Izrada kalkulatora u C#

Ružić, Saša

Undergraduate thesis / Završni rad

2017

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University North / Sveučilište Sjever**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:122:092610

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-28**

Repository / Repozitorij:

University North Digital Repository

Sveučilište
Sjever

Završni rad br. 404/EL/2017

Izrada kalkulatora u C#

Saša Ružić, 5559/601

Varaždin, rujan 2017. godine

Sveučilište
Sjever

Odjel za elektrotehniku

Završni rad br. 404/EL/2017

Izrada kalkulatora u C#

Student

Saša Ružić, 5559/601

Mentor

Emil Dumić, doc. dr. sc.

Varaždin, rujan 2017. godine

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR–42000 Varaždin

Prijava završnog rada

Definiranje teme završnog rada i povjerenstva

ODJEL: Odjel za elektrotehniku

PRISTUPNIK: Saša Ružić | MATIČNI BROJ: 5559/601

DATUM: 14.7.2017. | KOLEGIJ: Programski jezici i algoritmi

NASLOV RADA: Izrada kalkulatora u C#

NASLOV RADA NA ENGL. JEZIKU: C# calculator development

MENTOR: Emil Dumić | ZVANJE: docent

ČLANOVI POVJERENSTVA:

1. Miroslav Horvatić, dipl.ing.el. - predsjednik povjerenstva
2. mr. sc. Matija Mikac, dipl.ing.el. - član
3. doc. dr. sc. Emil Dumić - član
4. Dunja Srpak, dipl.ing.el. (zamjenski član)
5.

Zadatak završnog rada

BROJ: 404/EL/2017

OPIS

U završnom radu je potrebno napraviti kalkulator u programskom jeziku C#. Objasniti osnove objektno orijentiranih jezika, s naglaskom na korišteni C#. Razvoj C# programskog jezika. Povezati C# i druge objektno orijentirane jezike, kao i razliku sa proceduralnim programskim jezicima. Objasniti matematičke funkcije korištene prilikom izrade kalkulatora, te potrebne ugrađene funkcije. Objasniti i prikazati funkcije kalkulatora napravljenog u C# programskom jeziku, te postupak izrade C# kalkulatora.

U radu je potrebno:
-objasniti razvoj C# programskog jezika
-objasniti razliku objektno orijentiranih i proceduralnih programskih jezika
-objasniti sličnosti i razlike C# programskog jezika, te drugih objektno orijentiranih jezika
-objasniti matematičke funkcije korištene za izradu C# kalkulatora, kao i korištene ugrađene funkcije
-izraditi C# aplikaciju tj. kalkulator

ZADATAK URUČEN: 24.08.2017. | POTPIS MENTORA: Emil Dumić

REPUBLIKA HRVATSKA 5 SVEUČILIŠTE SJEVER

Predgovor

Zahvaljujem mentoru doc. dr. sc. Emilu Dumiću koji je pratio cijeli proces nastajanja završnog rada te me svojim savjetima i dobrom voljom usmjeravao da prevladam probleme na koje sam naišao prilikom pisanja ovog rada. Zahvalio bih se svim profesorima i vanjskim suradnicima koji su me pratili prilikom ovog studija. Od srca zahvaljujem svojim roditeljima i prijateljima koji su mi bili podrška tijekom mog školovanja.

Sažetak

Kroz ovaj rad bit će opisan postupak izrade kalkulatora u programskom jeziku C#. U prvom poglavlju ovog rada iznesena je sažeta povijest nastanka kalkulatora, odnosno računala te razvoj programskih jezika. U drugom poglavlju opisano je razvojno okruženje koje je korišteno prilikom izrade ove aplikacije te postupak izrade same aplikacije. U trećem poglavlju objašnjene su matematičke i ostale funkcije koje kalkulator može izvršavati i svi procesi koje sadržava aplikacija. Pohađanjem kolegija Programski jezici i algoritmi proizašla je i ideja za izradu ove aplikacije. Programiranje kod tog kolegija zamišljeno je u programskom jeziku C++, no odluka je ipak pala na izradu Windows aplikacije u programskom jeziku C#. Za izradu ovog rada bilo je potrebno proučiti određeni broj izvora podataka te primijeniti stečena znanja o programiranju da bi se dobila aplikacija koja, uz jednostavno i efikasno korisničko sučelje, ispunjava sve zahtjeve definirane zadatkom završnog rada.

Ključne riječi: C#, Microsoft Visual Studio, Windows aplikacija, kalkulator

Popis korištenih kratica

OOP Objektno orijentirano programiranje

CLR *Common Language Runtime*

JIT *Just in Time*

TMC *Tabulating Machine Company*

IBM *International Business Machine*

VB.NET *Visual Basic.NET*

MSIL *Microsoft Intermediate Language*

CIL *Common Intermediate Language*

JVM *Java Virtual Machine*

Sadržaj

1. Uvod 1
2. Razvoj programskih jezika 4
2.1. Strojni i niži simbolički jezici 5
2.2. Viši simbolički jezici 6
2.3. Proceduralni i objektno orijentirani programski jezici 8
3. Izrada C# windows aplikacije 13
3.1. C# u odnosu na Java-u i C++ 15
3.2. Izrada Windows aplikacije kalkulator 15
4. Funkcije kalkulatora 24
5. Zaključak 39
6. Literatura 41

1. Uvod

Viši programski jezici počeli su se koristiti zbog potrebe da se složeni i čovjeku teško razumljiv strojni jezik nadomjesti nekim prihvatljivijim programskim jezikom lakšim i prikladnijim za korištenje [1]. Razvijanjem viših programskih jezika čovjek si je olakšao pisanje programa. Pri takvoj vrsti programiranja, odnosno razvoja programa pomoću viših programskih jezika, jedna naredba zamjenjuje cijeli skup naredbi strojnog jezika. Logika programiranja korištenjem takve vrste jezika slična je matematičkoj logici i logici ljudskog razmišljanja, a naredbe koje koristimo su zapravo kratice engleskih riječi. Više naredbi strojnog jezika se kod viših programskih jezika može zamijeniti jednom naredbom dok je kod koji pišemo vrlo sličan matematičkim operacijama. Ljudi koji pišu programe u nekom programskom jeziku nazivaju se programeri. Viši programski jezici omogućili su pisanje programa i ljudima koji znaju vrlo malo o samoj građi računala. Program koji je napisan u nekom od viših programskih jezika može se primijeniti na različitim modelima procesora i računala. Kod strojnih jezika takva primjena nije se mogla upotrijebiti te se to smatra jednom od najvažnijih odlika viših programskih jezika. Na računalu mora postojati programski prevoditelj ukoliko program pišemo u višem programskom jeziku, a njegova je zadaća da taj program prevede na strojni jezik koji računalo jedino razumije te nakon toga može izvršiti određeni program. U današnje vrijeme programiranje se sve više koristi i razvijaju se različiti programski jezici koji omogućavaju pisanje programa stručnjacima različitih struka koji su prilagođeni za razna područja te imaju svoje specifične zadaće. Programski jezik C# spada u grupu vodećih programskih jezika koji se danas koriste. On je objektno orijentiran tako da je pisanje programa s OOP (objektno orijentirano programiranje) relativno jednostavno. Osim toga, prati ga moćni MS (*Microsoft*) Visual Studio s velikim brojem mogućnosti što razvoj programa maksimalno ubrzava [2]. C# (*C sharp*) je programski jezik projektiran za Microsoftovu platformu .NET. Osnovu .NET-a predstavlja svakako .NET Framework (okvir, kostur). Radi se o posebnoj infrastrukturi koja programerima nudi gotova rješenja i funkcionalnosti da bi ubrzala i pojednostavila razvoj aplikacija svih vrsta i oblika. Najvažnija sastavnica .NET Frameworka zove se CLR (*Common Language Runtime*), dok se instrukcije u izvorni kod prevode pomoću JIT (*Just in Time*) kompajlera. C# je namijenjen programerima koji rade na Windows platformi, a njegova sintaksa je slična jezicima C, C++ ili drugim objektno orijentiranim jezicima kao što su Java i JavaScript. Često se koristi za izradu Web aplikacija te Windows aplikacija. C#

omogućava vrlo jednostavnu izradu standardnih korisničkih sučelja za Windows desktop aplikacije i to uz nekoliko tipova predložaka (*Windows Forms*, WPF i sl.). Cilj ovog završnog rada bio je izraditi aplikaciju za kalkulator. Aplikacija je razvijena u programskom okruženju MS Visual Studio 2010 pomoću *Windows Forms* grafičkog sučelja. *Windows Forms* služi za izradu aplikacija pomoću .NET Frameworka. Aplikacija računa aritmetičke operacije, sinusne trigonometrijske i hiperboličke funkcije, korijene, potencije i sl. Također sadrži i tipke kojima se, kod računanja sinusnih funkcija, odabire jesu li uneseni stupnjevi ili radijani. Aplikacija je jednostavna za korištenje te koristi ljudima za računanje jednostavnijih matematičkih problema. Pomagala pri računanju ljudi su počeli koristi još u godinama prije Krista [3]. Najstarijim pomagalom za računanje smatra se Abacus kojeg su koristili stari Grci i Rimljani. Do 17. stoljeća koristio se u Europi, a nakon toga ljudi su počeli računati na papiru te su se počeli koristiti arapski brojevi. Wilhem Schickard, njemački matematičar i astronom, 1623. godine izumio je prvi mehanički kalkulator koji je mogao zbrajati, oduzimati, množiti i dijeliti. Njegov kalkulator sljedećih 300 godina ostaje nepoznat sve do rekonstrukcije 60-tih godina 20. stoljeća. Blaise Pascal, francuski filozof i matematičar, 1642. godine izradio je mehanički stroj koji je nazvan Pascalina prema prezimenu izumitelja. Njegov stroj mogao je velike brojeve relativno brzo zbrajati i oduzimati, a Pascal je taj stroj napravio za svog oca koji je bio poreznik te mu na taj način olakšao računanje. Gottfreid Wilhelm Leibniz, njemački filozof i matematičar, izradio je stroj sličan Pascalini, ali je uz funkcije zbrajanja i oduzimanja, koje je sadržavao stroj Blaisea Pascala, imao i funkcije množenja i dijeljenja. Oba stroja nisu bila upotrebljiva u praksi zbog njihove nepouzdanosti. Charles Babbage, engleski matematičar i filozof, došao je na ideju da izumi analitički stroj za računanje s ciljem da ukloni nedostatke s kojima su izrađeni prvi mehanički kalkulatori. Stroj nije izrađen za života Charlesa Babbagea, a bio je namijenjen za rješavanje različitih zadataka. Stroj se smatra mehaničkom pretečom današnjih računala jer je sadržavao sve elemente današnjih računala u koje ubrajamo: centralnu jedinicu, ulazni i izlazni uređaj, program na bušenim karticama i memoriju. Tvrtka TMC (*Tabulating Machine Company*) se godine 1924. udružuje s još nekoliko srodnih kompanija te nastaje danas nam vrlo poznata tvrtka IBM (*International Business Machine*) koja svojom proizvodnjom postaje jedna od najpoznatijih tvrtki proizvodnje računala u svijetu. Pri izradi oruđa u II. svjetskom ratu bilo je potrebno izvršiti velike količine računskih operacija što u ono vrijeme kalkulatori nisu mogli izvršavati. To je dovelo do izuma prvog elektroničkog računala kojeg je engleski matematičar

Alan Turing 1943. godine izradio sa svojim timom uglednih stručnjaka, a računalo je moglo dešifrirati njemačke tajne poruke. Turing je svoj stroj nazvao *Computer* što dolazi od engleske riječi „*to compute*“ što znači računati te su prvi put korištene elektronske cijevi. Na temelju tog elektroničkog kalkulatora nastala su današnja računala.

2. Razvoj programskih jezika

Programski jezici počeli su se razvijati tijekom 20. stoljeća s razvojem računala [4]. Razvijanjem računala razvijali su se i programski jezici. Za izradu, odnosno pisanje programa u nekom programskom jeziku koristi se stroj kojeg nazivamo računalo. Različiti programski jezici mogu se koristiti kod pisanja programa, a taj proces naziva se još i kodiranje. Skup svih riječi i znakova nekog programskog jezika naziva se leksička struktura. Programski jezici obuhvaćaju skup naredbi i pravila njihova pisanja, a postupak rješavanja nekog zadatka daje nam algoritam kojeg je pogodno prikazati dijagramom tijeka. Pravila slaganja dozvoljenih riječi u programu nazivamo sintaksom, dok semantika predstavlja značenje ispravno napisanih naredbi. Razvoj programskih jezika može se opisati kroz pet generacija. Kod prve generacije programskih jezika, softver je bio pisan na strojnom jeziku što je bilo veoma teško i podložno greškama [5]. Strojni jezik je bio usklađen s elektronskim komponentama računala za koje je bio namijenjen. Programeri koji su programe pisali u takvom jeziku zapravo su pisali binarni kod za bilježenje instrukcija i podataka te za adresiranje memorije, a mogli su izvršavati samo jedan po jedan program. Programer je morao dovesti računalo u početno stanje nakon izvršenja jednog programa da bi moglo početi izvršavanje drugog programa. Računala druge generacije bila su mnogo naprednija te su mogla izvršavati oko sto tisuća operacija u sekundi. Svaki računalni sustav ove generacije imao je u sebi memorijsku jedinicu u kojoj su bili zabilježeni programi i podaci. Za programiranje računala te generacije koristili su se asemblerski jezici, dok su računala bila u mogućnosti obrađivati samo jedan po jedan asemblerski program. Programiranje na asemblerskom jeziku učinilo je rad programera efikasnijim, a za programiranje na tom jeziku koristile su se skraćenice ili simbolička imena za kodove instrukcija i adrese u memorijskoj jedinici. Kod treće generacije javljaju se računala koja su mogla izvršavati oko milijun operacija u sekundi, a dolazi i do razvoja operacijskih sustava. Operacijski sustavi razvijeni su s ciljem da upravljaju i nadgledaju radom računalnog hardvera čime je omogućeno istovremeno izvršavanje više programa (*eng. multitasking*). Računala ove generacije programirala su se pomoću viših programskih jezika što je dovelo do toga da programer ne mora učiti za svako računalo njegov asemblerski jezik. Razvoj objektno orijentiranih programskih jezika karakterističan je za razvoj računala četvrte generacije. Programeri pri radu s takvom vrstom jezika ne moraju posjedovati nikakvo znanje o samoj konfiguraciji računala. Prilikom pisanja programa treba samo navesti što su ulazni podaci, kako

ih treba obraditi te na kraju predstaviti rezultate napisanog programa. Programski jezici četvrte generacije dizajnirani su za izradu posebnih programa, dok su jezici pete generacije dizajnirani kako bi računalo riješilo dati problem bez programera [6]. Takva vrsta programskih jezika namijenjena je za rješavanje problema korištenjem ograničenja koja su dana u programu, umjesto korištenja algoritama koje je napisao programer. Programer pri tome mora samo brinuti koji se problemi trebaju riješiti i koji se uvjeti trebaju ispuniti neovisno o algoritmu. Jezici ove generacije uglavnom se koriste u razvoju umjetne inteligencije (oponašanje ljudskog mozga).

2.1. Strojni i niži simbolički jezici

Strojni jezik nastao je u ranim 50-im godinama prošlog stoljeća, a predstavlja najnižu moguću razinu prikaza programa te je usko vezan uz građu računala, odnosno ovisan je o procesoru [1]. Strojni jezik je zapravo binarni prikaz programa (kombinacija nula i jedinica). Takav oblik programa jedino računalo "razumije" i može izvršiti, dok se svaki drugi oblik programa mora prije samog izvršenja prevesti u strojni jezik. Ne postoji strojni jezik takav da bi bio primjenjiv na drugoj vrsti računala, odnosno procesora, već svaki tip procesora ima svoj vlastiti strojni jezik. Pisanje programa na strojnom jeziku traje jako dugo te je velika mogućnost pogreške pri čemu je takva vrsta programiranja za običnog čovjeka preteška pa se time bave usko specijalizirani stručnjaci. Pisanje takvog programa je vrlo složeno pa se većinom ne koristi u praksi. Programeri su u vrijeme strojnih jezika tražili optimalni algoritam koji bi sadržavao manje koda te bi se time mogućnost pogreške smanjila [4]. Težili su razvoju programskih jezika kojima bi sve te silne nizove nula i jedinica zamijenili slovima ili riječima te na taj način olakšali čovjeku, odnosno programeru pisanje programa. Takvi programski jezici nazivaju se simbolički. Niži simbolički programski jezici nazivaju se još i asembleri, a uvedeni su da bi pojednostavili uporabu strojnih jezika [1]. Korištenje strojnog jezika pojednostavilo se na način da su u taj jezik uvedena imena za određene operacije i naredbe koje se logično lakše pamte od brojčanog zapisa nula i jedinica. Ostali dijelovi, osim imena naredbi i operacija, su i dalje vrlo složeni. Dio programa se i dalje sastoji od brojčanih kodova, odnosno kombinacija nula i jedinica, čime programer mora i dalje detaljno poznavati građu računala. Kod se putem specijalnog kompajlera pretvara u strojni kod koji procesor može direktno izvršiti. Programi napisani u asemblerskom jeziku odlikuju se mogućnošću slanja direktnih komandi procesoru, kao i iskorištavanju cijelog dosega arhitekture računala. Neke od glavnih mana takvih programa su loša čitljivost, što je posebno izraženo pri

velikim projektima i kompleksnosti koda, kao i praktična nemogućnost prebacivanja istog koda na drugu procesorsku arhitekturu. Zbog tih mana se asemblerski jezik danas koristi samo u vremenski kritičnim programima, kao što su adapteri (*engl. driver*) za grafičke kartice ili u prostorno kritičnim programima u integriranim sustavima. Primjeri takvih sustava su mikrokontroleri.

2.2. Viši simbolički jezici

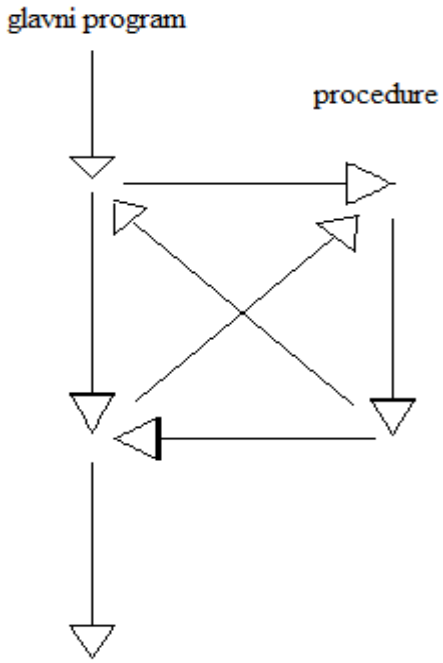
Viši simbolički programski jezici počeli su se koristiti s ciljem da se složeni, nepregledni i čovjeku težak za pamćenje strojni jezik nadomjesti prikladnijim i razumljivijim programskim jezikom koji bi programerima bitno olakšao posao [1]. Kod takvih se jezika manji ili veći skup naredbi strojnog jezika nadomješta jednom naredbom višeg programskog jezika. Mogu biti prilagođeni za izvršavanje različitih zadaća i primijenjeni u različitim strukama jer je zapis programa sličan matematičkom zapisu. Naredbe višeg simboličkog jezika također se, kao i kod asemblerskih jezika, prevode u binarni zapis, ali u ovom slučaju svaka se naredba prevede u mnogo bajtova. Kao što je već i prije spomenuto, programi se pomoću viših programskih jezika relativno jednostavno pišu jer je logika programiranja slična ljudskoj logici razmišljanja, a počeli su se koristiti iz razloga što se isti program može primijeniti na različitim vrstama računala, odnosno procesora. Za tvrtke koje izrađuju i prodaju programe to je vrlo bitno svojstvo jer se jedan te isti program ne mora ponovno izrađivati, već se može prodavati u više istih primjeraka. Program napisan na višem programskom jeziku mora biti preveden na strojni jezik koji je jedino razumljiv računalu da bi mogao izvršavati funkciju za koju je namijenjen. Za to služe programski prevoditelji. Prevođenje u strojni jezik se može izvršavati na više načina, a razlika je samo u načinu prevođenja. Ne postoje univerzalni programski prevoditelji, već postoje posebni programski prevoditelji ovisno o tome koja vrsta jezika i stroja se koristi. U današnje vrijeme prevođenje se prepušta računalima, dok programeri većinom pišu programe u nekom od viših programskih jezika. Programski prevoditelji mogu se podijeliti u dvije skupine: interpretere i kompajlere. Kod interpretera se, u trenutku izvođenja programa, svaka naredba izvornog programa prevodi se u jednu ili više naredbi strojnog jezika te se zatim te naredbe strojnog jezika mogu izvršiti [7]. Interpreter zatim uzima sljedeću naredbu korištenog višeg programskog jezika, prevodi u strojni jezik i tako sve dok ne dođe do kraja napisanog programa. Ukoliko se program ponovno pokreće, postupak prevođenja se ponavlja. U programu se mogu javiti i neke pogreške. Kod interpretera se prevođenjem programa naredbu po naredbu otkrivaju određene vrste

pogrešaka te pri pronalasku pogreške računalo javlja korisniku da je došlo do pogreške te zahtjeva njezino ispravljanje [1]. Nakon ispravljanja pogreške, postupak se nastavlja dalje te prevođenje može trajati neko određeno vrijeme što usporava izvođenje programa, a to za neke zahtjevnije i složenije programe može biti primjetno. Neki od nedostataka interpretera su relativno sporiji rad u odnosu na druge jezične prevoditelje te nužnost isporuke izvornog programa korisniku [7]. Kompajliranjem se naziva postupak prevođenja izvornog programa u izvršni pomoću jezičnog prevoditelja kojeg nazivamo kompajler, a to se vrši na način da se cjelokupni izvorni program prevodi u strojni jezik odjednom [8]. Programeri su nekad bili mnogo svjesniji postojanja kompajlera jer su se programi prvo pisali u nekom od uređivača teksta te se spremali kao tekstualni dokument [9]. Kompajler se nakon toga pozivao kao poseban program s ciljem oblikovanja objektnog programa koji se mogao izvršavati. U današnje vrijeme programeri nisu ni svjesni postojanja kompajlera jer su toliko dobro ugrađeni u razvojno okruženje da se ni ne primjećuje da računalo prevodi neki program pisan u višem programskom jeziku u strojni jezik. Kod kompajlera su, za razliku od interpretera, izvorni i izvršni program potpuno odvojeni i pri izvođenju neovisni [8]. Izvršni program se može izvršavati i bez postojanja izvornog programa pa korisnik dobiva samo izvršnu, odnosno prevedenu inačicu programa. Takva inačica programa za IBM kompatibilna računala ima ekstenziju *.com* ili *.exe*. Inačica *.exe* se stvara tek kada u programu nema grešaka, odnosno ukoliko se u programu pronađu greške, korisnik mora prvo greške ispraviti, a kad će program biti potpuno ispravan stvorit će se izvršna exe inačica [1]. U usporedbi s interpreterima, kompajleri pružaju brži i praktičniji rad iz razloga što se program prevodi samo jednom, no i interpreteri imaju svojih prednosti. Interpretirani program se može prekinuti u bilo kojem trenutku te se u njemu mogu obaviti neke izmjene i nastaviti izvođenje od mjesta prekida, dok to kod kompajliranih programa nije lako izvedivo, a ponekad i nemoguće [9]. Neki od programskih jezika po svojoj prirodi nisu pogodni za kompajliranje, nego samo za interpretiranje gdje se najčešće od proceduralnih jezika interpretira BASIC, dok se FORTRAN, Pascal i C isključivo kompajliraju. Prilikom pisanja programa, pošto računalo niti jedan od zadataka ne zna riješiti samo, potrebno mu je dati naputak za rješavanje određenog problema, a za to nam služi algoritam [1]. Algoritam je način, odnosno uputa računalu kako da riješi neki zadatak. Algoritam izrađuje u pravilu sam programer, a on se sastoji od niza naredbi čijim se izvršavanjem rješava neki problem. Pri pisanju algoritma programeri često koriste neka pomagala, primjerice dijagram tijeka, dok se kod manjih i jednostavnijih programa algoritam

može i sastaviti u glavi bez upotrebe posebnih pomagala. Prilikom pisanja programa može doći do pogrešaka koje možemo podijeliti u dvije skupine: sintaksne i logičke. Pogreške nastale nepoštivanjem propisanih pravila pisanja programa, odnosno greške koje se javljaju pri korištenju naredbi (npr. krivo napisana riječ, spojene dvije riječi i sl.), nazivaju se sintaksne pogreške. Računalo ih javlja korisniku tijekom prevođenja te ih je uglavnom lako ispraviti. Logičke pogreške nastaju zbog uporabe krivih formula, metoda i algoritama, no njih ne otkriva računalo, već je zadaća programera da to prepozna. Otkrivanje takvih pogrešaka ponekad zahtjeva dublju analizu programa, a otkrivaju se pokusnim izvođenjem programa koji obuhvaćaju sve moguće slučajeve. Kod definiranja algoritma spomenut je pojam dijagram tijeka poznat još i pod nazivom blok dijagram koji nam služi za grafički prikaz programa, a koristi programeru za lakše snalaženje u programu, za analizu programa, za lakše pronalaženje logičkih pogrešaka u programu te mu služi kao pomoć kod samog pisanja programa. Pri sastavljanju blok dijagrama koriste se posebni standardni simboli kojima se označavaju pojedine naredbe i operacije te je takav prikaz programa vrlo pregledan i jednoznačno određen.

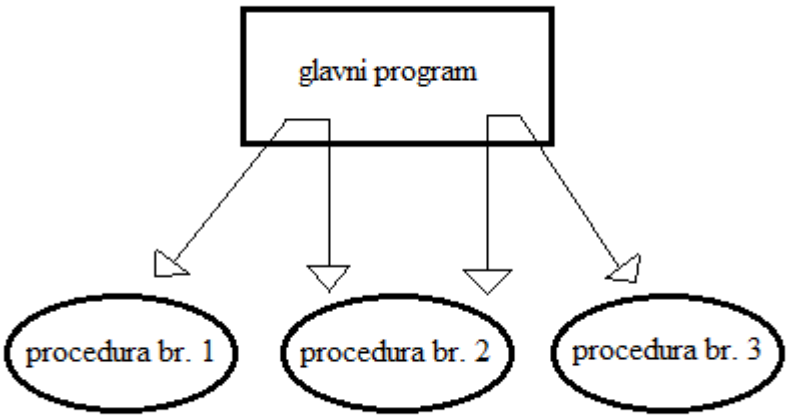
2.3. Proceduralni i objektno orijentirani programski jezici

Kod proceduralnog programiranja program se sastoji od niza naredbi koje računalo treba izvesti, odnosno opisuje se računalni proces u smislu stanja programa i naredbi koje mijenjaju stanje [10]. Viši proceduralni programski jezici upotrebljavaju varijable i složenije naredbe te omogućuju izračunavanje složenih mješovitih izraza koji se sastoje od relacijskih, aritmetičkih i logičkih operatora. Vrijednost izraza se nakon izračunavanja dodjeljuje memorijskoj lokaciji. Proceduralno programiranje se većinom bazira na naredbama uvjetnog i bezuvjetnog grananja te pomoću petlji. Kod uvjetnog grananja, da bi se blok naredbi izvršio, mora biti zadovoljen postavljeni uvjet, dok kod naredbi bezuvjetnog grananja nema nikakvih uvjeta, a one najčešće služe za bezuvjetni skok ili poziv potprograma, odnosno procedure. Petlje se koriste u svrhu izvršavanja naredbi određeni broj puta ili do promjene uvjeta. Pomoću proceduralnog programiranja moguće je kombinirati povratne dijelove naredbi na jednom mjestu [11]. Za pozivanje neke procedure, odnosno potprograma, koristimo ime tog potprograma te nakon njegovog pozivanja izvršavaju se naredbe koje se nalaze u tom potprogramu.



Slika 2.1 Izvršavanje procedura

Slika 2.1 pokazuje izvršavanje procedura: izvršava se prvi dio koda koji se nalazi u glavnom programu. Poziva se prva procedura te se s glavnog programa skače u pomoćni i izvršava se dio koda koji se nalazi u pomoćnom programu. Nakon izvršavanja tih instrukcija program se vraća natrag u glavni i nastavlja s izvršavanjem tamo gdje je prethodno bio prekinut. Nakon što se izvrši dio koda u glavnom programu ponovo se poziva nova procedura te se postupak ponavlja. Tako napisan program programeru omogućuje bolju preglednost i manja je mogućnost pogreške, odnosno ako je potprogram ispravno napisan, svaki put kad se pozove on daje točan rezultat.



Slika 2.2 Razdvajanje glavnog programa na procedure

Slika 2.2 prikazuje glavni program koji ima kontrolu nad procedurama te upravlja prijenosima argumenata svakoj od procedura. Tako se dobiva jedan program koji je podijeljen na više manjih dijelova nazvanih procedurama. Uz proceduralno programiranje u današnje vrijeme najzastupljenije je objektno orijentirano programiranje [12]. Proceduralno programiranje se u pravilu bazira na pozivu niza funkcija i njihovoj zadaći dok se OOP bazira na objektima i njihovoj međusobnoj komunikaciji. OOP je u svijet programiranja uvelo novosti koje ga svrstavaju u sam vrh današnjeg programiranja. Objektno orijentirani model omogućava logičnije projektiranje i realizaciju programa te na razumljiviji način oslikava realni svijet s obzirom da se svaki realni sustav može prikazati kao skup objekata koji su u nekakvom srodstvu ili su međusobno povezani. OOP je svoj razvoj započelo prije dvadesetak godina kao odgovor na tzv. softversku krizu. To je pojam koji je nastao kad su porasli zahtjevi za kompleksnijim softverom iz razloga što su se računala sve brže razvijala, a time i njihove mogućnosti. Organizacija velikih softverskih projekata više se nije mogla oduprijeti velikim zahtjevima korisnika, dok su projektiranje, izrada i održavanje softvera postali preskupi i preveliki poslovi u koje se ulagalo mnogo truda i rada, a dobivalo sve manje rezultata. Uz sve to greške u softveru postajale su sve teže uočljive i teže za otklanjanje, s obzirom da su se zahtjevi korisnika drastično povećali. Za to su uglavnom krivi i sami programeri koji su korisnicima pokazali što su sve računala u stanju postići. Ona su mogla postići mnogo više nego što su korisnici mogli očekivati. Korisnici su kao odgovor na to počeli tražiti mnogo više nego što su sami programeri mogli očekivati. Kako bi se korisnicima odgovorilo na sve veće zahtjeve, došlo je do potrebe da se prošire timovi programera

koji projektiraju složenije projekte. Kad velik broj ljudi radi na nekom većem i složenijem projektu potrebno je podijeliti posao tako da se dijelovi koje su projektirali različiti ljudi mogu sklopiti u cjelinu. Programerima koji su radili u timu jasno je koliko truda je potrebno uložiti da bi se dijelovi programa uspješno sklopili u cjelinu te se kao ključ uspjeha velikih softverskih projekata uzima podjela poslova na dijelove koji imaju jasno definirane zadaće i koji ne ulaze jedan drugome u "unutrašnjost". Kao što je već spomenuto, OOP se bazira na uporabi objekata koji su građeni od varijabli koje se nazivaju atributima, funkcija koje se nazivaju metodama te identifikatora koji pomaže u razlikovanju istovrsnih objekata [13]. Atributi su varijable kojima se opisuju trenutna svojstva pojedinog objekta. Metode su zapravo funkcije kojima se mijenjaju ili čitaju svojstva objekta. Grupiranjem objekata s istim atributima i metodama moguće je stvoriti klase i podklase kojima pripadaju ti objekti. Objektima se kod ove vrste programiranja mogu smatrati predmeti i pojave s kojima se susrećemo svakodnevno u današnjem svijetu. Osobama koje razvijaju aplikaciju tako je lakše zahtjeve koje im je zadao korisnik pretvoriti u programsko rješenje. OOP se u pravilu sastoji od tri osnovna koncepta: učahurivanje, nasljeđivanje i polimorfizam. Pomoću tih koncepata moguće je dizajnirati računalne sustave koji su iskoristivi u više različitih aplikacija. Učahurivanje je postupak da se onemogući direktni pristup vrijednostima atributa određenih objekata iz nekih drugih dijelova koda. Tu se prvenstveno misli da se uslijed vršenja radnji nad objektima otkloni mogućnost da se objekt nađe u nepredviđenom stanju. To se može dogoditi ukoliko se objektu pristupa iz više različitih dijelova koda te se objekt može dovesti u nepredviđeno stanje ako u metode za pristup vrijednostima njegovih varijabli nisu ugrađeni sigurnosni mehanizmi za takve slučajeve. Kod nasljeđivanja se klasa ili objekt baziraju na nekoj postojećoj klasi ili objektu pri čemu koriste postojeće atribute i metode. Te klase tada nasljeđuju sve ili samo neke dijelove svoje nadklase te postaju njena podklasa. Podklasa se može nadopunjavati novim atributima i metodama te može izmijeniti ili nadopuniti postojeće metode svoje nadklase, ali ne i njene atribute. Polimorfizam se može podijeliti u dva različita koncepta. Prvi koncept omogućuje objektu da posjeduje više metoda istog imena koje zatim kroz argumente primaju različite podatke. Određena metoda izvršava se ovisno o tome koji su podaci poslani kroz te argumente. Drugi koncept veže se uz nasljeđivanje te omogućava spremanje objekta tipa podklase u varijablu deklariranu za tip podatka njene nadklase. Takav pristup omogućuje čuvanje više varijabli različitih podklasa u jednoj listi deklariranoj za njihovu zajedničku nadklasu. OOP daje odgovore na mnogobrojne probleme. Ono nudi mehanizme koji

automatski kontroliraju interakcije između dijelova softvera. Njega su smislili programeri kao način da sebi olakšaju život. Neki od predstavnika programskih jezika kod OOP-a su: Java, C++, C#, Python, Delphi, PHP.

3. Izrada C# windows aplikacije

C# je objektno orijentiran programski jezik kojeg je razvio Microsoftov tim na čelu s Andresom Hejlsbergom [14]. Prva verzija (C# 1.0) C# programskog jezika pojavila se 2001. godine te su se ubrzo razvijale i novije verzije ovog programskog jezika. C# predstavlja nasljednika programskih jezika C i C++, a dobio je ime *Sharp* prema inspiraciji glazbene notacije što znači da se napisana nota izvodi za pola koraka više. Sintaksno ovaj jezik nije složen (ima oko 80 rezerviranih riječi), ali je vrlo izražajan u dijelu gdje je potrebno riješiti neki problem softverskog razvojnog procesa. Aplikacija za kalkulator napravljana je u razvojnom okruženju Microsoft Visual Studio 2010 u obliku Windows forme. Microsoft Visual Studio je integrirano razvojno okruženje kojega je napravila tvrtka Microsoft, a koristi se u svrhu razvoja računalnih programa za Windows, web aplikacije i sl. [15]. Visual Studio podržava različite programske jezike te dozvoljava programeru da podržava (u različitoj mjeri) skoro bilo koji programski jezik pod uvjetom da servis za taj jezik postoji. Ugrađeni jezici su C, C++, VB.NET (Visual Basic.NET)-a, C# i F#. Neke od najkorisnijih značajki dostupnih C# programerima dolaze iz .NET okvira koji pruža izvršno okruženje i biblioteke za C# i sve druge .NET jezike kao što je primjerice VB.NET [2]. C# je projektiran za .NET i jedna od glavnih prednosti njegove bliske povezanosti s .NET okvirom je što je rad sa značajkama okvira, kao što su biblioteka klasa, vrlo prirodan. .NET je ime najmodernije platforme koje Microsoft izdaje za softverske tehnologije budućnosti te predstavlja dugoročni i strategijski plan razvoja ne samo u Microsoftu. .NET daje realne osnove da postane osnovna platforma za razvoj modernih aplikacija. Radni okvir .NET razvijen je s ciljem da se osigura okruženje za razvoj svih modernih aplikacija na Windows operativnom sistemu. Jedna od osnovnih osobina ove platforme je njena orijentacija prema distribuiranim aplikacijama preko Interneta. Za razliku od .NET-a, pri razvoju aplikacija u Linux operativnom sustavu ili razvoju aplikacija za Andorid i iPhone mobilne uređaje koristi se Mono Framework.

Verzija	**Datum izlaska**	**.NET Framework**	**Visual Studio**
C# 1.0	Siječanj 2002.	.NET Framework 1.0	Visual Studio .NET 2002
C# 1.1 i 1.2	Travanj 2003.	.NET Framework 1.1	Visual Studio .NET 2003
C# 2.0	Studeni 2005.	.NET Framework 2.0	Visual Studio 2005
C# 3.0	Studeni 2007.	.NET Framework 2.0,	Visual Studio 2008

		.NET Framework 3.0, .NET Framework 3.5	Visual Studio 2010
C# 4.0	Travanj 2010.	.NET Framework 4.0	Visual Studio 2010
C# 5.0	Kolovoz 2012.	.NET Framework 4.5	Visual Studio 2012 Visual Studio 2013
C# 6.0	Srpanj 2015.	.NET Framework 4.6	Visual Studio 2015
C# 7.0	Ožujak 2017.	.NET Framework 4.6.2	Visual Studio 2017

Tablica 3.1 Razvoj programskog jezika C#

Tablica 3.1 prikazuje razvoj programskog jezika C# od prve verzije pa do danas te prikazuje verziju .NET platforme koju je određena verzija programskog jezika koristila u kojoj verziji razvojnog okruženja Microsoft Visual Studio. .NET Framework sadrži paket kodova koji je neophodan programerima u razvoju aplikacija [14]. Taj paket uključuje i virtualnu mašinu koja upravlja izvršavanjem programa pisanih za .NET Framework. Ta virtualna mašina poznata je pod nazivom CLR, a kao što je već i spomenuto u uvodu, to je softversko okruženje u kojem se izvršavaju programi pisani za .NET Framework. S CLR-om programeri ne moraju razmišljati o kapacitetima procesora koji će izvršavati njihove programe. Svi jezici koji su obuhvaćeni paketom Microsoft Visual Studio-a imaju podršku za .NET klase. Kako svi oni imaju "ispod" CLR to ih čini ravnopravnim više nego ikada. Drugi programski jezici na ovoj platformi su postojali i prije nastanka .NET-a. To ovom programskom jeziku omogućava optimalno korištenje takvog okruženja. Mnoge aplikacije koje su djela nezavisnih programera, kao preduvjet za instalaciju i rad zahtjevaju da neka verzija .NET Framework-a bude instalirana na računalu. Preporuka je da se izbjegava instaliranje zahtjevane verzije te da se umjesto toga instalira najnovija verzija .NET-a koju vaša verzija operativnog sustava Windows podržava. U okruženju .NET, programi se ne prevode u izvršne datoteke, već u programske sklopove koji sadrže instrukcije na Microsoftovom međujeziku koji se zove MSIL (*Microsoft Intermediate Language*) ili danas poznatiji kao CIL (*Common Intermediate Language*). CIL datoteke koje izradi C# vrlo su slične datotekama koje su nastale u drugim programskim jezicima koji rade s .NET okvirom. Program napisan na programskom jeziku C# prevodi se na CIL u fazi prevođenja (*engl. Build*) projekta. Međujezični kod se čuva u datoteci na disku, a kad se program pokrene međujezični kod se ponovno prevodi pomoću programskog prevoditelja, odnosno JIT kompajlera. Kao rezultat dobiva se strojni kod kojeg izvršava procesor. JIT prevoditelj analizira međujezični kod te daje vrlo efikasan strojni kod koji se brzo izvršava. Prilikom izvršavanja programa, JIT

prevoditelj prevodi samo po potrebi te se prevedeni kod čuva u memoriji kako bi se mogao ponovno koristiti.

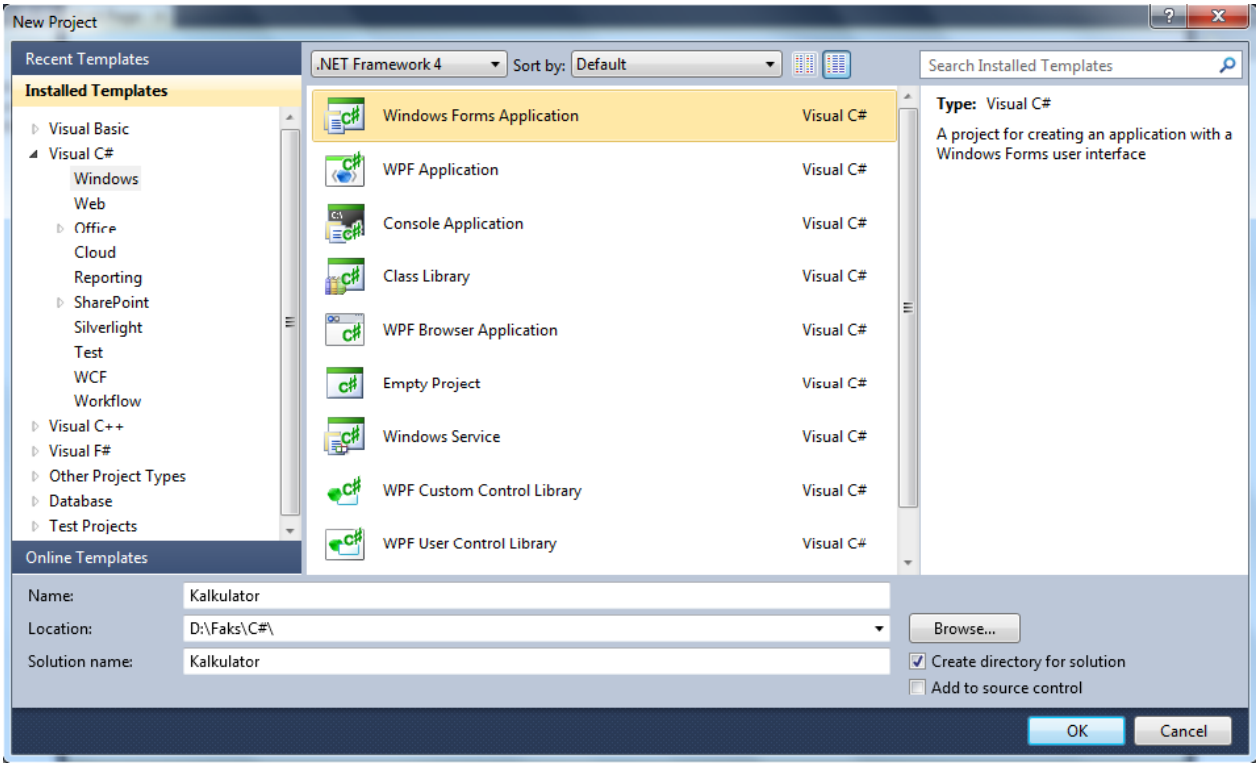
3.1. C# u odnosu na Java-u i C++

Danas su programski jezici, više nego ikada, moćni tako da kvaliteta softvera zavisi najviše o sposobnostima programera [16]. Ipak, razlike i sličnosti između jezika postoje i dobro ih je poznavati. C#, C++ i Java svrstavaju se u nasljednike jezika C. Pripadaju jezicima nove generacije i uključuju suvremene osobine. C# i Java koriste tzv. "*garbage collection*" (skupljač memorijskih otpadaka u upravljanoj dinamičkoj memoriji) koji oslobađa programera razmišljanja o dijelovima programiranja niskog nivoa, odnosno oslobađa memoriju. Objekte koji nisu duže vrijeme u upotrebi u nekom programu, sakupljač smeća "reciklira" te tako oslobađa memoriju. Kod C++-a to nije ugrađeno u sustav. Sintaksno su ovi programski jezici vrlo slični. Osnovna zajednička osobina za oba jezika je da se prevođenje (kompajliranje) vrši do međujezika. Za C# taj jezik se, kao što je već spomenuto, naziva MSIL ili CIL, dok je za Javu to *Java bytecode*. Kod Jave izvorni kod se ne prevađa u strojni, već se kompajlira do *bytecode-a* koji zahtjeva posebno okruženje da bi se mogao izvršiti. To okruženje naziva se JVM (*Java Virtual Machine*). Ideja je da ako se kod napiše i kompajlira na jednoj platformi (npr. Mac OS), taj isti *bytecode* može izvršavati na svim ostalim platformama koje imaju JVM (Windows, Linux) bez potrebe za ponovnim kompajliranjem na toj platformi [17]. Sličnosti C#-a s Javom su očigledne, no oba jezika nastala su kao nadogradnja na jezik C++. C++, Java i C# spadaju među najpopularnije programske jezike i oni se u današnje vrijeme najviše koriste. Prije svega C# i Java su izbor onih koji vole C ili C++ jezik, ali i onih koji su naučili programirati u C jeziku, odnosno C++-u, a željeli bi naučiti još više o novim tehnologijama. Budućnost definitivno pripada jezicima C# i Java.

3.2. Izrada Windows aplikacije kalkulator

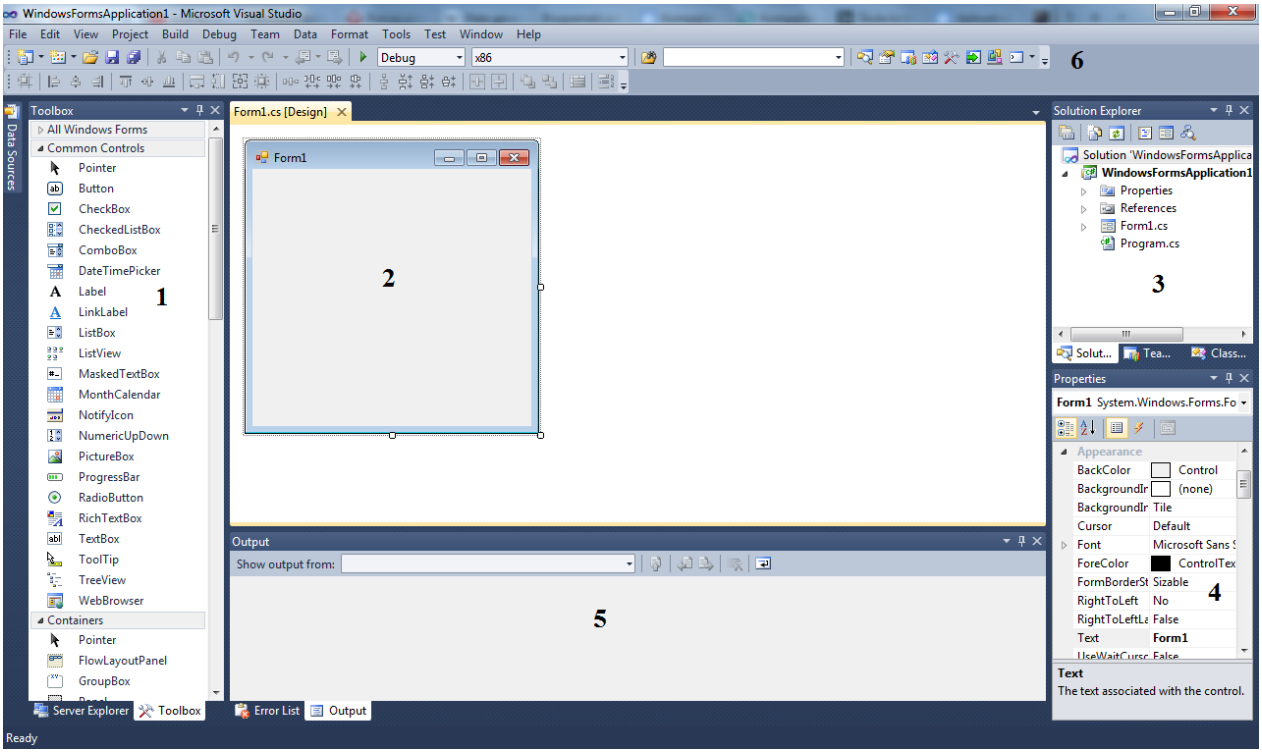
Karakteristika Windows aplikacija je bogato grafičko korisničko sučelje (GUI). Windows aplikacije se mogu sastojati od jedne ili više formi na kojima se nalaze različite kontrole (tipke, labele (natpisi), combobox, listbox, polje za unos teksta i sl.). Svaka forma i kontrola ima sposobnost prihvaćanja odgovarajućih događaja koje proizvodi korisnik (npr. lijevi klik miša ili samo klik, dvostruki lijevi klik, desni klik i sl.). Sve akcije se jednim imenom nazivaju događajima (*engl. Events*) tako da se ovakav način programiranja može nazvati i programiranje

upravljano događajima. Takva vrsta programiranja se zbog ove karakteristike naznačuje i drugačijim pristupom u razvoju aplikacije u odnosu na npr. proceduralno programiranje. Najprije se generira izgled korisničkog sučelja, a zatim se određuju događaji za pojedine forme i kontrole na koje će sustav reagirati, tj. dati neki odziv. Na kraju se za svaki događaj piše odgovarajuća funkcija kojom se definira odziv aplikacije na neku akciju korisnika. Pošto se u prvi plan stavlja sučelje, tj. forme i kontrole koje su većinom standardizirane po pitanju funkcionalnosti, suvremena okruženja za razvoj Windows aplikacija nude unaprijed definirane klase kojima su opisane forme i kontrole. Često se ovakav razvoj Windows aplikacija naziva i vizualnim programiranjem. Takva situacija je i s okruženjima Visual Studio-a koja nude bogat skup unaprijed definiranih biblioteka klasa. Aplikacija je, kao što je već i prije navedeno, izrađena pomoću Microsoftovog okruženja Visual Studio 2010 u obliku Windows forme.



Slika 3.1 Otvaranje novog projekta

Slika 3.1 prikazuje pokretanje okruženja Visual Studio gdje je kreiran novi projekt kojemu je dodijeljeno ime te određena lokacija spremanja projekta. Odabrano je da aplikacija bude u obliku Windows forme kao što je vidljivo na slici 3.1. Klikom na "OK" otvoreno je razvojno okruženje, odnosno svi potrebni alati za izradu Windows aplikacije koji su vidljivi na slici 3.2.

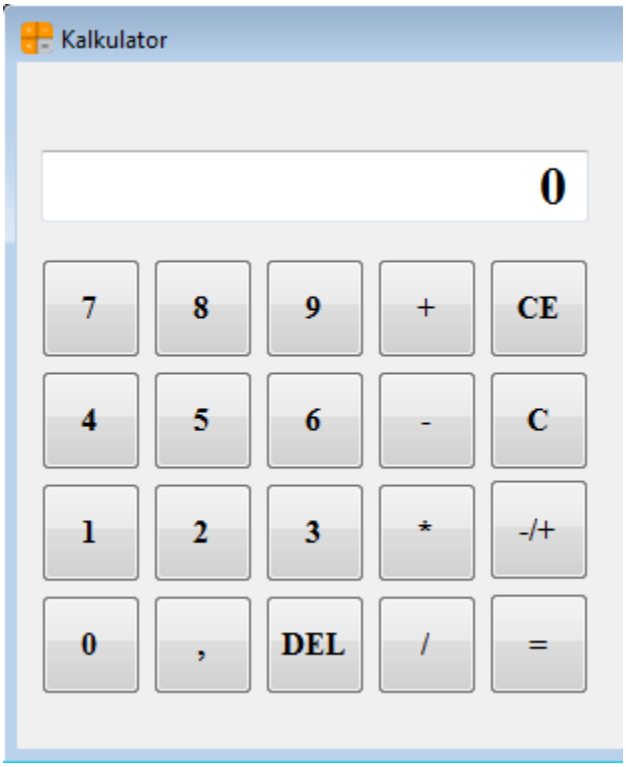


Slika 3.2 Razvojno okruženje za izradu aplikacije

Elementi razvojnog okruženja koji su označeni brojevima na slici 3.2 su:

- 1. Toolbox (traka s alatima) - sadrži gotove komponente (tipke, labele i sl.) koje se jednostavno stavljaju na formu (kliknemo na alat čime ga označimo te klikom na bilo koje mjesto u prozoru *Form1* stavljamo odabrani alat),
- 2. Forma - osnovno sučelje Windows aplikacije i na nju se stavljaju alati,
- 3. Solution Explorer - prozor u kojem se prikazuju svi dokumenti jednog projekta, odnosno *Solution-a* (jedan *Solution* može sadržavati više projekata kod ozbiljnijih softverskih proizvoda),
- 4. Properties - prozor u kojem se prikazuju sve osobine označenog objekta, kao i svi mogući događaji tog objekta,
- 5. Build Action - aktivira se pri pokretanju aplikaciju (*Build*), te daje obavijesti o tome ima li kakvih grešaka u aplikaciji,
- 6. Linija s padajućim izbornicima i tipkama koje služe kao prečac za određene akcije (npr. otvori novi projekt, spremi projekt, tipke za pokretanje i zaustavljanje kompajliranja i sl.).

U Solution Explorer-u mogu se, na slici 3.2, vidjeti neke datoteke koje je sučelje automatski kreiralo stvaranjem nove Windows forme. Obrazac *Properties* predstavlja postavke konfiguracije koje se odnose na cijeli projekt i pohranjene su u *.csproj* datoteci. U te postavke uključuju se neke sigurnosne postavke, postavke za razvoj forme i mnoge druge. Desnim klikom na mapu *Properties* te odabirom opcije *Open* otvara se *Project Designer* gdje se mogu mijenjati postavke otvorenog projekta. Ispod obrasca *Properties* nalazi se obrazac *References* koji sadrži binarnu datoteku koju aplikacija koristi kod pokretanja. Ona sadrži *.dll* datoteku iz biblioteke klasa koju sadrži .NET Framework. Ukoliko je program kreiran na primjeru klase koja je definirana u nekoj drugoj datoteci, prije nego što se on počne kompajlirati mora se staviti referenca na tu datoteku. Inače, svi C# projekti sadrže referencu na *mscorlib.dll* datoteku koja sadrži temelje klasa .NET Framework-a. Otvaranjem novog projekta sučelje također definira dvije datoteke od koje se jedna naziva *Form1.cs*, a druga *Form1.Designer.cs*. Kod koji programer sam piše ovisno o tome koju funkciju želi pridijeliti kojem alatu nalazi se u datoteci *Form1.cs*. U datoteci *Form1.Designer.cs Windows Form Designer* definira izvorni kod koji sadrži sve radnje koje je programer izvršio povlačenjem i ispuštanjem određenih alata na formu, mijenjanjem njihovih svojstva i sl. Ne preporuča se ovu datoteku mijenjati samostalno. Ako se pokrene korisnikova aplikacija pritiskom na tipku F5, kao rezultat će se dobiti prazna forma, tj. standardna Windows forma na kojoj neće biti ni jednog alata. Jedino što se može s tom formom raditi je: povlačiti je po zaslonu računala tako da se drži lijevim klikom miša, da se maksimizira odnosno minimizira (kvadratić u gornjem desnom kutu forme) te da se zatvori (znak x u gornjem desnom kutu forme). Pokretanjem novog Windows projekta samo okruženje je odradilo sav posao i kreiralo glavnu formu aplikacije koja predstavlja početnu točku, tj. osnovu u razvoju bilo koje Windows aplikacije. Znači, aplikacija će se razvijati tako što će se na osnovnu formu, odnosno prozor, dodavati neki elementi sučelja (*Toolbox*), a potom i napisati kod za odgovarajuće događaje forme, odnosno dodati određeni kod svakom elementu koji će se nalaziti na formi. Kod se određenom elementu dodaje dvostrukim klikom na taj element ili se odabire akcija koju će on izvršavati odabirom događaja (*Events*) u izborniku *Properties*. Razvojno okruženje je upravo tako projektirano da na jednostavan, ali efikasan način, dodaje akcije određenim elementima.



Slika 3.3 Prvi dio kalkulatora

Na slici 3.3 vidljivo je da se ime forme preimenovalo u "Kalkulator" te je stavljena ikona koja je preuzeta sa stranice "http://www.iconarchive.com/" koja služi isključivo za takvu primjenu. Izrada kalkulatora započeta je postavljanjem uokvirenog teksta (*Textbox*) koji će služiti za prikaz brojeva koji se žele unositi i za prikaz rezultata određene računske operacije, odnosno služit će kao zaslon (*display*) kalkulatora. Iznad tog uokvirenog teksta nalazi se natpis (*Label*) koji se u ovom slučaju ne vidi, a služi za spremanje međurezultata kod nekih matematičkih funkcija. Veličina uokvirenog teksta i naslova prilagođena je tako da se rezultati na prikladan način vide u cijelosti, odnosno da se vidi ukupan rezultat npr. kod računanja s velikim brojevima. Na formi kod slike 3.3 nalazi se dvadeset tipki (*Button*) kojima je definirana veličina takva da svi budu jednaki. Svakoj tipki dodijeljen je i tekst koji je postavljen na sredinu svake tipke radi bolje preglednosti. Tekstu je promijenjen font u "Times New Roman" te je podebljan (*Bold*) i stavljena mu je veličina slova na 12. Forma sa slike 3.3 sadržava tipke koje imaju funkcije: brisanja cijelog zaslona (C), brisanja zadnjeg unesenog broja (CE) ili brisanja zadnje unesene znamenke (DEL) na zaslonu, aritmetičke matematičke operacije (zbrajanje, oduzimanje, množenje, dijeljenje), tipku za promjenu predznaka (-/+) te sadrži tipke za unos brojeva od 0 do 9 uključujući decimalni zarez i tipku jednako za izračun određenih operacija. S obzirom na te promjene, tipke i dalje

nemaju nikakve akcije tako dugo dok za njih ne napišemo nekakav kod. Tipkama se mogu dodavati različite funkcije klikom na *Events*. Prilikom izrade ove aplikacije uglavnom će se koristiti događaj da tipka reagira na lijevi klik miša, a to se može postići i dvostrukim klikom na tipku kojoj se želi dodijeliti neka funkcija. Dvostrukim klikom na npr. tipku koja je prva postavljena na formu otvorila bi se funkcija u ovom obliku:

```
private void button1_Click(object sender, EventArgs e)
{

}
```

Vidljivo je da je ime ove funkcije "`button1_Click`", što je sučelje samo odredilo, ali se i to može promijeniti tako da se pod karticom *Properties* u rubrici *Name* stavi ime funkcije po želji. Ta promjena se najčešće vrši radi lakšeg snalaženja u kodu aplikacije, no ukoliko to programeru nije potrebno sučelje će samo dodjeljivati imena redoslijedom kojim se tipke stavljaju na formu. Unutar vitičastih zagrada piše se kod s obzirom na to što pojedina tipka radi. Iznad funkcije koja je pozvana klikom na tipku već se nalazi gotov kod, koji je također generiralo razvojno okruženje uz još neke varijable koje su definirane od strane autora, a on izgleda ovako:

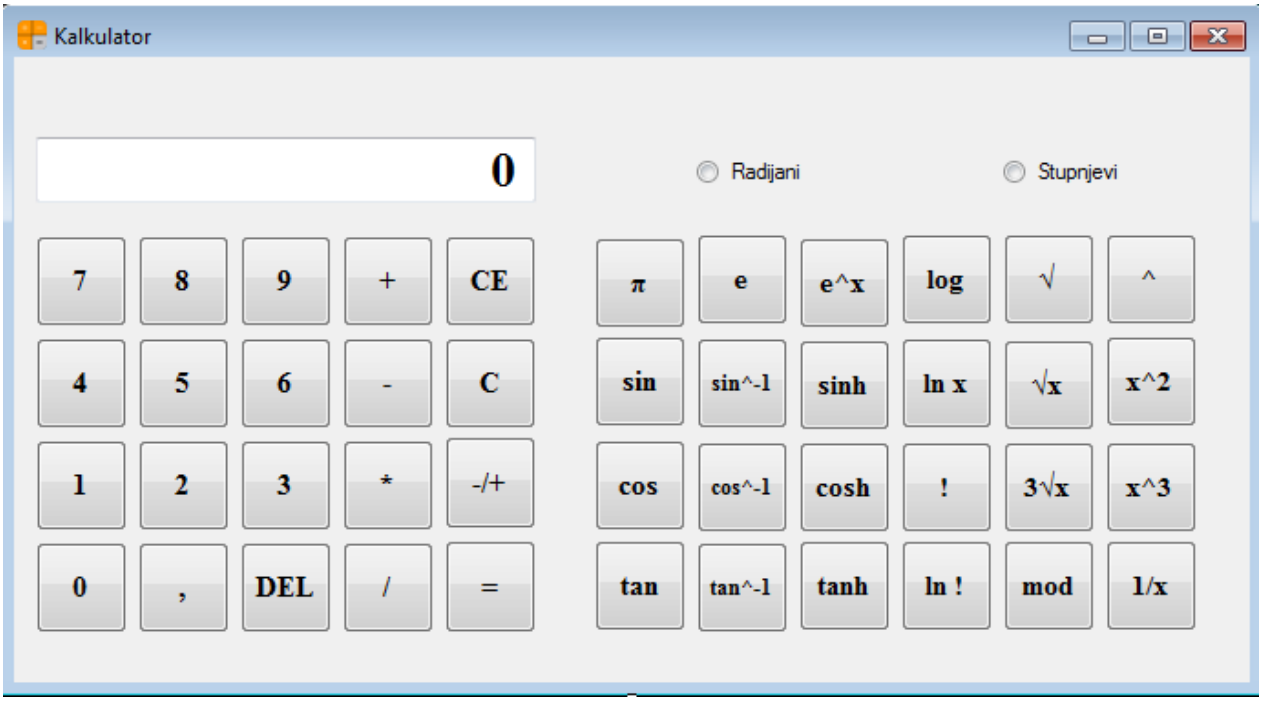
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Globalization;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Kalkulator_Završni_rad
{
    public partial class Form1 : Form
    {
        Double rezultat = 0;
        String operacija = "";
        bool operacija_izvrsena = false;
        double kut;
        double d;
        char dec_point = Convert.ToChar(CultureInfo.CurrentCulture.NumberFormat.Num
        berDecimalSeparator);

        public Form1()
        {
```

```
            InitializeComponent();
            button19.Text=dec_point.ToString();
        }
```

Razvojna okruženja su napravljena s ciljem da omogućuju što jednostavniji razvoj aplikacija i uključuju mnoge funkcionalnosti koje to i omogućavaju te ih treba tako i koristiti. U ovom kodu mogu se primijetiti naredbe koje počinju s "`using`", a na taj se način u kod uključuju odgovarajuće biblioteke s klasama koje one realiziraju, tj. odgovarajući *namespace-ovi*. Osim već definiranih *namespace-ova*, ubačen je `System.Globalization` *namespace* koji sadrži klase koje definiraju informacije vezane uz postavke računala uključujući jezik, državu/regiju, kalendar, formate datuma, redoslijed sortiranja za nizove i sl. Ime glavne forme može se mijenjati kao i ime određenih funkcija. Znakovna varijabla `dec_point` definirana je kao znak (*char*) koji se uzima iz sustava, a služi za pisanje decimalnog zareza. Ona ovisi o tome koje postavke računala koristimo. Na hrvatskom se koristi zarez, a da se postavke prebace na npr. engleski koristila bi se decimalna točka jer je tako standardom propisano te je zbog toga korišten `System.Globalization`. Kod postavki računala na hrvatskom, decimalna točka se može koristiti za odvajanje tisućica, dok se pritiskom na zarez odmah podrazumijeva da je riječ o decimalnom broju. Varijabla `dec_point` pridružena je tipki 19 koji je definiran kao zarez te je prebačena iz znakovne (*char*) varijable u niz (*string*). Ostale varijable koje su tu korištene bit će objašnjene u daljnjem dijelu.



Slika 3.4 Kalkulator sa svim funkcijama

Slika 3.4 prikazuje kalkulator sa svim funkcijama. Kalkulatoru su, s obzirom na prvi dio, pridružene još dvadeset i četiri tipke s različitim matematičkim funkcijama koje će biti objašnjene u daljnjem dijelu te su na formu dodane i dvije “Radio“ tipke koje služe za izračun sinusnih funkcija, ovisno o tome unose li se stupnjevi ili radijani. Font i veličina tipki je postavljena kao i u prvom dijelu prema postupku koji je ranije objašnjen. U prvi plan se stavlja izgled sučelja odnosno forme tako da je prvi stupanj kod izrade neke Windows aplikacije zapravo kreiranje izgleda sučelja aplikacije. Nakon toga dolazi drugi stupanj gdje se za svaki događaj (npr. klik lijevom tipkom miša, dupli klik lijevom tipkom miša, klik desnom tipkom miša i sl.) koji se nadovezuje na kontrole sučelja formiranog u prvom stupnju, generira kod funkcije i povezuje s događajem, tj. funkcije se izvršavaju kada se registrira događaj. Dakle, programiranje se zapravo svodi na pisanje koda funkcija koje se trebaju aktivirati kada se dogodi neki događaj. Zato se ovaj pristup programiranju naziva i programiranje vođeno ili upravljano događajima. Na prvi pogled reklo bi se da se sada programira "lokalno" jer programiramo funkcionalnosti koje se trebaju desiti kao odgovor na neki događaj (najčešće akciju korisnika programa). Ovaj način programiranja smatra se prirodnim jer ako se smatra da korištenje jedne aplikacije ne predstavlja ništa drugo nego pokretanje aplikacije i niza akcija korisnika iza kojih dolazi nekakav "odgovor" programa i tako sve dok se aplikacija ne završi. Jednom riječju korištenje aplikacije je niz akcija

korisnika koje očekuju adekvatan odziv aplikacije. Stoga se i samo kreiranje aplikacije svodi na kreiranje vizualnog izgleda aplikacije, tj. grafičkog korisničkog sučelja upotrebom Windows kontrola i pisanja odgovarajućeg koda kojim se opisuje željeno ponašanje, tj. "odgovor" aplikacije na pojedine akcije korisnika. Ovi koraci su izuzetno važni i veoma često se oni u toku realizacije aplikacije prepliću, tj. stvore se neke forme s nekim kontrolama, a zatim se ispiše odgovarajući kod za događaje kontrola te se nakon toga isti postupak ponovi za neke nove forme i kontrole. Dobra strana je što od same osnovne forme pa nadalje uvijek ima "neka verzija" aplikacije s odgovarajućim korisničkim sučeljem. Posebna pažnja na ovaj se način stavlja na projektiranje, tj. dizajn odnosno izgled sučelja jer on određuje u krajnjoj mjeri kompletan način funkcioniranja cijele aplikacije. Grubo se može reći da se razvoj jedne Windows aplikacije odvija tako što se najprije napravi sučelje koje obuhvaća i popis svih događaja i akcija koje trebaju slijediti te događaje. Nakon toga se u drugoj fazi piše kod za funkcije koje se pozivaju po aktiviranju događaja.

4. Funkcije kalkulatora

U prvom dijelu kalkulatora, kao što je prikazano na slici 3.3, nalaze se samo aritmetičke matematičke funkcije, ali bit će objašnjene i ostale funkcije koje taj dio obuhvaća. Brojevima od 0 do 9 uključujući i točku dodijeljen je isti događaj koji je nazvan `button_Click` te je opisan kodom:

```
private void button_Click(object sender, EventArgs e)
{
        Button button = (Button)sender;
        if ((textBox1.Text == "0") || (operacija_izvrsena))
            textBox1.Text = "";

        operacija_izvrsena = false;

        if (button.Text == dec_point.ToString())
        {
            if(!textBox1.Text.Contains(dec_point.ToString()))
            {
                if (textBox1.Text=="")
                textBox1.Text = "0" + textBox1.Text + button.Text;
                else
                textBox1.Text=textBox1.Text + button.Text;
            }
         }
            else
                textBox1.Text = textBox1.Text + button.Text;
}
```

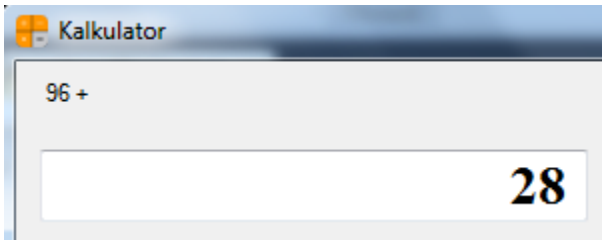
Na početku je vidljiv kod kojim se koristi varijabla `sender`. Ona služi u svrhu da se taj kod dodijeli svim tipkama kojima je pridijeljen događaj `button_Click`. Na početku forme, odnosno koda forme, definirana je varijabla `operacija_izvrsena` koja je tipa `bool` (može biti istina ili laž) i ona služi za ispitivanje da li je neka operacija izvršena ili nije. U ovom slučaju postavljena je na `false` (laž) jer se ne izvršava nikakva operacija. Kod pokretanja aplikacije zaslon kalkulatora prikazuje nulu. Da bi se ta nula uklonila, kada se počne pisati neki broj, odnosno kada se počnu pritiskati određene tipke, korištena je prva naredba `if`. Kod unutar druge naredbe `if` korišten je za pisanje decimalnih brojeva. Tu je korištena varijabla `dec_point` čije smo značenje već ranije objasnili. Ukoliko je broj u kalkulator unesen pomoću tipkovnice računala te je decimalni zarez pisan više puta, računalo će prepoznati da to nije broj i vratit će zaslon na 0 pritiskom na željenu operaciju. Svim aritmetičkim operacijama dodijeljen je događaj `button_aritmeticki` koji je opisan ovim kodom:

```
private void button_aritmeticki(object sender, EventArgs e)
```

```
{
        Button button = (Button)sender;

        operacija = button.Text;
        if (double.TryParse(textBox1.Text, out d))
        {
            rezultat = d;
            operacija_izvrsena = true;
            label1.Text = rezultat + " " + operacija;
        }
        else
            textBox1.Text = "0";
}
```

Na početku, kao i kod prethodne funkcije, vidljivo je da se s varijablom `sender` ponovo pridružuje funkcija svim tipkama koji će se koristiti za aritmetičke operacije (zbrajanje, oduzimanje, množenje i dijeljenje). Varijabla `operacija` tipa string definirana je na početku forme i služi za ispis znaka određene aritmetičke operacije. Ovdje je korištena `TryParse` metoda kod koje su navedena dva uvjeta. Kod prvog uvjeta u kalkulator se unosi niz (*string*) koji može sadržavati brojeve, slova i druge znakove, a drugi uvjet služi za provjeru je li taj niz broj napisan u ispravnom formatu. Ako se unosi broj u pogrešnom formatu (unošenje slova ili drugih znakova) ili izvan granica koje su definirane za tip *double*, doći će do pogreške odnosno neće se uspostaviti "komunikacija" između prvog i drugog uvjeta i morat će se željeni broj unositi ponovno. Ako je broj unesen u ispravnom formatu, varijabla `d` koja je definirana na početku forme tipa `double`, prepoznat će da je unos ispravan te ćemo taj broj spremiti u varijablu `rezultat` koja je također definirana na početku forme. Ova metoda koristit će se kod većine funkcija koje će kalkulator sadržavati, a pri svakoj od tih funkcija njezina zadaća je ista. Ovaj kod služi u svrhu da se na zaslonu kalkulatora ispisuje rezultat te da se u labelu, koja se koristi kao pomoćna varijabla, sprema prvi broj s kojim se želi izvršiti neka aritmetička operacija.



Slika 4.1 Primjer aritmetičke operacije zbrajanja

Na slici 4.1 se može vidjeti jednostavan primjer koji opisuje čemu služi ovaj kod. Vidljivo je da prvi broj koji je unesen jest broj 96 te se on sprema u labelu i vidi se u gornjem lijevom kutu

aplikacije. Zatim se odabire aritmetička operacija kojom se želi računati (u ovom slučaju zbrajanja) pritiskom na odgovarajuću tipku. Pritiskom na tipku 2 kojim je započeto pisanje drugog broja 28, broj 96 automatski se maknuo sa zaslona jer je varijabla `operacija_izvršena` u ovom slučaju postavljena na istina (*true*) odnosno izvršila se određena operacija. Pritiskom na tipku jednako na zaslonu se prikazao rezultat ove operacije koji iznosi 124. Kodom tipke jednako definirane su sve aritmetičke operacije, a on izgleda ovako:

```
private void button_jednako_Click(object sender, EventArgs e)
{
        if(double.TryParse(textBox1.Text, out d))
        {
        switch (operacija)
        {
            case "+":
                textBox1.Text = (rezultat + d) + "";
                break;
            case "-":
                textBox1.Text = (rezultat - d) + "";
                break;
            case "*":
                textBox1.Text = (rezultat * d) + "";
                break;
            case "/":
                textBox1.Text = (rezultat / d) + "";
                break;
            case "mod":
                textBox1.Text = (rezultat % d) + "";
                break;
            case "^":
                textBox1.Text = Math.Pow(rezultat, d) + "";
                break;
            case "√x":
                textBox1.Text = Math.Pow(d, 1/rezultat) + "";
                break;
            default:
                break;
        }
        operacija_izvrsena = false;
        }
            else
            textBox1.Text="0";
}
```

Na početku se koristi `TryParse` metoda za unos broja u ispravnom formatu. Za izvršavanje aritmetičkih operacija korištena je naredba uvjetnog grananja `switch case`. Kod naredbe `switch` vidljivo je se kao uvjet koristi varijabla `operacija` koja je tipa `string` te služi za

odabir operacije koju se želi izvršiti, a s naredbom `case` se bira koja će to operacija biti. Prvi uneseni broj sprema se u varijablu `rezultat` te nakon odabira određene operacije bira se drugi broj koji se sprema u varijablu `d`. Kod ovog bloka naredbi vidljivo je da se nalaze i neke funkcije koje ne spadaju pod aritmetičke, no one će biti objašnjene u daljnjem dijelu. Nakon izvršavanja naredbe `switch case` ponovo je varijabla `operacija_izvrsena` stavljena na `false` iz istog razloga koji je već prije objašnjen. U aplikaciji je korištena i tipka za promjenu predznaka broja kojeg se želi unositi, a on je opisan kodom:

```
private void button11_Click(object sender, EventArgs e)
{
        if (double.TryParse(textBox1.Text, out d))
        {
            if (textBox1.Text == "")
                textBox1.Text = d + "";
            else
                textBox1.Text = -1 * d + "";
        }
        else
            textBox1.Text="0";
}
```

Opet se prvo koristi `TryParse` metoda za odabir broja te je nakon toga tipka definirana pomoću `if` naredbe pod uvjetom da se na zaslonu ispisuje broj s pozitivnim predznakom, a kod pritiska na tipku broj se pomnoži s -1 te promjeni predznak, odnosno postaje negativan. Ova naredba omogućuje i to da se, ako se broj želi vratiti natrag u pozitivni, samo klikne ponovno na tipku i broj se ponovno prebaci u pozitivan. U ovom djelu ostale su još za objasniti funkcije za brisanje zaslona koje su opisane tipkama DEL, CE i C, a ovo je njihov kod:

```
private void button_del_Click(object sender, EventArgs e)
{
     if (textBox1.Text.Length > 0)
     {
     textBox1.Text = textBox1.Text.Remove(textBox1.Text.Length - 1, 1);
     }
     if (textBox1.Text == "")
     {
         textBox1.Text = "0";
     }
}

private void button_CE_Click(object sender, EventArgs e)
{
      textBox1.Text = "0";
}
```

```
private void button_C_Click(object sender, EventArgs e)
{
      textBox1.Text = "0";
      label1.Text = "";
      rezultat = 0;
      operacija_izvrsena = false;
      operacija = "";
}
```

Tipka DEL služi za brisanje zadnjeg broja koji se unosi u kalkulator što kod funkcije `button_del_Click` opisuje prva `if` naredba. Drugi `if` blok naredbi govori da se posljednji broj koji je ostao na zaslonu obriše te ponovno vrati nula na zaslon kao kod otvaranja aplikacije. Tipka CE koja je opisana funkcijom `button_CE_Click` služi da se obriše zadnji broj koji je unesen, odnosno broj koji se u tom trenutku nalazi na zaslonu kalkulatora. Broj koji je prvi upisan ostat će zapisan u labeli i neće se izbrisati. Tipka C koja je opisana funkcijom `button_C_Click` briše sve što je do tog trenutka uneseno u kalkulator. U drugom dijelu kalkulatora većinom se koriste matematičke funkcije koje su već definirane unutar okruženja Visual Studio-a, a naredba za pozivanje istih zove se `Math`. Pomoću ove naredbe mogu se definirati i konstante kao što su: konstanta pi (π) i konstanta e. Kod za ove funkcije izgleda ovako:

```
private void button_pi_Click(object sender, EventArgs e)
{
      textBox1.Text = Math.PI + "";
}
private void button_e_Click(object sender, EventArgs e)
{
      textBox1.Text = Math.E + "";
}
```

Pi je matematička konstanta danas često primjenjivana u matematici i fizici [17]. Broj pi je iracionalan broj što znači da se taj broj ne može prikazati u obliku razlomka čiji je brojnik cijeli broj, a nazivnik prirodan broj. Definira se kao omjer opsega kružnice i promjera kružnice ili kao omjer površine kruga i kvadrata radijusa. Kod aplikacije je vidljivo da se ta konstanta jednostavno poziva pomoću funkcije `Math.PI`, a klikom na tu tipku ta konstanta iznosi 3,14159265358979. Nakon pi, Eulerov broj ili Napierova konstanta e, najvažnija je matematička konstanta. Neke od najčešće korištenih definicija su:

- e je baza prirodnog logaritma $ln(x) = log_e(x)$,

- kao limes niza brojeva: $\lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n, n \in N$,
- e je jedinstveni realni broj takav da je nagib tangente povučene na graf funkcije *f(x)* = e^x u točki (0, 1) jednak 1.

Kod prikazane aplikacije ta konstanta se definira slično kao i konstanta pi naredbom `Math.E`, a ona iznosi 2,71828182845905. Ovisno o ovoj konstanti dodana je još jedna tipka koja je definirana kao e^x, odnosno kao potencija nekog broja na tu konstantu. Naredba koja je tu korištena glasi `Math.Pow`, a kod nje se moraju navoditi dva uvjeta. Kod prvog uvjeta navodi se broj koji se želi potencirati, a to je u ovom slučaju konstanta e. Kod drugog uvjeta navodi se broj potencije, u ovom slučaju to je broj koji se unosi u kalkulator pomoću tipki aplikacije ili tipkovnice računala. Pomoću iste naredbe definirane su i tipke x^2, x^3, 3√x (treći korijen od x), pa se može na nekim od tih primjera prikazati korištenje naredbe `Math.Pow`:

```
private void button_x2_Click(object sender, EventArgs e)
{
        if (double.TryParse(textBox1.Text, out d))
        {
            textBox1.Text = Math.Pow(d, 2) + "";
        }
        else
            textBox1.Text = "0";
}
```

Ovo je funkcija za tipku x^2. Uvjeti naredbe `Math.Pow` nalaze se unutar zagrada te je vidljivo da se kod prvog uvjeta odabire broj po želji pomoću određenih tipki aplikacije ili unosom pomoću tipkovnice računala, a kod drugog uvjeta, koji dolazi iza zareza, bira se broj s kojim se želi potencirati. U ovom slučaju to je broj dva s obzirom da je odabrana takva funkcija ove tipke. Tipka za izračun korijena i tipka 1/x definirane su također na jednostavan način pomoću `TryParse` metode. Kod izračuna korijena koristi se funkcija `Math.Sqrt` unutar čije se zagrade kao uvjet navodi samo izbor broja kojeg se želi korjenovati kao što je već i viđeno u prijašnjim primjerima. Tipka 1/x, gdje je x broj koji se unosi, ne koristi funkciju `Math`, a njegov kod izgleda ovako:

```
private void button_1x_Click(object sender, EventArgs e)
{
        if (double.TryParse(textBox1.Text, out d))
        {
            textBox1.Text = 1 / d + "";
        }
```

```
        else
            textBox1.Text = "0";
}
```

Tipke mod, ^ i √x, koje su prethodno najavljene da će biti objašnjene, nalaze se unutar naredbe `switch case`. Razlog zbog kojeg su pisane pomoću te naredbe jest da se te operacije moraju vršiti između dva broja. Ostatak cjelobrojnog dijeljenja ili mod danas se često koristi u programiranju. Njegov operator se kod programskih jezika najčešće označava kao %. Dijeljenje s ostatkom uči se u trećem razredu osnovne škole, a to je upravo ta funkcija. Npr. ako se broj 27 dijeli s brojem 5 dobit će se 5 i ostatak 2. Ako se to želi zapisati u programskom jeziku onda to izgleda kao 27%5 te se dobiva 2 kao rezultat te naredbe. Kod aplikacije, kod je zapisan unutar `switch case` naredbe na isti način kao i kod aritmetičkih operacija, samo se kao operator koristi %. Tipka ^ je tipka za potenciranje dvaju brojeva koji se odabiru po želji. Korištena je ponovno naredba `Math.Pow` gdje se kao prvi uvjet stavlja varijabla `rezultat` što znači broj koji se prvi unosi, a kao drugi uvjet stavlja se broj s kojim se tu varijablu želi potencirati i on se sprema u varijablu `d`. Tipka √x (n-ti korijen od x) koristi se za izračun takve funkcije. Kod ove tipke prvo se odabire broj n pa zatim broj x. Npr. ako se kao prvi broj uzme broj 3, a kao drugi broj 27 onda će zapis ove operacije izgledati kao $\sqrt[3]{27}$. Rezultat ovog primjera iznosit će 3 jer je 3^3 jednako 27. Za izračun logaritamskih funkcija izdvojene su dvije tipke. Jedna tipka služi za izračun logaritma s bazom 10, a druga za izračun prirodnog logaritma. Logaritam nekog pozitivnog realnog broja *x* u nekoj bazi *b* je broj *y* kojim se treba potencirati bazu da bi dobili zadanu vrijednost *x*, a to se zapisuje na sljedeći način: ako je $x=b^y$ onda je $y=log_b(x)$ [19]. Prirodni logaritam, od prije poznat kao hiperbolički logaritam, logaritam je za bazu *e* gdje je *e* iracionalna konstanta koja se već prije definirala, a označava se s ln [20]. Funkcija prirodnog logaritma može se definirati i kao inverzna funkcija eksponencijalne funkcije, što vodi do identiteta: $e^{ln(x)}=x$ i $ln(e^x)=x$ ako je x > 0. U našoj aplikaciji ove funkcije su prikazane tipkama log i ln x, a njihov kod izgleda ovako:

```
private void button_log_Click(object sender, EventArgs e)
{
        if (double.TryParse(textBox1.Text, out d))
        {
            textBox1.Text = Math.Log10(d) + "";
        }
        else
            textBox1.Text = "0";
}
```

```
private void button_ln_Click(object sender, EventArgs e)
{
        if (double.TryParse(textBox1.Text, out d))
        {
            textBox1.Text = Math.Log(d) + "";
        }
        else
            textBox1.Text = "0";
}
```

Prva funkcija `button_log_Click` služi za izračun logaritama s bazom 10. Vidljivo je da se ponovno koristi naredba `Math` ovaj put s ekstenzijom `Log10` koja jasno govori o čemu se radi u ovoj funkciji. Kod druge funkcije postupak je identičan samo se koristi naredba `Math.Log` koja služi za izračun prirodnih logaritama. Dvije tipke korištene su za izračun faktorijela. Broj n! (n faktorijela) je pozitivan cijeli broj nastao množenjem nekog broja sa svim prethodnim pozitivnim cijelim brojevima [21]. Ako se želi izračunati npr. 4! (četiri faktorijela) onda je to jednako umnošku brojeva $1 \cdot 2 \cdot 3 \cdot 4$ te je 4!=24. U našoj aplikaciji ta tipka je označena s !, a kod izgleda ovako:

```
private void button_fakt_Click(object sender, EventArgs e)
{
        double a = 1;
        int cijeli;
        if (int.TryParse(textBox1.Text, out cijeli))
        {
            if (cijeli > 170)
                textBox1.Text = "Infinity";
            else
            {
                for (int i = 1; i <= cijeli; i++)
                {
                    a = a * i;
                }
                textBox1.Text = a + "";
            }
        }
}
```

Funkcija je opisana jednostavnom `for` petljom koja broji odnosno množi brojeve od 1 pa do broja koji se upiše na zaslon kalkulatora. Na početku su definirane dvije varijable. Varijabli `a` tipa `double` dodijeljena je vrijednost 1 kao početni broj za izračun faktorijela, a druga varijabla tipa `int` nazvana je `cijeli`. Ta varijabla mora biti postavljena kao cjelobrojna (*integer*) jer se izračun faktorijela vrši samo s cijelim brojevima, a ona nam služi za unos cijelog broja pomoću `TryParse` metode. Kalkulator može računati faktorijel najviše do broja 170, a kod unosa broja

puno većeg od 170 kalkulator se zamrzne jer je broj kod takvog izračuna veći od opsega koje varijabla tipa *double* može poprimiti. Tip *double* koristi 8 bajta za obradu podataka [22]. Realni brojevi se u računalu generiraju pomoću bitova gdje se, npr. kod tipa *double*, 1 bit koristi za pohranu predznaka, 11 bitova se koristi za pohranu karakteristike te preostalih 52 bita za pohranu mantise. Karakteristikom od 11 bitova dobivaju se granice tipa *double* gdje je: *2^(+-2^10)=1.7*10^(+-308)*. Pri korištenju realnih brojeva koji su tipa *float* uzima se 4 bita za obradu podataka od čega se 1 bit koristi za predznak, 8 bitova za karakteristiku te preostala 23 za mantisu. Iz karakteristike se ponovno dobivaju granice koje se definiraju kao *2^(+-2^7)=3.4*10^(+-38)*. Za razliku od naše aplikacije koja većinom koristi tip *double* standardni Windows kalkulator može računati u puno većem rasponu. Njegove granice su od 10^-9999 do 10^10000. Iz tog razloga je postavljen uvjet pomoću `if` naredbe koji govori da ako se unosi broj veći od 170 na zaslonu ispiše "*Infinity*", odnosno beskonačno, a ukoliko je taj broj manji da se računa faktorijel prema prijašnjem postupku. Druga tipka korištena je za izračun faktorijela u logaritamskom omjeru pomoću Ramanujanove aproksimacije odnosno približnog izračuna faktorijela. Formula koja je korištena glasi: $ln(n!) \approx n \cdot ln(n) - n + \frac{1}{6} ln\left(n\left(1 + 4n(1 + 2n)\right)\right) + \frac{1}{2} ln(\pi)$ [23]. Kod za ovu tipku opisan je u funkciji `button_lnfakt_Click`:

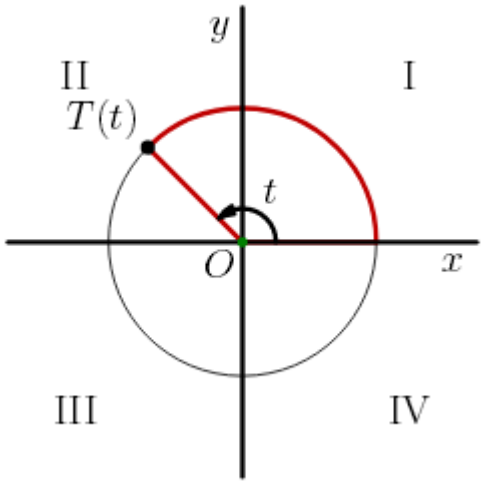
```
private void button_lnfakt_Click(object sender, EventArgs e)
{
        double logfakt;
        if (double.TryParse(textBox1.Text, out d))
        {
            logfakt = d * Math.Log(d) - d + 0.1666 * Math.Log(d * (1 +
                      4 * d * (1 + 2 * d))) + 0.5 * Math.Log(Math.PI);
            textBox1.Text = logfakt + "";
        }
        else
            textBox1.Text = "0";

}
```

Na početku je definirana varijabla `logfakt` te je u nju spremljena formula za izračun Ramanujanove aproksimacije koja je prije definirana. Pomoću `TryParse` metode unosi se broj s kojim se želi računati te se on sprema u varijablu `d`. U formuli su korištene potrebne matematičke funkcije `Math.Log` i `Math.PI`.

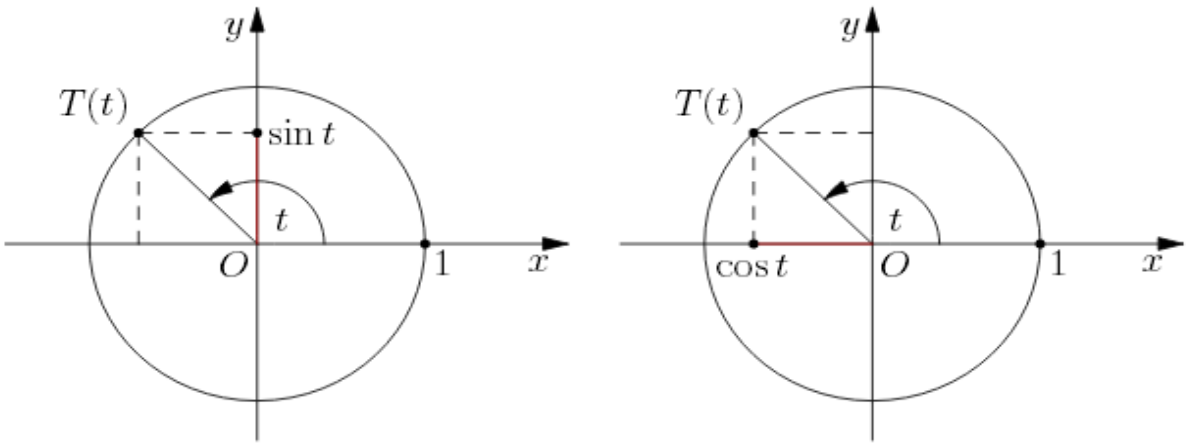
Za izračun sinusnih funkcija korišteno je devet tipki. Pomoću njih definirane su trigonometrijske funkcije sinus, kosinus, tangens te njihove inverzne funkcije dok su tri tipke iskorištene za

izračun hiperboličnih trigonometrijskih funkcija sinusa, kosinusa i tangensa. Mjerenje kutova može se objasniti na primjeru jedinične kružnice [24]. Može se pretpostaviti da se po kružnici jediničnog radijusa kut pomakne za vrijednost *t* u smjeru suprotnom od kazaljke na satu.



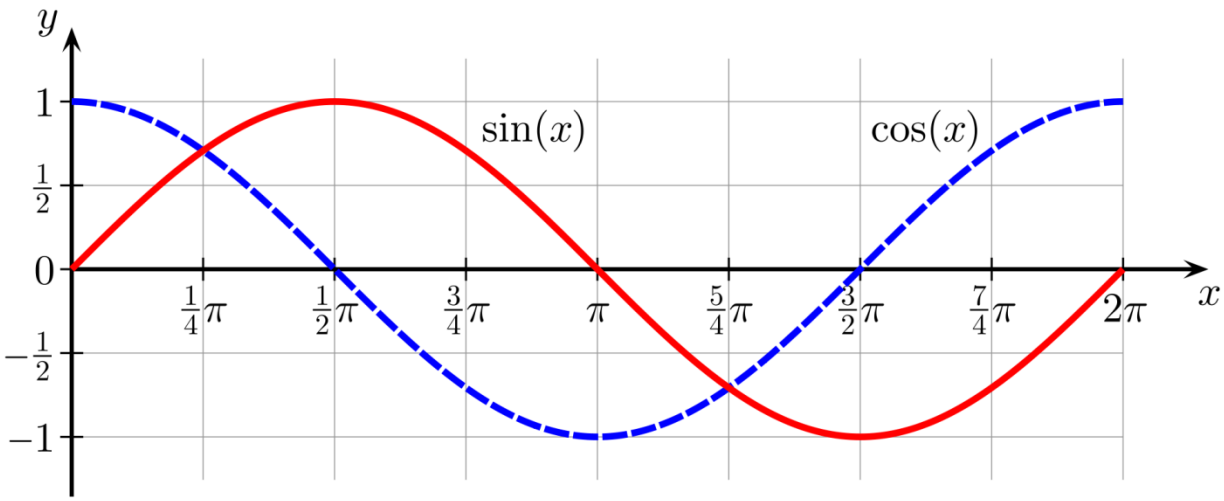
Slika 4.2 Definicija kuta pomoću jedinične kružnice [24]

Kut *t* u radijanima po definiciji je jednak duljini prijeđenog luka. Ako se promatra cijela kružnica duljina prijeđenog luka iznosit će 2π radijana (opseg jedinične kružnice). Osim u radijanima kutovi se mogu mjeriti i u stupnjevima te u ovom slučaju puni krug odgovara kutu od 360°. Funkcije sinus i kosinus također se mogu definirati na brojevnoj kružnici gdje je funkcija sinus ordinata, a funkcija kosinus apscisa točke *T(t)* na brojevnoj kružnici.



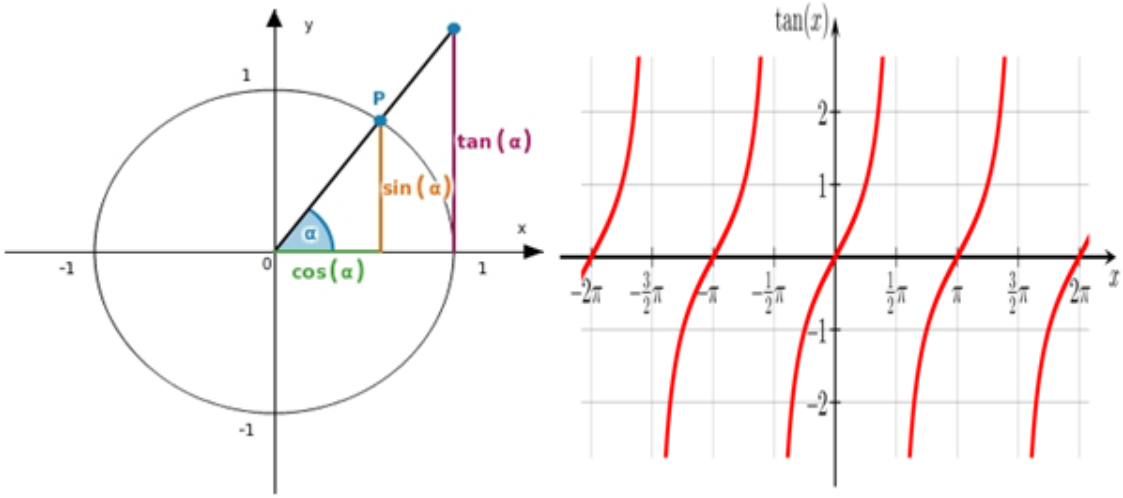
Slika 4.3 Definicije funkcija sinus i kosinus na brojevnoj kružnici [24]

Apsolutna vrijednost obje funkcije mora biti manja od radijusa jedinične kružnice pa se vrijednosti tih funkcija kreću u intervalima: $-1 \leq sinx \leq 1$ i $-1 \leq cosx \leq 1$, slika 4.3.



Slika 4.4 Konstrukcija grafa sinus i kosinus [25]

Funkciju tangens se definira kao omjer funkcija sinus i kosinus: $\tan x = \frac{\sin x}{\cos x}$. Funkcija se može, kao i prethodne dvije, definirati pomoću jedinične kružnice ili grafom, slika 4.5.



Slika 4.5 Prikaz funkcije tangens [24]

Iz slike 4.5 vidljivo je da tangens nije definiran u nul-točkama: $x_0 = \frac{\pi}{2} + k\pi, k \in Z$. Funkcija je neparna što se može provjeriti iz omjera: $\tan(-x) = \frac{\sin(-x)}{\cos(-x)} = \frac{-\sin x}{\cos x} = -\tan x$.

Kod naše aplikacije, pored tipki koje omogućavaju izračun takvih funkcija, ubačene su i dvije "Radio" tipke, odnosno tipke koje imaju funkciju biranja ovisno o tome želi li se kut računati u stupnjevima ili radijanima. Tipke se mogu vidjeti na slici 3.4 u obliku dva kružića gdje se onda bira između mjere kuta u stupnjevima ili radijanima. Prilikom prvog pokretanja aplikacije označeno je da `radiobutton1` bude pritisnut te se kut automatski računa u radijanima. Kut izražen u radijanima može se pretvoriti u stupnjeve pomoću formule: $\alpha(stupnjevi) = \frac{360}{2\pi} \cdot \alpha(radijani)$. Kod za jednu takvu funkciju izgleda ovako:

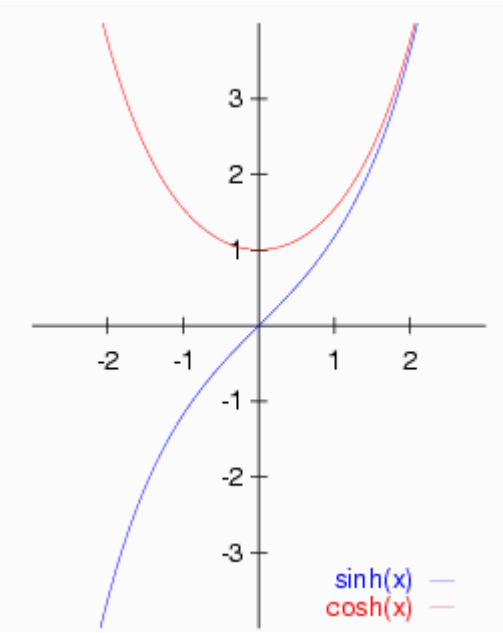
```
private void button_sin_Click(object sender, EventArgs e)
{
        if (double.TryParse(textBox1.Text, out d))
        {
            if (radioButton1.Checked)
            {
                textBox1.Text = Math.Sin(d) + "";
            }
            else if (radioButton2.Checked)
            {
                kut = d * (Math.PI / 180);
                textBox1.Text = Math.Sin(kut) + "";
            }
        }
        else
            textBox1.Text = "0";
}
```

Matematičke funkcije za izračun trigonometrijskih i hiperboličkih sinusnih funkcija u okruženju Visual Studio definirane su tako da se kut računa u radijanima. Mjera kuta u stupnjevima više se koristi u praksi te su iz tog razloga ubačene dvije "Radio" tipke da se može birati mjera kuta po želji. Kao i kod prijašnjih funkcija koristi se `TryParse` metoda za unos broja koji se sprema u varijablu `d`. Kod `if` naredbe za računanje mjere kuta u radijanima kao uvjet stavlja se da `radiobutton1` (tipka za odabir mjere kuta u radijanima) mora biti pritisnut te se koristi naredba `Math.Sin` koja služi za izračun sinusa nekog kuta. Kod `if` naredbe za izračun mjere kuta u stupnjevima kao uvjet stavlja se da `radiobutton2` (tipka za odabir mjere kuta u stupnjevima) mora biti pritisnut te je vidljivo da se pojavljuje varijabla kut koja je definirana na početku forme, a ona je tipa double. Ako se želi unositi mjera kuta u stupnjevima mora se broj koji se unosi prije uvrštavanja u sinus funkciju pomnožiti s $\frac{\pi}{180}$ te taj rezultat spremiti u varijablu `kut`. Nakon što se to napravi koristi se naredba `Math.Sin` te se kao uvjet navodi varijabla `kut`. Kod izračuna dobiva se mjera kuta u stupnjevima. Kod za funkcije kosinus i tangens je identičan, samo umjesto

naredbe `Math.Sin`, kao u ovom slučaju, koriste se naredbe `Math.Cos`, odnosno `Math.Tan`. Za inverzne trigonometrijske funkcije koriste se također tri tipke koje računaju inverzene funkcije prethodno objašnjenih funkcija sinus, kosinus i tangens. Ove funkcije koriste se za izračun kuta ukoliko je poznata vrijednost njegovog npr. sinusa, a nazivaju se još i arkus funkcije od latinske riječi *arcus* što znači luk ili kut. Tako se naznačene funkcije znaju zapisivati i kao arcsin, arccos i arctg. Definiciju ovih funkcija možemo objasniti na jednostavnom primjeru. Ako izraz y = sin(x) znači: "x je mjera kuta čiji je sinus jednak y" onda izraz y = arcsin x znači: "y je arkus sinus onog kuta čiji je sinus jednak x" [26]. Kod za jednu takvu funkciju izgleda ovako:

```
private void button_cos1_Click(object sender, EventArgs e)
{
        if (double.TryParse(textBox1.Text, out d))
        {
             if (radioButton1.Checked)
             {
                 textBox1.Text = Math.Acos(d) + "";
             }
             else if (radioButton2.Checked)
             {
                 kut = Math.Acos(d);
                 textBox1.Text = (kut * (180 / Math.PI)) + "";
             }
         }
         else
             textBox1.Text = "0";
}
```

Postupak je sličan prethodnom. Kod pritiska na tipku cos^-1 te kad je označena tipka `radiobutton1` računa se inverzna funkcija, u ovom slučaju, funkcije kosinus pomoću naredbe `Math.Acos` i dobiva se mjera u radijanima. Ukoliko je pritisnut `radiobutton2` računa se inverzna funkcija u stupnjevima, ali kod ovog slučaja prvo se računa inverz kuta u radijanima te se ta vrijednost sprema u varijablu `kut`. Nakon toga ta vrijednost se mora pomnožiti s $\frac{180}{\pi}$ te se dobiva mjera kuta u stupnjevima. Kôd za funkcije sinusa i tangensa identičan je samo su korištene naredbe `Math.Asin` i `Math.Atan`. Ostale su još za objasniti hiperbolne trigonometrijske funkcije. Hiperbolne funkcije definiramo pomoću eksponencijalne funkcije e^x [27]. Veze između hiperbolnih funkcija slične su vezama između trigonometrijskih funkcija. Sinus hiperbolni je funkcija: $\sinh x = \frac{e^x - e^{-x}}{2}$, a kosinus hiperbolni funkcija: $\cosh x = \frac{e^x + e^{-x}}{2}$.



Slika 4.6 Hiperbolne funkcije sinus i kosinus

Na slici 4.6 vidimo da je sinus hiperbolni neparna, a kosinus hiperbolni parna funkcija te da je sinus hiperbolni strogo rastuća funkcija. Kosinus hiperbolni se još zove i lančanica jer lanac obješen o dvije točke u gravitacijskom polju zauzme oblik dijela te krivulje. Slično kao kod trigonometrijskih funkcija, tangens hiperbolni definiramo kao omjer sinusa i kosinusa: $\tanh x = \frac{\sinh x}{\cosh x}$ iz čega proizlazi: $\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. Kôd za jednu takvu funkciju u našoj aplikaciji izgleda ovako:

```
private void button_tanh_Click(object sender, EventArgs e)
{
        if (double.TryParse(textBox1.Text, out d))
        {
            if (radioButton1.Checked)
            {
                textBox1.Text = Math.Tanh(d) + "";
            }
            else if (radioButton2.Checked)
            {
                kut = d * (Math.PI / 180);
                textBox1.Text = Math.Tanh(kut) + "";
            }
        }
        else
            textBox1.Text = "0";
}
```

Kod je gotovo identičan kao i kod računanja funkcija sinusa, kosinusa i tangensa. Pri pritisku na tipku tanh te kad je označena tipka `radiobutton1` računa se, u ovom slučaju, hiperbolična vrijednost funkcije tangens u radijanima pomoću naredbe `Math.Tanh`. Kod pritiska na tipku `radiobutton2` prvo se, kao i kod trigonometrijskih funkcija, broj pomnoži s $\frac{\pi}{180}$ i sprema u varijablu `kut`, a nakon toga se koristi naredba `Math.Tanh` s uvjetom te varijable. Kôd za hiperbolne funkcije sinusa i kosinusa je identičan samo se koriste naredbe `Math.Sinh` i `Math.Cosh`.

5. Zaključak

U današnje vrijeme razvoj aplikacija postao je svakodnevan posao. Razvojna okruženja projektirana su tako da izrada aplikacija, ljudima koji se bave time ili se žele baviti, čine posao jednostavnim i zanimljivim. Aplikacija Kalkulator izrađena je u okruženju Microsoft Visual Studio u obliku Windows forme u programskom jeziku C#. C# je vrlo moćan alat koji se, uz Javu i C++, sve više koristi i postaje jedan od vodećih predstavnika programskih jezika. Pokretanjem novog projekta razvojno okruženje automatski dodaje okvir forme na kojoj se izrađuje aplikacija. U projektu su već definirane biblioteke s klasama koje programeru omogućuju što jednostavniji razvoj aplikacija. Aplikacija je podijeljena u dva dijela radi lakšeg objašnjavanja funkcija koje ona sadrži. Aplikaciji je dodijeljeno ime Kalkulator te pridružena ikona koja je primjerena njegovim zadaćama. U prvom dijelu aplikacije na formu je postavljen zaslon kalkulatora te tipke za: odabir brojeva od 0 do 9, odabir decimalnog zareza te aritmetičkih operacija (zbrajanje, oduzimanje, množenje i dijeljenje), promjenu predznaka i tipke za brisanje te tipka jednako. Drugi dio kalkulatora sadrži tipke s različitim funkcijama od kojih su neke: tipke za trigonometrijske sinusne i hiperbolične sinusne funkcije, za izračun logaritama, potencija, faktorijela i sl. Aplikaciji su dodane i dvije tipke koje služe, kod izračuna sinusnih funkcija, za odabir mjere kuta u radijanima ili stupnjevima. Pri pisanju koda za određene tipke uglavnom se koristi tip double što omogućava da kalkulator računa u granicama od *1.7*10^-308* do *1.7*10^+308* što je za potrebe ove aplikacije sasvim dovoljno. Aplikacija je vrlo pregledna i jednostavna za korištenje te može služiti ljudima kod učenja te kod nailaska na neke jednostavnije matematičke probleme.

U Varaždinu 2.10.2017.

Sveučilište
Sjever

SVEUČILIŠTE
SJEVER

IZJAVA O AUTORSTVU
I
SUGLASNOST ZA JAVNU OBJAVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, SAŠA RUŽIĆ *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom IZRADA KALKULATORA U C# *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)

Saša Ružić
(vlastoručni potpis)

Sukladno Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Ja, SAŠA RUŽIĆ *(ime i prezime)* neopozivo izjavljujem da sam suglasan/na s javnom objavom završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom IZRADA KALKULATORA U C# *(upisati naslov)* čiji sam autor/ica.

Student/ica:
(upisati ime i prezime)

Saša Ružić
(vlastoručni potpis)

6. Literatura

[1] http://www.gimnazija-klasicna-zg.skole.hr/informatika/Programski_jezici_i_osnove_programiranja.pdf - Programski jezici i osnove programiranja.pdf, dostupno 10.09.2017.

[2] I. Griffiths, M. Adams i J. Liberty: prijevod: M. Lisjak i D. Breznik: Programiranje C# 4.0, Dobar Plan, Zagreb, 2011

[3] http://web.studenti.math.pmf.unizg.hr/~bozana/povijest.html, dostupno 09.09.2017.

[4] https://tesla.carnet.hr/pluginfile.php/21710/mod_resource/content/2/COURSE_2666625_M/my_files/Sadrzaj/Poglavlje%202/2_2_4%20Programski_jezik.htm, dostupno 09.09.2017.

[5] https://stefanpeka121.wordpress.com/, dostupno 09.09.2017.

[6] https://sr.wikipedia.org/sr-el/ - Peta generacija programskog jezika, dostupno 09.09.2017.

[7] https://sites.google.com/site/sandasutalo/oosnove-programiranja/jezicni-prevoditelji/interpreter, dostupno 10.09.2017.

[8] https://sites.google.com/site/sandasutalo/oosnove-programiranja/jezicni-prevoditelji/kompajler, dostupno 10.09.2017.

[9] https://www.scribd.com/doc/56472501/Kompajleri-i-interpreteri, dostupno 10.09.2017.

[10] http://skolakoda.org/imperativno-vs-funkcionalno-programiranje, dostupno 10.09.2017.

[11] https://bhrnjica.net/2006/11/07/proceduralno-programiranje/, dostupno 11.09.2017.

[12] https://svetinformatikeiprogramiranja.files.wordpress.com/2013/09/osnoveprogramskogjezikac-2.pdf - Osnove programskog jezika C#.NET.pdf, dostupno 11.09.2017.

[13] D. Radovčić: Android aplikacija „Rescue Team Tempus“, završni rad, Veleučilište u Rijeci, 2014.

[14] http://digis.edu.rs/pluginfile.php/1026/mod_resource/content/1/C%23 – C# programski jezik – zuov kurs.pdf, dostupno 11.09.2017.

[15] https://bs.wikipedia.org/wiki/Microsoft_Visual_Studio, dostupno 11.09.2017.

[16] Z. Ćirović, I. Dunđerski: Tehnike vizuelnog programiranja C#, Viša elektrotehnička škola, Beograd, 2005.

[17] https://bs.wikipedia.org/wiki/Java_(programski_jezik), dostupno 19.9.2017.

[18] https://bib.irb.hr/datoteka/708816.Matematicke_konstante_-_1.dio.pdf, dostupno 12.09.2017.

[19] https://hr.wikipedia.org/wiki/Logaritam, dostupno 12.09.2017.

[20] https://bs.wikipedia.org/wiki/Prirodni_logaritam, dostupno 12.09.2017.

[21] http://www.moje-instrukcije.com/index.php?option=com_content&view=article&id=3626:faktorijela&catid=174&Itemid=149, dostupno 12.09.2017.

[22] https://en.wikipedia.org/wiki/Double-precision_floating-point_format, dostupno 12.09.2017.

[23] https://moodle.vz.unin.hr/ - Programski jezici i algoritmi, Predavanja, 04_pja.ppt, dostupno 12.09.2017.

[24] http://www.phy.pmf.unizg.hr/~mbacani/Fizika/OF1/01_trigonometrija.pdf, dostupno 12.09.2017.

[25] https://math.stackexchange.com/questions/367938/symmetry-properties-of-sin-and-cos-why-does-cos-left-frac3-pi2-x, dostupno 12.09.2017.

[26] http://alas.matf.bg.ac.rs/~brankica/sadrzaj/naslov8.html, dostupno 12.09.2017.

[27] http://lavica.fesb.hr/mat1/predavanja/node97.html, dostupno 12.09.2017.

Popis slika

Slika 2.1 Izvršavanje procedura str. 9

Slika 2.2 Razdvajanje glavnog programa na procedure str. 10

Tablica 3.1 Razvoj programskog jezika C# str. 14

Slika 3.1 Otvaranje novog projekta str. 16

Slika 3.2 Razvojno okruženje za izradu aplikacije str. 17

Slika 3.3 Prvi dio kalkulatora str. 19

Slika 3.4 Kalkulator sa svim funkcijama str. 22

Slika 4.1 Primjer aritmetičke operacije zbrajanja str. 25

Slika 4.2 Definicija kuta pomoću jedinične kružnice str. 33 [24]

Slika 4.3 Definicije funkcija sinus i kosinus na brojevnoj kružnici str. 33 [24]

Slika 4.4 Konstrukcija grafa sinus i kosinus str. 34 [25]

Slika 4.5 Prikaz funkcije tangens str. 34 [24]

Slika 4.6 Hiperbolne funkcije sinus i kosinus str. 37 [27]