Blokovska struktura C++ programa ima četiri razine:

1. **razina datoteke (temeljna kompilacijska jedinica)**
2. **razina definicije (tijela) funkcije**
3. **razina bloka kontrolnih struktura (sekvenca, iteracija, selekcija)**
4. **razina bloka koji je omeđen vitičastim zagradama**

Blok niže razine može biti umetnut unutar bloka više ili iste razine proizvoljan broj puta, jedino se ne smije vršiti definicija funkcije unutar tijele neke druge funkcije.

Doseg **(eng. scope) nekog identifikatora je dio programa, odnosno blok u kojem se taj identifikator može koristiti.**

Vrijeme postojanja varijable **(eng. lifetime) je programsko vrijeme u kojem ta varijabla postoji u memoriji računala**

Doseg argumenata funkcije i lokalnih varijabli

Doseg argumenata funkcije je tijelo funkcije. Doseg lokalnih varijabli se proteže od mjesta deklariranja do kraja složenog iskaza koji je omeđen vitičastim zagradama

Identifikatori s različitim područjima dosega, iako mogu imati istu oznaku, međusobno se neovisni.

```
int main( void) {
     double eps, x, ex;
     ...                                  doseg ex, x
     return 0;
}

float my_exp( double x, double epsilon)
{                                                    doseg x
     int i;
     double ex = 1.0, preth_ex = 0.0, ... ;
     ...                                             doseg ex
     return ex;
}
```

Nije dozvoljeno više deklaracija s istim imenom u istom dosegu.

```
     float epowx( float x, float epsilon)
     {
       int x;  // greška:
       ...     // ime x je već pridjeljeno parametru
```

Lokalne deklaracije imaju prednost nad vanjskim deklaracijama. Kažemo da lokalna deklaracija prekriva **vanjsku deklaraciju.**

```
f( int x, int a)
{
   int y, b;
   y = x + a* b;

   if (… )
   {
     int a, b; // a prekriva parameter a
     ---       // b prekriva var. b iz vanjskog dosega
     y = x + a* b;
    }
}
```

Automatske varijable

Lokalne varijable imaju ograničeno vrijeme postojanja, pa se nazivaju i automatske varijable:

- **nastaju u memoriji pozivom funkcije u kojoj su deklarirane,**
- **nestaju iz memorije nakon povrata u pozivnu funkciju.**

```
void f( int a, int x)
{
   cout << " a =" << a << ", x = " << x <<endl;
   a = 3;
   {
     int x = 4;
     cout << " a =" << a << ", x = " << x <<endl;
   }
   cout << " a =" << a << ", x = " << x <<endl;
   x = 5; //nema nikakovi efekt
}

int main( void)
{
  int a = 1, b = 2;
  f( a, b);
  cout << " a =" << a << ", b = " << b <<endl;
  return 0;
}
dobije se ispis:
a = 1, x = 2
a = 3, x = 4
a = 3, x = 2
a = 1, b = 2
```

Lokalne statičke varijable

Ako se lokalna varijabla deklarira s prefiksom `static`, tada se za tu varijablu trajno rezervira mjesto u memoriji (postoji i nakon izvršenja funkcije), iako je njen doseg ograničen unutar tijela funkcije.

```
int incr_count(int mod)
{
   static int count=0;
   if(++count == mod) count = 0;
   return count;
}

int main(void)
{
  int i,mod=3;
  for(i=0; i<=10; i++)
      cout << incr_count(mod) <<  ", ";
  cout << "...\n";
  return 0;
}

c:>cl /GX countmod3.cpp
c:>countmod3
1, 2, 0, 1, 2, 0, 1, 2, 0, 1, ...
```

Važno je uočiti da kada se statička lokalna varijabla u deklaraciji i inicijalizira, (pr. `static count=0;`), ta inicijalna vrijednost vrijedi samo pri prvom pozivu funkcije. Djelovanje inicijalizacije statičkih varijabli nije isto kao kod lokalnih automatskih varijabli, gdje se inicijalizacija vrši pri svakom pozivu funkcije.

Globalne varijable

Globalne varijable su varijable koje se deklariraju izvan tijela fukcije. To su 'permanentne' variable koje trajno zauzimaju memoriju za vrijeme trajanja programa u kojem su definirane. Doseg globalnih varijabli je od točke definiranja do kraja datoteke u kojoj su definirane. Kasnije ćemo pokazati kako se njihov doseg može proširiti i na druge datoteke

```
int main( void)
{
        …      /* varijabla max ovdje ne postoji !! */
}

int max = 0; /* nadalje je definirana var.  max */

void compute( … )
{
     max =              /* var max ovdje postoji !! */
}
```

Argumenti funkcije i lokalne varijable prekrivaju globalne varijable, ako imaju isto ime.
Globalne varijable se inicijaliziraju na vrijednost 0.
U C++ jeziku se u svakom bloku može pristupiti globalnim varijablama korištenjem operatora ::, primjerice, nakon izvršenja programa

```
float max = 5.6;
void add2max(int max )
{ // max je lokalna varijabla,
  // ::max je globalna varijabla
    :: max += max;
}
```

Namespace

Dobra je praksa da se pri pisanju velikih programa identifikatori, koji pripadaju logičkoj cjelini, grupiraju u zajednički leksički prostor – namespace.

```
namespace grupa
{
    float max = 5.6;
    // ..... ostali članovi leksičke grupe
};


void add2max(int max )
{
    // max je lokalna varijabla,
    // grupa::max je globalna varijabla
    // iz namespace grupa

   grupa::max += max;
}
```

Napomena: svi identifikatori iz standardne biblioteke pripadaju u namespace std.

Statičke i eksterne globalne varijable

Globalne varijable se također mogu deklarirati s prefiksom `static`. U tom slučaju varijabla je vidljiva samo u kompilacijskoj jedinici unutar koje je i definirana. Ako se varijabla deklarira bez prefiksa `static`, ona se može dosegnuti i iz drugih kompilacijskih jedinica. U tom slučaju varijablu u drugim datotekama treba deklarirati kao vanjsku varijablu. Deklaracija vanjske varijable označava se prefiksom `extern`.

Primjer: program za brojač napisat ćemo u dvije datoteke.

U prvoj je definirana globalna varijabla `count` i funkcija `inc_counter` koja inkrementira vrijednost od `count` po modulu mod.

U drugoj datoteci je definiran glavni program koji koristi vrijednost od `count` i funkciju `inc_counter`.

```
/* Datoteka1: count.cpp */

int count=0;

incr_count(int mod)
{
  count++;
  if(count >= mod)
     count = 0;
}
```

```
/* Datoteka2: countmain.cpp */

#include <iostream>

extern int count;

void incr_count(int mod);

int main(void)
{
  int i,mod=3;
  for(i=0; i<=10; i++)
  {
       incr_count(mod);
       std::cout << count << endl;
  }
  std::cout << "...\n";
  return 0;
}
```

Program kompajliramo pomoću Microsoft VC kompajlera komandom:

```
c:>cl /GX countmain.cpp counter.cpp
```

Programski moduli

Modul je dio nekog programa koji sadrži skup međuovisnih globalnih varijabli i funkcija. Modul, koji se piše s ciljem opće namjene, obično se formira u dvije grupe datoteka:

1. **datoteke specifikacije modula (`ime.h`) s deklaracijama globalnih varijabli i funkcija, koje su implementirane unutar modula.**

2. **datoteke implementacije (`ime.cpp`) s definicijom varijabli i funkcija. Implementacijske se datoteke mogu kompajlirati kao samostalne kompilacijske jedinice, a dobiveni objektni kod se može pohraniti u biblioteku potprograma.**

Primjer: **formirat ćemo modul u kojem će se implementirati funkcija brojača po modulu mod. Problem ćemo obraditi nešto općenitije, kako bi modul prilagodili različitim primjenama.**

1. Specifikacija modula je opisana u datoteci mcounter.h.

```
/* Datoteka: mcounter.h
* specifikacija funkcija brojača po modulu mod
*/

void reset_counter(int mod);
/* inicira brojač na početnu vrijednost nula
 * i modul brojača na vrijednost mod. Ako je  mod<=1,
 * modul brojaca se postavlja na vrijednost INT_MAX
 */

int get_count(void);
/* vraća trenutnu vrijednost brojača */

int get_modulo(void);
/* vraća trenutnu vrijednost modula brojača */

int incr_count(void);
/* incrementira vrijednost brojača za 1
 * Ako vrijednost brojača postane jednaka ili veća,
 * od zadamog modula vrijednost brojača postaje nula.
 * Vraća: trenutnu vrijednost brojača
 */
```

2. Implementacija modula je opisana u datoteci mcounter.cpp.

```
// Datoteka: mcounter.cpp

 #include <limits.h>    // zbog definicija INT_MAX

// globalne varijable
 static int _count = 0   // početno stanje brojača
 static int _mod = INT_MAX;   //2147483647

 void reset_counter(int mod)
 {
    _count= 0;
    if(mod <= 1)    _mod = INT_MAX;
    else            _mod =  mod;
 }

 int get_count(void) { return _count;}
 int get_modulo(void){ return _mod; }

 int incr_count(void)
 {
    _count++;
    if(_count >= _mod) _count = 0;
    return _count;
 }
```

3.Testiranje modula provest ćemo programom testcount.cpp:

```
#include <iostream>
#include "counter.h"

int main(void)
{
  int i;
  reset_counter(0,5);
   std::cout << "Brojac po modulu "
             << get_modulo() << endl;

  for(i=0; i<=10; i++)  {
     incr_count();
     std::cout << get_count();
  }
  std::cout << "...\n");
  return 0;
}
```

Izrada programske biblioteke

Nakon što je modul testiran možemo primijetiti da on može biti koristan i u drugim programima. Pokazat ćemo da pri tome nije potrebno njegovo ponovno prevođenje, ako smo ga prethodno preveli u objektni kod. Primjerice, prethodni program smo mogli kompajlirati na slijedeći način:

Prvo, datoteka counter.cpp se prevede u objektni kod komandom:

```
c:>cl /c /GX mcounter.cpp
```

Ovime se dobije datoteka counter.obj. (parametar komandne linije –c je poruka kompajleru da se prevod izvrši u objektni kod).

Pokažimo još kako se formira biblioteka potprograma pomoću Microsoft program lib.exe.

Objektnu datoteku counter.obj uvrstit ćemo u biblioteku koju ćemo nazvati mylib.lib, slijedećom komandom:

```
c:>lib /OUT:mylib.lib  counter.obj
```

(/OUT: je parametar komandne linije iza kojeg se navodi ime biblioteke).

Izvršni program sada možemo dobiti komandom:

```
c:>cl /GX testcount.cpp mylib.lib
```

ZAKLJUČIMO

- **Programi se u C++ jeziku mogu pisati u više odvojenih datoteka - modula.**
- **Ponovno prevođenje cijelog programa uzima dosta vremena, dok se pojedina datoteka, koja je manja od ukupnog programa, prevodi mnogo brže.**
- **U modulu se može definirati određeni skup funkcija, koji se može koristiti i u drugim programima.**
- **Module, koji sadrže često korištene funkcije, može se u obliku objektnog koda uvrstiti u binarne biblioteke potprograma.**
- **Modul se može pisati, testirati, i ispravljati neovisno od ostatka programa. Proces ispravljanja je pojednostavljen, jer se analizira samo jedan mali dio programa.**
- **Moduli omogućavaju veću preglednost i logičku smislenost programskog koda, jer se u njima obično obrađuje jedinstvena problematika. Primjerice, za obradu matematičkih problema postoje različiti programski paketi s odvojenim modulima za rad s kompleksnim brojevima, vektorima, matricama, itd.**
- **Korištenjem principa odvajanja specifikacije od implementacije modula, i skrivanjem podataka koji bilježe stanja objekta koji modul opisuje, dobijaju se moduli neovisni od programa u kojem se koriste. To znatno olakšava timski rad na razvoju softvera.**