**C++ szintaxis**

Amikor figyelembe vesszük a C++ programot, úgy definiálhatjuk, hogy olyan objektumok gyűjteménye, amik kommunikálnak egymás módszerein keresztül. Most vizsgáljuk meg röviden, hogy mit jelent az osztály, az objektum, az eljárás és az azonnali változók.

**Objektumok:** Állapotuk és viselkedésük van. Pl.: Egy kutyának van állapota (szín, név, fajta,) és viselkedése (vándorlás,ugatás,evés). Az objektum egy osztály példánya.

**Osztályok:** Az adattartalmukban esetleg eltérő, de viselkedés szempontjából megegyező objektumokat tekintjük egy osztályba tartozóknak.

**Eljárás:** A módszer alapvetően viselkedés. Az osztály számos módszert tartalmazhat. Olyan módszerekben történik, ahol a logikát írják, az adatokat manipulálják, és minden műveletet végrehajtanak.

**Példányváltozók:** Minden objektum egyedi példányváltozókkal rendelkezik. Az objektum állapotát az ilyen példányváltozókhoz rendelt értékek hozzák létre.

**C++ program felépítése**

Nézzünk meg egy egyszerű kódot, amely kiírná a képernyőre a *Hello World* szavakat.

**#include <iostream>**

**using namespace std;**

**int main()**

**{**

**cout << "Hello World";**

**return 0;**

**}**

Nézzük meg a fenti program különböző részeit:

A C++ nyelv több **header**-t definiál, olyan információkat tartalmaznak, amik szükségesek vagy hasznosak a programhoz. Ehhez a programhoz az <**iostream**> header szükséges.

A **using namespace std** azt mondja a fordítónak, hogy használja az std névtért. A névterek viszonylag frissek a C++ -hoz.

Az **int** **main()**, ahol a program végrehajtása megkezdődik

Ezután a **cout<<”Hello World”;**,ami kiírja a képernyőre az adott szöveget.

Az utolsó sor a **return 0;** befejezi a main () függvényt, és azt eredményezi, hogy a 0 értéket visszaadja a hívási folyamatnak.

**A C++ program fordítása és futtatása**

Nézzük meg, hogyan lehet menteni a fájlt, fordítani és futtatni a programot. Kövesd az alábbi lépéseket:

Nyiss meg egy terminált (ctrl**+alt+t**).

Nyiss meg egy szövegszerkesztőt a terminálba (gedit **valami.cpp**) és másold be a fenti kódot. Mentsd el a fájlt(**ctrl+s**).

Írd be a **g++ valami.cpp** **-o valami** parancsot és nyomd le az entert a kód fordításához. Ha nincs hiba, akkor a parancssor kurzora a következő sorra lép.

Majd írd be **./valami** és a képernyőn megjelenik a Hello world szöveg.

**Pontos vessző és blokkok C++-ban**

A C++-ban a pontosvessző egy „utasítás terminátor”.Ez azt jelenti, hogy minden egyes utasítást pontosvesszővel kell „elválasztani”.

pl. a következő 3, különböző állítások.

**x=y;**

**y=y+1;**

**add(x, y);**

A **blokk** a logikailag összekapcsolt állítások halmaza, amiket a nyitó- és záró zárójelek segítségével „különítünk el”, így átláthatóbbá téve a kódunkat.

pl. **{**

**cout<<”Szeretlek anyu”;**

**return 0;**

**}**

A C++ nem ismeri fel a sor végét terminátorként. Emiatt nem számít ,hogy hová tettél egy állítást egy sorban.

pl.: **x=y;**

**y=y+1;**

**add(x, y);**

ugyanaz, mint

**x=y; y=y+1; add(x, y);**

**C++ azonosítók**

A C++ azonosító egy változó, funkció, osztály, modul vagy bármely más felhasználó által meghatározott elem azonosítására használt név.

Az azonosító egy **A-tól Z-ig** vagy **a-tól z-ig** vagy egy **aláhúzással** kezdődik, amit **nulla** vagy **több betű**, aláhúzás és számjegy követ (0-9).

A C++ nem engedélyezi a @,$,% írásjeleket. **Nagybetű- érzékeny** programozási nyelv. Tehát a **munkaerő** és a **Munkaerő** két különböző azonosító a C++-ban.

Példák: mohd, zara, abc, move\_name, a\_123, \_temp, a23b9, retVal

**C++ kulcsszavak**

A következő lista a C++-nak van fenn tartva, ezért nem használhatók konstans, változó vagy más azonosító nevekként.

|  |  |  |  |
| --- | --- | --- | --- |
| asm | else | new | this |
| auto | enum | operator | throw |
| bool | explicit | private | true |
| break | export | protected | try |
| case | extern | public | typedef |
| catch | false | register | typeid |
| char | float | reinterpret\_cast | typename |
| class | for | return | union |
| const | friend | short | unsigned |
| const\_cast | goto | signed | using |
| continue | if | sizeof | virtual |
| default | inline | static | void |
| delete | int | static\_cast | volatile |
| do | long | struct | wchar\_t |
| double | mutable | switch | while |
| dynamic\_cast | namespace | template |  |

**Whitespace a C++-ban**

A **whitespace-t** tartalmazó sort,esetleg egy **megjegyzéssel** ellátott sort, üres sorként ismerjük. és a C++ fordító teljesen figyelmen kívül hagyja azt.

A whitespace egy kifejezés, ami leírja az üreseket, lapokat, új sor karaktereket és a megjegyzéseket. A whitespace az utasítás egy részét elválasztja a másiktól, és lehetővé teszi a fordító számára, hogy azonosítsa, hogy egy utasításban egy elem, mint például az **int,** hol végződik és a következő elem hol kezdődik.

Pl.: **int age**

A fenti utasításban a fordítónak legalább egy üres helyet kell hagyni az **int** és az **age** között, hogy megtudja őket különböztetni.

Pl.: **fruit = apples + oranges;**

A fenti példában nincs szükség szóközre a fruit és az egyenlőségjel között, illetve az egyenlőségjel és az apples között sem, bár pár szóközt beilleszthetsz a jobb átláthatóság érdekében.

**Megjegyzések**

A program megjegyzései magyarázó szövegek, amiket a C++ kódba lehet beírni. Ezek a megjegyzések segítik a forráskód olvasását. Minden programozási nyelv megenged valamilyen megjegyzést.

A C++ támogatja az egysoros és többsoros megjegyzéseket. A C++ fordító figyelmen kívül hagyja a megjegyzéseken belül elérhető összes karaktert.

A kommentek **/\***-gal kezdődnek és **\*/** -rel végződnek.

Kommentelni **//**-rel is tudunk, ilyenkor figyelmen kívül hagyja a fordító, amit a // jel után írunk.

**Adattípusok**

Bármilyen nyelvű program írása során különféle változókat kell használnia a különböző információk tárolásához. A **változó** nem más, mint az **értékek tárolására fenntartott memóriahely.** Ez azt jelenti, hogy amikor egy változót hozol létre, akkor az egy bizonyos helyet foglal el a memóriában.

Különböző adattípusokat tárolhatsz, pl.: **character**, **wide** **character**, **integer**, **floating** **point**, **double** **floating** **point, boolean** stb. A változó adattípusa alapján, az operációs rendszer memóriát különít el és eldönti, hogy mit lehet tárolni a lefoglalt memóriában.

**Primitív beépített típusok**

A C++ a programozó számára a beépített, valamint a felhasználó által definiált adattípusok gazdag választékát kínálja. A következő táblázat 7 alapvető C++ adattípust sorol fel.

|  |  |
| --- | --- |
| típus | Kulcsszó |
| Logikai / Boolean | bool |
| Karakter / Character | char |
| Egész szám / Integer | int |
| Lebegőpont / Floating point | float |
| Dupla lebegőpont / Double Floating point | double |
| Értéktelen / Valueless | void |
| Széles karakter / Wide character | wchar\_t |

Az alaptípusok közül több módosítható, egy vagy több módosító segítségével.

pl.: **signed**, **unsigned**, **short**, **long**

Az alábbi táblázat mutatja a változók típusát, mennyi memóriát vesz igénybe az érték tárolása a memóriában, és mi a max. illetve min. érték, amik ilyen típusú változókban tárolhatók

|  |  |  |
| --- | --- | --- |
| Típus | Bitszélesség | Tartomány |
| char | 1byte | -127 to 127 or 0 to 255 |
| unsigned char | 1byte | 0 to 255 |
| signed char | 1byte | -127 to 127 |
| int | 4bytes | -2147483648 to 2147483647 |
| unsigned int | 4bytes | 0 to 4294967295 |
| signed int | 4bytes | -2147483648 to 2147483647 |
| short int | 2bytes | -32768 to 32767 |
| unsigned short int | Range | 0 to 65,535 |
| signed short int | Range | -32768 to 32767 |
| long int | 4bytes | -2,147,483,648 to 2,147,483,647 |
| signed long int | 4bytes | same as long int |
| unsigned long int | 4bytes | 0 to 4,294,967,295 |
| float | 4bytes | +/- 3.4e +/- 38 (~7 digits) |
| double | 8bytes | +/- 1.7e +/- 308 (~15 digits) |
| long double | 8bytes | +/- 1.7e +/- 308 (~15 digits) |
| wchar\_t | 2 or 4 bytes | 1 wide character |

A változó mérete eltérhet a fenti táblázatban látottaktól, a fordítótól és használt számítógéptől függően.

A következő egy példa, ami megfelelő méretű különböző adattípust készít a számítógépeden.

**#include <iostream>**

**using namespace std;**

**int main() {**

**cout << "Size of char : " << sizeof(char) << endl;**

**cout << "Size of int : " << sizeof(int) << endl;**

**cout << "Size of short int : " << sizeof(short int) << endl;**

**cout << "Size of long int : " << sizeof(long int) << endl;**

**cout << "Size of float : " << sizeof(float) << endl;**

**cout << "Size of double : " << sizeof(double) << endl;**

**cout << "Size of wchar\_t : " << sizeof(wchar\_t) << endl;**

**return 0;**

**}**

Ez a példa **endl**-t használ, ami minden sor és << operátor után új sor karaktert helyez be, hogy több értéket adjon ki a képernyőre. A **sizeof ()** operátort is használjuk a különböző adattípusok méretének megszerzéséhez.

Amikor a fenti kódot lefordítjuk és lefuttatjuk, a következőt hozza létre, ami persze gépenként változhat.

Size of char : 1

Size of int : 4

Size of short int : 2

Size of long int : 4

Size of float : 4

Size of double : 8

Size of wchar\_t : 4

**A typedef deklaráció**

A **typedef** használatával új nevet hozhatsz létre egy meglévő típushoz. A következő egy egyszerű szintaxis az új típus definiálásához a typedef használatával.

**Typedef type newname;**

Például, a következő azt mondja a fordítónak, hogy a feet egy másik neve az int-nek

**typedef int feet;**

Szóval, a következő deklaráció teljesen jó és **integer** változót hoz létre **distance** néven.

**feet distance;**

**Számozott típusok**

**Változó típusok**

A változó megadja nekünk a nevezett tárolóeszközt, amit a programunk manipulálni tud. Minden változónak van egy meghatározott típusa, ami meghatározza a változó memóriájának méretét és elrendezését;az adott memóriában tárolható értéktartomány és a változóra alkalmazható műveletek halmaza.

A változó neve betűkből, számokból és aláhúzás karakterekből állhat. Viszont betűvel vagy egy aláhúzással kell kezdődnie. A nagy- és kis betű eltérő karakternek számít, mert a C++ kis- - és nagybetű érzékeny.

A korábban már említett C++nyelv változóinak alapvető típusai a következők:

1. **bool** - Igaz vagy hamis értéket tárol
2. **char** - Jellemzően egy bájt. Egész szám típus.
3. **int** - Egész szám típus. A „legtermészetesebb” a gép számára.
4. **float** - Lebegőpontos érték
5. **double** - Dupla lebegőpontos érték
6. **void** - A típus hiányát jelenti.
7. **wchar\_t** - Széles karaktertípus

C++ meghatározz más különböző típusú változókat is, amiket ezekben a fejezetekben fogunk tárgyalni, úgy, mint **enumeráció**, **pointer**, **array, referenciák, adatstruktúrák** és **osztályok**.

A következő rész a különböző típusú változók **meghatározására**, **deklarálására** és **használatára** fog vonatkozni.

**Változó meghatározása**

Egy változó definíciója azt mondja a fordítónak, hogy hol és mennyi tárhelyet kell létrehozni a változó számára. A változó meghatározása adattípust határoz meg, és egy vagy több ilyen változó listáját tartalmazza, az alábbiak szerint:

**type variable\_list;**

A típusnak, érvényes C++ adattípust kell tartalmaznia, beleértve a **char, w\_char, int, float, double, bool** vagy bármely felhasználó által meghatározott objektumot stb., és a **variable\_list** egy vagy több azonosító nevet tartalmazhat vesszővel elválasztva. Néhány érvényes deklaráció:

**int i, j, k;**

**char c, ch;**

**float f, salary;**

**double d;**

Az **int i, j, k** sor egyaránt deklarálja és definiálja az **i, j, k** változókat, ami utasítja a fordítót, hogy int típusú **i, j, k** változókat hozzon létre.

A változók **inicializálhatók (**egy **kezdeti értékhez rendelhetők)** a deklarációban. Az inicializáció egyenlőség jelből áll, amit egy konstans kifejezés követ.

**Type variable\_name = value;**

Néhány példa:

extern int d = 3, f=5; a d és f deklarációja

int d = 3, f=5; a d és f definiálása és inicializálása

byte z = 22; a z definiálása és inicializálása

char x = 'x'; az x változó x értéket kapott

Az inicializáció nélküli definíció: a statikus tárolási időtartamú változók implicit módon inicializálódnak **NULL**-lal( **minden** **bájt** **értéke** **0**), az összes többi változó kezdeti értéke nincs meghatározva.

**Változó deklaráció**

A változó deklarálás biztosítja a fordító számára, hogy létezik egy változó adott névvel és típussal, így a fordító további fordítást végezhet anélkül, hogy teljes részletességgel lenne szüksége a változóra. A változó deklarációnak csak a fordítás idején van jelentősége, a fordítónak a program összekapcsolása idején tényleges változó definícióra van szüksége.

A változó deklarálás hasznos, ha több fájlt használsz, és a változót az egyik fájlban határozod meg, ami a program összekapcsolása idején elérhető lesz. Az **extern** kulcsszót a változó bármely helyen történő deklarálására fogjuk használni. Bár a C++-ban többször is deklarálhatsz egy változót, de csak egyszer lehet definiálni egy fájlban, függvényben vagy kódblokkban. Próbálkozz az alábbi példával, ahol egy változó van deklarálva a kód ”tetején”, de azt a **main** -ben definiáltuk.

**#include <iostream>**

**using namespace std;**

**// Változó deklaráció:**

**extern int a, b;**

**extern int c;**

**extern float f;**

**int main () {**

**// Változó definiálása:**

**int a, b;**

**int c;**

**float f;**

**// Inicializálás**

**a = 10;**

**b = 20;**

**c = a + b;**

**cout << c << endl ;**

**f = 70.0/3.0;**

**cout << f << endl ;**

**return 0; }** Amikor lefordítod és lefuttatod a kódot, ezt kapod eredménynek:

30

23.3333

Ugyanez a koncepció vonatkozik a funkció deklarálására is,ahol a deklaráció során funkció nevet adsz meg és ennek tényleges definíciója bárhol máshol is megadható. Például

// funkció deklarálása

int func();

int main() {

// funkció hívása

int i = func();

}

// funkció definiálása

int func() {

return 0; }

**Lvalues és Rvalues**

A C++2 fajta kifejezést tartalmaz:

1. **Lérték**: A memóriahelyre utaló kifejezéseket "lvalue" kifejezésnek nevezzük. Megjelenhet a bal illetve jobb oldalon is.
2. **Rérték**: Olyan adatértékre utal, amit a memória bizonyos címén tárol. Egy rvalue a hozzárendelés jobb oldalán megjelenthet , de a bal oldalán nem

A **változók** **Lértékek**, ezért megjelenhetnek a hozzárendelés **bal** oldalán. A numerikus literálok **Rértékek** ezért nem lehet hozzárendelni és nem jelenhetnek meg a bal oldalon. A következő egy érvényes utasítás:

**int g = 20;**

Ez viszont nem érvényes és compile-time error-t generál.

**10 = 20;**

**A változó scope-ja (hatóköre)**

A hatókör a program egy régiója és általánosságban 3 hely van, ahol a változók deklarálhatók:

1. Olyan funkció vagy blokk belsejében, amit helyi változóknak nevezzünk.
2. A függvényparaméterek definíciójában, amit formális paramétereknek nevezünk.
3. Az összes funkción kívül, amit globális változóknak nevezünk.

A következő fejezetekben megtanuljuk, hogy mi a funkció és a paraméter. De előtte ismerjük meg a helyi és globális változókat.

**Helyi változók**

A függvényben vagy blokkban deklarált változók a **helyi** változók. Csak olyan állításoknál használhatók, amik a függvényben vagy a kódblokkban vannak. A helyi változók nem ismertek a sajátjukon kívül. A következő példák a helyi változókat használják:

**#include <iostream>**

**using namespace std;**

**int main () {**

**// helyi változó deklarálása:**

**int a, b;**

**int c;**

**// inicializálás:**

**a = 10;**

**b = 20;**

**c = a + b;**

**cout << c;**

**return 0;**

**}**

**Globális változók**

A globális változók az összes függvényen kívül vannak meghatározva, általában a program tetején. Értéküket a program teljes időtartama alatt tartják.

Ezeket a változókat bármely funkció használhatja. Ez azt jelenti, hogy a deklaráció után a változó az egész programban használható. A következő példák a globális és helyi változókat használják.

**#include <iostream>**

**using namespace std;**

**// globális változó deklarációja:**

**int g;**

**int main () {**

**// Helyi változó deklarálása:**

**int a, b;**

**// inicializálás**

**a = 10;**

**b = 20;**

**g = a + b;**

**cout << g;**

**return 0;**

**}**

A programban adhatunk azonos nevet a helyi és globális változók számára, de a helyi változó értéke részesül előnyben a függvényben. Például:

**#include <iostream>**

**using namespace std;**

**// globális változó deklarációja:**

**int g = 20;**

**int main ()**

**{**

**//Helyi változó deklarálása:**

**int g = 10;**

**cout << g;**

**return 0;**

**}**

Amikor a fenti kódot lefordítjuk és lefuttatjuk, a következő eredményt kapjuk:

**10**

**Helyi és globális változók inicializálása**

Amikor egy helyi változó definiálva van, akkor a rendszer nem inicializálja azt, hanem nekünk kell azt inicializálni. A globális változókat a rendszer automatikusan inicializálja, amikor ezeket a következőképpen definiáljuk:

|  |  |
| --- | --- |
| Adattípus | Initializer |
| int | 0 |
| char | '\ 0' |
| float | 0 |
| double | 0 |
| pointer | NULLA |

Jó programozási gyakorlat a változók inicializálása, ellenkező esetben a program váratlan eredményt ad.

**Konstansok,literálok**

A **konstansok** olyan fix értékekre utalnak, amelyeket a program nem változtathat meg, és amiket **literálok**nak nevezünk.

**A konstanstípusok lehetnek az alapadattípusok bármelyike.**

A konstansokat úgy kezeljük, mint a szokásos változókat, azzal a különbséggel, hogy értékeik nem módosíthatók a definiálásuk után.

**Integer literál**

Az integer literál lehet decimális, oktális vagy hexadecimális konstans. Az előtag meghatározza az alapot: 0x vagy 0X hexadecimális, 0 az oktális, és semmi a decimális literálnak. Az integer tartalmazhat utótagot is, ami az U és az L kombinációja, az **unsigned**nak és a **long**nak. Az utótag nagybetűs vagy kisbetűs lehet, és a sorrendjük nem számít. Példák az integer literálról:

**212 // Legal**

**215u // Legal**

**0xFeeL // Legal**

**078 // Illegal: 8 is not an octal digit**

**032UU // Illegal: cannot repeat a suffix**

A következő példák az integer literálok különböző típusaira vonatkoznak.

**85 // decimal**

**0213 // octal**

**0x4b // hexadecimal**

**30 // int**

**30u // unsigned int**

**30l // long**

**30ul // unsigned long**

**Lebegőpontos literál**

A lebegőpontos literálnak egész része, tizedespontja, töredéke és exponens része van. Decimális vagy exponenciális formában jelenthetjük meg.

A **decimális** formát használva meg kell adnunk a **tizedespontot**, az **exponens** részt vagy **mindkettőt**. Ha **exponenciális** formában jelenítjük meg, akkor az **integer** részt és a **töredéket**, vagy **mindkettőt** meg kell adnunk. A signed exponenset e vagy E jelöli. Például:

**3.14159 // helyes forma**

**314159E-5L // helyes forma**

**510E // helytelen: nincs befejezve az exponens**

**210f // helytelen: nem decimális vagy exponenciális forma**

**.e55 // helytelen: hiányzik az integer rész**

**Logikai literál**

Két logikai literál van és C++ kulcsszavai közé tartoznak.

1. Az igaz érték értéke igaz.
2. A hamis értéke hamis.

Nem szabad az igaz értéket 1-nek , a hamisat 0-nak tekinteni.

**Karakter literál**

A karakterláncok egyetlen idézőjelbe kerülnek. Ha a literál L-lel kezdődik (csak nagy betűvel), akkor **wide** **character** (pl.: **L'x**), és **wchar\_t** típusú változóban kell tárolni. Ellenkező esetben keskeny karakteres (pl.:**'X'**) és egy egyszerű **char** változóban tárolható.

A karakter literál lehet **egyszerű** **karakter** (pl.:**'X'**), **menekülési** **szekvencia** (pl.:**'\t'**) vagy **univerzális** **karakter** (pl.:**'0202'**).

Vannak bizonyos karakterek a C++-ban, amiket,ha egy **backslash** előz meg, speciális jelentőséget kapnak, mint például:

|  |  |
| --- | --- |
| Menekülési szekvencia | Jelentés |
| \\ | \ karakter |
| \” | ' karakter |
| \” | " karakter |
| \? | ? karakter |
| \ a | Riasztás vagy csengő |
| \ b | Backspace |
| \ f | Form feed |
| \ n | Új sor |
| \ r | Szállítás vissza |
| \ t | Vízszintes lap |
| \ v | Függőleges lap |
| \ ooo | Egy-három számjegyű oktális szám |
| xhh. . . | Egy vagy több számjegy hexadecimális száma |

**#include <iostream>**

**using namespace std;**

**int main() {**

**cout << "Hello\tWorld\n\n";**

**return 0;}**

A kód kimenete: Hello World

**String literál**

A sztring literálokat idézőjelek közé rakjuk. A karakterláncokhoz hasonló karakterek: karakterek, menekülési szekvenciák és univerzális karakterek.

A hosszú sorokat több sorra tördelhetjük a **string** használatával és **elkülöníthetjük** őket a **szóközök** használatával. Például.: (Mindhárom forma azonos karakterlánc.)

"hello, dear"

"hello, \

dear"

"hello, " "d" "ear"

**Konstansok meghatározása**

Két egyszerű módja van a C++-ban:

1. A #define előfeldolgozó használata.
2. A const kulcsszó használata.

**A #define preprocessor**

Az alábbi egy formula, ami a **#define** előfeldolgozót használja a **konstans** meghatározásához:

**#define identifier value**

A következő példa részletesen megmagyarázza:

#include <iostream>

using namespace std;

#define LENGTH 10

#define WIDTH 5

#define NEWLINE '\n'

int main() {

int area;

area = LENGTH \* WIDTH;

cout << area;

cout << NEWLINE;

return 0;}

**A const kulcsszó**

Használhatjuk a **const** prefix-et egy meghatározott típusú konstans deklarálásához. Például.: **const type variable = value;**

A következő példa részletesen megmagyarázza:

**#include <iostream>**

**using namespace std;**

**int main() {**

**const int LENGTH = 10;**

**const int WIDTH = 5;**

**const char NEWLINE = '\n';**

**int area;**

**area = LENGTH \* WIDTH;**

**cout << area;**

**cout << NEWLINE;**

**return 0;}**

**Módosító típusok**

A C++ lehetővé teszi, hogy a **char, int** és **double** adattípusok rendelkezzenek az előttük lévő módosítókkal. A módosítót az alaptípus jelentésének megváltoztatására használják, hogy pontosabban illeszkedjen a különböző helyzetek igényeihez. (**Signed, unsigned, long, short)**

A signed, unsigned, long, short változók **integer** alaptípusokra alkalmazhatók. Továbbá a **signed** és **unsigned** alkalmazható a **char** változó prefixumaként és a **long** a **double** prefixumaként. A **signed** és az **unsigned** használható a **long** és **short** módosítók prefixumaiként. Pl.: **unsigned long int**

A C++ lehetővé teszi,hogy deklarálásnál rövidítsünk az unsigned, short vagy long integereknél. Simán használhatod csak a szavakat (unsigned, short, long) az int nélkül. A következő 2 példa **unsigned integer** változót deklarál.

**unsigned x;**

**unsigned int y;**

Ahhoz, hogy megértsük a módbeli különbséget a signed és unsigned integer módosítók között, futtatnunk kell az alábbi programot.

**#include <iostream>**

**using namespace std;**

**int main() {**

**short int i; // a signed short integer**

**short unsigned int j; // an unsigned short integer**

**j = 50000;**

**i = j;**

**cout << i << " " << j;**

**return 0; }**

A program kimenete: -15536 50000

A fenti eredmény azért van, mert az 50 000-et **short unsigned integerként** ábrázoló bitmintát egy -15,536-asnak kell értelmezni.

**Típus selejtezők**

A típus selejtezők további információkat szolgáltatnak változókról, amiket ”megelőznek”.

1. Const - A **const** típusú objektumokat nem tudjuk megváltoztatni a program végrehajtása során.
2. Volatile - A volatile módosító azt mondja a fordítónak, hogy a változó értéke megváltoztatható a program által meg nem határozott módon.
3. Restrict - A pointer

**Tárolási osztályok**

A tárolási osztály meghatározza a C++ programon belüli változók és /vagy funkciók hatókörét (láthatóságát) és élettartamát. Ezek a specifikációk megelőzik a módosított típusokat. A C++ programban használható tárolási osztályok: **auto, register, static, extern, mutable.**

**Az auto tárolási osztály**

Az automatikus tárolási osztály az összes helyi változó alapértelmezett tárolási osztálya.

**{ int mount;**

**auto int mount;**

**}**

A fenti példa két változót definiál ugyanarra a tárolási osztályra, az auto csak a funkción belül használható,mint például a helyi változók.

**A register tárolási osztály**

A regiszter tárolási osztályt, olyan helyi változók definiálásra használjuk, amiket a RAM helyett egy regiszterben kell tárolni. Ez azt jelenti, hogy a változó maximális mérete megegyezik a regiszter méretével (általában egy szóval) és nem tudja az egyváltozós ”&” operátort alkalmazni ( mivel nincs memóriahelye).

**{**

**register int miles;**

**}**

A regisztert csak olyan változóknál szabad használni, amik gyors hozzáférést igényelnek, például a **számlálók**. Azt is meg kell jegyezni, hogy a regiszter definiálása nem jelenti azt , hogy a változó a regiszterben lesz tárolva. Hanem azt jelenti, hogy lehetséges a tárolása regiszterben, a hardver és a végrehajtási korlátozások függvényében.

**A statikus tárolási osztály**

A statikus tárolási osztály utasítja a fordítót, hogy tartsa fenn a helyi változót a program élettartama alatt, ahelyett, hogy minden alkalommal létrehozza és elpusztítsa a változót, amikor az bejönne és kimenne a hatáskörből. Ezért a helyi változók statikussá tétele lehetővé teszi számunkra, hogy megtartsák az értékeiket a funkció (függvény) hívások között.

A statikus módosító a globális változókra is alkalmazható. Ha ez megtörténik, akkor ez a változó hatóköre azokra a fájlokra korlátozódik, amelyekben deklarálták,

A C++-ban, amikor egy statikus adatot használnak egy osztályadat- tagon, csak az egyik tag egy példányát osztja meg osztályának minden objektuma.

**#include <iostream>**

**// Function declaration**

**void func(void);**

**static int count = 10; /\* Global variable \*/**

**main() {**

**while(count--) {**

**func();**

**}**

**return 0;**

**}**

**// Function definition**

**void func( void ) {**

**static int i = 5; // local static variable**

**i++;**

**std::cout << "i is " << i ;**

**std::cout << " and count is " << count << std::endl;**

**}**

A fenti kód eredménye fordítás és futtatás után:

**i is 6 and count is 9**

**i is 7 and count is 8**

**i is 8 and count is 7**

**i is 9 and count is 6**

**i is 10 and count is 5**

**i is 11 and count is 4**

**i is 12 and count is 3**

**i is 13 and count is 2**

**i is 14 and count is 1**

**i is 15 and count is 0**

**Az extern tároló osztály**

Az extern tárolóosztályt egy globális változó hivatkozására használjuk, ami az összes programfájl számára látható. Amikor az **extern**-t használjuk a változó nem inicializálható.

Amikor több fájlod van és globális változót vagy függvényt definiálsz, amiket több fájlban is használsz, akkor az extern lesz használva, hogy megadja a már meghatározott változót vagy függvényt. Röviden, az extern-t globális változó vagy függvény deklarálásra használjuk más fájlokban.

Az extern módosítót a leggyakrabban akkor használjuk, ha két vagy több fájl ugyanazokkal a globális változókkal vagy funkciókkal (függvényekkel) rendelkezik, mint az alábbiakban ismertetjük.

Első fájl: main.cpp

**#include <iostream>**

**int count ;**

**extern void write\_extern();**

**main() {**

**count = 5;**

**write\_extern();**

**}**

Második fájl: Support.cpp

**#include <iostream>**

**extern int count;**

**void write\_extern(void) {**

**std::cout << "Count is " << count << std::endl;**

**}**

Itt az **extern** kulcsszót használják egy másik fájlban szereplő **count** deklarálására. Fordítsuk a két fájlt a következőképp: **g++ main.cpp support.cpp -o write**

Így egy write futtatható programot hozzunk létre, próbáld futtatni(**./write** és ellenőrizd az eredményt,ami **5.**

**A mutable tároló osztály**

A mutable specifikátor csak az osztályobjektumokra vonatkozik, amiket később tárgyalunk.

Lehetővé teszi egy objektum tagjának, hogy felülírja a **const** tag funkciót. Ez azt jelenti, hogy

egy **mutable tag** egy **konstans tag** funkcióval módosítható.

**Operátorok**

Az operátor egy olyan szimbólum, ami utasítja a fordítót, hogy konkrét matematikai vagy logikai manipulációkat végezzen. A C++ operátorokban gazdag, és a következő típusú operátorokat biztosítja:

**Feladatkezelők, aritmetikai, relációs, logikai, bitwise és egyéb operátorok.**

Ez a fejezet egyenként fogja megvizsgálni ezeket az operátorokat.

**Aritmetikai operátor**

Tegyük fel,hogy az A változó 10-est és a B pedig 20-ast tárol.

|  |  |  |
| --- | --- | --- |
| Operátor | Leírás | Példa |
| + | Összeadja a két változóban tárolt értéket | A + B 30-at ad |
| - | A másodikat kivonja az elsőből | A-B -10-et ad |
| \* | Összeszorozz A és B-t | A \* B 200-at ad |
| / | Elosztja a nevezőt a számlálóval | B / A 2 |
| % | Modulus Operátor és egy egész osztás utáni része | B% A 0 értéket ad |
| ++ | növeli az egész értéket eggyel | Az A ++ 11 -t ad |
| - | csökkenti az egész értéket eggyel | A-- 9-et ad |

**Relációs operátorok**

Tegyük fel,hogy az A változó 10-est és a B pedig 20-ast tárol.

|  |  |  |
| --- | --- | --- |
| Operátor | Leírás | Példa |
| == | Ellenőrzi, hogy a két operandus értéke egyenlő-e vagy sem, ha igen, akkor a feltétel valóra válik. | (A == B) nem igaz. |
| ! = | Ellenőrzi, hogy a két operandus értéke egyenlő-e vagy sem, ha az értékek nem egyenlőek, akkor a feltétel valóra válik. | (A! = B) igaz. |
| > | Ellenőrzi, hogy a bal operandus értéke nagyobb-e, mint a jobb operandus értéke, ha igen, akkor a feltétel valóra válik. | (A> B) nem igaz. |
| < | Ellenőrzi, hogy a bal operandus értéke kisebb-e, mint a jobb operandus értéke, ha igen, akkor a feltétel valóra válik. | (A <B) igaz. |
| > = | Ellenőrzi, hogy a bal operandus értéke nagyobb-e vagy egyenlő-e a jobb operandus értékével, ha igen, akkor a feltétel valóra válik. | (A> = B) nem igaz. |
| <= | Ellenőrzi, hogy a bal operandus értéke kisebb-e vagy egyenlő a jobb operandus értékével, ha igen, akkor a feltétel valóra válik. | (A <= B) igaz. |

**Logikai operátor**

Tegyük fel,hogy az A változó 1-est és a B pedig 0-ást tárol.

|  |  |  |
| --- | --- | --- |
| Operátor | Leírás | Példa |
| && | **Logikai AND operátor.** Ha mindkét operandus nem nulla, akkor a feltétel valóra válik. | (A && B) hamis |
| || | **Logikai OR operátor.** Ha a két operandus közül bármelyik nem nulla, akkor a feltétel valóra válik. | (A || B) igaz |
| ! | **Logikai NOT operátor.** Megfordítja az operandus logikai állapotát. Ha egy feltétel igaz, akkor hamissá válik és fordítva. | ! (A && B) igaz. |

**Bitwise operátor**

A bitwise operátor bittel működik és bit-bit műveletet hajt végre. Igazságtábla &, |, és ^ jelnek.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| p | q | p & q | p | q | p ^ q |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 |

Tegyük fel, ha A = 60; és B = 13; most bináris formátumban a következők lesznek:

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A | B = 0011 1101

A ^ B = 0011 0001

~ A = 1100 0011

A C ++ nyelv által támogatott Bitwise operátorok az alábbi táblázatban találhatók. Tegyük fel, hogy az A változó 60-at tartalmaz, és a B változó 13-at:

|  |  |  |
| --- | --- | --- |
| Operátor | Leírás | Példa |
| & | A bináris AND operátor egy bitet másol az eredményre, ha mindkét operandusban létezik. | (A & B) 12-et ad, ami 0000 1100 |
| | | A bináris VAGY operátor egy bitet másol, ha létezik egy operandusban. | (A | B) 61-et ad, ami 0011 1101 |
| ^ | A bináris XOR operátor átmásolja a bitet, ha egy operandusban van beállítva, de nem mindkettőben. | (A ^ B) 49-et ad, ami 0011 0001 |
| ~ | A bináris kiegészítők operátora egyváltozós, és ennek hatására a "bittel". | (~ A) -61-et ad, ami 1100 0011 a 2-es komplement formában egy signed bináris szám miatt. |
| << | Bináris bal shift operátor. A bal oldali operandusok értékét balra mozgatja, a jobb operandus által megadott bitek számával. | A << 2 240-et ad, ami 1111 0000 |
| >> | Bináris jobb shift operátor. A bal oldali operandusok értékét jobbra mozgatja, a jobb operandus által megadott bitek számával. | A >> 2 15-et ad, ami 0000 1111 |

**Feladatkezelők**

A C ++ nyelv támogatja a következő hozzárendelési operátorokat:

|  |  |  |
| --- | --- | --- |
| Operátor | Leírás | Példa |
| = | Egyszerű hozzárendelési operátor, Értékeket rendel a jobb oldali operandusoktól a bal oldali operandushoz. | C = A + B hozzárendeli az A + B értékét a C-hez |
| + = | Összeadás AND hozzárendelés operátor, a jobb oldali operandust hozzáadja a baloldali operandushoz, és hozzárendeli az eredményt a bal operandushoz. | C + = A egyenértékű C = C + A értékkel |
| - = | Kivonás AND hozzárendelés operátor, kivonja a jobb operandust a baloldali operandusból, és hozzárendeli az eredményt a bal operandushoz. | C - = A egyenértékű a C = C - A értékkel |
| \* = | Szorzás AND hozzárendelés operátor, a jobb operandust szorozza meg a bal operandussal, és az eredményt hozzárendeli a bal operandushoz. | C \* = A egyenértékű a C = C \* A értékkel |
| / = | Osztás AND hozzárendelés operátor, a baloldali operandust a jobb operandussal osztja, és az eredményt hozzárendeli a bal operandushoz. | C / = A egyenértékű a C = C / A értékkel |
| % = | Modulus AND hozzárendelési operátor, két operandust használ, és az eredményt hozzárendeli a bal operandushoz. | C% = A egyenértékű C = C% A értékkel |
| << = | Bal shift AND feladatkezelő | C << = 2 ugyanaz, mint C = C << 2 |
| >> = | Jobb shift AND feladatkezelő | C >> = 2 ugyanaz, mint C = C >> 2 |
| & = | Bitwise AND feladatkezelő | C és 2 azonos a C = C & 2 értékkel |
| ^ = | Bitwise kizáró VAGY feladatkezelő | C ^ = 2 ugyanaz, mint C = C ^ 2 |
| | = | Bitwise megengedő VAGY feladatkezelő | C | = 2 ugyanaz, mint C = C | 2 |

## Egyéb operátorok

Az alábbi táblázat a C ++ által támogatott egyéb operátorokat sorolja fel.

|  |  |
| --- | --- |
| Sr.No | Üzemeltető és leírás |
| 1 | **sizeof**  [A sizeof operátor](https://translate.googleusercontent.com/translate_c?depth=1&hl=hu&rurl=translate.google.com&sl=auto&sp=nmt4&tl=hu&u=https://www.tutorialspoint.com/cplusplus/cpp_sizeof_operator.htm&xid=17259,15700002,15700023,15700124,15700186,15700190,15700201,15700237,15700242,15700248&usg=ALkJrhjTG0hUEbwEuJJRZtjlOs-T66UQyA) visszaadja a változó méretét. Például méret (a), ahol az 'a' egész szám, és visszatér 4. |
| 2 | **Feltétel ?** **X: Y**  A  [condition operátor (?)](https://translate.googleusercontent.com/translate_c?depth=1&hl=hu&rurl=translate.google.com&sl=auto&sp=nmt4&tl=hu&u=https://www.tutorialspoint.com/cplusplus/cpp_conditional_operator.htm&xid=17259,15700002,15700023,15700124,15700186,15700190,15700201,15700237,15700242,15700248&usg=ALkJrhhghrd6DdIeJXEkWhRhwki4-HYisA) . Ha a feltétel igaz, akkor az X értéke visszatér, egyébként Y értéket ad vissza. |
| 3 | **,**  [A vessző operátor](https://translate.googleusercontent.com/translate_c?depth=1&hl=hu&rurl=translate.google.com&sl=auto&sp=nmt4&tl=hu&u=https://www.tutorialspoint.com/cplusplus/cpp_comma_operator.htm&xid=17259,15700002,15700023,15700124,15700186,15700190,15700201,15700237,15700242,15700248&usg=ALkJrhiaOUuJb8zAtdoGP8m4inQOs5uFRA) |
| 4 | **.** **(pont) és -> (nyíl)**  [A tag operátor](https://translate.googleusercontent.com/translate_c?depth=1&hl=hu&rurl=translate.google.com&sl=auto&sp=nmt4&tl=hu&u=https://www.tutorialspoint.com/cplusplus/cpp_member_operators.htm&xid=17259,15700002,15700023,15700124,15700186,15700190,15700201,15700237,15700242,15700248&usg=ALkJrhiIkOEhiOcgpigk2NY0keJd3nCg8g) az osztályok, struktúrák és egyesítések egyes tagjai számára hivatkoznak. |
| 5 | **Cast**  [A casting operátorok](https://translate.googleusercontent.com/translate_c?depth=1&hl=hu&rurl=translate.google.com&sl=auto&sp=nmt4&tl=hu&u=https://www.tutorialspoint.com/cplusplus/cpp_casting_operators.htm&xid=17259,15700002,15700023,15700124,15700186,15700190,15700201,15700237,15700242,15700248&usg=ALkJrhgEIrJ5YFFQ7hE5SWnge5NRxZLomQ) egy adattípust konvertálnak egy másikra. Például int (2,2000) visszatér 2-re. |
| 6 | **&**  [A pointer operátor &](https://translate.googleusercontent.com/translate_c?depth=1&hl=hu&rurl=translate.google.com&sl=auto&sp=nmt4&tl=hu&u=https://www.tutorialspoint.com/cplusplus/cpp_pointer_operators.htm&xid=17259,15700002,15700023,15700124,15700186,15700190,15700201,15700237,15700242,15700248&usg=ALkJrhgle-_OENdYCmePAcNWvtclb4pAbg) a változó címét adja vissza. Például &a; megadja a változó tényleges címét. |
| 7 | **\***  [A pointer operátor \*](https://translate.googleusercontent.com/translate_c?depth=1&hl=hu&rurl=translate.google.com&sl=auto&sp=nmt4&tl=hu&u=https://www.tutorialspoint.com/cplusplus/cpp_pointer_operators.htm&xid=17259,15700002,15700023,15700124,15700186,15700190,15700201,15700237,15700242,15700248&usg=ALkJrhgle-_OENdYCmePAcNWvtclb4pAbg) egy változó mutatója. Például \*var; a var változóra fog mutatni. |

**Operátorok elsőbbsége**

Az operátor elsőbbsége határozza meg a kifejezések csoportosítását. Ez hatással van egy kifejezés értékének az értékelésére. Néhány operátor nagyobb prioritással rendelkezik, mint mások,például a szorzás operátornak nagyobb prioritása van, mint az összeadásnak.

A legmagasabb prioritással rendelkező operátorok a táblázat tetején jelennek meg,míg a legalacsonyabbak pedig az alján. Egy kifejezésben a magasabb prioritású operátorokat értékeljük.

|  |  |  |
| --- | --- | --- |
| Kategória | Operátor | asszociativitás |
| Postfix | () [] ->. ++ - - | Balról jobbra |
| egyoperandusú | + -! ~ ++ - - (típus)\* & sizeof | Jobbról balra |
| multiplikatív | \* / % | Balról jobbra |
| Adalékanyag | + - | Balról jobbra |
| shift | << >> | Balról jobbra |
| összefüggő | <<=>> = | Balról jobbra |
| Egyenlőség | ==! = | Balról jobbra |
| Bitwise ÉS | & | Balról jobbra |
| Bitwise XOR | ^ | Balról jobbra |
| Bitwise VAGY | | | Balról jobbra |
| Logikai ÉS | && | Balról jobbra |
| Logikai VAGY | || | Balról jobbra |
| Feltételes | ?: | Jobbról balra |
| Feladat | = + = - = \* = / =% = >> = << = & = ^ = | = | Jobbról balra |
| Vessző | , | Balról jobbr |

**Ciklus fajták**

Előfordulhat, hogy többször kell végrehajtani egy kódblokkot. Általánosságban elmondható, hogy az utasításokat egymás után hajtják végre: Egy függvényben az első utasítás hajtódik végre először, majd a második és így tovább.

A programozási nyelvek különböző vezérlési struktúrákat biztosítanak, amik lehetővé teszik a bonyolultabb végrehajtási útvonalakat.

A **ciklusok** lehetővé teszik számunkra, hogy egy **utasítást** vagy utasítások csoportját **többször** is **végrehajtsuk.** A C++ az alábbi típusú ciklusokat biztosítja a követelmények kezelésére.

|  |  |
| --- | --- |
| Sr.No | ciklus típusa és leírása |
| 1 | while ciklus  Megismétli a utasítást vagy a utasítások csoportját, míg egy adott feltétel igaz. A ciklus test végrehajtása előtt teszteli a feltételt. |
| 2 | For ciklus  Az utasításokat többször egymás után hajtja végre, és rövidíti a kódot,ami kezeli a ciklusváltozót. |
| 3 | Do while ciklus  Olyan, mint egy while ciklus, kivéve, hogy a do while a ciklus test végrehajtása után teszteli a feltételt. |
| 4 | beágyazott ciklus  Használhatunk egy vagy több ciklust egy másik "while", "for" vagy "do..while" cikluson belül. |

**While ciklus szintaxisa**

**#include <iostream>**

**using namespace std;**

**int main () {**

**// Helyi változó deklarálás:**

**int a = 10;**

**// while ciklus végrehajtása:**

**while( a < 20 ) {**

**cout << "value of a: " << a << endl;**

**a++;**

**}**

**return 0;**

**}**

**For ciklus szintaxisa**

**#include <iostream>**

**using namespace std;**

**int main () {**

**// for ciklus futtatása**

**for( int a = 10; a < 20; a = a + 1 ) {**

**cout << "value of a: " << a << endl;**

**}**

**return 0;**

**}**

**Do while ciklus**

Hasonló a while ciklushoz, de a do while garantáltan legalább egyszer lefut. A feltétel a ciklus végén jelenik meg, így a ciklusban lévő utasítás egyszer lefut és csak utána történik meg a feltétel tesztelése.

**#include <iostream>**

**using namespace std;**

**int main () {**

**// helyi változó deklarálása:**

**int a = 10;**

**// do ciklus futtatása**

**do {**

**cout << "value of a: " << a << endl;**

**a = a + 1;**

**} while( a < 20 );**

**return 0;**

**}**

**Beágyazott ciklusok**

Egy ciklus beágyazható egy másik ciklusba. A C++-ban ez 256 mélységi szintig lehetséges.

**Szintaxis**

1. Beágyazott for ciklus

**for (i=1; i<10; i++)**

**{**

**for (init; feltétel; növelés)**

**{**

**utasítások;**

**}**

**utasítások; //ha akarsz belerakhatsz több utasítást is**

**}**

1. Beágyazott while ciklus

**while (feltétel) {**

**while (feltétel) {**

**utasítások;**

**}**

**utasítások; //ha akarsz belerakhatsz több utasítást is**

**}**

1. Beágyazott do while ciklus

**do {**

**utasítás;**

**do {**

**utasítás;**

**} while (feltétel) ;**

**} while (feltétel);**

A következő példa egy beágyazott for ciklussal keresi meg a prímszámokat 2-100 között.

**#include <iostream>**

**using namespace std;**

**int main ()**

**{**

**int i, j;**

**for ( i=2; i <100; i++) {**

**for ( j=2; j<= (i/j) ; j++)**

**if (!(i%j)) break;**

**if( j > (i/j)) cout << i << ”primszam” <<endl;**

**}**

**return 0;**

**}**

**Ciklus vezérlő utasítások**

A ciklusvezérlési utasítások megváltoztatják a végrehajtást a normális sorrendjéről egy másikra. Ha a végrehajtás „elhagyja” a hatókörét, az összes objektum, ami ebben a hatókörben lett létrehozva megsemmisül.

|  |  |
| --- | --- |
| Sr.No | Ellenőrzési nyilatkozat és leírás |
| 1 | **Break utasítás**  Lezárja a **ciklus** vagy a **kapcsoló** utasítását, és átadja a végrehajtást a ciklus vagy a kapcsoló után közvetlenül. |
| 2 | **Continue utasítás**  Azt okozza, hogy a ciklus a test többi részét kihagyja, és azonnal újra teszteli megismétli állapotát. |
| 3 | **Go to utasítás**  Átadja az ellenőrzést a címkézett utasításnak. Bár nem ajánlott a goto utasítás használata a programban. |

**Break utasítás**

**A break utasítás 2 lehetséges használata a C++-ban:**

1. Amikor a break utasítás egy cikluson belül találkozik, a ciklus azonnal megszűnik és a program folytatódik a ciklus utáni következő utasításban.
2. Használható arra, hogy leállítson egy **case**-t a **switch** utasításban.(Erről később.)

Ha beágyazott ciklust használsz, a **break** utasítás **leállítja** a legbelső ciklus végrehajtását és elkezdi végrehajtani a kód block után lévő következő sorát.

**#include <iostream>**

**using namespace std;**

**int main () {**

**// helyi változó deklarálása:**

**int a = 10;**

**// do ciklus végrehajtása:**

**do {**

**cout << "value of a: " << a << endl;**

**a = a + 1;**

**if( a > 15) {**

**// ciklus befejezése**

**break;**

**}**

**} while( a < 20 );**

**return 0; }**

**Continue utasítás**

A continue utasítás némileg hasonlít a break utasításhoz. Ahelyett, hogy a megszüntetést kényszerítené folytatja a ciklus következő iterációjának

A for ciklus esetében, a **continue** a **condition** és az **increment** rész végrehajtását eredményezi.

A **while** és a **do** **while** esetében a programvezérlés átadja a feltételeket.

**#include <iostream>**

**using namespace std;**

**int main () {**

**// helyi változó deklarálása:**

**int a = 10;**

**// do ciklus végrehajtása**

**do {**

**if( a == 15) {**

**// iteráció kihagyása**

**a = a + 1;**

**continue;**

**}**

**cout << "value of a: " << a << endl;**

**a = a + 1;**

**}**

**while( a < 20 );**

**return 0; }**

**A goto utasítás**

Feltétel nélküli ugrást biztosít a **goto**-tól egy címkézett utasításhoz ugyanabban a függvényben.

A goto utasítás használata nem ajánlatos, mert megnehezíti a program vezérlési folyamatának nyomon követését, így a program nehezen érthető és módosítható. Bármely program átírható, úgy, hogy ne kelljen használni a goto utasítást.

A címke egy azonosító, amely azonosítja a címkézett utasítást. A címkézett utasítás, minden olyan utasítás, amit egy azonosító előz meg, majd egy kettőspont követ.

#**include <iostream>**

**using namespace std;**

**int main () {**

**// helyi változó deklarálása:**

**int a = 10;**

**// do ciklus végrehajtása**

**LOOP:do {**

**if( a == 15) {**

**// iteráció kihagyása.**

**a = a + 1;**

**goto LOOP;**

**}**

**cout << "value of a: " << a << endl;**

**a = a + 1;**

**}**

**while( a < 20 );**

**return 0; }**

**Végtelen ciklus**

A ciklus végtelen ciklussá válik, ha a feltétel sosem válik hamissá. Többnyire a **for** ciklust használjuk erre a célra. Mivel a for ciklust alkotó három kifejezés közül egyik sem szükséges,

végtelen ciklust készíthetsz úgy, hogy a feltétel kifejezést üresen hagyod.

**#include <iostream>**

**using namespace std;**

**int main () {**

**for( ; ; ) {**

**cout << "Szeretem Anyukámat!\n";**

**}**

**return 0; }**

Ha a **feltétel** kifejezés hiányzik, akkor feltételezhető, hogy igaz. Lehet neked **init** és **increment** kifejezésed, de a C++ programozók legtöbben csak, így csinálnak végtelen ciklust **for( ; ; )**. A végtelen ciklust a **ctrl+c** gombok lenyomásával tudod **leállít**ani.

**Döntéshozó utasítás**

A döntéshozó struktúrák megkövetelik, hogy a programozó egy vagy több feltételt határozzon meg, amelyet a program értékelni vagy tesztelni kíván, valamint egy olyan utasítást vagy utasításokat, amelyet végre kell hajtani, ha a feltétel igaz.

**A C++ következő utasításokat biztosítja:**

1. **if utasítás**

Logikai kifejezésből áll, amit egy vagy több állítás követ.

1. **if else utasítás**

Az if utasítást követheti egy **else** utasítás, ami akkor hajtódik végre, ha a logikai kifejezés hamis.

1. **switch utasítás**

Lehetővé teszi, hogy egy változó az értékek listájával, egyenlőséggel tesztelhető legyen.

1. **nested if**

Használhatsz egy **if** vagy egy **else** **if** utasítást egy másik **if** vagy **else** **if** utasításban.

1. **nested switch**

Használhatsz egy switch-t egy másik switch-n belül.

**if**

Ha a logikai kifejezés értéke igaz, akkor az if utasításban a kódblokk végrehajtásra kerül. Ha a logikai érték hamis, akkor az első kódrészlet az utasítás vége után végrehajtódik.

**#include <iostream>**

**using namespace std;**

**int main () {**

**// local variable declaration:**

**int a = 10;**

**// check the boolean condition**

**if( a < 20 ) {**

**// if condition is true then print the following**

**cout << "a is less than 20;" << endl;**

**}**

**cout << "value of a is : " << a << endl;**

**return 0;**

**}**

**if else**

**#include <iostream>**

**using namespace std;**

**int main () {**

**int a = 100; // local variable declaration:**

**if( a < 20 ) { // check the boolean condition**

**cout << "a is less than 20;" << endl; // if condition is true then print the following**

**} else {**

**cout << "a is not less than 20;" << endl; // if condition is false then print the following**

**}**

**cout << "value of a is : " << a << endl;**

**return 0;**

**}**

**If… else if... else**

Az if utasítást követhet egy else if… else utasítás, ami nagyon hasznos a különböző feltételek teszteléséhez.

Amikor if, else if, else utasításokat használsz a következőt tartsd szem előtt:

* f
* f
* f

#include <iostream>

using namespace std;

int main () {

int a = 100;

if( a == 10 ) {

cout << "Value of a is 10" << endl;

} else if( a == 20 ) {

cout << "Value of a is 20" << endl;

} else if( a == 30 ) {

cout << "Value of a is 30" << endl;

} else {

cout << "Value of a is not matching" << endl;

}

cout << "Exact value of a is : " << a << endl;

return 0;

}

**Switch**

A következő szabályok vonatkoznak a switch utasításra:

* A kifejezésnek, amit a switch használ **integral** vagy **enumerated** típusúnak kell lennie, vagy olyan **osztálytípus**sal kell rendelkeznie, amiben az osztály egyetlen **korverziós függvényt** tartalmaz egy integral vagy enumerated típushoz.
* A switchen belül tetszőleges számú utasítás lehet. Minden case-t egy érték követ, amit egy kettőspont követ.
* A **case** konstans kifejezésének, ugyanolyan típusúnak kell lennie, mint a switch változója, és ennek **konstans**nak vagy **literál**nak kell lennie.
* A változó, amelyik **switchelve** lett megegyezik a case változójával, akkor az utasítások addig fognak végrehajtódni, amíg el nem érik a **break** utasítást.
* Amikor a break utasítást elérjük, a switch megszűnik és a vezérlő a switch utasítást követő következő sorra ugrik.
* Nem minden case-nek kell tartalmaznia break-et. Ha nincs break, a vezérlőa következő case-re megy,amíg nem ér el egy break-et.
* A switch utasításnak lehet egy alapértelmezett case-e, aminek meg kell jellenie a switch végén. Az alapértelmezett case használható egy feladat végrehajtására, ha az case-k egyike sem igaz. Az alapértelmezett case-ben nincs szükség szünetre.

**#include <iostream>**

**using namespace std;**

**int main () {**

**// local variable declaration:**

**char grade = 'D';**

**switch(grade) {**

**case 'A' :**

**cout << "Excellent!" << endl;**

**break;**

**case 'B' :**

**case 'C' :**

**cout << "Well done" << endl;**

**break;**

**case 'D' :**

**cout << "You passed" << endl;**

**break;**

**case 'F' :**

**cout << "Better try again" << endl;**

**break;**

**default :**

**cout << "Invalid grade" << endl;**

**}**

**cout << "Your grade is " << grade << endl;**

**return 0; }**

**Nested if**

**#include <iostream>**

**using namespace std;**

**int main () {**

**// local variable declaration:**

**int a = 100;**

**int b = 200;**

**// check the boolean condition**

**if( a == 100 ) {**

**// if condition is true then check the following**

**if( b == 200 ) {**

**// if condition is true then print the following**

**cout << "Value of a is 100 and b is 200" << endl;**

**}**

**}**

**cout << "Exact value of a is : " << a << endl;**

**cout << "Exact value of b is : " << b << endl;**

**return 0;**

**}**

**Nested Switch**

Lehetséges, hogy egy switch tartalmazzon egy belső switch-et. Ha a belső és a külső switch konstansai közös értékeket tartalmaznak, nem keletkezik konfliktus.

A C++-ban legalább 256 mélységi szintig lehet beágyazni a switch utasítást.

**#include <iostream>**

**using namespace std;**

**int main () {**

**// local variable declaration:**

**int a = 100;**

**int b = 200;**

**switch(a) {**

**case 100:**

**cout << "This is part of outer switch" << endl;**

**switch(b) {**

**case 200:**

**cout << "This is part of inner switch" << endl;**

**}**

**}**

**cout << "Exact value of a is : " << a << endl;**

**cout << "Exact value of b is : " << b << endl;**

**return 0;**

**}**

**Feltételes operátor**

Háromkomponensű operátor, mert 3 operandust igényel. Helyettesíti az if else utasítást. Például ebből a kódból:

**if(y < 10) {**

**var = 30;**

**} else {**

**var = 40;**

**}**

Ez az egyszerűbb,rövidebb kód lesz, ugyanazzal a kiementtel:

**var = (y < 10) ? 30 : 40;**

**#include <iostream>**

**using namespace std;**

**int main () {**

**// Local variable declaration:**

**int x, y = 10;**

**x = (y < 10) ? 30 : 40;**

**cout << "value of x: " << x << endl;**

**return 0;**

**}**