

Prvi C++ program: hello.c

```
#include <iostream>
using namespace std;

int main()
{
    // ovo je komentar - sve do kraja linije

    cout << "Hello World!" << endl
    return 0;
}
```

- **using namespace std** - program koristi imena iz standardne biblioteke
- **cout** je izlazni objekt (konzola), koji pomoću operatora
- **<<** prima poruku "Hello World!" koju ispisuje na konzoli
- **endl** je tzv C++ manipulator za novu liniju ('\n' iz C-a)
- **//** je oznaka za početak komentara, koji vrijedi do kraja linije

Namespace – leksičko označavanje objekata i funkcija

Prema ANSI C++ standardu grupa funkcija i objekata može se dodatno leksički označiti imenovanim područjem – namespace. Tako je predviđeno da sve standardne funkcije i objekti pripadaju leksičkom području std.

```
// C++ program: hello2.cpp
```

```
#include <iostream>
```

```
int main()
```

```
{
```

```
std::cout << "Hello world!"  
          << std::endl;
```

```
    return 0;
```

```
}
```

```
// C++ program: hello2.cpp
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    cout << "Hello world!"  
        << endl;
```

```
    return 0;
```

```
}
```

Ako se ne napiše (using namespace std) tada treba ispred izlaznog objekta cout i manipulatora endl napisati std::, što znači da ovi objekti pripadaju leksičkom području std (operator :: naziva se rezolucijski operator).

C++ klase spremanja memorijskih objekata

Programer mora voditi računa o zauzeću memorije za tri tipa objekata:

1. Statički objekti - postoje za cijelo vrijeme izvršenja programa

2. Automatski objekti - su objekti koji se definiraju unutar funkcije ili kao argumenti funkcija.

Postoje samo za vrijeme izvršenja funkcije – u memorijskom području koje se naziva programski stog - "run- time stack".

3. Dinamički objekti

Dinamički objekti nastaju korištenjem tehnike dinamičkog alociranja memorije, a nestaju dealociranjem memorije (u tu svrhu se u C++ jeziku koriste `new` i `delete` operatori).

Dinamičko alociranje pomoću new i delete operatora

```
#include <iostream>
using namespace std;

char *ime;

int main()
{
    unsigned god;
    ime = new char(30);

    cout << "Tvoje ime:"
    cin >> ime;
    cout << "Tvoje godine:"
    cin >> god;
    cout << ime << "je"
         << god << "star"
         << endl;
    delete ime;
    return 0;
}
```

```
// globalna varijabla
// (definirana izvan funkcije)

// lokalna varijabla
// dinamička varijabla
// alocira se 30 bajta na heapu

// globalni eksterni objekti;
//  cout - izlazni tok
//  cin  - ulazni tok

// dealocira se memorija
```

Uoči da I/O objekti prepoznaju tip podatka na koji djeluju!

Deklaracije i definicije

❑ **Deklaracije** varijable (ili funkcije) najavljuje da ta varijabla (funkcija) postoji, ali je njena definicija dana negdje drugdje.

❑ **Definicijom** varijable (funkcije) rezervira se potrebna memorija i u nju se smješta sadržaj varijable ili programski kod funkcije

☰ **Varijabla mora biti deklarirana ili definirana prije same upotrebe.**

C++ ne podržava eksplicitno koncept modularnog programiranja — već se taj koncept posredno ostvaruje odvajanjem source i header datoteka. "Source" datoteka tipično sadrži definicije koje se mogu separatno kompajlirati. "Header" datoteka tipično sadrži deklaracije pomoću kojih drugi dijelovi programa znaju kako koristiti varijable i funkcije koje su definirane u nekom vanjskom – "extern" modulu.

```
extern int size;      // deklaracija
void hello( void);    // prototip funkcije
int size;             //definicija varijable



void hello( void)      // definicija funkcije
{ cout << "hello!" << endl; }
```

Hello projekt

prog. cpp	Sadrži main funkciju:	<pre>#include "hello. h" //potrebno zbog deklaracije hello() int main(void) { hello(); return 0; }</pre>
hello. h	deklaracija funkcija definiranih u hello. cpp:	<pre>void hello(void);</pre>
hello. cpp	sadrži "header" datoteke s deklaracijom funkcija standardne biblioteke i iz hello.cpp	<pre>#include <iostream> using namespace std; //deklaracija cout i endl #include "hello.h" // deklaracija funkcije hello() void hello (void) { cout << "hello world" << endl; }</pre>



Kompajliranje jednostavnih C++ programa

Kompajliranje programa koji je zapisan u jednoj datoteci - prog.cpp:

-  `cl prog. cpp`
generira izvršnu datoteku prog.exe
-  `cl /Fe myprog.exe prog.cpp`
generira izvršnu datoteku myprog.exe

Kompajliranje programa koji je zapisan u više datoteka - prog.cpp i hello.cpp::

-  `cl /Fe myprog.exe prog.cpp hello.cpp`
Kompajliranje s pre-kompajliranom objektnom datotekom:

-  `cl -c hello. cpp`
generira se objektna datoteka hello.obj
-  `cl /Fe myprog.exe prog.cpp hello.obj`
kompajlira se prog.cpp i linkuje hello.obj

<Header.h> sadrže deklaracije potrebne za povezivanje (linkanje) odvojenih objektnih "modula"

Makefile – i program make (nmake)

Makefile je tekstualna datoteka koja opisuje procesa kompiliranja kojim rukovodi program imena make (nmake kod VC). Sastoji se od komentara (#...) makro definicija (CXX=cl), i linija u kojima se navode komande za kompajliranje i linkanje izvršnog programa:

```
#Version of the C++ compiler; link and compile options:
CXX = cl
CFLAGS = /GX

#Popis objektnih datoteka:
OBJ = prog.obj hello.obj

#stvaranje izvršnog programa prog.exe:
prog.exe: ${OBJS}
    ${CXX} ${CFLAGS} ${OBJS} /Fe prog.exe

#prog.obj i hello.obj zavise o sadržaju prog.c, hello.c i hello.h:

prog.obj : prog.cpp hello.h
    ${CXX} ${CFLAGS} -c prog.cpp
hello.obj : hello.cpp hello.h
    ${CXX} ${CFLAGS} -c hello.cpp
```

Posebni program imena nmake (na Unixu - imena make) se poziva s komandne linije:

```
C:> nmake          ( ako je datoteka spremljena pod imenom "makefile") Ili
C:> nmake -f hello.mak  ( ako je datoteka spremljena pod imenom "hello.mak")
```