# 5. előadás

## 22-én szünet?

27-én le kell dolgozni? ELTE általában nem kéri a ledolgozást.

## Összetettebb logikai feladatok

**if(**felt1**&&**felt2**)**

Mikor mi értékelődik ki?

Az sem biztos, hogy mindig a várt sorrendben fog kiértékelődni.

### Szekvencia​pont

Garantálja, hogy a megelőző részkifejezések kiértékelődtek.

Mikor fontos ez:

bool f**(){**

 std**::**cout**<<**'f'**;**

 **return** **false;**

**}**

bool g**(){**

 std**::**cout**<<**'g'**;**

 **return** **true;**

**}**

bool h**(){**

 std**::**cout**<<**'h'**;**

 **return** **false;**

**}**

A három alapfüggvény. Ez a feltétel:

**if(**f**()==**g**()==**h**()){**

 std**::**cout**<<**'a'**;**

**}else{**

 std**::**cout**<<**'b'**;**

**}**

Mi íródik ki? \_ \_ \_

Miért? Mert az vagyis kifejezés eredménye hamis lesz. Ezt hasonlítjuk eredményével, akkor igazat kapunk. Mivel igaz, az a betű fog kiíródni.

Az első három kiírt karakter sorrendje 6 féleképpen lehetséges.

## Szekvencia​pontok

### Legismertebb a pontosvessző.

### Van még egy, a vessző operátor.

int x**;**

**while(**std**::**cin**>>**x**,** x**){**

 //...

**}**

Kiértékeli a bal oldali függvény eredményét, majd a jobb oldalt visszaadja.

Meddig fut az a ciklus? Amíg (a felhasználó bemenete) nem .

A vessző garantálja, hogy először lesz a beolvasás, utána a feltétel.

Ezt a szabadságot könnyű elrontani. De a fordító nem tud optimalizációt végezni.

### Rossz példa

int gcd**(**int a**,** int b**);** //lnko

int x**;**

int y**=**gcd**(**x**++,**x**++)**

Nem mindegy, milyen sorrendben történik az x++. (Mondjuk itt pont az. Ezért rossz.)

int i**=**x**++;**

int j**=**x**++;**

gcd**(**i**,**j**);** //Ekkor fix a sorrend.

gcd**(**x**,**y**);** //Jó.

gcd**((**x**,**y**));** //Itt nem szeparátor a vessző, hanem megint szekvenciapont.

gcd**((**y**,**x**),(**x**,**y**));** //Így már jó, de túlbonyolítottuk. Fölöslegesen kiértékeli x-et és y-t még egyszer.

A második egy egyparaméteres -t akarna meghívni, y értékével.

### Jó példa

int x**[]**//...

**for** int i**=**0**;**i**<**N**;++**i**){**

 x**[**i**]=**i**++;**

**}**

Mi történik meg előbb? meghatározása vagy kiszámolása? Nincs definiált sorrend.

# Deklarációk és definíciók

Csapatmunka esetén konfliktusok (névütközések) elkerülése fontos. Modulokra bonthatóság, kisebb részeket kelljen csak újrafordítani hiba esetén.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

Ha megváltoztattuk pl. -t, akkor csak azt kell újrafordítani. (És a többit csak újra kell linkelni.)

Ha nincs meg a forráskódja egy modulnak, akkor is működik a fordítás. (Ha könyvtárként vagy object fájlként megvan.)

## Függvények

Van egy fordítási egység:

Ha ebben szeretnénk használni a függvényt, akkor el kell mondanunk a fordítónak függvénydeklarációval:

int gcd**(**int**,**int**);**

int gcd**(**"abc"**,**"def"**);** //Fordítási hiba. Azért, mert tudja a fordító, hogy int kell, nem string. Ha nem lenne előzetesen deklarálva, akkor nem fordulna le.

Régen C-ben nem kellett deklarálni a függvényeket, minden ismeretlenre -et várt.

Meg van írva a függvény a fájlban.

int gcd**(**int a**,** int b**){**

 //...

**}**

Ez a függvény definíciója. Ha a függvénytörzs nincs megírva sehol: "Undefined reference"

Ehhez általában tartozik , amiben minden függvény szignatúrája benne van:

int gcd**(**int**,**int**);**

Így nem nekünk kell beirkálni mindent, hanem elég egy #include "numeric.h".

### One definition rule

Egy függvényhez nem tartozhat kétféle definíció.

Hibaüzenet: "Ambiguous reference" Linkelési hiba.

## Globális változók

Nem csak függvények esetében létezik a fent leírt megosztás.

A program futása során végig a memóriában van.

Ezeket is meg lehessen osztani.

a**.**cpp**{**

 int a**;**//Globális változó **definíció**ja

**}**b**.**cpp**{**

 //Itt nem lehet "int a"-t mondani, mert többször lenne definiálva

 extern int a**;**//extern = Másik fordítási egység fogja definiálni.

**}**c**.**cpp**{**

 extern int a**;**//Akárhányszor lehet extern kulcsszót használni, vagyis deklarálni. Csak definiálni nem.

**}**

Globális változót használni nem jó. Több fájlban megosztottat még inkább nem jó.

## Osztályok

Az osztály olyan entitás, ami adatokat és azokkal együttműködő műveleteket ír le.

class Complex**{**

 double re**,**im**;** //Reprezentáció

public**:**

 double abs**();**

**};**

Mivel reprezentáció van benne, ezért ez osztály definíció

Mégis header fájlba kell, kerüljön. Miért?

Java-ban példányosított osztályok lokális változóként is speciális memóriában, a heap-ben jönnek létre. A **new** operátorral hozunk létre mindent. Minden változó referencia a létrehozott objektumra.

C++-ban a lokális változók között, a stack memórián jön létre. Complex c**;**

### Stack

|  |
| --- |
|  |
| ez egy stack frame |
|  |

Stack = "aktivációs rekord"

Ezen belül van:

* SP stack pointer
* BP bázis pointer
* lokális változók

Az aktívan futó függvény frame-je van a legtetején.

Emiatt a fordítónak tudnia kell, hány bájtos minden változó az osztályban.

Ha futásidőben változó adatokra van szükségünk, azt nekünk kell létrehozni () és felszabadítani () a heap-ben.

### Osztálydeklaráció

class Complex**;**

Ez nem elég. A fordítónak tudnia kell a definíciót is. Hogy tudjon két változót lefoglalni a stack-en.

Pointert (Complex**\*** p**;**) lehet létrehozni definíció nélkül is.

Java-ban költségesebb, mert nem tud stack-en, optimalizáltan dolgozni. Minden megy a heap-be.

## Pointerek

**{**/\*lokális blokkon belül vagyunk\*/ static int a**;}**

: Statikus lokális változó: túl fogja élni ezt a függvényhívást. A statikus tárterületen jön létre. Tárolási osztály.
: Típus
: Azonosító

int **\***a**,**b**;**//a pointer, b integer lesz

A csillag az azonosítóhoz, nem a típushoz tartozik.

A szóköz helye nem befolyásolja. int**\*** a**,**b**;** ugyan az. Mindenki úgy írja, ahogy neki tetszik.

int**\*** x**[**40**];**

Ez egy 40 elemű, pointereket tartalmazó tömb.

Hogyan írjunk 40 elemű integer tömbre mutató pointert? Összezárójelezzük a csillagot az x-szel.

int **(\***x**)[**40**];**

Ez mi:
int **\***p**(**int**);**
Deklaráció. Mit deklarál? Egy pointert visszaadó függvényt. Neve , -et vár paraméterként, pointert ad vissza. Törzsét majd valaki más deklarálja.

És ez mi?
int **(\***q**)(**int**);**
Ez nem függvény. Ez egy függvényre mutató pointer. Olyan függvényre mutat, ami -et vár és -et fog visszaadni. Mivel ez pointer, futásidőben változhat, mire mutat. C-ben ez volt az egyetlen épeszű felhasználói kódrészletet átadó megoldás. C++-ban már van jobb.

## A "Minden, ami deklarációnak minősül, az egyben deklaráció is" szabály furcsa mellékhatásai

class X**{**

public**:**

 x**(**int i**){**/\*...\*/**}**

 //...

**};**

//...

double r**=**6.46**;**

X x**(**int**(**r**));**//Függvénydeklaráció.

Miért? Ez egy kis függvény, -et vár paraméterként, nagy -et ad vissza. Nem a konstruktor meghívása.

Erre van kapcsos zárójeles konstruktorhívás.