VELEUČILIŠTE U VARAŽDINU

ELEKTROTEHNIKA

PROGRAMSKI ALATI 2

Stručni studij: Elektrotehnika

„Uvod u C++“

Zadaci i upute za laboratorijske vježbe

VARAŽDIN, 2009.

Laboratorijske vježbe – općenite napomene i bodovanje

Trajanje laboratorijskih vježbi:

12 tjedana – početak od 2. tjedna predavanja

Bodovanje vježbi:

svaka vježba – 10 bodova
dvije kontrolne vježbe po 50 bodova
ukupno 200 bodova

Plan laboratorijskih vježbi:

1. Upoznavanje razvojnog okruženja
2. Razvojno okruženje - debugiranje. Osnovni tipovi podataka.
3. Standardne ulazno-izlazne operacije. Kontrola toka – uvjetno grananje.
4. Kontrola toka - petlje
5. Polja. Funkcije.
6. Primjena stečenog znanja
7. KONTROLNA VJEŽBA 1
8. Obrada podataka – tip *string*
9. Rad s datotekama I
10. Rad s datotekama II
11. Primjena stečenog znanja
12. KONTROLNA VJEŽBA 2
13. Nadoknade

Laboratorijske vježbe su obavezne za svakog studenta. Student je obavezan doći u svakom od termina laboratorijskih vježbi. U slučaju izostanaka student može u tjednu nadoknada nadoknaditi do najviše dva termina vježbi. Dolazak na kontrolnu vježbu je obavezan. Izostanak s kontrolne vježbe se može bodovati negativnim brojem bodova te onemogućava studentu oslobađanje od pismenog i usmenog ispita.

Plan prema datumima

Tjedan	Vježba	Tjedan	Vježba
12.10.-17.10.	1	30.11.-05.12.	8
19.10.-24.10.	2	07.12.-12.12.	9
26.10.-31.10.	3	14.12.-19.12.	10
02.10.-07.11.	4	11.01.-16.01.	11
09.10.-14.11.	5	18.01.-23.01.	**KONTROLA 2**
16.10.-21.11.	6	25.01.-30.01.	NADOKNADE
23.10.-28.11.	**KONTROLA 1**		

Uvjet za potpis iz kolegija

Kako bi stekao pravo na potpis student mora sakupiti **minimalno 80 bodova** tijekom laboratorijskih vježbi.

Oslobađanje od pismenog i usmenog ispita

U slučaju kada student tijekom semestra urednim izvršavanjem zadataka sakupi više od 130 bodova oslobađa se polaganja pismenog i usmenog dijela ispita te mu se dodjeljuje ocjena sukladna tablici:

Bodova	Ocjena
130 – 150	Dovoljan (2)
151 – 175	Dobar (3)
176 – 190	Vrlo dobar (4)
191 – 200	Odličan (5)

Ovako formirana ocjena upisuje se na prvom ispitnom roku nakon održanih vježbi. Student je dužan prijaviti ispit za prvi ispitni rok na kojem mu se ocjena upisuje u indeks.

Prvim izlaskom na pismeni ispit student gubi pravo na oslobađanje od pismenog i usmenog ispita.

Pismeni i usmeni ispit

Ukupan broj bodova koje student može dobiti polaganjem pismenog i usmenog dijela ispita je 100:

- 50 bodova za pismeni ispit
- 50 bodova za usmeni ispit

Dobiveni bodovi se zbrajaju s bodovima vježbi. Ocjena se formira prema slijedećoj tablici:

Bodova	Ocjena
120 – 200	Dovoljan (2)
201 – 250	Dobar (3)
251 – 275	Vrlo dobar (4)
276 – 300	Odličan (5)

Razvojno okruženje

Cilj laboratorijskih vježbi iz kolegija „Programski alati II“ je omogućiti studentima savladavanje osnova programskog jezika C++ i učiniti ih sposobnima samostalno implementirati određene funkcionalnosti korištenjem nekog od dostupnih razvojnih okruženja. Na laboratorijskim vježbama se koristi besplatna inačica Microsoftovog Visual C++ razvojnog okruženja – *Visual C++ 2008 Express Edition*.

Alternativna besplatna razvojna okruženja koje studenti mogu proučiti i koristiti u razvoju su *Dev C++*, *Turbo C++ Builder*, te dodaci za C++ programiranje u primarno Java razvojnim okruženjima *NetBeans* i *Eclipse*.

NAPOMENA:
Sve vježbe u laboratoriju se odrađuju u Visual C++ Express Edition, te je to jedino okruženje koji studenti koriste – sve zadatke je potrebno realizirati u tom okruženju, neovisno od toga što se koristi u slobodno vrijeme ili za vježbu! Budući da se prilikom ocjenjivanja među ostalim ocjenjuje i snalažljivost i korištenje razvojnog okruženja, mole se studenti da o tome vode računa!

Adrese za preuzimanje besplatnih razvojnih okruženja:

Visual C++ 2008 Express Edition – http://www.microsoft.com/Express/VC/
Dev C++ - http://www.bloodshed.net/devcpp.html
Turbo C++ Builder Explorer – http://www.turboexplorer.com/cpp
NetBeans C++ - http://www.netbeans.org/features/cpp
Eclipse C++ - http://www.eclipse.org/cdt

Preporuka za online čitanje http://msdn.microsoft.com/en-us/beginner/cc305129.aspx

Literatura:
Schildt: „C++: The Complete Reference“,McGraw-Hill/Osborne, 2003.
Stroustrup: „The C++ Programming Language“, Addison-Wesley, 1997.

VJEŽBA 1 – Upoznavanje razvojnog okruženja

Razvojno okruženje koje se koristi na laboratorijskim vježbama je *Visual C++ 2008 Express* (u nastavku VC++). Radi se o profesionalnom razvojnom alatu tvrtke Microsoft koji uključuje sve osnovne funkcije potrebne za razvoj.

U prvoj vježbi, studenti se upoznaju sa korisničkim sučeljem i osnovnim funkcijama razvojnog okruženja. Nakon uspješno završene vježbe, studenti bi trebali biti sposobni:

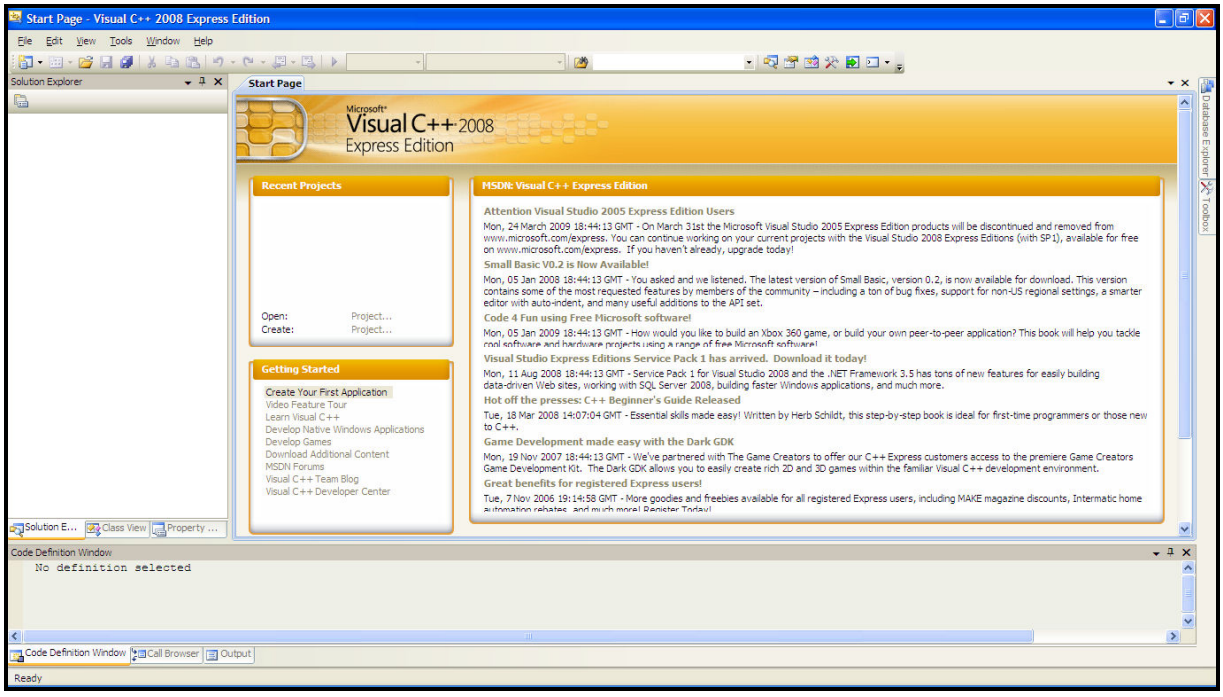
- samostalno otvoriti novi projekt
- samostalno obrađivati i korektno oblikovati izvorni kod programa
- samostalno pokrenuti program i uočiti i ispraviti eventualne pogreške u kodu

Bodovanje (maksimalno 10 bodova) će provesti asistent prema procjeni snalaženja u razvojnom okruženju i prema sposobnostima snalaženja i izvršenju postupaka opisanih u tekstu vježbe. S razumijevanjem pročitati kompletne upute za provođenje vježbe!

Programski projekt u VC++

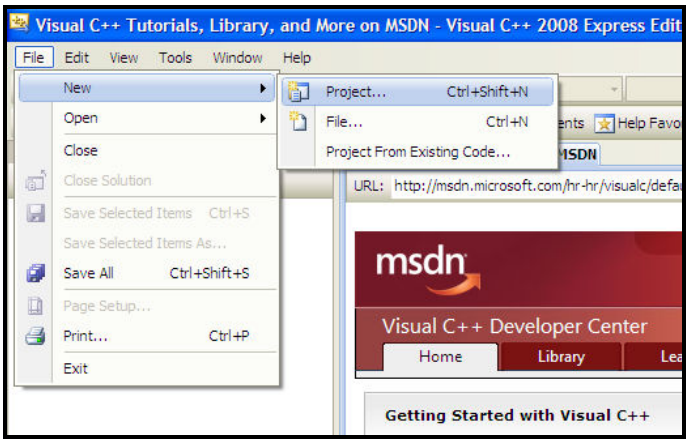
Na *Moodle* stranicama kolegija je dostupan *Flash* video prikaz otvaranja novog projekta

VC++ se pokreće iz *Start* izbornika, stavkom *Programs – Microsoft Visual C++ 2008* Express *Edition* ili korištenjem prečice postavljene na *Desktop*. Osnovno korisničko sučelje kod prvog pokretanja VC++ prikazuje slika 1.



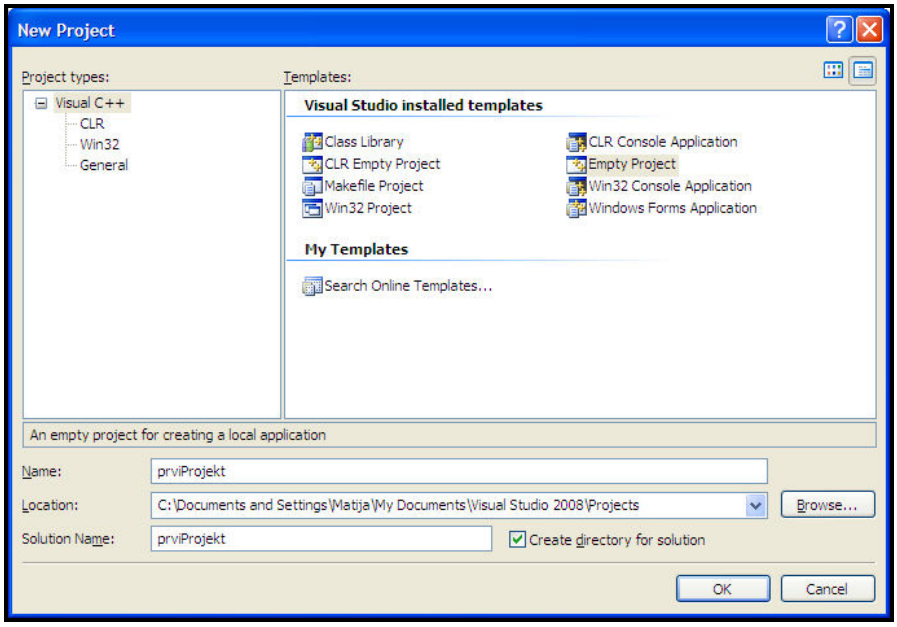
Slika 1: Osnovno korisničko sučelje VC++

Da bi započeli s programiranjem, potrebno je napraviti novi projekt (ili učitati neki postojeći ukoliko se *nastavlja* rad nad nekim već započetim programom). Izrada novog projekta započinje izborom stavke *File* - *New* - *Project* u glavnom izborniku (slika 2).



Slika 2: Izrada novog projekta

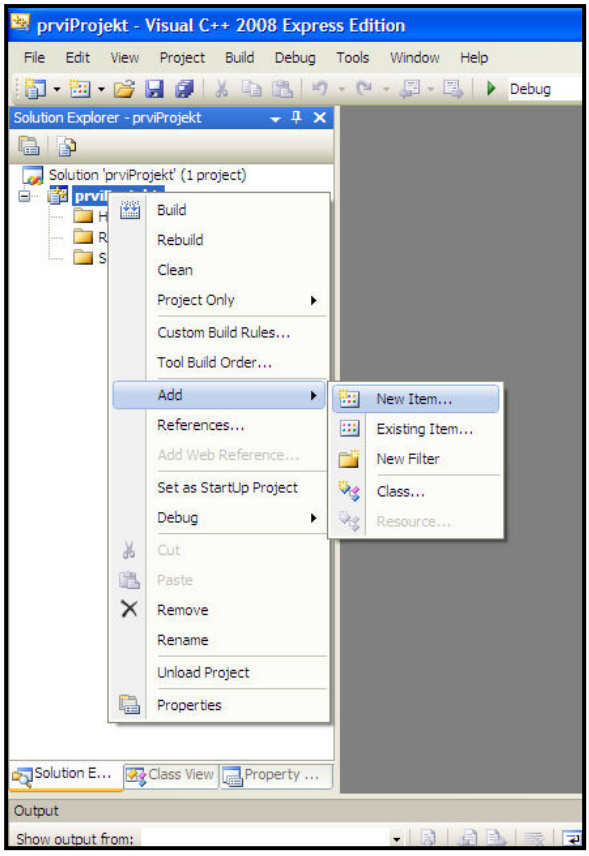
Otvara se prozor u kojem se definira tip predloška, naziv projekta i direktorij u koji će se projekt snimati.



Slika 3: Definiranje tipa i naziva projekta

Bitno je kao tip projekta u popisu *Project types* odabrati ***Empty Project*** kao početni predložak za sve programe koji će biti izrađivani u okviru ovih laboratorijskih vježbi. Nakon definiranja naziva (*Name*) projekta i lokacije (*Location*) na koju će se isti snimati, kliknuti OK. Naziv projekta odrediti u skladu s vježbom i zadacima koji se implementiraju.

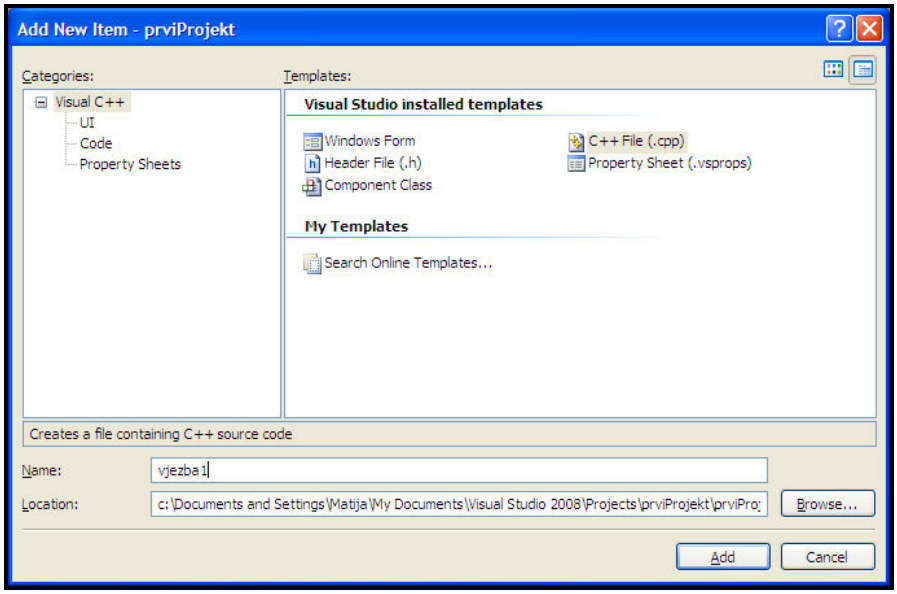
Sve datoteke s kodom i ostale datoteke koje VC++ koristi snimaju se u direktorij koji se definira u polju Location. Nakon uspješnog prevođenja (kompajliranja) u tom direktoriju (odnosno Debug i Release poddirektorijima) je moguće pronaći izvršnu .exe datoteku koju je moguće pokrenuti preko komandne linije ili direktno iz Windows Explorera. U radu s probnim programima isprobati pokretanje programa na taj način!



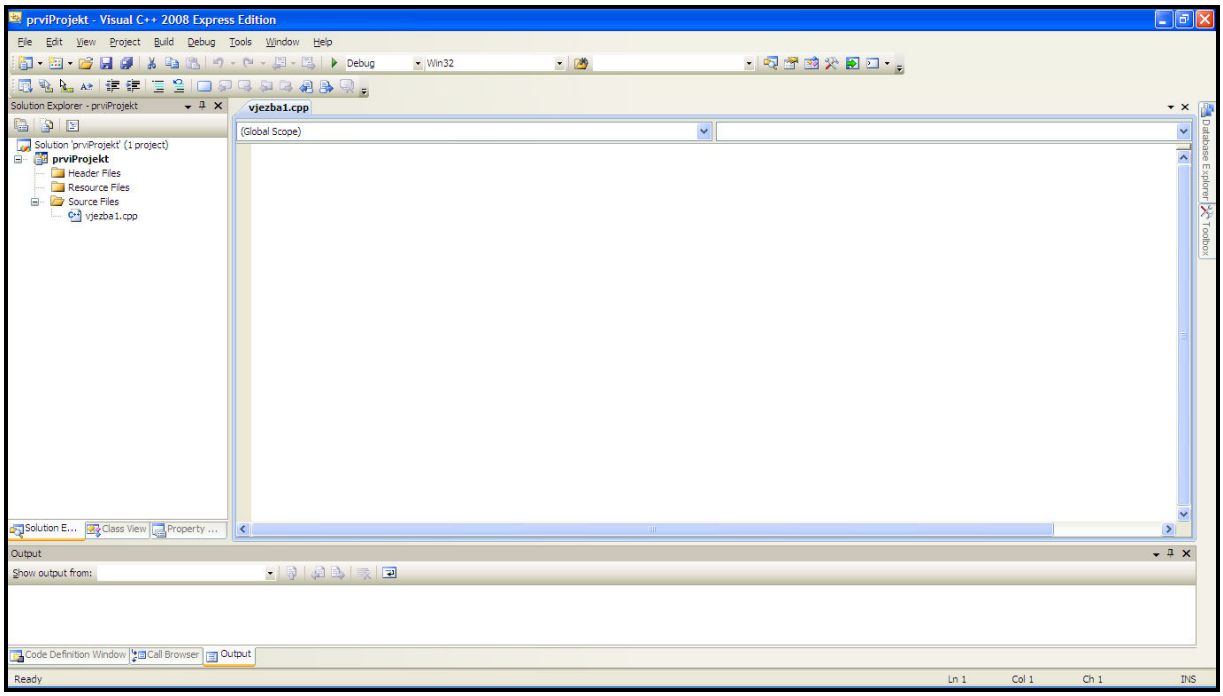
Novi prazni projekt ne sadrži nijednu datoteku, pa je za početak unosa programskog koda potrebno napraviti novu. Desnom tipkom miša iznad naziva projekta odabrati *Add – New Item* (ili u glavnom izborniku *Project – Add New Item*), kao što prikazuje slika 4.

Slika 4: Dodavanje nove datoteke projektu

VC++ nudi mogućnost dodavanja različitih tipova datoteka - za unos programskog koda otvaramo novu .cpp datoteku (*C++ File*), kao što prikazuje slika 4. U donjem dijelu prozora definiramo naziv datoteke.



Slika 5: Dodavanje CPP datoteke projektu, definiranje naziva



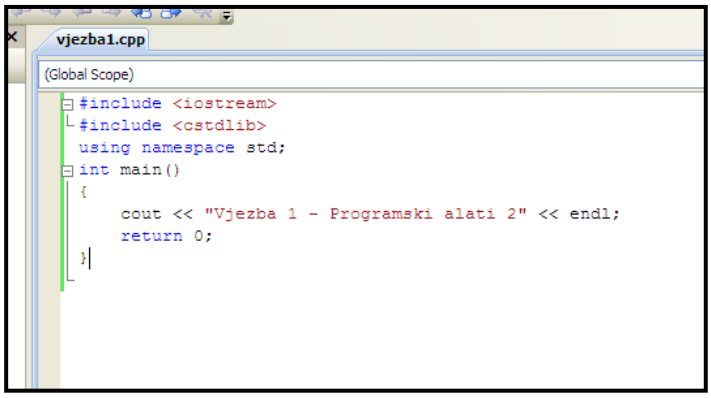
Slika 6: Sučelje za unos programskog koda

Po stvaranju nove datoteke otvara se prozor koji omogućava unos programskog koda.

Obrada (unos) izvornog C++ koda

Unesite sljedeći kôd:

```
#include <iostream>
using namespace std;
int main()
{
      cout << "Vjezba 1 - Programski alati 2" << endl;
      return 0;
}
```



Slika 7: Automatsko formatiranje koda

Pri unosu VC++ formatira uneseni kod, bojama označava ključne riječi itd. – sve kako bi programeru olakšao posao i snalaženje u kodu. Naravno, te korisne funkcije do izražaja dolaze tek kod izrade složenijih programa.

Pokretanje programa

Uneseni program je prije pokretanja potrebno prevesti – programsko okruženje to u pravilu obavlja za korisnika/programera (iako i dalje postoji mogućnost prevođenja direktno iz komandne linije). U slučaju VC++ za pokretanje programa je dovoljno pozvati funkciju *Start Without Debugging* u izborniku *Debug* (ili skraćeno kombinacijom tipki CTRL + F5).

U slučaju da želimo analizirati izvođenje programa (*Debug*), program pokrećemo stavkom *Debug – Start Debug* u glavnom izborniku (tipka F5).

Da bi se izvršilo prevođenje programa i stvorila izvršna .exe datoteka potrebno je u izborniku *Build* pokrenuti stavku *Build naziv_projekta* odnosno Rebuild *naziv_projekta*. Kao što je prije navedeno, izvršne datoteke (.exe) se spremaju u direktorij određen kod izrade projekta – pokuštajte nakon stvaranja izvršne datoteke pronaći istu i pokrenuti ju iz komandne linije Windowsa i direktno pokretanjem iz Windows Explorera!

Oblikovanje koda

Kratki primjer koda koji smo unijeli predstavlja najjednostavniji mogući C++ program koji se svodi na ispis određenog teksta na standardni izlaz (`cout`). Čak i funkcija `main` (prema standardu jezika C, početna tj. ulazna funkcija programa koja se prva izvršava) je pojednostavljena time da ne prima ulazne parametre.

Unesimo sada malo složeniji kod, baš u obliku danom u nastavku:

```
#include<iostream>
using namespace std;int main(int argc,char*argv[])
{int a;
cout<<"primjer koda"<<endl;
cout<<"novi redak (endl) ili \\n\n";
cout<<"\n\n";
cout<<"broj parametara: "<<argc<<endl;
cin>>a;return 0;}
```

Očito je da kod izgleda neobično, neuređeno. Nečitak je već i u ovom jednostavnom primjeru. Isti kôd oblikovan prema nekim standardnim pravilima mogao bi izgledati kao:

```
#include <iostream>

using namespace std;
int main(int argc, char *argv[])
{
	int a;
	cout << "primjer koda" << endl;
	cout << "novi redak (endl) ili \\n\n";
	cout << "\n\n";
	cout << "broj parametara: " << argc << endl;
	cin.get();

	return 0;
}
```

Iz primjera je jasno da je dobro oblikovan mnogo čitljiviji. Samim time, programiranje uz dobro oblikovanje koda je lakše i povećava produktivnost programera. Većina programskih okruženja nudi mogućnost poluautomatskog oblikovanja koda pri unosu. Kod unošenja neoblikovanog koda sa slike, primijetite da VC++ sam započinje formatiranje, tako da je u stvari teže unijeti loše oblikovan od dobro oblikovanog koda. Oba programa (uređeni i neuređeni kôd) se mogu prevesti i pokrenuti bez ikakvih problema. Ono što nedostaje u gornjem kodu su neophodni razmaci i prazne linije koje bi ga učinile čitljivijim. Unutar pojedine linije kôda čitljivost se može povećati ubacivanjem razmaka između pojedinih naredbi i znakova. Tako je npr. dio koda:

```
int argc,char*argv[]
```

sintaksno ispravan no kada bi bio napisan kao

```
int argc, char *argv[]
```

tada dobiva na svojoj čitkosti. Primijetite ubačene razmake. Navedeni dio kôda biti će sintaksno ispravan i slučaju kada se napiše na slijedeći način:

```
int argc, char * argv []
```

no ovime njegova čitkost nije povećana. Ključ je u dobroj mjeri. Prvi primjer nema niti jedan razmak te je teško razabrati logičke cjeline dok treći primjer ima previše razmaka te je opet teško prepoznati logičke cjeline. Kôd će znatno dobiti na čitkosti u slučaju kada se unutar linije, ali i unutar kompletnog programa, oblikuje s obzirom na logičke cjeline. Sama struktura programa kada se baci letimični pogled na njega mora odavati njegove logičke cjeline, što u osnovi znači da se operacije i pojmovi koje čine cjelinu grupiraju tako da se iz njihovog fizičkog smještaja može vidjeti logička struktura programa.

Povećanju čitkosti koda posebno doprinosi pravilno uvlačenje blokova koda. Blok koda započinje u novom retku naredbom nakon koje slijedi otvorena vitičasta zagrada „**{**„ te se nastavlja naredbama koje su uvučene s obzirom na blok koji ih sadržava. Blok završava zatvorenom vitičastom zagradom „**}**“ koja se nalazi u stupcu naredbe početka bloka. Uputno je na kraj bloka uz vitičastu zagradu staviti komentar u kojem piše kojem bloku pripada vitičasta zagrada. Česta sintaksna pogreška u kôdu je neodgovarajući broj zagrada. Naredbe se tijekom pisanja kôda uvlače koristeći tipku TAB.

```
blok {
|
|       naredba1;
|       naredba2;
|       naredba3;
|
|       unutarnji_blok_razine_1 {
|       |
|       |     naredba4;
|       |     naredba5;
|       |
|       |     unutarnji_blok_razine_2 {
|       |     |
|       |     |     naredba6;
|       |     |     naredba7;
|       |     |
|       |     } // kraj bloka 2
|       } // kraj bloka 1
} // kraj glavnog bloka
```

Otkrivanje pogrešaka u krivo unesenom kodu

Preduvjet uspješnog prevođenja jest sintaksna ispravnost programskog koda. Drugim riječima, kod mora zadovoljavati određena pravila definirana standardom (gramatika jezika) – npr. razdvajanje naredbi znakom „;“, dodijeljivanje vrijednosti varijablama sa „=“ i slično. To su pravila koja se provjeravaju prije prevođenja i ukoliko nisu zadovoljena razvojno okruženje će prijaviti tzv. sintaksne pogreške. Tek kad je kod sintaksno ispravan prelazi se na prevođenje. Naravno, to ne garantira ispravno funkcioniranje programa jer ono ovisi isključivo o logici ugrađenoj u kod.

VC++ prije pokretanja programa vrši sintaksnu provjeru koda. Ukoliko se uoče nedostaci i pogreške u kodu, to se signalizira programeru. Neka razvojna okruženja osim provjere prije pokretanja prevođenja programa nude i provjeru sintakse u stvarnom vremenu, pa kod samog upisa programskog koda naznačuju moguće pogreške.

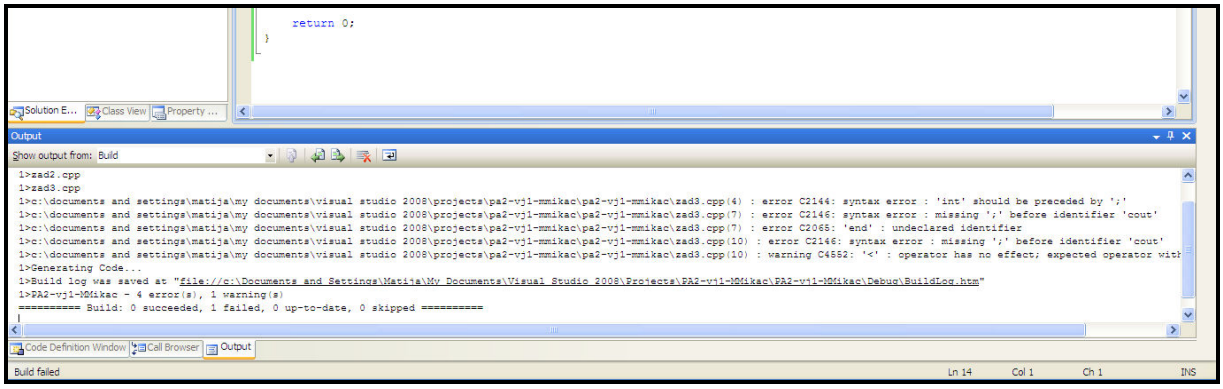
Unesite sljedeći kod:

```
#include <iostream>

using namespace std
int main(int argc, char *argv[])
{
	int a
	cout << "primjer koda" << end;
	cout << "novi redak (endl) ili \\n\n";
	cout < "\n\n"
	cout << "broj parametara: " << argc << endl;
	cin >> a;

	return 0;
}
```

Pokrenite program sa CTRL+F5. Zbog pogrešaka u kodu kompajliranje nije moguće, te VC++ o tome obavještava u prozoru na dnu ekrana.



Proučite poruke o pronađenim pogreškama i upozorenja i pokušajte ih ispraviti.
Ukoliko uspješno ispravite pogreške, program bi se trebao uspješno izvršiti.

VJEŽBA 2 – Osnovni tipovi podataka. Analiza izvođenja programa.

U ovoj vježbi studenti se upoznaju s osnovnim tipovima podataka i njihovim korištenjem kroz gotove primjere. Na istim primjerima programskog koda upoznaju se s mogućnostima analize izvođenja programa i vrijednosti varijabli, korištenjem debuggera ugrađenog u VC++ razvojno okruženje. Nakon uspješno završene vježbe, studenti bi trebali biti sposobni:

- objasniti razliku među osnovnim tipovima podataka
- samostalno odabrati odgovarajući tip podataka za zadane varijable
- samostalno analizirati izvođenje programa i pratiti vrijednosti varijabli pri izvođenju

Bodovanje (maksimalno 10 bodova) će provesti asistent prema procjeni snalaženja u razvojnom okruženju i na temelju znanja koje student pokaže u usmenom ispitivanju. S razumijevanjem pročitati kompletne upute za provođenje vježbe!

Osnovni tipovi podataka

Programski jezik C++ preuzima osnovne tipove podataka iz C-a. U osnovne tipove se ubrajaju `char`, `int`, `float` i `double` tipovi, s tim da se u novijim verzijama C-a uvodi tip `wchar_t`, a C++ dodaje tip `bool`. Dodatno, osnovne tipove je moguće koristiti uz modifikacije ključnim riječima (tzv. modifieri) `signed`/`unsigned`, te im povećavati ili smanjivati broj bita koje koriste za pohranu vrijednosti korištenjem ključnih riječi `short` i `long`. Tablica u nastavku daje pregled tipova i modificiranih varijanti, zajedno s uobičajenom veličinom memorije potrebne za pohranu podataka određenog tipa (broj bita korištenih za pohranu može ovisiti o arhitekturi računala – npr. na starim računalima `int` je 16 bitni.)

Tip i varijante	Opseg (moguće vrijednosti)	Broj bita
`bool`	`true/false`	8

`char`
`signed char`
`unsigned char`
	-128 .. 127
	-128 ... 127
	0 .. 255
		8

`int`
`signed int`
`unsigned int`
`(long int` identično `int!)`
	-2^{31} ... $2^{31}-1$
	-2^{31} ... $2^{31}-1$
	0 ... $2^{32}-1$
		32

`short int` `unsigned short int`	-2^{15} ... $2^{15}-1$ -32768... 32767 0 .. $2^{16}-1$ 0... 65535	16
`float`		32
`double`		64
`long double`		80 (64)
`wchar_t`		16

Svaka varijabla koja se koristi u programu mora se deklarirati – definira se tip i naziv varijable. C++ je za razliku od nekih drugih programskih jezika (npr. *Pascal, C* do standarda C99) po tom pitanju vrlo fleksibilan i omogućava deklaraciju varijabli doslovno na svakom mjestu u programu, s tim da su deklarirane varijable vidljive samo unutar bloka koda u kojem

su deklarirane (radi se o lokalnim varijablama, a njihov doseg uključuje samo aktualni blok koda) . Ipak, zbog lakše čitljivosti i boljeg razumijevanja koda preporuka je varijable deklarirati na početku funkcijskih blokova (npr. odmah na početku `int main()` funkcije).

Osnovni operatori

Operatori u C++ služe za izvršavanje određenih operacija među varijablama. Osnovni operatori koji se koriste na ovoj vježbi su: operator pridruživanja =, aritmetički operatori +, -, *, /, %, ++, --, logički operatori >, <, ==, != i "posebni" operator ?. Operator ? Je specifičan po tome da omogućava logičku provjeru nekog izraza, te ovisno o tom statusu kao rezultat daje jednu od dvije ponuđene vrijednosti.

Zadatak

Unesite navedeni kod i pokrenite program:

```
1  #include <iostream>
2  using namespace std;
3
4  int main()
5  {
6          char varChar;
7          int varInt;
8          short int varSInt;
9          unsigned short int varUSInt;
10         float varFloat;
11         bool valBoolean;
12         long double varLDouble;
13
14         varInt=258;
15         varChar=varInt;
16
17         varLDouble=3.12345678901234567890;
18         varFloat=varLDouble;
19
20         cout << "varChar    = " << varChar << "  (int)varChar = " << (int)varChar << endl;
21         cout << "varInt     = " << varInt << endl << endl;
22
23         cout.precision(16);
24         cout << "varLDouble = " << varLDouble << endl;
25         cout << "varFloat   = " << varFloat << endl << endl;
26
27         varInt=60000;
28         varSInt=varInt;
29         varUSInt=varInt;
30
31         cout << "varInt     = " << varInt << endl;
32         cout << "varSInt    = " << varSInt << endl;
33         cout << "varUSInt   = " << varUSInt << endl;
34
35         cin.get();
36
37         return 0;
38 }
```

Zapišite i objasnite rezultate ispisa:

Zbog čega ispis varijable `varChar` u retku 20 ne vraća brojčanu vrijednost?

Što znači `(int)` prije ispisa varijable `varChar` u retku 20?

Iskoristite funkciju `sizeof()` i saznajte broj okteta potrebnih za pohranu određenih tipova. Zapišite rezultate funkcije za osnovne tipove – `char`, `short int`, `int`, `bool`, `float`, `double`, `long double`. Postoje li razlike u odnosu na tablicu s definicijama tipova?

```
sizeof(char)      =                    sizeof(float)          =
sizeof(short int) =                    sizeof(double)         =
sizeof(int)       =                    sizeof(long double)    =
sizeof(bool)      =
```

Dodajte navedeni kod nakon zadnjeg ispisa u programu (redak 34):

```
varBoolean = (varSInt<0) ? true:false;
cout << "varBoolean = " << boolalpha << varBoolean << endl;
varBoolean = (varInt<0) ? true:false;
cout << "varBoolean = " << varBoolean << endl;
```

Izvršite program i objasnite dobiveni rezultat unesenog koda!
Izvedite isti kod, ali prije izvođenja uklonite boolalpha naredbu. Opišite razliku.

Koja je maksimalna vrijednost koju je moguće dodijeliti varijabli `varChar`?

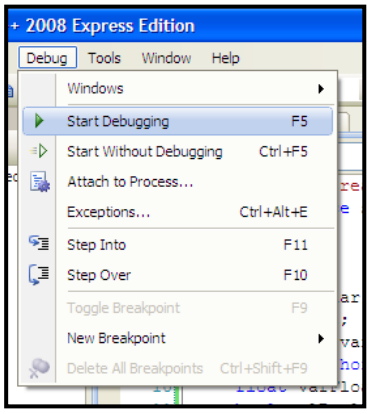
Koliko bitova je potrebno za spremanje `bool` varijable, a koliko ih se koristi u praksi?

Analiza izvođenja – debugger

Izvođenje programa moguće je nakon što je programski kod uspješno preveden. Kao što je opisano u vježbi 1, uspješnom prevođenju prethodi provjera sintaksne ispravnosti koda. No, izvođenje programa ne znači da će program od prve dobro funkcionirati. Bez priprema (a i s

njima), programer često u radu može naići na probleme – može doći do nehotičnog uvođenja neke logičke pogreške u kod, određeni dijelovi koda mogu zbog pogreške (sintaksno ispravno!) izvesti neplanirane operacije (npr. pogreška u programskoj petlji i slično)...

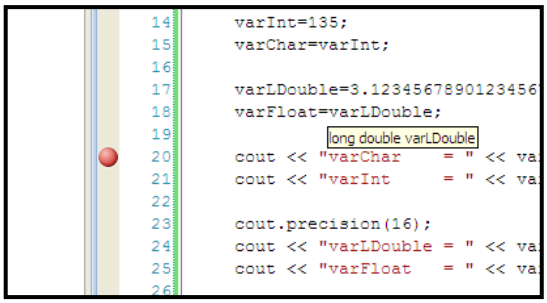
Zbog toga je prije produkcijske faze (korištenje programa u svrhu za koju je i izrađen) potrebno obaviti testiranja programa. U složenijim programima mogu se koristiti različiti standardni postupci testiranja, no najčešće sve počinje tzv. debuggiranjem – analizom izvođenja programa i praćenjem ponašanja promjenjivih podataka (varijabli). Sasvim je izvjesno da u slučaju nepravilnog rada programa negdje na putu do rezultata mora postojati neka logička pogreška – budući da programer razumije što je isprogramirao i zna koja su mu očekivanja, praćenjem vrijednosti varijabli kroz izvođenje programa može u gotovo svim slučajevima utvrditi gdje se u kodu (ili u samom postupku – dakle, u razmišljanju i ideji rješenja problema) nalazi greška.



Slika 1: *Debug* izbornik

Pokretanje programa uz analizu izvođenja (debuggiranje) je moguće funkcijom *Debug – Start Debugging* u glavnom izborniku (slika 1), odnosno tipkom F5. Ukoliko programer ne intervenira u samom razvojnom okruženju i ne definira točke u kojima želi da se izvođenje zaustavi i omogući analiza varijabli i ostalih parametara, izvođenje će biti identično onom bez debuggiranja (CTRL + F5).

Slika 2: *Breakpoint* oznaka



Točke u kojima se zaustavlja izvođenje programa nazivaju se *breakpoint*-ovi. Točke se definiraju jednostavno dvoklikom miša uz lijevi rub koda – oznaka točke izvođenja je vidljiva na slici 2. Nakon što su točke zaustavljanja postavljene, pokretanjem sa F5, izvođenje će stati na željenom mjestu, a programeru će se dozvoliti analiza korištenjem proširenog *Debug* izbornika i pomoćnog padajućeg izbornika u glavnom prozoru.

Neke od mogućnosti debuggiranja:

- ukoliko za vrijeme prekida izvođenja kursor miša dovučemo iznad neke varijable ili parametra u kodu, VC++ daje osnovne informacije o aktualnoj vrijednosti varijable ili neke druge podatke u informativnom prozoru
- moguće je definiranje tzv. *watch*-eva – definira se izraz po želji (npr. naziv varijable, neka funkcija nad varijablama – npr. `3*varInt+5*varChar`), a VC++ u prozoru prikazuje vrijednosti tih izraza (ukoliko ih je moguće odrediti prema stanju izvođenog programa)
- programer može u potpunosti kontrolirati daljnji tok izvođenja – dopustiti izvođenje do sljedeće točke zaustavljanja (*Continue* F5), nastaviti izvođenje redak po redak (*Step Over* F10) i nakon svakog retka iskoristiti mogućnost analize stanja
- ostale mogućnosti uključuju pregled statusa procesorskih registara, ... itd.. što nije nužno za izvođenje ovih vježbi

Na *Moodle* stranicama kolegija je dostupan *Flash* video prikaz analize izvršavanja programa

Zadatak

Unijeti kod u nastavku – radi se o programu koji mora na osnovu unesenih duljina stranica pravokutnika izračunati opseg, površinu i duljinu dijagonale pravokutnika. Ispraviti moguće **sintaksne greške** u kodu, uspješno prevesti i pokrenuti program (CTRL+F5).

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4
5  int main()
6  {
7          float a,b;
8          double c;
9
10         cout << "Unesite duljinu stranice a = "; cin >> a;
11         cout << "Unesite duljinu stranice b = "; cin >> b;
12
13         c=a+2*b;
14         cout << "Opseg pravokutnika = " << c << endl;
15
16         c=a*a;
17         cout << "Povrsina pravokutnika = " << c << endl;
18
19         c=sqrt(a+b*b);
20
21         a=a-1
22         b=b+1:
23         c=2*a+b;
24
25         cin.get();
26
27         return c;
28 }
```

Koje sintaksne greške ste pronašli i ispravili?

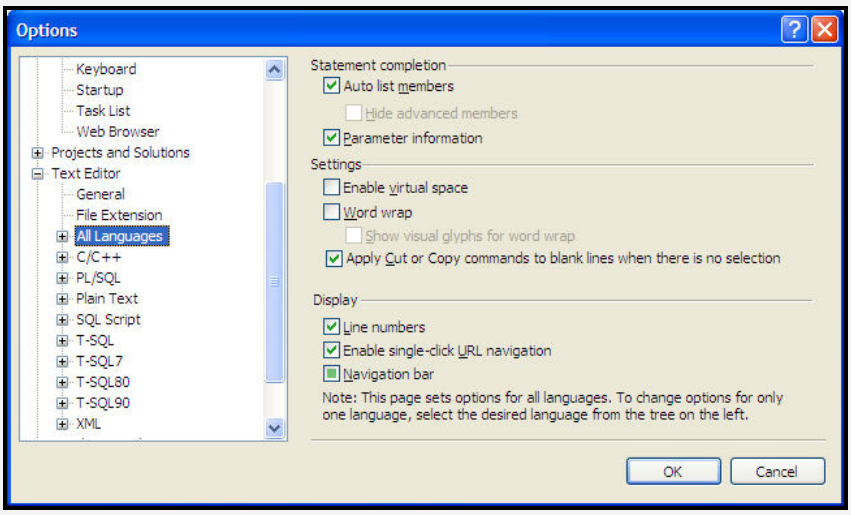
Nakon uspješnog pokretanja i završetka izvođenja definirajte točke prekida u retku 19 i 23. Pokrenite program, unesite za vrijednosti `a`=5 i `b`=4, te analizirajte izvođenje – koje vrijednosti poprima varijabla `c`?

Prije izvršena linije 19	`c =`
Nakon izvršenja linije 19	`c =`
Nakon linije 23	`c =`

Korištenjem izbornika *Debug* promatrajte vrijednosti svih lokalnih varijabli (*Debug – Windows – Locals*), definirajte proizvoljne *watch*-eve (npr. 2*a+2*b, sqrt(a*a+b*b)).

Ispraviti **logičke pogreške** u kodu, dodajte kod koji će ispisati duljinu dijagonale.
Koje logičke pogreške ste pronašli i ispravili?

Za lakše snalaženje u kodu, preporuča se uključiti prikaz broja redaka – za uključivanje prikaza koristiti *Tools – Options* u glavnom izborniku, te odabrati *Text Editor – All Languages* (ili *C/C++*) kao na slici i uključiti stavku *Line numbers*.



VJEŽBA 3 – Standardne ulazno-izlazne operacije. Uvjetno grananje

Na dosadašnjim vježbama i predavanjima studenti su upoznati sa osnovnim ulazno-izlaznim operacijama, dovoljnim za samostalnu izradu programa za unos i ispis podataka na ekran (standardni ulaz i izlaz – *stdin* i *stdout*, odnosno `cin`, `cout` u C++). Dodatno se obrađuju osnove kontrole toka programa. Ovom vježbom od studenata se traži veći angažman nego na prve dvije vježbe, a rezultat bi trebali biti samostalno izrađeni programi prema dodijeljenim zadacima.

Bodovanje (maksimalno 10 bodova) će provesti asistent – do 5 bodova studenti dobivaju temeljem ispravno riješenih zadataka koji im se dodijele na početku vježbe, a preostalih 5 bodova dodjeljuje asistent na temelju znanja koje student pokaže u usmenom ispitivanju.

NAPOMENA:
Programski kod koji se daje asistentu na pregled, osim ispravne funkcionalnosti, mora zadovoljavati i osnovna pravila oblikovanja koda, dakle biti čitak i jasan. Razumijevanje i znanje studenata provjerava se usmenim ispitivanjem.

Standardni ulaz i izlaz

C++ standard uvodi korištenje tokova (*stream*) za ulazno-izlazne operacije. Radi se o logičkim modelima uređaja koji podržavaju ulazno-izlazne operacije. Osnovne predefinirane ulazno-izlazne operacije uključuju dohvat podataka s tipkovnice i prikaz podataka na ekranu računala. Tokovi koji se za to koriste su `cin` i `cout`. Dodatno, postoji `cerr` izlazni tok koji se koristi za ispis pogrešaka. U slučaju spomenutih standardnih tokova radi se o preusmjeravanju na ekran (`cout` i `cerr`) odnosno preuzimanju podataka s tipkovnice (`cin`). Biblioteka za korištenje standardnih ulaza i izlaza u C++ standardu je **`<iostream>`** i potrebno ju je uključiti naredbom `#include` – alternativa je `<iostream.h>`, no radi se o starijoj inačici prema prije važećim standardima.

Slanje i primanje podataka obavlja se korištenjem operatora << (operator umetanja – *insertion operator* – koristi se kod izlaznih tokova) i >> (operator izdvajanja – *extraction operator* – koristi se kod ulaznih tokova).

Pomoćne funkcije za rad s standardnim `cout` izlazom omogućavaju prilagodbu ispisa (npr. oktalni ili heksadecimalni ispis cjelobrojnih varijabli, promjena ispisne preciznosti kod brojeva s pomičnim zarezom...). Neke od funkcija je moguće pozivati direktno na ios razredu, a dodatno postoje specijalizirane funkcije tzv. *manipulators* koje je moguće direktno uključiti u tok. Primjeri *manipulatora* su, bez detaljnih obrazloženja, dani u odvojenom okviru na kraju vježbe. Primijetite da je već korišteni `endl` također *manipulator*!

Specifičnosti unosa znakovnih nizova

U programskom jeziku C++ znakovni niz se prikazuje kao polje znakova (`char[]`). Polje može biti ograničene ili pak nedefinirane duljine (prevodilac određuje duljinu temeljem inicijalizacijskog sadržaja). Prilikom unosa s tipkovnice, na kraj takvog polja automatski se dodaje \0 (ASCII kod 0) što naznačuje kraj niza.

Međutim, unos pomoću operatora >> ne omogućava unos nizova koji sadrže razmaknicu (*space*) – kod unosa takvih nizova, samo dio niza do prve razmaknice se izdvaja i dodjeljuje varijabli. Da bi se taj problem eliminirao koristi se funkcija `cin.get` ili `cin.getline`.

U standardni jezik ugrađene su različite funkcije koje olakšavaju rad sa znakovnim nizovima (npr. `strcmp`, `strcat`, `strcpy`...).

Znakovni nizovi su preuzeti iz C-a, a novost u C++ je uvođenje tipa *string*. Za korištenje tipa potrebno je uključiti biblioteku sa **#include <string>.** Radi se o razredu (klasi) koji može sadržavati znakovne nizove, a omogućava lakšu manipulaciju podacima zahvaljujući niz ugrađenih funkcija.

Primjer unosa znakovnih nizova (`char[]` i `string`) i rezultati:

```
char chNiz[50];
cout << "Unesite niz: "; cin >> chNiz;
cout << "Unos = " << chNiz << endl;
cin.get();
```

```
Unesite niz: PA2-Elektrotehnika 2009/2010
Unos = PA2-Elektrotehnika
```

(Primijetite da iako je duljina niza 50, razmaknica prekida unos!)

```
char chNiz[50];
cout << "Unesite niz: "; cin.get(chNiz,80);
cout << "Unos = " << chNiz << endl;
cout << "Duljina = " << strlen(chNiz) << endl;
cin.get();
```

```
Unesite niz: PA2-Elektrotehnika 2009/2010
Unos = PA2-Elektrotehnika 2009/2010
```

(Uspješni unos!)

Zamijenite `cin.get` sa `cin.getline`! Primjećujete li razliku u unosu i/ili izvođenju programa?!

```

```

Korištenje tipa `string` prikazuje sljedeći primjer:

```
#include <string>                          obavezno na početku!
...
string chNiz;
cout << "Unesite niz: "; getLine(cin, chNiz);
cout << "Unos = " << chNiz << endl;
cout << "Duljina = " << chNiz.length() << endl;
cin.get();
```

Osnove pretvorbe brojeva u znakovne nizove i obrnuto

Često u praksi postoji potreba za pretvaranjem unesenog broja u znakovni niz ili obrnuto. U VC++ za pretvorbu cijelih brojeva u znakovni niz koristimo `_itoa` i `_itoa_s` funkcije (standardizirane ISO C++ funkcije za razliku od `itoa` koja je dio C standarda) – parametri koje primaju su redom broj, znakovni niz i baza – npr. `_itoa(10,niz,2)` će broj 10 u binarnom obliku zapisati u niz.

Jedan od pristupa je i korištenje `stringstream` razreda. U tok se ubaci broj proizvoljnog tipa, te ga se očita kao `string`. Primjer:

```
# include <sstream>
..
..
string pretvoreniBroj;
float fBroj;
int iBroj;

stringstream pom;

cin << fBroj;                       // učitamo float broj
pom << fBroj;                       // ubacimo u stringstream
pretvoreniBroj = pom.str();         // očitamo string (metoda str())

cin << iBroj;                       // učitamo int broj
pom << iBroj;                       // ubacimo u toko
pretvoreniBroj = pom.str();
```

Pretvorba znakovnih nizova u numeričke tipove se također obavlja ili korištenjem ugrađenih C funkcija (`atoi`, `atof` i slično) ili korištenjem `stringstream` razreda.

Primjer pretvorbe znakovnog niza u cijele i realne brojeve:

```
# include <sstream>
..
..
float fBroj;
int iBroj;

stringstream pom;

pom << "123.4567";                 // ubacimo tekst u tok
pom >> fBroj;                       // izdvojimo unos kao float

pom.clear();                        // čistimo tok
pom << "123.4567";                 // ubacimo tekst u tok
pom >> iBroj;                       // izdvajamo unos kao int (123)
```

Primjeri korištenja *manipulatora* za formatiranje i kontrolu ispisa

Za korištenje je potrebno uključiti **<iomanip>**!

izraz za ispis	**rezultat**	**opis**
`cout << hex << 100 << endl;`	64	heksadecimalno 100
`cout << setfill('*') << setw(10) << 123.45;`	****123.45	popunjavanje i širina ispisa
`cout << left << setfill('*') << setprecision(5) << setw(10) << 1234.567;`	1234.5****	popunjavanje slijeva
`cout << scientific << 123.456;`	1.23456e+002	sci prikaz decimalnih brojeva
`cout << fixed << setprecision(2) << 123.4567;`	123.46	decimalne znamenke...

ostali često korišteni manipulatori (isprobajte!):
`fixed`, `setbase(int base` – 8,10,16), `oct`, (no)`uppercase`, (no)`showbase`, (no)`showpos`, (no)`showpoint`

Kontrola toka – uvjetno grananje (if-else, switch-case)

Implementacija složene logike programa nije moguća bez kontrole toka. U C++ kontrola toka i uvjetno grananje se obavlja `if-else` blokovima i `switch-case` blokovima. Kontrola sa svodi na provjeru logičkih uvjeta (`if`) ili vrijednosti cjelobrojnih izraza (`switch`) i izvođenju različitih blokova naredbi ovisno o uvjetima. Kontrolni blokovi su shematski prikazani na slici.

```
if (uvjet1)
{
      // zadovoljen prvi uvjet
      // blok 1
} else if (uvjet2)
{
      // zadovoljen drugi uvjet
      // blok 2
} else if (uvjet3)
{
      // zadovoljen treci uvjet
      // blok 3
} else
{
      // svi ostali slucajevi
}
```

```
switch (izraz)
{
      case vrijednost1:
            // blok 1
      case vrijednost1:
            // blok 2
      case vrijednost1:
            // blok 3
            break;
      default:
            // za sve ostale
}
```

Prije spomenuta mogućnost korištenja operatora ? za kontrolu toka u stvari zamjenjuje jednostavni `if-else`. Sljedeći kod će u oba slučaja dati isti rezultat, ali je varijanta sa operatorom ? kraća. Drugim riječima,

```
varBoolean = (varSInt<0) ? true:false;
```

zamjenjuje kod:

```
if (varSInt<0)
{
      varBoolean=true;
} else
{
      varBoolean=false;
}
```

Zadaci

Iz svake grupe zadataka studentu se dodijeli 1 zadatak. Zadaci prve grupe donose 1 bod, a druge i treće po 2 boda. Na te bodove (max 5) dodaju se bodovi (max 5) temeljem usmenog ispitivanja od strane asistenta.

1. Potrošnja goriva – unijeti podatke za dva vozila (naziv i model vozila, registarsku oznaku, broj prijeđenih kilometara i količinu potrošenog goriva). Izračunajte i ispišite prosječnu potrošnju goriva na 100 km za oba vozila. Ukoliko vozilo troši manje od 6L/100km ispisati "štedljivo vozilo", 6-10L "standardno", iznad 10L "vozilo s visokom potrošnjom".
2. Određivanje kamata – odredite iznos kamate koje će dužnik platiti poreznoj upravi na kraju godine – kao parametre unesite iznos duga (glavnica G) i godišnju kamatnu stopu K. Koristite jednostavni kamatni račun. Ukoliko je zbroj duga i kamata veći od 5.000.000 kn, ispisati "veliki dužnik", manji od 10.000 "omiljeni dužnik", za ostalo "dužnik".
3. Odredite struju kroz otpornik – na otporniku otpora R izmjeren je napon U. Omogućite unos vrijednosti otpora i napona, te odredite struju i ispišite struju I koja prolazi otpornikom. Ukoliko je iznos struje manji od 1A, ispišite vrijednost u miliamperima.

---

4. Unesite ime, prezime i godinu rođenja za 3 osobe. Ispišite popis tako da prvo ispisujete prezime pa ime tako da je maksimalna širina svakog ispisa 10 znakova, zatim godinu rođenja u dekadskom, oktalnom i heksadecimalnom formatu, te na kraju ime velikim štampanim slovima. Ispišite prosječnu starost unesenih osoba! Odredite najmlađu osobu i ispišite je!
5. Trgovina - unesite naziv i cijenu bez PDV-a za tri proizvoda. Za svaki proizvod unesite prodanu količinu. Ispišite račun – naziv proizvoda velikim štampanim slovima maksimalne širine 12 znakova, jediničnu cijenu pozicioniranu desno (širina 10), količinu, ukupnu cijenu i iznos PDV-a za svaku stavku. Na kraju računa ispišite ukupni iznos za naplatu i ukupni iznos PDV-a. Odredite najskuplji proizvod!
6. Putni nalog – tri djelatnika tvrtke vraćaju se s službenog putovanja na kojem su koristili vlastita osobna vozila. Unesite ime i prezime djelatnika, broj pređenih kilometara i iznos priznatih troškova. Ispišite djelatnike te ukupne iznose koje im tvrtka treba isplatiti uz pretpostavku da za svaki pređeni kilometar dobiju 2 kn. Iznose ispišite pozicionirane desno, sa dvije decimalne znamenke. Ispišite djelatnika koji je prošao najdulji put i djelatnika kojem će se isplatiti najveći iznos.

---

7. JMBG – unesite JMBG osobe i ispišite (znak po znak) datum rođenja osobe! Odredite zbroj brojeva (znamenki) u datumu rođenja, ispišite ga kao cijeli broj i kao znakovni niz.
8. Unesite datum rođenja osobe u formatu DDMMGGGG. Ispišite datum rođenja u formatu DD/MM/GGGG! Odredite zbroj brojeva (znamenki) u datumu rođenja, ispišite ga kao cijeli broj i kao znakovni niz.
9. Unesite tri cijela broja, generirajte niz znakova koji sadrži te brojeve razdvojene znakom +. Ispišite niz znakova. Pretvorbu brojeva u znakovni niz izvedite na oba prije opisana načina.