VEŽBA 1 - Okruženje Microsoft Visual Studio 2013

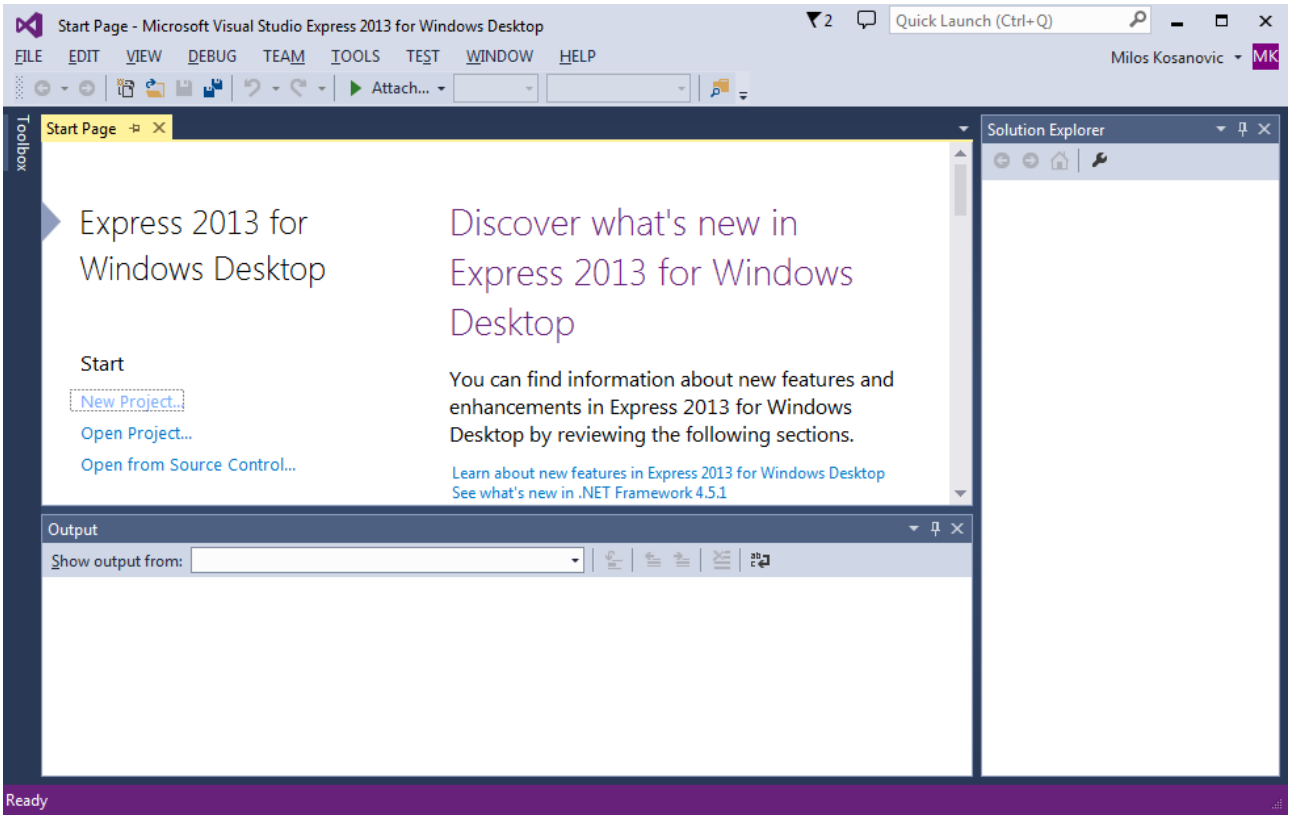
Microsoft Visual Studio 2013 predstavlja integrisano okruženje za razvoj softvera. Sastoji se od sledećih celina:

- *Visual C++*,
- *Visual Basic*,
- *Visual C#*
- *MSDN Library*.
- *Druge*…

Visual studio C++ je okruženje koje omogućava razvoj i pisanje programa u programskom jeziku C++, a samim tim i u programskom jeziku C.

Pokretanje Visual Studio 2013 okruženja

Okruženje *Visual Studio 2013* se može otvoriti klikom na taster *Start* i izborom stavke *Microsoft Visual Studio 2013* iz liste programa. Nakon startovanja otvoriće se osnovi prozor okruženja *Visual Studio 2013* koji je prikazan na slici 1.

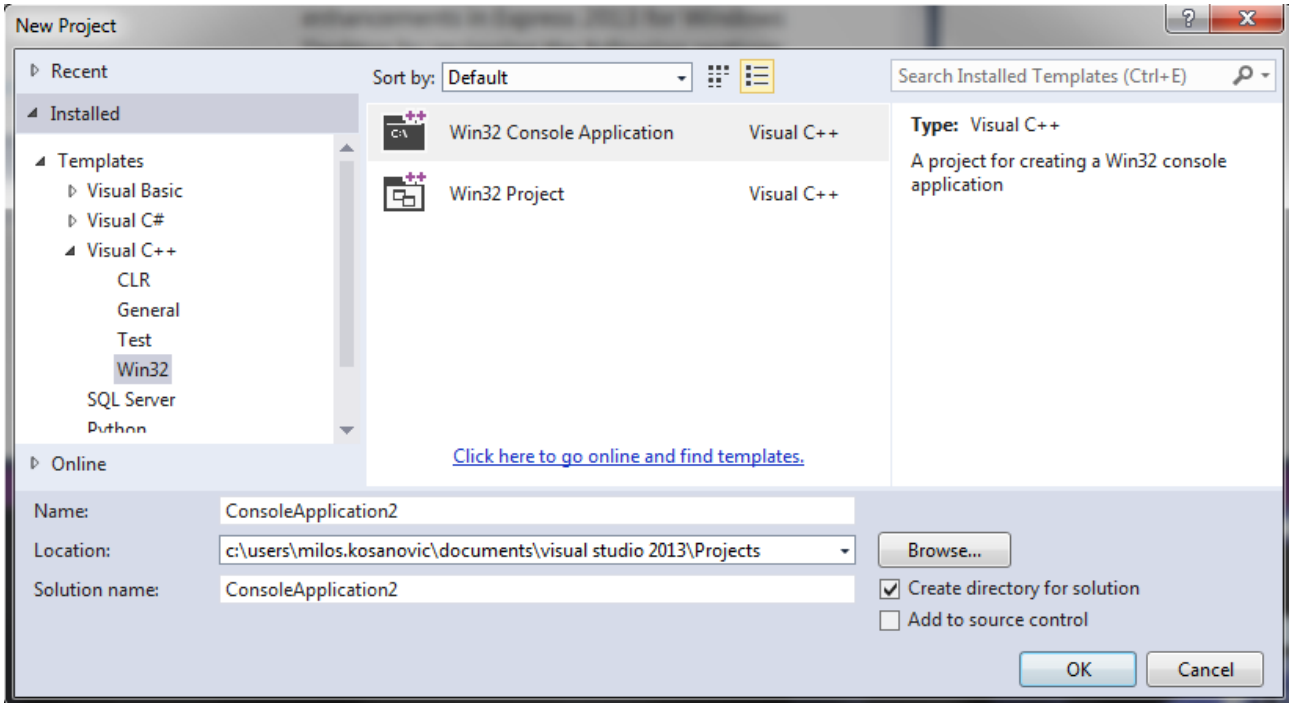


Slika 1. Izgled osnovnog prozora po otvaranju *Visual C++*-a

Kreiranje projekta za konzolnu aplikaciju

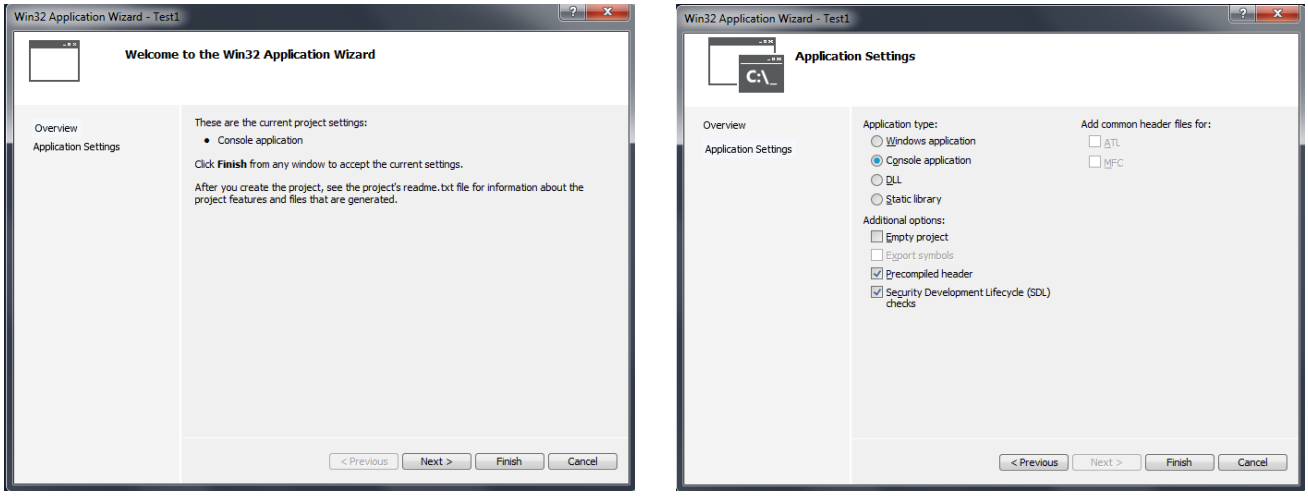
Da bi se napisao, kompilirao, izvršio i eventualno debagirao C program, u *Visual Studiju* je potrebno kreirati projekat koji sadrži odgovarajuće datoteke sa izvornim kodom programa. Za potrebe pisanja programa u C-u biće korišćen najjednostavniji tip projekta koji obezbeđuje pravljenje konzolnih aplikacija. Konzolne aplikacije su programi koji se izvršavaju u komandnom (DOS) prozoru i koriste standardni ulaz i izlaz.

Kreiranje projekta se vrši startovanjem stavke iz menija *File/New*, izborom kartice *Project* i stavke *Visual C++ i zatim Win32 Console Application* (slika 2).



Slika 2. Izgled prozora za kreiranje novog projekta

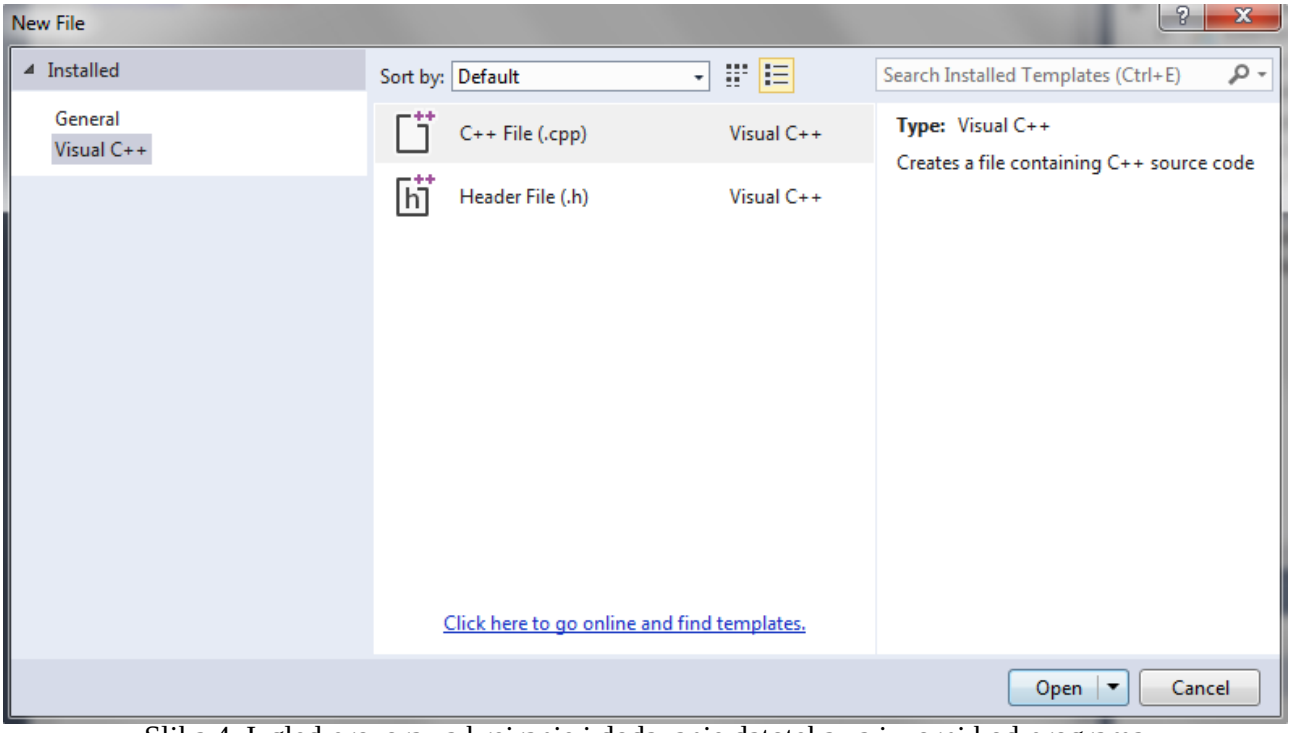
Pri kreiranju projekta potrebno je uneti naziv projekta u polje *Project name*, i eventualno promeniti direktorijum na disku gde će projekat biti kreiran (polje *Location*). Po unosu naziva projekta aktiviraće se taster OK. Klikom na ovaj taster pojaviće se prozor za izbor tipa konzolnog projekata koji se želi kreirati (slika 4). U ovom slučaju ne treba ništa menjati jer je već odabrana opcija za kreiranje praznog projekta (*An empty project*). Da bi se kreirao projekat potrebno je kliknuti na taster *Finish*. Ukoliko želite možete kliknuti i na dugme *next*. U tom slučaju će Vam biti prikazan sledeći prozor sa dodatnim opcijama.



Slika 3. Izgled prozora za izbor tipa konzolnog projekta koji će biti kreiran

Kreiranje i dodavanje datoteka za izvorni kod programa

Nakon kreiranja praznog projekta za konzolnu aplikaciju potrebno je kreirati i dodati u projekat datoteke koje će se koristiti za unos izvornog koda programa. U VS2013 obično je za Vas već kreiran fajl koji se zove isto kao vaš projekat. Ukoliko nije, potrebno je kreirati i dodati datoteku na sledeći način. Pozivate stavku iz menija *File/New File* i stavke Visual *C++*. Zatim možete izabrati između *C++ File* ili *Header File* (slika 4). Potrebno je uneti željeni naziv datoteke u polje *File name* i kliknuti na taster *OK*.



Slika 4. Izgled prozora za kreiranje i dodavanje datoteka za izvorni kod programa

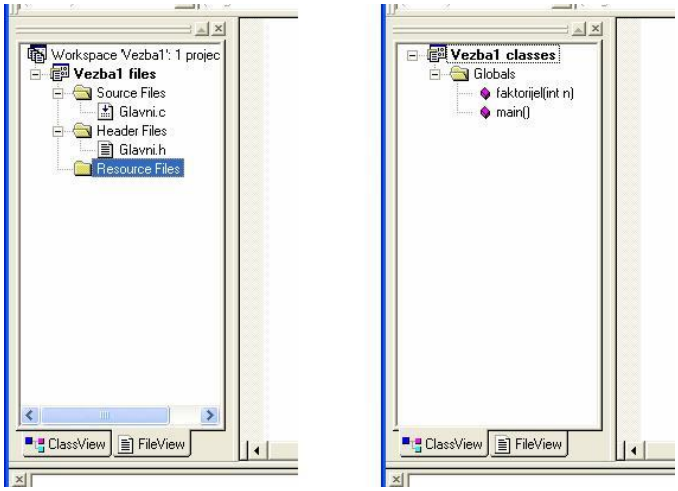
Po kreiranju i dodavanju nove datoteke u projekat potrebno je uneti odgovarajući sadržaj (ukucati program). Za to se koristi centralni deo glavnog prozora (slika 5). *Visual Studio* poseduje neke napredne opcije za prikaz izvornog koda programa kao što su bojenje ključnih reči i komentara.

```
// Glavni.c - datoteka sa izvornim kodom (definicije funkcija)

#include <stdio.h>
#include "Glavni.h"

main()
{
    int n, fn;
    printf("Unesi broj: ");
    scanf("%d", &n);
    fn = faktorijel(n);
    printf("Faktorijel broja %d iznosi %d\n", n, fn);
}

int faktorijel(int n)
{
    int fn = 1;
    int i = n;
    while(i > 1) {
        fn = fn * i;
    }
    return fn;
}
```

Slika 5. Deo za unos i prikaz sadržaja datoteka sa izvornim kodom

Prikaz i otvaranje datoteka i funkcija koje projekat sadrži

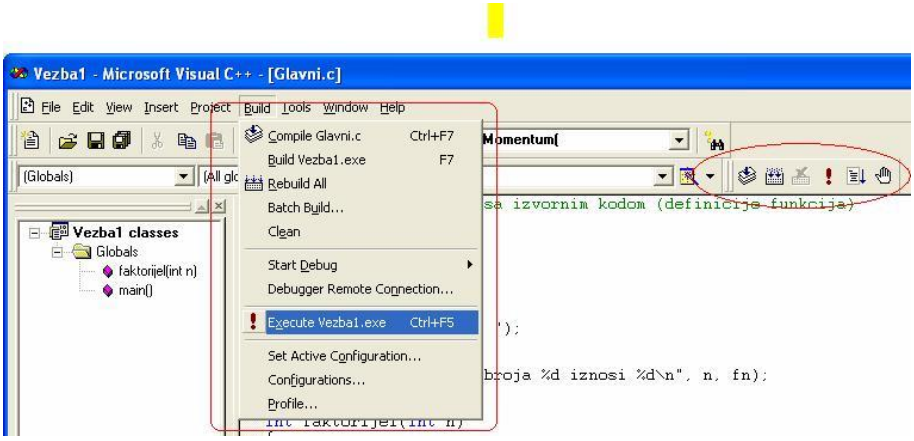
Za prikaz datoteka koje su uključene u projekat koristi se panel sa leve strane osnovnog prozora *Visual C++*-a. Ovaj panel sadrži dva kartice *ClassView* i *Solution Explorer*. Kartica *Solution Explorer* omogućava pregled i otvaranje datoteka koje su dodate u projekat. Datoteke su grupisane po tipu koji je određen ekstenzijama (*Source Files* - *.cpp datoteke, *Header Files* - *.h datoteke). Duplim klikom na naziv datoteke, sadržaj odgovarajuće datoteke će biti prikazan u centralnom delu prozora *Visual C++*-a. Naziv datoteke koja je trenutno otvorena prikazan je u naslovnoj liniji glavnog prozora.
Kartica *ClassView* je izvorno namenjena za prikaz informacija o klasama definisanim u C++ programu. Kako programski jezik C ne podržava rad sa klasama u prikazu vezanom za ovu karticu pobrojane su samo funkcije definisane u datotekama koje projekat sadrži. Duplim klikom na naziv funkcije otvara se odgovarajuća datoteka i postavlja se na početak odgovarajuće funkcije. Na slici 6 ilustrovani su *FileView* i *ClassView* prikazi.



Slika 6. Prikaz datoteka i funkcija koje projekat sadrži

Kompiliranje programa i izveštaj o greškama

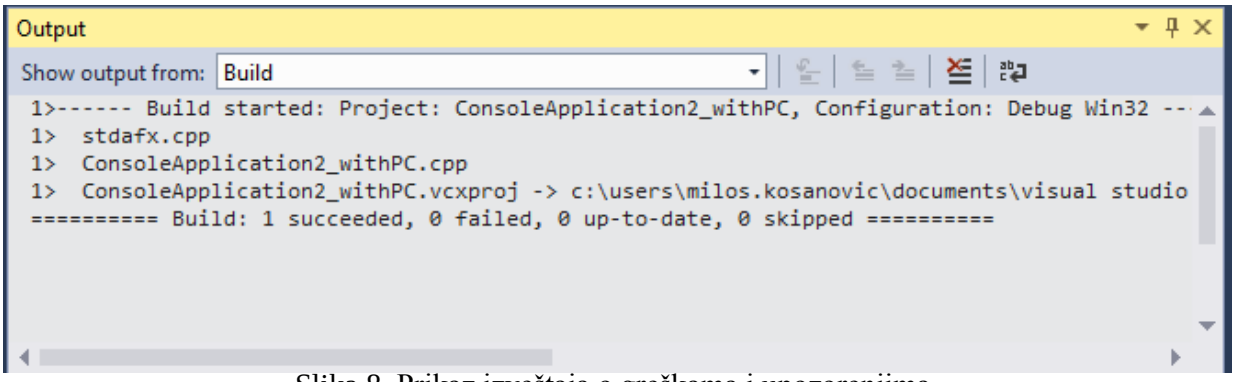
Nakon unosa koda programa u odgovarajuće datoteke moguće je izvršiti kompiliranje, povezivanje i izvršavanje programa. U ovu svrhu se koriste stavke iz menija *Build* ili iz odgovarajuće prečice sa alatima (vidi sliku 7).



Slika 7. Alati za kompiliranje programa

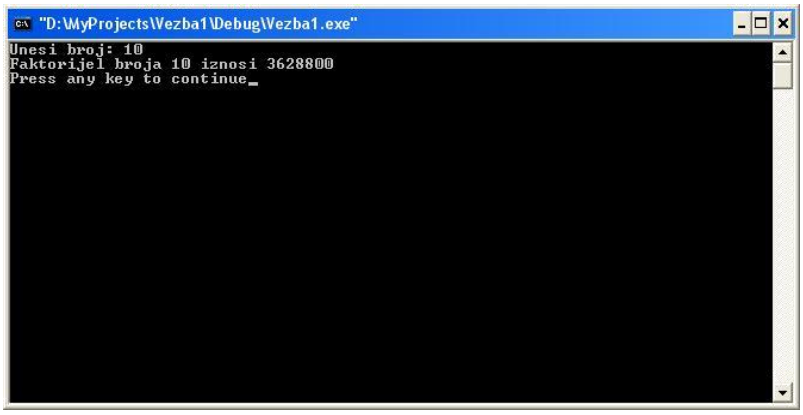
Najjednostavniji način za iniciranje procesa kompiliranja svih datoteka sa izvornim kodom, kreiranje izvršne (EXE) datoteke programa i njeno startovanje je korišćenjem stavke *Build/Execute...*, kombinacije tastera ***Ctrl+F5*** sa tastature ili odgovarajuće prečice (ikonica "▶"). Ukoliko program

nije prethodno preveden ili je izmenjen od vremena kada je poslednji put preveden, pojaviće se prozor sa pitanjem da li treba izvršiti prevođenje (kompajliranje) programa. U oba slučaja potrebno je kliknuti na dugme *Yes* nakon čega će uslediti kompiliranje i povezivanje (linkovanje) izvršnog programa. Ukoliko je program uspešno preveden automatski će se izvršiti njegovo startovanje, u suprotnom u donjem delu osnovnog prozora će biti prikazan izveštaj o otkrivenim greškama (slika 8). Za svaku otkrivenu grešku ili upozorenje pored koda i opisa, navedena je datoteka i linija koda u kojoj je greška nađena. na liniju sa opisom greške u ovom izveštaju automatski se otvara odgovarajuća datoteka i vrši postavljanje kursora na problematičnu liniju.



Slika 8. Prikaz izveštaja o greškama i upozorenjima

Nakon ispravljanja grešaka potrebno je ponovo izvršiti startovanje procesa prevođenja i izvršenja programa. Na slici 9 prikazan je prozor u kome se izvršava program. Nakon završetka izvršenja programa biće ispisan tekst "*Press any key to continue*". Pritiskom na neki taster prozor u kome se program izvršavao će biti zatvoren.



Slika 9. Izgled prozora u kome se izvršava program

Prekompajlirani hederi

Prekompajlirani hederi predstavljaju C i C++ heder fajlove koji se pre kompajliranja pripreme i prebace u *intermediate* formu koju kompajler može brže da kompajlira. Ova forma se naziva još i prekompajlirani heder. Korišćenje ovakvih hedera može skratiti vreme kompajliranja, naročito kada se primeni na velike heder fajlove. Problem kompajliranja velikog broja uključenih fajlova postoji i kod drugih programskih jezika i svaki od njih ga rešava na neki svoj način. Demonstrirajmo primerom kako se to rešava u C++ jeziku. Uzmimo za primer projekat koji ima dodata dva fajla: heder fajl header.h i fajl sa izvornim kodom source.cpp:

```
//header.hpp
...
```

```
//source.cpp
#include "header.hpp"
```

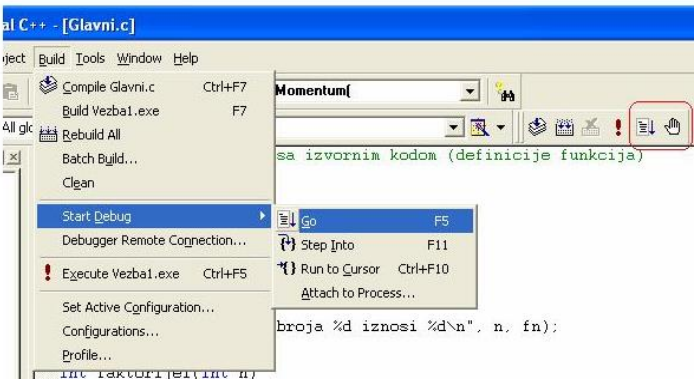
```
. . .
```

U slučaju da je opcija za prekompajlirane hedere uključena, kompajler će generisati prekompajlirani heder fajl *header.pch*. Kada sledeći put budete kompajlirali fajlove, ukoliko se vreme zadnje promene fajla header.h nije promenilo, računar će preskočiti kompajliranje i iskoristiti već postojeći header.pch fajl. U suprotnom novi fajl heaader.pch će biti kreiran, a stari obrisan.

Prilikom korišćenja Visual Sudia 2013 postoji fajl stdafx.h koji predstavlja prekompajlirani heder za sve one biblioteke koje želite da uključite u vaš program, a koje se retko menjaju. To znači da sve *include* direktive treba pisati u ovom fajlu.

Debagiranje programa

Pored sintaksnih grešaka koje se otkrivaju u toku prevođenja, program može da sadrži i logičke greške koje sprečavaju da se izvršava na način koji je programer predvideo. Ovakve greške se nazivaju bug-ovi, a proces njihovog uklanjanja debugging. Proces debagiranja podrazumeva postavljanje prekidnih tačaka (*breakpoints*) u kojima će se izvršenje programa privremeno prekinuti u cilju očitavanja vrednosti pojedinih promenljivih i sl. Postavljanje (ili uklanjanje) prekidnih tačaka vrši se postavljanjem kursora na odgovarajuću liniju i klikom na dugme u obliku šake (vidi sliku 11). Druga varijanta za dodavanje (ili uklanjanje) prekidne tačke je desni klik na odgovarajuću programsku liniju i izbor stavke Insert/Remove Brakepoint iz padajućeg menija. Za startovanje izvršenja programa u ovom modu koristi se stavka iz menija Build/Start Debug/Go, taster F5 ili odgovarajuća prečica iz linije sa alatima (vidi sliku 10).

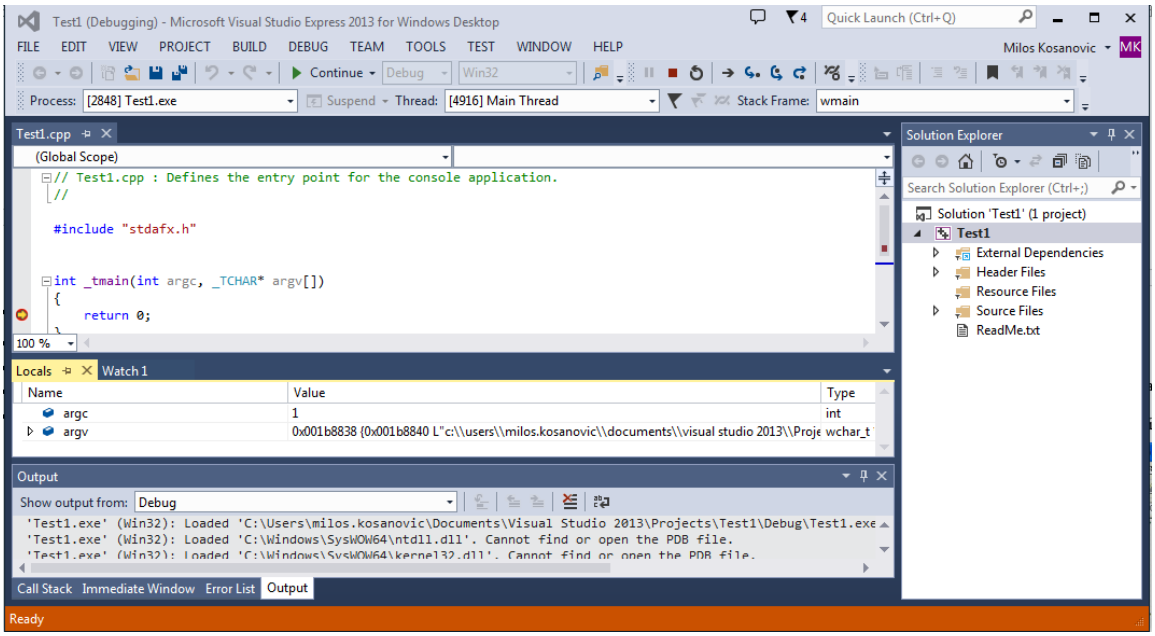


Slika 10. Alati za debagiranje programa

Pri debagiranju, program će se izvršavati normalno sve dok se ne dođe do neke od zadatih prekidnih tačaka. U tom trenutku se zaustavlja izvršenje i postaje aktivno okruženje *Visual C++*-a prekonfigurisano za debagiranje. Na slici 11 prikazan je izgled okruženja u modu za debagiranje.

Za kontrolu izvršenja koriste se komande menija *Debug*. Značenja pojedinih komandi su sledeća:

- *Go* (taster *F5*) – nastavlja izvršenje do naredne prekidne tačke.
- *Restart* (kombinacija tastera *Ctrl+Shift+F5*) – resetuje izvršenje programa u modu za debagiranje.
- *Stop Debugging* (kombinacija tastera *Shift+F5*) – prekida debagiranje programa.
- *Step Into* (taster F11) – ulazi u funkciju koja čeka na izvršenje. Ukoliko je u pitanju neka druga naredba izvršava je i prelazi na narednu liniju u kodu.
- *Step Over* (taster F10) – izvršava funkciju ili naredbu i prelazi na narednu liniju u kodu.
- *Step Out* (kombinacija tastera Shift+F11) – izvršava tekuću funkciju do kraja i izlazi na nivo iz koga je pozvana.



Slika 11. Okruženje za debagiranje programa

- Run to Cursor (kombinacija tastera Ctrl+F10) – izvršava program do linije koda na koju ukazuje položaj kursora.

Po zaustavljanju izvršenja moguće je očitati vrednosti pojedinih promenljivih i prikaz konteksta (funkcija) iz kojeg se došlo do prekidne tačke. Očitavanje vrednosti promenljivih se može izvršiti na nekoliko načina:

1. Zadržati pokazivač miša iznad naziva promenljive negde u kodu, što će rezultovati ispisivanjem vrednosti pored pokazivača.
2. Dodavanjem naziva promenljive čije se praćenje želi u panelu *Watch* (desno ispod koda).
3. Direktno, ukoliko je promenljiva automatski izlistana u spisku promenljivih koje se trenutno koriste (panel levo ispod koda).

Samostalni rad studenta

Zadatak 1: Napisati i pokrenuti program.

```
void main()
{
	printf("Moj prvi program u C jeziku\n");
}
```

Zadatak 2: Napisati i pokrenuti program.

```
void main()
{
	int  razlika;
	razlika = 100 -90;
	printf("Razlika 100 -90= %d\ n", razlika);
}
```

Zadatak 3:Napisati i izvršiti program. Probajte da debagirate program. Kolika je vrednost promenljive i nakon pre ulaska u petlju (linija 5) i nakon izlaska iz petlje (linija 10)?

```
void main()
{
	int zbir = 0;
	int i;
	for (i=0; i<10; i++)
```

```
	{
		zbir = zbir + i;
		printf("i=%d  zbir=%d\n", i, zbir);
	}
	printf("\nzbir=%d\n", zbir);
}
```

Zadatak 4:Nacrtati algoritam za sledeće zadatke:

A. Sastaviti dijagram toka i napisati program kojim se izračunava vrednost funkcije:

$$y = \begin{cases} x, & x < 2 \\ 2, & 2 \le x < 3 \\ x - 1, & x \ge 3 \end{cases}.$$

B. Nacrtati algoritam koji štampa vaše ime 1000 puta.
C. Nacrtati algoritam koji računa N5
D. Nacrtati algoritam koji računa Nx

Pitanja za odbranu vežbe

1. Šta je Visual studio 2013?
2. Kako se kreira novi projekat? Koje vrste projekta postoje?
3. Šta je konzola?
4. Kako se dodaje nova biblioteka u projekat?
5. Kako se dodaje novi fajl u projekat?
6. Šta je *ClassView?*
7. Šta je Solution Explorer? File view?
8. Koji sve fajlovi postoje ili se kreiraju u toku bildovanja projekta?
9. Šta je kompajliranje programa?
10. Šta je linkovanje programa?
11. Šta je debagiranje programa?
12. Kako se dodaje breakpoint?
13. Šta su to prekompajlirani hederi?
14. Čemu služi fajl stdafx.h?
15. Čemu služi fajl stdio.h?
16. Nakon zaustavljanja programa u debug modu, na koji način se sve može očitati vrednost neke promenljive?
17. U čemu je razlika između naredbi Step Into i Step Over u debug modu?