



Programiranje I – RIN Računalništvo I – MA

OSNOVNI GRADNIKI PROGRAMSKIH JEZIKOV 1. DEL

Vsebina – pregled gradnikov PJ

- Primer preprostega programa
- Spremenljivke (kasneje *predmeti*)
- Osnovni tipi (kasneje *razredi*)
- Kaj je polje?
- Operatorji → izrazi (aritmetični, logični, ...)
- Stavki in bloki
- Kontrola toka programa
- Funkcije in programi



Preprost program

```
/**
 * Preprost program Pozdrav, ki na standardni izhod izpise
 * "Lep pozdrav :-)".
 */

class Pozdrav {
    public static void main(String[] args) {
        System.out.println("Lep pozdrav :-)"); // Prikaže niz.
    }
}

Mšinjek@tarije:~/glavna$ javac Pozdrav.java // ...
```

Spremenljivke – kaj so?

- spremenljivka je predmet, ki ima lastnost, da:

- **ohrani** vrednost, ki ji jo **priredivo**:

spremenljivka = vrednost

$$a = 7$$

- kadarkoli lahko **pogledamo** njeno vrednost

druga Spremenljivka = **spremenljivka**

$$b = a$$

- konstanta je posebna vrsta spremenljivke, kateri vrednost priredimo in jo kasneje ne moremo spreminjati

Spremenljivke – pravila poimenovanja

- Spremenljivke naj imajo smiselna imena:
 - *ime_spremenljivke, imeSpremenljivke*
 - *povrsina, najvVrednost, stevec*
 - ...
 - v naših primerih bodo uporabljena krajša imena:
a, b, c, i, j, k, ...
- Niso vsa imena dobra:
 - pravila poimenovanja, rezervirane besede PJ

Spremenljivke – (osnovni) tipi

- **tip spremenljivke** =
vrsta podatka, ki ga spremenljivka hrani
- Osnovni tipi:
 - cela števila (int) – `1, -4, 0, ...`
 - števila v plavajoči vejici (float) –
`0,1E+45 (0,1*1045) ; 0,456E-1 (0,456*10-1) ; ...`
 - Boolove spremenljivke (boolean) – `true, false`
 - znaki (char) – `'a', 'z', '3', '@', ...`
 - nizi (string) – `"ime", "EMSO", ...`
- Sestavljeni tipi in razredi



Tipi spremenljivk v JAVI

- JAVA pozna 4 osnovne tipe spremenljivk:
 - Cela števila: *byte*(8), *short*(16), *int*(32), *long*(64)
 - Števila v plavajoči vejici: *float*(32), *double*(64)
 - Boolove spremenljivke (1)
 - Znaki (unicode 16)
- Skupaj s podtipi 8
- Vse ostalo so *razredi*

Spremenljivke – deklaracija, definicija

- Deklaracija spremenljivk ima dvojni namen:
 1. spremenljivkam določimo **ime** in **tip**
 2. prevajalniku povemo **koliko spomina** naj zaseže za potrebe določene spremenljivke
- Če smo bolj natančni se uporablja izraz deklaracija za označevanje 1. točke, 2. točki pa pravimo definicija spremenljivke
- V nekaterih PJ je deklaracija spremenljivk **implicitna** (npr.: C, FORTRAN, PERL, PYTHON, PHP, ...)
 - v JAVI je **explicitna**



Zakaj uporaba tipov? (1)

- V programskem jeziku JAVA mora biti za vsako spremenljivko, ki jo uporabimo, določeno, iz katere množice je = katerega tipa je (**eksplicitna deklaracija**)
 - kasneje: iz katerega razreda je

- Čemu to?

Zakaj uporaba tipov? (2)

- Kaj pomenijo naslednje prireditve:

```
char b;
```

```
b = 'T';           // v redu
```

```
a = b;           // morda, če je char a;
```

- Tipe potrebujemo za to, da lahko preverimo, če so naši ukazi smiselni
 - kasneje bomo spoznali razrede, katerih ena od vlog je povsem enaka

Zakaj uporaba tipov? (3)

- Kaj pa pri programskih jezikih, kjer je deklaracija tipov **implicitna**?
- Pri teh jezikih so spremenljivke še vedno iz neke množice (so določenega tipa), le da se ta določi implicitno – ob prvi uporabi:

`a = 3;` // a je torej celo število

`b = '4';` // b je znak

`c = a + b;` // hm, kaj pa je sedaj c?

Zakaj uporaba tipov? (4)

`a = 3;`

`b = '4';`

`c = a + b;`

- Imamo (vsaj) naslednji možnosti:

1. izraz ni dovoljen – **napaka**: to velja v JAVI in v večini programskih jezikov
2. izraz ni napaka in sistem poskuša samodejno (implicitno) narediti pretvorbo tipa ene spremenljivke v tip druge spremenljivke – **type casting**
 - `b` se pretvori v celo število, izraz se sešteje in `c` je potem ponovno celo število
 - težava nastopi, ko imamo `b = 'z'`
 - Je še kaka možnost?

- Eksplicitna VS. implicitna **pretvorba tipov**



Spremenljivke – primeri, inicializacija

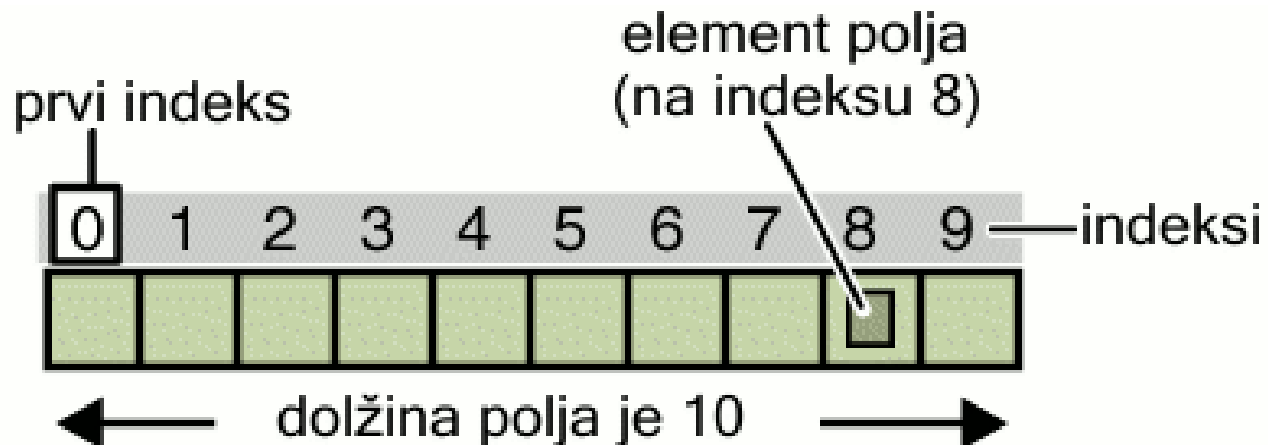
```
boolean rezultat = true;  
char velikic = 'C';  
byte b; // deklaracija spremenljivke brez prireditve vrednosti  
short s = 10000;  
int i = 100000;  
double d1 = 123.4; // pozor, decimalna pika !!!  
double d2 = 1.234e2; // isto, le v znanstveni notaciji
```

v JAVI je **implicitna**

Inicializacija spremenljivke = prireditev vrednosti

Polje (*predmetov*)

- **Polje** je predmet (objekt), ki lahko vsebuje **fiksno število vrednosti** istega tipa
- **Dolžina polja** se določi ob njegovi kreaciji
- Do elementov polja dostopamo prek njihovih indeksov





Polje – deklaracija, definicija, inicializacija

■ Primer:

```
class PoljeDemo {
    public static void main(String[] args) {
        int[] toPolje; // deklaracija polja celih št.
        toPolje = new int[10]; // alokacija spomina za 10 celih št.
        toPolje[0] = 100; // inicializacija prvega elementa
        toPolje[1] = 200; // inicializacija drugega elementa
        ...
        toPolje[9] = 1000; // inicializacija zadnjega elementa
        System.out.println("Element na indeksu 0: " + toPolje[0]);
        System.out.println("Element na indeksu 1: " + toPolje[1]);
        ...
        System.out.println("Element na indeksu 9: " + toPolje[9]);
    }
}
```

Operatorji

- Kaj sedaj, ko smo deklarirali (in inicializirali) spremenljivke?
- Vloga operatorjev = izvajanje **operacij** nad spremenljivkami (operandi) → **izrazi**
- Operatorje delimo v skupine glede na:
 - prednost izvajanja
 - število operandov
 - tip
 - morda še kaj ...



Operatorji v JAVI – po prednosti

- postfix `izr++ izr--`
- unarni `++izr --izr +izr -izr ~ !`
- multiplikativni `* / %`
- additivni `+ -`
- premični `<< >> >>>`
- relacijski `< > <= >= instanceof`
- enakosti `== !=`
- bitni AND `&`
- bitni XOR `^`
- bitni OR `|`
- logični AND `&&`
- logični OR `||`
- ternarni `? :`
- prireditveni `= += -= *= /= %= &= ^= |= <<= >>= >>>=`

**določanje prednosti
izvajanja operatorjev?**

Aritmetični in logični operatorji

■ Aritmetični operatorji:

□ osnovni: + - * / %

□ prireditveni: = += -= *= /= %=

(a = a+1 ≡ a += 1)

Pozor !!!

■ Logični operatorji:

□ osnovni: && || !

□ relacijski, primerjalni: < > <= >= == !=

Razlika med

priréditvijo in *primerjavo*

Bitni operatorji in operator “? :”

- Uporabljajo se za delo s posameznimi biti
 - osnovni: & | ^ ~
 - premični: << >> >>>
 - prireditveni: <<= >>= >>>=
- Ternarni operator “? :”
(**b = (a > c) ? a : c**)

Posebna operatorja ++ in --

- Operatorju ++ pravimo tudi *inkrement*, saj poveča vrednost poljubne celoštevilске spremenljivke za 1
- Operatorju -- pa pravimo tudi *dekrement*, saj zmanjša vrednost poljubne celoštevilске spremenljivke za 1
- Operatorja se lahko uporabljata v dveh oblikah:
 - prefiksni: ++i --j
 - postfiksni: i-- j++

Primer uporabe operatorja ++

```
class PrePostDemo {  
    public static void main(String[] args) {  
        int i = 3;  
        i++;  
        System.out.println(i); // "4"  
        ++i;  
        System.out.println(i); // "5"  
        System.out.println(++i); // "6"  
        System.out.println(i++); // "6"  
        System.out.println(i); // "7"  
    }  
}
```

Zakaj se 2x izpiše 6-ica?

Kaj je izraz?

- Za *izraz* velja, da:
 1. je sestavljen iz spremenljivk, operatorjev in klicev funkcij/metod (???) po sintaktičnih pravilih določenega programskega jezika;
 2. mu lahko vedno določimo vrednost, ki je določenega tipa

- Kakšnega tipa je vrednost nekega izraza?



Primeri izrazov v JAVI

```
int stevec = 0;
```

```
tabela[0] = 100;
```

```
System.out.println("Element 1 na 0-tem mestu: " + tabela[0]);
```

```
int rezultat = 1 + 2; // rezultat je sedaj 3
```

```
if(vred1 == vred2)
```

```
    System.out.println("vred1 = vred2");
```

Sestavljeni izrazi

- Izraze lahko sestavljamo/gneznimo
 - pri določanju vrednosti sestavljenih izrazov se uporabljajo pravila prednosti operatorjev (in oklepaji)
- Primeri sestavljenih izrazov:
 - $1*2*3$
 - $x+y/100$ // dvoumni zapis; ni priporočljiv
 - $(x+y)/100$ // nedvoumni zapis
 - $x+(y/100)$ // nedvoumni zapis

Stavki

- Stavke nekega PJ lahko primerjamo s stavki naravnega jezika
 - **stavek** v PJ tvori neko zaključeno celoto, ki se izvede in mu včasih pravimo tudi **ukaz**
 - v JAVI stoji na koncu vsakega stavka podpičje (;)

- Primeri stavkov v JAVI:

```
aVred = 8933.234 ; // prireditveni stavek
```

```
aVred++ ; // inkrement stavek
```

```
double aValue = 8933.234 ; // deklaracijski stavek
```



Bloki

- **Bloki** so množice nič ali več stavkov

- Primer:

```
class BlokDemo {
    public static void main(String[] args) {
        boolean pogoj = true;
        if (pogoj) { // zacetek 1. bloka
            System.out.println("Pogoj je resničen.");
        } // konec 1. bloka
        else { // zacetek 2. bloka
            System.out.println("Pogoj ni resničen.");
        } // konec 2. bloka
    }
}
```

Kontrola toka programa (1)

- Ukazi/stavki nekega programa se izvajajo *eden za drugim* (**sekvenčno** ali **zaporedno**)
- Primer:
 1. $a = 2;$ *shranimo vrednost 2 v a*
 2. $a += 3;$ *prištejemo a-ju 3*
 3. $b = a+4;$ *shranimo v b vrednost v a povečano za 4*
 4. ... *...*

Kontrola toka programa (2)

- Programski jeziki vsebujejo ukaze, ki omogočajo “rušenje” zaporednosti izvajanja drugih ukazov
- Taki ukazi vplivajo na tok programa tako, da omogočajo:
 - vejitve
 - zanke
 - skoke

Vejitve

- Omogočajo izbiro med dvema ali več “potmi” izvajanja programa glede na nek pogoj
- Pogoj je lahko:
 - logični (izid `true/false`) → 2 možne “poti”
 - vezan na vrednost neke spremenljivke
→ več možnih “poti” izvajanja



Vejitve – “if-then [-else]

- Če je **pogoj** izpolnjen, potem ...[sicer ...]
- Primer (*bremzanje s kolesom*):

```
void bremzaj() {  
    if (sePremika) {  
        trenutnaHitrost--; }  
    else {  
        System.err.println("Kolo že stoji!");  
    }  
}
```



Vejitve – “switch”

- Za razliko od “if...” stavka omogoča vejitev na več kot dve možne “poti”
- Primer:

```
class SwitchDemo {  
    public static void main(String[] args) {  
        int mesec = 8;  
        switch (mesec) {  
            case 1: System.out.println("Januar"); break;  
            case 2: System.out.println("Februar"); break;  
            ...  
            case 12: System.out.println("December"); break;  
            default: System.out.println("Napaka."); break;  
        }  
    }  
}
```

Zanke

- Omogočajo ponavljanje istega opravila večkrat (npr.: za vse podatke)
- Izvajanje zank je vedno vezano na nek pogoj – podobno kot pri vejitvah



Zanke – “while”

- Dokler je **pogoj** izpolnjen ponavljaj ukaze v zanki (*lahko se nikoli ne izvedejo !!!*)
- Primer:

```
class WhileDemo {  
    public static void main(String[] args) {  
        int stej = 1;  
        while (stej < 11) {  
            System.out.println("Prestel do: " + stej);  
            stej++;  
        }  
    }  
}
```



Zanke – “do . . . while”

- Ponavljalj ukaze v zanki dokler je **pogoj** izpolnjen (*vsaj enkrat se izvedejo !!!*)
- Primer:

```
class DoWhileDemo {
    public static void main(String[] args) {
        int stej = 1;
        do {
            System.out.println("Prestel do: " + stej);
            stej++;
        } while (stej <= 11);
    }
}
```

Zanke – “for”

- **for** zanka je daleč najbolj uporabljana zanka, ker lahko z njeno pomočjo:
 - obiščemo elemente nekega polja
 - v enakomernih korakih spreminjamo vrednost neke spremenljivke
 - na enostaven način opravljamo iteracije

- Splošna oblika:

```
for (inicijalizacija; pogoj; korak) {  
    [stavki]  
}
```



Zanke – “for” – primeri

■ Primer 1:

```
class ForDemo {
    public static void main(String[] args) {
        for (int i=1; i<11; i++) {
            System.out.println("Prestel do: " + i);
        }
    }
}
```

■ Primer 2:

```
class RazsirjenForDemo {
    public static void main(String[] args) {
        int[] stevila = {1,2,3,4,5,6,7,8,9,10};
        for (int i : stevila) {
            System.out.println("Prestel do: " + i);
        }
    }
}
```

Neskončne zanke

- Kdaj so uporabne neskončne zanke?
- Primer 1:

```
while (true) {  
    //neskoncna while zanka  
}
```

- Primer 2:

```
for ( ; ; ) {  
    //neskoncna for zanka  
}
```

Skoki

- **Skoki** – kot že samo ime pove – omogočajo “skok” na poljubno mesto v programu
 - to mesto je lahko **označeno** s t.i. labelo (*label*)
- Če skok sledi nekemu (logičnemu) **pogoju**, mu pravimo **pogojni skok**
 - velika večina skokov v PJ je pogojnih

Skoki – “break”

- Poznamo dve vrsti **break** stavka: neoznačeno in označeno
- Služi predčasnim izhodom iz **for**, **while** ali **do...while** zank
 - izhod iz “trenutne” zanke – neoznačeni **break**
 - izhod iz poljubno vgnezdene zanke – označeni **break**
 - **takoj** “skoči” iz zanke



Skoki – “break” – primer (1)

■ Neoznačeni **break** (iskanje elementa v polju):

```
class BreakDemo {
    public static void
        main(String[] args) {
        int[] polje = {32,87,3,589,12,
            1076,2000,8,622,127};
        int iscem = 12;
        int i;
        boolean nasel = false;
        for (i=0; i<polje.length; i++) {
            if (polje[i] == iscem) {
                nasel = true;
                break;
            }
        }
    }
}
```

```
        if (nasel) {
            System.out.println("Našel " +
                iscem + " na indeksu " + i);
        } else {
            System.out.println(iscem +
                " nisem našel v polju");
        }
    }
}
```




Skoki – “break” – primer (2)

■ Označeni **break** (iskanje elementa v 2D polju):

```
class BreakDemo2 {
    public static void
        main(String[] args) {
        int[][] polje2D = { {32,87,3,589},
                          {12,1076,2000,8},
                          {622,127,77,955} };

        int iscem = 12;
        int i;
        int j = 0;
        boolean nasel = false;
isci:
        for (i=0; i<polje2D.length; i++) {
            for (j=0; j<polje2D[i].length; j++) {
                if (polje2D[i][j] == iscem) {
                    nasel = true;
                    break isci;
                }
            }
        }
    }
}
```

```
        if (nasel) {
            System.out.println("Našel " + iscem +
                " na mestu " + i + ", " + j);
        } else {
            System.out.println(iscem +
                " ni v polju");
        }
    }
}
```

Skoki – “`continue`”

- Poznamo dve vrsti `continue` stavka: neoznačeno in označeno
- Služi predčasnim izhodom iz `for`, `while` ali `do...while` zank
 - neoznačena in označena različica delujeta podobno kot sorodni različici `break` stavka
 - `continue` “preskoči” vse ukaze do konca zanke (velja za trenutni iteracijo zanke)



Skoki – “continue” – primer (1)

■ Neoznačeni `continue` (pojavitve znaka v nizu):

```
class ContinueDemo {
    public static void main(String[] args) {
        String isciMe = "peter pipec je pobral polno pest pisanih peres";
        int max = isciMe.length();
        int stPjev = 0;
        for (int i = 0; i < max; i++) {
            // zanimajo nas le p-ji
            if (isciMe.charAt(i) != 'p')
                continue;
            // naletali na p
            stPjev++;
        }
        System.out.println("Našel " + stPjev + " p-jev v nizu.");
    }
}
```



Skoki – “continue” – primer (2)

■ Označeni `continue` (iskanje podniza v nizu):

```
class ContinueDemo2 {  
  
    public static void main(String[] args) {  
  
        String isciMe = "Najdi podniz v meni."  
        String podniz = "pod";  
        boolean nasel = false;  
        int max = isciMe.length() -  
                    podniz.length();  
  
        test:  
        for (int i = 0; i <= max; i++) {  
            int n = podniz.length();  
            int j = i;  
            int k = 0;  
            while (n-- != 0) {  
                if (isciMe.charAt(j++) !=  
                    podniz.charAt(k++)) {  
                    continue test;  
                }  
            }  
            nasel = true;  
            break test;  
        }  
        System.out.println(nasel ? "našel" :  
                            "nisem našel");  
    }  
}
```

Skoki – “goto”



- Ukaz `goto` “takoj skoči” na označeno mesto
 - Včasih mu pravimo *brezpogojni skok*
 - JAVA ga ne pozna

■ Primer:

```
void prikazi(int matrika[3][3]) {
    int i,j;
    for (i = 0; i < 3; i++)
        for (j = 0; j < 3; j++) {
            if ( (matrika[i][j] < 1) || (matrika[i][j] > 6) )
                goto izven_mej;
            printf("matrika[%d][%d] = %d\n",i,j,matrika[i][j]);
        }
    return;
    izven_mej:
        printf("število mora biti med 1 in 6\n");
}
```

(NE!)uporaba skokov

- Uporaba skokov v programih prinaša s seboj številne potencialne “nevarnosti”
 - več o tem (v angleščini) na:
http://www.roseindia.net/javatutorials/java_break_to_label_tatement.shtml
<http://en.wikipedia.org/wiki/GOTO>
- Uporaba “skočnih” ukazov zato velja za slabo programersko prakso in se odsvetuje
 - namesto “skočnih” ukazov lahko (skoraj) vedno uporabimo vejitve ali zanke

Skoki – “return”

- Ukaz `return` povzroči izhod iz trenutne funkcije/metode; program nadaljuje z izvajanjem tam, od koder je bila funkcija/metoda klicana
- Ukaz `return` lahko *vrne vrednost* ali pa *ne*

`return ++stej;`

`return;`

Funkcije (1)

■ Kaj je funkcija?

- definicija funkcije v matematiki

([http://en.wikipedia.org/wiki/Function_\(mathematics\)](http://en.wikipedia.org/wiki/Function_(mathematics)))

- kaj pa v računalništvu?

Definicija (2):

Funkcije (in blok kode) so tisti **deli programa** (in blok kode), ki so **lahko sprejmejo parametre** in **vratajše vrednosti**.
Programi: lažje za vzdrževanje

Functions (also known as **usable code**) that can be passed **parameters** and can **return a value** – which **can be called** from another part of the program.
Generally, functions greatly enhance the space-efficiency and maintainability of computer programs.

Vir: http://www.123flashchat.com/flash/12_understanding9.html

Vir: <http://www.true-type-typography.com/ttglossf.htm>

Funkcije (2)

- *Kaj so* torej funkcije v računalništvu?
 1. So bloki kode / zaključeni deli programa;
 2. Lahko sprejemajo (parametre) in vračajo vrednosti;
 3. Lahko jih kličemo iz drugih delov programa.
- *Razlogi za uporabo* funkcij?
 - Pisanje večjih programov postane lažje (če jih razbijemo na manjše enote)
 - Vzdrževanje (in razhroščevanje) programov je lažje
 - Lažje sodelovanje pri pisanju programov

Vir: <http://www.cs.utah.edu/~hamlet/release/lessons/fortran08/fortran08/node4.shtml>

Primer funkcije



■ Splošna sintaksa

```
[tip rezultata] <ime funkcije>(parametri) {
    <deklaracija spremenljivk>;
    <stavki>;
    [return (<vrednost>) ;]
}
```

Podpis funkcije (pointing to the signature line)

Telo funkcije (pointing to the body lines)

Vračanje (vrednosti) (pointing to the return statement)

■ Funkcija v C-ju – primer:

```
double power(double val, int pow) {
    double ret_val = 1.0;
    int i;
    for(i = 0; i < pow; i++)
        ret_val *= val;
    return(ret_val);
}
```

Klic funkcije

result = power(val, pow);



Funkcije v JAVI = *metode*

■ Sintaksa

```
[prilastki] <tip_rezultata> <ime_metode>(parametri) [throws] {  
  <deklaracija_spremenljivk>;  
  <stavki>;  
  [return <vrednost>;]  
}
```

■ Primer:

```
public static int izracunajPloscino(int sirina, int visina) {  
  int ploscina; // To je lokalna spremenljivka  
  ploscina = sirina * visina;  
  return ploscina;  
}
```

`plscn = izracunajPloscino(4,5);`

Kaj je program?

- Se še spomnite iz uvodnega predavanja?
- (RAČUNAL.) PROGRAM
 - (Računalniški) program je zbirka ukazov, ki opisujejo neko nalogo ali množico nalog, katere naj se izvajajo na določenem računalniku.
- COMPUTER PROGRAM
 - A computer program is a collection of instructions that describes a task, or set of tasks, to be carried out by a computer.

Vir: http://en.wikipedia.org/wiki/Computer_program

(program == funkcija) = 1 ?

- Kaj že je funkcija?
 - blok (= zaključen del) kode; ✓
 - lahko sprejema (parametre) in vrača vrednosti; ✓
 - lahko jo kličemo. ✓
- Kaj pa **program**?
- Kaj že je funkcija?
 - Mar ni (glavni) program (C/C++, JAVA, ...) le `main()` funkcija?

Povzetek

Viri

- http://en.wikipedia.org/wiki/Computer_program
- [http://en.wikipedia.org/wiki/Function_\(mathematics\)](http://en.wikipedia.org/wiki/Function_(mathematics))
- <http://en.wikipedia.org/wiki/GOTO>
- <http://java.sun.com/docs/books/jls/>
- <http://java.sun.com/docs/books/tutorial/java/nutsandbolts/>
- http://www.123flashchat.com/flash/12_understanding9.html
- <http://www.cs.utah.edu/~hamlet/release/lessons/fortran08/fortran08/node4.shtml>
- <http://www.roseindia.net/javatutorials/>
- <http://www.true-type-typography.com/ttglossf.htm>

Naloge