SVEUČILIŠTE JOSIPA JURJA SROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET
Sveučilišni preddiplomski studij računarstva

PARSIRANJE MATEMATIČKIH IZRAZA

Završni rad

Božidar Patača

Osijek, 2010. Godina

SADRŽAJ

1. UVOD

Cilj ovog rada je razviti algoritam za parsiranje matematičkih izraza. Pojam „parsiranje“ dolazi od engleske riječi „parse“ što znači analizirati, obraditi, raščlaniti. Ulaz u program je niz znakova u kojem je sadržan određeni matematički izraz, a izlaz je rezultat toga izraza. Niz znakova koji unosi korisnik pomoću tipkovnice potrebno je normirati pomoću funkcije normiraj. Ta funkcija izbaci sve razmake te pojednostavni zapis pojedinih matematičkih funkcija. Matematički izraz može sadržavati i varijable (x, y, z), ali njihova vrijednost mora biti zadana prije pokretanja programa. Ukoliko korisnik unese neispravan matematički izraz (neparan broj zagrada, greška u zapisu ili domeni matematičkih funkcija itd.) program treba ispisat o kakvoj greški se radi te prekinuti s radom. To se vrši pozivom funkcije kraj. Ključna funkcija u ovom programu je, kao što se to može zaključiti iz njezinog imena, funkcija racunaj. Ona će imati tri implementacije a samim time i tri naziva, racunaj_rekurzija, racunaj_stog i racunaj_stog1. U funkciji racunaj_stog stog će biti realiziran pomoću povezanog popisa, a u funkciji racunaj_stog1 pomoću niza. Dakle, funkcija racunaj biti će realizirana kao rekurzivna funkcija i kao funkcija koja koristi stog. Rekurzivna funkcija je ona koja sama sebe poziva. Stog je struktura podataka iz koje se podatci mogu uzimati samo obrnutim redom nego što su u nju stavljeni, često se slikovito prikazuje kao posuda. Postoji mnogo detalja o kojima treba voditi računa pri realizaciji ovakvog algoritma. Prije svega treba provjeriti da li je uneseni izraz matematički ispravan, zatim pri analizi niza treba voditi računa o prioritetu operatora. Uključivanje biblioteke math.h omogoćava nam korištenje elementarnih funkcija kao što su sinus, kosinus, prirodni logaritam idt. Algoritam koji koristi stog trebao bi imati manje vrijeme izvršavanja od onoga koji koristi rekurzivnu funkciju, a to će biti ispitano mjerenjem i uspoređivanjem vremena izvršenja. Osim toga, ovaj rad će pokazati kako je bolje realizirazi stog, pomoću povezanog popisa ili pomoću niza podataka.

2. POMOĆNE FUNKCIJE

Najvažnija funkcija u ovom radu je funkcija racunaj, sve ostale funkcije su sporedne i vrlo jednostavne za napisati. Funkcije normiraj priprema uneseni izraz za funkciju racunaj, funkcija kraj poziva se unutar funkcije racunaj ukoliko dođe do greške prilikom izvršavanja programa, a main funkcija zapravo samo poziva funkciju racunaj. U ovom radu pojavljuju se još dvije gotovo iste pomoćne funkcije, to je funkcija formiraj_stog i formiraj_stog1. Budući da one dolaze u „paketu“ sa funkcijama racunaj_stog i racunaj_stog1, biti će obrađene u četvrtom poglavlju gdje je riječ upravo o tim funkcijama.

2.1. Funkcija kraj

Uloga ove funkcije je obradu grešaka. Ona obavještava korisnika da se dogodila greška te o kakvoj je greški riječ. To radi ispisivanjem odgovarajuće poruke pomoću funkcije printf. Da bismo mogli koristiti tu funkciju moramo uljučiti biblioteku stdio.h (eng. standard input output) koja sadrži funkcije za unos i ispis podataka. Budući da funkcija kraj obrađuje samo pet grešaka, njezin parametar je varijabla *zasto* tipa int koja poprima vrijednosti od 1 do 5 (svaka vrijednost označava jednu vrstu greške). Varijabla ima takvo ime zato što nam ovisno o njenoj vrijednosti funkcija daje do znanja zašto je došlo do pogreške. Hoće li funkcija „kraj“ biti pozvana ili neće ispituje se u uvijetu if naredbe.

U ovoj funkciji obrađuje se pet vrsta grešaka od kojih se četiri mogu registrirati prilikom izvođenja funkcije racunaj (napomena: kada u nastavku teksta bude pisalo samo „racunaj“ to se odnosi na sve tri implementacije te funkcije, racunaj_rekurzija i racunaj_stog i racunaj_stog1). To su:

1) nejednakost u broju otvorenih i zatvorenih zagrada (ili se zatvorena zagrada pojavila prije otvorene)
2) neodređen izraz, ukoliko kod oparacije dijeljanja oba operanda imaju vrijednost 0.
3) grešla u domeni funkcije prirodni logaritam, npr. log(0)
4) greška u domeni funkcije drugi korijen, npr. $\sqrt{-3}$
5) greška u otvaranju datoteke

Poziv funkcije kraj(1) može biti izvršen u bilo kojoj iteraciji for petlje funkcije racunaj u kojoj se vrši kontrola broja zagrada, te nakon nje.
Primjer poziva:

```
if (brzag!=0) kraj( 1 );
```

kraj(2) se izvršava ukoliko se ukoliko i je potrebno izvršiti operaciju djeljenja, a djeljenik i djelitelj imaju vrijednost 0. Primjer:

```
if (lij==0 && des==0) kraj( 2 );
```

Pozivi kraj(3) i kraj(4) uključeni u samu implementaciju funkcije prirodni logaritam i funkcije drugi korijen. Primjer poziva funkcije kraj prilikom greške u domeni funkcije drugi korijen:

```
if(test>=0) return sqrt( test );
else kraj(4);
```

Varijabla test predstavlja argument funkcije korijen u svrhu testiranja istog.
Greška pod rednim brojem 5 može se pojaviti zato što će se koristiti datoteka za pohranu rezultata određenog matematičkog izraza za određen raspon vrijednosti neke od varijabli.
Na osnovu tih vrijednosti crtat će se graf funkcije pomoću MS excel-a. Poziv kraj(5) vrši se u glavnoj funkciji (funkciji main).
Primjer:

```
if(f==NULL) kraj(5);
```

Varijabla f predstavlja datoteku rezultati.txt i ukoliko je otvaranje navedene datoteke bilo neuspješno ona poprima vrijednost NULL.
Funkcija kraj je veoma korisna zato što se vrlo lako može dogoditi da korisnik unese matematički izraz koji sadrži jednu od prve četiri greške na koje ova funkcija upozorava.
Na petu grešku korisnik nema nikakav utjecaj i da nema ove funkcije vrlo teško bi ju mogao registrirati. Naredba exit(0) označava izlaz is programa, odnosno prekid rada programa. Nakon pojave bilo koje od navedenih pet grešaka korisnik mora iznova pokrenuti program te unjeti matematički izraz.

2.2. Funkcija normiraj

Matematički izraz koji unese korisnik može izgledati npr. ovako:

sin (x) - (20+ y) / 17 + 2

Kao što vidimo ovaj izraz je matematički ispravan ali sadrži puno beskorisnih razmaka i tabulatora koje je potrebno ukloniti kako bi izraz bio pregledniji i jednostavniji za obradu. Svaki uneseni matematički izraz potrebno je najprije u cijelosti analizirati znak po znak, a potom i po dijelovima pomoću rekurzivnih poziva. Kada se ne bi uklonili razmaci i tabulatori program bi samo nepotrebno gubio vrijeme na njih.

Radi jednostavnije analize niza, imena svih korištenih funkcija zamijene se s jednim velikim slovom po slijedećem principu:

Tablica 2.1. *zapis elementarnih funkcija nakon normiranja*

nenormirano	normirano
sin(x)	S(x)
cos(x)	C(x)
exp(x)	E(x)
log(x)	L(x)
tan(x)	T(x)
sqrt(x)	K(x)
pow(x,y)	P(x,y)

Izraz s početka ovog potpoglavlja nakon normiranja izgledao bi ovako:

sin (x)-(20+ y) /17+2

Funkciji se kao parametar predaje pokazivač na prvi član ulaznog niza kojeg treba normirati.

Ona od njega napravi normirani niz s koji se predaje kao parametar funkciji racunaj pri njezinom pozivu. Da bi to bilo moguće varijabla s mora biti deklarirana kao globalna kako bi bila vidljiva u svim funkcijama. Funkcija normiraj radi tako da određeni znak iz ulaznog niza kopira ili ne kopira u niz s. Sastoji se od jedne složene for petlje. Ako se pri analizi ulaznog niza naiđe na razmak ili tabulator oni se jednostavno preskaču te se prelazi na slijedeću iteraciju petlje pomoću naredbe continue. Ukoliko se naiđe na uobičajene znakove kao što su '(', '),' 'x', 'y', 'z', '+', '-', '*', '/', '.', ',' ili na neku brojčanu konstantu, oni se jednostavno kopiraju u niz s. Poseban slučaj su funkcije. Kod njih se prvo slovo imena funkcije kopira u niz s, ali se predhodno pomoću funkcije toupper pretvori u veliko slovo. Ostali slova u imenu funkcije se preskaču, odnosno ne kopiraju se u niz s. Ako se primjerice u ulaznom nizu pojavi znak 'e', mi znamo da su slijedeća dva znaka 'x' i 'p' te da se radi o eksponencijalnoj funkciji. Dakle, potrebno je samo slovo 'e' pretvoriti u veliko, a preostala slova preskočiti. Problem nastaje pri pojavi funkcija sinus i drugi korijen jer njihova imena počinju s istim imenom (koriste se funkcije sin i sqrt iz zaglavne datoteke math.h). U tom slučaju nije dovoljno analizirati samo prvo slovo u imenu funkcije već je potrebno provjeriti i drugo. Ako je drugo slovo 'i', znamo da je riječ o funkciji sinus te u niz s kopiramo slovo 'S' a zadnje slovo u imenu funkcije preskočimo. Ako je pak drugo slovo 'q', znamo da se radi o sunkciji drugi korijen te u niz s kopiramo slovi 'K' a preostala dva slova preskočimo.

Nakon poziva ove funkcije imamo niz s koji je spreman da bude predan kao parametar funkciji racunaj.

2.3. Funkcija main

Main funkcija je glavna funkcija koju svaki C program mora imati. Ona se prva izvršava i poziva sve ostale funkcije.

U ovoj funkciji se unosi matemetički izraz i sprema se u ulazni niz ul. Zatim se poziva funkcija normiraj koja od nenormiranog ulaznog niza napravi normirani niz s koji se predaje funkciji racunaj pri njezinu pozivu koji se također vrši u glavnoj funkciji. Budući da je niz s deklariran kao globalna varijabla, vidi se i u main funkciji. Nakon što funkcija racunaj odradi svoje i vrati rezultat u varijablu rez tipa float, vrši se ispis toga rezultata pomoću funkcije printf. Taj rezultat se ispisuje na zaslon ali i u datoteku rezultati.txt.

Može se realizirati i for petlja kako bi se računale vrijednosti određenog matematičkog izraza za određen raspon neke od varijabli. Ova for petlja ispisuje na ekran i u datoteku rezultati.txt rezultate unesenog matematičkog izraza kada se neka od varijabli mijenja u određenom rasponu vrijednosti. Ispiše se i vrijednost varijable za koju je pojedini rezultat dobiven. Ova for petlja ima veliku ulogu i kod mjerenja vremena izvršenja funkcije racunaj. Ako bismo željeli da funkcija racunaj ima vrijeme izvršenja veće od nula milisekundi, trebali bismo unijeti veoma dugačak i složen matematički izraz što zahtijeva mnogo vremena. Jednostavnije je da se neki manje složen izraz izračuna više puta, što se može ostvariti korištenjem for petlje.

Za mjerenje vremena se koristi funkcija clock iz zaglavne datoteke time.h. Parametri ove funkcije su varijable t1 i t2 tipa time_t. Prije početka mjerenja vremena vrši se poziv t1=clock() što pokreće brojanje vremena, a ne kraju se poziva t2=clock() što zaustavlja brojanje vremena.

Razlika između ta dva vremenska trenutka sprema se u varijablu vrijeme tipa double. Vrijeme se izražava u milisekundama. Za potrebe programa uključene su slijedeće zaglavne datoteke iz standardne biblioteke programskog jezika C++:

1) stdio.h - sadrži funkcije za unos i ispis podataka
2) string.h - sadrži konstante, definicije funkcija i tipova podataka koje se koriste za rukovanje stringovima i funkcije za rukovanje promjenjivom memorijom
3) stdlib.h – sadrži funkcije za alokaciju memorije, kontrolu procesa itd.
4) ctype.h – sadrži funkcije za klasifikaciju znakovnih podataka.
5) math.h – sadrži brojne matematičke funkcije
6) time.h – sadrže funkcije za rad s vremenom i datumom (time and date functions).

3. PARSIRANJE REKURZIVNOM FUNKCIJOM

3.1. Rekurzivna funkcija

Prije nego što počnem pisati o realizaciji parsera matematičkih izraza pomoću rekurzivne funkcije, potrebno je objasniti kako radi rekurzivna funkcija. To je , dakle, funkcija koja poziva samu sebe. Najjednostavniji oblik rekurzivne funkcije sadrži samo jedan rekurzivni poziv i nerekurzivni izraz koji je uvijet završetka funkcije. Kada ne bi bilo nerekurzivnog izraza imali bismo baskonačnu petlju, npr.

```
rekurzivna_funkcija(){

return rekurzivna_funkcija();
}
```

U ovom slučaju rekurzivni pozivi bi se izvršavali u beskonačnost, zato je potrebno imati nerekurzivni izraz. Jedan od najjednostavnijih primjera rekurzivne funkcije je funkcija koja računa faktorijel od zadanog broja (n!=1*2*3*4*....*n). Vidimo da je faktorijel od nekog broja jednak umnošku tog broja i faktrijela toga broja umanjenog za 1, n!=n-(n-1)!. Isto tako vrijedi da je faktorijel od 0 jednako 1, 0!=1. Na temelju ta dva uvijeta možemo realizirati rekurzivnu funkciju faktorijel.

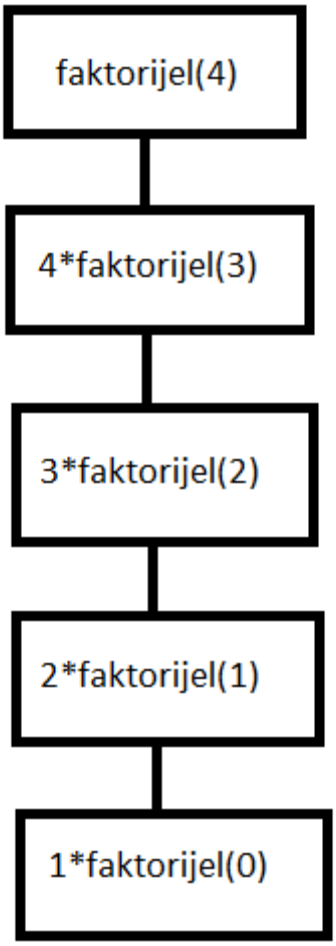
```
faktorijel(n){

if(n=0) return 1;

else return n*faktorijel(n-1);

}
```

Razvoj ove funkcije prilikom uzvršavanja za parametar 4 može se prikazati ovako:



Slika 3.1. *Razvoj rekurzivne funkcije faktroijel(4)*

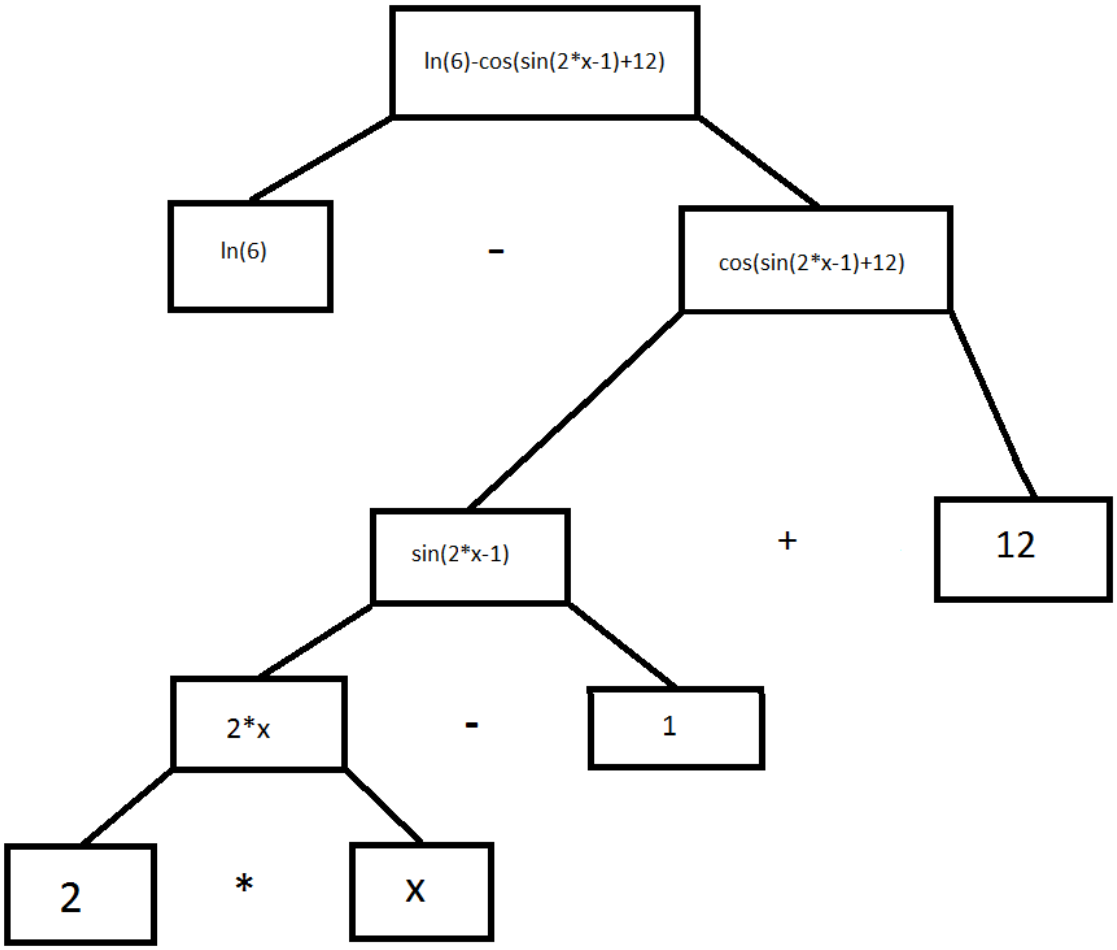
Funkcija se razvija prema dolje sve dok se ne dođe do zadnjeg rekurzivnog poziva koji vraća 1, nakon čega se rezultati šalju redoslijedom obrnutim od redoslijeda pozivanja. To znači da zadnji rekurzivni potiv prvi vraća rezultat, a prvi rekurzivni poziv zadnji vraća razultat. Dakle, pozvana rekurzivna funkcija vraća rezultat onoj koja ju je pozvala. Funkcija faktorijel je vrlo jednostavna rekurzivna funkcija sa samo jednim rekurzivnim pozivom. Rekurzivne funkcije mogi biti vrlo složene, što znači da mogu imati više rekurzivnih poziva. Rekurzivna funkcija za parsiranje matematičkih izraza je jedna od takvih.

3.2. Funkcija racunaj_rekurzija

Uzmimo da su zadana dva matematička izraza koja je potrebno izračunati:

sin(2*x-1)+12 i ln(6)-cos(sin(2*x-1)+12)

Vidimo da prvi izraz sadržan u drugom, a najbitnije je to da se oba izraza rješavaju na isti način. Neki složeni matamatički izraz rješava se na isti način kao i njegovi pojedini dijelovi. To nam daje ideju da se to može riješiti korištenjem rekurzivne funkcije. Glavni rekurzivni poziv za rješavanje prvog izraza bio bi samo jedan od sporednih poziva u rješavnju drugog izraza. Slikoviti prikaz razvoja rekurzivne funkcije za drugi izraz izgledao bi ovako:



Slika 3.2. *Razvoj rekurzije za funkciju ln(6)-cos(sin(2*x-1)+12)*

Na slici 3.2. vidi se da se razvojem rekurzivne funkcije dobije određeno stablo u kojem podređeni čvorovi vraćaju rezultat nadređenima sve dok se ne dođe do korijenskog čvora u koji se pohranjuje konačan rezultat. Dakle, ideja za rješavanje ovog problema je da se matematički izraz rastavlja na dva dijela te da se svaki od ta dva dijela, ako je potrebno, opet rastavlja na dva

dijela, sve dok se se ne dobije konkretan broj s kojim se može računati. Ako je zadan matematički izraz:

3.14*x+1/sin(62*x-0.04),

rekurzivna funkcija racunaj_rekurzija(3.14*x+1/sin(62*x-0.04) zapravo radi ovako:

racunaj_rekurzija (3.14*x) + racunaj_rekurzija (1/sin(62*x-0.4)).

To je ,naravno, slikoviti prikaz jer funkciji racunaj_rekurzija kao parametar ne predajemo gotovi izraz već pokazivać na prvi član niza koji sadrži taj izraz. Sada se svaki od ta dva priprojnika obrađuje posebno, prvi se opet rastavlja na dva dijela:

racunaj_rekurzija (3.14) * racunaj_rekurzija(x).

Dobiveni izrazi se ne mogu dalje rastaviti, izraz racunaj_rekurzija (3.14) vraća vrijednost 3.14, a izraz racunaj_rekurzija (x) vraća vrijednost x.

Drugi pribrojnik se rastavlja na:

racunaj_rekurzija (1) / racunaj_rekurzija (sin(62*x-0.4)).

Izraz racunaj_rekurzija (1) kao što smo već vidjeli vraća vrijednost 1, a izraz racunaj_rekurzija(sin(62*x-0.4)) je slučaj u kojem ne dolazi do rastavljanja na dva dijela. Budući da argument funkcije sinus nije konkretna vrijednost, on se predaje kao parametar funkciji racunaj_rekurzija i dobije se:

sin(racunaj_rekurzija (62*x-0.4)).

Argument funkcije sinus se dalje razvija na:

racunaj_rekurzija (62*x) - racunaj_rekurzija (0.4).

racunaj_rekurzija (0.4) vraća vrijednost 0.4, što je konkretan broj, a racunaj_rekurzija (62*x) se rastavlja na:

racunaj_rekurzija (62) * racunaj_rekurzija (x).

Sada je postupak rastavljanja niza završen, a kada svaki od izvršenih rekurzivnih poziva vrati vrijednost funkciji unutar koje je taj poziv izvršen može se izračunati rezultat cijelog matematičkog izraza.

Ako pogledamo izvorni C++ kod funkcije racunaj_rekurzija vidimo da je riječ je o jednoj veoma složenoj rekurzivnoj funkciji s mnogo rekurzivnih poziva. Ova funkcija ima samo jedan parametar, a to je pokazivač na niz koji sadrži uneseni matematički izraz. Taj pokazivač je zapravo adresa prvog člana niza. U samom početku tijela funkcije deklarirane su neke varijable koje su potrebne u ovoj funkciji. Varijabla brzag predstavlja broj zagrada, i postavljena je na vrijednost 0. To je ujedino i vrijednost na kojoj treba ostati ukoliko u unesenom matematičkom izrazu ne postoji greška u broju zagrada. Varijabla i deklarirana je kao globalna varijabla za potrebe for petlje, a u varijablu n sprema se podatak o broju znakova koji sadrži uneseni niz koji kao rezultat vraća funkcija strlen (eng. string lenght – duljina stringa). Varijabla pr označava mjesto ma kojem se niz prelama ako do toga dođe, a ako ne, njezina vrijednost ostaje -1 kao što je postavljeno pri deklaraciji. Pod mjestom na kojem se niz pralama smatra se indeks člana niza pri čemu treba imati u vidu da prvi član ima index 0.

Niz se može prelomiti samo na onom mjestu koji sadrži neki od operatora kao što su +, -, * i /. For petlja služi tome da pronađe mjesto prelamanja niza i da provjeri da li niz sadrži jednak broj otvorenih i zatvorenih zagrada. Ona se sastoji od tri if naredbe od kojih se samo jedna može izvršiti u pojedinoj iteraciji petlje. Prve dvije if naredbe služe za provjeru broja zagrada. Kod pojave otvorene zagrade varijabla brzag se poveća za 1 a kod pojave zatvorene zagrade se vrati na vrijednost 0. Ukoliko se zatvorena zagrada pojavi prije otvorene program završava s radom i ispisuje se poruka „Greška: izraz nepravilan zbog zadrada.“ Ako je izraz pravilan što se tiče zagrada varijabla brzag ostaje na vrijednosti 0. Treća if naredba pronalazi mjesto na kojem se niz dijeli na dva dijela, ukoliko takvo mjesto postoji, pri čemu treba voditi računa o prioritetu operatora. Niz se dijeli na mjestu operatira * ili / samo ako u nizu ne postoje operatori + i -. Mjesto dijeljenja niza mora biti takvo da za svaku otvorenu zagradu prije toga mjesta postoji točno jedna zatvorena zagrada koja se također nalazi ispred toga mjesta. Kada to ne bi vrijedilo moglo bi se dogoditi da se npr. matematički izraz „(x+3)*2 razlomi na:

„(x“ i „3)*2“,

Što nema nikakvog smisla. U već spomenutoj for petlji dobije se informacija o tome da li će se uneseni niz razdijeliti na dva dijela ili neće te se ovisno o tome program grana na dva dijela. Za takvo grananje najbolje je upotrijebiti kombinaciju naredbi if-else. Ako se ne izvrši if naredba izvršit će se else naredba. If naredba se izvršava ako niz treba podijeliti na dva dijela, a else

naredba se izvršava ako to nije slučaj. If naredba koja se izvršava ako se uneseni niz dijeli sadrži još četiri if naredbe, ovisno o operatoru na čijem je mjestu došlo do prelamanja niza.

Ta četiri operatora su +, -, * i /. Primjerice ako je do prelamanja došlo na mjestu operatora +, funkcija vraća vrijednost:

racuna_rekurzijaj(s) + racunaj_rekurzija (s+pr+1),

a vrijednost na mjestu na kojem je bio operator + se postavlja na ASCII vrijednost 0 da rekurzivna funkcija dobije informaciju o tome gdje je kraj prvog dijela niza kada za njega bude izvršila rekurzivni poziv. Kada se vrši rekurzivni poziv za prvi dio niza funkciji se za kao parametar predaje adresa prvog člana niza, odnosno adresa člana s[0]. Kada se funkcija poziva za drugi dio niza funkciji se predaje adresa onog člana koji slijedi iza opeatora +, a to je član s[pr+1]. Poseban slučaj je ako se niz dijeli na mjestu operatora /. U tom slučaju moramo paziti da nam se ne bi pojavio nedozvoljeni izraz 0/0. Primjerice izraz 1/0 je dozvoljen, to je beskonačno i s time program može dalje računati u programskom jeziku C++.

Ako dani matematički izraz nije potrebno dijeliti na dva dijela, kao što je već rečeno, izvršava se else naredba. Kada se izraz dijeli, kao što smo vidjeli, vrše se dva rekurzivna poziva. U ovom slučaju potreban nam je samo jedan rekurzivan poziv. Ako je primjerice unesen izraz „cos(x+2)“, koji će nakon normiranja poprimiti oblik C(x+2), potrebno se samo vratiti vrijednost

cos(racuna_rekurzijaj(s+2)),

ali prije toga je potrebno zadnji član niza postaviti na ASCII vrijednost 0 što označava kraj niza. Izravno korištenje matematičkih funkcija omogućili smo uključivanjem datoteke math.h naredbom #include<math.h>. Funkciji racunaj_rekurzija se predaje parametar s+2 zato što joj je potrebno predati pokazivač na niz „x+2“. Dakle zadnja zagrada postaje 0, a slovo C i prva zagrada se preskaču. Postupak je isti i za funkcije sinus, tangens i eksponencijalnu funkciju.
Za funkcije prirodni logaritam i drugi korijen moramo biti oprezni kada je u pitanju domena funkcije. Naime, ne možemo izračunati korijen od negativnog broja, a prirodni logaritam možemo računati samo za pozitivne brojeve. Ako je zadan niz „L(2-2)“ ne možemo samo vratiti vrijednost

log(racunaj_rekurzija (s+2))

jer bi dobili log(0) što nije moguće izračunati. Potrebno je vrijednost racunaj_rekurzija (s+2) spremiti u neku varijablu tipa float (u ovom slučaju je to varijabla test), zatim ispitati da li je njena vrijednost veća od 0 i tek ako je, potrebno je vratiti tu vrijednost. U suprotnom je potrebno ispisati da je došlo do greške u domeni funkcije prirodni logaritam. Postupak je isti i za funkciju drugi korijen, samo što varijabla test može poprimiti i vrijednost 0. Poseban slučaj predstavlja funkcija pow. To je također jedna od funkcija iz datoteke math.h. To je ujedino i jedina funkcija koja ima dva parametra. Ona omogućava računanje bilo koje potencije određenog broja. Dakle, prvi parametar je baza, a drugi potencija.
Zapis pow(x,3) zapravo predstavlja x^3. Ovo je, kao što vidimo, jedan vrlo jednostavan zapis ove funkcije ali šta kada nam se pojavi ovakav izraz

pow(log(2*x-4/3) + 17 , 3*(7*x+1)) ?

Prvo što moramo napravit pri realizaciji funkcije pow je pronaći mjesto na kojem se nalazi zarez jer on razdvaja jedina dva parametra ove funkcije. Kada odredimo na kojem se mjestu nalazi zarez, rekurzivnim pozivima posebno računamo izraz prije i poslije zareza. U navedenom primjeru nije problem pronaći zarez, ali šta ako imamo npr. ovakav zapis

pow(pow(x,y) , (2-x)) ?

U tom slučaju prvo bismo registrirali zarez od unutarnje funkcija pow te bismo računali $pow(x^{y),(2-x)}$ što je potpuno besmisleno. Dakle ako nam se u prvom argumentu funkcije pow pojavi još jedna takva funkcija moramo znati da prvi zarez koji registriramo pri analizi niza pripada unutarnjoj funkciji. To je u programu realizirano korištenjem varijable test tipa int. Treba napomenuti da ta varijabla nema nikakve veze s varijablom test koja je korištena za ispitivanje ispravnosti domene kod funkcija drugi korijen i prirodni logaritam. Ta je varijabla pri deklaraciji inicijalizirana na vrijednost 0. Ukoliko se u prvom parametru funkcije pow registrira još jedna takva funkcija varijabla test se postavi na vrijednost 1. Kada pri obilasku niza naiđemo na zarez ispitivamo vrijednost varijable test i ako je ona jednaka 1 znamo da se radi o zarezu unutarnje funkcije te da moramo tražiti dalje. Varijabla m tipa int ima sličnu primjenu kao i varijabla pr već spomenuta u tekstu. Treba napomenuti da se funkcija korijen može računati korištenjem funkcije pow jer vrijedi

$\sqrt{x} = x^{1/2}$.

Osim pojave neke od funkcija, postoje još tri jednostavna slučaja kada ne dolazi do razdvajanja matematičkog izraza. Naime, izraz se neće razdvajati pri pojavi neke od varijabli, brojčane konstante ili izraza unutar zagradi. U tom slučaju potrebno je samo vratiti dotičnu vrijednost. Za ispitivanje varijabli korištena je funkcija toupper iz datoteke ctype.h koja pretvara mala slova u velika. Ako kao parametar primi malo slovo pretvara ga u veliko, a ukoliko to nije moguće, vrati parametar takakv kakav jest. Ta funkcija nam dozvoljava da varijable unosimo kao mala ili kao velika slova. Kada bi ispitivanje varijablji vršili na ovaj način:

```
if (s[0]=='x') return x;
```

Index niza na kojem je pronađen zarez koji tražimo sprema se u varijablu m, a funkcija pow se realizira ovako:

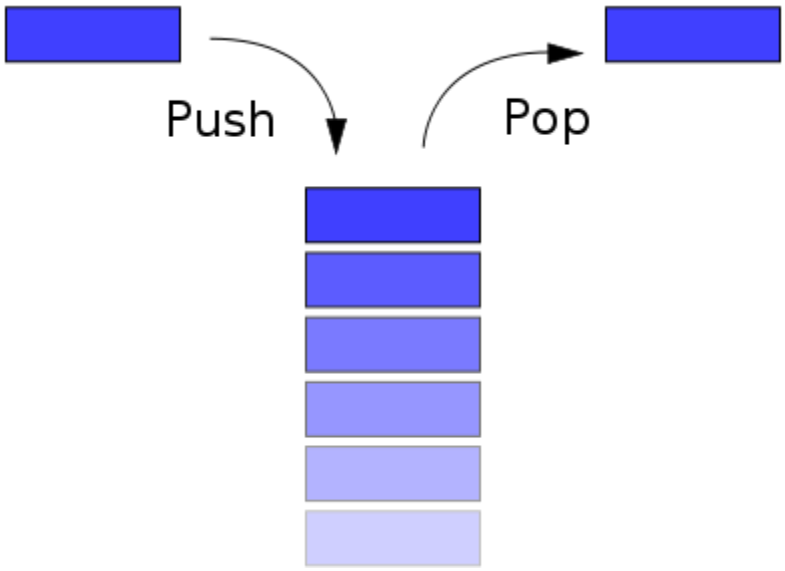
```
pow( racunaj_rekurzija (s+2) , racunaj_rekurzija (s+m+1).
```

program bi ne bi znao šta raditi u slučaju kada bismo za varijablu x koristili veliko slovo 'X'.
Za ispitivanje da li se radi o brojčanoj konstanti korištena je funkcija isdigit, također iz datoteke ctype.h. Ako funkcija isdigit kao parametar primi jednu od znamenki u rasponu od 0 od 9,vrati ih kao rezultat, u suprotnom vrati vrijednost „NULL“. Brojevi mogu biti zapisani i u ovom obliku: +2, -2, a dozvoljen je i zapis .2 što podrazumijeva 0.2. Kada se ustanovi sa se radi o brojčanoj konstanti potrebno ju je pomoću funkcije atof (ASCII to float) pretvoriti u varijablu tipa float, da bi se s njom moglo dalje računati. Treći slučaj kada ne dolazi do razdvajanja niza je pojava izraza unutar zagradi. U tom sličaju je potrebno preskočiti otvorenu zagradu, zatvorenu postaviti na vrijednost 0 te rekurzivnim pozivom izračunati izraz unutar zagradi na slijedeći način. Ovim a funkcija pokazuje koliko se toga može obaviti rekurzivnim pozivima i koliko to olakšava posao programeru.

4. PARSIRANJE POMOĆU STOGA

4.1. Stog kao struktura podataka

Stog je struktura podataka koja radi po principu FIFO (eng. First In First Out) što znači da se u određenom trenutku može dohvatiti samo onaj podatak koji je zadnji stavljen na stog. Upravo zato se svaka rekurzivna funkcija može realizirati pomoću stoga. Naime, rezultati koje daje pojedini rekurzivni poziv uzimat će se obrnutim redoslijedom od redoslijeda pozivanja. Dakle, rezultat koji vraća zadnji rekurzivni poziv biti će prvi potreban, a tek nakon što svi rerurzivni pozivi vrate rezultate možemo se odrediti rezultat prvog rekurzivnog poziva. O važnosti stoga dovoljno govori podatak da sveki mikroprocesorski sustav ima svoj sistemski stog. Da bismo mogli izvršavati operacije sa stogom moramo implementirati četiri osnovne funkcije za rad sa stogom, a to su funkcija push, pop, clear i is_empty. Funkcija push stavlja jedan element na stog i upravo taj element je njezin jedini parametar. Ne vraća ništa kao rezultat. Funkcija pop briše zadnji element stoga i vraća ga kao rezultat. Funkcija clear briše sve elemente sa stoga, a funkcija is_empty ispituje da li je stog prazan te vraća true ako je, a false ako nije prazan. Najjednostavnija realizacija stoga u programskom jeziku C++ je ona pomoću niza. Za takvu izvedbu potrebne su dvije varijable, pokazivač na zadnji član stoga (stack pointer) i varijabla koja označava broj elemenata niza. Pomoću te dvije varijable mogu se realizirati sve navedene funkcije za rad sa stogom. Pritom je potrebno paziti na neke detalje, npr. potrebno je osigurati da se ne poziva funkcija pop ako je stog prazan. Slijedeća slika prikazuje princip rada sa stogom:



Slika 4.1. *princip rada sa stogom*

4.2. Funkcija racunaj_stog

Osnovna ideja ove funkcije je da se stablo kao na slici 3.2. prikaže pomoću linearne strukture podataka kao što je stog. Na prvi pogled funkcija racunaj_stog ne izgleda ništa složenije od funkcije racunaj_rekurzija ali bitno je napomenuti da ona ne može funkconirati samostalno. Veliki dio posla obavi funkcija formiraj_stog koja popuni stog sa operandima i operatorima, a funkcija racunaj_stog samo uzima podatke sa stoga i izvršava određene računske operacije.

Glavni stog u ovom zadatku je stog S koji je realiziran tako da može sadržavati složene tipove podataka odnosno čitave strukture podataka. Osim njega koriste se još dva stoga, stog G i stog M. Stog G koristi se u funkciji formiraj_stog i služi za spremanje granica ulaznog normiranog niza sa kojim ta funkcija radi. Stog M služi za spremanje međurezultata prilikom izvršavanja funkcije racunaj_stog. Zadaća funkcije formiraj_stog je da popuni stog S pri čemu se služi sa pomoćnim stogom G. Ona prima samo jedan parametar, a to je pokazivač na prvi član niza od kojega je potrebno formirati stog S. Na stog S se stavljaju strukture podataka tipa cvor_ koje se sastoje od vrste operacije koju treba izvršiti(npr. ztrajanje, oduzimanje, računanje neke elementarne funkcijei slično) i vrijednosti operanada nad kojima se te operacije izvršavaju. Varijabla koja predstavlja vrstu operacije (varijabla tip) je tipa int zato što su osnovne operacije predstavljene cijelim brojevima (1 za zbrajanje, 2 za oduzimanje, 21 za funkciju sinus, 22 za funkciju kosinus i slično) što se najbolje može vidjedi u C++ kodu u prilogu. Također postoji slučaj u kojem nije potrebno obavljati nikakve matematičke operacije već samo vratiti određenu brojčanu vrijednost. U tom slučaju varijabla tip poprima vrijednost 90 ukoliko treba vratiti vrijednost x (analogno tome poprima vijednosti 91 i 92 za varijeble y i z) ili 95 ukoliko je potrebno vratiti konstantnu brojčanu vrijednost.
Ako je potrebno izvršiti npr. operaciju zbrajanja, niz se razdvaja na dva dijela kao što je to bio slučaj kod rekurzivne funkcije, ali se ne vrše rekurzivni pozivi već se na stog G stavljaju donja i gornja granica dobivenih podnizova. U tom slučaju se na stog S stavlja element e tipa čvor_ čija varijabla tip ima vrijednost 1 (oznaka za operaciju zbrajanja). Ako se pak radi o nekoj elementarnoj funkciji, potrebno je staviti na stog varijablu tipa cvor_ čija varijabla tip ima oznaku od 21 do 27 ali isto tako je i potrebno staviti donju i gornju granicu parametra dotične elementarne funkcije. To je potrebno zato što taj parametar može biti nekakav složeni izraz kojeg je potrebno dodatno obrađivati. Funkcija formiraj_stog treba samo formirati stog S, ne treba vratiti nikakvu vrijednost. Kada funkcija formiraj_stog napravi svoje, poziva se funkcija racunaj_stog.

Za razliku od rekurzivne funkcije ona ne prima nikakav parametar, mora samo vratiti konačan rezultat unesenog matematičkog izraza. Prilikom izvršavanja te funkcije formira se stog M koji služi za pohranu međurezultata.

Ta funkcija radi tako da dohvaća operande sa stoga S te nakon toga vrši matematičke operacije nad njima. Funkcija racunaj_stog ne izvršava Push operacije nad stogom S tako da on ostaje čitav nakon što se ona izvrši. Primjerice, ako smo sa stoga S dohvatili dva elementa, znamo da će nad njima biti izvršena neka binarna operacija. U ovom slučaju to mogu biti samo jedna od četiri osnovne matematičke operacije (zbrajanje, oduzimanje, množenje ili dijeljenje) . Ukoliko smo dohvatili jedan operand nakon kojeg slijedi element tipa čvor_ čija varijabla tip ima vrijednost 21, znamo da je nad tim operandom potrebno izvršiti operaciju sinus. Ne smije se dogoditi da nakon dva operanda slijedi operacija sinus zato što je to unarna operacija koja radi samo sa jednim operandom. Operandi koji se dohvaćaju sa stoga S spremaju se na stog M, a nakon što se izvrši željena operacija, njezin rezultat se također sprema na stog M. Kod funkcije formiraj_stog postoji „beskonačna“ for petlja koja završava nakon što se stog G isprazni. Funkcija racunaj_stog također ima jednu takvu petlju samo što je uvijet njenog završetka pražnjenje stoga S. Nakon što se dohvati zadnji element sa stoga S izvrši se zadnja operacija te se konačan rezultat sprema na stog M. Upravo taj rezultat vraća funkcija racunaj_stog.

Treba napomenuti da svaki stog prije korištenja treba isprazniti. Funkcija koja služi za pražnjenje stoga, funkcija koja ispituje da li je stog prazan kao i mnoge druge funkcije nalaze se u zaglavnoj datoteci stog.h koju je također potrebno uključiti prije pokretanja programa. U toj datoteci je stog realiziran kao klasa istoimenog naziva. Ako pogledamo tijelo klase Stog vidimo da je za realizaciju stoga korišten povezani popis. Budući da samo prvom elementu povezanog popisa možemo pristupiti izravno, uvijek se uzima upravo taj element dok drugi element povezanog popisa zauzima njegovo mjesto. Ukoliko dodajemo element na stog stavljamo ga na prvo mjesto povezanog popisa, ispred elementa koji je predhodno bio na prvom mjestu. Datoteku stog.h možemo uključiti u bilo koji algoritam koji koristi stog zato što se na stog mogu stavljati podatci bilo kojeg tipa.

Ako pogledamo definiciju funkcija Push i Pop u datotekama stog.h i stog1.h, vidimo da nije striktno definiran tip podataka sa kojima stog radi. To nam omogućuje naredba template. Taj tip je potrebno definirati prije poziva funkcije racunaj_stog. U ovom radu to neće biti jedan od osnovnih tipova koje podržava programski jezik C++ već posebno definirana struktura podataka cvor_.

4.3. Funkcija racunaj_stog1

Ako pogledamo sadržaj datoteke stog.h i tijelo funkcije racunaj_stog, vidimo da je realizacija stoga pomoću povezanog popisa vrlo kompleksna. Kao što je to već spomenuto u potpoglavlju 4.2., postoji mnogo jednostavnija realizacija stoga, a to je realizacija pomoću niza podataka.

Funkcije racunaj_stog i racunaj_stog1 u suštini su iste, razlika je samo u izvedbi stoga. Funkcija racunaj_stog treba funkciju formiraj_stog da joj „pripremi teren," odnosno formira stog S od ulaznog niza. Ista je stvar i sa funkcijum racunaj_stog1, samo što njena pomoćna funkcija nosi naziv formiraj_stog1. Funkcije formiraj_stog i formiraj_stog1 se, osim po imenu, razlikuju samo po imenima varijabli s kojima rade i imenima funkcija koje pozivaju. Da bi od niza realizirao stog potrebne su nam samo dvije varijable, pokazivač stoga (eng. stack pointer ili skraćeno SP) i varijabla N koja označava duljinu niza od kojeg se kreira stog. Varijabla N je u programu postavljena na vrijednost 1000 jer se smatra da nijedan uneseni matematički izraz neće sadržavati više znakova. Kada je stog prazan varijabla SP je postavljena na vrijednost 0 i pokazuje na prvi član niz. U tom slučaju nije dopušteno pozvati naredbu pop zato što se nema što uzeti sa stoga. Na prvom mjestu niza nalazit će se jedini podatak ako nad praznim stogom izvršimo operaciju push, a varijabla SP pokazivat će na njega. Nakon još jedne naredbe push varijabla SP se uvećava za jedan i pokazuje na drugi podatak u nizu. Dakle, varijabla SP uvijek poprima indeks onog člana niza koji je zadnji stavljen na stog. Naredba push se može izvršavati sve dok varijabla SP ne poprimi vrijednost N. Nakon toga je u potpunosti popunjen definirani niz.

Prilikom izvršavanja funkcija racunaj_stog i racunaj_stog1 nizom push i pop naredbi kreira se i uništi stog M. Glavni stog S ostaje netaknut zato što njega samo „pregledavamo." Kao što za korištenje funkcije racunaj_stog moramo uključiti datoteku stog.h, analogno za korištenje funkcije racunaj_stog1 treba uključiti datoteku stog1.h. Ta datoteka sadrži definiciju stoga pomoću niza, a u njoj je stog definiran kao klasa isto kao kod datoteke stog.h. Evo primjera osnovnih funkcija za rad sa stogom iz te datoteke:

5. EKPERIMANTALNI REZULTATI

Mjerenjem vremena izvršenja programa ovisno o tome da li je korištena funkcija racujan_rekurzija, racunaj_stog ili raacunaj_stog1 dobio sam slijedeću tablicu:

Tablica 5.1. *Prikaz vremena izvršenje programa ovisno o korištenoj funkciji*

x	racunaj_rekurzija	racunaj_stog	racunaj_stog1
100	6	9	0
1000	73	80	47
10000	516	611	187
100000	5105	6109	1419
1000000	49240	60022	13713
10000000	496232	584917	136282

Ova tablica je dobivena računanjem metemetičkog izraza:

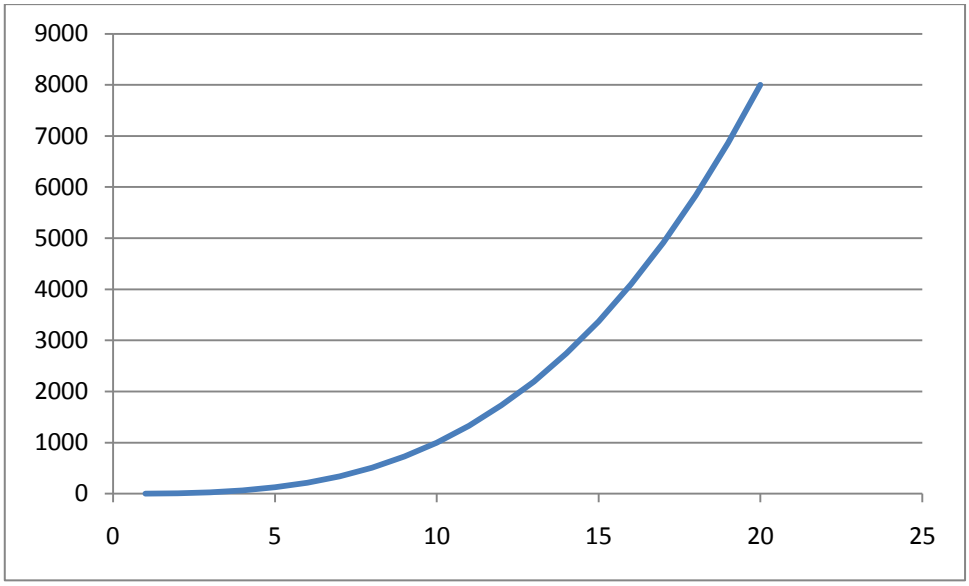
sin(2*x-1) + log(2/3) * tan(exp(x/1223) - pow(3/x,3)) - sqrt(109 * cos(12)) * x + 202 + pow(x,2)/ sin(2*x-1) + log(2/3) * tan(exp(x/1223) - pow(3/x,3)) - sqrt(109 * cos(12)) * x + 202 + pow(x,2)*1+2+14*x

Stupac „x“ ozačava do koje vrijednosti ide varijabla x u for petlji main funkcije počevši od 0.

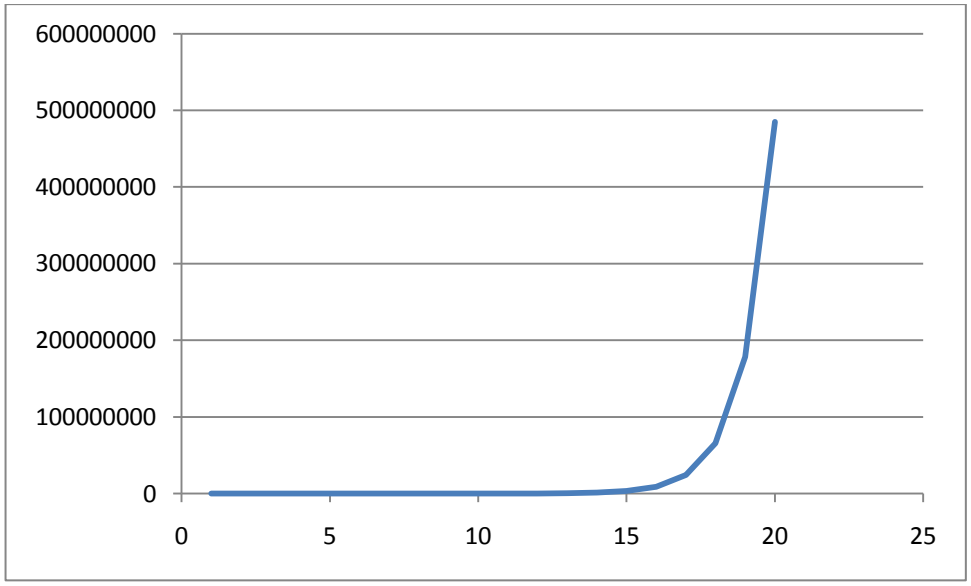
Dana vremena su u milisekundama.

Ako rezultate koje daje funkcija racunaj u pojedinoj iteraciji for petlje spremimo u tekstualnu datoteku, možemo ih kopirati u MS Excel te nacrtati graf funkcije.

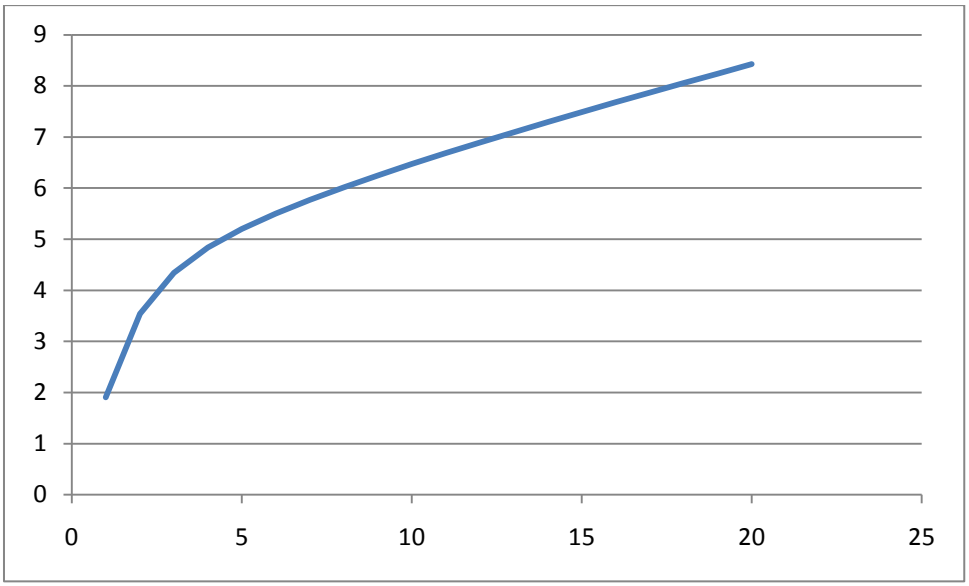
Evo nekoliko primjera takvih grafova funkcija:



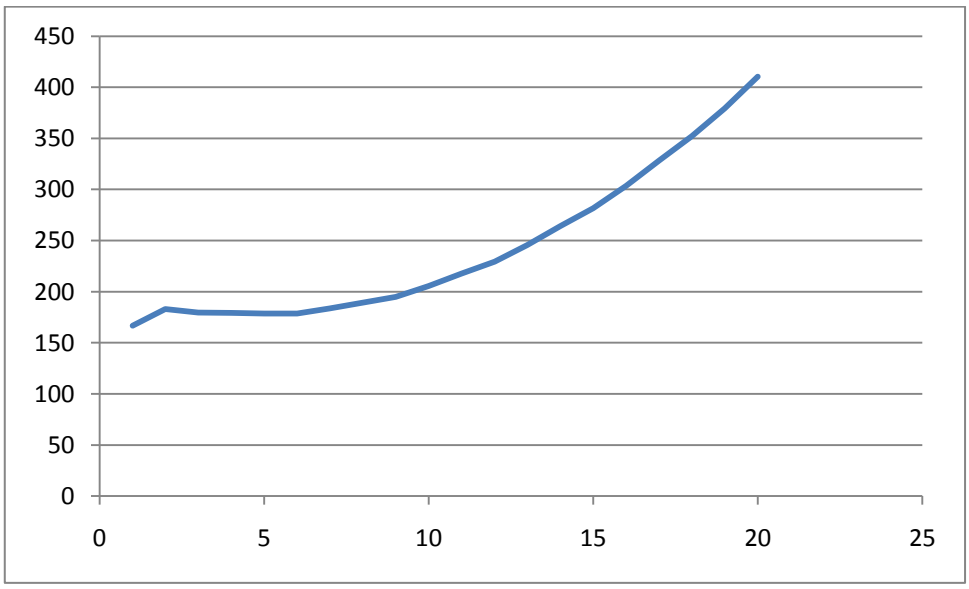
Slika 5.1. *graf funkcije* x^3



Slika 5.2. *graf funkcije* e^x



Slika 5.3. *graf funkcije sin(log(2x))-2/e^x+$\sqrt{4x}$*



Slika 5.4. *graf funkcije* sin(2*x-1) + log(2/3) * tan(exp(x/1223)

- pow(3/x,3)) - sqrt(109 * cos(12)) * x + 202 + pow(x,2)

6. ZAKLJUČAK

Funkcija za parsiranje matematičkih izraza je veoma složena funkcija zato jer ima puno detalja o kojima treba voditi računa pri izračunavanju nekog složenijeg matematičkog izraza. Što je više matematičkih operacija omogućeno, parsiranje je složenije. Takva funkcija se može realizirati na dva načina, pomoću stoga i pomoću rekurzije. Svaki algoritam koji koristi stog može se realizirati i pomoću rekurzivne funkcije, vrijedi i obratno. Obično je lakše napisati program koji koristi rekurzivnu funkciju zato što ima mnogo manje koda od programa koji koristi stog, ali realizacija sa stogom slovi kao brža za izvođenje. Lakša realizacija rekurzivnog algoritma olakšava posao programeru, ali algoritam koji koristi stog zbog bržeg izvođenja olakšava korištenje toga programa. Cilj ovoga rada bio je uspredba efikasnosti takva dva algoritma koji za svrhu imaju riješenje istog problema.

Teorija nalaže da algoritam koji koristi stog ima brže vrijeme izvođenja. Ovaj rad pokazuje da je bitna i činjenica kako je stog realiziran. Funkcija racunaj_stog1 u kojoj je stog realiziran pomoću niza ima tri do četiri puta manje vrijeme izvršavanja od funkcije racunaj_rekurzija zato što je stog realiziran na mnogo jednostavniji način. Funkcija racunaj_stog u kojoj je stog realiziran pomoću povezanog popisa, osim što je znatno složenija od funkcije racunaj_rekuruija, ima i veće vrijeme izvršavanja. Razlog tomu je činjenica da svaka push naredba dinamički alocira memoriju, a svaka pop naredba oslobađa taj dio alocirane memorije. Budući da prilikom izvršavanja funkcije racunaj_stog izvršava jako puno push i pop naredbi, jasno je zašto se funkcija racunaj_stog toliko sporo izvršava. S druge strane, kod izvedbe stoga s nizom samo jednom se zauzima memoriju za taj niz, a na kraju kad taj stog više nije potreban, oslobađa se zauzeta memoriju. Za izvršavanje push i pop naredbi kod funkcije racunaj_stog1 nije potrebno dinamički zauzimati memoriju već se samo vrši dohvat podatka iz niza što gotovo da i ne zauzima vrijeme. U nekim slučajevima je bolje koristiti povezani popis umjesto niza zato što kod povezanog popisa možemo podatak spremiti bilo gdje u memoriji, a pri korištenju niza moramo rezervirati određeni dio memorije zato što članovi niza moraju biti spremljeni na susjednim memorijskim lokacijama. Korisnika obično ne zanima koliko je neki program složen za realizirati već mu je bitno da je program jednostavan za korištenje. Brzina izvođenja svakako olakšava korištenje programa ali u ovom radu to nije od presudne važnosti zato što je gotovo nemoguće unijeti toliko složen matematički izraz koji bi se izvršavao dulje od jedne sekunde. Upravo zbog toga sam morao višestruko izračunavati isti izraz kako bih mogao uspoređivati vremena izvršavanja

već navedenih dviju implementacija parsera matematičkih izraza. Za to mi je poslužila for petlja u kojoj se isti matematički izraz izračunavao i do deseet milijuna puta za različite vrijednosti varijable x.

Složenost programa bitna je kod njegove izrade, a brzina izvođenja kod svakog izvršavanja. Budući da se program koji je jednom napisan obično poslije nebrojeno puta izvršava, dajem prednost brzini izvođenja pred složenosti realizacije te zaključujem da je algoritam koji koristi stog realiziran pomoću niza najefikasniji. Dakle, najefikasnija je funkcija racunaj_stog1 zato što pokazuje najbolji kompromis između brzine izvođenja i složenosti realizacije.

LITERATURA

[1] Julijan Šribar i Boris Motik, Demistificirani C++, Zagreb

[2] Horowitz & Sahni: Fundamentals of Computer Algorithms, Pitman, London

[3] Wirth: Algorithms + Data Structures = Programs, Prentice-Hall

[4] http://www.ucionica.net/novosti/Uvod-u-C-programski-jezik/

SAŽETAK

Ovim radom prikazan je proces izrade funkcije za parsiranje matematičkih izraza. Matematički izraz koji unosi korisnik sprema se u znakovni niz . U tom izrazu mogu se pojaviti osnovne matematičke operacije, zagrade i elementarne matematičke funkcije ali može i sadržavati jednu ili više varijabli. Potrebno je voditi računa o prioritetima matematičkih operacija. Realizirane su tri funkcije, racunaj_rekurzija koja koristi rekurzivnu funkciju, racunaj_stog koja koristi stog realiziran pomoću povezanog popisa i racunaj_stog1 koja koristi stog realiziran pomoću niza podataka. Za sve tri funkcije korišten je programski jezik C++. Uspoređivana je efikasnost tih triju funkcija. Efikasan je onaj algoritam koji ima minimalno vrijeme izvršenja i minimalnu složenost. Funkcija racunaj_rekurzija je najjednostavnija za izradu, slijedi racunaj_stog1, a najsloženija je funkcija racunaj_stog. Najkraće vrijeme izvođenja ima funkcija racunaj_stog1, slijedi racunaj_rekurzija, a najsporije se izvršava funkcija racunaj_stog. Budući da funkcija racunaj_stog1 ima najkraće vrijeme izvršavanja, a nije najsloženija, zaključujem da je ona najefikasnija. Prikazan je i primjer vizualizacije funkcije s jednom varijablom u obliku grafa.

Ključne riječi: parsiranje, rekurzivna funkcija, stog

ABSTRACT

Mathematical expression parsing

This work shows the process of creating function for mathematical expression parsing. Mathematical expression which is entered by the user need to be saved in character array. This expression may contain basic mathematical operations, braces, elementary mathematical functions but it also may contain one or more varriable. It is necessary to have in mind priority of mathematical operations. Three function are made, racunaj_rekurzija which uses recursive function, racunaj_stog which uses a stack made by linked list and racunaj_stog1 which uses a stack made by array. All of those three function are made in C++ programming language. It was compared efficacy of those two functions. Program is effective if it has minimal execution time and minimal complexity. Function racunaj_rekurzija is the easiest to made, then comes racunaj_stog1 and the most complex is racunaj_stog. Function racunaj_stog1 is the fastest to execute, then comes racunaj_rekurzija and the slowest is racunaj_stog. Since racunaj_stog1 has the lowest execution time and it is not the most complex, my conclusion is that this function is the most effective. It is showed an example of vizualization of one variable function by a graph.

Key words: parsing, recursive function, stack

ŽIVOTOPIS

Božidar Patača rođen je 1. rujna 1988. godine u Osijeku, Hrvatska. Prva četiri razreda osnovne škole završio je u Krčeniku, a slijedeća četiri u Osnovnoj školi Ante Starčevića u Viljevu. Svih osam razreda završio je s odličnim uspjehom. U šestom razredu sudjelovao je na međuopćinskom natjecanju iz informatike u Našicama i osvojio četvrto mjesto.

2003. godine upisuje Opću gimnaziju u Srednjoj školi Donji Miholjac u Donjem Miholjcu. Sva četiri razreda završio je s odličnim uspjehom.

2007. godine dobiva priliku za izravan upis na Elektrotehnički fakultet u Osijeku te upisuje smjer računarstvo. Od samog početka studiranja prima stipendiju za studente koji se obrazuju za deficitarna zanimanja od Osječko baranjske županije.

_____

Božidar Patača

PRILOZI

Cjelokupni C++ izvorni kod sa svim funkcijama obrađenim u ovom radu:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>
#include <time.h>
#include "stog.h"
#include "stog1.h"

char s[1000];
int i;
float x=1, y=1, z=1;
// funkcija koja ispisuje grešku ako dođe do problema
// u računanju, ova funkcija također završava program
void kraj( int );
float racunaj_rekurzija( char * );
void normiraj( char *v );
struct cvor_
{
  int tip;
  float rez;
};
typedef struct cvor_ Element;
void formiraj_stog( char * );
void formiraj_stog1( char * );
float racunaj_stog();
float racunaj_stog1();
Stog<Element *> S;
Stog1<Element *> S1;
// Ovo je glavna funkcija koja samo testira kako sve radi
```

```
int main()
{
	float rez; char ul[1000], ul2[1000];
	time_t t1, t2;
	double vrijeme;
	gets( ul );
	normiraj( ul );
	strcpy( ul2, s );
	FILE *f;
	f=fopen("rezultati.txt","w");
	t1 = clock();	// ALTERNATIVNO: time(&t1);
	formiraj_stog( s );
	formiraj_stog1( s );
//for(x=1;x<=10;x++)
 {
   rez = racunaj_stog1();
	rez = racunaj_stog();
	strcpy( s, ul2 );
	rez = racunaj_rekurzija( s );
		fprintf(f,"%f\n",rez);
 }
   t2 = clock();	ALTERNATIVNO: time(&t2);
	vrijeme = t2-t1;	ALTERNATIVNO: vrijeme=difftime(t2,t1);
	printf( "Rezultat = %f\n", rez );
	fprintf(f,"%f\n",rez);
	printf( "vrijeme izvodenja = %f\n", vrijeme );
	return 0;
}

float racunaj_rekurzija( char *s )
{
  int brzag = 0;
  int n = strlen( s ), pr = -1;
 // puts( s ); // ovaj ispis treba maknuti, ali dobro je da ostane
```

```
        // zbog debagiranja da se vidi šta svaki rekurzivni
        // poziv dobiva kao podstring

 // ova for-petlja testira da li je sve u redu sa zagradama
 // i usput nadje mjesto pr gdje treba prelomiti string
 for (i=1;i<n;i++)
 {
  if (s[i]=='(')
        {
         brzag++;
         continue;
        }
        if (s[i]==')')
        {
    brzag--;
         if (brzag<0) kraj( 1 );
    continue;
        }
        if (brzag==0 && (s[i]=='+' || s[i]=='-' || s[i]=='*' || s[i]=='/'))
        {
         if (pr==-1) pr = i;
         else if (s[i]=='+' || s[i]=='-') pr = i; // ovdje uzimamo u obzir prioritet operatora+ i - u odnosu na * i /
        }
 }
 if (brzag!=0) kraj( 1 );
 if (pr!=-1)
 {  // ako je došlo do razdvajanja stringa, obavljamo računsku operaciju
        if (s[pr]=='+'){ s[pr] = 0; return racunaj_rekurzija( s ) + racunaj_rekurzija(s+pr+1 ); }
        if (s[pr]=='-'){ s[pr] = 0; return racunaj_rekurzija( s ) - racunaj_rekurzija(s+pr+1 ); }
        if (s[pr]=='*'){ s[pr] = 0; return racunaj_rekurzija( s ) * racunaj_rekurzija(s+pr+1 ); }
        if (s[pr]=='/'){ s[pr] = 0; float lij = racunaj_rekurzija( s ); // djeljenje je uvijek malo specifično, 0/0 ne smijemo pustiti, ali 1/0 je ok, to je +beskonačno (probaj!)
                        float des = racunaj_rekurzija( s+pr+1 );
```

```
                    if (lij==0 && des==0) kraj( 2 );
                    else return lij/des;
                }
}
else
{ // ovo su specijalni slučajevi kada se izraz ne razbija na dva dijela
 if (toupper(s[0])=='X') return x;
      if (toupper(s[0])=='Y') return y;
      if (toupper(s[0])=='Z') return z;
 // u sljedećem if-u gledamo ako je to neka brojčana konstanta
      if (isdigit(s[0]) || s[0]=='.' || s[0]=='-' || s[0]=='+')
            return atof( s ); // funkcija aotf (ascii to float) prevodi string u broj
      if (s[0]=='(')
      {  // ako izraz počinje zagradom, onda i završava zagradom!
       s[n-1] = 0;
       return racunaj_rekurzija( s+1 );
      }
      if (s[0]=='S')
      { // ovo je za elementarnu funkciju sinus
       s[n-1] = 0;
       return sin( racunaj_rekurzija( s+2 ) );
      }
      if (s[0]=='C')
      { // ovo je za elementarnu funkciju kosinus
       s[n-1] = 0;
       return cos( racunaj_rekurzija( s+2 ) );
      }
      if (s[0]=='E')
      { // ovo je za exponencijalnu funkciju
       s[n-1] = 0;
       return exp( racunaj_rekurzija( s+2 ) );
      }
```

```
if (s[0]=='L')
{ // ovo je za funkciju prirodni logaritam
 s[n-1] = 0;
 float test=racunaj_rekurzija(s+2);
 if(test>0) return log( test );
 else kraj(3);
}
if (s[0]=='T')
{ // ovo je za elementarnu funkciju sinus
 s[n-1] = 0;
 return tan( racunaj_rekurzija( s+2 ) );
}
if (s[0]=='K')
{ // ovo je za funkciju drugi korijen
 s[n-1] = 0;
 float test=racunaj_rekurzija(s+2);
 if(test>=0) return sqrt( test );
 else kraj(4);
}
if(s[0]='P')
{ // ovo je za funkciju power
        s[n-1]=0;
        int m, i, test=0;
        for(i=1;i<n;i++)
        {
                if(s[i]=='P') test++;

                        if(s[i]==',')
                        {
                                if(test==0){ m=i; s[m]=0; }
                                test--;
                        }
        }
                return pow( racunaj_rekurzija(s+2), racunaj_rekurzija(s+m+1) );
```

```
        }
  }
  return 0;
}

float racunaj_stog1()
{
  Stog1<Element *> M;  // stog za spremanje međurezultata
  M.Isprazni1();
  Element *e, *f, *g;
// ova petlja sada ide po i unazad, znači od vrha stog1a prema dnu
  for (i=S1.SP-1;i>=0;i--)   //Cilj je da ne uništimo podatke sa stog1a, nego da ih samo sve 'pregledamo'
  {
//  Ovako dohvaćamo elemenat stog1a
      e = S1.V[i];
        if (e->tip==90)
        {
          M.Push1( e );
          continue;
        }
        if (e->tip==95)
        {
          M.Push1( e );
          continue;
        }
        if (e->tip==1)   // Ako je to neka od računskih operacija, onda skinemo dva zadnja
        {                       // međurezultata sa M i zbrojimo ih, te taj zbroj dodamo kao novi međurezultat
          f = M.Pop1();
          g = M.Pop1();
          e->rez = f->rez + g->rez;              //za operaciju zbrajanja
          M.Push1( e );
          continue;
```

```
        }
        if (e->tip==2)
        {
          f = M.Pop1();
          g = M.Pop1();
          e->rez = f->rez - g->rez;              //za operaciju oduzimanja
          M.Push1( e );
          continue;
        }
        if (e->tip==3)
        {
          f = M.Pop1();
          g = M.Pop1();
          e->rez = f->rez * g->rez;              //za operaciju množenja
          M.Push1( e );
          continue;
        }
if (e->tip==4)
        {
          f = M.Pop1();
          g = M.Pop1();
          if (f->rez==0 && g->rez==0) kraj( 2 );
          else
          {
          e->rez = f->rez / g->rez;                    //za operaciju dijeljenja
          M.Push1( e );
          }
          continue;
        }
if (e->tip==21)  // Ako je to neka od funkcija, onda skinemo zadnji međurezultat sa M
        {              // izračunamo tu funkciju i opet vratimo kao novi međurezultat na stog1 M
          f = M.Pop1();
          e->rez = sin( f->rez );          //za funkciju sinus
          M.Push1( e );
```

```
            continue;
          }
          if (e->tip==22)
          {
            f = M.Pop1();
            e->rez = cos( f->rez );        //za funkciju kosinus
            M.Push1( e );
            continue;
          }

if (e->tip==23)
          {
            f = M.Pop1();
            e->rez = exp( f->rez );        //za eksponencijalnu funkciju
            M.Push1( e );
            continue;
          }
if (e->tip==24)
          {
            f = M.Pop1();
            if(f->rez>0)
            {
            e->rez = log( f->rez );        //za funkciju prirodni logaritam
            M.Push1( e );
            continue;
            }
            else kraj(3);
          }
if (e->tip==25)
          {
            f = M.Pop1();
            e->rez = tan( f->rez ); //za funkciju tangens
            M.Push1( e );
            continue;
```

```
        }
if (e->tip==26)
        {
         f = M.Pop1();
         if(f->rez>=0)
         {
         e->rez = sqrt( f->rez ); //za funkciju drugi korijen
         M.Push1( e );
         continue;
         }
         else kraj(3);
        }
if (e->tip==27)
        {
         f = M.Pop1();
         g = M.Pop1();
         e->rez = pow( g->rez, f->rez ); //za funkciju power
         M.Push1( e );
         continue;
        }
  }
  f = M.Pop1(); // Na kraju kad cijela petlja završi na stog1u M postoji samo 1 element i to je konačni rez
  return f->rez;
}

float racunaj_stog()
{
  Stog<Element *> M;  // stog za spremanje međurezultata
  M.Isprazni();
  Element *e, *f, *g;
  OE<Element *> *t;
  t = S.prvi;
  for (;;)  // Princip je sljedeći: vadimo stalno sa stoga S, ako je to specijalni slučaj
```

```
{       // onda ga jednostavno stavimo na stog međurezultata M ...
        if (t==NULL) break;
        e = t->x;
        t = t->sljedeci;
        if (e->tip==90)
        {
          e->rez = x;
          M.Push( e );
          continue;
        }
        if (e->tip==95)
        {
          M.Push( e );
          continue;
        }
        if (e->tip==1)   // Ako je to neka od računskih operacija, onda skinemo dva zadnja
        {                       // međurezultata sa M i zbrojimo ih, te taj zbroj dodamo kao novi
međurezultat
          f = M.Pop();
          g = M.Pop();
          e->rez = f->rez + g->rez;             //za operaciju zbrajanja
          M.Push( e );
          continue;
        }
        if (e->tip==2)
        {
          f = M.Pop();
          g = M.Pop();
          e->rez = f->rez - g->rez;             //za operaciju oduzimanja
          M.Push( e );
          continue;
        }
        if (e->tip==3)
        {
```

```
            f = M.Pop();
            g = M.Pop();
            e->rez = f->rez * g->rez;              //za operaciju množenja
            M.Push( e );
            continue;
           }
    if (e->tip==4)
           {
            f = M.Pop();
            g = M.Pop();
            if (f->rez==0 && g->rez==0) kraj( 2 );
            else
            {
            e->rez = f->rez / g->rez;                    //za operaciju dijeljenja
            M.Push( e );
            }
            continue;
           }
    if (e->tip==21)  // Ako je to neka od funkcija, onda skinemo zadnji međurezultat sa M
           {             // izračunamo tu funkciju i opet vratimo kao novi međurezultat na stog M
            f = M.Pop();
            e->rez = sin( f->rez );         //za funkciju sinus
            M.Push( e );
            continue;
           }
           if (e->tip==22)
           {
            f = M.Pop();
            e->rez = cos( f->rez );         //za funkciju kosinus
            M.Push( e );
            continue;
           }
if (e->tip==23)
           {
```

```
            f = M.Pop();
            e->rez = exp( f->rez );        //za eksponencijalnu funkciju
            M.Push( e );
            continue;
           }
if (e->tip==24)
           {
            f = M.Pop();
            if(f->rez>0)
            {
            e->rez = log( f->rez );        //za funkciju prirodni logaritam
            M.Push( e );
            continue;
            }
            else kraj(3);
           }
if (e->tip==25)
           {
            f = M.Pop();
            e->rez = tan( f->rez ); //za funkciju tangens
            M.Push( e );
            continue;
           }
if (e->tip==26)
           {
            f = M.Pop();
            if(f->rez>=0)
            {
            e->rez = sqrt( f->rez ); //za funkciju drugi korijen
            M.Push( e );
            continue;
            }
            else kraj(3);
           }
```

```
if (e->tip==27)
        {
         f = M.Pop();
         g = M.Pop();
         e->rez = pow( g->rez, f->rez ); //za funkciju power
         M.Push( e );
         continue;
        }
   // ovdje ide dalje podrška za ostale funkcije... tipa 23,24,...
  }
  f = M.Pop(); // Na kraju kad cijela petlja završi na stogu M postoji samo 1 element i to je konačni rez
  return f->rez;
}

void kraj( int zasto )
{
  if (zasto==1) puts( "Greska: izraz nepravilan zbog zagrada." );
  if (zasto==2) puts( "Greska: neodredjeni izraz -> NaN." );
  if (zasto==3) puts( "Greska: greska u domeni funkcije prirodni logaritam, argument mora biti veci od nule." );
  if (zasto==4) puts( "Greska: greska u domeni funkcije drugi korijen, argument mora biti veci ili jednak nuli." );
  if (zasto==5) puts( "Greska u otvaranju datoteke" );


  exit( 0 );
}

void formiraj_stog( char *ul )
{
  int dg, gg;
  S.Isprazni(); // U stog1 S stavljat ćemo operande i operatore (+,-,/,*) i funkcije!
  Stog<int> G;  // stog1 G prati granice u ulaznom stringu, tako da se sve dobro isparsira
```

```
 G.Isprazni();
 int brzag = 0;
 int n = strlen( ul ), pr;
 G.Push( 0 );   // najprije stavimo početne granice
 G.Push( n-1 );
 for (;;)
 {
  if (G.JePrazan()) break;   // uvjet za kraj je kad se G Isprazni1
  gg = G.Pop();            // uvijek skinemo granice s vrha stog1a
        dg = G.Pop();
  pr = -1;
  for (i=(dg+1);i<=gg;i++)  // ova petlja je ista kao u rekurzivnoj funkciji, razbija string ul na dva dijela ili nalazi specijalni slučaj
  {
        // putchar(ul[i]);
    if (ul[i]=='(')
        {
         brzag++;
         continue;
        }
        if (ul[i]==')')
        {
     brzag--;
         if (brzag<0) kraj( 1 );
     continue;
        }
        if (brzag==0 && (ul[i]=='+' || ul[i]=='-' || ul[i]=='*' || ul[i]=='/'))
        {
         if (pr==-1) pr = i;
         else if (ul[i]=='+' || ul[i]=='-') pr = i; // ovdje uzimamo u obzir prioritet + i - u odnosu na * i /
        }
  }
  // putchar( '\n' );
```

```
  if (brzag!=0) kraj( 1 );  // isto kao u rekurzivnoj funkciji

  if (pr!=-1)
  {  // ako je došlo do razdvajanja stringa, obavljamo računsku operaciju
        G.Push( pr+1 );  // ovdje je došlo do razdvajanja na dva podstringa
        G.Push( gg );    // prvo stavljamo desni dio, zato što tako radi i
   G.Push( dg );    // rekurzivna funkcija, ali isto je i ako prvo stavimo
        G.Push( pr-1 );  // lijevi pa desni dio
        Element *e = new Element;  // Stvaramo novi element na dinamički način...
        if (ul[pr]=='+') { ul[pr] = 0; e->tip = 1; }   // operacije su 1 -> +
        if (ul[pr]=='-') { ul[pr] = 0; e->tip = 2; }   // 2 -> -
        if (ul[pr]=='*') { ul[pr] = 0; e->tip = 3; }   // 3 -> *
        if (ul[pr]=='/') { ul[pr] = 0; e->tip = 4; }   // 4 -> /
        S.Push( e );  // .. i stavljamo ga na stog1 S
  }
  else
      { // ovo su specijalni slučajevi kada se izraz ne razbija na dva
   if (toupper(ul[dg])=='X')
        {
            Element *e = new Element;
            e->tip = 90;
            e->rez = x;
            S.Push( e );
        }
if (toupper(ul[dg])=='Y')
        {
            Element *e = new Element;
            e->tip = 91;
            e->rez = y;
            S.Push( e );
        }
if (toupper(ul[dg])=='Z')
        {
            Element *e = new Element;
```

```
            e->tip = 92;
            e->rez = z;
            S.Push( e );

        }
// u sljedećem if-u gledamo ako je to neka brojčana konstanta
        if (isdigit(ul[dg]) || ul[dg]=='.' || ul[dg]=='-' || ul[dg]=='+')
        {
          Element *e = new Element;
            e->tip = 95;
            e->rez = atof( ul+dg );
            S.Push( e );
        }
        if (ul[dg]=='(')
        {  // ako izraz počinje zagradom, onda i završava zagradom!
          G.Push( dg+1 );
            G.Push( gg-1 );
        }
        if (ul[dg]=='S')
        { // ovo je za elementarnu funkciju sinus (tip 21)
          Element *e = new Element;
            e->tip = 21;
            S.Push( e );
            G.Push( dg+2 );
            G.Push( gg-1 );
        }
        if (ul[dg]=='C')
        { // ovo je za elementarnu funkciju kosinus
          Element *e = new Element;
            e->tip = 22;
            S.Push( e );
            G.Push( dg+2 );
            G.Push( gg-1 );
        }
```

```
if (ul[dg]=='E')
        { // ovo je za eksponencijalnu funkciju
          Element *e = new Element;
              e->tip = 23;
              S.Push( e );
              G.Push( dg+2 );
              G.Push( gg-1 );
        }
if (ul[dg]=='L')
        { // ovo je za elementarnu funkciju prirodni logaritam
          Element *e = new Element;
              e->tip = 24;
              S.Push( e );
              G.Push( dg+2 );
              G.Push( gg-1 );
        }
if (ul[dg]=='T')
        { // ovo je za elementarnu funkciju tangens
          Element *e = new Element;
              e->tip = 25;
              S.Push( e );
              G.Push( dg+2 );
              G.Push( gg-1 );
        }
if (ul[dg]=='K')
        { // ovo je za funkciju drugi korijen
          Element *e = new Element;
              e->tip = 26;
              S.Push( e );
              G.Push( dg+2 );
              G.Push( gg-1 );
        }
if (ul[dg]=='P')
        { //ovo je za funkciju power
```

```
            Element *e = new Element;
                int m, i, test=0;
                for(i=dg+1;i<gg;i++)
                {

                        if(ul[i]=='P') test++;

                                if(ul[i]==',')
                                {
                                        if(test==0){ m=i; ul[m]=0; }
                                        test--;
                                }
                }
                e->tip = 27;
                S.Push( e );
                G.Push( dg+2 );
                G.Push( m-1 );
                G.Push( m+1 );
                G.Push( gg-1 );
           }
         } // od else za posebne slučajeve
  } // od beskonačne for
}

void formiraj_stog1( char *ul )
{
  int dg, gg;
  S1.Isprazni1(); // U Stog11 S stavljat ćemo operande i operatore (+,-,/,*) i funkcije!
  Stog1<int> G;  // Stog11 G prati granice u ulaznom stringu, tako da se sve dobro isparsira
  G.Isprazni1();
  int brzag = 0;
  int n = strlen( ul ), pr;
  G.Push1( 0 );   // najprije stavimo početne granice
  G.Push1( n-1 );
```

```
 for (;;)
 {
  if (G.JePrazan1()) break;   // uvjet za kraj je kad se G Isprazni11
  gg = G.Pop1();            // uvijek skinemo granice s vrha Stog11a
        dg = G.Pop1();
  pr = -1;
  for (i=(dg+1);i<=gg;i++)  // ova petlja je ista kao u rekurzivnoj funkciji, razbija string ul na dva dijela ili nalazi specijalni slučaj
  {
        // putchar(ul[i]);
   if (ul[i]=='('
        {
         brzag++;
         continue;
        }
        if (ul[i]==')')
        {
     brzag--;
         if (brzag<0) kraj( 1 );
     continue;
        }
        if (brzag==0 && (ul[i]=='+' || ul[i]=='-' || ul[i]=='*' || ul[i]=='/'))
        {
         if (pr==-1) pr = i;
         else if (ul[i]=='+' || ul[i]=='-') pr = i; // ovdje uzimamo u obzir prioritet + i - u odnosu na * i /
        }
  }
  // putchar( '\n' );
  if (brzag!=0) kraj( 1 );  // isto kao u rekurzivnoj funkciji

  if (pr!=-1)
```

```
  {  // ako je došlo do razdvajanja stringa, obavljamo računsku operaciju
        G.Push1( pr+1 );  // ovdje je došlo do razdvajanja na dva podstringa
        G.Push1( gg );    // prvo stavljamo desni dio, zato što tako radi i
        G.Push1( dg );    // rekurzivna funkcija, ali isto je i ako prvo stavimo
        G.Push1( pr-1 );  // lijevi pa desni dio
        Element *e = new Element;  // Stvaramo novi element na dinamički način...
        if (ul[pr]=='+') { ul[pr] = 0; e->tip = 1; }   // operacije su 1 -> +
        if (ul[pr]=='-') { ul[pr] = 0; e->tip = 2; }   // 2 -> -
        if (ul[pr]=='*') { ul[pr] = 0; e->tip = 3; }   // 3 -> *
        if (ul[pr]=='/') { ul[pr] = 0; e->tip = 4; }   // 4 -> /
        S1.Push1( e );  // .. i stavljamo ga na Stog11 S
  }
  else
      { // ovo su specijalni slučajevi kada se izraz ne razbija na dva
   if (toupper(ul[dg])=='X')
        {
            Element *e = new Element;
            e->tip = 90;
            e->rez = x;
            S1.Push1( e );
        }
if (toupper(ul[dg])=='Y')
        {
            Element *e = new Element;
            e->tip = 91;
            e->rez = y;
            S1.Push1( e );
        }
if (toupper(ul[dg])=='Z')
        {
            Element *e = new Element;
            e->tip = 92;
            e->rez = z;
            S1.Push1( e );
```

```
        }
    // u sljedećem if-u gledamo ako je to neka brojčana konstanta
        if (isdigit(ul[dg]) || ul[dg]=='.' || ul[dg]=='-' || ul[dg]=='+')
        {
          Element *e = new Element;
                e->tip = 95;
                e->rez = atof( ul+dg );
                S1.Push1( e );
        }
        if (ul[dg]=='(')
        {  // ako izraz počinje zagradom, onda i završava zagradom!
          G.Push1( dg+1 );
                G.Push1( gg-1 );
        }
        if (ul[dg]=='S')
        { // ovo je za elementarnu funkciju sinus (tip 21)
          Element *e = new Element;
                e->tip = 21;
                S1.Push1( e );
                G.Push1( dg+2 );
                G.Push1( gg-1 );
        }
        if (ul[dg]=='C')
        { // ovo je za elementarnu funkciju kosinus
          Element *e = new Element;
                e->tip = 22;
                S1.Push1( e );
                G.Push1( dg+2 );
                G.Push1( gg-1 );
        }
if (ul[dg]=='E')
        { // ovo je za eksponencijalnu funkciju
          Element *e = new Element;
                e->tip = 23;
```

```
                S1.Push1( e );
                G.Push1( dg+2 );
                G.Push1( gg-1 );
        }
if (ul[dg]=='L')
        { // ovo je za elementarnu funkciju prirodni logaritam
          Element *e = new Element;
                e->tip = 24;
                S1.Push1( e );
                G.Push1( dg+2 );
                G.Push1( gg-1 );
        }
if (ul[dg]=='T')
        { // ovo je za elementarnu funkciju tangens
          Element *e = new Element;
                e->tip = 25;
                S1.Push1( e );
                G.Push1( dg+2 );
                G.Push1( gg-1 );
        }
if (ul[dg]=='K')
        { // ovo je za funkciju drugi korijen
          Element *e = new Element;
                e->tip = 26;
                S1.Push1( e );
                G.Push1( dg+2 );
                G.Push1( gg-1 );
        }
if (ul[dg]=='P')
        { //ovo je za funkciju power
          Element *e = new Element;
                int m, i, test=0;
                for(i=dg+1;i<gg;i++)
                {
```

```
                        if(ul[i]=='P') test++;

                                if(ul[i]==',')
                                {
                                        if(test==0){ m=i; ul[m]=0; }
                                        test--;
                                }
                }
                e->tip = 27;
                S1.Push1( e );
                G.Push1( dg+2 );
                G.Push1( m-1 );
                G.Push1( m+1 );
                G.Push1( gg-1 );
          }
        } // od else za posebne slučajeve
  } // od beskonačne for
}

void normiraj( char *v )
{
        int i,j=0;
        int n=strlen(v);

        for(i=0;i<n;i++){

                if(v[i]==' ' || v[i]=='\t') continue;
                        if(v[i]=='(' || v[i]==')' || isdigit(v[i]) || v[i]=='x' || v[i]=='y'
                                || v[i]=='z' || v[i]=='+' || v[i]=='-' || v[i]=='*'
                                ||  v[i]=='/' || v[i]=='.' || v[i]==',') {
                                s[j]=v[i];
                                j++;
                                continue;
```

```
            }
if(v[i]=='e'){
            s[j]=toupper(v[i]);
            i=i+2;
            j++;
            continue;
}
if(v[i]=='c'){
            s[j]=toupper(v[i]);
            i=i+2;
            j++;
            continue;
}

if(v[i]=='l'){
            s[j]=toupper(v[i]);
            i=i+2;
            j++;
            continue;
}

if(v[i]=='t'){
            s[j]=toupper(v[i]);
            i=i+2;
            j++;
            continue;
}

if(v[i]=='e'){
            s[j]=toupper(v[i]);
            i=i+2;
            j++;
            continue;
}
```

```
            if(v[i]=='s'){

                i++;
                    if(v[i]=='i'){
                        s[j]='S';
                        i++;
                        j++;
                        continue;
                    }

                    if(v[i]=='q'){
                        s[j]='K';
                        i=i+2;
                        j++;
                    }
            }
            if(v[i]=='p'){
                s[j]='P';
                i=i+2;
                j++;
            }
        }
        s[j]=0;
}
```

Izvorni kod datoteke stog.h:

```
#include <stdio.h>
#include <stdlib.h>
// DEFINICIJA STOGA POMOĆU POVEZANOG POPISA

template <typename type>
struct OE
```

```
{
  type x;
  OE<type> *sljedeci;
};

template <typename mytype>
class Stog
{
public:
  OE<mytype> *prvi;

  Stog() { prvi = NULL; }
  ~Stog() { Isprazni(); }
  void Push( mytype pod );
  mytype Pop();
  void Isprazni() { while (!JePrazan()) Pop(); }
  int JePrazan() { if (prvi==NULL) return 1; return 0; }
};

template <typename mytype> void Stog<mytype>::Push( mytype pod )
{
  OE<mytype> *N = new OE<mytype>;
  if (N==NULL) { puts( "STACK OVERFLOW" ); exit( 0 ); return; }
  N->x = pod;
  N->sljedeci = prvi;
  prvi = N;
}

template <typename mytype> mytype Stog<mytype>::Pop()
{
  if (JePrazan()) { puts( "INVALID POP" ); exit( 0 ); return 0; }
  mytype rez = prvi->x;
  OE<mytype> *t = prvi;
  prvi = prvi->sljedeci;
```

```
 delete t;
 return rez;
}
```

Izvorni kod datoteke stog1.h:

```
#include <stdio.h>
#include <stdlib.h>
// DEFINICIJA STOGA POMOĆU NIZA

#define N 1000

template <typename mytype>
class Stog1
{
public:
 int SP;
 mytype *V;

 Stog1() { V = new mytype[N]; SP = 0; }
 ~Stog1() { delete [] V; }
 void Push1( mytype pod );
 mytype Pop1();
 void Isprazni1() { SP = 0; }
 int JePrazan1() { if (SP==0) return 1; return 0; }
};

template <typename mytype> void Stog1<mytype>::Push1( mytype pod )
{
 if (SP==N) { puts( "STACK OVERFLOW" ); exit( 0 ); return; }
 V[SP] = pod;
 SP++;
}
```

```
template <typename mytype> mytype Stog1<mytype>::Pop1()
{
  if (JePrazan1()) { puts( "INVALID POP" ); exit( 0 ); return 0; }
  SP--;
  return V[SP];
}
```