3

Mjerenje trajanja PC operacija

Da bi smo procijenili valjanost nekog sistema kao sistema u realnom vremenu morati ćemo se upoznati sa vremenskim trajanjem njegovih operacija. Nekada su takva vremena dio specifikacije sistema pa ih je dovoljno testirati, a u slučaju da nisu dio specifikacije sistema tada ih je potrebno odrediti.

Princip mjerenja je krajnje jednostavan. Bilježimo vrijeme na arbitru vremena neposredno prije početka operacije i neposredno po okončanju operacije. Razlika drugog izmjerenog vremena i prvog izmjerenog vremena je traženo trajanje operacije. Pseudokod mjerenja izgleda ovako:

```
Pocetak = VrijemeNaArbitruVremena()
OperacijaCijeTrajanjeMjerimo()
Kraj = VrijemeNaArbitruVremena()
Rezultat = Kraj – Pocetak
```

Međutim jednostavnost principa ne garantuje da nećemo naići i na poteškoće sa detaljima mjerenja. Prije svega potrebno je izabrati adekvatnog arbitra vremena. Korištenje eksternog sata prilikom mjerenja trajanja računarskih operacija zahtijevalo bi određenu komunikaciju između računara i arbitra vremena kojom se arbitru vremena signalizira početak i kraj računarske operacije. Ova komunikacija mora biti dovoljno brza kako ne bi značajno uvećala grešku mjerenja. Jeftinija alternativa je korištenje unutarnjeg sata računara sa kojim je komunikacija osigurana putem funkcija koje su dio osnovnih biblioteka gotovo svakog programskog alata.

Uvidom u neku od ovih biblioteka saznati ćete da su vrijednosti koje se očitavaju zapravo vrijednosti sistemskog sata čiji je najmanji vremenski interval najčešće jedna milisekunda. Ovo je prilično dug vremenski period unutar kojeg se može izvršiti veliki broj računarskih operacija. Mjereći trajanje računarske operacije od nekoliko mikrosekundi korištenjem

sistemskog sata, za rezultat ćemo dobiti ili nula milisekundi ili jednu milisekundu, zavisno od trenutka u kojem je mjerenje izvršeno. Ovo je izuzetno velika relativna greška koja rezultate mjerenja čini neupotrebljivim.

Na koji način je moguće riješiti problem arbitra vremena koji unosi veliku relativnu grešku u proces mjerenja? Naravno moguće je tražiti boljeg arbitra vremena ukoliko takav postoji i dostupan je. Međutim postoji i jednostavnije rješenje koje ne zahtijeva promjenu arbitra vremena. Prisjetimo se mjerenja perioda matematičkog klatna uz pomoć štoperice. Klatno se zanjiše, a potom se mjeri trajanje dovoljnog broja njihanja klatna. Potom se izmjereno vrijeme podijeli sa brojem njihanja u toku mjerenja da bi se dobilo trajanje perioda klatna. Ovakvim načinom mjerenja izbjegava se relativno velika greška mjerenja koja bi nastala kada bi štopericom pokušali direktno mjeriti period klatna mjerenjem trajanja jednog njegovog njihanja.

Ovaj princip je u potpunosti primjenjiv i na mjerenje trajanja računarskih operacija. Dakle potrebno je mjeriti trajanje niza operacija, a ne jedne operacije kako bi se došlo do što tačnijeg rezultata. Ovo u principu znači da je operaciju, čije trajanje mjerimo, potrebno staviti unutar programske petlje koja će se izvršavati dovoljan broj puta, formirajući niz izvršenja jedne te iste operacije. Broj prolazaka kroz petlju odnosno dužinu niza ponavljajućih operacija zvati ćemo kratko „broj ponavljanja“. Vrijeme bilježimo prije izvršavanja petlje i nakon izvršenja petlje. Razlika ovih vremena je vrijeme izvršavanja niza identičnih operacija. Dijeljenjem vremena izvršavanja niza identičnih operacija sa brojem ponavljanja, dobivamo vrijeme trajanja jedne operacije.

```
Pocetak = VrijemeNaArbitruVremena()
For BrojacPonavljanja = 1 to BrojPonavljanja
  OperacijaCijeTrajanjeMjerimo()
Next
Kraj = VrijemeNaArbitruVremena()
Rezultat = (Kraj – Pocetak) / BrojPonavljanja
```

Ovim je riješen problem mjerenja razmjerno kratkih operacija u odnosu na minimalni interval arbitra vremena. Međutim ostaje da se odgovori na pitanje koliki je to „dovoljan“ broj ponavljanja, odnosno koliko puta treba unutar petlje ponavljati operaciju čije trajanje mjerimo?

Greška mjerenja postaje sve manja kako trajanje izvršavanja niza ponavljanja operacije koju mjerimo postaje sve duže od najmanjeg

vremenskog intervala arbitra vremena. Drugim riječima, za veći broj ponavljanja operacije imamo i tačnije rezultate mjerenja. To bi nas moglo navesti na zaključak da je inicijalni izbor veoma velikog broja ponavljanja operacije ispravna strategija. Međutim, izvršavanje ovakve petlje može trajati veoma dugo, pa je u stvarnosti izbor broja ponavljanja operacije kompromis između tačnosti mjerenja koju moramo postići i vremena koje nam je na raspolaganju da mjerenje završimo. Obično počinjemo sa manjim brojem ponavljanja operacije da bi ga postepeno povećavali. Rezultati mjerenja sve više teže stvarnom vremenu trajanja operacije kako povećavamo broj ponavljanja operacije.

Za svaki izabrani „broj ponavljanja“ potrebno je izvršiti više od jednog mjerenja. Ova mjerenja ćemo zvati „grupom identičnih mjerenja“, a ukupan broj mjerenja u grupi identičnih mjerenja zvati ćemo kratko „broj mjerenja“. Potreba za ponavljanjem identičnih mjerenja dolazi od stohastičke prirode onoga što mjerimo. Naime, svako mjerenje će biti ometano drugim procesima unutar računara koji se odvijaju paralelno našem mjerenju. Priroda ovakvih smetnji nije predvidiva i njihov utjecaj na rezultate mjerenja može biti umanjen računanjem srednje vrijednosti identičnih mjerenja. Dakle rezultat grupe identičnih mjerenja za neki izabrani broj ponavljanja operacije je srednja vrijednost rezultata pojedinačnih mjerenja.

```
Rezultat = 0
For BrojacMjerenja = 1 to BrojMjerenja
  Pocetak = VrijemeNaArbitruVremena()
  For BrojacPonavljanja = 1 to BrojPonavljanja
    OperacijaCijeTrajanjeMjerimo()
  Next
  Kraj = VrijemeNaArbitruVremena()
  Rezultat = Rezultat + (Kraj – Pocetak) / BrojPonavljanja
Next
Rezultat = Rezultat / BrojMjerenja
```

Razlika između rezultata, dobivenih korištenjem opisanog algoritma, postaje sve manja pri daljem povećavanju „broja ponavljanja“, a rezultati su sve bliže stvarnom vremenu trajanja operacije. Kada ova razlika postane dovoljno mala, srednji rezultat grupe identičnih mjerenja sa najvećim „brojem ponavljanja“ uzimamo kao konačan rezultat mjerenja.

Algoritam, koji je upravo opisan pseudokodom, je potpuno ispravan, međutim način na koji ovaj algoritam prikuplja podatke mjerenja može

maskirati određene periodične pojave. Naime prilikom mjerenja, koja ćemo uskoro provesti, spašavati ćemo u fajl i rezultate pojedinačnih mjerenja radi naknadne analize. Ukoliko se upis u fajl obavlja između dva pojedinačna mjerenja, tada će neizbježno između njih postojati i vremenski razmak čija dužina odgovara trajanju upisa u fajl. Ovakve pauze unutar mjerenja mogu sakriti postojanje periodičnih promjena u brzini izvršavanja operacije koje su posljedica periodičnog dodjeljivanja vremena od strane operativnog sistema. Tako bi algoritam koji ne uključuje spašavanje u fajl mogao dati rezultate iz kojih je, na primjer, vidljivo da je svako treće pojedinačno mjerenje trajalo duplo duže od prethodna dva pojedinačna mjerenja. U algoritmu koji uključuje spašavanje pojedinačnih rezultata u fajl, ovakva pojava bi mogla ostati neopažena jer ne bi imala pravilan period.

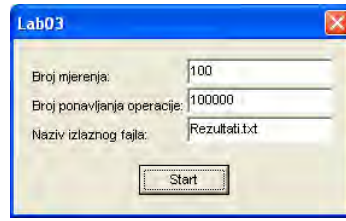
Modifikovani algoritam pohranjuje rezultate pojedinačnih mjerenja u niz, a tek po okončanju svih mjerenja spašava rezultate u fajl. Da bi se vremenski razmak između dva pojedinačna mjerenja dodatno smanjio, modifikovani algoritam pamti samo prolazna vremena, a rezultate pojedinačnih mjerenja računa, također, tek po okončanju svih pojedinačnih mjerenja.

```
ProlaznoVrijeme[0] = VrijemeNaArbitruVremena()
For BrojacMjerenja = 1 to BrojMjerenja
  For BrojacPonavljanja = 1 to BrojPonavljanja
    OperacijaCijeTrajanjeMjerimo()
  Next
  ProlaznoVrijeme[BrojacMjerenja] = VrijemeNaArbitruVremena()
Next
For BrojacMjerenja = 1 to BrojMjerenja
  Rezultat = (ProlaznoVrijeme[BrojacMjerenja] -
              ProlaznoVrijeme[BrojacMjerenja - 1]) / BrojPonavljanja
  IspisLinijeFajla(Rezultat)
Next
```

Zadatak

Koristeći *Borland Builder* načinite *Windows* aplikaciju koja omogućava mjerenje trajanja računarskih operacija baziranu na prethodno opisanom modifikovanom algoritmu. Arbitar vremena neka bude funkcija clock. Aplikacija treba da omogućava unos broja mjerenja, broja ponavljanja operacije i naziva izlaznog tekst fajla u koji se spašavaju rezultati pojedinačnih mjerenja. Rezultati pojedinačnih mjerenja trebaju biti

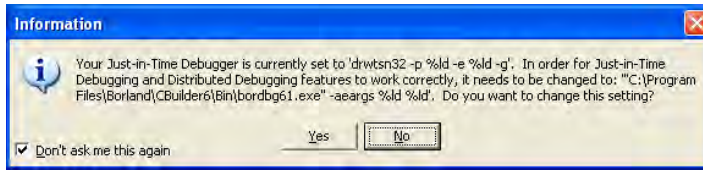
izraženi u broju operacija u sekundi i svaki od rezultata treba biti ispisan u zasebnoj liniji teksta izlaznog fajla.



Slika 3-1. Očekivani izgled aplikacije „Lab03“

Rješenje

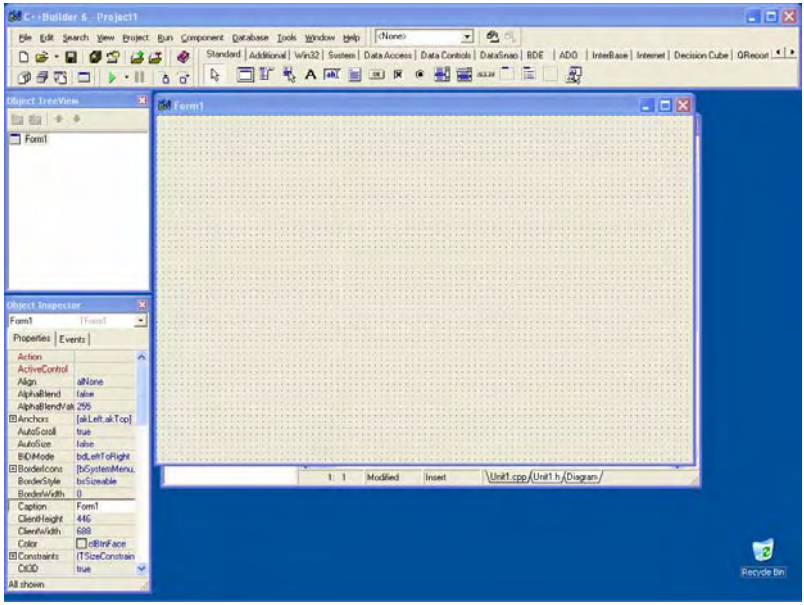
Pokrenite *Borland Builder* izborom *C++Builder 6* iz grupe programa *Borland C++Builder 6*. Ukoliko je ovo prvo pokretanje *Builder*-a od njegove instalacije vidjeti ćete slijedeću poruku.



Slika 3-2. *Just-in-Time* i *Distributed Debugging* informacija

Builder vas obavještava da *Just-in-Time* i *Distributed Debugging* nisu pod kontrolom *Builder*-a. Naime, ukoliko nemate drugih razvojnih alata, najvjerovatnije je da *Dr. Watson*, kao sastavni dio *Windows*-a, ima kontrolu nad *Debugging*-om. Aplikacije koje ćete kreirati u toku ovih vježbi ne zahtijevaju da *Builder* preuzme kontrolu nad *Just-in-Time* i *Distributed Debugging*-om i dovoljno je izabrati da ne želite promijeniti postojeću postavku. Klikom na *Check Box „Don't ask me this again“* izbjeći ćete ponavljanje pitanja prilikom slijedećih pokretanja *Builder*-a.

Builder automatski otvara novi projekt koji sadrži šablon *Windows* aplikacije. Projekt sadrži praznu formu i taman toliko izvornog koda koliko je potrebno da aplikacija posjeduje osnovnu funkcionalnost i bude spremna za pokretanje.



Slika 3-3. *Builder Windows* aplikacija

Builder projekt se sastoji od više fajlova koji sadrže forme, izvorni kod, parametre projekta, izvršni program i druge resurse. Tu su i privremeni fajlovi koji nastaju u toku kompajliranja i linkovanja. Spašavanje više od jednog projekta u jedan te isti folder vodi ka otežanom raspoznavanju relacije između projekta i njemu pripadajućih fajlova, što dalje otežava pojedinačno spašavanje projekta na backup medij i generalno se smatra lošom praksom. Savjetuje se da odmah po kreiranju projekta izaberete opciju spašavanja svih fajlova projekta. U toku ove operacije *Builder* će vas pitati za lokaciju na koju želite spasiti projekt, što vam ostavlja mogućnost da kreirate novi folder i na taj način grupišete sve fajlove projekta. Za spašavanje svih fajlova projekta postupite na slijedeći način:

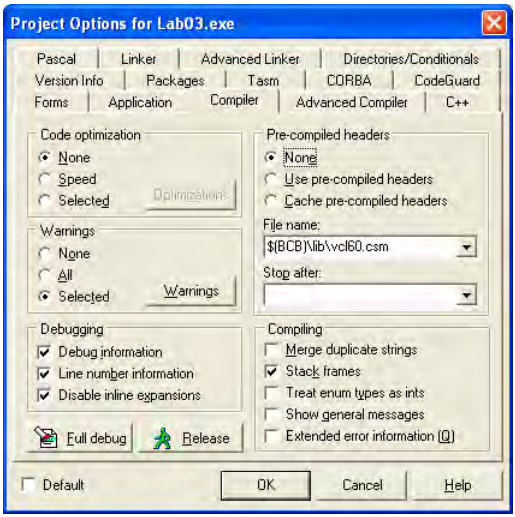
- Sa glavnog menija izaberite „*File*“ i potom „*Save All*“ (kratica *Shift*+*Ctrl*+*S*)
- U prozoru otvorenom u prethodnom koraku izaberite „*My Documents*“ i kliknite na ikonicu za kreiranje novog foldera.
- Dodijelite novom folderu ime „Lab03“ i izaberite ga kao destinaciju za fajlove projekta.
- Potvrdite snimanje fajla koji sadrži formu nakon što promijenite njegov naziv iz „Unit1“ u „GlavnaForma“.

- Potvrdite snimanje fajla projekta nakon što promijenite njegovo ime iz „Project1“ u „Lab03“.

Aplikacija je spremna za pokretanje i možete je pokrenuti izborom „*Run*“ sa glavnog menija i opet „*Run*“ sa podmenija (kratica *F9*). Ukoliko prilikom pokušaja pokretanja aplikacije *Builder* prijavi grešku „*F1013 Error writing output file*“, najvjerovatnije niste kreirali zaseban folder za svoj projekt. Do greške je došlo zbog pokušaja *Builder*-a da snimi vaše fajlove u folder nad kojim nemate odgovarajuće pravo pristupa.

Moguća kompajlerska upozorenja „*W8058 Cannot create pre-compiled Header: write failed*“, unutar *Build* prozora *Builder*a-a, su posljedica ne posjedovanja administratorskih prava pristupa nad folderom u koji je *Builder* instaliran. Ovo upozorenje možete zanemariti ili isključiti na slijedeći način:

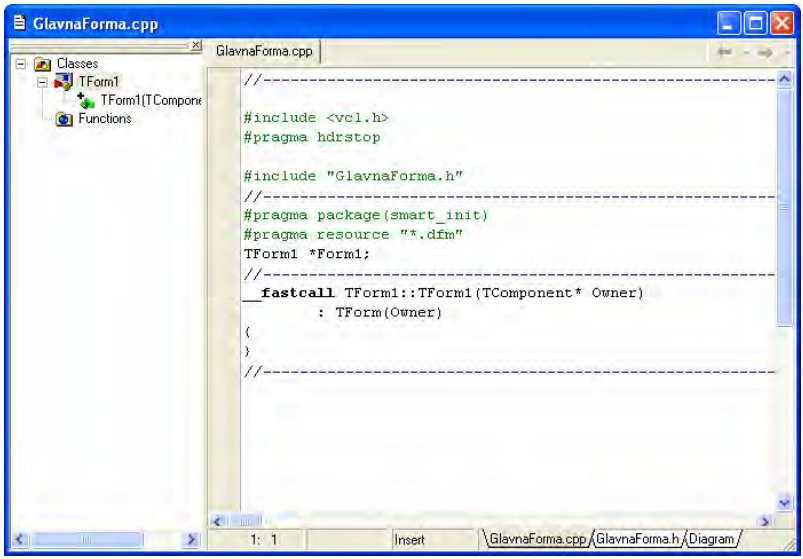
- Sa glavnog menija izaberite „*Project*“ i potom „*Options…*“ (kratica *Shift+Ctrl+F11*).
- U prozoru koji je otvoren u prethodnom koraku izaberite tab „*Compiler*“.
- U grupi „*Pre-compiled Headers*“ izaberite opciju „*None*“
- Zatvorite prozor klikom na OK.



Slika 3-4. *Pre-compiled Headers*

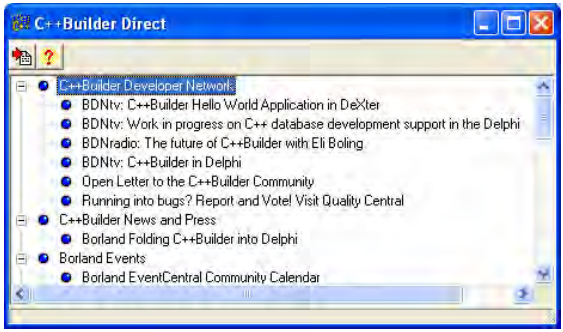
Ne zaboravite da je *„Pre-compiled Headers“* opcija na nivou projekta, što znači da je prethodnu proceduru potrebno ponoviti za svaki novi projekt u kojem želite isključiti upozorenje.

Za navigaciju i obradu izvornog koda koristi se *ClassExplorer* do kojeg, ukoliko nije otvoren, možete doći izborom opcije *„View“* sa glavnog menija i potom klikom na *„ClassExplorer“*. Ukoliko u toku rada ne možete da locirate vizuelnu prezentaciju forme, jednostavan način da dođete do nje je da izaberete *„View“* sa glavnog menija i potom *„Forms“* (kratica *Shift+F12*) nakon čega možete izabrati željenu formu po njenom nazivu.



Slika 3-5. *ClassExplorer*

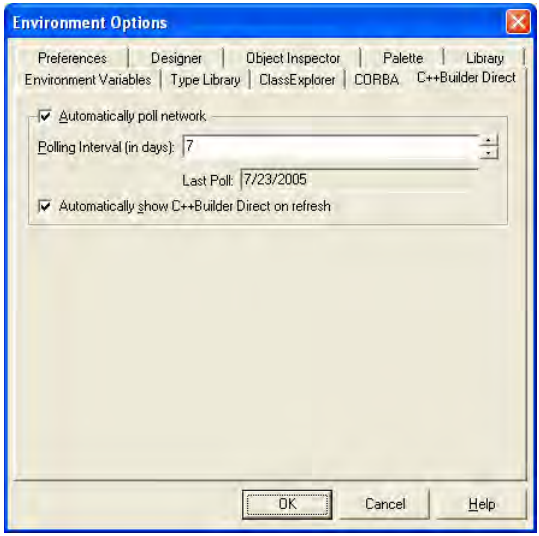
Vaš rad sa *Builder*-om može se učiniti ugodnijim promjenom nekoliko opcija. *BuilderDirect* je pozadinski program koji nastoji da se konektuje na *Borland*-ov *Web Site* u potrazi za posljednjim informacijama vezanim za *Builder*.



Slika 3-6. *BuilderDirect*

Na računaru skromnijih mogućnosti, pokušaji pozadinskog konektovanja *BuilderDirect*-a mogu osjetnije usporiti reakcije *Builder*-a na akcije tastature i miša što otežava rad korisnika. Ukoliko posjedujete administratorska prava nad folderom u kojem je *Builder* instaliran, *BuilderDirect* možete isključiti na slijedeći način:

- Sa glavnog menija izaberite „*Tools*“ i potom izaberite „*Environment Options…*“.
- Izaberite tab „*C++Builder Direct*“.
- Isključite *Check Box* „*Automatically poll network*“.
- Zatvorite prozor klikom na OK.

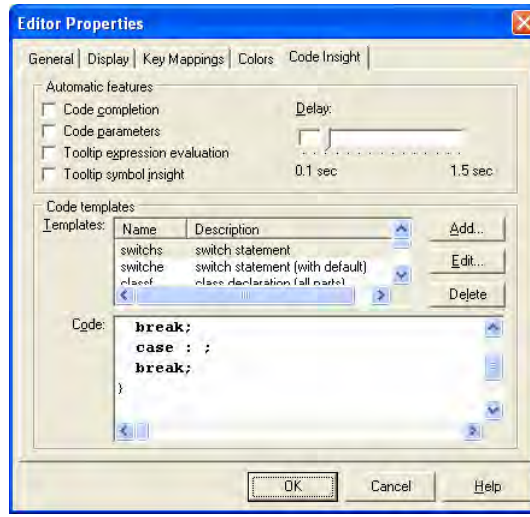


Slika 3-7. Isključivanje *BuilderDirect*

Ukoliko ne posjedujete administratorska prava nad folderom u kojem je *Builder* instaliran, biti ćete primorani da sačekate pojavljivanje prozora *BuilderDirect*-a i potom ga zatvoriti. Na sreću *BuilderDirect* se po inicijalnoj postavci pojavljuje tek svakog sedmog dana.

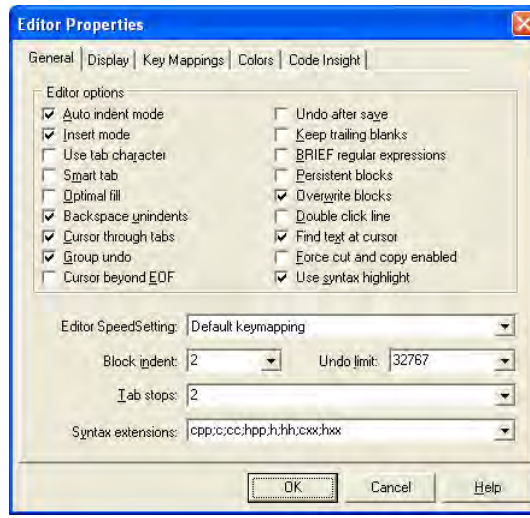
Code Insight je korisna funkcionalnost *Builder*-a koja interaktivno pokušava dovršiti dijelove izvornog koda dok ga unosite. Na žalost, pokušaji da se ponude sve moguće varijante nastavka izraza znaju trajati predugo, pa u nekim slučajevima postaju više smetnja nego pomoć. *Code Insight* možete isključiti čak i ukoliko nemate administratorska prava nad *Builder* instalacionim folderom. Ipak, u tom slučaju postavku ne možete spasiti, pa ćete je morati nanovo postavljati sa svakim pokretanjem *Builder*-a. Za isključenje *Code Insight*-a poduzmite slijedeće:

- Sa glavnog menija izaberite „*Tools*“ i potom izaberite „*Editor Options…*“
- Izaberite tab „*Code Insight*“
- Isključite *Check Box*-ove pod „*Automatic Features*“
- Zatvorite prozor klikom na OK



Slika 3-8. Isključivanje *Code Insight*

Preglednost izvornog koda možete poboljšati izborom nešto manjih tabulatora. Tabulatori od dva mjesta daju manju, ali dovoljnu „nazubljenost“ izvornog koda. Za promjenu tabulatora u već otvorenom prozoru izaberite tab „*General*“ i promijenite vrijednost „*Tab stops*“.



Slika 3-9. Promjena tabulatora izvornog koda

Za više informacija o *Builder*-u upućujemo vas na *Builder Help*, koji je u velikom broju slučajeva osjetljiv na kontekst u kojem je pozvan pritiskom na *F1*.

Kreirajmo sada grafički interfejs aplikacije dodajući kontrole na formu. Izaberite *Label* kontrolu sa *Toolbar*-a i dodajte je na formu. Koristeći *Object Inspector* promijenite Caption kontrole iz „Label1“ u „Broj mjerenja:“

Slika 3-10. ***Object Inspector***

Potom promijenite Name iz „Label1“ u „LabelBrojMjerenja“. Na isti način dodajte drugu *Label* kontrolu, postavite njen Caption na „Broj ponavljanja operacije:“ i Name na „LabelBrojPonavljanja“. Posljednja, treća, *Label* kontrola koju dodajemo treba imati Caption jednak „Naziv izlaznog fajla:“ i Name jednako „LabelFajl“.

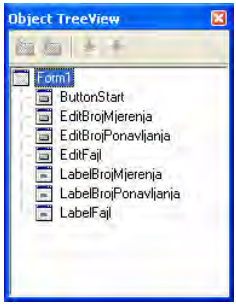
Dodajte sada prvu *Edit* kontrolu. Promijenite njen Name iz „Edit1“ u „EditBrojMjerenja“. Druga *Edit* kontrola koju dodajete imati će Name jednak „EditBrojPonavljanja“. Treća *Edit* kontrola imati će Name jednak „EditFajl“. Izaberite konačno sve tri *Edit* kontrole istovremeno i postavite njihov Text na „“ (prazan string; navodnici se ne unose).

Posljednja kontrola koju dodajemo na formu je *Button*. Promijenite Caption kontrole iz „Button1“ u „Start“. Promijenite Name iz „Button1“ u „ButtonStart“.

Izaberite sada svih sedam kontrola koje smo dodali na formu. U *Object Inspector*-u, razgranajte stavku Font i promijenite njenu pod-stavku Charset

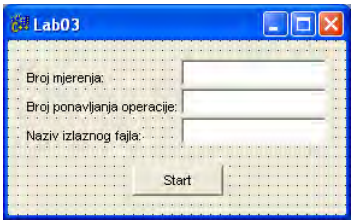
iz „DEFAULT_CHARSET“ u „EASTEUROPE_CHARSET“. Ovim su omogućena slova našeg latiničnog alfabeta na svim kontrolama.

Nakon što smo postavili kontrole na formu promijeniti ćemo i neke postavke same forme. Izaberite formu klikom na dio njene površine na kojem se ne nalazi niti jedna kontrola ili izaberite Form1 u *Object TreeView*.



Slika 3-11. *Object TreeView*

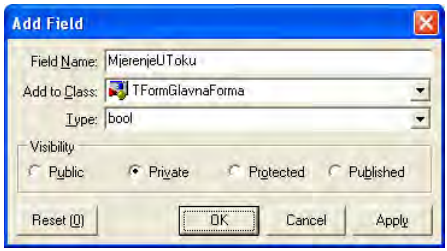
Podesite veličinu forme tako da obuhvati sve svoje kontrole ne ostavljajući previše praznog prostora oko kontrola. Promijenite Caption iz „Form1“ u „Lab03“. Promijenite Name iz „Form1“ u „FormGlavnaForma“. Spriječite promjenu veličine prozora forme promjenom BorderStyle iz „bsSizeable“ u „bsDialog“. Sa glavnog menija izaberite „*Edit*“ pa potom „*Tab Order…*“. Uredite da kontrole forme budu složene onim redom koji odgovara njihovom položaju na formi. Ukoliko ste uspješno pratili sve korake do sada, pred vama bi trebali biti forma koja izgleda kao na slici.



Slika 3-12. Forma aplikacije „Lab03“

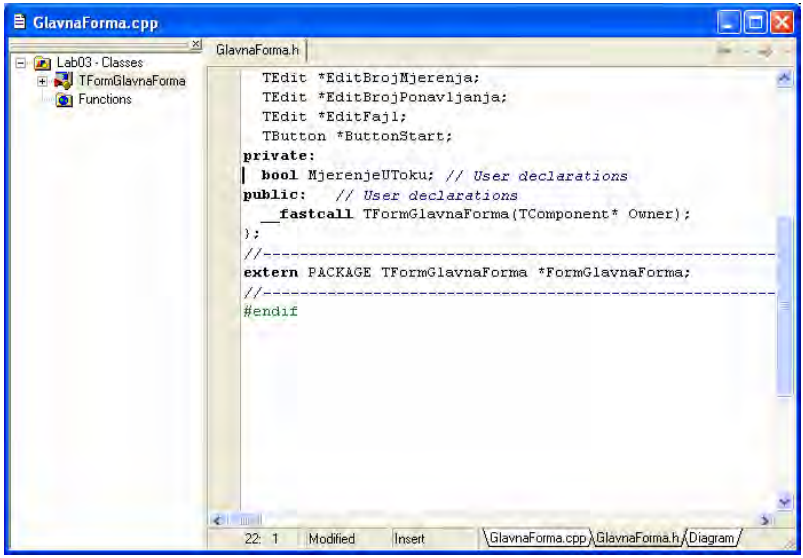
Posvetimo sada pažnju ponašanju koje se zahtijeva od upravo kreiranog grafičkog korisničkog interfejsa (GUI). GUI će imati dva modusa ponašanja. Prvi modus ponašanja je ponašanje GUI za vrijeme unosa

parametara mjerenja, a drugi modus ponašanja je ponašanje GUI za vrijeme trajanja mjerenja. Da bi bili u mogućnosti detektovati trenutni modus ponašanja GUI, formi FormGlavnaForma dodajemo polje MjerenjeUToku koje signalizira da li je mjerenje u toku. Polje dodajemo tako što u *ClassExplorer*-u kliknemo desnim klikom na TFormGlavnaForma i izaberemo „*New Field…*“. U „*Field Name*“ unosimo „MjerenjeUToku“. U „*Type*“ unesimo „bool“ jer se radi o logičkom podatku.



Slika 3-13. Polje MjerenjeUToku

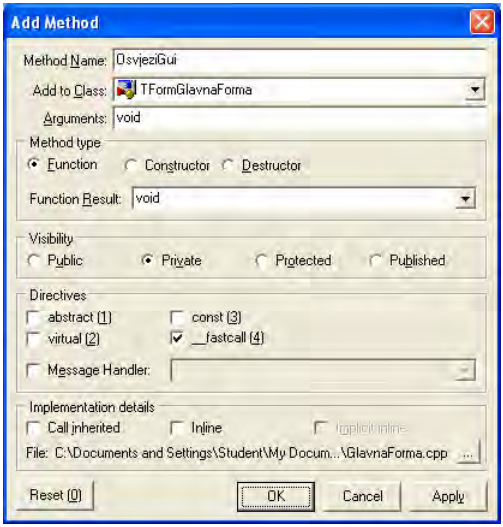
Nakon klika na OK, definicija polja biva dodana u *Header* fajl.



Slika 3-14. Definicija polja MjerenjeUToku u *Header* fajlu

Ponašanje GUI zavisi od MjerenjeUToku na način da kada je ovo polje jednako true tada sve kontrole forme moraju biti onemogućene. Kada je MjerenjeUToku jednako false tada su kontrole forme omogućene sa jednim mogućim izuzetkom. Naime, ButtonStart je izuzetak na način da čak i kada je MjerenjeUToku jednako false, još uvijek je moguće da ButtonStart ne treba biti omogućen. To se dešava kada barem jedan od parametara mjerenja još uvijek nije zadan pa se ne smije dozvoliti da mjerenje počne. Napišimo metod koji implementirati ovu funkcionalnost.

Metod ćemo nazvati OsvjeziGui. OsvjeziGui nema ulaznih parametara i ne vraća rezultat. Vidljivost metoda je private. *Borland* sugeriše da svi metodi klase koriste __fascall direktivu. Metod dodajemo na sličan način kao i polje. Desni klik na klasu i izbor *„New method…“*.



Slika 3-15. Metod OsvjeziGui

Na mjesto komentara *„TODO: Add your source code here“* potrebno je unijeti izvorni kod koji implementira funkcionalnost metoda.

```
void __fastcall TFormGlavnaForma::OsvjeziGui(void)
{
  if (MjerenjeUToku){
    // Ako je mjerenje u toku, potrebno je onemoguciti kontrole forme.
    // Prije svake promjene kontrole, provjeri da li je neophodna!
    // Ovim se izbjegava "treperenje" kontrola.
```

```
    if (EditBrojMjerenja->Enabled) EditBrojMjerenja->Enabled = false;
    if (EditBrojPonavljanja->Enabled) EditBrojPonavljanja->Enabled = false;
    if (EditFajl->Enabled) EditFajl->Enabled = false;
    if (ButtonStart->Enabled) ButtonStart->Enabled = false;
  }
  else{
    // Ako mjerenje nije u toku, tada je potrebno omoguciti kontrole.
    if (!EditBrojMjerenja->Enabled) EditBrojMjerenja->Enabled = true;
    if (!EditBrojPonavljanja->Enabled) EditBrojPonavljanja->Enabled = true;
    if (!EditFajl->Enabled) EditFajl->Enabled = true;

    // Sve je spremno za start samo ako su zadani svi ulazni parametri.
    if (!(EditBrojMjerenja->Text.IsEmpty() ||
        EditBrojPonavljanja->Text.IsEmpty() || EditFajl->Text.IsEmpty())){
      // Ako je sve spremno za start omoguci taster Start.
      if (!ButtonStart->Enabled) ButtonStart->Enabled = true;
    }
    else{
      // Ako nije, onemoguci Start taster.
      if (ButtonStart->Enabled) ButtonStart->Enabled = false;
    }
  }
}
```

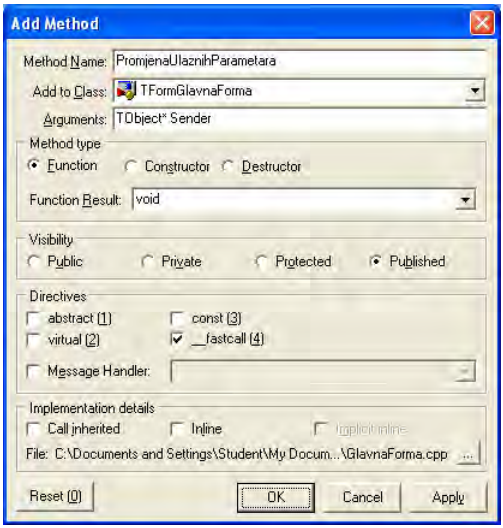
U kojim situacijama je potrebno pozivati OsvjeziGui? Inicijalno, prilikom prvog prikazivanja forme, potrebno je postaviti MjerenjeUToku na false kako bi se signaliziralo da mjerenje nije u toku i potom osvježiti GUI da odražava stanje forme. Ovo moramo uraditi prije pojavljivanja forme, te je idealno mjesto za to konstruktor forme.

```
__fastcall TFormGlavnaForma::TFormGlavnaForma(TComponent* Owner)
  : TForm(Owner)
{
  // Postavi polje koje govori da li je mjerenje u toku.
  MjerenjeUToku = false;
  // Osvjezi GUI da odrazava stanje forme.
  OsvjeziGui();
}
```

Osim inicijalno, GUI je potrebno osvježiti sa svakom promjenom ulaznih parametara jer se stanje ButtonStart mijenja u zavisnosti od toga da li su svi ulazni parametri zadani ili ne. Dakle na svaku promjenu sadržaja

EditBrojMjerenja, EditBrojPonavljanja i EditFajl potrebno je pozvati OsvjeziGUI. Napisati ćemo zajednički *Event Handler*, pod nazivom PromjenaUlaznihParametara, koji će biti zadužen da reaguje na promjenu sadržaja svake od tri *Edit* kontrole.

Event Handler je metoda forme koja ima tačno jedan ulazni parametar tipa pokazivača na instancu klase TObject. Pokazivač je usmjeren na objekt koji je izazvao događaj. *Event Handler* ne vraća rezultat. Vidljivost *Event Handler-a* je published.



Slika 3-16. *Event Handler* PromjenaUlaznihParametara

Povežimo sada *Event Handler* PromjenaUlaznihParametara sa događajima na koje će reagovati. Izaberite kontrolu EditBrojMjerenja i u *Object Inspector*-u izaberite tab *„Events“*. Za događaj OnChange izaberite *Event Handler* PromjenaUlaznihParametara. Istu proceduru ponovite i za dvije preostale *Edit* kontrole, EditBrojPonavljanja i EditFajl.

Konačno napišimo izvorni koji implementira funkcionalnost *Event Handler*-a PromjenaUlaznihParametara.

```
void __fastcall TFormGlavnaForma::PromjenaUlaznihParametara(TObject* Sender)
{
  // Prilikom svake promjene ulaznih parametara osvjezi GUI.
  // Ovo je neophodno da bi sprijecili klik na Start
```

```
  // u trenutku dok neki od ulaznih parametara nedostaje.
  OsvjeziGui();
}
```

U ovom trenutku GUI posjeduje svu funkcionalnost koja bi trebala da spriječi korisnika u „pogrešnom“ korištenju aplikacije. Ipak i dalje je moguće da korisnik odabere nulu ili negativan broj mjerenja ili ponavljana, kao i ime fajla koje uključuje nedozvoljene karaktere. Definišimo ograničenja ulaznih parametara i poruke o greškama u *Header* fajlu „GlavnaForma.h“. Svoje direktive uvijek pozicionirajte nakon *Builder*-ovih direktiva i pragmi. U protivnom vaše direktive mogu imati nepredviđen utjecaj na *Builder*-ove direktive i pragme.

```
#define BROJ_MJERENJA_MAX 10000
#define BROJ_PONAVLJANJA_MAX 100000000
#define GRESKA_NASLOV "Poruka o greski"
#define GRESKA_BROJ_MJERENJA "Pogresan broj mjerenja!"
#define GRESKA_BROJ_PONAVLJANJA "Pogresan broj ponavljanja operacije!"
#define GRESKA_FAJL "Ne mogu otvoriti i pisati izlazni fajl!"
```

Napišimo metod UlazniParametriIspravni koji će provjeriti ispravnost ulaznih parametara iz EditBrojMjerenja i EditBrojPonavljanja. Vrijednost parametara spašavamo u polja BrojMjerenja i BrojPonavljanja kako bi podaci bili dostupni i drugim metodima klase. Kreirajmo polja BrojMjerenja i BrojPonavljanja tipa int, a potom napišimo izvorni kod metoda UlazniParametriIspravni. Metod ne uzima parametre, vraća rezultat tipa bool i ima vidljivost private.

```
bool __fastcall TFormGlavnaForma::UlazniParametriIspravni(void)
{
  // Pokusaj konvertovati ulazne parametre u cjele brojeve.
  // Ukoliko nisu cijeli brojevi, dodijeli im vrijednost 0.
  BrojMjerenja = EditBrojMjerenja->Text.ToIntDef(0);
  BrojPonavljanja = EditBrojPonavljanja->Text.ToIntDef(0);

  // Provjeri da li su uneseni podaci zaista cijeli brojevi i
  // da li su unutar ogranicenja.
  if (BrojMjerenja <= 0 || BrojMjerenja > BROJ_MJERENJA_MAX)
  {
    Application->MessageBox(GRESKA_BROJ_MJERENJA, GRESKA_NASLOV, MB_OK);
    return(false);
  };
```

```
  if (BrojPonavljanja <= 0 || BrojPonavljanja > BROJ_PONAVLJANJA_MAX)
  {
    Application->MessageBox(GRESKA_BROJ_PONAVLJANJA, GRESKA_NASLOV, MB_OK);
    return(false);
  };

  // Ako su podaci prosli oba testa smatramo da su korektni.
  return(true);
}
```

Ispravnost naziva izlaznog fajla i mogućnost pisanja u izlazni fajl moguće je utvrditi tek prilikom otvaranja fajla i provjerom rezultata funkcije fopen. Za korištenje funkcija za rad sa fajlovima neophodno je uključiti *Header* fajl „stdio.h“. Definišimo polje IzlazniFajl tipa pokazivača na FILE.

```
  FILE* IzlazniFajl;
```

Arbitar vremena je funkcija clock. Ona je deklarisana u *Header* fajlu „time.h“ koji je potrebno uključiti u *Header* fajl „GlavnaForma.h“. Funkcija clock vraća proteklo vrijeme u *Clock* ciklusima za koje je deklarisan tip podataka clock_t. Jedna sekunda ima CLK_TCK *Clock* ciklusa, gdje je CLK_TCK konstanta definisana u „time.h“ i njena vrijednost je 1000. U slučaju greške, rezultat clock će biti -1. Dakle prolazna vremena će biti formata clock_t, pa je potrebo definisati polje ProlaznoVrijeme koje će biti niz od [BROJ_MJERENJA_MAX+1] clock_t podataka, jer je (maksimalan) broj prolaznih vremena za jedan veći od (maksimalnog) broja mjerenja.

```
clock_t ProlaznoVrijeme[BROJ_MJERENJA_MAX + 1];
```

Došli smo do centralnog dijela aplikacije. Potrebno je kreirati metod za mjerenje trajanja računarskih operacija prema prethodno opisanom algoritmu. Metod će se zvati MjeriTrajanjeOperacije. Ovaj metod nema ulaznih parametara niti vraća rezultat.

```
void __fastcall TFormGlavnaForma::MjeriTrajanjeOperacije(void)
{
  // Varijable koje se koriste kao brojaci unutar petlji.
  int BrojacMjerenja, BrojacPonavljanja;
```

```
  // Postavi prvo prolazno vrijeme.
  ProlaznoVrijeme[0] = clock();

  // Vanjska petlja broji mjerenja.
  for (BrojacMjerenja = 1; BrojacMjerenja <= BrojMjerenja;
      BrojacMjerenja++){
     // Unutarnja petlja broji ponavljanje operacije koju mjerimo.
    for (BrojacPonavljanja = 1; BrojacPonavljanja <= BrojPonavljanja;
      BrojacPonavljanja++){
       // Ovdje ubaciti operaciju cije trajanje mjerimo.
    };
     // Zabljezi prolazno vrijeme nakon svakog mjerenja.
    ProlaznoVrijeme[BrojacMjerenja] = clock();
  };
}
```

Kreirajmo metod za ispis rezultata mjerenja, pod nazivom IspisRezultataMjerenja. Ovaj metod nema ulaznih parametara niti vraća rezultat. IspisRezultataMjerenja podrazumijeva da je IzlazniFajl prethodno otvoren za pisanje.

```
void __fastcall TFormGlavnaForma::IspisRezultataMjerenja(void)
{
  // Varijabla koja se koristi kao brojac.
  int BrojacMjerenja;
  // Rezultat mjerenja u clock ciklusima.
  clock_t Rezultat;

  // Petlja za ispis rezultata pojedinacnih mjerenja.
  for (BrojacMjerenja = 1; BrojacMjerenja <= BrojMjerenja;
      BrojacMjerenja++){
    Rezultat = ProlaznoVrijeme[BrojacMjerenja] -
        ProlaznoVrijeme[BrojacMjerenja-1];
    // Paznja! Ako je rezulat jednak 0, niz operacija je bio prebrz
    // da bi ga izmjerili. Sprijeci dijeljenje sa 0.
    if(Rezultat == 0)
      fprintf(IzlazniFajl, "INF\n");
    else
      fprintf(IzlazniFajl, "%d\n",
          BrojPonavljanja * (long)CLK_TCK / Rezultat);
  }
}
```

Mjerenje trajanja računarske operacije pokreće se klikom na ButtonStart. Dakle potrebno je, unutar *Click Event Handler*-a komande ButtonStart pozvati MjeriTrajanjeOperacije i IspisRezultataMjerenja. Kreirajmo *Click Event Handler* za kontrolu ButtonStart duplim klikom na ButtonStart i napišimo izvorni koji provjerava ispravnost ulaznih parametara, mijenja status forme, osvježava GUI, otvara izlazni fajl, poziva algoritam mjerenja i ispisa, te potom zatvara izlazni fajl i opet osvježava GUI.

```
void __fastcall TFormGlavnaForma::ButtonStartClick(TObject *Sender)
{
  // Da li su ulazni parametri korektni?
  if (UlazniParametriIspravni()){
    // Mjerenje je u toku.
    MjerenjeUToku = true;
    // Osvjezi GUI da odrazava novo stanje forme.
    OsvjeziGui();
    // Mjerenje je intenzivna operacija, zato osvjezi formu odmah.
    Repaint();

    // Pokusaj odmah otvoriti izlazni fajl za pisanje.
    // U protivnom moguce je cekati do kraja veoma dugog mjerenja
    // i tek potom saznati da rezultati ne mogu biti spaseni!
    if ((IzlazniFajl = fopen(EditFajl->Text.c_str(), "w")) == NULL)
      Application->MessageBox(GRESKA_FAJL, GRESKA_NASLOV, MB_OK);
    else
    {
      // Izmjeri trajanje operacije i ispisi rezultate mjerenja u fajl.
      MjeriTrajanjeOperacije();
      IspisRezultataMjerenja();

      // Zatvori izlazni fajl.
      fclose(IzlazniFajl);
    }

    // Mjerenje je okoncano.
    MjerenjeUToku = false;
    // Osvjezi GUI da odrazava novo stanje forme.
    OsvjeziGui();
  }
}
```

Aplikacija je ovim dovršena.

Zadatak

Koristeći kreiranu aplikaciju odredite trajanje funkcije gettime. Broj mjerenja neka bude 100. Odredite zadovoljavajući broj ponavljanja operacije i objasnite i dokumentujte vaše kriterije za njegov izbor.

Da bi u aplikaciji koristili funkciju gettime, morate uključiti *Header* fajl „dos.h“. Funkcija gettime zahtijeva ulazni parametar tipa adrese struct time. Vodite računa i o slijedećem:

- Prilikom mjerenja trajanja operacije potrebno je zatvoriti sve druge aplikacije koji bi mogle utjecati na rezultate mjerenja. Pod „drugim aplikacijama“ podrazumijevamo i sam *Builder*.
- Aplikacija kompajlirana i linkovana kao *Debug* verzija sadrži dodatne računarske instrukcije namijenjene otklanjanju bagova. Nije nam cilj mjeriti trajanje ovih dodatnih računarskih instrukcija, već isključivo trajanje operacije gettime. Utvrdite da li postoji značajna razlika u brzini *Debug* i *Release* verzije aplikacije.
- Petlja u kojoj ponavljamo gettime, ima vlastito trajanje koje ne treba pripisivati funkciji gettime. Pokušajte odrediti ovo vrijeme i na osnovu toga korigovati rezultate dobivene za gettime.

Rješenje

Modifikujmo aplikaciju tako da mjeri trajanje operacije gettime. Uključimo „dos.h“ u „GlavnaForma.h“.

```
#include <dos.h>
```

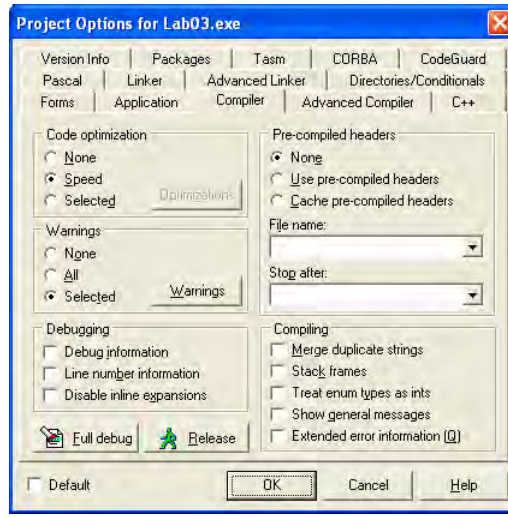
Izvršimo potrebne promjene unutar metoda MjeriTrajanjeOperacije kako bi mjerili trajanje gettime.

```
void __fastcall TFormGlavnaForma::MjeriTrajanjeOperacije(void)
{
  // Parametar za gettime
  struct time t;
  // Varijable koje se koriste kao brojaci unutar petlji.
  int BrojacMjerenja, BrojacPonavljanja;

  // Postavi prvo prolazno vrijeme.
  ProlaznoVrijeme[0] = clock();
```

```
  // Vanjska petlja broji mjerenja.
  for (BrojacMjerenja = 1; BrojacMjerenja <= BrojMjerenja;
      BrojacMjerenja++){
     // Unutarnja petlja broji ponavljanje operacije koju mjerimo.
    for (BrojacPonavljanja = 1; BrojacPonavljanja <= BrojPonavljanja;
      BrojacPonavljanja++){
       gettime(&t);
    };
     // Zabljezi prolazno vrijeme nakon svakog mjerenja.
    ProlaznoVrijeme[BrojacMjerenja] = clock();
  };
}
```

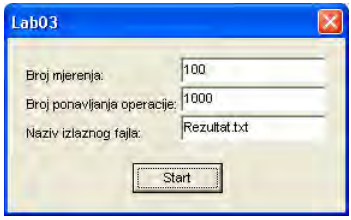
Promijenimo postavke projekta izborom „*Project*“ sa glavnog menija, pa zatim „*Options…*“. Na tabu „*Compiler*“ kliknite na taster „*Release*“.



Slika 3-17. Aktiviranje *Release* postavki

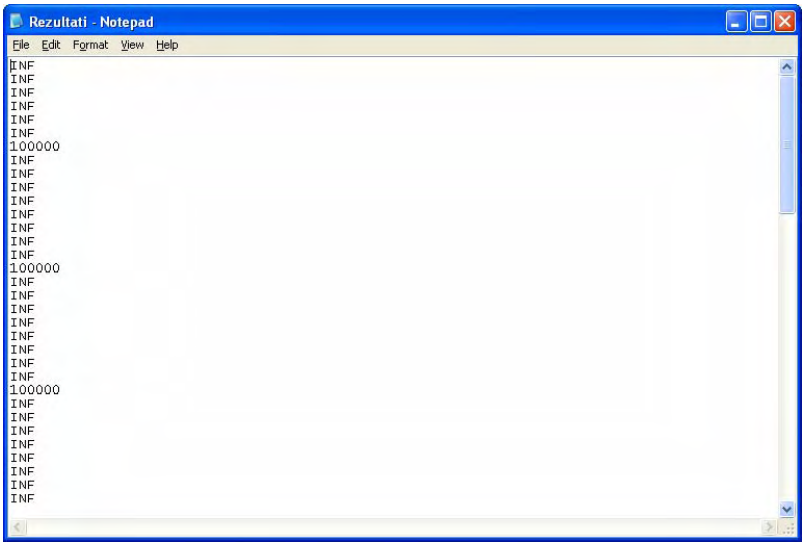
Zatvorite prozor klikom na OK. *Release* verzija izvršnog programa biti će načinjena prvim pokretanjem programa ili izborom „*Make*“ ili „*Build*“ sa „*Project*“ podmenija. Da bi pristupili mjerenju potrebno je zatvoriti sve aplikacije, uključujući i sam *Builder*.

Pokrenimo „Lab03“ i izvršimo prvo mjerenje. Pokušajmo sa brojem ponavljanja od 1000.



Slika 3-18. Mjerenje gettime pri 1,000 ponavljanja

Na računaru sa *Intel Pentium III* procesorom na 667MHz, 512MB memorije, i *Windows XP* operativnim sistemom, rezultati mjerenja izgledaju ovako:

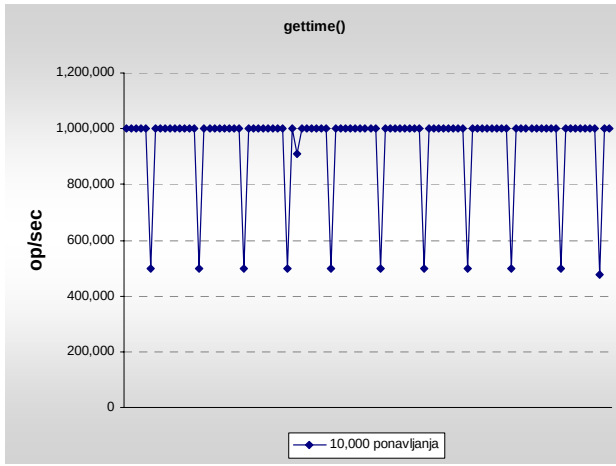


Slika 3-19. Rezultati mjerenja za 1,000 ponavljanja

Sva mjerenja koja za rezultat imaju INF (beskonačan broj operacija u sekundi) odigrala su se unutar minimalnog koraka arbitra vremena. Drugim riječima, arbitar vremena je pokazivao isto vrijeme u trenutku kada je mjerenje počelo i u trenutku kada je mjerenje završilo. To znači da je niz operacija trajao 0 vremenskih jedinica što je ekvivalentno beskonačnom broju operacija u jedinici vremena. Povremeni brojevi koji se, gotovo pravilno, pojavljuju u fajlu, rezultati su mjerenja koja su počela malo prije pomaka arbitra vremena i završila neposredno nakon pomaka

arbitra vremena. Postojanje makar i jednog INF rezultata u fajlu govori da izabrani broj ponavljanja nije dovoljno velik da bi se izmjerilo trajanje operacije.

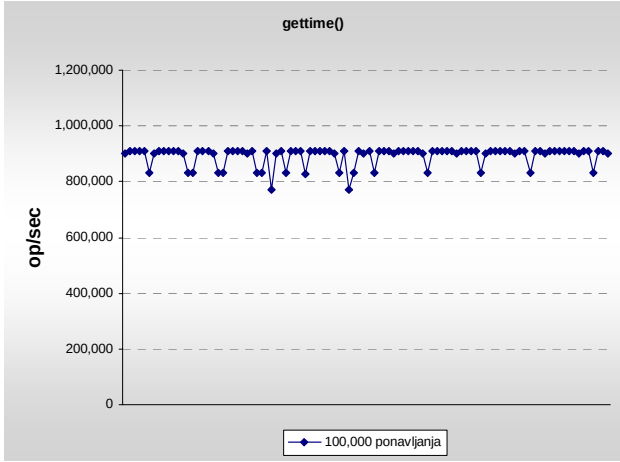
Povećajmo broj ponavljanja na 10,000 i ponovimo mjerenje. Rezultati mjerenja nemaju niti jednu INF liniju i sada je moguće nacrtati grafik rezultata pojedinačnih mjerenja kao i odrediti njihovu srednju vrijednost koja predstavlja rezultat mjerenja.



Slika 3-20. Rezultati 100 mjerenja za 10,000 ponavljanja

Srednja vrijednost mjerenja je 943,852.80 operacija u sekundi. Sa grafika je vidljivo da 90% mjerenja za rezultat daje 1,000,000 operacija u sekundi, a oko 10% mjerenja (koja se periodično ponavljaju) za rezultat daje 500,000 operacija u sekundi. Prema tome, postoji znatno odstupanje rezultata pojedinih mjerenja od srednje vrijednosti mjerenja.

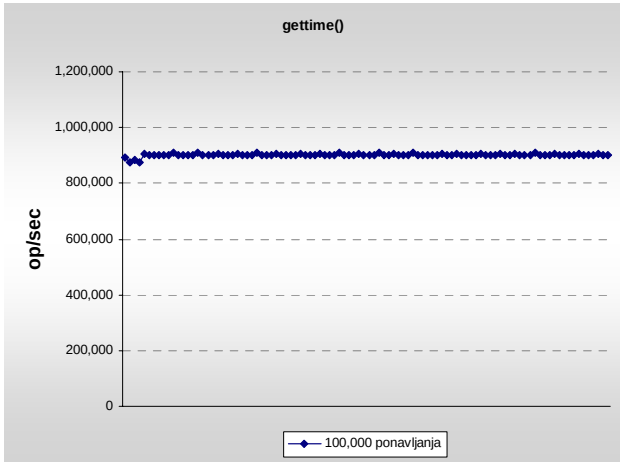
Povećajmo broj ponavljanja na 100,000 i ponovimo mjerenje.



Slika 3-21. Rezultati 100 mjerenja za 100,000 ponavljanja

Srednja vrijednost mjerenja je 892,792.41 operacija u sekundi. Sa grafika je vidljivo da postoje gotovo periodična odstupanja od srednje vrijednosti mjerenja, ali neuporedivo manja nego sa 10,000 ponavljanja.

Povećajmo broj ponavljanja na 1,000,000 i ponovimo mjerenje.



Slika 3-22. Rezultati 100 mjerenja za 1,000,000 ponavljanja

Srednja vrijednost mjerenja je 900,770.21 operacija u sekundi. Odstupanja od srednje vrijednosti mjerenja su veoma mala. Za 1,000,000 ponavljanja, mjerenje traje nekoliko minuta i dalje povećavanje broja ponavljanja rezultirati će mjerenjima čije je trajanje nekoliko desetina minuta.

Naša želja da potvrdimo tačnost rezultata provodeći mjerenje i sa 10,000,000 ponavljanja, nenadano nam je otkrila grešku u ispisu rezultata mjerenja. Naime po okončanom mjerenju sa 10,000,000 ponavljanja utvrdili smo da je srednja vrijednost mjerenja 127,094.92 operacija u sekundi, što je apsolutno kontradiktorno svim prethodnim mjerenjima. Pogledajmo još jednom izvorni kod koji ispisuje rezultate mjerenja.

```
    fprintf(IzlazniFajl, "%d\n",
        BrojPonavljanja * (long)CLK_TCK / Rezultat);
```

Dio koda na koji usmjeravamo našu pažnju je izraz za računanje rezultata mjerenja unutar kojeg se BrojPonavljanja množi sa CLK_TCK (nakon što ovaj biva pretvoren u tip long) i potom dijeli sa Rezultat.

Varijable BrojPonavljanja i Rezultat su tipa int, a CLK_TCK biva pretvoren u tip long. U 32 bitnom okruženju tip int i long su identičan tip podataka koji može predstaviti cjelobrojne vrijednosti od -2,147,483,648 pa do 2,147,483,647. Dakle rezultat ovog izraza će također biti tipa int. Ukoliko je rezultat unutar spomenutih granica, ne bi smjelo doći do prekoračenja koje bi rezultiralo odsijecanjem vrijednosti.

Rezultat koji očekujemo, znajući rezultate prethodnih mjerenja, je reda 1,000,000 što je apsolutno unutar dozvoljenih granica tipa int. Šta je onda moglo izazvati odsijecanje rezultata?

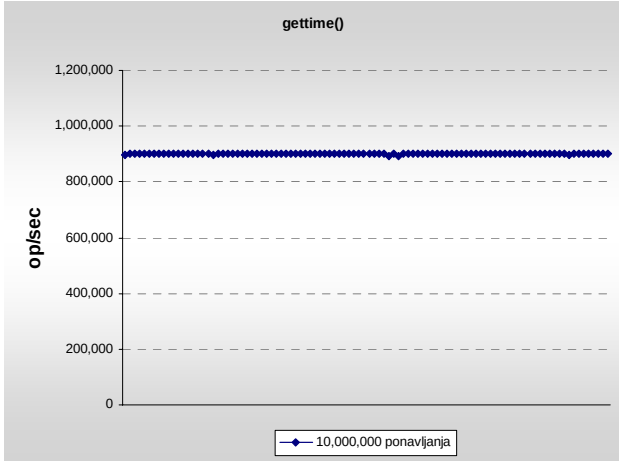
Svaki izraz se računa dio po dio, a međurezultati se smještaju u privremenu varijablu koja je tipa najkrupnijeg operanda i to najkrupnijeg po tipu, a ne po vrijednosti. Naš izraz se računa u dvije etape. Prvo se množe BrojPonavljanja i CLK_TCK da bi se dobio međurezultat koji se smješta u privremenu varijablu tipa int. Potom se međurezultat dijeli sa Rezultat da bi se dobio konačan rezultat. U konkretnom slučaju vrijednost BrojPonavljanja je 10,000,000, a vrijednost CLK_TCK je 1,000. Međurezultat je njihov produkt čija je vrijednost 10,000,000,000. Ovaj broj je prevelik za privremenu varijablu tipa int. Zbog toga dolazi do odsijecanja vrijednosti kako bi se međurezultat smjestio u privremenu varijablu. Potom se ovaj netačan međurezultat dijeli sa Rezultat.

Prekoračenje u međurezultatu se može izbjeći rearanžiranjem izraza na način da se prvo obavlja dijeljenje BrojPonavljanja sa Rezultat, a tek potom

množenje sa CLK_TCK,. Međutim u ovom slučaju cjelobrojni međurezultat će odbaciti sve cifre iza decimalnog zareza, pa će nakon množenja međurezultata sa CLK_TCK tri zadnje cifre konačnog rezultata uvijek biti nule. Bolje rješenje je konverzija članova izraza u tip double, i potom konverzija izraza natrag u int.

```
fprintf(IzlazniFajl, "%d\n", (int) (
    (double)BrojPonavljanja * (double)CLK_TCK / (double)Rezultat));
```

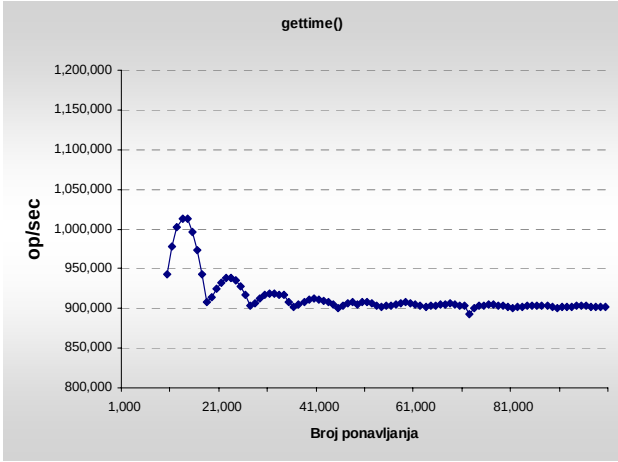
Nakon ispravka greške u ispisu spremni smo za mjerenje sa 10,000,000 ponavljanja.



Slika 3-23. Rezultati 100 mjerenja za 10,000,000 ponavljanja

Srednja vrijednost mjerenja je 901,295.77 operacija u sekundi, što se razlikuje za manje od 0.06% od prethodnog mjerenja koje je trajalo deset puta kraće.

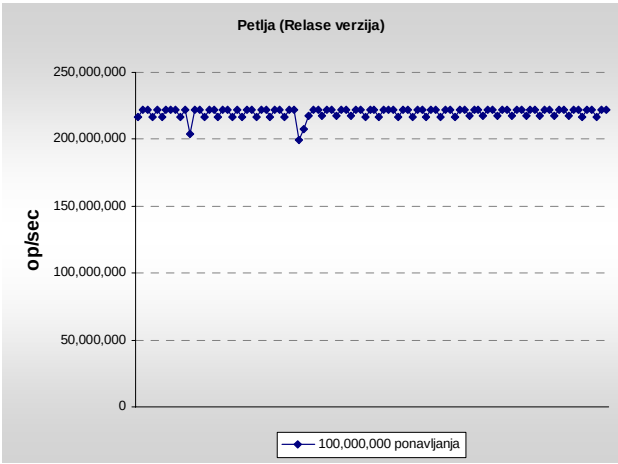
Uz nešto dodatnog truda i programiranja, moguće je izračunati rezultate većeg broja mjerenja pri različitom broju ponavljanja i tako napraviti grafik koji pokazuje kako rezultat teži svojoj konačnoj vrijednosti u zavisnosti od broja ponavljanja.



Slika 3-24. Rezultat u zavisnosti od broja ponavljanja

Grafik neodoljivo podsjeća na prigušene oscilacije.

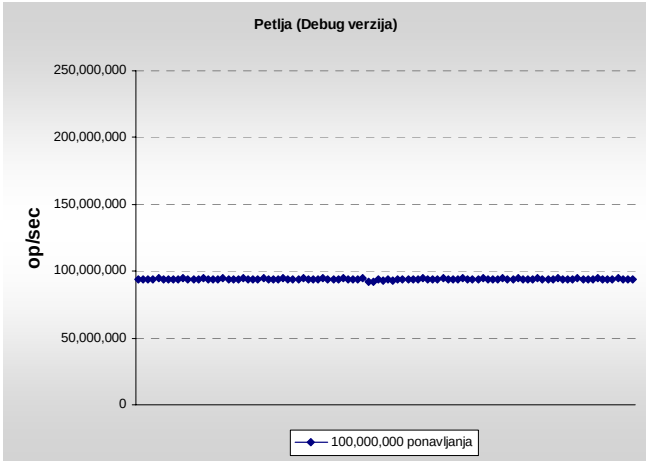
Odredimo sada koliko vremena otpada na izvršavanje same petlje unutar koje se izvršava operacija čije trajanje mjerimo. Da bi izmjerili trajanje petlje, dovoljno je iz petlje ukloniti operaciju gettime i izvršiti mjerenje na istovjetan način.



Slika 3-25. Brzina petlje u *Release* verziji

Petlja se prosječno može izvršiti 219,820,071.06 puta u sekundi. Ovaj podatak govori da je utjecaj petlje na rezultate mjerenja reda 0.41%.

Kreirajmo *Debug* verziju aplikacije za mjerenje trajanja petlje kako bi utvrdili da li postoji razlika u brzini *Release* i *Debug* verzije petlje.



Slika 3-26. Brzina petlje u *Debug* verziji

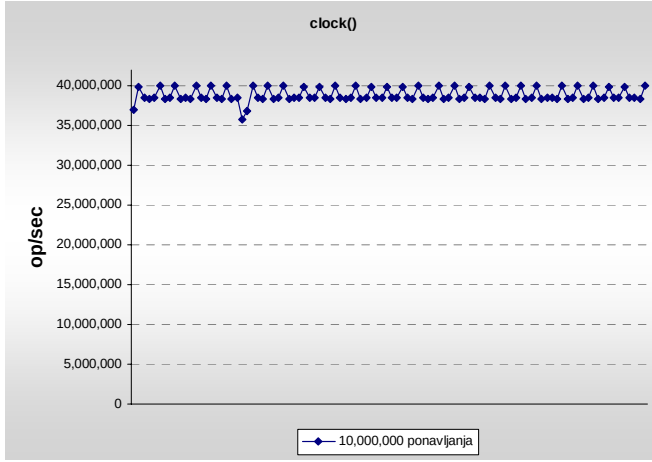
Petlja u *Debug* verziji je nešto više od dva puta sporija od petlje u *Release* verziji aplikacije.

Zadatak

Sugerisano vam je da za arbitra vremena koristite clock. Funkcija gettime također mjeri proteklo vrijeme i moguće je da bi gettime bila bolja alternativa za arbitra vremena. Znajući da obje funkcija imaju identičan minimalni interval vremena, kriterij koji odlučuje o podobnijem arbitru vremena je „koja od dvije funkcije, svojim trajanjem, unosi manju grešku u mjerenje“. Odredite trajanje funkcije clock i uporedite ga sa trajanjem gettime.

Rješenje

Dovoljno je modifikovati metod MjeriTrajanjeOperacije tako da se umjesto poziva gettime poziva clock. Potom se mjerenje obavlja na već poznati način dok se ne dođe do pouzdanog rezultata.



Slika 3-27. Rezultati 100 mjerenja za 10,000,000 ponavljanja

Srednja vrijednost mjerenja je 38,811,616.96 operacija u sekundi. Dakle clock je oko 43 puta brža operacija od gettime.