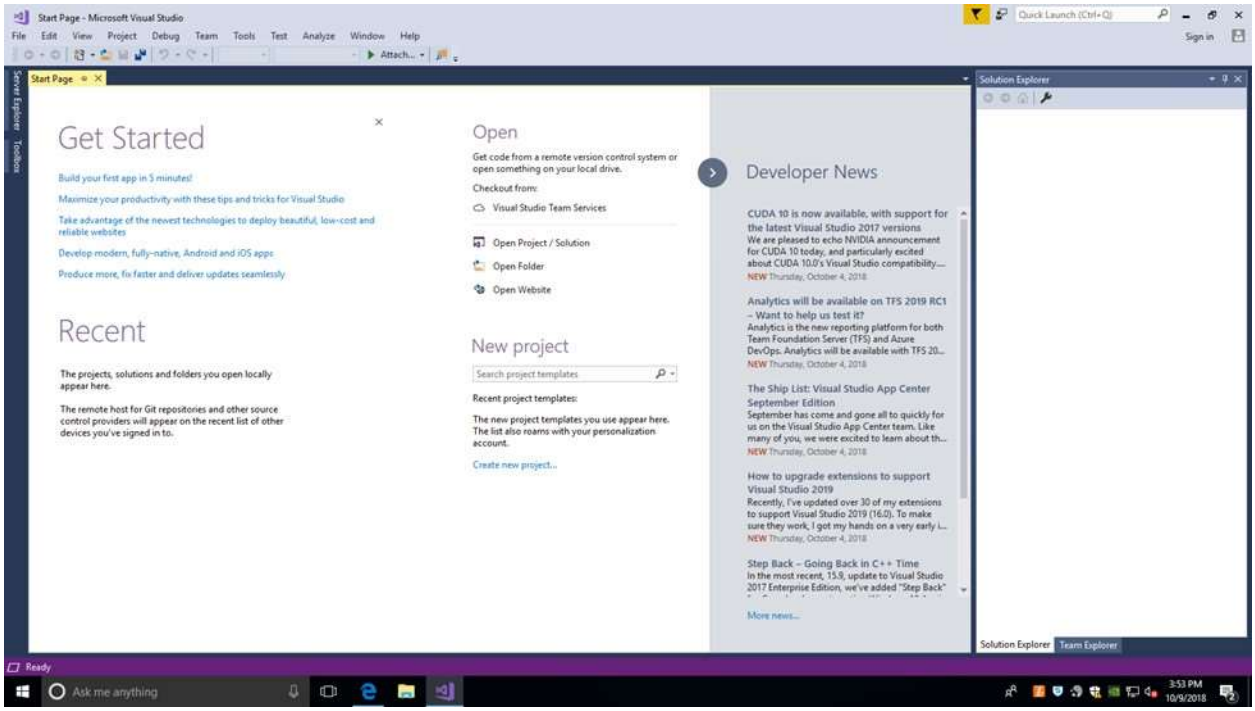
Vežba 1 - Okruženje Microsoft Visual Studio

Microsoft Visual Studio 2017 predstavlja integrisano okruženje za razvoj programske podrške. U okviru istog nalazi se *Visual C++* okruženje (između ostalih okruženja koja nisu predmet izučavanja u okviru ovog kursa) koje omogućava razvoj i pisanje programa u programskom jeziku C++, a samim tim i u programskom jeziku C.

1. Pokretanje Visual C++ okruženja

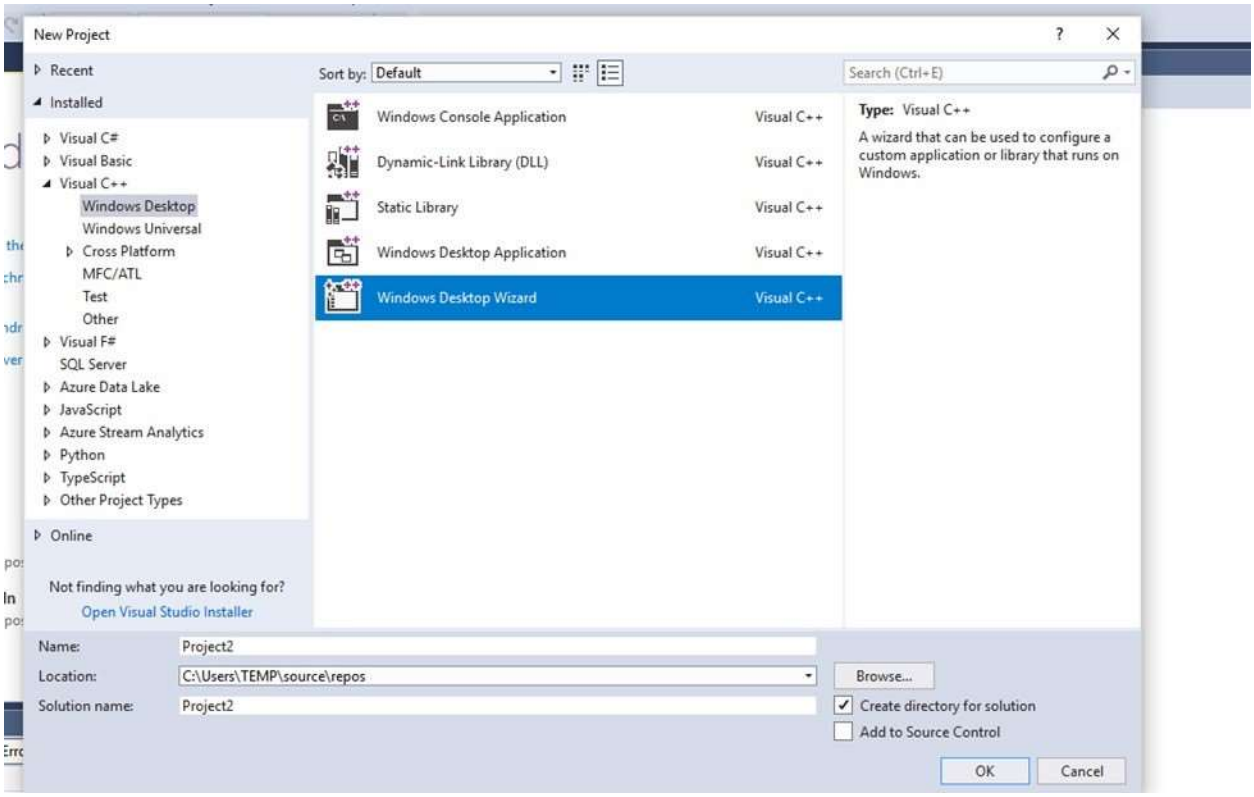
Okruženje *Visual C++* se može otvoriti klikom na taster *Start* i izborom stavke *Microsoft Visual Studio 2017/Microsoft Visual Studio 2017* iz liste programa (grupa *All Programs*). Nakon startovanja otvoriće se osnovni prozor *Microsoft Visual Studio 2017* okruženja i ponuđena opcija za izbor radnog okruženja. Potrebno je izabrati opciju radnog okruženja *Visual C++*. Na slici 1. prikazan je osnovni prozor radnog okruženja.



Slika 1. Izgled osnovnog prozora *Microsoft Visual Studio 2017*

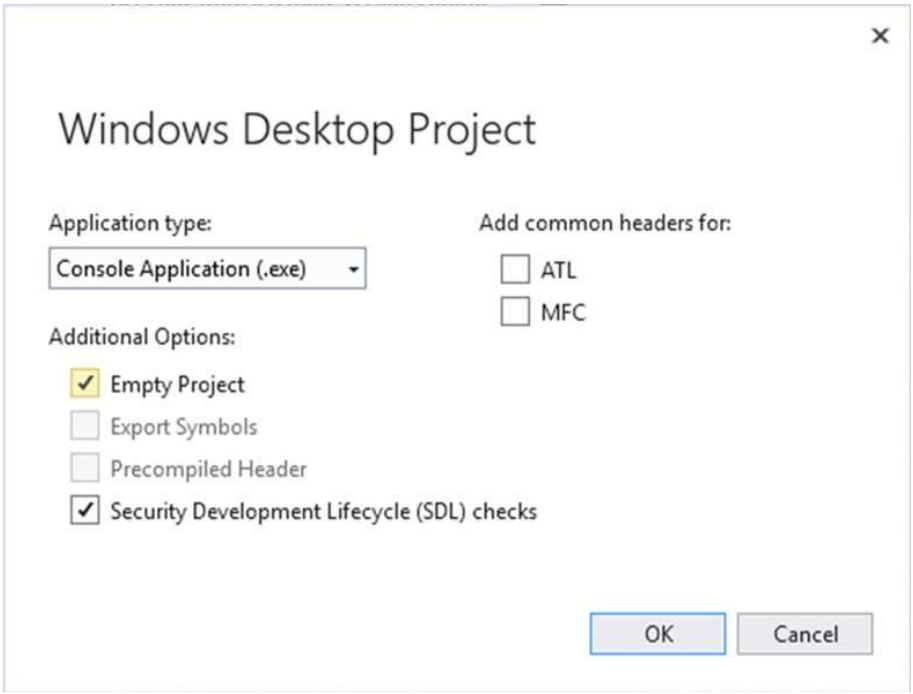
2. Kreiranje projekta za konzolnu aplikaciju

Da bi se napisao, preveo i izvršio C/C++ program, u *Visual C++*-u 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 *Windows Desktop/ Windows Desktop Wizard* (slika 2).



Slika 2. Izgled prozora za kreiranje novog projekta

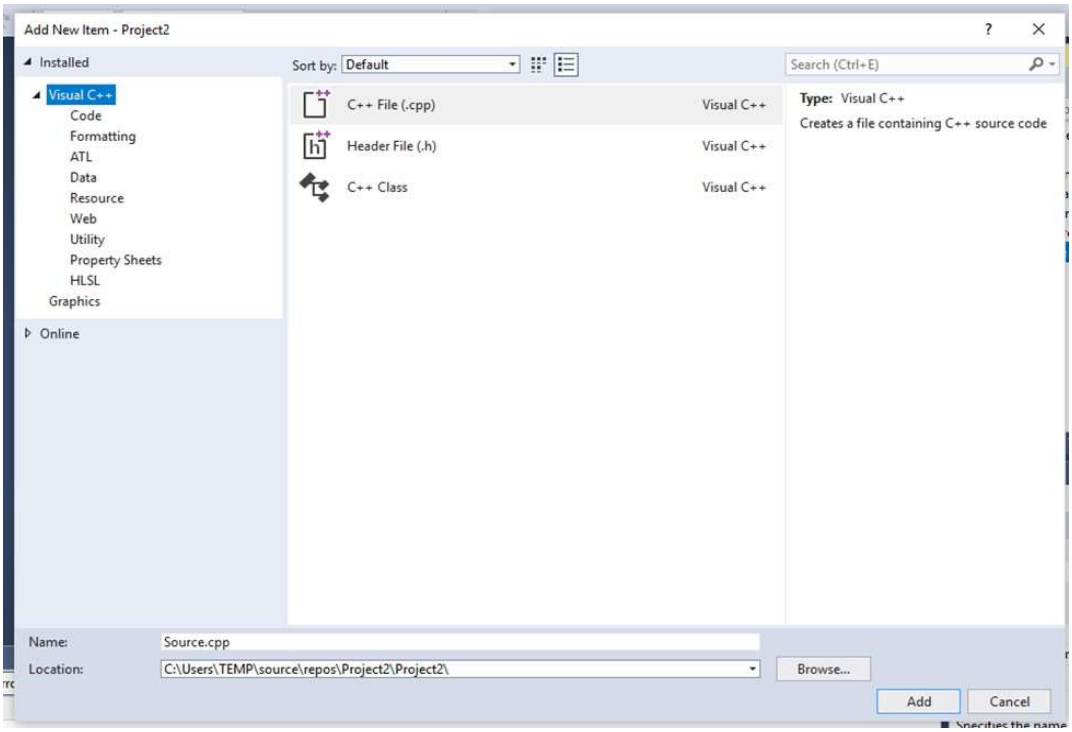
Pri kreiranju projekta potrebno je uneti naziv projekta u polje *Name*, uneti naziv rešenja u polje *Solution name* (ono može biti isto kao i ime projekta unutar njega) i eventualno promeniti direktorijum na disku gde će projekat biti kreiran (polje *Location*). Po unosu naziva projekta aktiviraće se dugme *OK*. Klikom na ovo dugme pojaviće se prozor za izbor tipa desktop projekta koji se želi kreirati (slika 3). U padajućoj listi treba da je izabran tip *Console Application* i dodatno odabrati opciju za kreiranje praznog projekta (*Empty project*). Da bi se kreirao projekat potrebno je kliknuti na dugme *OK*.



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

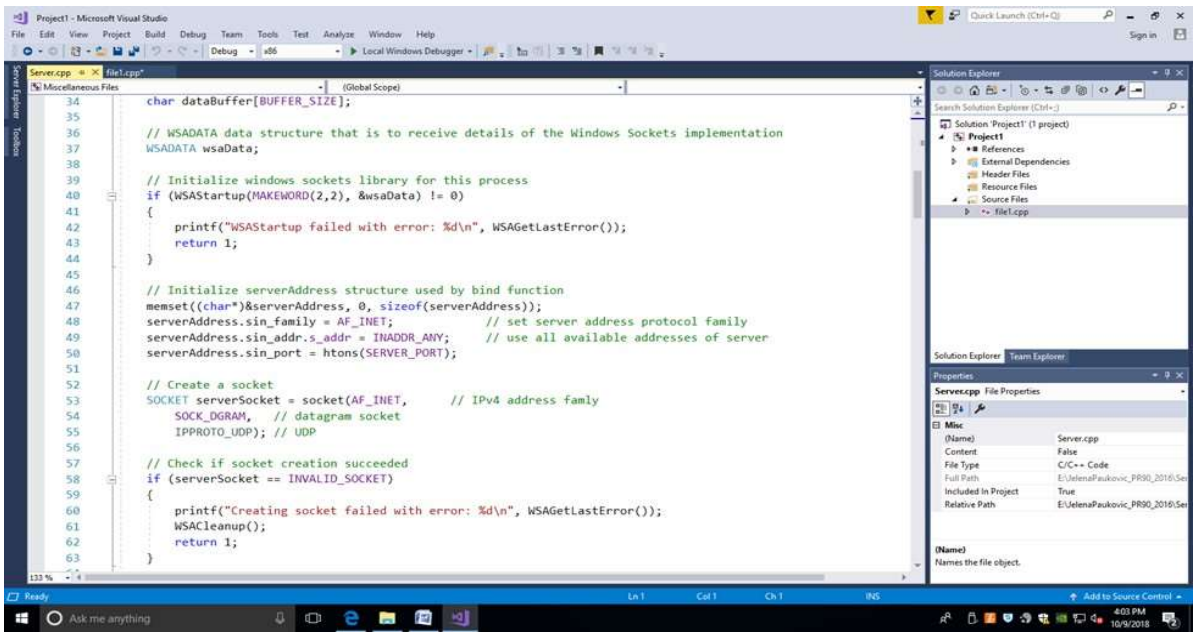
3. Kreiranje i dodavanje datoteka za izvorni kod programa

Nakon kreiranja praznog projekta za konzolnu aplikaciju potrebno je kreirati i dodati u projekat datoteke koje će se koristiti za unos izvornog koda programa. Kreiranje i dodavanje datoteke se vrši ponovnim startovanjem stavke iz menija *File/New*, izborom kartice *File* i stavke *C++ Source File* ili *C/C++ Header File* (slika 4). Potrebno je uneti željeni naziv datoteke u polje *Name* i kliknuti na dugme *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 (program). Za to se koristi centralni deo glavnog prozora (slika 5). Visual C++ poseduje neke napredne opcije za prikaz izvornog koda programa kao što su boja ključnih reči i komentara.

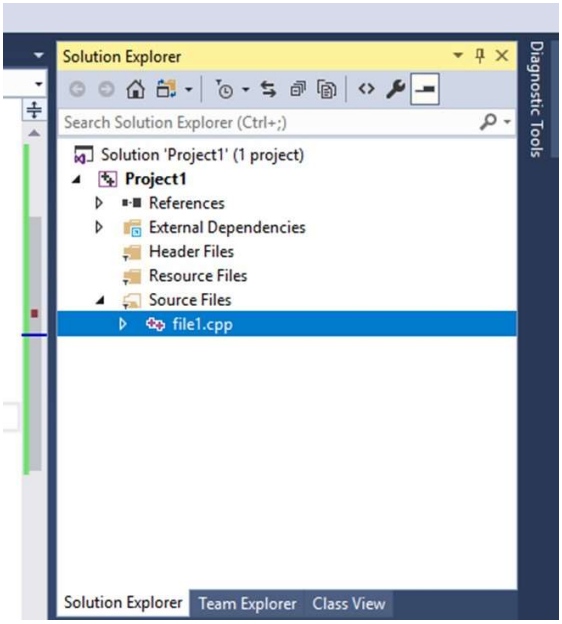


Slika 5. Deo za unos i prikaz sadržaja datoteka sa izvornim kodom

4. 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 dve kartice *Solution Explorer* i *ClassView*. Kartica *Solution Explorer* omogućava pregled i otvaranje datoteka koje su dodate u projekat. Datoteke se grupisane po tipu koji je određen ekstenzijama (*Source Files* - *.c 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.

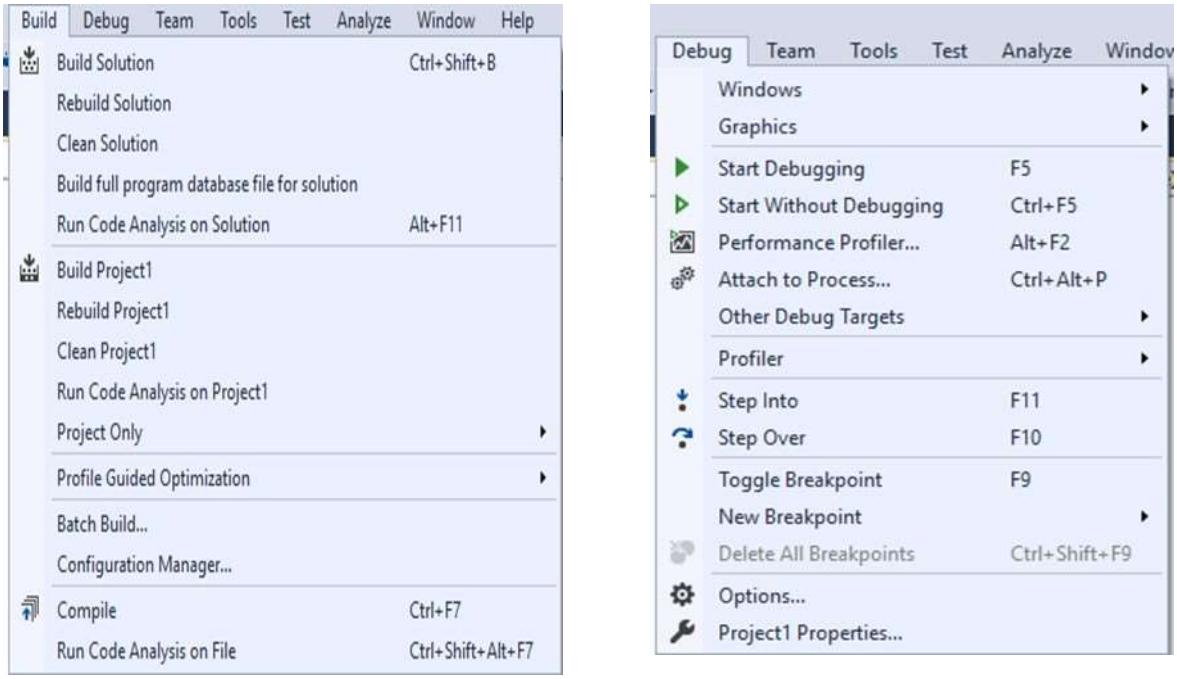
Kartica *ClassView* je izvorno namenjena za prikaz informacija o klasama definisanim u C++ programu. Kako programski jezik C ne podržava rad sa klasama u prikazu vezanom za ovu karticu pobrojane su samo funkcije definisane u datotekama koje projekat sadrži. Duplim klikom na naziv funkcije otvara se odgovarajuća datoteka i postavlja se na početak odgovarajuće funkcije. Na slici 6 ilustrovani su *Solution Explorer* i *ClassView* prikazi.



Slika 6. Prikaz datoteka koje projekat sadrži *(Solution Explorer)*

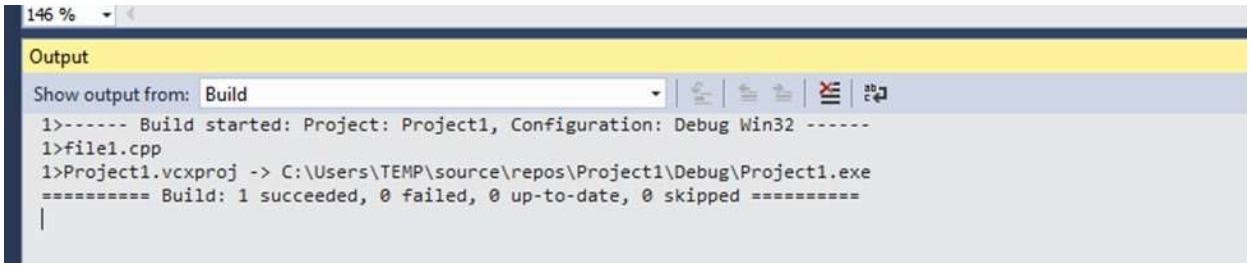
5. Prevođenje programa i izveštaj o greškama

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

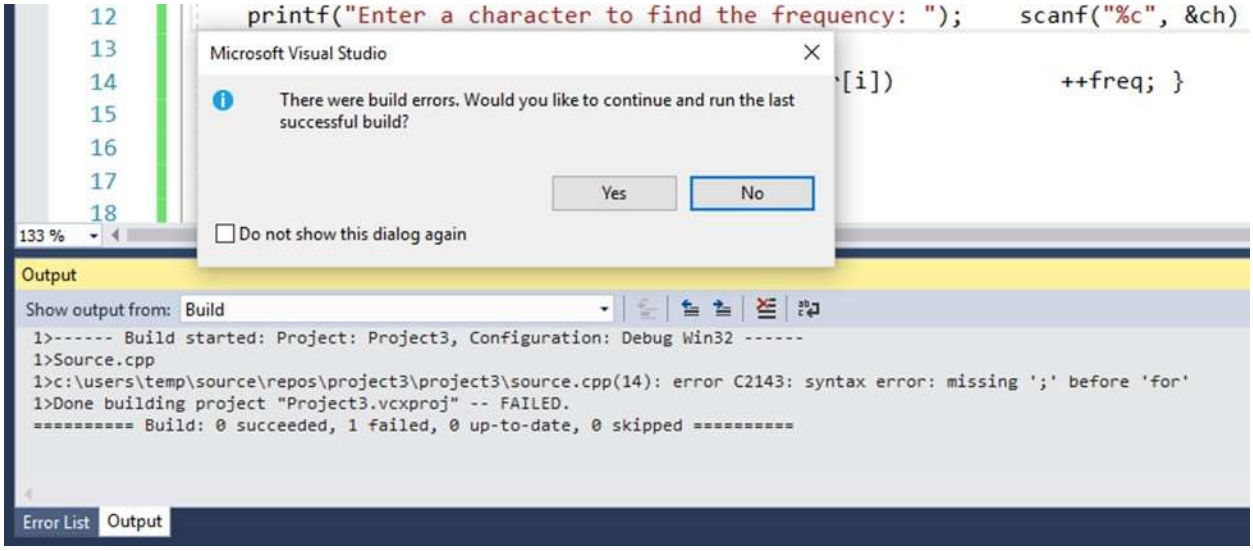


Slika 7. Alati za prevođenje programa (*Build* meni) i alati za kontrolu izvršenja (*Debug* meni)

Najjednostavniji način za iniciranje procesa prevođenja svih datoteka sa izvornim kodom, kreiranje izvršne (EXE) datoteke programa i njeno startovanje je korišćenjem stavke *Start Without Debugging*, kombinacije tastera *Ctrl+F5* sa tastature ili odgovarajuće prečice (ikonica *run*). 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 programa. U oba slučaja potrebno je kliknuti na dugme *Yes* nakon čega će uslediti prevođenje 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-a i 8-b). Za svaku otkrivenu grešku ili upozorenje pored koda i opisa, navedena je datoteka i linija koda u kojoj je greška nađena. Duplim klikom na liniju sa opisom greške u ovom izveštaju automatski se otvara odgovarajuća datoteka i vrši pozicioniranje kursora na problematičnu liniju.



Slika 8-a. Prikaz izveštaja o uspešnom prevođenju programa



Slika 8-b. Prikaz izveštaja o neuspešnom prevođenju programa i otkrivenim greškama

Nakon ispravljanja greški 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 u datom primeru biće ispisan tekst "Press any key to close this window". Pritiskom na neki taster, prozor u kome se program izvršavao će biti zatvoren.

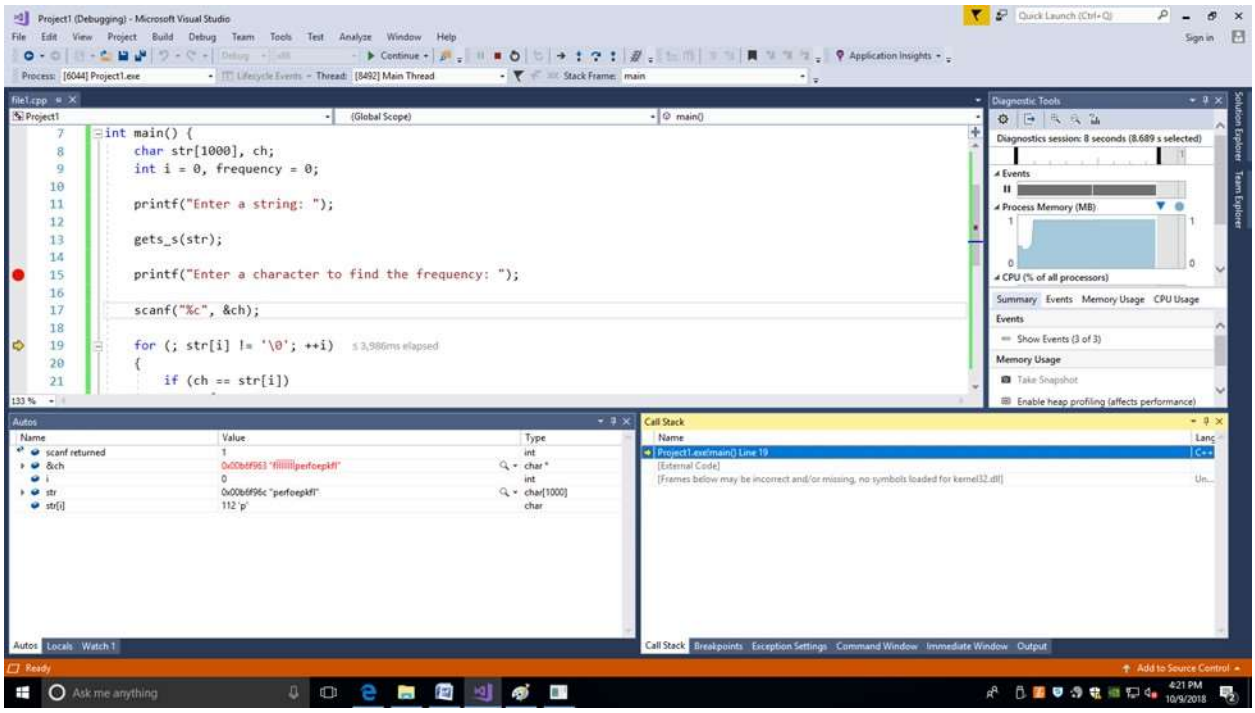
Slika 9. Izgled prozora u kome se izvršava program

6. Kontrolisano izvršenje programa (*debugging*)

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 otklanjanja grešaka podrazumeva postavljanje prekidnih tačaka u kojima će se izvršenje programa privremeno prekinuti u cilju očitavanja vrednosti pojedinih promenljivih i sl. Postavljanje (ili uklanjanje) prekidnih tačaka vrši se pozicioniranjem kursora na odgovarajuću liniju. 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 režimu rada koristi se stavka iz menija *Debug*, taster F5 ili odgovarajuća prečica iz linije sa alatima.

Tokom ovog procesa, 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 kontrolisano izvršenje programa. Na slici 10 prikazan je izgled okruženja za kontrolisano izvršenje programa.



Slika 10. Okruženje za kontrolisano izvršenje programa

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

- *Go* (taster *F5*) – nastavak izvršenja programa do naredne prekidne tačke.
- *Stop Debugging* (kombinacija tastera *Shift+F5*) – prekid režima rada kontrolisanog izvršenja programa.
- *Step Into* (taster F11) – ulazak u funkciju koja čeka na izvršenje. Ukoliko je u pitanju neka druga naredba ista se izvršava i prelazi se na narednu liniju u programskom kodu.

- *Step Over* (taster F10) – izvršavanje funkcije ili naredbe i prelazak na narednu liniju u programskom kodu.
- *Step Out* (kombinacija tastera Shift+F11) – izvršavanje tekuće funkcije do kraja i izlazak na nivo iz koga je ista pozvana.
- *Run to Cursor* (kombinacija tastera Ctrl+F10) – izvršavanje programa 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:

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

DODATAK - Nastanak programa

Nastanak programa može se podeliti na:

- Pisanje izvornog koda.
- Prevođenje izvornog koda.
- Povezivanje u izvršni kod.
- Proveravanje programa.

Izvorni kod

Kombinacijom naredbi programskog jezika nastaje izvorni programski kod (engl. *source code*). Izvorni kod je moguće pisati u bilo kom programu za uređivanje teksta (engl. *text editor*). Danas se uglavnom programi za pisanje izvornog koda objedinjuju u celinu sa programskim prevodiocem i povezivačem (integrisana razvojna okruženja, *IDE*). Izvorni kod programa sprema se u datoteku izvornog koda pod smislenim imenom i ekstenzijom (*.c).

Izvršni oblik

Programi se mogu izvršiti na računaru samo ako su u binarnom obliku. Takav se oblik programa naziva izvršni oblik (engl. *executable*). Izvorni se kod mora prevesti u izvršni. Prevodi se pomoću programa koji se nazivaju programski prevodilac (engl. *compiler*) i povezivač (engl. *linker*). Programski prevodilac prevodi izvorni kod iz višeg programskog jezika u mašinski oblik i proverava sintaksu napisanog izvornog koda. Ako pronađe greške (engl. *compile-time error*), ispisuje poruke i upozorenja o njima. Otkrivene greške treba ispraviti pa ponovo pokrenuti program za prevođenje.

Prevođenjem nastaje datoteka objektnog koda (engl. *object code*), sa ekstenzijom *.obj. Objektni kod nije izvršni program i ne može se direktno izvršiti na računaru. Objektni kod je međukorak do izvršnog koda i uz ostalo omogućava uključivanje gotovih delova programa iz drugih datoteka.

Biblioteke

Datoteke koje sadrže gotove delove programa nazivaju se biblioteke (engl. *libraries*). Takvi se gotovi delovi programa mogu koristiti u drugim programima. Korišćenjem gotovih biblioteka više nije potrebno iznova zapisivati funkcije koje se često koriste. Takve se funkcije u program uključuju iz postojećih biblioteka.

Povezivač

Program koji povezuje objektnu datoteku s bibliotekama i drugim potrebnim datotekama naziva se povezivač (engl. *linker*). Ako se pri povezivanju pojavi greška (engl. *link-time error*), biće ispisana poruka o tome. Grešku je potrebno ispraviti pa ponovno pokrenuti prevođenje i povezivanje. Rezultat uspešnog povezivanja je izvršna datoteka (*.exe). U principu, izvršnoj datoteci nisu potrebni nikakvi dodaci pa se može izvršavati i bez izvornog programa, objektnih datoteka, prevodioca, povezivača itd.

Izvršna datoteka

Izvršna datoteka je oblik programa i može se direktno izvršiti na računaru za koji je prevedena. Na primer IBM i Apple računari su međusobno nekompatibilni, pa se izvršni program preveden za IBM PC ne može izvršiti na Macintosh računaru i obrnuto.

Projekat

Da bi se stvorila izvršna datoteka programa potrebno je pokrenuti nekoliko programa (tekst editor, prevodilac, povezivač). Kao posledica toga, nastaje više datoteka koje su međusobno vezane. Korisniku koji zadatak rešava određenim programom nepraktično je pamtiti koje programe i kojim redosledom treba pokrenuti, kao i koje su sve datoteke potrebne za stvaranje izvršne datoteke. Stoga se korisniku posao olakšava pomoću takozvanog projekta (engl.*project*). Projekat je datoteka u kojoj su zapisane sve potrebne informacije o prevodiocu, povezivaču, datotekama, bibliotekama i ostalom potrebnom za izradu izvršne datoteke. Projekat dakle "brine" o svemu što je potrebno učiniti da bi od izvornog koda nastala datotekaizvršnog koda.

Greške

Tokom rada mogu se javiti tri vrste grešaka:

- Sintaksne greške (otkriva ih programski prevodilac).
- Greške povezivanja (otkriva ih program povezivač).
- Logičke greške (mora ih pronaći programer-korisnik).

Osnovne funkcije u Winsock biblioteci

1. Teorijske osnove

Windows utičnice

Za svaki od transportnih protokola (TCP i UDP) možemo definisati utičnicu (engl. *socket*), odnosno par - IP adresa/broj porta koji jedinstveno identifikuje softverski proces (mrežnu aplikaciju) i host na kome je proces pokrenut. Ovako definisane utičnice su krajnje tačke komunikacije na transportnom sloju. U okviru ovog kursa, svaka serverska aplikacija će po pokretanju kreirati TCP ili UDP utičnicu (u zavisnosti od tipa serverske aplikacije).

Kreiranje utičnice znači da na datoj IP adresi i na portu serverski aplikacijski servis očekuje poruke koje će stići sa utičnice (par IP adresa klijenta/port > 1023) koji je otvorila klijentska aplikacija. Na primer: kada se na serveru pokrene softverski proces Web servera, on kreira - otvori utičnicu na IP adresi servera na dobro poznatom portu 80. Kada klijent na svojoj radnoj stanici pokrene Web pretraživač i u adresnom polju pretraživača unese adresu servera, Web pretraživač otvori utičnicu na IP adresi klijenta na portu većem od 1023, na primer 39536. Krajnje tačke komunikacije na transportnom nivou su dve utičnice.

Jedna utičnica se može koristiti za više istovremenih komunikacionih veza. Drugim rečima, dve ili više veza mogu završavati na istoj utičnici. Veze se raspoznaju prema identifikatorima utičnica na oba kraja veze (socket1, socket2).

Usluge koje TCP i UDP nude aplikativnom sloju se razlikuju, pa su i utičnice ova dva protokola razlikuju. TCP nudi pouzdanu uslugu sa uspostavljanjem veze, pa TCP utičnice imaju svoje stanje, u smislu da li je TCP konekcija ostvarena, ili je u toku ostvarivanje odnosno prekid konekcije. Osim toga ukoliko je konekcija ostvarena poznata je i druga utičnica, na udaljenom hostu koji je drugi kraj veze. UDP servis ne poznaje uspostavu konekcije, pa UDP utičnice nemaju posebno stanje.

2. Winsock biblioteka

Inicijalizacija Winsock biblioteke

Svaka *Winsock* aplikacija mora na početku da učita odgovarajuću verziju *Winsock DLL*. Ukoliko se neuspešno učita *Winsock* biblioteka pri pozivu neke *Winsock* funkcije vratiće se SOCKET_ERROR sa kodom greške WSANOTINITIALISED.
Učitavanje *Winsock* biblioteke postiže se pozivom *WSAStartup* funkcije, koja je deklarisana na sledeći način:

```
int WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData);
```

Funkcija:	Opis:
`WSAStartup`	Učitavanje i inicijalizacija *Winsock* biblioteke
Parametri:	Opis:
`wVersionRequested [in]`	Koristi se za specifikaciju verzije *Winsock* biblioteke koja se učitava. Bajt višeg reda određuje podverziju biblioteke, dok bajt nižeg reda određuje verziju. Makro *MAKEWORD (X,Y)* se koristi za postavljanje ispravne vrednosti parametra `wVersionRequested`. *X* predstavlja viši bajt, a *Y* niži bajt.
`lpWSAData [out]`	Pokazivač na strukturu `LPWSADATA` u koju se smeštaju informacije vezane za verziju biblioteke koja se učitava
Povratna vrednost	Opis
`int`	Ako se funkcija uspešno izvrši vraća vrednost 0. U suprotnom, vraća se kod greške.

Primer:

```
WSADATA wsaData;
// Initialize windows sockets library for this process
if (WSAStartup(MAKEWORD(2,2), &wsaData) != 0)
{
	printf("WSAStartup failed with error: %d\n", WSAGetLastError());
	return 1;
}
```

Završetak rada sa Winsock bibliotekom

Kada aplikacija potpuno završi sa korišćenjem *Winsock* interfejsa, trebalo bi pozvati funkciju *WSACleanup* koja omogućava da se oslobode svi resursi alocirani od strane Winsock-a i da se otkažu svi čekajući pozivi koje je aplikacija napravila. Funkcija *WSACleanup* je deklarisana:

```
int WSACleanup(void);
```

Funkcija:	Opis:
`WSACleanup`	Završetak rada sa *Winsock* bibliotekom
Parametri:	Opis:
Nema parametara	
Povratna vrednost	Opis
`int`	Ako se funkcija uspešno izvrši vraća vrednost 0. U suprotnom, vraća se SOCKET_ERROR, a kod konkretne greške se dobija nakon poziva funkcije WSAGetLastError.

Primer:

```
WSACleanup();
```

Dobijanje informacije o nastaloj grešci

Ukoliko prilikom poziva *Winsock* funkcija dođe do greške funkcija će to signalizirati povratnom vrednošću (najčešće nenulta vrednost). Ali da bi se saznao konkretan uzrok greške i prekida izvršavanja, potrebno je pozvati funkciju `WSAGetLastError`. Ova funkcija vraća kod greške koja je izazvala prekid funkcije.

```
int WSAGetLastError (void);
```

Funkcija:	Opis:
`WSAGetLastError`	Vraća kod greške za poslednju *Winsock* funkciju koja se neuspešno izvršila
Parametri:	Opis:
	Nema ulaznih parametara
Povratna vrednost	Opis
`int`	Ukazuje na kod greške koja je izazvala prekid izvršavanja poslednje funkcije.

```
if (WSACleanup() != 0)
{
	printf("WSACleanup failed with error: %d\n", WSAGetLastError());
	return 1;
}
```

Funkcije za rad sa mrežnim redosledom okteta

Različiti procesori predstavljaju višebajtne podatke (brojeve) na 2 načina: tzv. „*big-endian*“ i „*little-endian*“ zapis. „*Big-endian*“ zapis označava da su bajti (okteti) u višebajtnim podacima poređani od najviše značajnog bajta (*MSB*) do najmanje značajnog bajta (*LSB*). „*Little-endian*“ označava obrnut zapis, odnosno bajti u višebajtnim podacima su poređani od najmanje značajnog bajta do najviše značajnog bajta.
Unutar nekog hosta IP adresa i broj porta (koji su višebajtni podaci) se zapisuju u tzv. *„host-byte“* redosledu odnosno u redosledu koji odgovara procesoru datog hosta.
Ali kada se IP adrese i brojevi porta specificiraju za mrežu, prema Internet standardu se moraju predstaviti u mrežnom redosledu okteta *(„network-byte“* redosled) koji odgovara„*big-endian“* zapisu. Dakle, pojam mrežnog redosleda okteta je uveden da bi se postigao zapis podataka nezavisan od platforme. Da bi programski kod bio nezavisan od platforme, koriste se funkcije koje odgovarajuće vrednosti prevode u vrednosti sa mrežnim redosledom okteta.
U svrhu prevođenja višebajtnih podataka iz *„host-byte“* u *„network-byte“* redosled, kao i obrnuto, postoji nekoliko gotovih API funkcija.
Sledeće dve funkcije služe za konverziju višebajtnog broja iz *„host-byte“* u *„network-byte“* redosled:

```
unsigned long htonl(unsigned long hostlong);
```

Funkcija:	Opis:
`htonl`	Konverzija višebajtnog broja iz „*host-byte*“ u „*network-byte*“ redosled pri čemu je zapis dug 4 bajta (*long integer*). Ime funkcije je skraćenica iz „**h**ost **to** **n**etwork **l**ong“.
Parametri:	Opis:
`hostlong [in]`	4-bajtni broj u *host byte* redosledu
Povratna vrednost	Opis
`unsigned long`	Ako se funkcija uspešno izvrši, vraća 4-bajtni broj u „*network-byte*“ redosledu.

```
unsigned short htons(unsigned short hostshort);
```

Funkcija:	Opis:
`htons`	Konverzija višebajtnog broja iz „*host-byte*“ u „*network-byte*“ redosled pri čemu je zapis dug 2 bajta (*short integer*). Ime funkcije je skraćenica iz „**h**ost **to** **n**etwork **s**hort“.
Parametri:	Opis:
`hostshort [in]`	2-bajtni broj u *host byte* redosledu
Povratna vrednost	Opis
`unsigned short`	Ako se funkcija uspešno izvrši, vraća 2-bajtni broj u „*network-byte*“ redosledu.

Sledeće dve funkcije vrše obrnutu konverziju, odnosno pretvaraju višebajtni broj iz „*network-byte*“ u „*host-byte*“ redosled:

```
unsigned long ntohl(unsigned long netlong);
```

Funkcija:	Opis:
`ntohl`	Konverzija višebajtnog broja iz „*network-byte*“ u „*host-byte*“ redosled pri čemu je zapis dug 4 bajta (*long integer*). Ime funkcije je skraćenica iz „**n**etwork **to** **h**ost **l**ong“.
Parametri:	Opis:
`netlong [in]`	4-bajtni broj u *network byte* redosledu
Povratna vrednost	Opis
`unsigned long`	Ako se funkcija uspešno izvrši, vraća 4-bajtni broj u „*host-byte*“ redosledu.

```
unsigned short ntohs(unsigned short netshort);
```

Funkcija:	Opis:
`ntohs`	Konverzija višebajtnog broja iz „*network-byte*“ u „*host-byte*“ redosled pri čemu je zapis dug 2 bajta (*short integer*). Ime funkcije je skraćenica iz „**n**etwork **to** **h**ost **s**hort“.
Parametri:	Opis:
`netshort [in]`	2-bajtni broj u *network byte* redosledu
Povratna vrednost	Opis
`unsigned short`	Ako se funkcija uspešno izvrši, vraća 2-bajtni broj u „*host-byte*“ redosledu.

Primer:

```
unsigned long some_long = 10;
unsigned short some_short = 20;
unsigned long network_byte_order;

// convert and  send
network_byte_order = htonl(some_long);
send(s, &network_byte_order, sizeof(unsigned int), 0);

some_short == ntohs(htons(some_short)); // this expression is true
```

Pretvaranje IPv4 adresa

Kao što je ranije pomenuto, potrebno je IPv4 adresu iz standardnog decimalnog zapisa sa tačkama pretvoriti u 32-bitnu vrednost tipa `unsigned long integer`. Ovu ulogu vrši funkcija inet_addr.

```
unsigned long inet_addr(const char *cp);
```

Funkcija:	Opis:
`inet_addr`	Pretvaranje IP adrese iz standardnog decimalnog zapisa u binarni zapis
Parametri:	**Opis:**
`cp[in]`	IP adresa u standardnoj decimalnoj notaciji (npr. 205.3.4.1) zapisana kao niz znakova (karaktera) koji se završava sa '\0'.
Povratna vrednost	**Opis**
`unsigned long`	Funkcija vraća IP adresu u binarnoj predstavi (zapisanu u vidu 32-bitnog `unsigned long integer` broja). Napomena: zapis je u mrežnom redosledu bajtova.

Funkcija koja vrši konverziju u suprotnom smeru (pretvaranje binarno zapisane adrese u standardnu decimalnu notaciju) zove se `inet_ntoa`. Deo imena je nastao kao skraćenica od engleskog naziva „numeric to ascii“.

```
char* inet_ntoa(struct in_addr in);
```

Funkcija:	Opis:
`inet_ntoa`	Pretvaranje IPv4 adrese iz binarnog zapisa u standardni decimalni zapis
Parametri:	**Opis:**
`in [in]`	in_addr struktura koja sadrži IPv4 adresu hosta
Povratna vrednost	**Opis**
`char*`	Ako je konverzija uspešna, vraća pokazivač na niz karaktera koji predstavljaju adresu u standardnoj decimalnoj notaciji. U slučaju greške vraća NULL pokazivač.

Primer

```
struct sockaddr_in socketAddress;
char * ipAddress;

// store IP in socketAddress
socketAddress.sin_addr.s_addr = inet_addr("10.0.0.1");
ipAddress = inet_ntoa(socketAddress.sin_addr);       // return the IP
printf("%s\n", ipAddress);                           // prints "10.0.0.1"
```

Zadavanje adrese

Kada klijent želi da komunicira sa serverom on mora specificirati serverovu IP adresu kao i broj porta da bi odredio protokol aplikacijskog sloja od kog traži servis. Takođe, i server koji osluškuje dolazne zahteve klijenata mora da specificira IP adresu i port na kome „osluškuje“.
Za zadavanje IPv4 adrese i porta koristi se *sockaddr_in* struktura.

```
struct in_addr
{
	uint32_t s_addr;	// IP adresa (32 bita)
};

struct sockaddr_in
{
	short		sin_family;
	unsigned short	sin_port;
	struct in_addr	sin_addr;
	char		sin_zero[8];
};
```

Sledi opis pojedinih polja strukture:

`sin_family`: ovo polje određuje adresnu familiju. Postavlja se na vrednost AF_INET što označava da *Winsock* koristi IPv4 adresnu familiju. Ako bi se koristila IPv6 adresna familija, onda bi se ovom polju dodelila vrednost AF_INET6.

`sin_port`: ovo polje definiše port koji će TCP ili UDP koristiti da identifikuje protokol aplikacijskog sloja. Aplikacije prilikom biranja porta moraju da znaju da je skup vrednosti od 0-1023 skup rezervisanih portova za postojeće protokole.

`sin_addr` : ovo polje predstavlja IPv4 adresu (32 bitna adresa). IP adrese se uobičajeno zadaju u „tačka decimalnom zapisu“, kao npr. 192.0.2.6. U nastavku ćemo opisati funkciju `inet_addr` koja IP adresu iz standardnog „tačka decimalno“ zapisa pretvara u 32-bitnu vrednost tipa `unsigned long integer`.

`sin_zero`: ovo polje se popunjava nulama, a služi da bi struktura `sockaddr_in` bila iste veličine kao i struktura `sockaddr`.

Sledi kratki primer kako da `SOCKADDR_IN` strukturu popunimo sa određenom IPv4 adresom ("136.149.3.29") i brojem porta (5150).

```
SOCKADDR_IN socketAddress;
short nPortId = 5150;

// Koristimo IPv4 adresnu familiju
socketAddress.sin_family = AF_INET;

// Pomocu funkcije inet_addr konvertovati datu IPv4 adresu (136.149.3.29) iz
// standardne decimalne notacije u 4-bajtni integer i dodeliti ga polju sin_addr
socketAddress.sin_addr.s_addr = inet_addr("136.149.3.29");

// Promenljiva nPortId je zapisana u host-byte redosledu. Pomocu funkcije htons
// pretvaramo nPortId u network-byte redosled, i dodeljujemo vrednost polju
// sin_port.
socketAddress.sin_port = htons(nPortId);
```

Neke funkcije WinSock biblioteke kao parametar poziva funkcije koriste strukturu `SOCKADDR` umesto strukture `SOCKADDR_IN`.

```
struct sockaddr {
        unsigned short sa_family;
        char sa_data[14];
};
```

U tom slučaju potrebno je samo kastovati strukturu adrese, pošto se za adresni prostor IPv4 biti ovih struktura poklapaju u memoriji. U nastavku je dat primer.

```
struct sockaddr_in * socketAddress1;
struct sockaddr * socketAddress2;


// deo koda u kome se incijalizuje socketAddress1
socketAddress1 = ...

if (socketAddress1->sa_family == AF_INET)
{
    socketAddress2 = (struct sockaddr *) socketAddress1;
}
```

Kreiranje utičnice (soketa)

Za kreiranje utičnice koristi se funkcuija *socket* deklarisana u *winsock2.h* zaglavlju.

```
SOCKET socket (int af, int type, int protocol);
```

Funkcija:	Opis:
`socket`	Kreiranje utičnice
Parametri:	**Opis:**
`af [in]`	Označava adresnu familiju. Pošto ćemo koristiti samo utičnice koje se oslanjaju na IPv4 protokol ovo polje se postavlja na vrednost `AF_INET`
`type [in]`	Označava tip utičnice. - SOCK_STREAM : Kada kreiramo utičnicu koja pruža uslugu pouzdanogtoka bajtova (*stream sockets*). Ove utičnice koriste TCP - SOCK_DGRAM: Kada kreiramo utičnicu koja pruža uslugu slanja datagrama bez kontrole greške (*datagram sockets*). Ove utičnice koriste UDP.
`protocol`	Označava konkretni transportni protokol. - Vrednost IPPROTO_TCP: u slučaju TCP protokola - Vrednost IPPROTO_UDP: u slučaju UDP protokola
Povratna vrednost	**Opis**
`SOCKET`	Vraća deskriptor kreirane utičnice. U slučaju greške vraća vrednost INVALID_SOCKET. Pozivom funkcije *WSAGetLastError* saznaje se kôd nastale greške.

Primer:

```
// kreiracemo „stream socket“ koji se oslanja na TCP/IP protokol stek.
SOCKET s;
s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
```

Povezivanje utičnice sa adresom

```
int bind(SOCKET s, const struct sockaddr *name, int namelen);
```

Funkcija:	Opis:
`bind`	Povezivanje utičnice sa IP adresom lokalnog interfejsa i brojem porta
Parametri:	**Opis:**
`s [in]`	Utičnica koja se povezuje. U slučaju servera to je utičnica na kojoj se čekaju klijentske konekcije.
`name [in]`	Pokazivač na adresnu strukturu lokalne adrese koja će biti pridružena utičnici.
`namelen [in]`	Veličina adresne strukture koja se prosleđuje (u bajtima)
Tip povratne vrednosti	**Opis**
`int`	Ako se funkcija uspešno izvrši vraća vrednost 0. U suprotnom, vraća se SOCKET_ERROR, a kod konkretne greške se dobija nakon poziva funkcije WSAGetLastError.

Primer:

```
sockaddr_in serverAddress;
SOCKET listenSocket;
unsigned short SERVER_PORT = 20;

// Initialize serverAddress structure used by bind
memset((char*)&serverAddress, 0, sizeof(serverAddress));

serverAddress.sin_family = AF_INET; //set server address protocol family
serverAddress.sin_addr.s_addr = INADDR_ANY;
serverAddress.sin_port = htons(SERVER_PORT);

// Create socket
listenSocket = socket(AF_INET, SOCK_STREAM,IPPROTO_TCP);

// Bind port number and local address to socket
iResult = bind(listenSocket,(struct sockaddr*) &serverAddress,
       sizeof(serverAddress));
```

Dobijanje informacija o utičnici

Za dobavljanje informacija o IP adresi i broju porta na koji je povezana utičnica koristi se funkcija `getsockname`. Tražene informacije se nalaze u ulazno-izlaznom argumentu funkcije tipa `sockaddr`. Ova funkcija je naročito korisna kada je za neku utičnicu pozvana funkcija `connect` bez prethodnog poziva `bind`. U tom sučaju `getsockname` obezbeđuje jedini način da se odredi lokalna adresa postavljena od strane sistema.

```
int getsockname(SOCKET s, struct sockaddr *localAddress, int *addrLen);
```

Funkcija:	Opis:
`getsockname`	Vraćanje lokalne adrese utičnice
Parametri:	**Opis:**
`s [in]`	Deskriptor utičnice
`localAddress [out]`	Pokazivač na SOCKADDR strukturu koja će prihvatiti adresu utičnice
`addrLen [in, out]`	Veličina bafera `localAddress` u bajtima
Povratna vrednost	**Opis**
`int`	Ako nema greške vraća 0. U slučaju greške vraća SOCKET_ERROR (pozivom funkcije *WSAGetLastError* saznaje se kôd konkretne greške.)

Primer:

```
SOCKET s;
struct sockaddr_in socketAddress;
int socketAddress_len = sizeof(socketAddress);

// place in code where socket is bind to some address and port
// ...

// Ask getsockname to fill in this socket's local adress
if (getsockname(s, (sockaddr *)&socketAddress, &socketAddress_len) == -1)
{
    printf("getsockname() failed.\n");
    return -1;
}

// Print the IP address and local port

printf("Local IP address is: %s\n", inet_ntoa(socketAddress.sin_addr));
printf("Local port is: %d\n", (int) ntohs(socketAddress.sin_port));
```

Dobijanje informacije o lokalnom računaru

```
int gethostname(char *name, int namelen);
```

Funkcija:	Opis:
`gethostname`	Funkcija vraća naziv lokalnog računara
Parametri:	Opis:
`name [out]`	Pokazivač na bafer koji će primiti ime lokalnog hosta
`namelen [in]`	Dužina (u bajtima) bafera na koji pokazuje `name`
Povratna vrednost	Opis
`int`	Ako nema greške vraća 0. U slučaju greške vraća SOCKET_ERROR (pozivom funkcije *WSAGetLastError* saznaje se kôd konkretne greške.)

Primer:

```
char localName[128] = "";
WSADATA wsaData;
WSAStartup(MAKEWORD(2, 2), &wsaData);
gethostname(localName, sizeof(localName));
printf("%s", localName);
```

```
struct hostent *gethostbyname(const char *name);
```

Funkcija:	Opis:
`gethostbyname`	Funkcija vraća podatke o lokalnom računaru na osnovu njegovog naziva
Parametri:	Opis:
`name [in]`	Pokazivač na ime hosta
Povratna vrednost	Opis
`struct hostent *`	Ako nema greške, vraća pokazivač na *hostent* strukturu (sadrži ime hosta, adresni tip, IP adrese). U slučaju greške vraća null pokazivač, a pozivom funkcije *WSAGetLastError* saznaje se kôd konkretne greške.

Struktura hostent

```
typedef struct hostent {
  char *     h_name;
  char **    h_aliases;
  short      h_addrtype;
  short      h_length;
  char **    h_addr_list;
} HOSTENT, *PHOSTENT, *LPHOSTENT;
```

Funkcija `gethostbyname` vraća pokazivač na `hostent` strukturu koja sadrži rezultate uspešne pretrage za hostom navedenim u parametru `name`. `Hostent` struktura se koristi za smeštanje informacija o hostu, konkretno o njegovom imenu i IP adresi.
Memorija za `hostent` strukturu se interno alocira, i na kraju rada interno oslobađa.
Sledi opis polja `hostent` strukture.

`h_name`: zvanično ime hosta. Ako se koristi DNS onda je to „fully qualified domain name“.
`h_aliass`: niz alternativnih imena hosta.
`h_addrtype`: tip adrese (AF_INET (IPv4), AF_INET 6 (Ipv6)).
`h_length`: dužina adrese u bajtima.
`h_addr_list`: lista adresa hosta. Adrese se vraćaju zapisane u mrežnom redosledu okteta.

U nastavku sledi primer u kome je prikazano dobavljanje informacija o imenu i ip adresi lokalnog računara (u tom slučaju parametar funkcije `gethostbyname` je prazan string).

```
// Get and print local ip address and host name
hostent* hostDetails = gethostbyname("");
printf("Hostname: %s.\n", hostDetails->h_name);

struct in_addr hostAddress = *(struct in_addr *)(hostDetails->h_addr_list[0]);
char* ip = inet_ntoa(hostAddress);
printf("\nIP address: %s.\n", ip);
```

Iz koda se može primetiti da se pristupalo samo prvom članu niza `h_addr_list`. Iako postoji mogućnost da računar ima više ip adresa, ova osobina neće biti korišćena u toku kursa.

Zatvaranje utičnice

Za svaki uspešno izvršen poziv funkcije `socket`, potrebno je po završetku korišćenja date utičnice pozvati funkciju `closesocket` da bi se oslobodili svi resursi koje je utičnica koristila. Nakon poziva funkcije `closesocket` pozivi za slanje i prijem podataka preko te utičnice će vratiti grešku.

```
int closesocket(SOCKET s);
```

Funkcija:	Opis:
`closesocket`	Funkcija za zatvaranje kreirane utičnice. Za svaki uspešno izvršen poziv funkcije `socket`, potrebno je po završetku korišćenja utičnice pozvati funkciju `closesocket` da bi se oslobodili svi resursi koje je utičnica koristila.
Parametri:	Opis:
`s [in]`	Deskriptor utičnice koja treba da se zatvori
Povratna vrednost	Opis
`int`	U slučaju uspešnog izvršenja, funkcija vraća 0. U slučaju greške vraća vrednost SOCKET_ERROR. Pozivom funkcije *WSAGetLastError* saznaje se kôd nastale greške.

Primer:

```
SOCKET s = socket(...);
connect(s, ...);

// prijem i slanje podatka preko uticnice
// na kraju rada sa uticnicom, ona se zatvara
closesocket(s);
```

ZADACI VEŽBE

OSNOVNI ZADATAK

1. Inicijalizovati Winsock biblioteku
   - **WSAStartup()**
2. Kreirati adresnu strukturu koja koristi ip adresu 127.0.0.1 i broj porta 55555.
   - **struct sockaddr_in**
3. Kreirati jedan TCP soket (utičnicu)
   - **socket()**
4. Povezati soket sa kreiranom adresnom strukturom
   - **bind()**
5. Zatvoriti soket
   - **closesocket()**
6. Zatvoriti Winsock biblioteku
   - **WSACleanup()**

DODATNI ZADATAK - za one radoznale

1. Dobaviti i ispisati lokalno ime i adresu računara
   - **gethostbyname()**
2. Kreirati adresnu strukturu koja koristi dobijenu ip adresu iz prethodnog koraka (umesto ipadrese 127.0.0.1)
   - **struct sockaddr_in**
3. Dobaviti i ispisati adresu soketa nakon povezivanja soketa sa kreiranom adresom
   - **getsockname()**

ZADATAK

Data je implementacija programa koji omogućava da se iz teksta koji je uneo korisnik programa izračuna učestalost određenog slova.

1. Kreirati novi prazan projekat u Visual Studio okruženju. U projekat je potrebno dodati novu .cpp datoteku i u nju uneti primer koji je dat u nastavku.
2. U implementaciji datog primera skrivene su greške. Koristeći prozor za prikaz grešaka potrebno je otkriti mesta u kodu gde se greške nalaze i ispraviti greške.
3. Izmeniti program tako da se korisniku omogući da može više puta proveriti učestalost određenog karaktera.
4. Omogućiti kraj izvršenja programa ako korisnik unese karakter “q”.
5. Potrebno je uzeti u obzir ponavljanja i malih i velikih slova koje je uneo korisnik.
6. Korišćenjem kontrolisanog izvršenja programa saznati koja je vrednost promenljive “i” nakon izvršenja for petlje.

```
#ifdef _MSC_VER
	#define _CRT_SECURE_NO_WARNINGS
#endif

#include <stdio.h>
#include <stdlib.h>

int main()
{
	char str[1000], ch;
	int i = 0, frequency = 0;
	printf("Enter a string: ")
	gets(str);
	printf("Enter a character to find the frequency: ");
	scanf("%c",&ch);
	for(; str[i] != '\0'; ++i)
	{
		if(ch == str[i])
			++frequency;
	}
	printf("Frequency of %c = %d", ch, freq);
	return 0;
}
```