SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Matija Govedić

VIZUALIZACIJA APSTRAKTNIH TIPOVA PODATAKA

ZAVRŠNI RAD

Varaždin, 2013.

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Matija Govedić
Redoviti student
Broj indeksa: 39064/10-R
Smjer: Informacijski sustavi
Preddiplomski studij

VIZUALIZACIJA APSTRAKTNIH TIPOVA PODATAKA

ZAVRŠNI RAD

Mentor: Tihomir Orehovački, mag. inf.

Varaždin, rujan 2013.

Sadržaj

1. Uvod

Jedan od idejnih stupova objektno-orijentiranih programa za razvoj aplikacija su apstraktni tipovi podatka [Wiener, 2000]. Oni su skup struktura i operacija nad tim strukturama koje su opisane na način da su neovisne o njihovoj implementaciji.

Problem koji je potaknuo izradu ovog rada jest nemogućnost pojedinaca da implementiraju neki apstraktni tip koristeći samo definicije funkcija te vlastiti um. Naime problem koji se javlja je taj što mnogi od tih pojedinaca ne mogu zamisliti kako bi to trebalo izgledati, tj. ne mogu si vizualizirati sliku koja bi im pojasnila kako i na koji način bi se trebao posložiti programski kod. U tu svrhu napravljen je ovaj rad čiji je cilj grafički prikazati način izvršavanja operacija apstraktnih tipova podataka. Takvo grafičko prikazivanje implementirano je u programskom jeziku C#. On je u potpunosti objektno - orijentirani programski jezik, razvijen od Microsofta, sa korijenima u C obitelji [Hejlsberg i suradnici, 2004].

Razlog odabira ovog programskog jezika jest njegova pristupačnost korisnicima te mogućnost korištenja već implementiranih procedura za određene apstraktne tipove podataka. Način realizacije te izgled same aplikacije biti će prikazan u praktičnom dijelu.

Da bi se shvatila zadaća i smisao funkcija apstraktnih tipova podataka, ovaj rad započinje teorijskim dijelom. U njemu će biti objašnjen svaki pojedini tip podataka, mehanizmi agregacije i sve ostalo što je potrebno za implementaciju tih funkcija.

2. Teorijski dio

2.1. Tipovi podataka i mehanizmi agregacije

Suština programiranja je logička obrada podataka različitih tipova. Tako tip podataka, osim što definira izgled memorijskog zapisa podatka u memoriju računala, definira i moguće operacije i logičke obrade koje je moguće odraditi nad podatkom. Primjerice, tekstualne ili slovne tipove podataka nije moguće množiti kao što je to moguće sa cjelobrojnim tipovima podataka. Programski jezik C# je objektno orijentirani jezik koji je namijenjen najširoj skupini programera, te ujedno prati i tzv. moderne trendove poput funkcionalnog i dinamičkog programiranja. Ovaj jezik ujedno podržava tipove podatka, ali i mehanizme agregacije.

2.1.1. Tipovi podataka

Tipovi podataka mogu se podijeliti na brojevne, tekstualne, logičke i apstraktne tipove podataka. S druge strane mehanizmi agregacije su polja, znakovni nizovi, zapisi i unije. Ovaj rad bavit će se apstraktnim tipovima podataka, no kako su ostali tipovi sustavni dio implementacije apstraktnih najprije nešto o njima.

Brojevni tipovi

Brojevni ili numerički tipovi podataka spadaju u skupinu jednostavnih tipova podataka. Različiti programski jezici tako definiraju različite tipove podataka. Numerički tipovi, s obzirom na njihov raspon vrijednosti, mogu se podijeliti na [Robinson i suradnici, 2004]:

- byte i sbyte – ovaj tip podataka ima najmanji raspon vrijednosti i to od 0 do 255, odnosno od -128 so +127, ukoliko se radi o sbyte tipu
- decimal – raspon vrijednosti ovog numeričkog tipa nalazi se u sljedećem intervalu: $-7{,}9 * 10^{28} – 7.9*10^{28}$
- double – numerički tip double ima raspon vrijednosti od $5*10^{324}$ do $1.7*10^{308}$
- float – raspon vrijednosti ovog tipa nalazi se u intervalu od $-3.4*10^{38}$ do $3.4*10^{38}$
- int i uint – ovo je cjelobrojni numerički tip podataka čiji je raspon vrijednosti od -2.147.483.648 do 2.147.483.647. Uint ili insigned int je interval istog raspona vrijednosti, ali u intervalu od 0 do 4.294.967.295.

- long i ulong – Long je nešto veći od int tipa podataka, a raspon vrijednosti seže mu od –9.223.372.036.854.775.808 do +9.223.372.036.854.775.807. Raspon vrijednosti ulonga (*eng. unsigned long*) nalazi se u intervalu 0 do 18.446.744.073.709.551.615.
- short i ushort – Raspon vrijednosti ovog brojčanog tipa nalazi se u intervalu od -32.768 do +32.767. Kod ushorta (*eng. unsigned short*), raspon vrijednosti nalazi se u intervalu od 0 do 65.535.

Slovni tipovi

Slovni tipovi predstavljaju znakovni odnosno alfanumerički tip podataka. Taj tip podataka omogućava pohranu slova, interpunkcija i posebnih znakova. Slovni tipovi podataka mogu biti char ili string. Kod char slovnog tipa podataka potrebna memorija je 1 byte što znači da se po ASCII kodu kodira sa 8 bitova. S druge strane tip string predstavlja sekvencu od 0 ili više znakova. Pošto se radi o sekvenci više znakova taj tip podataka spada u skupinu mehanizama agregacije [Robinson i suradnici, 2004].

Ostali tipovi

U skupinu ostalih tipova podataka spadaju [Robinson i suradnici, 2004]:

- Logički tip bool – je jednostavan tip podataka koji može imati dvije vrijednosti. Vrijednost logičke jedinice ili vrijednost logičke nule. Vrijednost logičke jedinice predstavlja vrijednost istine (engl. true) dok vrijednost logičke nule predstavlja vrijednost neistine (engl. false)
- Object – je temeljni tip koji je baza za sve ostale tipove podataka. To podrazumijeva da su svi objekti ili varijable nekog tipa ujedno i tipa objekt.

Apstraktni tipovi

Apstraktni tipovi podataka su tipovi podataka koje osmišljava sam programer, a koji se zadaju na način da se navede jedan ili više tipova podataka te jedne ili više operacija – u ovom slučaju funkcija. Kod apstraktnih tipova podataka su zadani jedan ili više tipova podataka, te jedna ili više operacija. Više o apstraktnim tipovima podataka i vrstama istih biti će u nastavku poglavlja.

2.2. Mehanizmi agregacije

U ovu skupinu podrške programskog jezika C# spadaju [Microsoft MSDN Library, 2012]:

- polje - mehanizam agregacije kojim se od jednostavnijih tipova podataka tvori složeniji tip podataka. To znači da je to ujedno niz varijabli istog tipa koje su imenovane zajedničkim identifikatorom, a nalaze se na uzastopnim memorijskim lokacijama.
- znakovni nizovi - predstavljaju jednodimenzionalno polje čiji su članovi znakovi (*eng. char*). Sadržaj se u znakovni niz unosi između para dvostrukih navodnika i taj sadržaj predstavlja znakovni niz (*eng. string*)
- zapisi – čine još jedan način udruživanja manjih cjelina u veće strukture. Predstavlja više elemenata koji ne moraju biti istog tipa ali su pohranjene na uzastopnim adresama. Ideja je da svaki pojedini element bude komponenta zapisa. Valja naglasiti kako se zapisi mogu kombinirati u polja.
- unije – također mehanizam agregacije koja omogućuje da se alternativni podaci čuvaju u istom memorijskom prostoru.

2.3. Apstraktni tipovi podataka

Ovakav tip podataka je tip koji najčešće osmišljava programer. Apstraktni se tipovi podataka zadaju navođenjem jednog ili više tipova podataka te jedne ili više operacija, odnosno funkcija. Kada želimo definirati apstraktni tip podataka na računalu oni se mogu definirati na različite načine. Ukoliko se želi provesti odnosno**,** realizirati tip podataka u nekom programu, implementacija će se sastojati od definicije za strukturu podataka kojom se prikazuju podaci iz apstraktnog tipa podataka te različitih potprograma i funkcija kojima će se operacije iz apstraktnog tipa podataka ostvariti pomoću odabranih algoritama. Valja naglasiti kako je moguće osmisliti više različitih implementacija za isti apstraktni tip podataka. Različite implementacije razlikovat će se po tome što će koristiti različite strukture za prikaz podataka. Ujedno će koristiti i različite algoritme kojima se izvršavaju operacije [Sedgewick, 2002].

Neki najčešći apstraktni tipovi podataka su: lista, stog, red, općenito stablo, binarno stablo te polinom. Njihov detaljan opis slijedi u nastavku ovog poglavlja.

2.3.1. Lista

U računalnoj znanosti, lista je apstraktni tip podataka koji se implementira kao konačni slijed elemenata. Svaka instanca vrijednosti u listi se obično naziva ulazom ili elementom liste, te ako se ista pojavljuje više puta ona se svaki put gleda kao drugačiji, tj. novi element. U C# programskom jeziku postoji već prije implementirana lista sa svim svojim procedurama te je njeno korištenje svedeno na najjednostavniju moguću razinu [Aho i suradnici, 1987].

Primjer inicijalizacije:

```
List<int> mojaLista= new List<int>();
```

Iako koncept liste pomalo podsjeća na polje, jer se u njoj nalaze podaci kojima je pridodan određeni indeks(redni broj elementa u listi), postoje i određene razlike:

1. Lista je dinamičke duljine,
2. Pristup elementima je različit negoli u polju

Dinamička duljina

Dinamička duljina liste označava da se prilikom inicijalizacije ne navodi njena fiksna duljina, nego se ona mijenja prilikom dodavanja novog elementa, tj. na početku je njena duljina 0 dok nakon unosa n-tog elementa duljina iznosi n.

Različit pristup elementima

Kako se elementima u polju pristupa pomoću indeksa, tj. rednog polja elementa počevši od 0, te se tako preskaču cijeli nizovi elemenata nailazimo na bitnu razliku između polja i liste. U listi se elementima pristupa sekvencijalno, odnosno da bi učitali vrijednost n-tog elementa moramo prije toga proći kroz n-1 elemenata počevši od prvog nultog elementa.

Operacije na listama

Operacije koje je moguće izvršiti na listama su kako slijedi [Keogh, 2004]:

- FirstL(L) - procedura koja nam vraća prvi element liste, tj. element sa indeksom 0. Ako je ona prazna vraća nam pogrešku.
- EndL(L) - Služi za vraćanje pozicije iza zadnjeg elementa liste.
- NextL(p,L) – Funkcija koja vraća sljedbenika unesenog elementa p. Ako je p=EndL(L) javlja nam pogrešku.
- PreviousL(p,L) – Funkcija koja vraća prethodnika unesenog elementa p. Ako je p=FirstL(L) javlja nam pogrešku.
- LocateL(x,L) – Procedura koja vraća prvi element sa vrijednošću x. Ako on ne postoji vraća se EndL(L).
- InsertL(x,p,L) – Dodaje novi element sa vrijednošću x na poziciju p. Svi elementi koji su u listi na poziciji većim ili jednakim p se pomiču za jedno mjesto dalje.
- DeleteL(p,L) – Briše se element na poziciji p iz liste. Svi elementi koji su se nalazili iza tog elementa u listi se pomiču za jedno mjesto unaprijed.
- RetrieveL(p,L) – Funkcija koja vraća vrijednost elementa na poziciji p.
- DeleteAllL(L) - Briše sve elemente iz liste L
- InitL(L)- služi za iniciranje liste na praznu listu.

2.3.2. Stog

Stog je apstraktni tip podataka koji pohranjuje elemente istog tipa. Specifičan je po tome što se operacije dodavanja i brisanja novog elementa izvršavaju na njegovom vrhu. Poveznica između tih operacija je ta što kod stoga zadnji element koji se upiše će iz njega biti prvi uklonjen (eng. LIFO: Last In First Out). Iz toga slijedi da je jedini način pristupanja n-tom elementu uklanjanje svih elemenata iznad njega, tj. potrebno je ukloniti n-1 element [Kruse i Ryba, 2000].

Kako postoji više načina implementacije stoga, moguće je da on bude ograničen brojem elemenata koje će sadržavati. Znači, ako je stog potpuno popunjen te nema više mjesta za novi element tada dolazi do prelijevanja (engl. overflow). Također postoji mogućnost da se želi izbaciti element iako u samom stogu više nema ničega pa tada dolazi do podlijevanja(engl. underflow).

Operacije koje se nad stogom mogu izvršiti su sljedeće [Dale, 2003]:

- TopS(S) – procedura koja vraća vrijednost elementa koji se nalazi na samom vrhu stoga.
- PushS(x,S) - dodaje novi element sa vrijednošću x na vrh stoga.
- PopS(S) - funkcija koja briše element s vrha stoga.
- InitS(S) – inicira prazan stog
- IsEmptyS(S)- logička funkcija koja odgovara sa „true“ ako je stog prazan ili sa „false“ ako se u njemu nalaze elementi.

2.3.3. Red

Red je apstraktni tip podataka koji pohranjuje elemente istog tipa. Njegova specifičnost je u tome što je on zapravo FIFO lista (engl. First In First Out) te su operacije čitanja i brisanja dozvoljene samo na čelu reda, a operacija unosa novog elementa je moguća samo na njegovom začelju. To zapravo znači da element koji je tek unesen može biti izbrisan tek kad izbrišemo sve elemente koje smo ranije unijeli. Iz tog razloga red možemo nazvati specifičnom listom sa ograničenim skupom operacija [Aho i suradnici, 1987].

Najčešća implementacija reda jest implementacija pomoću cirkularnog polja te dva kursora od kojih jedan pokazuje na čelo reda, a drugi na njegovo začelje. U C# programskom jeziku red je već prije definiran zajedno sa svojim operacijama te je potrebna samo njegova inicijalizacija.

Primjer inicijalizacije:

```
Queue<string> mojRed= new Queue<string>();
```

U kontekstu apstraktnog tipa podataka red moguće je implementirati sljedeće funkcije [Dale, 2003]:

- FrontQ(Q) - procedura koja vraća vrijednost elementa koji se nalazi na čelu reda.
- EnQueueQ(x,Q) - dodaje novi element na začelje reda
- DeQueueQ(Q) - funkcija koja briše element sa čela reda.
- InitQ(Q) – inicira prazan red.
- IsEmptyQ(Q) – logička funkcija koja vraća „true“ ako je red prazan ili sa „false“ ako se u redu nalaze elementi.

2.3.4. Općenito stablo

Općenito stablo je apstraktni tip podataka koji pohranjuje elemente istog tipa, a nazivamo ih čvorovi. Kako je njegov prikaz zapravo graf možemo ga nazvati specifičnom vrstom grafova. Ono sadrži korijen, naravno i njegovu vrijednost, te podstabla kao njegovu djecu. Djetetom nekog čvora nazivamo čvor koji je povezan sa prethodnim, ali se nalazi na nižoj hijerarhijskoj razini, dok roditeljem možemo nazvati čvor koji se nalazi na većoj hijerarhijskoj razini. Stupanj nekog čvora definira se kao broj njegovih podstabala dok je stupanj stabla maksimalan stupanj svih njegovih čvorova. Put od čvora x do čvora y je niz čvorova $x=x_1,...,x_n=y$ sa svojstvom da su svi čvorovi potomci čvora x. Potomak čvora x je svako njegovo dijete. Prema tome ako je čvor y potomak čvora x , onda su i djeca od y potomci čvora x [Putambekar, 2007].

Obilazak stabla

Obilazak stabla je algoritam kojim u konačnom vremenu posjećujemo čvorove stabla tako da svaki čvor posjetimo točno jedanput. Njegova namjena je obrada vrijednosti nad svim čvorovima te uspostavljanje linearnog uređaja među njima.

Najčešće se implementacija obilaska stabla zadaje rekurzivno te postoji više načina od kojih se češće upotrebljavaju:

- preorder
- inorder
- postorder

Preorder je način obilaska stabla u kojem prvo posjećujemo korijen stabla, a zatim podstabala krećući od lijeva prema desno. Postorder način obilaska posjećuje najprije podstabala od lijeva prema desno te na kraju korijen, a inorder najprije posjećuje najljevije podstablo zatim korijen te zatim preostala podstabla krećući se također od lijeva prema desno [Aho i suradnici, 1987].

Operacije na općenitom stablu su sljedeće [Aho i suradnici, 1987]:

- ParentT(n,T) - procedura koja vraća roditelja od čvora n, a ukoliko je taj čvor korijen vraća se pogreška,
- FirstChildT(n,T) - funkcija vraća prvo, znači najljevije dijete čvora n, a ukoliko taj čvor nema dijete (ako je on list) vraća se pogreška,
- NextSiblingT(n,T) - vraća sljedećeg brata čvora n, a ukoliko on ne postoji vraća se pogreška,
- LabelT(n,T) - procedura koja vraća vrijednost čvora n,
- RootT(T) - funkcija koja vraća vrijednost korijena stabla,
- CreateT(x,n,T) - dodaje novi čvor sa vrijednošću x kao dijete čvora n.
- ChangeLabelT(x,n,T) - zamjenjuje vrijednost čvora n sa vrijednošću x.
- DeleteT(n,T) - procedura koja briše čvor n i s time sve njegove potomke iz stabla T,
- InitT(x,T) - inicijalizira stablo s korijenom x.

2.3.5. Binarno stablo

Apstraktni tip podataka kojeg možemo definirati kao uređenu trojku (x,L,D), pri čemu su L i D binarna stabla ili su prazni. Ono nije specijalizacija općenitog stabla nego apstraktni tip podataka sa drugačijim svojstvima. Glavna razlika između binarnog te općenitog stabla jest ta što čvor u binarnom stablu može imati samo lijevo i desno dijete. Ta razlika utjecala je i na drugačiju implementaciju ovog stabla. Njegova glavna prednost je u tome što je veoma jednostavno za pretraživanje [Preiss, 2004].

Implementacija binarnog stabla pomoću polja sastoji se od jednodimenzionalnog polja u kojem se spremaju vrijednosti te jedne logičke varijable koja pokazuje da li je određeni element u polju iskorišten ili ne.

Operacije na binarnom stablu su kako slijedi [Aho i suradnici, 1987]:

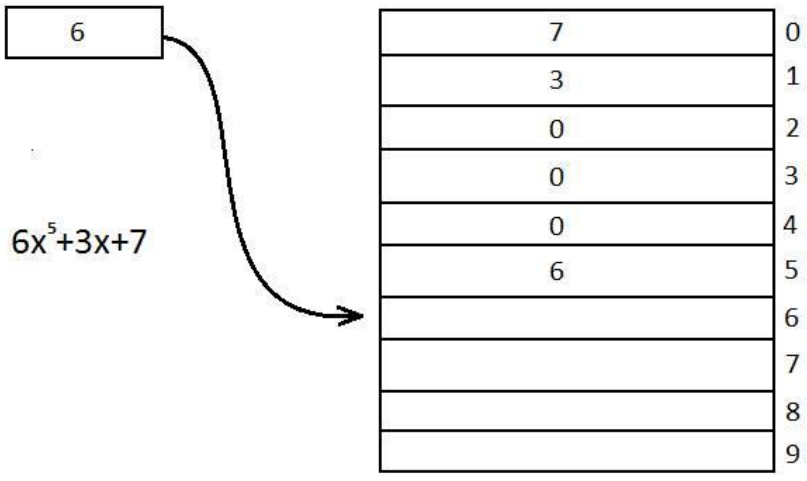
- ParentB(n,T) - vraća roditelja čvora n, a ukoliko je taj čvor korijen vraća grešku,
- LeftChildB(n,T) - procedura koja vraća lijevo dijete čvora n, a ukoliko ono ne postoji vraća pogrešku,

- RightChildB(n,T) - procedura koja vraća lijevo dijete čvora n, a ukoliko ono ne postoji vraća pogrešku,
- LabelB(n,T) - vraća vrijednost čvora n,
- ChangeLabelB(x,n,T) - mijenja vrijednost čvora n na vrijednost x,
- RootB(T) - procedura koja vraća korijen stabla,
- CreateLeftB(x,n,T) - dodaje lijevo dijete čvoru n sa vrijednošću x, a ukoliko već postoji lijevo dijete javlja se pogreška,
- CreateRightB(x,n,T) - dodaje desno dijete čvoru n sa vrijednošću x, a ukoliko već postoji desno dijete javlja se pogreška,
- DeleteB(n,T) - briše čvor n i sve njegove potomke sa stabla te
- InitB(x,T) - inicijalizira stablo sa vrijednošću korijena x.

2.3.6. Polinom

Apstraktni tip podataka koji nam služi za pohranu te manipulaciju polinomom čuvajući njegove značajke. One se čuvaju na način da se koeficijenti spremaju na onu poziciju u polju koja je određena potencijom uz x [Sane i Deshpande, 2006].

Način na koji se polinom sprema u polje prikazan je na slici 1.



Slika 1. Prikaz implementacije polinoma

Apstraktni tip podatka polinom moguće je implementirati pomoću sljedećih funkcija [Weiss, 1996]:

- Zero(p) - procedura koja inicira nul-polinom,
- IsZero(p) - logička funkcija koja odgovara na pitanje da li je polinom nul-polinom,
- Coef(p, pot) - vraća koeficijent u polinomu uz potenciju p,
- Attach(p, pot, val) - pridružuje vrijednost određenom koeficijentu,
- Degree(p) - procedura koja vraća stupanj polinoma,
- Add(p1, p2, p3) - procedura koja zbraja dva polinoma i
- Mult(p1, p2, p3)-funkcija koja množi dva polinoma.

3. Praktični dio

Aplikacija namijenjena vizualizaciji apstraktnih tipova podataka izrađena je u programskom jeziku C# te je njena osnovna namjena olakšati samostalnu implementaciju obrađenih apstraktnih tipova podataka.

3.1. Početni obrazac

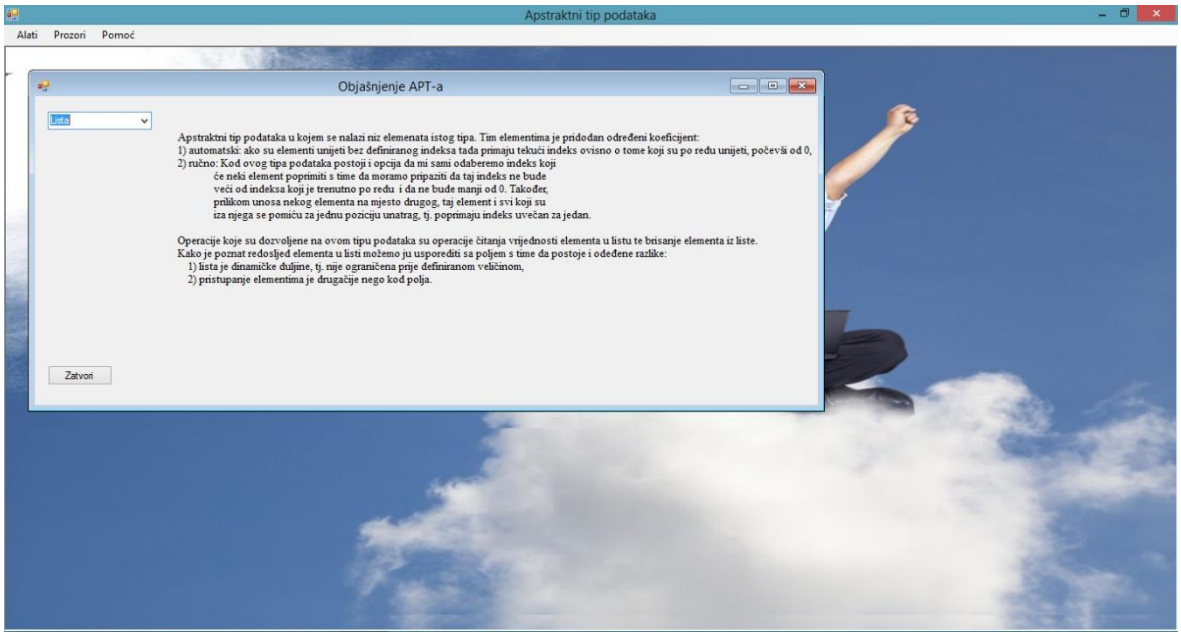
Prilikom pokretanja aplikacije pojavi se prozor kao na slici 2. On se prostire kroz cijelu dužinu ekrana, tj. „maksimiziran“ je kako bi korisnik imao ljepši pregled animacija. U gornjem dijelu ovog obrasca nalazi se menuStrip koji nam omogućuje odabir funkcionalnosti aplikacije. Na slici se također može vidjeti način podjele aplikacije na: animacije, teoriju te na popis funkcija. Pojam *Animacije* u aplikaciji podrazumijeva vizualni prikaz funkcija apstraktnih tipova podataka u ovisnosti o izboru korisnika sa izbornika. S time je zapravo ostvarena i namjena ove aplikacije. Teorija podrazumijeva opis pojedinog apstraktnog tipa podataka dok popis funkcija objašnjava svaku funkciju zasebno i način njenog korištenja. Slika koja je umetnuta kao pozadina na ovom obrascu preuzeta je radi estetskog dojma[1].

Slika 2. Početni obrazac

[1] Preuzeto sa: http://www.novilist.hr/Vijesti/Hrvatska/Baustela-nove-generacije-mladi-hrvatski-strucnjaci-na-radu-u-inozemstvu

3.2. Teorija

Iz slike 3 se može vidjeti kako se svaki novootvoreni obrazac iz izbornika otvara u početnoj te da se ovaj obrazac sastoji od jednog comboBox-a, gumba *Zatvori* i od jedne oznake. ComboBox nam omogućava izbor pojedinog apstraktnog tipa podataka te se ovisno o izboru u oznaci ispisuje tekst. Ta funkcionalnost realizirana je na način da se provjerava koji je indeks u comboBox-u izabran jer je svakom pojmu, odnosno nazivu pojedinog apstraktnog tipa podataka pridružen broj. Gumb *Zatvori* ima veoma jednostavnu funkcionalnost da zatvara obrazac na kojem se trenutno nalazi. Najveći problem u ovom dijelu aplikacije bio je to što se tekst, zapravo objašnjenja apstraktnih tipova koja su gotovo ista kao i u teorijskom dijelu, morao unijeti u oznaku u odgovarajućem obliku, a programski jezik C# nema ugrađen poseban tekst editor za tu svrhu.



Slika 3. Obrazac Teorija

Iz tog razloga prikazani tekst u programskom kodu izgleda ovako:

```
if (comboBox1.SelectedIndex == 0)
            {

                label1.Text = @"
```

Apstraktni tip podataka u kojem se nalazi niz elemenata istog tipa. Tim elementima je pridodan određeni koeficijent:

1) automatski: ako su elementi unijeti bez definiranog indeksa tada primaju tekući indeks ovisno o tome koji su po redu unijeti, počevši od 0,

2) ručno: Kod ovog tipa podataka postoji i opcija da mi sami odaberemo indeks koji će neki element poprimiti s time da moramo pripaziti da taj indeks ne bude veći od indeksa koji je

trenutno po redu i da ne bude manji od 0. Također, prilikom unosa nekog elementa na mjesto drugog, taj element i svi koji su iza njega se pomiču za jednu poziciju unatrag, tj. poprimaju indeks uvećan za jedan.

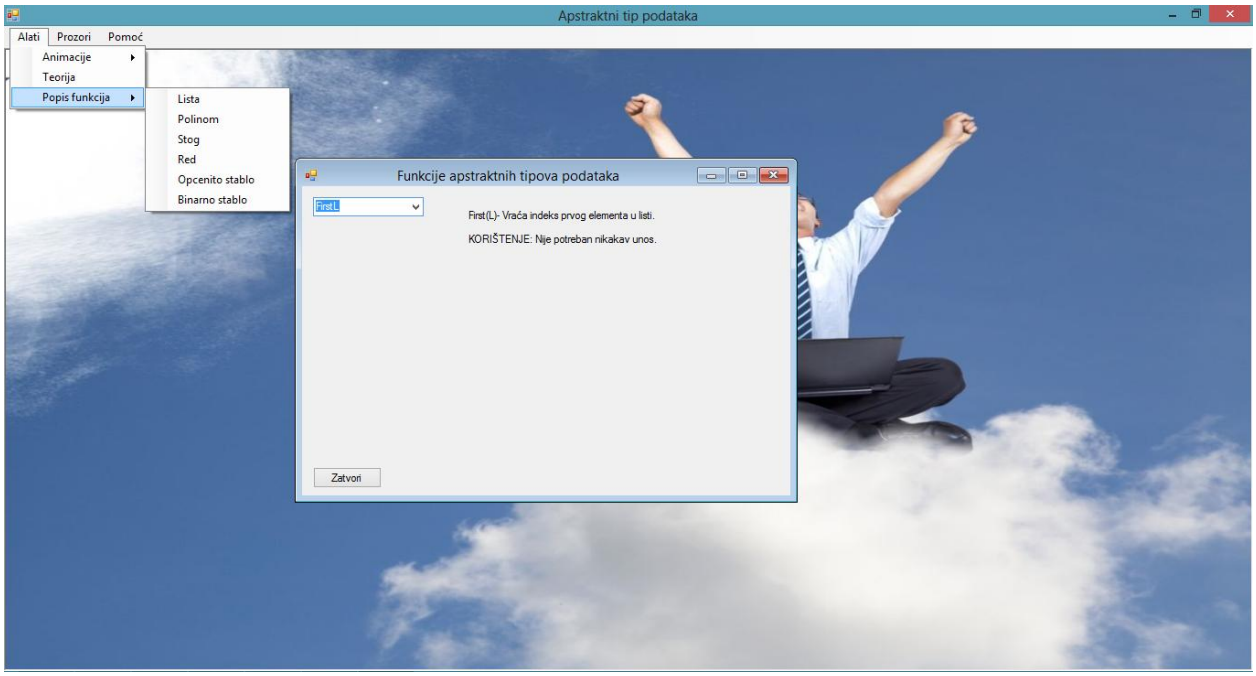
Operacije koje su dozvoljene na ovom tipu podataka su operacije čitanja vrijednosti elementa u listu te brisanje elementa iz liste.
Kako je poznat redoslijed elementa u listi možemo ju usporediti sa poljem s time da postoje i određene razlike:

1) lista je dinamičke duljine, tj. nije ograničena prije definiranom veličinom,
2) pristupanje elementima je drugačije nego kod polja."**;**

}

3.3. Popis funkcija

Ovaj dio aplikacije veoma je sličan prethodnom dijelu pod nazivom *Teorija*. Slika 4. prikazuje izbornik koji nam služi za odabir pojedinog apstraktnog tipa podataka te se u ovisnosti o odabiru pojavljuje obrazac s pripadajućim funkcijama. Obrazac je sastavljen od istih dijelova kao i prethodno objašnjen te je razlika samo u tome što se ispisuje drugačiji tekst i u tome što se u comboBox-u nalaze druge opcije.



Slika 4. Funkcije apstraktnih tipova podataka

Važno je napomenuti da su sva objašnjenja funkcija, znači svi apstraktni tipovi podataka realizirani na istom obrascu. To je omogućeno kroz poziv obrasca sa drugačijim parametrom, npr. kada u meniju izaberemo polinom obrazac se poziva sa parametrom polinom, za stog parametar je stog, itd.

Programski kod poziva funkcije sa parametrom izgleda ovako:

```
private void polinomToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            pokreni(new frmProgKod("polinom"));
        }

        private void stogToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            pokreni(new frmProgKod("stog"));
         }
```

Kada funkcija primi taj parametar, ona ga uspoređuje i prema njemu unosi stavke u comboBox:

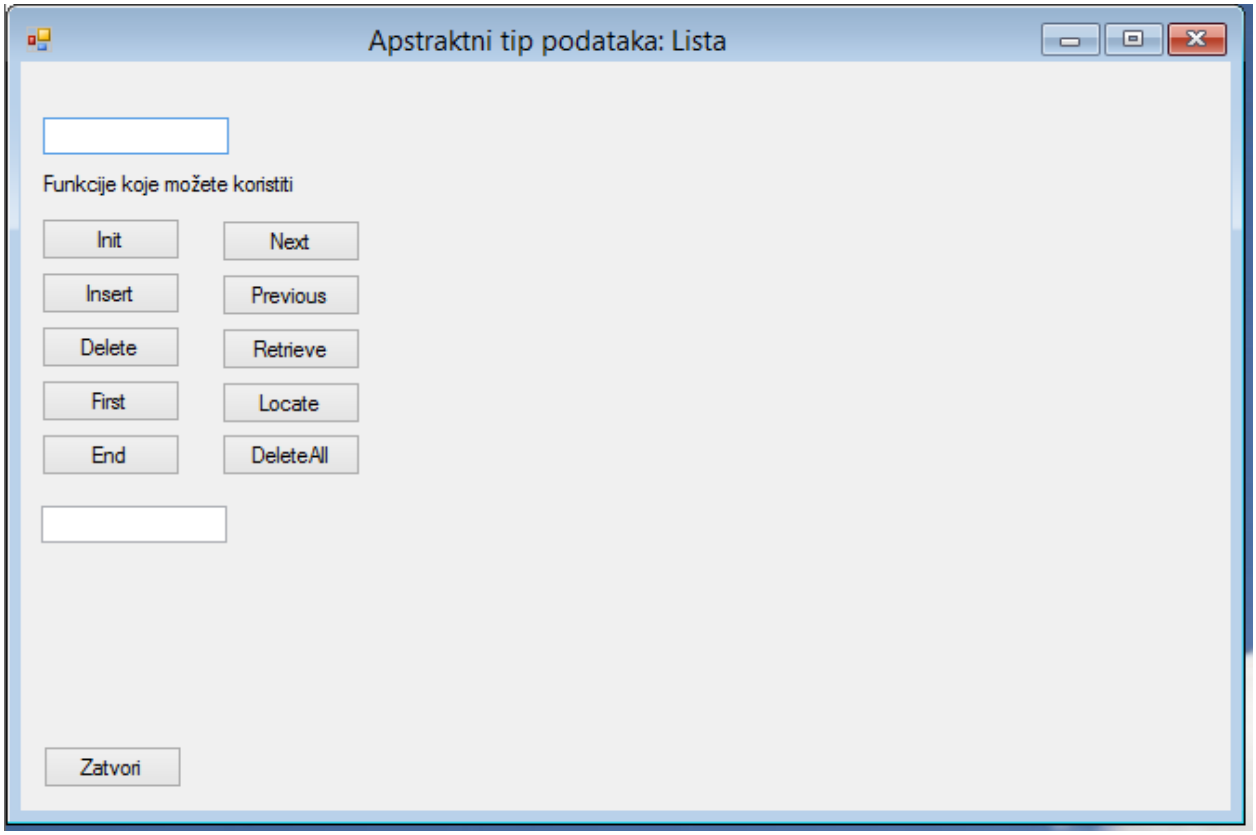
```
if (string.Compare(odabir, "stog", false) == 0) {
                 comboBox1.Items.Add("TopS");
                comboBox1.Items.Add("PushS");
                comboBox1.Items.Add("PopS");
                comboBox1.Items.Add("InitS");
                comboBox1.Items.Add("IsEmptyS");
            }
```

3.4. Animacije

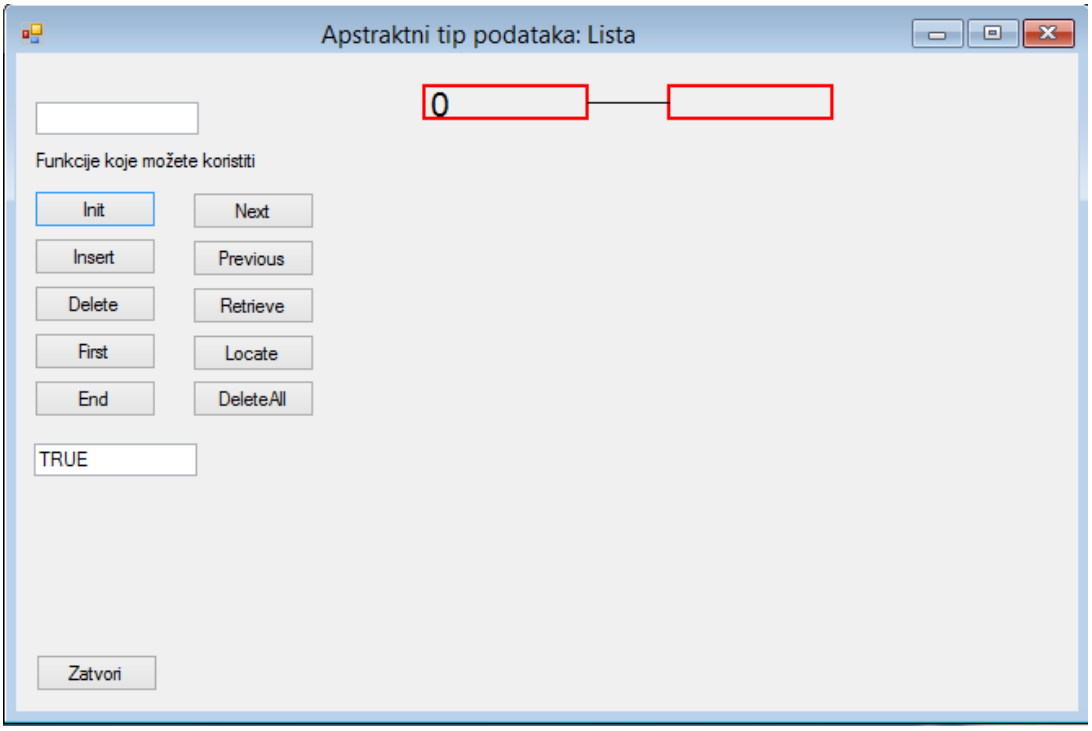
U ovom dijelu rada dolazimo do vizualnog prikaza načina rada funkcija apstraktnih tipova podataka.

3.4.1. Lista

Slika 5 prikazuje da se na formi lista nalaze 10 gumbi koji predstavljaju funkcije za manipuliranje listom. Također možemo uočiti da se nalaze 2 textboxa od kojih gornji služi za unos vrijednosti elementa, dok donji služi za ispis rezultata određene funkcije. Također isto tako postoji gumb *Zatvori* za zatvaranje obrasca.



Slika 5. Obrazac lista



Slika 6. Vizualizacija funkcije Init

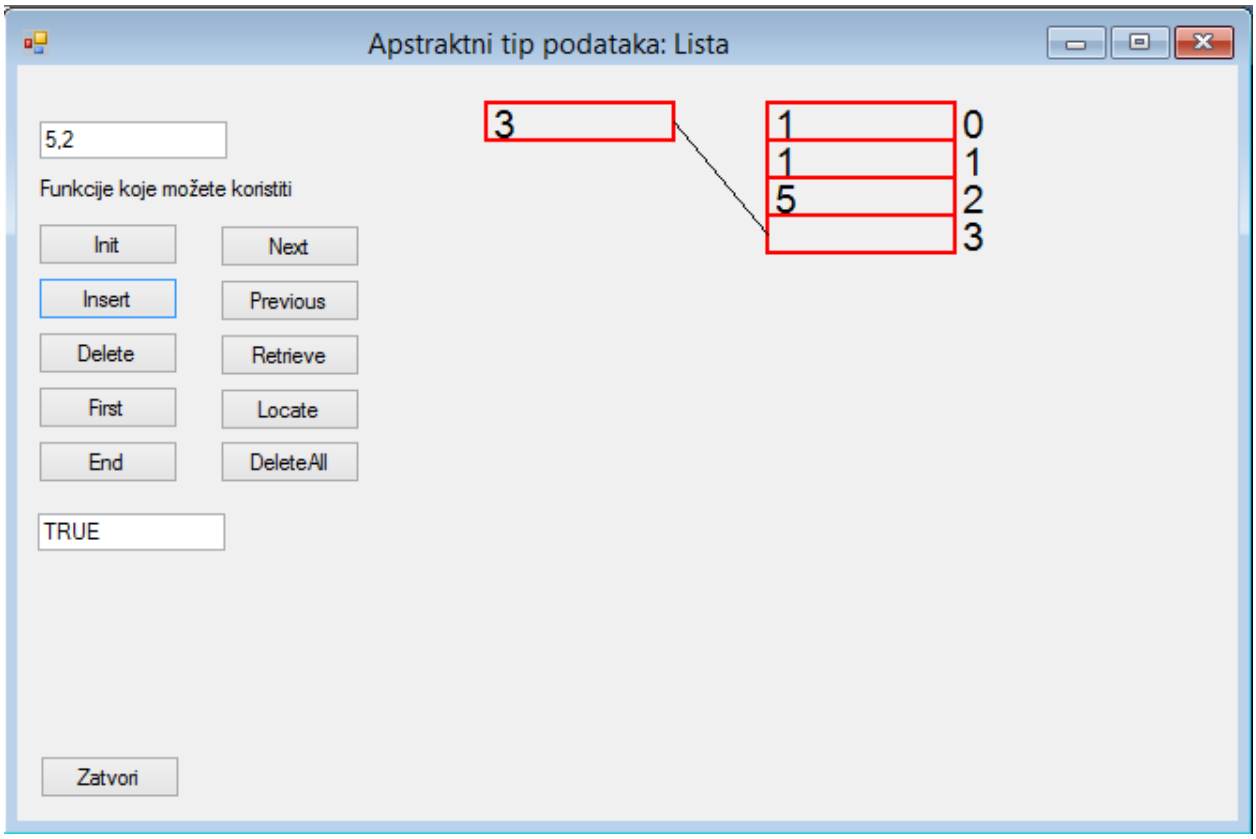
Na slici 6 prikazan je način vizualizacije funkcije *Init* koja služi za inicijaliziranje liste. njena vizualizacija sastoji se od dva pravokutnika, brojke te linije koja ih povezuje. Prvi, odnosno lijevi pravokutnik označava kursor, tj. prikazuje koliko je trenutno elemenata u polju dok drugi, prazni pravokutnik označava da u listi još nema ničega. Za prikaz pravokutnika, koji je sastavni dio svake vizualizacije korišten je programski kod koji je preuzet ali i prerađen sa Microsoftove stranice MSDN[2] (Microsoft Developer Network, 2003):

```
private void DrawIt(int x, int y, int velicina)
        {
            System.Drawing.Graphics graphics = this.CreateGraphics();
            System.Drawing.Rectangle rectangle = new
System.Drawing.Rectangle(
              x, y, 100, 20);
            System.Drawing.Pen mojaOlovka;
            mojaOlovka = new
System.Drawing.Pen(System.Drawing.Color.Red, velicina);
            graphics.DrawRectangle(mojaOlovka, rectangle);
        }
```

Funkcija *DrawIt(x,y,velicina)* na svom početku generira objekt *graphics* te pomoću tog objekta te objekta *mojaOlovka* i *rectangle* iscrtava na poziciju x,y pravokutnik veličine 100, 20 te debljine *velicina*. Također za prikaz znakova, odnosno brojeva na samoj formi korišten je programski kod koji je preuzet sa iste stranice[3] te također prerađen za potrebe ovog završnog rada:

```
public void DrawString(string tekst, float x, float y)
        {
            System.Drawing.Graphics formGraphics =
this.CreateGraphics();
            string drawString = tekst;
            System.Drawing.Font drawFont = new
System.Drawing.Font("Arial", 16);
            System.Drawing.SolidBrush drawBrush = new
System.Drawing.SolidBrush(System.Drawing.Color.Black);

            System.Drawing.StringFormat drawFormat = new
System.Drawing.StringFormat();
            formGraphics.DrawString(drawString, drawFont, drawBrush,
x, y, drawFormat);
            drawFont.Dispose();
            drawBrush.Dispose();
            formGraphics.Dispose();
        }
```

[2] http://msdn.microsoft.com/en-us/library/aa287521(v=vs.71).aspx, pristupano 22.8.2013.
[3] http://msdn.microsoft.com/en-us/library/aa287524(v=vs.71).aspx, pristupano 22.8.2013.



Slika 7. Prikaz aplikacije nakon korištenja funkcije Insert

Slika 7 prikazuje ponašanje, odnosno sam prikaz aplikacije nakon korištenja funkcije *Insert*. Da bi listu prikazali kao polje korišten je programski kod koji iscrtava pravokutnike ovisno o broju elemenata u listi te ispisuje vrijednost elementa pomoću funkcije *DrawString(tekst, x, y)*

Prilikom svakog poziva funkcije *nacrtajListu()* pomoću metode *Clear* čisti se obrazac od prethodnog crteža, odnosno vizualizacije na način da ju ispunjavamo predefiniranom bojom koju svaki obrazac ima na svojem početku. Zatim se iscrtava pravokutnik na poziciji 250, 20 koji služi za upis vrijednosti kursora, te dolazimo do for petlje koja se kreće od 1 pa do trenutnog broja elemenata u listi. For petlja kreće od 1 a ne od 0 iz tog razloga što ona zapravo služi za iscrtavanje pravokutnika i vrijednosti elementa u listi, te se pomoću nje određuje početna pozicija objekta koji će se iscrtati.

Programski kod funkcije *nacrtajListu()* izgleda ovako:

```
public void nacrtajListu() {
            System.Drawing.Graphics grafika =
this.CreateGraphics();
```

```
            grafika.Clear(DefaultBackColor);
            DrawIt(250, 20, 2);
            for (int i = 1; i <= brojacElemenata; i++) {
                DrawIt(400, 20*i, 2);
                DrawString(mojaLista[i-1].ToString(), 400, i*
20);
                DrawString((i-1).ToString(), 500, i * 20);
            }
            DrawIt(400, 20 * (brojacElemenata + 1), 2);
            DrawString(brojacElemenata.ToString(), 500,
(brojacElemenata + 1) * 20);
            DrawString(brojacElemenata.ToString(), 250, 20);
            nacrtajLiniju(350, 30, 400, 20 *(brojacElemenata+1)
+ 10);
        }
```

Kako kursor mora pokazivati na prazno mjesto u listi koji se nalazi iza zadnjeg elementa, zadnje četiri linije u ovom programskom kodu imaju upravo tu svrhu. One iscrtavaju zadnji pravokutnik te povezuju sa linijom taj, zadnji pravokutnik sa kursorom.

Sve preostale procedure, osim *Locate* koja koristi timer, u obrascu lista realizirane su kombinacijom funkcija *nacrtajListu, DrawIt* i *DrawString*. Funkcija *Locate* drugačija je po tome što koristi timer kako bi svakih 100 milisekundi iscrtala novu sliku i tako korisniku dala predodžbu animacije. Ona zapravo iscrtava listu na način da koristi funkciju *nacrtajListu* i liniju koja nam pokazuje kojem se elementu trenutno ispituje vrijednost.

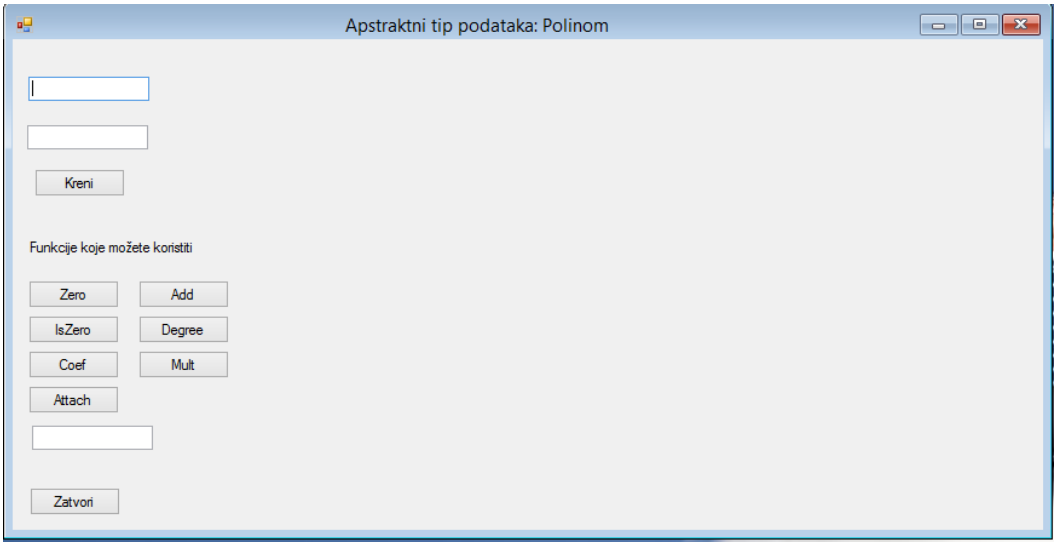
Realizacija funkcije timer 2 izgleda ovako:

```
private void timer2_Tick(object sender, EventArgs e)
        {

            nacrtajLiniju(530, 20 * (brojac + 1) + 10, 560, 20
* (brojac + 1) + 10);
            DrawIt(400, 20 * (brojac + 1), 4);
            timer1.Start();

            if (mojaLista[brojac] == uneseno2) {
                DrawIt(400, 20 * (brojac + 1), 4);
```

```
                DrawString(mojaLista[brojac].ToString(), 565,
20 * (brojac+ 1));
                textBox3.Text = mojaLista[brojac].ToString();

                timer2.Stop();
                timer1.Start();
            }
            if (brojac == brojacElemenata - 1) {

                timer2.Stop();
                timer1.Start();
            }
            brojac++;
        }
```

3.4.2. Polinom

Slika 8 prikazuje smještaj elemenata na obrascu polinom. Taj obrazac sastoji se od tri textboxa, 9 gumba i jedne oznake sa fiksnim tekstom. Prvi ili gornji textbox služi za upis vrijednosti prvog polinoma na kojem se mogu obavljati sve operacije, dok drugi služi za upis vrijednosti drugog polinoma kako bi se korisniku prikazale i procedure *Add* te *Mult*. Zadnji tekstbox, onaj koji se nalazi sasvim pri dnu forme, služi za ispis rezultata određene procedure. Slika također prikazuje i gumb *Kreni* koji nema naziv uobičajene funkcije za polinom, a on nam služi za početno iscrtavanje prvog polinoma.

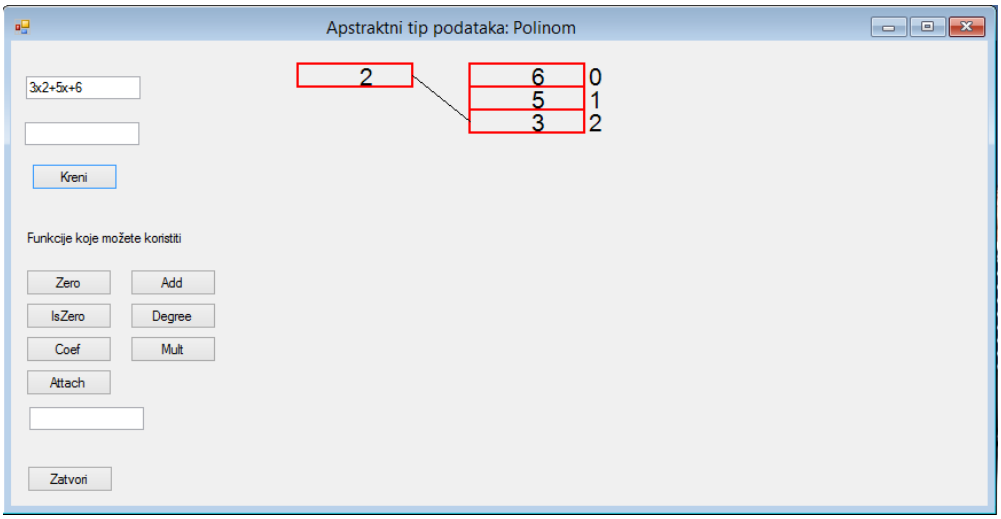


Slika 8. Obrazac polinom

Kako se polinom unosi u ovoj aplikaciji unosi u formatu axn + bx(n-1) + c potrebno je na samom početku ovog dijela aplikacije odvojiti koeficijente te pripadajuće potencije bez x-eva. To je realizirano pomoću funkcije *izvadiBroj(string tekst, lista a, lista b)*, gdje parametar tekst označava polinom upisan u prvi tekstbox, lista a označava strukturu kamo će se upisati koeficijenti te lista b koja označava mjesto upisa potencija. Implementacija te procedure realizirana je na ovaj način:

```
public void izvadiBroj(string upTekst,List<int> xxx, List<int> yyy){
            string[] upisaniTekst = upTekst.Split('+');
            string[] brojevi;
            foreach (string r in upisaniTekst) {
                brojevi = r.Split('x');

                if (brojevi.Length > 1)
                {
                    if (brojevi[0] == "")
                    {
                        xxx.Add(1);
                    }
                    else
                    {
                        xxx.Add(int.Parse(brojevi[0]));
                    }
                    if (brojevi[1] == "")
                    {
                        yyy.Add(1);
                    }
                    else
                    {
                        yyy.Add(int.Parse(brojevi[1]));
                    }
                }
                else {
                    xxx.Add(int.Parse(brojevi[0]));
                    yyy.Add(0);
                }
            }}
```

Prema rezultatima iz te funkcije iscrtava se polinom na način da se koeficijenti upisuju u pravokutnike koji se izrađuju na formi pomoću prije objašnjene procedure *DrawIt*, dok se potencije ispisuju sa desne strane pravokutnika. Također je bilo potrebno napraviti još jedan pravokutnik, čija je zadaća da u sebi sadrži kursor na najveću potenciju u polinomu da bi se već na prvi pogled dao zaključiti stupanj polinoma.



Slika 9. Prikaz unesenog polinoma u obrazac

Iscrtavanje kao na slici 9 realizirano je pomoću procedure *nacrtajPolje*, koja na svom početku briše prethodnu sliku na obrascu. Zatim traži maksimalnu potenciju u unesenom polinomu kako bi imala broj pravokutnika koje treba iscrtati. Početna pozicija svakog pravokutnika dobiva se tako da se pomnoži trenutna vrijednost brojača u for petlji sa pozicijom prvog.

Funkcije *nacrtajPolje()* implementirana je na sljedeći način:

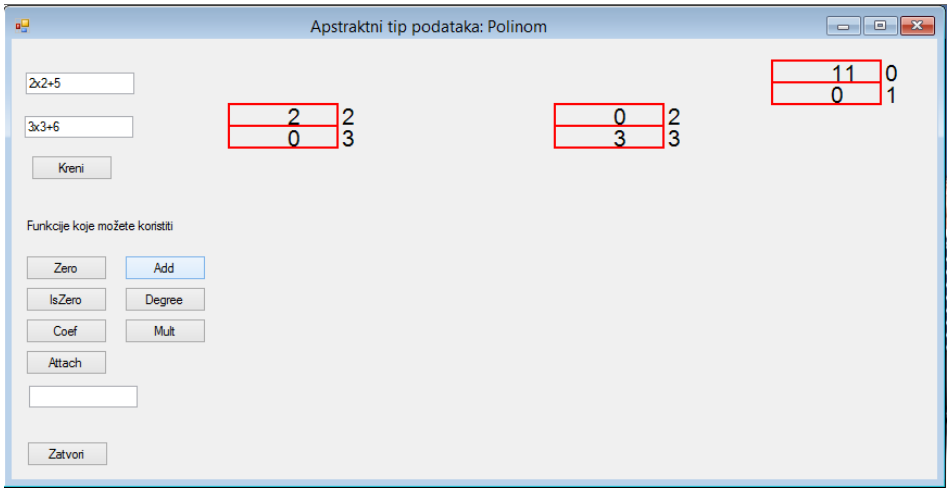
```
public void nacrtajPolje() {
            System.Drawing.Graphics grafika = this.CreateGraphics();
            grafika.Clear(DefaultBackColor);
            foreach (int k in b)
            {
                if (k > max) max = k;
            }
            for (int i = 1; i <= max + 1; i++)
            {
                DrawIt(400, i * 20, 2);
                string tekstZaIspis = (i - 1).ToString();
                DrawString(tekstZaIspis, 500, i * 20);
```

```
                bool ispisNule = true;
                for (int j = 0; j < b.Count; j++)
                {
                    if (b[j] == i - 1)
                    {
                        DrawString(a[j].ToString(), 450, i * 20);
                        ispisNule = false;
                    }
                }
                if (ispisNule == true)
                {
                    DrawString(0.ToString(), 450, i * 20);
                }
            }
        }
```

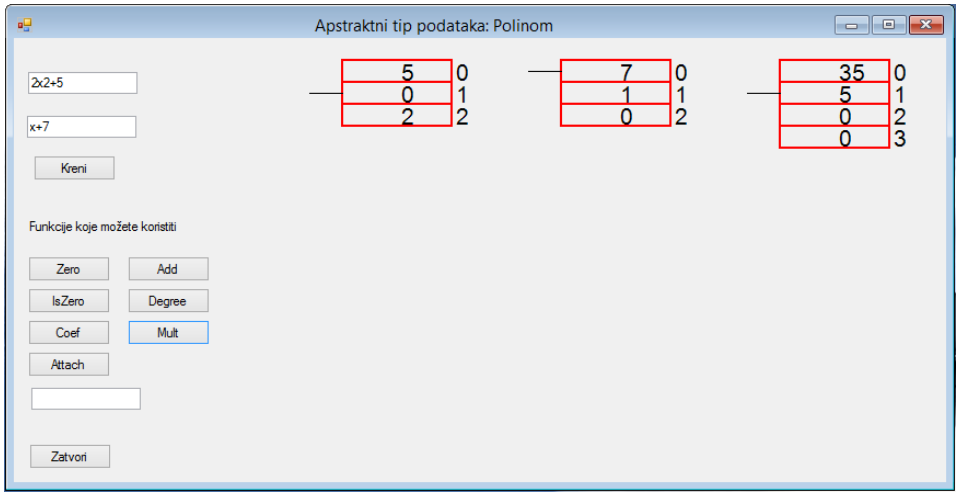
Kod obrasca polinom najzanimljivije realizacije funkcija su *Attach, Add* i *Mult* iz razloga što one koriste dinamičku vizualizaciju. Funkcija *Attach* služi za dodavanje neke vrijednosti određenom koeficijentu. To radi na način da se linija, koja spaja pravokutnik u kojem se nalazi vrijednost kursora sa zadnjim elementom u polju, pomiče prema dolje. Kada dođe na sam vrh zadnjeg pravokutnika u polju dodaje novi pravokutnik sa pripadajućim indeksom i vrijednošću 0, pa naravno poveća kursor za jedan. Kao i kod prijašnjih dinamičkih funkcija za realizaciju korišten je timer:

```
private void timer1_Tick(object sender, EventArgs e)
        {
            nacrtajPolje();
            DrawIt(250, 20, 2);
            DrawString(max.ToString(), 300, 20);
            nacrtajLiniju(350, 30, 400, (max2 + 1) * 20 +10 +
brojac);
            brojac++;
            if ((max2 + 1) * 20 + brojac >= (max + 1) * 20 + 20)
            {

                b.Add(max + 1);
                if (max + 1 == broj[1]) a.Add(broj[0]);
                else a.Add(0);
            }
            if(brojac>=(kranjaGranica-max2)*20+5)timer1.Stop();
        }
```

Iz ovog programskog koda je vidljivo da funkcija koristi varijablu *brojac* kako bi odredila poziciju linije te pravokutnika. Ta ista varijabla koristi se za izračun krajnje granice, tj. za određivanje prestanka rada timera.



Slika 10. Zbrajanje dva polinoma



Slika 11. Množenje dva polinoma

Da bi se uopće mogle koristiti funkcije *Add* (slika 10) i *Mult* (slika 11) potreban je već ranije uneseni polinom te novi polinom kojeg upisujemo u drugi textbox. Kao što je i prije spomenuto, funkcija *Add* služi nam za zbrajanje dvaju polinoma, dok funkcija *Mult* množi dva polinoma. One su obje realizirane na skoro pa identičan način s vrlo malenim izmjenama. Ideja realizacije ovih funkcija je da se vrijednosti polinoma prepišu u pomoćne liste jer su se kod ranijih funkcija u liste upisivali samo koeficijenti uz potencije koji su se nalazili u polinomu. Npr. polinom 2x3+5 kod ranijih funkcija u listama bi izgledao kao 2,5 te 3,0 bez prve i druge potencije. Nakon što su se izradile pomoćne liste, ove funkcije iscrtaju oba polinoma u obliku polja bez kursora. Tada funkcija *Add* briše po element iz svakog polinoma, pa stvara novi čija je vrijednost jednaka zbroju vrijednosti prvobitna dva, dok funkcija *Mult*,

krečući se linijom kroz oba polinoma množi vrijednosti i njihov zbroj dodaje već isto tako stvorenom trećem polinomu, čije su vrijednosti na samom početku sve nule.

Programski kod za vizualiziranje zbrajanja polinoma napravljen je na prikazan način:

```
private void timer2_Tick(object sender, EventArgs e)
        {
            System.Drawing.Graphics grafika = this.CreateGraphics();
            grafika.Clear(DefaultBackColor);

            nacrtajPolje2(pomocna,pomocna2, 200, max,pocni);
            nacrtajPolje2(pomocna_2P, pomocna_22P, 500,
max_2P,pocni);
            zbrajanje.Add(pomocna[brojac2] + pomocna_2P[brojac2]);
            zbrajanje2.Add(brojac2);
            nacrtajPolje2(zbrajanje, zbrajanje2, 700);
                if (pocni == Mmax + 2)
                {
                    timer2.Stop();

                }
            pocni++;
            brojac2++;
        }
```

Za vizualizaciju množenje polinoma koristi se *timer3*:

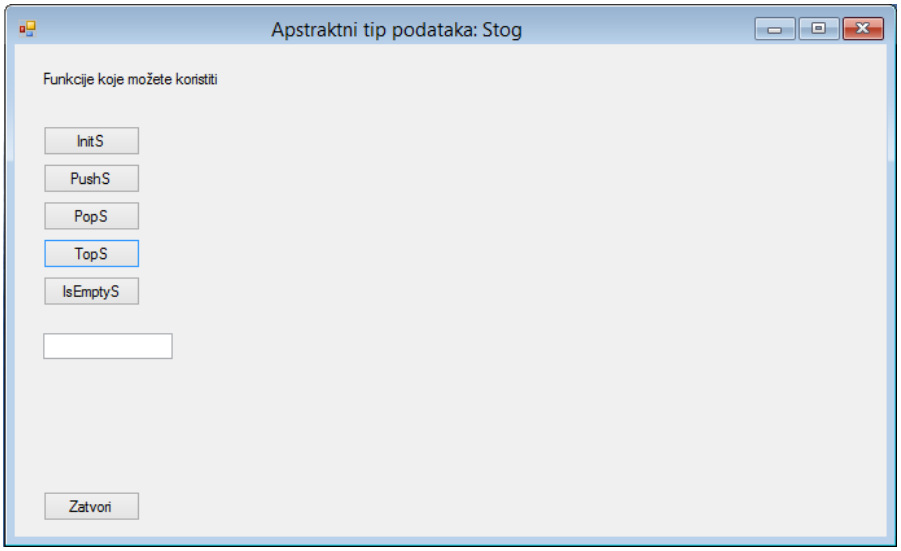
```
private void timer2_Tick(object sender, EventArgs e)
        {
            System.Drawing.Graphics grafika = this.CreateGraphics();
            grafika.Clear(DefaultBackColor);

            nacrtajPolje2(pomocna,pomocna2, 200, max,pocni);
            nacrtajPolje2(pomocna_2P, pomocna_22P, 500,
max_2P,pocni);
            zbrajanje.Add(pomocna[brojac2] + pomocna_2P[brojac2]);
            zbrajanje2.Add(brojac2);
            nacrtajPolje2(zbrajanje, zbrajanje2, 700);
                if (pocni == Mmax + 2)
                {
                    timer2.Stop();

                }
            pocni++;
            brojac2++;
        }
```

Kao i kod svake vizualizacije i ovdje se u oba slučaja na početku briše, odnosno čisti obrazac. Zatim se iscrtavaju polja koristeći funkciju *nacrtajPolje2*, koja je veoma slična proceduri *nacrtajPolje*. Funkcija za zbrajanje koristi brojač *pocni* koji govori funkciji za crtanje koliko elemenata treba iscrtati. On se povećava prilikom svakog takta timera sve dok ne dosegne

maksimalnu veličinu, koja je u ovom slučaju, maksimalni stupanj od dva unesena polinoma zbrojena sa 2. Funkcija *Add* koristi dvije varijable kao brojače (*brojac3*, *brojac4*) jer se mora kretati polinomima neovisno jedan od drugog zbog toga što se množenje bazira na tome da se vrijednosti u polinomu pomnože svaka sa svakom. Sama vizualizacija, odnosno, iscrtavanje polja jednaka je iscrtavanju početnog polinoma.

3.4.3. Stog

Kao što je i prije navedeno stog je apstraktni tip podataka koji pohranjuje elemente istog tipa, a pošto je dozvoljeno mjesto rada njegovih funkcija samo na vrhu stoga, nazivamo ga listom s ograničenim skupom operacija. Stog ima veoma malen broj procedura od kojih samo dvije mijenjaju njegovo stanje. To su funkcije *PushS* te *PopS*.



Slika 12. Obrazac stog

Prilikom njegovog odabira iz početnog izbornika pojavi se obrazac kao na slici 12. Da bi mogli koristiti stog potrebno ga je naravno inicijalizirati te to radimo sa tipkom *InitS*. Prilikom pritiska na nju pojavit će se dva pravokutnika, od kojih jedan sadrži brojač elemenata koji nam prikazuje, kao što mu i samo ime govori, koliko je trenutno elemenata u stogu. Drugi sadrži tekstbox kako bi se korisniku omogućio unos podataka u stog. Valja napomenuti da ova realizacija stoga nije ograničena prije definiranom veličinom i ne može se dogoditi preljev stoga, no broj elemenata može utjecati na sam prikaz vizualizacije funkcija. Prilikom svakog unosa podataka u stog taj podataka ispisuje se zajedno sa pravokutnikom na poziciju

(brojElemenata+1) pomnoženo sa početnom pozicijom, kako bi korisnik mogao uočiti gdje se nalazi podatak koji je prvo unio.

Najvažniji, odnosno vizualno najljepši dio u ovom dijelu aplikacije jest u funkciji *PopS* jer je ona prikazana dinamički. Naime, prilikom svakog klika na tipku PopS svi pravokutnici zajedno s nazivima elemenata, osim prvog, pozicionirat će se za jednu poziciju gore dok će se prvi pozicionirat najprije desno od njih, a zatim će krenuti prema dolje, pa lijevo krećući se 1 piksel po milisekundi.

Slika 13. Vizualizacija funkcije PopS

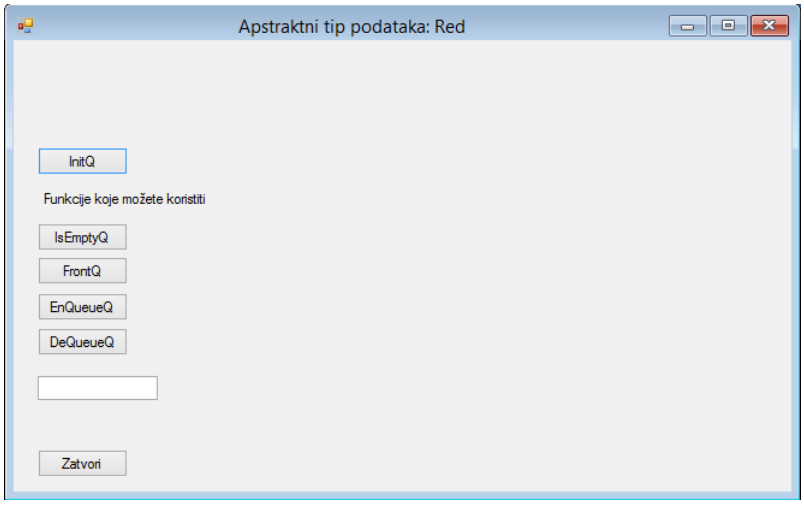
Realizacija slike 13, odnosno funcije PopS izgleda ovako:

```
private void timer1_Tick(object sender, EventArgs e)
        {
            System.Drawing.Graphics grafika = this.CreateGraphics();
            grafika.Clear(DefaultBackColor);
            nacrtajStog();
            if (desno == false)
            {
                DrawIt(500 + brojacPixela, 40, 2);
                DrawString(izletjelo, 500 + brojacPixela, 40);
                brojacPixela++;
                if (brojacPixela == 20)
                {
                    desno = true;
                    brojacPixela = 1;
                }
            }
            if (desno == true && dolje == false)
            {
                DrawIt(500, 40 + brojacPixela, 2);
                DrawString(izletjelo, 500, 40 + brojacPixela);
                brojacPixela++;
                if (brojacPixela == 270)
```

```
            {
                dolje = true;
                brojacPixela = 500;
            }
        }
        if (desno == true && dolje == true && lijevo == false)
        {
            DrawIt(brojacPixela, 310, 2);
            DrawString(izletjelo, brojacPixela, 310);
            brojacPixela--;
            if (brojacPixela == 22)
            {
                textBox3.Text = izletjelo;
                lijevo = false;
                dolje = false;
                desno = false;
                brojacPixela = 0;
                timer2.Start();
                timer1.Stop();
            }
        }
    }
```

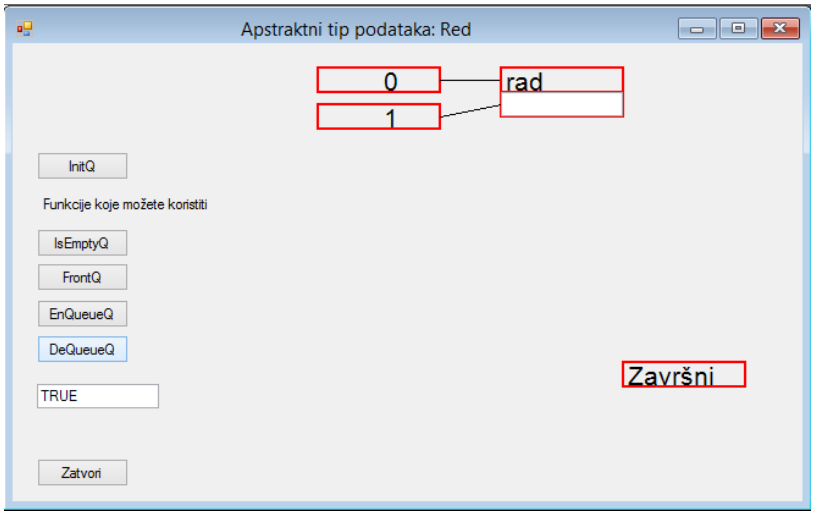
3.4.4. Red

Obrazac red (slika 14) sastoji se od 5 gumbi koji predstavljaju funkcije apstraktnog tipa podataka red, 2 textboxa te gumba *zatvori* i oznake s fiksnim tekstom. Da bi korisnik mogao započeti s unošenjem vrijednosti u red, kao i kod apstraktnog tipa podataka stog, mora kliknuti na gumb *InitQ*. Tada mu se na formi pojave tri pravokutnika od kojih jedan sadrži textbox za unos, a druga dva na samom početku sadrže nulu. Gornji pravokutnik označava početak reda, dok donji označava njegov završetak. Unosom neke vrijednosti u textbox te klikom na gumb *EnQueueQ* iscrtava se još jedan pravokutnik na poziciji gdje je prije toga bio ovaj sa tekstboxom, a taj se sada iscrtava na poziciju ispod njega.



Slika 14. Obrazac red

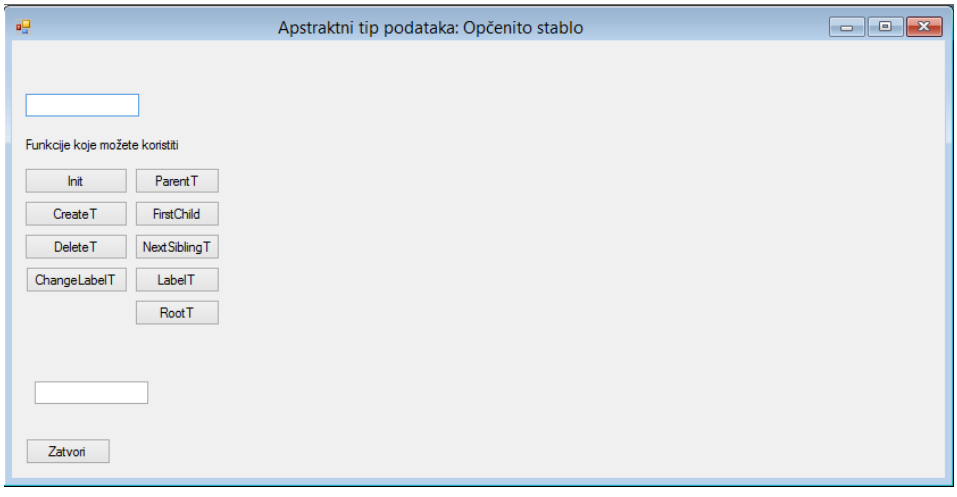
Funkcija *DeQueueQ* napravljena je na isti način kao i funkcija *PopS* kod stoga, pa se nećemo posvetiti njoj nego ćemo samo prikazati izgled obrasca nakon pritiska na nju (slika 15). Apstraktni tip podataka red vizualiziran je također pomoću polja, odnosno pravokutnika, te se za tu svrhu koristila funkcija *nacrtajRed()* koja je veoma slična prethodnoj, uz male izmjene dimenzija iscrtavanja pravokutnika.



Slika 15. Vizualizacija funkcije DeQueueQ

3.4.5. Općenito stablo

Općenito stablo je hijerarhijski uređen apstraktni tip podataka koji također pohranjuje elemente istog tipa.



Slika 16. Obrazac općenito stablo

Specifičnost obrasca na slici 16 jest da se unese vrijednost u textbox prije same inicijalizacije. Nakon pritiska na tipku *Init* pojavi se polje sastavljeno od tri povezana pravokutnika u kojima se upisuje vrijednost čvora, indeks njegovog prvog djeteta i indeks sljedećeg brata. Također valja spomenuti da se za realizaciju ovog apstraktnog tipa podataka koristi klasa kako bi dobili polje koje sadrži odgovarajuće varijable za pohranu:

```
class stablo
    {
        public string vrijednost;
        public int prvoDijete, sljedeciBrat;
    }
```

Najveći problem kod izrade ovog dijela aplikacije bio je brisanje potomka čvorova jer se prilikom brisanja jednog čvora moraju izbrisati i svi njegovi potomci. Za tu svrhu korištena je rekurzivna funkcija *obrisi*. Rekurzivna funkcija jest zapravo funkcija koja poziva samu sebe sve dok se ne zadovolji uvjet koji je napisan unutar nje.

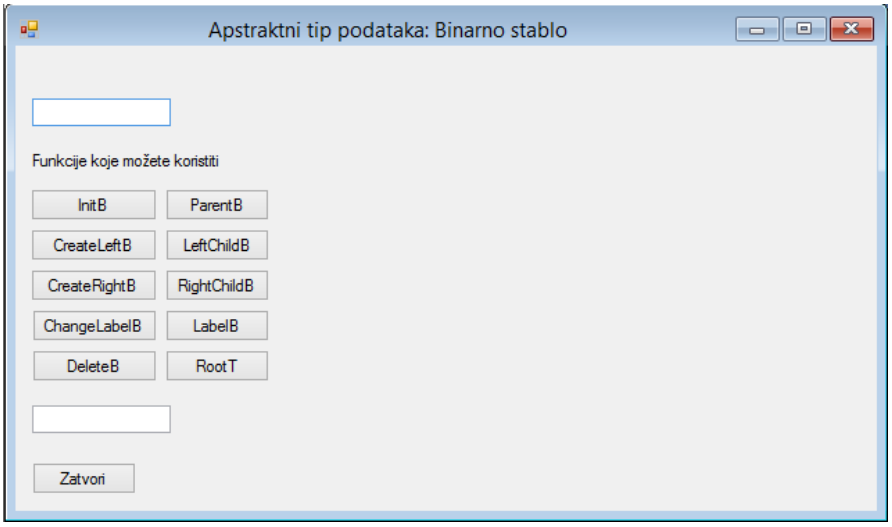
Implementacija funkcije *obrisi* izgleda ovako:

```
public void obrisi(int element) {
            if (element == -1) return;
            obrisi(mojeStablo[element].prvoDijete);
            obrisi(mojeStablo[element].sljedeciBrat);
            mojeStablo[element].vrijednost = "";
            mojeStablo[element].prvoDijete = -1;
            mojeStablo[element].sljedeciBrat = -1;
        }
```

Ova funkcija se najprije kreće po polju tako da dođe do najnižeg djeteta, odnosno zadnjeg potomka unesenog čvora i obriše ga. Zatim se kreće po polju pretražujući njegovu braću te kada dođe do zadnjeg također ga obriše. Tada se vraća sve do čvora, koji je prvobitno unesen kao parametar funkcije, pritom brisajući čvorove.

3.4.6. Binarno stablo

Realizacija ovog apstraktnog tipa podatka veoma je slična prethodnom (slika 17). Ona je na neki način jednostavnija jer se lijevo dijete nalazi na poziciji 2n, dok se desno nalazi na poziciji 2n+1, pa je tako jednostavnije realizirati funkciju *DeleteB* te *CreateLeftB* i *CreateRightB.*



Slika 17. Obrazac binarno stablo

U ovom dijelu nije korištena nikakva posebna klasa, pa i to govori o jednostavnosti realizacije naprama prethodnom tipu. Za iscrtavanje ovog tipa podatka koristi se funkcija *nacrtajStablo():*

```
public void nacrtajStablo()
        {
            System.Drawing.Graphics grafika = this.CreateGraphics();
            grafika.Clear(DefaultBackColor);
            Binarno ispis = new Binarno();
            for (int i = 0; i < brojElemenata+1; i++)
            {
                ispis = bStablo[i];
                DrawIt(400, 20 + 20 * i, 2);
                DrawIt(500, 20 + 20 * i, 2);
                DrawString(ispis.vrijednost, 400, 20 + 20 * i);
                if (ispis.koristeno == false)
                {
                    DrawString("FALSE", 500, 20 + 20 * i);
                }
                else {
                    DrawString("TRUE", 500, 20 + 20 * i);
                }
                DrawString(i.ToString(), 600, 20 + 20 * i);
            }}
```

Također valja napomenuti kako će se ispisati poruka o pogreški, ukoliko nismo upisali neku vrijednost prilikom same inicijalizacije, a o tome je li inicirano stablo ovise ostale mogućnosti aplikacije, tj. ukoliko nismo inicirali nećemo moći koristiti niti jednu od funkcija binarnog stabla.

Binarno stablo veoma je značajno u računalnoj znanosti jer je veoma brzo za pretraživanje i lako za implementiranje

4. Zaključak

Apstraktne tipove podataka osmišljava sam programer. Omogućuju jasnije i čišće oblikovanje programske arhitekture, povećavaju razumljivost aplikacija i njihovo lakše održavanje.

Aplikacija opisana u završnom radu izrađena je sa svrhom da se olakša shvaćanje funkcija apstraktnih tipova podataka: lista, polinom, stog, red, općenitog stabla te binarnog stabla. Izrađena je u programskom jeziku C# i najveći problem prilikom njene izrade bio je vizualiziranje tih tipova. Nažalost C# ne posjeduje nikakav dodatni alat za izradu animacija, pa je za izradu svake slike potrebno iscrtavati piksel po piksel. Kako se pokretne slike ne mogu drugačije realizirati, osim preko timer-a, on je korišten skoro u svakom obrascu, pa se zna dogoditi da slika na milisekundu nestaje. Također jedan od problema bio je i nestajanje slike kada se prozor smanji sa mišem na manju veličinu te je on i ostao neriješen.

Za izradu ovakve aplikacije, tj. za izradu animacije ne bi se trebalo koristi isključivo samo C#, nego bi se trebalo orijentirati prema tome da se pomoću C# omoguće samo dodatne funkcionalnosti, a animacije da se izrađuju u nekom drugom alatu. Naravno postoje i druga rješenja poput izrade web ili Android aplikacije u kojima bi se ovakva vizualizacija mogla realizirati na prikladniji način. U budućem će radu aplikacija razvijena za potrebe ovog rada biti proširena implementacijom spomenutih apstraktnih tipova podataka pomoću pokazivača. Pored toga, sljedeća inačica aplikacije će obuhvaćati i dodatno apstraktne tipove podataka kao što su primjerice grafovi. Spomenutim će proširenjem aplikacija pokriti sve relevantne apstraktne tipove podataka, olakšati njihovo razumijevanje te biti od pomoći prilikom izrade vlastitih implementacija.

5. Literatura

1. Aho, V. A., Hopcroft, E. J., Ullman, D.J.: Data Structures and Algorithms, Addison Wesley Profesional,1987.
2. Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C.: Introduction to Algorithms. Third Edition, The MIT Press, 2009.
3. Dale N.: C++ Plus Data Structures, Third Edition, Jones and Bartlett Publishers, 2003.
4. Hejlsberg, A., Wiltamuth, S., Golde, P.: The C# Programming Language, Pearson Education, Inc., 2004.
5. Keogh J.: Data Structures Demystified, McGraw-Hill, Osborne, 2004.
6. Kruse, L. R., Ryba, J. A.: Data Structures and Program Design in C++, Prentice-Hall, Inc., 2000.
7. Microsoft Developer Network, Code: Drawing a Filled Rectangle on a Form (Visual C#), 2003, http://msdn.microsoft.com/en-us/library/aa287521(v=vs.71).aspx, pristupano: 22.8.2013
8. Microsoft Developer Network,Code: Drawing Text on a Form (Visual C#), 2003, http://msdn.microsoft.com/en-us/library/aa287524(v=vs.71).aspx, pristupano: 22.8.2013
9. Microsoft MSDNA Library, Built-In Types Table (C# Reference), 2012 http://msdn.microsoft.com/en-us/library/ya5y69ds.aspx, pristupano: 2.9.2013
10. Preiss, R. B.: Data Structures and Algorithms with Object-Oriented Design Patterns in C#, SOMA Networks, 2004.
11. Puntambekar, A. A.: Data Structures, Second Revised Edition, Technical Publications Pune, 2007.
12. Robinson S., Nogel, C., Watson K., Glynn J., Skinner M., Evjen Bill, Professional C#, Third Edition, Wiley Publishing, Inc., Indianapolis, India, 2004.
13. Sane, S. S., Deshpande, A. N.: Data Structures and Algorithms, First Edition, Technical Publications Pune, 2006.
14. Sedgewick, R. Algorithms in C++, Parts 1-5: Fundamentals, Data Structures, Sorting, Searching, and Graph Algorithms. Third Edition, Addison Wesley Professional, 2002.
15. Sedgewick, R.: Alogithms in Java, Third edition, Parts 1-4, Fundamentals, Data structures, Sorting, Searching, Pearson Education, Inc., 2002.
16. Simičević Vedrana, Bauštela nove generacije: mladi stručnjaci na radu u inozemstvu, 2013, http://www.novilist.hr/Vijesti/Hrvatska/Baustela-nove-generacije-mladi-hrvatski-strucnjaci-na-radu-u-inozemstvu, pristupano 2.9.2013
17. Weiss, A. M. : Data Structures and Algorithm Analysis in C, Second Edition, Addison-Wesley Professional, 1996.
18. Wiener R., Lewis, J.P. :Fundamentals of OOP and Data Strucutes in Java, Cambridge university press, 2000.