INTERPRETCIJA PROGRAMA
PROJEKTNI ZADATAK

Uvod u programiranje (NL)

Grupa G
Matea Čotić
Manuela Pleša
Katarina Šupe

Sadržaj

1 Uvod

U ovoj skripti upoznat ćemo se s osnovnim pojmovima u programiranju. Skripta je popraćena primjerima u NL (*new language*) jeziku uz pomoć kojeg ćemo naučiti kako napisati neke jednostavne programe.

2 Osnove programskog jezika NL

2.1 Tipovi podataka i pridruživanja

Zašto je u programima korisno da neka varijabla ima svoje ime? Recimo da želimo u programu koristiti riječ `"programiranje"` na više različitih načina, u raznim operacijama i/ili u provjeravanju raznih uvjeta. Korisno je tada tu riječ *negdje* pospremiti. Da bismo je pospremili, moramo joj dati **ime**. U NL-u ime nužno započinje slovom, nakon čega slijedi niz slova i/ili znamenki. Ime koje služi za pozivanje podataka koji su spremljeni u memoriji naziva se **varijabla**. **Vrijednost** varijable x u primjeru (1) je `"programiranje"`. Varijablu zamislite kao neku kutiju u koju pospremamo neke podatke koje ćemo iz kutije po potrebi vaditi. Stoga, jedna varijabla pokazuje uvijek na istu kutiju, ali na različite vrijednosti, ovisno o tome što je u kutiji. Ono što je u kutiji možemo mijenjati pomoću **jednostavnog pridruživanja** (operator $=$).

```
x = "programiranje";
```
(1)

Sada se možemo pitati kojeg je *tipa* naša varijabla. **Tip podatka** predstavlja skup vrijednosti koje varijabla može poprimiti. Osnovni tipovi podataka u NL-u su cijeli brojevi (`int`) te znakovni nizovi (`string`). Kako je u primjeru (1) varijabla x neki znakovni niz, onda je očito tipa `string`. Primijetimo da je stringovnu vrijednost ispravno zapisivati u navodnicima.

Osim jednostavnog, postoje i proširena pridruživanja, od kojih možemo spomenuti $-=$ i $+=$ koji su podržani u NL-jeziku. Takva proširenja pridruživanja ujedno su skraćeni operatori zbrajanja i oduzimanja. U primjeru niže možemo vidjeti kako koristimo $+=$, a analogno ćemo koristiti $-=$.

```
x = 10;
x += 10;
cout << x;

Output:
20
```

Sada je jasno da $x \mathrel{+}= 10$ predstavlja zapravo pridruživanje $x = x + 10$, gdje x pridružujemo njegovu vrijednost uvećanu za 10.

Također, varijablama možemo pridružiti vrijednost uvećanu ili umanjenu za 1, tako da ispred ili iza imena varijable dodamo `++` ili `--`. Takav operator naziva se pred i post inkrement, odnosno dekrement. U nekim programskim jezicima postoji razlika u korištenju prefiksnog i postfiksnog operatora, ali u NL-u je to isto.

Neka je `i` neka varijabla kojoj smo prethodno pridružili neku vrijednost. Korištenje `i++` ili `i--` isto je kao `i = i+1` ili `i = i-1`, tj. skraćeno `i += 1` ili `i -= 1`. Inkrement i dekrement će nam biti posebice korisni u petljama, o kojima ćemo više reći u poglavlju 4.2.

Budete li ste upoznavali s nekim višim programskim jezicima, kao što su C i C++, vidjet ćete još neke tipove kao što su `float` za decimalne brojeve, `char` za jedan znak, `bool` za logički tip te operatore $/=$, $*=$ i slično.

Napomena 1. Program je napisan u NL-u, stoga je kraj svake narebe označen točkom sa zarezom.

Zadatak 1. Pokrenite interpeter za NL te pridružite varijabli imena `jasambroj` cjelobrojnu vrijednost 5.

2.2 Ulazno izlazne funkcije

Ukoliko ste riješili zadatak 1, sigurno se pitate što sada sve možete sa svojom novom varijablom. Da bismo se uvjerili da je ta varijabla zaista negdje pospremljena, možemo je ispisati. U jeziku NL funkcija `cout` popraćena operatorom ispisa `<<`, iza čega slijedi ime varijable ili neka vrijednost, služi nam za ispis neke poruke ili vrijednosti varijable na zaslon monitora. Nakon što ste je inicijalizirali, možete pokušati ispisati vrijednost vaše varijable iz zadatka 1. Sada ćemo ispisati vrijednost varijable `x` iz primjera (1).

```
x = "programiranje";
cout << x;

Output:
programiranje
```

Recimo sad da želimo ispisati još neku vrijednost, ali u idućem retku. Tada ćemo koristiti naredbu `endl` (*end of line*) koja govori da idući ispis funkcijom `cout` ide u idući redak. Pogledajmo najprije kako bi bilo da nismo koristili naredbu `endl`:

```
x = "programiranje1";
y = "programiranje2";
cout << x << y;

Output:
programiranje1 programiranje2
```

Dvije riječi su nam se ispisale jedna za drugom u istom retku, s razmakom (tako je definirana `cout` funkcija), a htjeli smo da svaka bude u svom retku. Pokušajmo stoga iskoristiti naredbu `endl`:

```
x = "programiranje1";
y = "programiranje2";
cout << x << endl;
cout << y;

Output:
programiranje1
programiranje2
```

Sada smo dobili ono što smo htjeli. Naredbom `endl` stoga dobivamo na preglednosti našeg ispisa.
Osim što varijabli možemo pridružiti vrijednost direktno, pišući kod, možemo tražiti od korisnika da unese neku vrijednost, koju će program pospremiti u neku varijablu. Za to nam služi funkcija `cin` nakon koje slijedi operator unosa `>>` te ime varijable u koju će upisana vrijednost biti pospremljena. Pogledajmo na primjeru kako to izgleda:

```
cout << "Unesite neki broj: " << endl;
cin >> x;
x += 10;
cout << x << endl;
```

Ukoliko testirate ovaj program u NL interpreteru, zatražit će vas da unesete neki broj. Recimo da smo unijeli broj 7. Nakon toga funkcija `cin` usmjerava podatke s tipkovnice na ulazni tok te posprema vrijednost varijable x, tj. u *kutiju* varijable imena x stavlja broj 7. Zatim vrijednost u kutiji povećamo za 10 pa je sada vrijednost na koju varijabla pokazuje jednaka 17. Nakon toga ispisujemo vrijednost varijable x te prelazimo u novi red. Dakle, za unos 7, ispis ovog programa je 17.

2.3 Komentari

Kada pišete veći kod koji kasnije namjeravate pogledati ili ga šaljete nekom na pregled, korisno je imati komentare koje su smjernice čitaocu za bolje razumijevanje koda. Vjerovali ili ne, sami ćete često zaboraviti što ste točno mislili određenom linijom koda, stoga se uvijek potrudite da vam kod bude što čitljiviji uz dodatak komentara na kompliciranijim dijelovima. Komentare dijelimo na **linijske** i **višelinijske**. Linijski komentari su kratki komentari koji ispunjavaju maksimalno jedan redak te započinju s dvije kose crte `//`. Višelinijske komentare možete koristiti kada vam se komentar

proteže kroz nekoliko redaka, većinom pri objašnjavanju što neka funkcija zapravo radi ili kada želite na početku napisati što je zadatak od vas tražio. Višelinijski komentari započinju sa `/*`, a završavaju sa `*/`. Na primjeru niže možete vidjeti primjenu:

```
/*Ovo je program koji prima unos nekog broja od strane
korisnika te uvecava uneseni broj za 10 */
cout << "Unesite neki broj: " << endl;
cin >> x;
x += 10; //uvecaj za 10
cout << x << endl;
```

Koliko god su korisni, s komentarima možete i pretjerati. Bitno je stavljati ih na ona mjesta gdje će njihov sadržaj uistinu biti jasan, razumljiv i od pomoći. Inače samo postoji hrpa viška teksta u kodu, što zasigurno ne doprinosi preglednosti.

3 Operacije

3.1 Brojevne operacije

Brojevnim operacijama smatramo četiri osnovne operacije (zbrajanje, oduzimanje, množenje i dijeljenje), usporedbe (`<`, `>`, `<=`, `>=`, `==`, `!=`) koje vraćaju broj 1 ili 0, ovisno o tom je li usporedba istina ili laž, redom.

```
x = 10;
y = 5;
z = x + y;
cout << z << endl;

Output:
15
```

U ovom primjeru smo varijabli imena z pridružili vrijednost zbroja vrijednosti varijabli x i y, nakon čega smo je ispisali. U idućem primjeru varijabla z će također poprimiti vrijednost 15, ali na malo drukčiji način:

```
x = 10;
z = x + 5;
cout << z << endl;

Output:
15
```

Sada je jasno da smo isto tako mogli napisati `z = 10 + 5`. Analogno bismo koristili preostale tri osnovne operacije.
Usporedbe ćemo najviše koristiti u uvjetima grananja i petljama pa ćemo detaljnije o njima kasnije. Tamo ćemo još koristiti i operator `!` (negacija) koji istinu (1) pretvara u laž (0) i obratno.

Ponekad će nam biti korisno pretvoriti neki brojevni tip u stringovni. Vratimo se na primjer (1) i recimo da varijabli x želimo nadodati broj 1. O takvom *zbrajanju* stringova ćemo kasnije, za početak će nam biti korisno broj 1 pretvoriti u `string`. Za to nam služi funkcija `toStr(ime, vrijednost)` koja će varijabli imena `ime` tipa `string` pridružiti vrijednost `vrijednost`.

```
toStr(y, 1);
cout << y << endl;

Output:
1
```

Premda je ispis jednak kao da je varijabla `y` još uvijek broj, njezina vrijednost je ipak tipa `string`, tj. `"1"`. NL nam ispisuje sadržaj stringa bez dvostrukih navodnika koji ga omeđuju. Da je uistinu `string`, uvjerit ćemo se u sljedećem poglavlju.

3.2 Stringovne operacije

Sada možemo nastaviti prethodni primjer te na njemu pokazati stringovnu operaciju konkatenaciju koja će vrijednost dva stringa spojiti u jednu varijablu tipa `string`.

```
x = "programiranje";
toStr(y, 1);
z = x + y;
cout << z << endl;

Output:
programiranje1
```

Da varijablu `y` nismo pretvorili u tip `string`, ovakva operacija (konkatenacija) ne bi bila moguća. Kada je slijeva `string`, tada NL i zdesna očekiva `string` koji će konkatenirati s lijevim. Ukoliko se zdesna nalazi brojevni tip, interpreter će vam javiti semantičku grešku. Analogno je za zbrajanje broja sa stringom.
Osim konkateniranja, NL jezik podržava još i test jednakosti koji vraća broj (1 za istinu, 0 za laž) te pretvaranje u broj. Test jednakosti ćemo koristiti u uvjetima grananja, a sada pokažimo kako bismo varijablu tipa `string` pretvorili u varijablu tipa `int`.

Kao što smo brojeve pretvarali u stringove, tako bismo htjeli stringove pretvarati u brojeve. Međutim, jasno je da string `"programiranje"` nećemo moći pretvoriti u brojevni tip. Stoga, ukoliko je string takav da sadrži isključivo znamenke, onda ga možemo pretvoriti u broj koristeći se funkcijom `toInt(ime, vrijednost)`.

```
x = 10;
toInt(y, "5");
z = x + y;
cout << z << endl;

Output:
15
```

Vidimo da je ovo korisno ukoliko želimo koristiti brojevne operacije između varijabli brojevnog i stringovnog tipa, ukoliko stringovni tip sadrži isključivo znamenke.

4 Upravljanje tokom programa

4.1 Grananja

Grananje je programska struktura koja ovisno o rezultatu nekog postavljenog uvjeta omogućuje različiti tijek programa. U *NL*-jeziku postoje `if` te `if else` grananja. Doslovno prevedno, to znači da **ako** se ispuni neki uvjet naveden kraj naredbe `if`, onda program nastavlja izvoditi blok naredbi omeđen vitičastim zagradama. **Inače**, ukoliko taj isti uvjet nije zadovoljen, tada može postojati `else` naredba, čiji se blok izvršava u svim drugim slučajevima osim u onom kojeg je pokrio `if` dio. Dakle, jasno je da će se u programu izvršiti ILI `if` ILI `else` dio, tj. uvjeti te dvije naredbe su disjunktni. Ukoliko blok naredbi ima samo jednu naredbu, tada se vitičaste zagrade mogu izostaviti, kako *NL* zna da iza `if` i `else` slijedi neka naredba koja završava točkom sa zarezom. Blokovi naredbi se obično pišu uvučeno zbog preglednosti.

4.1.1 If grananje

Algoritam 1 If grananje s jednom naredbom

```
1: ako (uvjet) onda
2:     naredba;
3: kraj ako
```

Algoritam 2 If grananje s blokom naredbi

1: **ako** (uvjet) **onda**
2: blok naredbi u kojem svaka naredba završava s ";" omeđen s vitičastim zagradama
3: **kraj ako**

Uvjet koji se nalazi u zagradama nakon naredbe `if` je neka stringovna ili brojevna operacija koja vraća 0 ili 1 tj. laž ili istinu. Ukoliko je uvjet istinit, tj. jednak 1, tada kažemo da je on ispunjen. Inače, uvjet nije ispunjen. Niže možemo vidjeti neke primjere u NL-u koje možete i sami isprobati.

```
x = 10;
y = 20;
if(x < y)
    x += 1;
if(x >= y)
{
    x += 1;
    y += 2;
}
cout << x << endl;
cout << y << endl;

Output:
11
20
```

Dakle, `x` je očito strogo manji od `y` pa je uvjet prvog `if` grananja ispunjen. Stoga se izvršava naredba `x += 1;` koja uvećava vrijednost varijable `x` za 1. Uvjet drugog `if` grananja očito nije ispunjen pa blok naredbi koji slijedi program neće izvršiti. Kada na kraju ispišemo vrijednosti varijabli `x` i `y`, vidimo da se vrijednost varijable `x` usitinu povećala za 1, dok je vrijednost varijable `y` ostala nepromijenjena.

4.1.2 If else grananje

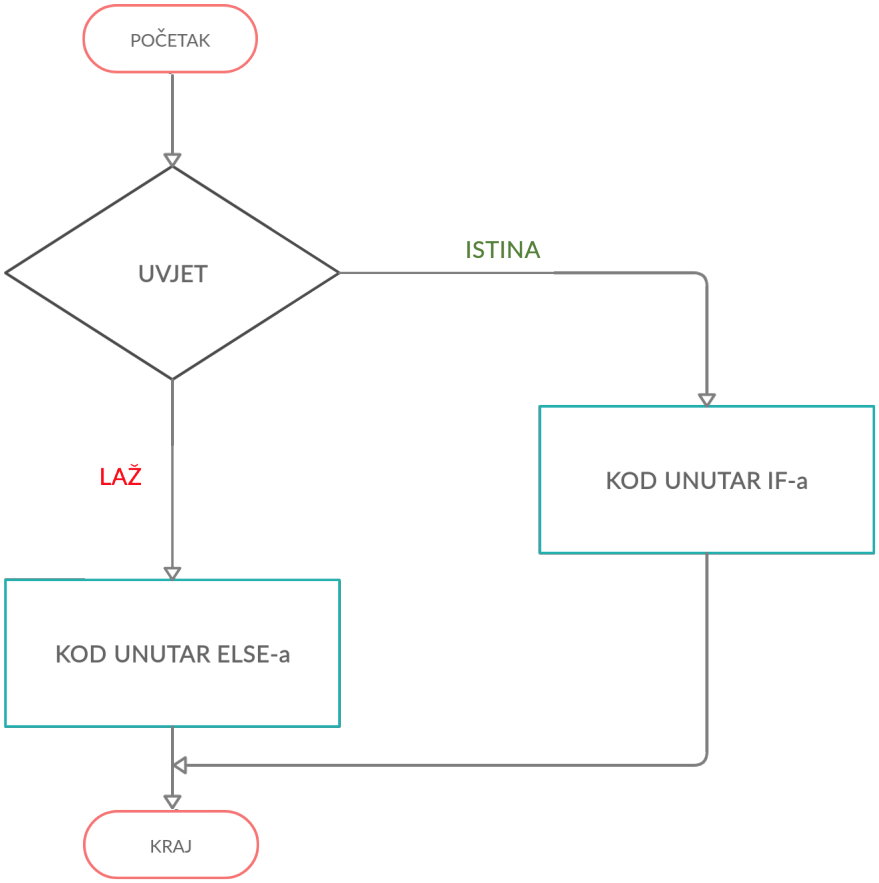
Pogledajmo najprije kako izgleda `if else` grananje u pseudokodu:

Algoritam 3 If else grananje s jednom naredbom

1: **ako** (uvjet) **onda**
2: naredba;
3: **inače**
4: naredba;
5: **kraj ako**

S blokom naredbi bismo radili kao i u `if` grananju. Na dijagramu uočite koji je zapravo tok izvođenja programa.

Slika 1: Dijagram toka `if else` grananja



Pogledajmo sada kako bi to izgledalo u NL-u:

```
x = 10;
y = 20;
if(x < y)
    x += 1;
else
{
    x += 1;
    y += 2;
```

```
}
cout << x << endl;
cout << y << endl;

Output:
11
20
```

Situacija je ista kao kod primjera u `if` grananju, ali umjesto da smo navodili neki drugi uvjet, jednostavno smo dodali `else` koji je obuhvatio sve slučajeve osim onog koji je uvjet kod `if`-a.

Zadatak 2. Promijenite uvjet kod `if`-a u prethodnom primjeru tako da ispis bude

11

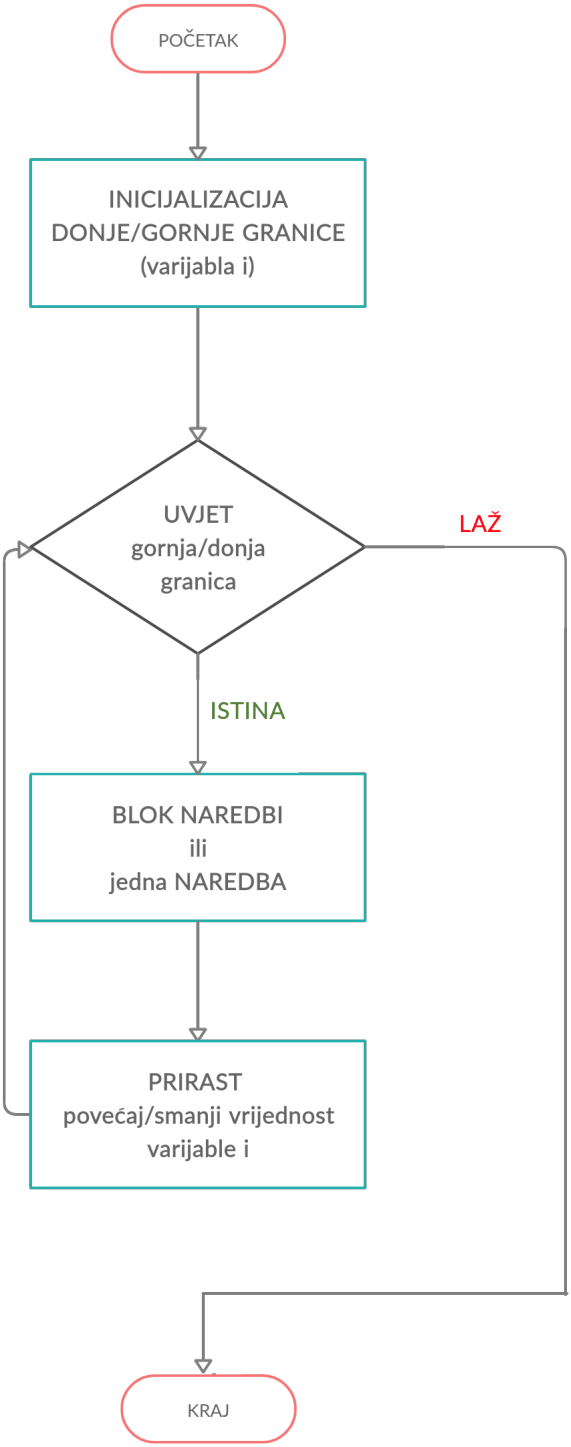
22

4.2 Petlje

4.2.1 For petlja

`For` petlju koristimo kada neki dio programa želimo ponoviti unaprijed poznati broj puta. Sintaksa `for` petlje su brojevni izrazi u obloj zagradi odvojeni točkom sa zarezom. U oblim zagradama imamo početno stanje (donju/gornju granicu), iza kojeg slijedi uvjet, tj. gornja/donja granica. Uvjet je usporedba dviju varijabli brojevnog ili stringovnog tipa (koja je istinita ili lažna). Nakon toga imamo neki prirast, koji je najčešće u obliku predinkrementa, postinkrementa ili operacija `+=` i `-=`. Ukoliko je u prvom brojevnom izrazu (početnom stanju) korištena jedna varijabla, važno je da u preostala dva uvjeta imamo tu istu varijablu. Nakon uvjeta slijedi naredba ili blok naredbi omeđen vitičastim zagradama. Najprije pogledajmo dijagram toka:

Slika 2: Dijagram toka `for` petlje



Zatim ćemo bolje razumijeti primjer u NL-u.

```
x = 10;
for(i = 0; i < 10; i++)
    x++;
cout << x << endl;

Output:
20
```

U ovom primjeru varijablu `i` postavljamo na 0, tj. donju granicu postavljamo na 0. Gornja granica je usporedbom s brojem 10 (gdje je operator usporedbe `<`) postavljena na 9. Vidimo da prirast određen postinkrementom, stoga se vrijednost varijable `i`, svaki prolazom kroz petlju, povećava za 1. Dakle, ova petlja se ponavlja 10 puta (za `i` od 0 do 9). U tijelu petlje nalazi se samo jedna naredba koja povećava vrijednost varijable `x` za 1 (kao da piše `x = x+1;`). Kako je početna vrijednost varijable `x` bila 10, te smo je 10 puta povećali za 1, onda je izlazna vrijednost 20.

Zadatak 3. Promijenite uvjete i tijelo `for` petlje tako da se petlja vrti 5 puta te da na kraju petlje vrijednost varijable `x` bude također 20. (*Hint.* sjetite se operatora += koji možemo koristit i u prirastu.)

4.2.2 While petlja

Iza `while` petlje nalazi se pripadni uvjet u oblim zagradama, slično kao kod `if` grananja. Ukoliko je uvjet ispunjen, tj. istinit, onda se izvršava tijelo petlje (naredba ili blok naredbi omeđen vitičastim zagradama). Inače se tijelo petlje preskače te se proceduralno nastavlja izvršavati kod ispod petlje, ako postoji. Tijelo petlje ponavlja se sve dok je dani uvjet ispunjen, a on se može promijeniti u tijelu petlje.

```
x = 1;
y = 2;
while(x < 10)
{
    y += 2;
    x++;
}
cout << x << endl;
cout << y << endl;

Output:
10
20
```

U gornjem primjeru petlja se vrtila 9 puta, a u zadnjem prolazu se vrijednost varijable `x` povećala na 10 pa uvjet `while` petlje više nije bio zadovoljen, što je rezultiralo izlaskom iz petlje. Dakle, vrijednost varijable `y` smo uvećali za 2 devet puta, a kako joj je početna vrijednost bila 2, to znači da joj je nakon petlje vrijednost narasla na 20.

Ukoliko se uvjet koji je ispunjen ne mijenja u tijelu petlje ili nikad ne postane lažan, tada ćemo upasti u *beskonačnu petlju*, što nikako nije poželjno. Stoga u tijelu `while` petlje često povećavamo/smanjivamo neku varijablu koja se nalazi u uvjetu, kako bi, nakon konačnog broja ponavljanja, petlja uistinu stala. Česta je praksa koristiti naredbu `break` za izlazak iz petlje. Tada unutar `while` petlje imamo neko `if` grananje. Ukoliko je uvjet `if`-a zadovoljen, onda pozovemo naredbu `break` čime izlazimo iz `while` petlje.

```
x = 5;
y = 2;
while(1 == 1)
{
    y++;
    if(x == y)
        break;
}
cout << y;

Output:
5
```

Kako je uvjet `while` petlje uvijek istinit (1 = 1), da nismo dodali `if` grananje i naredbu `break` upali bismo u beskonačnu petlju. Što se ovdje dogodilo? Svakim prolaskom petlje smo uvećavali vrijednost varijable `y` te smo nakon toga u uvjetu `if` grananja provjeravali je li `x` jednak `y`. Kako je početna vrijednost varijable `y` jednaka 2, a varijable `x` jednaka 5, u trećem prolasku `while` petlje uvjet `if` grananja će biti ispunjen, nakon čega će biti pozvana naredba `break`, kojom ćemo izaći iz petlje. Vrijednost varijable `x` ostaje nepromijenjena, a vrijednost varijable `y` je postala jednaka vrijednosti varijable `x`, tj. 5.

4.2.3 Do while petlja

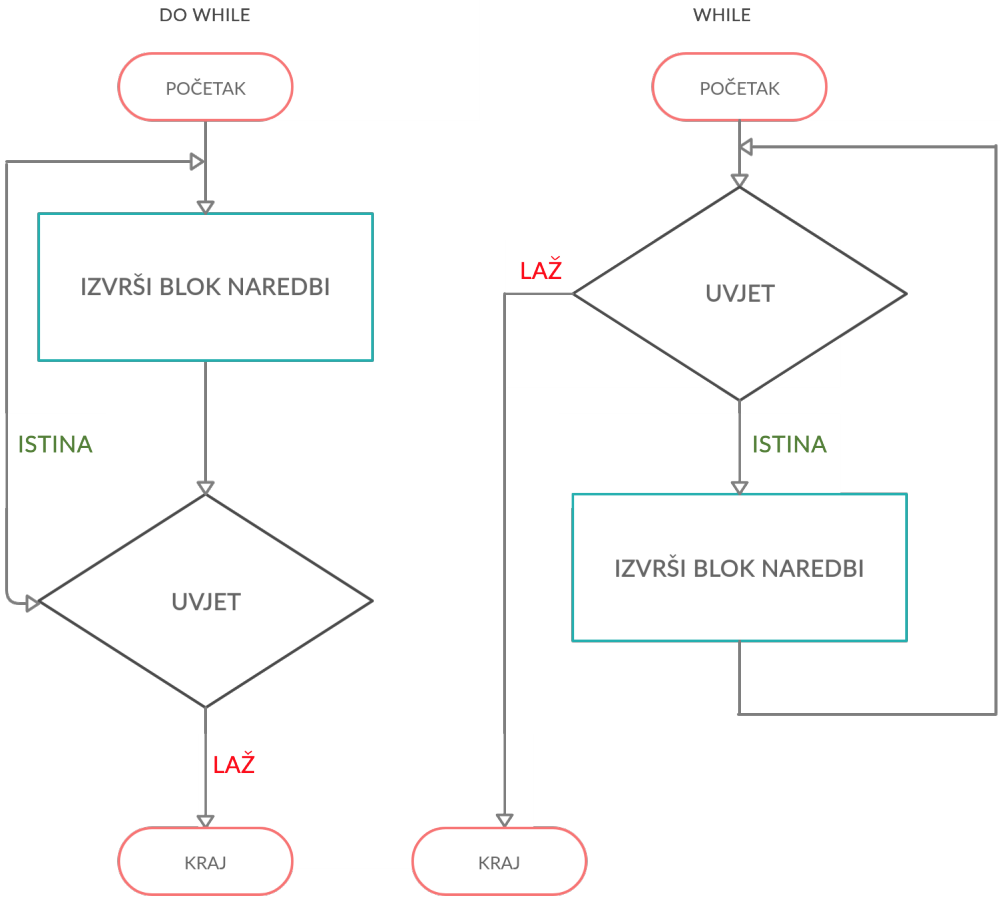
`Do while` petlja započinje naredbom `do` nakon koje slijedi tijelo petlje omeđeno vitičastim zagradama. Programski kod unutar tijela petlje se u prvom prolazu bezuvjetno izvršava. Nakon tijela petlje slijedi `while` iza kojeg slijedi uvjet u oblim zagradama te točka sa zarezom. Tijelo petlje će se ponavljati sve dok je uvjet ispunjen. Dakle, `do while` petlja je jako slična `while` petlji, uz iznimku prvog bezuvjetnog prolaza. Korisno je kada prvo želimo neki dio koda barem jednom izvršiti, a tek onda provjeravati neki uvjet koji bi potencijalno mogao zaustaviti našu petlju.

```
cout << "Unesi broj:";
cin >> x;
max = x; //pretpostavimo da je prvi najveci
do{
     cout << "Unesi broj: ";
     cin >> x;
     if(x > max)
          max = x;
}while(x > 0);

cout << "Najveci broj je: " << max << endl;
```

U gornjem primjeru nalazimo najveći uneseni broj koji je ujedno prirodan. Na početku pretpostavimo da je prvi uneseni broj najveći te zatim provjeravamo je li uneseni broj veći od najvećeg. Ukoliko jest, mijenjamo vrijednost varijable `max`. Nakon toga tražimo od korisnika da unese novi broj, za kojeg iznova provjeravamo je li najveći dosad, samo ukoliko je prirodan (`while(x > 0)`).
Za bolje razumijevanje razlike `do` i `do while` petlje, proučite dijagram 3.

Slika 3: Dijagram toka



5 Zaključak

Nadamo se da ste u ovoj skripti naučili barem neke osnove programiranja u NL-u. Premda je veoma sličan drugim poznatim programskim jezicima, ipak je bitno drukčiji, kako su u njega uloženi brojni sati našeg truda u želji da vama, našim učenicima, približimo programiranje i učinimo ga zabavnim.