Razlikovni rječnik Pascal - C - FORTRAN

Pascal	C	FORTRAN

`var i,n:integer;`
`  x,y:real;`
`  a:array [1..100] of integer;`
	`int i,n;`
	`float x,y;`
	`int a[100];`
		`INTEGER i,n`
		`REAL x,y`
		`INTEGER a(100)`

`writeln('Unesi prirodan broj');` `read(n);`	`printf("Unesi prirodan broj\n");` `scanf("%d",&n);`	`PRINT *,'Unesi prirodan broj'` `READ *,n`
`x := 3.14;` `y := sin(x) - x*x*x;`	`x = 3.14;` `y = sin(x) - pow(x,3);`	`x = 3.14` `x = SIN(x) - x**3`

`if (n=0) then`
`begin`
`  ⋮`
`end;`
	`if (n==0)`
	`{`
	`  ⋮`
	`}`
		`IF (n .EQ. 0) THEN`
		`  ⋮`
		`ENDIF`

`for i:= 1 to n do` `  a[i] := 0;`	`for (i=0; i<n; ++i)` `  a[i] = 0;`	`  DO 10, i=1,n` `10  a(i) = 0`

`while (n>0) do`
`begin`
`  ⋮`
`end;`
	`while (n>0)`
	`{`
	`  ⋮`
	`}`
		`10  IF (n .LE. 0) GOTO 20`
		`  ⋮`
		`  GOTO 10`
		`20  CONTINUE`

`repeat`
`  ⋮`
`until (n<=0);`
	`do {`
	`  ⋮`
	`} while (n>0);`
		`10  CONTINUE`
		`  ⋮`
		`  IF (n .GT. 0) GOTO 10`

`procedure ispis(n:integer);`
`begin`
`  writeln('Rezultat je ',n);`
`end;`
	`void ispis(int n)`
	`{ printf("Rezultat je %d\n",n);`
	`}`
		`SUBROUTINE ispis(n)`
		`INTEGER n`
		`PRINT *,'Rezultat je ',n`
		`RETURN`
		`END`

`(* Aritmeticka sredina *)`
`function sredina(x,y:real):real;`
`begin`
`  sredina := (x+y)/2;`
`end;`
	`/* Aritmeticka sredina */`
	`float sredina(float x, float y)`
	`{ return (x+y)/2;`
	`}`
		`C  Aritmeticka sredina`
		`  REAL FUNCTION sredina(x,y)`
		`  REAL x,y`
		`  sredina = (x+y)/2`
		`  RETURN`
		`  END`

UVOD U PROGRAMIRANJE

Programiranje = pisanje algoritama u nekom jeziku
Obično je konačni cilj = izvršavanje tog programa na računalu

Jezik za zapis algoritama:
- u fazi razvoja algoritma - može bilo što, samo da korektno i jasno odražava strukturu algoritma
  (Napomena: što je algoritam - v. ranije)
  Obično koristimo "materinji" jezik, samo je dolik zapisa prilagođen standardnim programskim jezicima
  (tzv. pseudo-jezik)
- na kraju - pišemo u nekom programskom jeziku za kojeg postoji compiler - prevoditelj u tzv. strojni jezik, tako da se program može izvršiti na računalu.
  (compile - link - run / execute)

Programski jezici, kao i obični, imaju svoja pravila kako se korektno pišu programi.
- Postoje razne vrste pravila
  - leksička = slaganje znakova u riječi
  - sintaktička = slaganje riječi u rečenice (programe)
    [To odgovara običnoj gramatici - pravila pisanja]
  - semantička = smislenost rečenica / programa.

Ta pravila su stroža nego za obične jezike, s ciljem efikasnog prevođenja u strojni jezik (naredbe), što uključuje i provjeru korektnosti
(Nepravilno napisan program se ne prevodi!)

- Postoji i matematička teorija jezika!

Trenutno nam nije cilj naučiti detaljno sva pravila nekog programskog jezika (poput C-a), već naučiti sastavljati i pisati tzv. osnovne algoritme tako da ih kasnije lako možemo korektno zapisati kao programe.

Dakle - naglasak na korektnom obavljanju posla (kako se rade neke stvari), a ne na tome kako se to strogo piše.

— • —

Gruba struktura programa u standardnim programskim jezicima:

- Cijeli program može imati više dijelova (blokova, modula) od kojih je jedan glavni (on se prvi počinje izvršavati).
- Svi blokovi imaju sličnu strukturu i sastoje se iz 2 bitna dijela:

(1) deklaracije objekata s kojima će se nešto raditi (i tu objekti dobiju ime, tip i slične atribute. Jedna od svrha deklaracija je rezervacija memorije za te objekte (objekti dobiju svoje adrese)

(2) naredbe (ili popis naredbi) koje treba izvršiti (s navedenim objektima, s tim da možemo sami kreirati i umještavati objekte odredenim naredbama - ali o tom kasnije!)

Već smo rekli da postoje tzv. jednostavni tipovi podataka (operacije su direktno izvodive arhitekturom procesora, odn. računala).

Većina modernih programskih jezika dozvoljava konstrukciju tzv. složenih tipova podataka, odgovarajućim konstrukcijama u deklaraciji

(nizovi ili polja - array = niz podataka ISTOG tipa,
zapisi - strukture ili rekordi = niz podataka različitih tipova,
itd.)

Na početku koristimo samo jednostavne tipove podataka, a kasnije ćemo raditi s nizovima (vektori, matrice).

Upozorenje: u većini programskih jezika vrijedi pravilo da svaki objekt koji koristimo mora na neki način biti deklariran (naveden), i to prije nego što ga koristimo.

Ovo pravilo primarno služi za kontrolu grešaka (tj. korektnosti programa), a najčešće još i za rezervaciju memorije za objekt (tj. za njegovu vrijednost).

Klasično se ovakvi objekti zovu varijable. Deklaracija rezervira prostor u memoriji za objekt, tj.

- varijabla dobiva adresu (prva za cijeli prostor).

Dakle, takav objekt uvijek zauzima isti (fiksni) prostor u memoriji, a sadržaj tog prostora (= vrijednost objekta, odn. varijable) se može mijenjati.
(Zato i naziv "varijabla" - može imati razne vrijednosti)

Upozorenje: deklaracija rezervira prostor, ali ne sprema vrijednost u njega (osim ako to eksplicitno ne navedemo).

Drugim riječima: osim deklaracije, svakoj "varijabli" treba dodijeliti neku vrijednost prije korištenja!

Kako? Čitanjem ili naredbom dodjeljivanja (tj. računanjem nekog izraza i spremanjem te vrijednosti na odgovarajuće mjesto).

Još jedna, čisto "jezična" napomena:
imena objekata se moraju pisati po odredenim pravilima. Tipična pravila:

- ime = jedna riječ (nema praznina - osim u starom FORTRANU)
- riječ počinje slovom (a-z, A-Z) ("alfa" znak)
- riječ može sadržavati i znamenke (0-9) ("num" znak)
- katkad je dozvoljen i "underline" znak ("donja crta")

na_primjer_47

- ne smije sadržavati neke druge ("specijalne") znakove
- maksimalna duljina može biti ograničena!

U većini jezika su kao imena objekata zabranjene tzv. ključne riječi - koje imaju posebna značenja u samom jeziku (na pr. "imena" naredbi)
(Primjer: if, for, while, ...)

- Razlog za ovo pravilo je očit: → lakše prevođenje (nema dvosmislenosti!)

- Tipični primjeri deklaracije objekata standardnih jednostavnih tipova (cijeli brojevi, realni brojevi, znakovi, logičke vrijednosti), u programskim jezicima C, Paslal, FORTRAN

C:

```
int i, n;                  ← cijeli brojevi s predznakom
unsigned int k1, k2;       ← cijeli brojevi BEZ predznaka
```

(riječ int se može ispustiti)

U C-u još možemo "variirati" dužinu cijelih brojeva riječima short, long. Na pr:

```
unsigned long file_length;
```

Pascal: `var i,n : integer;` (⇔ int u C-u)

Stroži standard "nema" ovuge cjelobrojne tipove, ali većina implementacija dozvoljava

```
k: longint;   (⇔ long int u C-u)
```

(Standard nema imena za tip cijelih brojeva BEZ predznaka)

FORTRAN: `INTEGER i, n`

(riječ INTEGER se može pisati i malim slovima, ali se tradicionalno - po standardu do FORTRAN 77, koriste VELIKA slova!)

Varijacije u dužini:

```
INTEGER*2   kratki
INTEGER*8   dugi
```

↑ broj byte-ova za spremanje cijelog broja ≙ predznakom.

Ni FORTRAN "ne zna" cijele brojeve BEZ predznaka.

Napomena: od Fortran 90 standarda, nadalje, pravilo pisanja deklaracija se promijenilo u:

```
INTEGER :: i, n.
```

Većina numeričkih biblioteka (recimo na Netlibu) još uvijek je pisana u F77 standardu.

— . —

Realni tipovi (single, double, extended precision)

C

```
float x, y;          ← single precision
double dx, dy;       ← double precision
long double ex, ey;  ← extended precision
```

Za srabi konkretni C treba provjeriti točno značenje tipova float, double, long double!

Pascal

```
(var)  x, y : real;
```

Standard Pascal ima samo tip "real", i on obično znači "double".

Bivši Turbo Pascal (danas Delphi) i GNU Pascal imaju tipove single, double, extended (IEEE, 4, 8, 10 byte-a)

```
(var)  x, y : single;
       dx, dy : double;
       ex, ey : extended;
```

Napomena: U TP(6,7), tip real ne odgovara niti jednom od ova 3 tipa, već ima 6 byte-a (naslijeđe iz pradavnih vremena i radi preko biblioteke, a ne hardware-om, tj. užasno je spor).

FORTRAN

```
REAL x, y                 (<=> single)
DOUBLE PRECISION dx, dy   (<=> double)
```

i nema standardnog imena za extended.

Vrlo rijetko prolazi slična deklaracija kao kod INTEGER, preko broja byte-ova:

```
REAL*8  dx, dy   (ovo je OK)
REAL*10 ex, ey   (rijetko).
```

Opet, od F90, dozvoljeno je svašta, ali extended
vrlo rijetko radi korektno.

Nažalost, jezik za "numerku" ima najslabiju podršku
za odgovarajuće tipove.

Znakovni tip, tj., tip u kojem su dozvoljene vrijednosti
znakovi (u odgovarajućem kôdu):

C char ch;

s tim da je char zapravo "kratki integer" koji sadrži
kôd znaka u odgovarajućem skupu znakova (recimo,
proširenu ASCII).

Pascal (var) ch : char;

FORTRAN CHARACTER*1 ch

Zadnji jednostavni tip su logičke vrijednosti - laž, istina.

C - nema posebni tip za logičke vrijednosti.
Standardna interpretacija logičkih vrijednosti je preko
cijelih brojeva (bez obzira na predznak)

$$0 \Leftrightarrow \text{laž}$$
$$\neq 0 \Leftrightarrow \text{istina}$$

s tim da kad se traži konkretna vrijednost za istinu,
obično se koristi

$$1 \Leftrightarrow \text{istina}.$$

U "pedantnom" programiranju možemo koristiti naredbe
pretprocesora da definiramo "vlastite" logičke vrijednosti

```
#define FALSE 0        (supstitucija 0 umjesto
                        FALSE, gdje god se FALSE
                        javlja)

#define TRUE 1
```

Pascal: (var) flag : boolean

gdje je boolean tzv. pobrojani tip s međuvrijednostima

(false , true)
⇕ ⇕
0 1

FORTRAN: LOGICAL flag

a pripadne međuvrijednosti se zovu (ripišu!) .FALSE.
i .TRUE.

Kad smo već kod deklaracija, spomenimo još kako se deklariraju NIZOVI (odn. matematički rečeno, vektori).

NIZ - općenito = složeni tip podataka, sastavljen od podataka ISTOG tipa, a pojedini podaci se razlikuju po položaju u nizu, tj. po indeksu.

Matematički, niz a od n podataka nekog tipa, odgovara uređenoj n-torki (vektoru) podataka:

$$(a_1, a_2, \ldots, a_n)$$

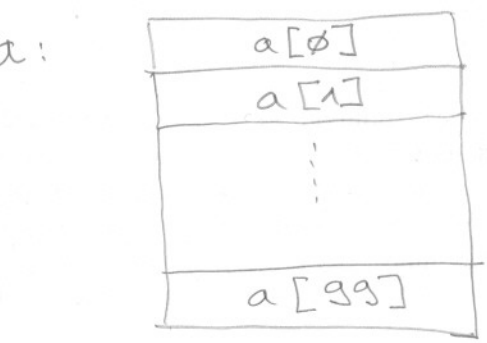
a pojedinom podatku a_i se pristupa indeksom i, koji kaže da tražimo i-ti podatak (komponentu, element) vektora a.

C - je malo "šašav", jer se komponente broje od 0, a ne od 1 (razlog je tzv. aritmetika adresa - v. kasnije)

Deklaracija:

```
int a[100];
```

rezervira prostor (memoriju) za 100 varijabli tipa int (i to u bloku susjednih adresa), a poredak tih 100 objekata (s imenima, odn. indeksima) je:



ime a = adresa prvog elementa a[0]
(i to se koristi za računanje adresa svih ostalih elemenata:
a[i] - početna adresa
= a + i * duljina jednog elem.

U Pascalu i FORTRANu se elementi standardno indeksiraju od 1 (ako ne kažemo drugačije):

Pascal: (var) a: array [1..100] of integer

a može i

a: array [0..99] of integer.

FORTRAN INTEGER a(100)

ili INTEGER a(0:99)

(Opel F90 ima svoje dodatke!)

- Pojedini element polja a (niza a) dobivamo pisanjem (cjelobrojnog) indeksa u istim zagradama kao i u deklaraciji:

a[i] C, Pascal

a(i) FORTRAN

— . —

Time smo, barem elementarno, gotovi s deklaracijama (bar za varijable). Ostatak deklaracija se malo više razlikuje od jezika do jezika.

Na UuR slično nećemo pisati deklaracije za većinu algoritama (v. konačne programe!)

— . —

A sad idemo na "interesantniji" dio programa (ili pojedine cjeline) — koji se sastoji od niza naredbi.

Kako izgledaju naredbe i kakvih sve naredbi ima?

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

Prvo što nam treba je ulaz i izlaz.

```
while (izraz)
    naredba
```

→ izrač. $izraz$, ako je $\neq 0$, izvrši naredbu i opet izrač. $izraz$!

..., sve dok jednom ne dobijemo 0.
Onda izvš. nastavlja na PRVOJ naredbi iza.

```
for (izraz1 ; izraz2 ; izraz3)
    naredba
```

$\Leftrightarrow$

```
izraz1 ;
while (izraz2) {
    naredba ;
    izraz3 ;
}
```

Standard:
- $izraz_1$, $izraz_3$ su dodjelj
- $izraz_2$ je relacijski izraz

Ako nema $izraz_1$, $izraz_3$ — onda kao da ih nema a ako nema $izraz_2$ - uzima se kao da je stalno ISTINA!

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

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