F# na .NET platformi

Seminarski rad u okviru kursa
Metodologija stručnog i naučnog rada
Matematički fakultet

Tijana Todorov, Tamara Garibović,
David Nedeljković, Mihajlo Vićentijević
tijana.todorov710@gmail.com, t.garibovic1995@gmail.com,
dnedeljkovic710@gmail.com, mihajlovicent@gmail.com

24. april 2019

Sažetak

Ovaj rad predstavlja specifičnosti i zanimljivosti o programskom jeziku F#. U njemu ćete pronaći podatke o njegovom istorijskom poreklu. Zatim, zašto je korisno naučiti ovaj jezik i kakve sve mogućnosti on pruža. Upoznaćete F# kroz funkcionalno, paralelno i asinhrono programiranje. Možete pročitati zašto je .NET platforma važna za ovaj jezik i kako je ona nastala. Saznaćete kako instalirati i pokrenuti ovaj jezik na Windows i Linux operativnim sistemima. Na *Fizz Buzz* problemu predstavljena je ukratko sintaksa jezika F#. Opisan je mehanizam mernih jedinica (eng. *Units of Measure*) koji se u ovom jeziku može koristiti.

Sadržaj

1 Uvod

F# je programski jezik koji danas ima veoma široku upotrebu. Pronašao je svoju primenu u raznim oblastima među kojima su i bioinformatika, finansijsko modelovanje, statistika, baze podataka i mnoge druge, što je objašnjeno u poglavlju 3. Nastao je velikim trudom svog autora, Don Sajma (eng. *Don Syme*), koji je želeo da preusmeri jezike iz familije strogo tipiziranih funkcionalnih jezika na .NET platformu. Ova platforma je razvijena u okviru kompanije *Microsoft* i danas ima veoma široku upotrebu. Zahvaljujući tome ovaj jezik je kompatibilan sa svim jezicima koje platforma .NET podržava, što pruža jako velike mogućnosti. Više o tome možete pronaći u poglavlju 2. Jezik je pretežno funkcionalan, ali podržava još mnogo paradigmi među kojima su i paralelna i asinhrona koje su objašnjene u poglavlju 6. Postoji širok izbor okruženja za rad koja se mogu koristi za kreiranje programa u ovom programskom jeziku i nešto više o njima možete pronaći u poglavlju 7.

Cilj nam je bio da izdvojimo neke posebne karakteristike jezika koje ga razlikuju od ostalih i da motivišemo čitaoca za izučavanje F#-a. Prikazali smo kroz zanimljive primere 9 jednostvanu sintaksu koju ovaj jezik poseduje, a koja je dovoljno moćna da reši i složene matematičke probleme.

2 Poreklo programskog jezika F#

U 2018. godini F# opisan je u dokumentaciji kao *funkcionalni programski jezik koji se pokreće na .NET platformi* [20], ali odakle je on potekao? Istorija programskog jezika F# datira još od 1970. godine pa sve do danas. U ranim 70-im godinama na Univerzitetu u Edinburgu Robin Milner sa još nekoliko svojih kolega je razvio jezik ML (*Meta-Language*) baziran na programskom jeziku LISP. Njegova osnovna namena bila je pragmatičnog karaktera. Osmišljen je da pomogne u dokazivanju LCF (*Logical computable functions*) [17] teorema. ML jezik je koren svih jezika koji pripadaju familiji strogo tipiziranih funkcionalnih jezika, kao npr: *Miranda, Haskell, Standard ML, Ocaml, EdinburghML, ReasonML, PureScript*, a među njima i F#. Sve do danas ključne ideje ove familije jezika ostale su osnova jezika F# koji se nadograđivao kasnije iz dana u dan.

Tokom 80-ih godina došlo je do velike ekspanzije u kompjuterskoj industriji. Kako su se brzo razvijali softverski alati tako su se takođe pojavljivali novi jezici i programske paradigme. U tom periodu i *Microsoft* je doživeo veliku ekspanziju kao kompanija koja je razvijala operativne sisteme i aplikacije. Međutim, u ovom periodu, a najviše u kasnim 80-im godinama pojavio se novi, objektno-orjentisani talas razmišljanja u programiranju koji je veoma uticao na razvoj softvera.

Početkom 90-ih godina, dok je *Microsoft* težio da održi monopol na tržištu i dok je u fokusu i dalje bila objektno-orjentisana paradigma, strogo tipizirani funkcionalni jezici bili su manje aktivni i okrenuti su ka razvitku na drugim poljima. Uglavnom su se primenjivali u pronalaženju bagova, održavanju sistema i verifikaciji hardvera i softvera. Primeri jezika koji su se koristili za izgradnju ovakvih sistema su: *Edinburgh ML, Standard ML, Ocaml, Caml Light, LISP*. Takođe, neki funkcionalni jezici razvijeni su u cilju istraživanja u okviru paralelnog programiranja, kao što su *Parallel ML* i vrsta paralelnog *Haskell*-a [20].

2.1 Zašto je nastao jezik F#?

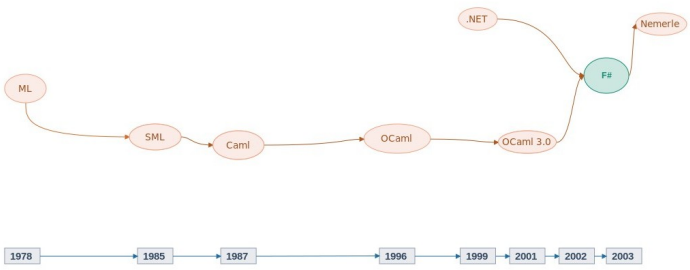
Tokom 90-ih godina *Microsoft* je razvio .NET [12, 20] platformu za razvoj softvera. Ideja je bila da se omogući međusobna kompatibilnost više programskih jezika, odnosno da svaki jezik podržan na ovoj platformi može da koristi kôd nekog drugog jezika platforme. Glavni cilj *Microsoft*-a bio je da se na ovoj platformi podrži što veći broj jezika iz različitih paradigmi.

U okviru ovoga Don Sajm (eng. *Don Syme*) je razvio projekat SML.NET koji je za ideju imao preusmeravanje *Standard ML*-a na .NET. Sistem je imao visok kvalitet, ali nije bio usvojen od strane programera. Početkom 2000-ih platforma .NET je već uveliko zaživela, ali za jezik SML.NET nije bilo većeg interesovanja. Autor je imao veliku želju da implementira strogo tipiziran funkcionalni jezik na način koji bi zainteresovao veliki broj programera. U razmatranje je ušao i jezik *OCaml*, ali je prethodno došlo do pokušaja implementacije *Haskell*-a za .NET. Ovaj pokušaj bio je samo delimično uspešan i primenjen na malim programima. Rad na daljoj implementaciji je zaustavljen. U decembru 2001. godine Don Sajm (eng. *Don Syme*) je u razmatranje vratio jezik *OCaml* i razvio projekat Caml.NET koji se kasnije preimenovao u F#.

Inicijalna ideja F# programskog jezika bila je jednostavna. Trebalo je da poveže prednosti *OCaml* programskog jezika sa prednostima .NET platforme. Godine 2002. pojavila se prva verzija F# 0.5 koja je bila slabo primećena. Don Sajm (eng. *Don Syme*) 2004. godine nastavlja intenzivno da razvija ovaj jezik, a početkom 2005. godine je izbacio prvu potpunu verziju F#-a [20]. Poslednja aktuelna verzija jezika je F# 4.1 [5].

2.2 Koji programski jezici su najviše uticali na razvoj jezika F#?

Najveći značaj za razvoj jezika F# imaju jezik SML (*Standard ML*) i CAML (*Categorical Abstarct Machine Language*) porodica jezika koja je razvijana na Nacionalnom institutu za istraživanje informatike i automatizacije u Francuskoj 1994. godine [7]. U okviru ovog projekta pod nazivom *"Cristal project"* razvija se jezik *Objective Caml* koji ima veoma visoke performanse i naučnici ga koriste na Linux i MAC OS X platformama. Na slici 1 može se videti deo razvojnog stabla koji prikazuje uticaj svih pomenutih jezika na F#. Čak možemo primetiti da je ovaj jezik uticao na pojavu još jednog zanimljivog jezika koji je takođe dizajniran za .NET platformu. To je jezik *Nemerle*.



Slika 1: Razvojno stablo

3 Primena i mogućnosti

Snaga F# programskog jezika leži u svedenoj sintaksi koja omogućava laku čitljivost koda, kao i efikasan razvoj programa koji zahtevaju primenu složenih matematičkih algoritama. Jezik omogućava brzo generisanje prototipova i njihovu brzu transformaciju u produkcioni kôd. Kôd napisan u F#-u lako se može paralelizovati, što je posebno značajno danas kada svi novi računari imaju više jezgara. F# danas ima široku primenu u obradi baza podataka, finansijskom modelovanju, statistici i bioinformatici. Takođe, domen primene svakodnevno raste.

F# je jezik koji ima pretežno funkcionalne karakteristike, ali on je svoju primenu pronašao u još mnogim vrstama programiranja. Neke od njih su: imperativno, objektno-orjentisano, paralelno, distribuirano, asinhrono, meta programiranje, veb programiranje, skript programiranje, analitičko programiranje, i sl. Danas se on može koristiti na velikom broju sistema, kao što su: Linux, MAC, Windows, Android, iOS, i sl. O ovome će biti više reči u nastavku.

4 Osnovne osobine programskog jezika F#

Neke od osnovnih osobina programskog jezika F# su:

• **Bezbedan**: F# programi se pre izvršavanja temeljno matematički proveravaju korišćenjem posebnih alata čime se garantuje njihova ispravnost, što ga čini bezbednim za pokretanje.
• **Funkcionalan**: Funkcije mogu biti ugnježdene, prosleđene kao argumenti drugim funkcijama i vraćene kao povratna vrednost.
• **Strogo tipiziran**: Provera tipova se vrši u toku kompilacije, kako bi se utvrdilo da su dobro definisani.
• **Automatski zaključuje tipove**: Tipovi se automatski zaključuju prilikom kompilacije u zavisnosti od konteksta u kome se javljaju, što dovodi do toga da se eksplicitno navođenje retko dešava.
• **Kompatibilan**: F# programi mogu da pozivaju programe koji su napisani na nekom od jezika koje podržava .NET platforma, kao na primer C# i da budu pozvani od strane istih.

Još neke specifičnosti jezika:

• Za razliku od drugih programskih jezika struktura **if/elif/else** u svakoj grani ima povratnu vrednost.
• Povratna vrednost ne mora uvek biti vraćena ključnom reči **return**. Ukoliko je return izostavljen, kao povratna vrednost se uzima poslednja izvršena naredba.
• Korišćenjem ključne reči **let** definišemo promenljive koje se ne mogu menjati što kôd čini sigurnijim od slučajnih promena. Ukoliko smo sigurni da će se vrednost promenljive menjati, uz let se dodaje ključna reč **mutable**.
• Struktura **switch/case** je u F# zamenjena *pattern matching*-om[3].
• Pored uobičajenih tipova podataka podržanih u većini programskih jezika, F# podržava novi tip, opcioni tip (eng. *option type*), koji označava vrednost promenljive koja može da postoji a i ne mora. Dve moguće vrednosti ovog tipa su: *Some(x)* i *None*.

5 Pattern matching u funkcionalnom programiranju

Kao glavna, funkcionalna paradigma programskog jezika F# podržava osnovne koncepte koji su podržani i od strane nekih drugih programskih jezika. U ovom odeljku nećemo davati prevelik značaj osnovama funkcionalne paradigme, već ćemo akcenat staviti na poklapanje obrazaca (eng. *pattern matching*) [3] koji je jedan od važnih mehanizama u F#. Predstavlja se kao *match ... with ...* konstrukcija koja kombinuje mehanizme dekompozicije i kontrole toka podataka. Kao takav, možemo ga koristiti sa jednostavnim kolekcijama (na primer: *list,tuple*) i *option* vrednostima. Naredni primer 1 pokazuje kako se definiše poklapanje obrazaca.

```
let urlFilter url agent =
 match (url,agent) with
 | "http://www.matf.bg.ac.rs", 99 -> true
 | "http://www.fsharp.org" , _ -> false
 | _, 86 -> true
 | _ -> false
```

Listing 1: Primer poklapanja obrazaca [3]

Izvedeni tip funkcije je:

```
val urlFilter: string -> int -> bool
```

Izraz nakon ključne reči *match* je tuple tipa *(string * int)*. Svako pravilo obrasca se uvodi sa simbolom | iza koga sledi pravilo. Nakon navođenja pravila sledi simbol -> nakon čega sledi rezultat. Kada se izvrše, pravila obrasca se koriste jedan po jedan i prvi obrazac koji je poklopljen određuje rezultat. Prvi obrazac se poklapa ukoliko su navedene identične vrednosti, dok poslednja tri, na mestima gde se javlja simbol _ za ulazne parametre, mogu da imaju bilo koju vrednost. U tabeli 1 ćemo prikazati koje vrste obrazaca postoje u F# i na koji način ih možemo formirati.

Tabela 1: Načini formiranja obrazaca u F#

Opšti oblik	**Vrsta**	**Primer**
(pat_1, ... ,pat_n)	Torka	(1,2,("3",x))
[pat_1, ... ,pat_n]	Lista	[x;y;z]
[|pat_1, ... ,pat_n|]	Niz	['a';'b';'c']
null	Null test	null

6 Asinhrono i paralelno programiranje

Razlog zbog kojeg izučavamo asinhrono i paralelno programiranje se ogleda u maksimalnoj iskorišćenosti hardvera. Ranije se ovaj način programiranja izbegavao zbog dodatne vremenske složenosti i rizika od bagova. Sa pojavom velikog broja biblioteka u kombinaciji sa funkcionalnim programiranjem i izbegavanjem bočnih efekata ovaj problem je smanjen.

Fokus ćemo staviti na to kako ubrzati računanje u F# koristeći asinhrono i paralelno programiranje. U F# bibliotekama mogu se naći moćni mehanizmi za rad sa nitima. Mogu se potpuno iskoristiti prednosti više procesora i jezgara tako što će se podeli jedan posao na više manjih.

6.1 Asinhrono programiranje

Asinhrono programiranje opisuje programe i operacije koje se izvršavaju u pozadini i završavaju nakon nekog vremena (na primer: preuzimanje novog email-a bez čestog proveravanja). Asinhroni programi na .NET platformi su pisani korišćenjem modela asinhronog programiranja (APM) [9]. APM je obrazac koji deli asinhrone operacije na dve metode: *BeginOperation* i *EndOperation*. Kada se pozove *BeginOperation* operacije počinju sa asinhronim izvršavanjem koje po završetku pozivaju *EndOperation* koji dohvata rezultat asinhronog izračunavanja. Problemi koji nastaju korišćenjem ovog obrasca su:

- Izostavljanjem poziva *EndOperation* dolazi do neželjenih efekata i curenja memorije.
- Ukoliko se nekoliko asinhronih izračunavanja vrati nazad teško je pratiti tok programa.
- Posledica prethodnog problema uzrokuje pojavu špageti programiranja.

Umesto APM-a sve asinhrone operacije mogu se izvršavati pomoću F# biblioteke asinhronih radnih tokova [2]. Ključna reč *async* se koristi za kreiranje bloka u koji se smešta kôd za izvršavanje korišćenjem *let!* i *do!* operatora. Rezultat ovog izvršavanja će biti tipa $Async<T>$ [2].

6.2 Paralelno programiranje

Paralelno programiranje se koristi za podelu poslova na n delova kako bi se dobilo n puta brže izvršavanje, što je retko u praksi. Za razliku od asinhronog, paralelni tok izvršavanja nije dostupan na prethodnim verzijama .NET platforme. Najjednostavniji način korišćenja paralelnog programiranja na .NET 4.0 platformi se zasniva na upotrebi *Parallel Extesions* biblioteke (PFX) [2]. Jedna od pogodnosti PFX biblioteke je ta što programer ne mora da misli o kontroli rada niti. Osnovnu strukturu PFX paralelizma predstavlja *Task* objekat, koji ima zadatak da izvrši posao nakon nekog vremena. Kao takav, nije dovoljan za paralelne aplikacije. Zbog problema koji se javlja pri radu Task objekta sa deljivim podacima uvodi se zaključavanje podatka (eng. *exclusive lock*). PFX biblioteka uvodi nove tipove kolekcija *System.Collections.Concurrent* [13] kao rešenje prethodnog problema.

7 Radni okviri

Najpoznatiji i najznačajniji radni okvir (eng. *framework*) za ovaj programski jezik je .NET koji je ranije već pomenut. .NET je pojednostavio razvoj RP(*Reactive Programming*) [19] aplikacija. Ovaj radni okvir je potpuno besplatan i na njemu je moguće kreirati veliki broj različitih aplikacija. On podržava veliki broj programskih jezika iz različitih paradigmi, editora i biblioteka za izgradnju mobilnih i veb aplikacija. Razvijene su dve glavne komponente za korišćenje rekurzivnog paralelnog

izračunavanja na lokalnoj mreži i obe su dostupne programima na .NET platformi [21].

Temelj .NET platforme je zajednička jezička infrastruktura CLI (*Common Language Infrastructure*) [10]. Pokretanje F# koda na ovoj platformi se vrši tako što na samom početku kompajler prevodi kôd u binarni fajl koji je preveden na asemblerski jezik višeg nivoa koji se zove MSIL (*Microsoft Intermediate Language*) [8]. Ako se MSIL asemblerski kôd prevede pomoću CLI kompajlera i potom izvrši, takvo izvršavanje je mnogo brže nego da se samo interpretira [2]. Postoje neke prednosti za korišćenje CLI-a, a ne običnog kompiliranja na mašinski nivo:

- kompatibilnost među jezicima
- mogućnost rada na više platformi
- mašinska nezavisnost

.NET platforma ima još jednu prednost, a to je prikupljanje smeća (eng. *garbage collector*) što omogućava programeru da ne razmišlja mnogo o alociranju i oslobađanju memorije, već da se samo skoncentriše na pisanje koda. Iako ne mora da obraća pažnju na smeće i rad sa memorijom poželjno je da programer zna na koji način taj sakupljač smeća radi, kako on oslobađa memoriju i rešava taj problem.

Naravno da ova platforma nije jedina koja se koristi za ovaj programski jezik, pored nje postoje: veb radni okviri (na primer: *Suave, Fable, ASP.NET Core, Giraffe, WebSharper, Freya, NancyFx, Serving Requests with IHttpHandler, Serving Requests with Azure, Functions, Pure F# Web API 2.0, SignalR, ServiceStack, ASP.NET Blazor*) i radni okviri za testiranje veba (na primer: *Web Testing, Frameworks, Canopy for Client-side Testing, Unit Testing Libraries*) [4].

8 Uputstvo za instalaciju

U ovom poglavlju biće opisano uputstvo za instalaciju programskog jezika F#. Prikazaćemo ih na Windows i Linux operativnim sistemima. Pored instalacije biće opisano i uputstvo za pokretanje F# programa.

8.1 Uputstvo za Windows

Na ovom operativnom sistemu postoje tri načina za instalaciju F#-a. To su: *Visual studio* [15], *Visual Studio Code* [16] i *JetBrains Rider* [1].

Pokretanje F# koda na Windows-u se vrši tako što prvo označite kôd ili deo koda koji želite da se izvrši. Zatim, desnim klikom na miša izaberite opciju *pošalji interaktivno* (eng. *send to interactive*) ukoliko je interaktivni prozor tj. prozor na kom se ispisuje rezultat otvoren ili iskoristite prečicu **Alt** + **Enter**.

8.1.1 Visual studio

Na Windows operativnim sistemima se uglavnom koristi *Visual Studio* alat. *Visual studio 2017* je alat koji dolazi sa podrškom za F# u svim verzijama: *Professional, Enterprise* i *Community* (verzija je potpuno besplatna). Međutim, ukoliko imate već instaliran *Visual Studio 2012/13/15 Professional* možete ga takođe koristiti zato što i on podržava alate za F# iako nije toliko napredan kao *Visual Studio 2017*.

8.1.2 Visual Studio Code

Visual Studio Code je besplatna platforma koja podržava mnogo jezika, a među njima i F#. On je podržan od strane *Ionide* [11].

1.korak: Instalirati *Visual Studio Code* za Windows.

2.korak: Pritisnuti **Ctrl**+**Shift**+**p** i ukucati sledeću naredbu za instalaciju *Ionide* paketa za *Visual Studio Code*:

```
ext install Ionide-fsharp
```

8.1.3 JetBrains Rider

JetBrains Rider je platforma .NET IDE izgrađena pomoću *InteliJ* i *ReSharper* tehnologije. *JetBrains* nudi podršku .NET i .NET Core aplikacijama na svim platformama.

1.korak: Instalirati *JetBrains* za Windows.

2.korak(opciono): Instalirati poslednju vrziju .NET Core SDK.

Takođe vam je potrebno da instalirate ceo *Visual Studio* ili F# kompajler.

8.2 Uputstvo za Linux

Kada koristite jezik koji podržava .NET platforma, kao što je F#, onda on zahteva Mono [18] softversku platformu. Većina Linux distributera sadrži neku verziju te platforme koja omogućava programerima lakše kreiranje aplikacija. Instalacija F# na Linuxu se izvršava na potpuno identičan način za *Ubuntu, Mint* i *Debian* verzije. Za pokretanje F# programa na Linux-u preko terminala koristi se sledeća naredba ukoliko se nalazite u direktorijumu u kom se nalazi primer koji pokrećete.

```
fsharpc primer.fs
```

8.2.1 Ubuntu/Mint/Debian

1.korak: Dodati *Mono* [18] repozitorijum vašem menadžeru paketa.

2.korak: Instalirati F# koji će istovremeno povući novu verziju *Mono*-a ukoliko je to potrebno.

```
sudo apt-get update
sudo apt-get install fsharp
```

9 Fizz Buzz problem

Fizz Buzz je problem koji je osmišljen za intervjuisanje kandidata koji konkurišu za posao programera. Treba napisati program koji štampa brojeve od 1 do 100 pri čemu, ako je broj deljiv sa 3 tada umesto tog broja

ispisuje *Fizz*, ako je deljiv sa 5 ispisuje *Buzz*, dok ako je deljiv i sa 3 i sa 5 ispisuje *FizzBuzz*.

```
let (|Fizz|Buzz|FizzBuzz|Other|) n =
    match (n % 3, n % 5) with
    | 0, 0 -> FizzBuzz
    | 0, _ -> Fizz
    | _, 0 -> Buzz
    | _ -> Other n

let fizzBuzz =
    function
    | Fizz -> "Fizz"
    | Buzz -> "Buzz"
    | FizzBuzz -> "FizzBuzz"
    | Other n -> n.ToString()

seq { 1..100 } |> Seq.map fizzBuzz|> Seq.iter (printfn "%s")
```

Listing 2: Fizz Buzz [6]

Navedeni primer 2 sadrži korisničko definisani šablon koji prepoznaje četiri moguća obrasca. Obrazac je dat u obliku para (eng. *tuple*) pri čemu je četvrti obrazac *wildcard* i vraća *Other* zajedno sa brojem koji odgovara tom obrascu. Šablon dalje koristimo u *pattern matching* izrazu koji nam vraća niske za ispis. Na kraju sprovodimo sekvencu od 1 do 100 kroz *Seq.map* koja primenjuje *pattern matching* nad svakim elementom date sekvence i zamenjuje ga odgovarajućom niskom. Konačno, funkcija *Seq.iter* iterira kroz sekvencu i ispisuje rezultat.

10 Jedinica mere - osobina svojstvena jeziku F#

Jedinica mere (eng. *Units of Measure*), svojstvena samo za F#, je mehanizam koji omogućava da se numeričkim tipovima pridruže dodatne informacije i na taj način spreče potencijalne greške pri radu sa različitim mernim jedinicama [14]. U nastavku je dat demonstrativni program 3 koji računa godišnju zaradu u evrima.

```
[<Measure>] type rsd
[<Measure>] type eur
[<Measure>] type hour
[<Measure>] type week
[<Measure>] type year

let hoursBilledPerWeek = 40.0<hour/week>
let weeksWorkedPerYear = 35.0<week/year>
let rsdPerHour = 1000.0<rsd/hour>
let exchangeRate = 0.008547<eur/rsd>

let eurPerYear = rsdPerHour * hoursBilledPerWeek *
    weeksWorkedPerYear * exchangeRate

let bonus = 500.0<eur/year>

printfn "%f" (eurPerYear + bonus)

// Output: 12465.8
```

Listing 3: Primer jedinice mere

Jedinica mere se definiše kao **[<Measure>] type** nakon čega sledi njen naziv. U našem primeru imamo vremenske jedinice (sat, vikend i godina) kao i jedinice koje označavaju valutu (dinar i evro). Dalje, pri definisanju promenljivih, dodatno pišemo informaciju o jedinici. Zbog toga, ako definišemo zaradu po satu dopisujemo **<rsd/hour>**. Program na kraju ispisuje zaradu sa godišnjim bonusom. Ukoliko bi promenili 10. liniju koda i stavili suprotan odnos valuta 4, tj. vrednost evra u odnosu na dinar, došlo bi do greške u fazi kompilacije 5.

```
10 let exchangeRate = 117.0<rsd/eur>
```

Listing 4: Promenjen odnos valuta

```
1 Error: The unit of measure 'eur/year' does not match the unit of
      measure 'rsd ^ 2/(eur year)'
```

Listing 5: Kompajler prepoznaje grešku

Daljom proverom vidimo da, ako na ovaj način definišemo odnos valuta, moramo da izmenimo formulu. Ukoliko u 12. liniji koda 6, umesto što množimo sa kursnom razlikom, podelimo sa istom, kompajler se više neće buniti i dobićemo validnu vrednost.

```
12 let eurPerYear = rsdPerHour * hoursBilledPerWeek *
      weeksWorkedPerYear / exchangeRate
```

Listing 6: Ispravljena linija

11 Zaključak

Ovim radom predstavili smo neke zanimljive koncepte koje F# poseduje, ali se nismo bavili osnovama koje mogu pomoći u učenju ovog jezika. Savetujemo učenje jezika ili unapređivanje znanja o njemu ukoliko ste se već susretali sa njim i ranije. Razlog za to je što ovaj jezik ima velike mogućnosti. Jednu njegovu važnu primenu naveli smo u radu kroz asinhrono i paralelno programiranje koje je danas veoma zastupljeno. Objasnili smo kako se na dva sistema mogu koristiti platforme za rad u F#-u. U literaturi koju smo koristili postoje knjige, članci i sajtovi na ovu temu koji u učenju mogu pomoći. Osim toga, postoji još puno lako dostupne literature na slične teme.

Literatura

[1] JetBrains s.r.o. Developed with drive and IntelliJ IDEA. Jet Brains rider, 2000-2019. on-line at: https://www.jetbrains.com/rider/.

[2] Smith Chris. *Programing F#*. O'Reilly Media, 1005 Gravenstein Highway North, 2009.

[3] Antonio Cisternino Don Syme, Adam Granicz. *Expert F#*. Kinetic Publishing Services, LLC, New York, NY 10013, 2007.

[4] F# Software Foundation and individual contributors. List of frameworks, 2012-2018. on-line at: https://fsharp.org/guides/web/.

[5] F# Software Foundation and individual contributors. The F# Language specification, 2012-2018. on-line at: https://fsharp.org/specs/language-spec/.

[6] Dave Fancher. *Book of F# Breaking Free with Managed Functional Programming*. No Starch Press, March 2014.

[7] Jon Harrop. *F# for Scientists*. Wiley-Interscience, New York, NY, USA, 2008.

[8] Microsoft. Assemblies in the Common Language Runtime (MSIL), 2019. on-line at: https://docs.microsoft.com/en-us/dotnet/framework/app-domains/assemblies-in-the-common-language-runtime.

[9] Microsoft. Asynchronous Programming Model (APM), 2019. on-line at: https://docs.microsoft.com/en-us/dotnet/standard/asynchronous-programming-patterns/asynchronous-programming-model-apm.

[10] Microsoft. Get started with F# with the .NET Core CLI, 2019. on-line at: https://docs.microsoft.com/en-us/dotnet/fsharp/get-started/get-started-command-line.

[11] Microsoft. Ionide, 2019. on-line at: http://ionide.io/.

[12] Microsoft. .NET framework, 2019. on-line at: https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet.

[13] Microsoft. System.Collections.Concurrent, 2019. on-line at: https://docs.microsoft.com/en-us/dotnet/api/system.collections?view=netframework-4.7.2.

[14] Microsoft. Tour of F#, 2019. on-line at: https://docs.microsoft.com/en-us/dotnet/fsharp/tour.

[15] Microsoft. Visual Studio, 2019. on-line at: https://visualstudio.microsoft.com/.

[16] Microsoft. Visual Studio Code, 2019. on-line at: https://code.visualstudio.com/.

[17] Robin Milner. Logic for computable functions: Description of a machine implementation. Technical report, Stanford, CA, USA, 1972.

[18] Mono Project. Mono, 2019. on-line at: https://www.mono-project.com/.

[19] ReactiveX. Reactive programing. on-line at: http://reactivex.io/.

[20] Don Syme. The Early History of F#. on-line at: https://fsharp.org/history/hopl-draft-1.pdf.

[21] V. V. Vasilchikov. On the recursive parallel programming for the .net framework. *Automatic Control and Computer Sciences*, 2014.