je slična if.
⇔ ponavljanje pod kontrolom logičkog uvjeta.

To radi ovako:

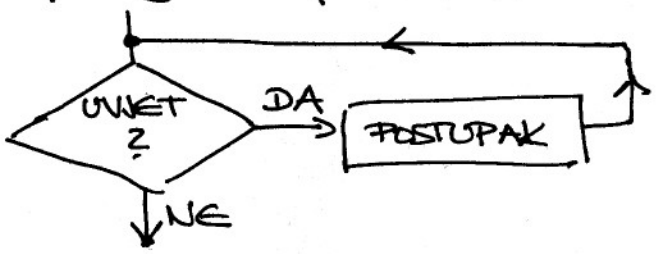
sve dok je uvjet ispunjen ponavljaj
[postupak (naredbu ili blok)

Naravno, ideja je da postupak ima neki utjecaj na uvjet! (inače → ∞ petlja!). Zašto?

Malo detaljnije - while radi ovo:

- prvo izračuna uvjet
  - ako je istinit, onda izvrši postupak i "skoči na početak"
  - opet izračuna uvjet, i tako redom...

... sve dok jednom ne dobije da je uvjet lažan (što se moglo i prvi puta dogoditi). Tada ne radi postupak ("preskače" ga), naredba while je gotova, a izvršavanje se nastavlja na prvoj sljedećoj naredbi



- Bitno - test uvjeta ide PRIJE postupka, tako da se on ne mora napraviti (tj. 0 puta se ponovi!)
- Dakle - da izađem iz ponavljanja (ako sam tamo ušao) moram jednom dobiti (prije ili kasnije) da je

uvjet lažan

Zapiši:

C: while (uvjet)
    naredba;

PAS: while uvjet do
    naredba

F: stari nema - nego preko GO TO (skokovi!)
novi ima:

```
DO WHILE (uvjet)
  ⋮        } naredbe
ENDDO
```

Druga bitna petlja (tzv for) je ponavljanje nekog postupka zadani broj puta - pod kontrolom brojača (C dozvoljava malo više - v. dolje).
Osnovna svrha - obrada nizova (polja), matrica.

Pseudo-zapis:

za ime brojača (varijabla) = poč. vrij (od) do krajnja vrij.
ponavljaj
[postupak

Značenje - za svaku vrijednost brojača od početne do krajnje - ponovi postupak.
- početna i krajnja vrij. (uostalom i brojač) su CIJELI brojevi
- "korak" brojača je 1 (za toliko se poveća! - svaki put nakon jednog ponavljanja)
- ako je na početku - početna > krajnje nema ponavljanja (0 puta)
- inače - broj ponavljanja = krajnja - poč + 1

Zapis u C-u – bitno šire značenje:

for ($izraz_1$; $izraz_2$; $izraz_3$)
    naredba

Standardno: $izraz_1$, $izraz_3$ su dodjeljivanja (to jesu izrazi u C-u!)

$izraz_2$ je relacijski izraz.

To radi ovako – prevedeno u while:

$izraz_1$;
while ($izraz_2$) {
    naredba ;
    $izraz_3$;
}

- Ako nema $izraz_1$ i/ili $izraz_3$ – ovdje kao da ih zaista nema.
- Ako nema $izraz_2$ – uzima se kao da je stalno ISTINA!

```
for ( ; ; ) {
    ....
}
```

} → do-petlja iz koje se (valjda) izlazi na neki drugi način (break, return)

PAS: for var = $izraz_1$ to $izraz_2$ do
                downto
    naredba
(to – korak 1, downto – korak –1, mogu i drugi tipovi)

FOR: DO var = $izraz_1$, $izraz_2$ (, $izraz_3$)
    ....
    END DO
ili brojem naredbe!