|  |  |
| --- | --- |
| **#include <iostream>**  **using namespace std;**  **int main()**  **{**  **cout << "Hello world!" << endl;**  **return 0;**  **}**  Hibák felismerése | **#include <iostream>**  **using namespace std;**  **int main()**  **{**  ***/\** Ez egy C stílusú megjegyzés**  **ami addig tart, *amig* a fordító**  **meg nem találja a záró csillag-perjel kettőst \* *I***  **cout << "Hello world!" << endl;**  ***//* Ez egy c*++* stílusú megjegyzés, ami a sor végéig tart**  **cout << ”itt a megjegyzés vége!";**  **return 0;**  **}** |

Megjegyzések a programban: a megjegyzés javítja az olvashatóságot, különösen, ha nem beszédes változóneveket használunk.

C++ stílusú megjegyzések a sor elején // jelekkel kezdődnek és 1 sor hosszúak

C stílusú megjegyzések /\* -al kezdődnek és a \*/ -ig tartanak, azaz több sorosak is lehetnek.

|  |  |
| --- | --- |
| **#include <iostream>**  **using namespace std;**  ***II* Függvényhívás bemutatása**  ***II Ü*zenet megjelenítése a képernyőn**  **void DemonstrationFunction()**  **{**  **cout << "A DemonatrationFunction függvényben vagyunk\n ";**  **}**  ***II* A main() függvény - kiír egy üzenetet , majd**  ***II* meghivja a DemonstrationFunction nevű függvényt , végül**  ***II* pedig kiír egy második üzenetet** | **int main()**  **{**  **cout << "A Main függvényben vagyunk\n";**  **DemonstrationFunction() ;**  **cout << "Ismét a main-ben vagyunk\n";**  **return 0;**  **}** |

Egy függvény vagy egyetlen adattal tér vissza, vagy visszatérési értéke void, ami azt jelenti, hogy nem ad vissza semmit a hívónak. Ha például írunk egy függvényt, ami két egész számot tud összeadni, akkor az összeget nyilván célszerű a visszatérési értékben elhelyezni. Ilyenkor tehát a függvényünk egész (int) típusú visszatérési értékkel rendelkezik.

Ha ugyanakkor van egy olyan függvényünk, ami csupán egy üzenetet ír ki , de amúgy nem csinál semmi említésre méltót, akkor nyilván nincs mit visszaadni a hívó félnek. A függvény visszatérési értéke tehát ilyen esetben célszerűen void.

A fejléc:

int Sum (int a, int b)

|  |  |
| --- | --- |
| **#include < iostream>**  **using namespace std;**  **int Add (int x, int y )**  **{ cout << "Itt az Add ( ) függvény, a következő értékeket kaptam " << x << " és " << y <<" \n " ;**  **return (x+y) ;**  **}** | **int main()**  **{ cout << "A main() függvényben vagyunk \ n " ; cout << " \nMeghivjuk az Add() függvényt\n';**  **cout << "A visszaadott érték : " <<Add (3 , 1) ;**  **cout << " \nIsmét a main() függvényben vagyunk. \n" ;**  **cout << "\nKilépünk ... \n\n" ;**  **return 0;**  **}** |

A változó egy terület a számítógép memóriájában. Minden változónak van neve. A nevek az angol abc betűiből, számokból és az aláhúzás jelből állhatnak. Kötelezően betűvel kell kezdődniük.

A változóknak meg kell mondani a típusát. Ez dönti el, hogy a fordító mennyi helyet foglaljon a memóriában.

Típus Hely alsó felső

unsigned short int 2 bájt 0 65535

short int 2 bájt -32768 32767

unsigned longint 4 bájt *0 4294967295*

long int 4 bájt -2147483648 2147483647

Int 4 bájt -2147483648 2147483647

unsigned int 4 bájt *0 4294967295*

char 1 bájt 256 féle karakter

bool 1 bájt igaz vagy hamis

float 4 bájt -1.2e-38 3.4e+38

double 8 bájt 2 .2e-308 1.8e:308

**Deklaráció:**

int Eletkor; unsigned short x; unsigned short y; unsigned short x,y;

A kis és nagybetűk különbözőek! Foglalt kulcsszavakat ne használjunk változónévnek!

**Kezdőérték hozzárendelése változóhoz:**

int hossz, szelesseg;

hossz=5; szelesseg=3;

int hossz=5, szelesseg=3;

A változók értéke a deklarását követően határozatlan!!! Minden változónak először értékadás bal oldalán kell szerepelnie (vagy azzal egyenértékű utasításban).

Túlcsordulások!!!

C típusú állandók bevezetése:

**#define szelesseg 1024**

**#define magasság 768**

A programban mindenhol használható a **szelesseg** és **magassag** konstans. Ilyenkor egyszerűen helyettesítődik az 1024 és 768 értékkel. Értékadás bal oldalán nem állhat!!!

C++ típusú állandók:

const típus név=érték alakú

Pl.: **const int szelesseg=1024;**

**const int magassag=768;**

Értékadás:

Alakja: **változó = kifejezés;**

Pl.: x = a \* b

Akár többszörös értékadás is elképzelhető : x = y = a + b;

Kifejezés: konstansok, változók operátorokkal összefűzve. Legegyszerűbb egyetlen konstans : 28, vagy egyetlen változó z

Pl.: X = 28; Y = Z;

Operátorok: Összeadás (+), Kivonás (-), Szorzás (\*), Osztás (/), Maradék képzés (%)

Az osztás egész jellegű változókra egész osztás (24/5 = 4). A maradékképzés is csak egész jellegű változókra értelmes (23%5 = 3) Nagyon gyakori az önhozzárendelés. Pl.: **a = a + 2** Ilyenkor használhatjuk az **a += 2** kifejezést is. Szintén gyakori, hogy valamit eggyel növelni vagy csökkenteni kell. **a = a + 1** Ezt megtehetjük az a++ kifejezéssel (a--). Vagy akár ++a (--a) módon.

**Ha kifejezésben szerepel, akkor nem mindegy!!!**

**Pl.: a = 5; b = a ++; c = ++ a**

Itt b értéke 5 lesz, majd **a** hatra nő, c értéke 7 lesz, mert előbb növekszik **a** és utána rendelődik a c-hez.

Relációs operátorok:

Egyenlő (==), Nem egyenlő (!=), Kisebb (<), Nagyobb (>), Kisebb egyenlő (<=), Nagyobb egyenlő (>=)

A relációs operátorok által alkotott kifejezések igaz vagy hamis értéket adnak vissza.

Pl.: a < b; a != b; a == b

Logikai műveletek: És (&&), Vagy (||), Nem (!)

A logikai hamis érték a 0, minden nem nulla érték igaznak minősül

A műveletek erő sorrendje:

1 ( ) balról jobbra

2 [ ] ++ -- balról jobbra

3 - + ! (UNÁRIS) jobbról balra

4 \* / % balról jobbra

5 + - balról jobbra

6 << >> balról jobbra

7 < <= > >= balról jobbra

8 == != balról jobbra

9 & balról jobbra

10 ^ balról jobbra

11 | balról jobbra

12 && balról jobbra

13 || balról jobbra

14 = \*= /= + = -= %= jobbról balra

Az IF utasítás

**if (kifejezés)**

**utasítás1;**

**else**

**utasítás2;**

**utasítás3;**

A működése: a program kiértékeli a kifejezést, ha az értéke igaz, akkor az utasítás1-t hajtja végre, egyébként az **else** utáni utasítás2-t. A program az utasítás3-al folytatódik.

Az else ág elhagyható, ilyenkor ha a feltétel hamis, akkor utasítás3 kerül végrehajtásra (mellesleg ez akkor is végrehajtásra kerül, ha a feltétel igaz).

Az IF utasítás minden ágában csak egyetlen utasítás állhat!!! Ha több mindent kell végrehajtani, akkor ÖSSZETETT UTASÍTÁST kell alkalmazni!

**Összetett utasítás: {utasítás1; utasítás2; … ; utasításn;}**

if (a<5)

{utasítás1; utasítás2; …}

Általános szabály, hogy minden ELSE a hozzá legközelebbi olyan IF-hez tartozik, amelynek még nincs ELSE ága.

Beolvasni a **cin** utasítással lehet.

Pl.: int a,b;

cin >> a;

cin >> a >> b;

Vigyázat semmi nem utal arra, hogy adatot vár a program!!!

Helyesen cout <<"Kérem az adatot "; cin >> a;

cout <<"Kérem az adatokat szóközzel elválasztva "; cin >> a >> b;

Kezdetben feltételezzük, hogy mindig a jó adatot adjuk meg. A példában számokat szóközzel elválasztva. Valójában elég macerás a beolvasás, mert a felhasználó rossz adatot fog megadni, amit még külön ellenőrizni kell.

**Olvass be egy számot és írd ki a paritását!**

**Olvass be egy évszámot és mond meg, hogy szökőév-e!**

**Olvass be egy karaktert és mond meg, hogy a hét melyik napja kezdődik ezzel!**

A SWITCH utasítás

Láttuk, hogy az alábbi konstrukció nagyon macerás…

|  |  |
| --- | --- |
| **if (kifejezés)**  **utasítás1;**  **else if (kifejezés)**  **utasítás2;**  **else if (kifejezés)**  **utasítás3;** | Helyette a switch utasítás:  **switch (kifejezés)**  **{**  **case érték1: utasítás1; break;**  **case érték2: utasítás2; break;**  **case érték3: utasítás3; break;**  **default: utasítás4;**  **}** |

**A WHILE ciklus**

while (feltétel) utasítás;

Az utasítás addig ismétlődik, amíg a feltétel igaz. (Lehet, hogy 0-szor) Egyetlen utasítás szerepelhet. Általában egynél több utasítás van, ezért használjunk összetett utasítást.

Pl.:

int szam = 0;

while (szam<5) {

szam++;

cout << szam <<endl;}

cout <<"A vegen a szam erteke: " << szam;

Általában a WHILE alkalmazandó, ha nem tudjuk pontosan, hogy hányszor hajtódik végre. Ez egy rossz példa, mert most tudjuk és nem is ezt kéne alkalmazni.

**A break és a continue utasítás**

A break utasítás mindig valamilyen ciklushoz kötődik. Hatására a program kilép a ciklusból és a ciklust követő utasításra kerül.

A continue hatására a ciklusban a continue utáni utasítások kimaradnak és azonnal a ciklusfej kerül kiértékelésre.

**Helyes algoritmusszervezéssel ezek az utasítások elkerülhetők!!!**

|  |  |
| --- | --- |
| while (feltétel1) {  utasítás1  if (feltétel2) continue;  if (feltétel3) break;  utasítás2;  } | while (feltétel1) {  utasítás1;  if (!feltétel2) {  if (!feltétel3)  utasítás2;  else a feltétel1 legyen hamis  }  } |

A számokat véletlen számoknak nevezzük, ha eloszlásuk egyenletes, azaz az intervallum minden értékének előfordulása ugyan olyan valószínűségű. A c++-ban két függvény szolgál a véletlen számok kezelésére. A használatukhoz a program elején szerepelnie kell a következőknek:

#include <stdlib.h>

#include <time.h>

A kezdőérték beállítására szolgál a **srand(unsigned int érték)** függvény.

Ha kezdőértéknek valami számot állítasz be, akkor mindig ugyan azt a számsorozatot fogja generálni!!! Teszteléshez jól jön!

A tényleges véletlen sorozathoz használd a **srand(time(NULL))** utasítást!

Ilyenkor a kezdőértéket a beépített órából veszi.

Létezik egy előre definiált konstans, ami megmutatja a legnagyobb generálható értéket:

**RAND\_MAX**

A generálás a **rand()** függvénnyel történik, ami egy **0..RAND\_MAX** közti számot generál.

Az **1..a** intervallumban **rand() % a +1** módon kaphatsz értéket,

Az **a..b** intervallumban **rand() % (b-a+1)+a**

**Generálj kockadobást!**

**Generálj pontosan kétjegyű számokat!**

**A francia kártya lapjai 2..13-ig vannak számozva (a 10 felett J, Q, K, A). Generálj egy kártyalapot és írd ki!**

A tömb azonos típusú objektumok összessége.

Deklaráció: **típus név [elemek száma1] [elemek száma2] …**

Pl.: **long szamok [10];**

**short tabla[8] [8];**

**int elet[2] [100] [100];**

Vigyázat!!! Az elemek számozása 0-tól indul!!! A **szamok** tömb 10 elemű, de az indexei 0..9!!!

Hivatkozni az indexeléssel lehet. A **szamok** tömb első eleme **szamok [0]** utolsó eleme **szamok[9]**.

Kezdőérték is rendelhető a tömbhöz. Az elemeket { } között vesszővel elválasztva soroljuk fel.

long szamok [5] = { 1, 2, 3, 4, 5 }

Ha kezdőértéket is adunk, akkor az elemek száma a deklarációban elhagyható!!!

long szamok [] = { 1, 2, 3, 4, 5 }

A fordító akkora tömböt hoz létre, hogy a megadott elemek elférjenek!!!

Ha a tömb határán túl hivatkozunk, akkor komoly (és nehezen felderíthető) hibákat kapunk. Klasszikus hiba, hogy a **szamok [5]** tömb 5. elemére hivatkozunk (**szamok[5]**), pedig az utolsó elem **szamok[4]**.

Több dimenziós tömbök kezdőértéke:

short matrix [5] [3] = { {11,12,13}

{21,22,23}

{31,32,33}

{41,42,43}

{51,52,53} }

**A DO WHILE utasítás**

do utasítás; while (feltétel);

Az utasítás egyszer biztosan végrehajtódik. A feltétel kiértékelése a végén történik. A ciklus akkor ismétlődik, ha a feltétel igaz. Gondoskodnunk kell a ciklusban arról, hogy a feltétel hamissá váljon.

**Olvass be pontosan kétjegyű számokat, a bevitelt ellenőrizd!!! A végjel a 99 legyen!**

**A FOR ciklus**

for ( utasítás11; utasítás12, … feltétel; utasítás21 utasítás22;…) utasítás;

A működése: elvégzi az utasítás11-et, utasítás12-t, stb. Ezt nevezzük inicializációs résznek. Magyarul a kezdőértékek beállításának. Majd ellenőrzi a feltételt, ha a feltétel igaz, akkor végrehajtja az utasítást. Ezt nevezzük a ciklus magjának. Ha több utasítást akarunk, akkor használjunk összetett utasítást. A mag végrehajtása után elvégzi az utasítás21-t, utasítás22-t, stb. Ez a léptetés. Ezután ismét a feltétel kiértékelése következik, ha igaz, akkor ismét a ciklusmagban lévő utasítás…

Pl.: a legjellemzőbb…

for ( i = 0; i < 10; i++)

cout <<i <<endl;

**Írd ki a számokat 1-10-ig!**

A képernyőkezelés a WINDOWS-on keresztül történik.

#include <windows.h > szükséges

void gotoxy( int x, int y )

{

COORD p = { x, y };

SetConsoleCursorPosition( GetStdHandle( STD\_OUTPUT\_HANDLE ), p );

}

Lényegében a beépített windows eljárások megfelelő hívásával pozícionálhatunk a képernyőn.

A képernyő 80 karakter széles, és alapértelmezésben 25 karakter magas. A bal felső sarok az 0,0 pozíció. A gotoxy(3, 5); függvényhívás a kurzort a 3, 5 pozícióba viszi, a következő kiírás ezen a helyen kezdődik.