Šolski center Celje
Splošna in strokovna gimnazija Lava

Program za risanje grafov

(raziskovalna naloga)

Mentor:
Mojmir KLOVAR, univ. dipl. inž.

Avtorja:
Boris ŠPOLJAR, GL – 4. F
Anže ŽOLNIR, GL – 4. F

Celje, marec 2005

Kazalo

Kazalo slik

Povzetek

Za raziskovalno nalogo sva naredila program, ki izrisuje osnovne matematične funkcije. Deluje tako, da uporabnik izbere eno izmed danih funkcijo, ter nato vpiše koeficiente, ki so za izbrano funkcijo značilni. Vključila sva vse funkcije, ki smo se jih učili pri pouku matematike. Na okno se lahko izriše več grafov, ki se med seboj ločijo po barvi in debelini črte. Programirala sva v programskem jeziku C++ v programu Borland C++ Builder 6.0.

Uvod

Predstavitev raziskovalnega problema

Za raziskovalno nalogo sva izdelala program za risanje grafov. Idejo sva dobila, ko nam je profesor za računalništvo v šoli izdelal zelo enostaven program, ki zna izrisati vnaprej določen polinom. Ideja se nama je zdela zelo zanimiva, hkrati pa je program ponujal možnosti za nadgradnjo. Tudi sicer nama je bilo pri pouku matematike risanje različnih grafov vedno boljši del matematičnih ur.

Hipoteza in cilji

Po posvetovanju z mentorjem, sva se odločila, da bi naredila program, ki bo izrisal poljubno funkcijo. Uporabniku naj bi bilo na voljo tekstovno okno, kamor bi vpisal enačbo funkcije. Program bi to enačbo prebral kot običajen tekst, ter s prevajanjem in razčlembo tega teksta razbral enačbo in izrisal njen graf. Uporabnik bi lahko določal tudi velikost enote na koordinatnem sistemu.

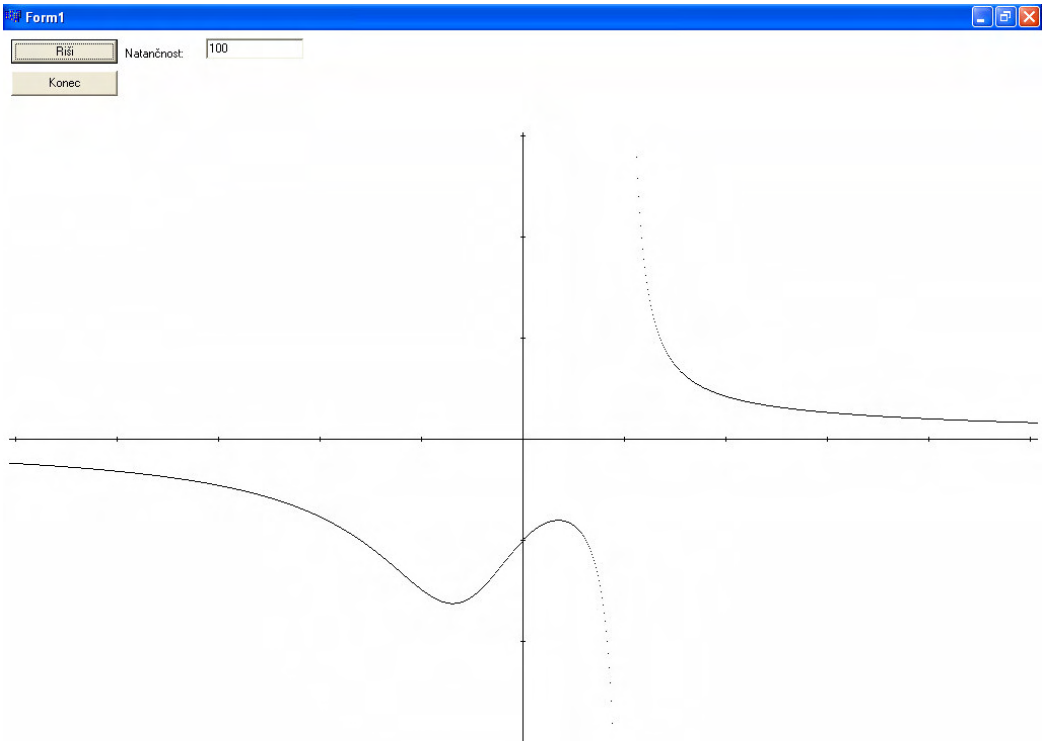
Raziskovalne metode

Raziskovalno nalogo sva izdelala v programu Borland C++ Builder 6.0. To je vizualno programsko orodje, v katerem sva programirala v programskem jeziku C++, katerega osnovne značilnosti se učimo pri urah računalništva.

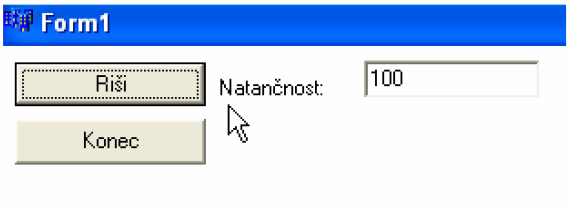
Opis izdelave programa

Idejni program

Idejni program je izdelal mentor kot praktični primer izrisovanja grafov pri uri RSO-ja (Računalniški sistemi in omrežja).



Slika 1: Idejni program



Slika 2: Možnosti idejnega programa

Iz zgornjih slik je razvidna enostavnost in nezahtevnost idejnega programa. Znal je narisati funkcijo, ki si jo moral predhodno pravilno zapisati v sintakso programa. Lahko si določil velikost ali natančnost grafa (v okno natančnost si vpisal število), s katero je bila funkcija narisana. Prav tako si moral predvideti vse morebitne napake in zanke, ki bi lahko sprožile napake v preračunavanju funkcije, ter jih odpraviti. Program se nama je zdel zelo zanimiv, saj ponuja zaradi svoje enostavnosti veliko možnosti spreminjanja, prilagajanja in dodajanja novih ukazov.

Razumevanje idejnega programa

Za boljše razumevanje idejnega programa sva ga natančno preučila. Pomembna nama je bila predvsem funkcija, ki je izrisovala pike na zaslon. Osnovna oblika te funkcije je:

```
Canvas->Pixels[x][y];
```

Spremenljivki x in y predstavljata absciso oziroma ordinato posamezne točke. Naredila sva programček, ki omogoča risanje točk na koordinatni sistem. Ugotovila sva, da ima vsaka točka na oknu dve lastnosti, ki določata njeno lego. To sta parametra Left in Top. Parameter Left pove, koliko je oddaljena točka od levega roba okna, parameter Top pa pove, koliko je točka oddaljena od zgornjega roba okna. Iz tega sledi, da je izhodišče koordinatnega sistema v levem zgornjem robu okna. Da bi torej izhodišče navidezno premaknila v središče okna, sva uporabila naslednjo vrstico:

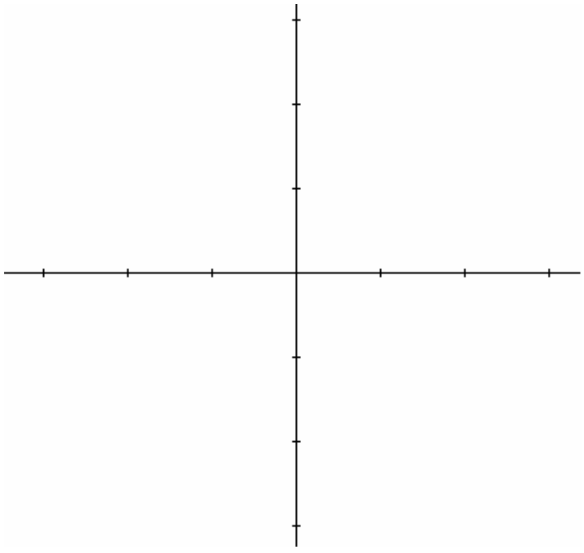
```
Canvas->Pixels[Form1->ClientWidth/2+x][Form1->ClientHeight/2-y];
```

Form1->ClientWidth/2	pomeni polovično velikost zaslona po širini.
Form1->ClientHeight/2	pomeni polovično velikost zaslona po višini.

Pomembno je tudi, da moramo x koordinato prištevati, y koordinato pa odštevati, saj le tako dobimo pravilen rezultat.

O nadaljnjem delu sva se posvetovala z mentorjem. Ta nama je svetoval izdelati program, ki bi lahko izrisoval poljubno funkcijo, ki jo uporabnik vpiše v tekstovno okence. Odločila sva se, da bova za začetek izdelala program, v katerem bi uporabnik imel možnost vnašanja koeficientov vnaprej določenih funkcij.

Koordinatni sistem



Slika 3: Koordinatni sistem

Za začetek sva oblikovala koordinatni sistem, na katerem bi bile razvidne enote. To sva naredila z dvema zankama. Prva je narisala sredinsko črto glede na širino, druga pa glede na dolžino zaslona.
V zanki, ko gre vrednost x iz leve strani zaslona na desno stran, še pazimo, da pri vsakem večkratniku velikosti enote naredimo dve piki, eno gor in eno dol. Podobno ravnamo tudi, ko rišemo sredino po širini zaslona. Tako dobimo na koordinatnem sistemu tudi velikost enote.

```
for(i=-Form1->ClientWidth/2; i<=Form1->ClientWidth/2; i++)
    Form1->Canvas->Pixels[Form1->ClientWidth/2+i][Form1->ClientHeight/2-0]=clBlack;
 for(i=-Form1->ClientHeight/2; i<=Form1->ClientHeight/2; i++)
    Form1->Canvas->Pixels[Form1->ClientWidth/2+0][Form1->ClientHeight/2-i]=clBlack;
 for(i=-Form1->ClientWidth/2; i<=Form1->ClientWidth/2; i++)
    if(i%StrToInt(Label3->Caption)==0)
      {Form1->Canvas->Pixels[Form1->ClientWidth/2+i][Form1->ClientHeight/2-1]=clBlack;
      Form1->Canvas->Pixels[Form1->ClientWidth/2+i][Form1->ClientHeight/2-2]=clBlack;
      Form1->Canvas->Pixels[Form1->ClientWidth/2+i][Form1->ClientHeight/2+1]=clBlack;
      Form1->Canvas->Pixels[Form1->ClientWidth/2+i][Form1->ClientHeight/2+2]=clBlack;
      }
 for(i=-Form1->ClientHeight/2; i<=Form1->ClientHeight/2; i++)
    if(i%StrToInt(Label3->Caption)==0)
      {Form1->Canvas->Pixels[Form1->ClientWidth/2+1][Form1->ClientHeight/2-i]=clBlack;
      Form1->Canvas->Pixels[Form1->ClientWidth/2+2][Form1->ClientHeight/2-i]=clBlack;
      Form1->Canvas->Pixels[Form1->ClientWidth/2-1][Form1->ClientHeight/2-i]=clBlack;
      Form1->Canvas->Pixels[Form1->ClientWidth/2-2][Form1->ClientHeight/2-i]=clBlack;
      }
```

Form1	je okno v katero se izriše koordinatni sistem
Label3->Caption	je velikost enote na koordinatnem sistemu
If (i%StrToInt(Label3->Caption)==0)	to je pogoj, ki preveri, ali je vrednost x oziroma y koordinate večkratnik izbrane enote

Spreminjanje enot

Na velikost enote moramo paziti tudi pri izrisovanju točk. Če bi uporabljala funkcijo za izrisovanje pik, ki sva jo navedla na prejšnji strani, bi ena zaslonska pika predstavljala velikost ene enote. Pravo razmerje dobimo torej, če enostavno vsako x in y koordinato posamezne točke pomnožimo z velikostjo enote. To predstavlja naslednja vrstica iz najinega programa.

```
Canvas->Pixels[Form1->ClientWidth/2+enota*x][Form1->ClientHeight/2-enota*y];
```

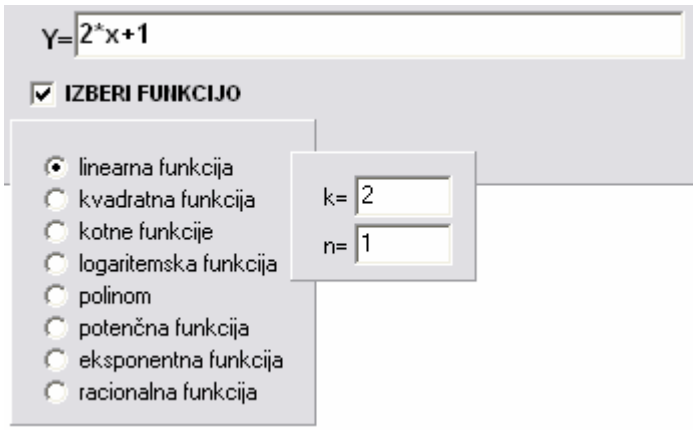
Enota je spremenljivka, ki jo lahko uporabnik spreminja preko drsnika. Velikost enote sva smiselno omejila na večkratnike števila 10 od najmanj 10, do največ 100 zaslonskih pik.

Slika 4: Uporabnik lahko spreminja velikost enote s pomočjo drsnika

Določanje in izbiranje funkcij

Najin program je začenjal dobivati podobo, ko sva določila prve funkcije. Začela sva pri najbolj osnovni funkciji, pri linearni funkciji. Funkcijo, ki jo želi uporabnik izrisati, izbere tako, da klikne na gumb za izbiro, pri katerem piše ime želene funkcije. S tem

se mu prikažejo okenca, v katera uporabnik vnaša koeficiente izbrane funkcije. Pri linearni funkciji sta to na primer smerni koeficient in odsek na y osi.



Slika 5: Vnašanje koeficientov za linearno funkcijo

Program izrisuje funkcije v zanki, v kateri postavimo koordinato x na levo stran zaslona. Medtem ko jo premikamo proti desni strani zaslona, za vsak x izračunamo tudi koordinato y, ki je odvisna od funkcije in prebranih koeficientov. Ko imamo za določen x tudi y koordinato izpišemo to točko na zaslon.

```
if (RadioButton1->Checked==true)
    {if(Edit2->Text=="")Edit2->Text=0;
     if(Edit3->Text=="")Edit3->Text=0:
    for(x=-500;x<500;x++)
            {y=StrToFloat(Edit2->Text)*x+(StrToFloat(Edit3->Text)*100);
                Canvas->Pixels[Form1->ClientWidth/2+x][Form1->ClientHeight/2-y];
            }
    }
```

Edit2 je okence, v katerega uporabnik zapiše koeficient k.

Edit3 je okence, v katerega uporabnik vpiše koeficient n (če vzamemo, da je osnovna oblika linearne funkcije y=k*x+n).

RadioButton je gumbek za izbiro, vsaka funkcija ima svoj gumbek za izbiro, naenkrat pa je lahko izbran le en. Ko uporabnik klikne na gumb »Nariši«, program preveri kateri je izbran in izvrši zanko izbrane funkcije

Postopoma sva dodala še vse ostale funkcije, ki smo jih obravnavali pri pouku matematike. V programu nastopajo linearna funkcija, kvadratna funkcija, kotne funkcije, logaritemska funkcija, polinomi, potenčna funkcija, eksponentna funkcija in racionalna funkcija. Vse funkcije imajo izpostavljen y. Enačbo izbrane funkcije z vsemi vnesenimi parametri lahko uporabnik vidi v okencu v zgornjem levem delu zaslona.

Slika 6: Sproti se v okence izpisuje oblika enačbe z vsemi vnesenimi parametri

Osnovne oblike vseh vključenih funkcij:

linearna funkcija
y=k*x+n

kvadratna funkcija
y=a*x^2+b*x+c

kotne funkcije
y=a*sin(x+b)+c
y=a*cos(x+b)+c
y=a*tan(x+b)+c
y=a*ctan(x+b)+c

logaritemska funkcija
y=a*logx(b)+c

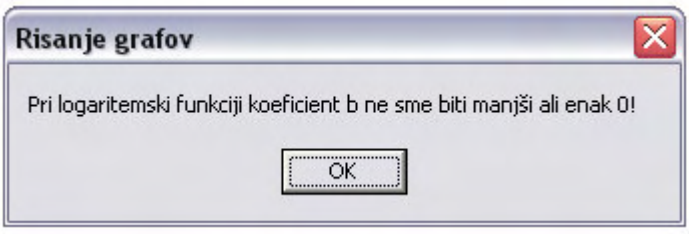
polinom
y=a*x^5+b*x^4+c*x^3+d*x^2+e*x+f

potenčna funkcija
y=a*(x+b)^c

eksponentna funkcija
y=a*b^(x+c)

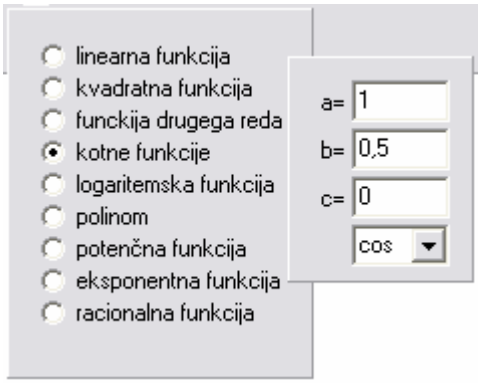
racionalna funkcija
y= (a*x^4+b*x^3+c*x^2+d*x+e)/(a1*x^4+b1*x^3+c1*x^2+d1*x+e1)

Za vsako funkcijo sva predvidela, do kakšnih napak lahko pride pri izračunu koordinat, ter na podlagi specifikacij posameznih funkcij omejila vrednosti koeficientov. Če uporabnik vpiše napačen koeficient, program nemudoma opozori uporabnika.



Slika 7: Program prikaže opozorilo, če uporabnik vpiše napačen koeficient

Ob izbiri vsake funkcije se pokažejo okenca, v katere uporabnik vpiše koeficiente, ki so za določeno funkcijo značilni.



Slika 8: Funkcije, ki jih program lahko izrisuje

Optimizacija izrisa

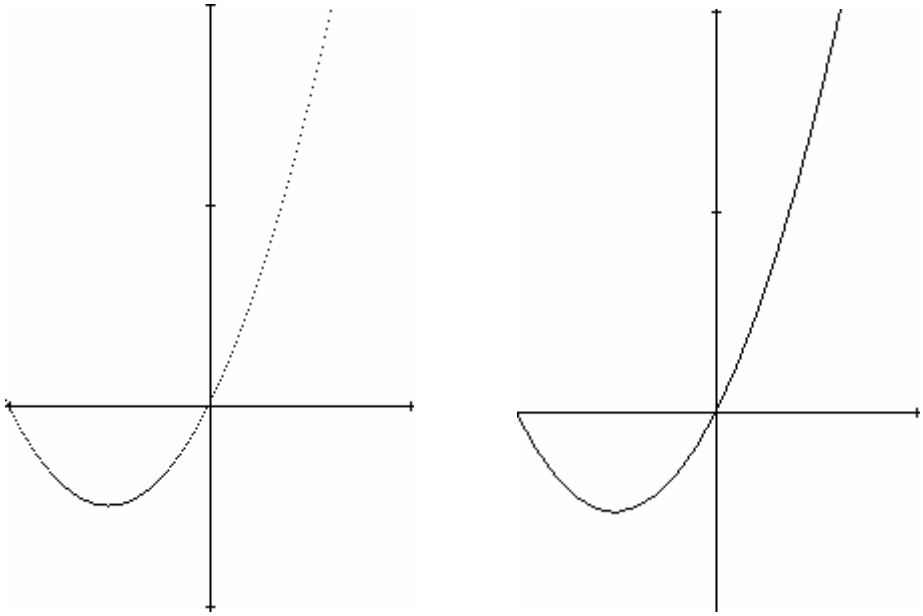
Ko se je v programu nabralo že veliko funkcij sva spoznala, da so nekateri ukazi pri vseh enaki. Zato sva že na začetku pazila, da vsakemu praznemu okencu dodeliva vrednost 0, da ne pride do težav pri pretvarjanju v številske vrednosti. Na začetku sva tudi določila spremenljivke, v katere sva zapisala vrednosti tekstovnih okenc, kamor je uporabnik vpisal vrednosti koeficientov. Tako sva močno poenostavila najino nadaljnje delo.

Druga stvar, ki sva jo opazila je ta, da mora biti glede na različno enoto tudi meja v funkcijah izrisovanja različna. Če je enota na primer velika 10 zaslonskih enot, mora x potekati od -50 do 50, ter če je enota velika 100 zaslonskih enot, mora x potekati od -5 do 5. Zato sva uvedla novo spremenljivko meja, ki jo dobiva z enačbo:

```
meja=(Form1->Width/2-10)/(float)enota;
```

Od polovice širine zaslona sva odštela 10, da dobiva rob ob oknu, na katerega se nič ne izriše.

Tretja stvar, ki sva jo opazila je ta, da najin program riše razmeroma počasi. Izrisovanje sva pospešila z uporabo funkcij MoveTo in LineTo. Funkcija MoveTo se postavi na želeno mesto na zaslonu ter predstavlja začetek daljice. Konec črte pa določimo s funkcijo LineTo. To je omogočilo, da sva lahko število korakov pri zankah izrisa močno zmanjšala. Te možnosti pa nisva uporabila pri risanju polinomov, kotnih funkcij tangens in kotangens ter pri racionalni funkciji, ker se je dogajalo, da program riše funkcijo tudi preko polov ali pa pride do raznih napak pri izrisu. Prav tako sva s to funkcijo rešila še eno izmed težav, to je da funkcija kljub izračunom na nekaj decimalk natančno ni bila zvezna, temveč pikčasta, ker so bile pikice bolj ali manj narazen. Z MoveTo-LineTo funkcijo pa sva jih povezala in s tem dobila (vsaj na videz) zvezno, sklenjeno funkcijo, kar se lepo vidi na sliki 9.



Slika 9: Slika funkcije brez uporabe MoveTo-LineTo in z njim

Za pravilno delovanje mora program paziti le na to, da si v prvem koraku v zanki zapomni vrednosti x in y ter ju izkoristi v drugem koraku. Tako program v zanki s prejšnjima vrednostma x in y najprej postavi izhodišče daljice s funkcijo MoveTo ter s trenutnima vrednostnima x in y postavi konec daljice. Če upoštevamo, da v prvem koraku ne smemo risati ter da uporabimo natančni meji za x, dobimo osnovno zanko za vse funkcije:

```
drugi_korak=0;
if (RadioButton1->Checked==true)
    {
     for(x=-meja;x<=meja;x+=1)
       {y=a*x+b;
       if(drugi_korak)
         {Canvas->MoveTo(Form1->ClientWidth/2+x1*enota,Form1->ClientHeight/2-y1*enota);
           Canvas->LineTo(Form1->ClientWidth/2+x*enota, Form1->ClientHeight/2-y*enota);
          }
             y1=y;
             x1=x;
             drugi_korak++;
             }
    }
```

Zanka izvrši funkciji MoveTo in LineTo le ko je spremenljivka drugi_korak različna od nič. Ta pa je enaka nič v vsakem prvem koraku.

Brisanje

Že na začetku programiranja sva dodala gumbek, ki je zbrisal celoten Form. Vendar je bil način brisanje povsem drugačen, kot pa je v zadnji verziji. Na začetku sva naredila zanko, ki je šla po zaslonu in vsaki zaslonski piki dodelila belo barvo. Ta zanka je bila zelo počasna.

```
for(x=Form1->ClientWidth;x>=0;x--)
for(y=Form1->ClientHeight;y>=0;y--)
Canvas->Pixels[x][y]=clWhite;
```

Nato pa sva po sreči našla zanimiv ukaz, katerega namen je ravno ta, da v oknu izbriše vse črte. Našla sva ga poleg funkcij MoveTo in LineTo.

```
Canvas->FillRect(ClientRect);
```

Ta ukaz je spraznil okno znatno bolj hitro kot zgornja zanka.

Najina zamisel je bila, da bi imela dve okni. V enem bi uporabnik izbral funkcijo ter vpisal parametre, funkcije bi se pa izrisovale na drugem oknu čez celotni zaslon. Ko sva to poskušala realizirati, sva odkrila nama neljubo lastnost okna. Okno si namreč ničesar ne zapomni. To pomeni, da sva vsakič, ko sva drugo okno skrila in ga ponovno prikazala, dobila vrnjeno prazno okno, kljub temu da sva nanj že izrisala funkcijo. To lastnost pa sva lahko izkoristila pri brisanju zaslona, saj sedaj izbriševa okno enostavno tako, da ga osveživa.

```
Form1->Refresh();
```

Poprava izrisa

Kot sva že zapisala, sva ugotovila, da si okno ničesar ne zapomni. Problem se je pojavil pri javljanju opozoril programa. Ko je uporabnik opozorilo potrdil, se je na delu okna, ki ga je opozorilo zakrivalo, vse zbrisalo. Delno rešitev sva našla v dogodku onPaint. Vsak objekt v programu Borland C++ 6 ima svoje dogodke, ki se lahko na njem zgodijo. Ugotovila sva, da če izrisujeva na okno v dogodku onPaint, si okno bolje zapomni vsebino. Še vedno pa se vsa vsebina okna zbriše, če pred okno najinega programa postavimo drugi program ali če okno najinega programa minimiziramo. Rešila sva le primer z opozorilnimi okni, zaradi katerih se vsebina okna ne zbriše več. Ukaze, ki sva jih imela zapisane pod gumbom »Nariši«, sva prestavila na dogodek okna onPaint. V gumb »Nariši« pa sva zapisala ukaz, ki pokliče omenjeni dogodek. To opravi ukaz:

```
Form1->FormPaint(Sender);
```

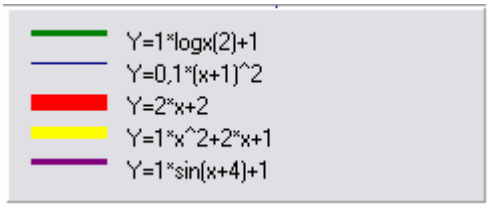
Razni dodatki

Pri velikem številu izrisanih grafov lahko pride do zmede na zaslonu. Da bi bili grafi med seboj dovolj razvidni, sva dodala možnost, da uporabnik izbere debelino in barvo grafa.

Slika 10: Spreminjanje barve grafa

Slika 11: Spreminjanje debeline grafa

Da bi bilo delo s programom še bolj razumljivo in enostavno, sva dodala še legendo, na kateri lahko uporabnik vidi zapis zadnjih petih izrisanih grafov: njihovo debelino, barvo in enačbo. Če je enačba predolga, se nam ob kliku na njen zapis v tabeli pojavi opozorilno okno, v katerem se izpiše celotna enačba funkcije.



Slika 12: Legenda beleži zadnjih pet izrisanih grafov

za primer, če bi se uporabnik zmotil, ali bi enostavno rad zbrisal zadnji graf, sva dodala tudi to možnost. To možnost lahko uporabnik izkoristi tako, da klikne na gumb »Zbriši zadnjo funkcijo«, ki sicer le preriše obstoječi graf z belo barvo (koeficienti morajo seveda ostati enaki).

Zaključek

Ob začetku izdelave programa sva si zadala cilj izdelati program, ki bi lahko narisal poljubno funkcijo. A najino znanje zaenkrat še ni na dovolj visokem nivoju za ta podvig. V bistvo morava priznati, da se zmedeva že ob misli na vsa preverjanja in pogoje, ki bi bili potrebni, da bi iz tekstovne oblike razbrali enačbo. Morebitno rešitev sva našla na internetu v obliki zbirk knjižnic muParser. S temi knjižnicami je mogoče ustvariti program, ki je zmožen poljuben račun v obliki teksta razčleniti in izračunati. Vendar bi bilo izračunavanje posamezne točke, če upoštevamo število korakov pri posamezni funkciji, prepočasno. Poleg tega bi na zaslonu lahko videli le pike in ne črt grafa. Kljub vsemu misliva, da je najina raziskovalna naloga dovolj obsežna ter je primerna kot učni pripomoček za opazovanje, kako se spreminja oblika grafa, če spreminjamo njegove koeficiente.

Viri in literatura

- Bob Swart, Mark Cashman, Paul Gustavson in Jarrod Hollingworth.: Borland C++ Builder Developer's Guide. (e-Knjiga). Indianapolis, Indiana, USA. SAMS. December 2002. ISBN: 0-672-32480-6.

- Kent Reisdorph in Ken Henderson.: Teach Yourself Borland C++ Builder in 21 Days. (e-Knjiga). Indianapolis, Indiana, USA. SAMS. Februar 1997. ISBN: 0-672-31020-1.

- M. Jelen, Matura matematika, Gimnazija-Moste Ljubljana, Ljubljana 1995/96.

- A. Blaznik, A. Cokan in G. Pavlič, Matematični priročnik za srednje šole, DZS d.d., Ljubljana 1998.

- R. Brilej, B. Nikič, T. Pavlinič, T, Robič in Z. Rojs, ALFA 3, Ataja d.o.o., Ljubljana 2003.

Priloge

Na CD-ju:

- Program za risanje grafov

- Raziskovalna naloga

- e-knjiga: Bob Swart, Mark Cashman, Paul Gustavson in Jarrod Hollingworth: Borland C++ Builder Developer's Guide

- e-knjiga: Kent Reisdorph in Ken Henderson: Teach Yourself Borland C++ Builder in 21 Days

Zahvala

Zahvaljujeva se mentorju, ki naju je z napotki usmerjal pri raziskovanju.

Zahvaljujeva se tudi vsem, ki so najin program testirali ter nama z napotki oziroma opozorili pomagali da dokončava najino raziskovalno nalogo.