Upute za 2. laboratorijsku vježbu

2. Uvod

U ovoj vježbi nastavljamo sa učenjem jezika Cool. U posljednjem zadatku prošle vježbe pokrenuli smo kod koji sa tipkovnice učitava jedan cijeli broj, povećava ga za jedan, te ga ispisuje na ekran. U tom kodu smo promijenili način unosa broja – prije bi unosili string, pa vršili konverziju, dok ovdje izravno unosimo cijeli broj pomoću naredbe **`in_int()`**:

```
class Main inherits IO {
	i : Int <- new Int;
	main():Object {{
		i <- in_int();
		i <- i + 1;
		out_int(i);
		out_string("\n");
	}};
};
```

2.1 Izrada nove metode klase

Povećanje broja za jedan ćemo prebaciti u funkciju. Funkciju ćemo definirati kao metodu klase Main.

```
class Main inherits IO {
	i : Int <- new Int;
	main():Object {{
		i <- in_int();
		i <- fact(i);
		out_int(i);
		out_string("\n");
	}};
	fact(i:Int):Int { i+1 };
};
```

Dodali smo funkciju **`fact()`**. Ona prima jedan cijeli broj i vraća također cijeli broj, uvećan za jedan. Ovu funkciju zovemo iz funkcije **`main()`**.

Sada ćemo napisati funkciju koja računa faktorijel cijeloga broja. Nju umetnite u klasu **`Main`** umjesto postojeće.

```
	fact(i:Int):Int {
		if(i=0) then
			1
		else
			i*fact(i-1)
		fi
	};
```

Funkciju za faktorijele smo napisali rekurzivno. Koristili smo **`if then else`** grananje, a kao granični uvjet provjeru je li argument jednak 1. U tom slučaju vratili smo 1, a inače ulazimo u rekurzivni poziv. U jeziku COOL **`if`** grananje s završava sa **`fi`**.

Napišimo funkciju za faktorijel iterativno:

```
fact(i:Int):Int {
      let f:Int <- 1 in {
            while(not(i=0)) loop {
                  f <- f * i;
                  i <- i - 1;
            } pool;
            f;
      }
};
```

U ovom slučaju koristimo petlju **`while loop pool`**. Kao brojač koristimo argument **`i`**. Uvjet petlje je da brojač nije jednak nuli – koristimo relacijski predikat **`not`**. Za proračun će nam trebati i lokalna varijabla u koju ćemo spremati umnožak. Lokalnu varijablu **`f`** deklarirati ćemo pomoću naredbe **`let`**. U jeziku COOL lokalne varijable smiju biti istog imena kao i funkcije. Naredba **`let`** nakon definicije imena varijable obavezno treba imati i označavanje tipa varijable, te inicijalizaciju, odnosno dodjelu vrijednosti varijabli. Iza toga ide ključna riječ **`in`** pa blok unutar kojega se može koristiti nova lokalna varijabla. Taj blok predstavlja doseg i vrijeme trajanja lokalne varijable.

Važno je primijetiti da se za dodjelu vrijednosti u jeziku COOL koristi operator **`<-`**

Ako umjesto toga greškom stavimo operator **`=`** program će se uredno prevesti, ali će pri pokretanju uđe u beskonačnu petlju, te ćemo nekoliko puta dobiti poruku **`Increasing heap...`** i nakon toga će program prekinuti rad.

2.2 Izrada nove klase

Pokazati ćemo kako se u jeziku COOL izrađuje nova klasa na primjeru izrade samo-referentne strukture podataka – liste. Najprije ispišimo **`Hello World!`** na malo drugačiji način.

```
class Main inherits IO {
      main():Object {
            let   hello: String <- "Hello ",
                  world: String <- "World!",
                  newline: String <- "\n"
            in
                  out_string(hello.concat(
                        world.concat(newline)))
      };
};
```

Pomoću naredbe **`let`** smo definirali tri lokalne varijable. Jednom **`let`** naredbom može se definirati više lokalnih varijabli. Definicije se međusobno povezuju sa operatorom zareza. Iza ključne riječi **`in`** dolazi dio koda u kojem vrijede nove lokalne varijable. U njemu se pomoću funkcije **`concat()`** međusobno povezuju sve tri lokalne varijable (stringa) i rezultat ispisuje na ekran.

Napravimo sada klasu za samo-referentnu strukturu podataka – listu. Klasu **`List`** dodajte u kod ispred klase **`Main`**.

```
class List {
      item: String;
      next: List;
      init(i: String, n: List): List {
            {
                  item <- i;
                  next <- n;
                  self;
            }
      };
};
```

Ova klasa definira čvor liste stringova. Klasa ima dva atributa **`item`** i **`next`**: prvi atribut predstavlja sadržaj čvora, a drugi je veza na idući čvor. Klasa ima jednu metodu - **`init()`**, koja postavlja vrijednost čvora liste. Metoda **`init()`** vraća sam objekt, kako bi mogli rekurzivno nadovezivati čvorove liste. Objekt za kojeg je metoda pozvana se u klasi referira pomoću ključne riječi **`self`**.

Preradimo sada glavnu klasu i metodu, kako bi za ispis koristili listu. Staru verziju definicije let:

```
      let   hello: String <- "Hello ",
            world: String <- "World!",
            newline: String <- "\n"
      in
            out_string(hello.concat(
                  world.concat(newline)))
```

zamijenite sa novom:

```
      let   hello: String <- "Hello ",
            world: String <- "World!",
            newline: String <- "\n",
            nil: List,
            list: List <-
                  (new List).init(hello,
                   (new List).init(world,
                    (new List).init(newline, nil)))
      in
            out_string(hello.concat(
                  world.concat(newline)))
```

Promjena je u tome što smo kao lokalne varijable dodali još dva objekta: **`nil`** i **`list`**. Varijabla **`nil`** je ne inicijalizirani objekt, koji nam služi kao **`NULL`** pokazivač – naime u COOL jeziku ne postoji **`NULL`** pokazivač. Varijabla **`list`** je sama lista, koja je deklarirana, alocirana i inicijalizirana rekurzivno.

Ovdje treba istaknuti jedno svojstvo deklaracije lokalnih varijabli pomoću ključne riječi **`let`** – naime lokalnih varijabli može biti više, a deklariraju se odvojene zarezom. Nakon što navedemo lokalnu varijablu, nju možemo koristiti već za slijedeću lokalnu varijablu, iza zareza. Dakle njena deklaracija i alokacija se vrši odmah, ne čeka se da se uđe u blok naredbi iza ključne riječi **`in`**.

Listu ispisujemo pomoću funkcije **`flatten()`**. Potrebno je dodati njenu definiciju u klasu **`List`**:

```
flatten(): String {
     if( isvoid next ) then
          item
     else
          item.concat(next.flatten())
     fi
};
```

Funkcija **`flatten()`** radi rekurzivno. Njen granični uvjet je da **`next`** član nije alociran. Alokaciju objekta provjeravamo pomoću ključne riječi **`isvoid`**.

Da bi novu funkciju pozvali potrebno je zamijeniti stari ispis sa funkcijom **`concat()`** iz metode **`main()`** klase **`Main`**:

```
out_string(hello.concat(
     world.concat(newline)))
```

sa novim ispisom pomoću funkcije **`flatten()`** :

```
out_string(list.flatten())
```

2.3 Polimorfizam

Preraditi ćemo klasu **`List`** kako bi mogla sadržavati bilo koji objekt, a ne samo stringove. Našu listu možemo generalizirati promijenivši tip varijabli koje lista sadrži, iz **`String`** u **`Object`.**

Staru klasu **`List`** izbrišite, a umjesto nje ubacite novu:

```
class List inherits A2I {
     item: Object;
     next: List;
     init(i: Object, n: List): List {
          {
               item <- i;
               next <- n;
               self;
          }
     };
};
```

Potom je potrebno je napraviti novu funkciju **`flatten()`**. U klasu **`List`** dodajte kod:

```
flatten(): String {
      let string: String <-
            case item of
                  i: Int => i2a(i);
                  s: String => s;
                  o: Object => { abort(); ""; };
            esac
      in
            if( isvoid next ) then
                  string
            else
                  string.concat(next.flatten())
            fi
};
```

Ispis trebamo prilagoditi tipu objekta koji je pohranjen u varijabli **`item`**. Tip objekta ćemo doznati pomoću grananja **`case of`**. Ovo je posebna vrsta grananja, koja kao uvjet uzima ulaznu varijablu **`item`**, te je potom ovisno o njenom tipu vraća varijable **`i`** ili **`s`** ili **`o`**. Promijenjena varijabla će poprimiti sadržaj koji se nalazi sa desne strane operatora **`=>`** iz pojedine od grana **`case of`** . Ako je varijabla **`item`** tipa **`String`** zadržati će vrijednost koju već sadrži. Ako je tipa **`int`** izvršiti će se konverzija pomoću funkcije **`i2a()`**. Ukoliko je varijabla tipa **`Object`** program se prekida. Za to služi funkcija **`abort()`**. Iza poziva ove funkcije stavljamo prazan string, kako bi zadovoljili provjeru tipova.

U glavnoj funkciji glavne klase ćemo i listu jedan element liste koji nije string, kako bi isprobali polimorfizam. Staru funkciju **`main()`** izbrišite, a umjesto nje ubacite:

```
main():Object {
      let   hello: String <- "Hello ",
            world: String <- "World!",
            i: Int <- 42,
            newline: String <- "\n",
            nil: List,
            list: List <-
                  (new List).init(hello,
                  (new List).init(world,
                  (new List).init(i,
                  (new List).init(newline, nil))))
      in
            out_string(list.flatten())
};
```

U ovoj verziji liste iza dva string smo dodali smo broj 42. To našu listu čini polimorfnom.

2.4 Zadatak:

Za kraj druge laboratorijske vježbe trebate samostalno napraviti jedan zadatak. Izaberite jedan od slijedećih zadataka:

1. napišite funkciju koji računa potenciju broja b na p – napraviti iterativno i rekurzivno. Isprobati obje funkcije iz glavne funkcije glavne klase.
2. napraviti klasu za binarno stablo koje sadrži stringove, sa funkcijama za dodavanje čvora, te ispis sadržaja stabla u infix redoslijedu. Klasu isprobati u glavnoj funkciji glavne klase.
3. smislite sami jedan zadatak po uzoru na jedan od prethodna dva, te napišite tekst zadatka i njegovo rješenje