# A C++ programozási nyelv 2.
# FELADATMENTES

2019.10.28.

Koschek Vilmos

1

## Miről lesz szó?

1. Bővítések
2. Osztályok
   1. Osztályok használata
   2. Kivételek
   3. Osztályok és dinamikus memóriakezelése
   4. Az osztályok további tulajdonságai
   5. Öröklődés és többrétűség
   6. Operátorok átdefiniálása
   7. Névterek
3. Sablonok



2

# 1. rész

## Bővítések

3

## Bemeneti és kimenti stream használata

- cin
- cout
- cerr

A fentiek nem része a C++ nyelvnek, de definiálva vannak a stream könyvtárban (+iostream).



4

## cout: standard output stream

**HELLO.C**

```
#include <stdio.h>

int main()
{
    printf("Hello, world \n");
    return(0);
}
```

**HELLO.CPP**

```
#include <iostream>

int main(void)
{
    cout <<"Hello, world\n";
    return(0);
}
```

**A << operátor jelentése függ a környezettől, amiben használjuk !**

***MS VS 2008-tól!***
***#pragma warning(disable : 4996)***
***using namespace std;***
***iostram.h -> iostream !!!***



5

## cout

Formátum string nélkül

C

```
printf( "%d" , i );
```

CPP

```
cout<<i;
```

Típusok összekapcsolása

```
#include <iostream>

int main(void)
{
        int i = 123;
        cout << „A szám értéke" << i << '.';
        return(0);
}
```



6

## cout

Formázott kimenet

```
#include <iostream>

int main(void)
{
        int i = 123;
        cout    << dec << i << ' '
                << oct << i << ' '
                << hex << i;
        return(0);
}
```

A string + decimális + karakter típusok összekapcsolása a << operátorral !



7

## cin: standard input stream

Szám bekérése

```
#include <iostream>

int main(void)
{
        int i;
        cout << "Egy számot kérek ...\n";
        cin >> i;
        cout << "A begépelt szám : " << i;
        return(0);
}
```

& ?



8

## cin

Karakter tömb…

```
#include <iostream>

int main(void)
{
        char nev[5];

        cout << "Egy nevet kérek ...\n";
        cin >> nev;
        cout << "A begépelt név : " << nev;

        return(0);
}
```

Hossz? (Release!)

```
char s2[6] = "bbbbb";
char nev[5] = "0123";
char s1[6] = "aaaaa";
```

Memória túlírás, ékezet –> kód!



9

## Függvények prototípusa

Minden függvényt deklarálni **KELL** (C-ben nem kötelező !), azaz meg kell adni a függvény prototípusát (kézjegy).

Prototípus

- függvény neve
- visszatérési érték
- paraméterek típusa
- paraméterek száma



10

## Prototípus

```
#include <iostream>

void display( char * s );                    //...prototípus


main()
{
        display("Hello, world");
        return(0);
}

void display( char * s )
{
        cout << s;
}
```



11

## Alapértelmezett függvény argumentum

A prototípusban szereplő paraméterek kezdeti értéket kaphatnak.
Ha hívásnál valamelyik (!!!) paraméter elmarad, a fordító automatikusan ezekkel az értékekkel hívja meg a függvényt.

```
void f( int i=5, double d = 1.23 );
```

Lehetséges függvényhívások:

```
f( 12, 3.45 );          //...felülírja az alapértelmezett értékeket
f( 3 )                  //...f( 3, 1.23 )
f();                    //...f( 5, 1.23 );
```

**HIBA, hibás hívás:** f( , 3.5 );



12

## Példa

```
#include <iostream>

void show( int elso=1, float masodik=2.3, long harmadik=4 );

void main(void)
{
    show();
    show( 5 );
    show( 6, 7.8 );
    show( 9, 10.11, 12L );
}

void show( int elso, float masodik, long harmadik )
{
    cout << "\nElső = "  << elso;
    cout << ", Második = "      << masodik;
    cout << ", Harmadik = "  << harmadik;
}
```



13

## A változó definíciók helye

Változót bárhol lehet definiálni a kódban.

```
#include <iostream>

main()
{
    cout << "Egy számot kérek ...\n";
    int i;
    cin >> i;
    cout << "A begépelt szám : " << i;
    return(0);
}
```

**Nem megengedett** : if (int k==2)



14

## Scope

```
#include <iostream>

int main(void)
{
    for( int i = 0; i < 3; i++ )
    {
        int k=22;
        cout << "\ni= " << i  << " k= " << k;
    }
    //...i a teljes blokkban ! MSVS 2005-től állítható!

    if( i == 3 )
        cout << "\ni= 3";
    return (0);
}
```

A fenti példában a k változó csak a for blokk-ban él , még az i változó a definiálástól a main() végéig.



15

## Scope operátor

C-ben az azonos nevű lokális és globális változóknál a lokálisnak van nagyobb precedenciája.

C++ -> (::)

```
#include <iostream>

int i=123;

main()
{
    int i=456;
    cout << ::i;          //...külső
    cout << '\n';
    cout << i;            //...belső
    return(0);
}
```



16

## Példa a Scope operátorra

```
void CWnd::GetWindowText(CString& rString) const
{
   ASSERT(::IsWindow(m_hWnd));
.
.
   ::GetWindowText(     m_hWnd,
                  rString.GetBufferSetLength(nLen),
                        nLen+1);

.
}
```

GetBufferSetLength: Returns a pointer to the internal character buffer



17

## Inline függvények

**Az inline kulcsszó alkalmazásakor a fordító (ha lehetséges) a függvényt bemásolja a hívás helyére.**

**inline int max( int a, int b );**

**Előnye : gyorsabb futás**
**Hátránya: nagyobb kódméret (??)**

**Előnyei a makróval szemben:**

**paraméterek ellenőrzése fordításnál**
**un. "Side-effect" elkerülése**



18

## A bool típus

Logikai igaz / hamis érték

Olvashatóbb kód

Automatikus konverzió int – bool, 0/1 !!!!

```
int i = 3;
bool betoltve = true;

betoltve = false;

betoltve = i;       // warning!
```



19

## Felsorolt típus

**Az enum kulcsszó alkalmazásával egy új adattípust lehet létrehozni.**

```
//...értékek:  0,    1,        2,     3,      4
enum szin { piros, narancs, sarga, zold,  kek };
.
.
szin szinek;              //...C-ben :  enum szin szinek;
szinek = kek;
.
.
int i;
i = szinek;               //...i=4
szinek =  4;              //...HIBA: int->szin konverzió
szinek = (szin)4;         //...szin = 4 ( kek)
```



20

## Függvénynevek átdefiniálása (overloaded functions)

**Feladat**: olyan függvény (függvények) készítése, amely kiírja az időt, de a time_t típust és a tm struktúrát is tudja kezelni bemenő paraméterként.

```
//MSVS2010 time.h,.
typedef long __time32_t;   /* 32-bit time value */
typedef __time32_t time_t; /* time value, 19700101 */

struct tm {
      int tm_sec;          /* seconds after the minute - [0,59] */
      int tm_min;          /* minutes after the hour - [0,59] */
      int tm_hour;         /* hours since midnight - [0,23] */
      int tm_mday;         /* day of the month - [1,31] */
      int tm_mon;          /* months since January - [0,11] */
      int tm_year;         /* years since 1900 */
      int tm_wday;         /* days since Sunday - [0,6] */
      int tm_yday;         /* days since January 1 - [0,365] */
      int tm_isdst;        /* daylight savings time flag */
      };
```



21

## Megoldás ?

- egy függvény, de +1 paraméter

```
//...type:0=time_t, 1= struct tm
void display_time( void * tim,  int type);
```

- két különböző függvény

```
void diplay_time0(time_t  * tim);
void diplay_time1(struct tm * tim);
```



22

## És C++, függvénynév átdefiniálás

```
#include <iostream>
#include <time.h>

void display_time( const struct tm * tim )
{    cout << "1.  dátum,idő: " << asctime( tim ) << '\n';}

void display_time( const time_t * tim )
{    cout << "2.  dátum,idő: " << ctime( tim ) << '\n';}

void main()
{    time_t tim = time(NULL);
     struct tm * ltim = localtime( &tim );
     display_time( ltim );
     display_time( &tim );
}
```

Paraméterek típusából dönti el a fordító, hogy melyik függvényt kell hívni!



23

## És még valami…

```
#include <iostream>
#include <string.h>

inline void string_copy( char * dest, const char * src )
{    strcpy( dest, src );}

inline void string_copy( char * dest, const char * src, int len )
{    strncpy( dest, src, len );}

 void main()
{
     static char stringa[20], stringb[20];
     string_copy( stringa, "Alma" );
     string_copy( stringb, "Eper egy finom gyümölcs.", 4 );
     cout<< stringb << " és " << stringa;
}
```

Paraméterek számából dönti el a fordító, hogy melyik függvényt kell hívni!



24

## Néhány apróság…

**HIBA**, ha a paramétereik és a nevük megegyezik, de a visszatérési értékük más:

```
int search( char * kulcs );
char *search(char * nev);
```

HIBA, ha a prototípusok nem egyértelműek:

```
string_copy(char * dest, const char * src, int len=10);
                .
                .
                .
string_copy( stringa, "Alma" );    //...?
```



25

## C függvény hívása C++ -ból

CPP-ben

```
int CPPfuggveny(int i)
{
    return(i*i);
}
```

```
00029e8 00 00 00 00 call ?CPPfuggveny@@YAHH@Z; CPPfuggveny
```

C-ben

```
int Cfuggveny(int i)
{
   return (i*i);
}
```

```
00038e8 00 00 00 00 call _Cfuggveny
```



26

## Akkor mi a probléma?

CPP-ben

```
int CPPfuggveny(int i);
int Cfuggveny(int i);

int main(void)
{
        int i= 3;

        i = CPPfuggveny(i);
        i = Cfuggveny(i);

        return(0);
}
```



27

## Hivatkozási típus

A hivatkozott objektum definíciója:

```
int  i;
int   & hivatkozas_i = i;
```

Alias típusnak is nevezik, mivel ezzel alternatív elnevezés jön létre ahhoz az objektumhoz, amellyel inicializáltuk.

Valamennyi művelet , amelyet a hivatkozott objektummal végrehajtunk , arra az objektumra van hatással, amelyre az adott alias nevet létrehoztuk.



28

## Példa

```
#include <iostream>

main()
{
    int i=10;
    int & hivatkozas_i = i;

    cout << '\n' << &i << '\t' << &hivatkozas_i;

    hivatkozas_i++;

    cout << '\n' <<i;

    return(0);
}
```

Cím?



29

## Inicializálás

**MINDIG INICIALIZÁLNI KELL HASZNÁLAT ELŐTT !**

Kivételek:

ha deklarálva van extern-nel, mert ebben az esetben valahol máshol inicializálják

osztály tagváltozójánál, mert ilyenkor a konstruktorban inicializálják

ha paraméternek van deklarálva egy függvény definíciónál vagy deklarációnál, mert itt a függvény hívásánál kap értéket

ha visszatérési értéknek van deklarálva, mert ilyenkor a függvény visszatérésénél kap értéket



30

## Hivatkozási típus és a pointerek kapcsolata

```
int i = 10;
int & hivatkozas_i = i;
int * const mutato_i  =  &i;
```

hivatkozas_i  = = mutato_i

különbségek:
- a hivatkozásnál nem kell a * operátor
- címükhöz nem lehet hozzáférni (előző példa)

A const jelentése a mutato_i-nel:
mutato_i tartalmát nem lehet megváltoztatni !



31

## Mint függvény paraméter

Paraméterek átadásának lehetőségei

- érték szerinti (C,C++)
- cím szerinti (C,C++)
- hivatkozási típus segítségével (C++)



32

## Akkor hogyan is?

```
#include <iostream>

struct nagystruktura
{
    int    egesz;
    char szoveg[1000];
}   ns = {123, "Ez egy nagy struktúra"};

void val(nagystruktura v);

void ptr(nagystruktura * p);

void ref(nagystruktura & r);
```



33

## Folytatás…

```
int main(void)
{
    val( ns );
    cout<<'\n'<<ns.egesz;
    ptr( &ns );
    cout<<'\n'<<ns.egesz;
    ref( ns );
    cout<<'\n'<<ns.egesz;
    return(0);
}
 void val(nagystruktura v)
{   cout<<'\n'<<v.egesz++;    cout<<'\n'<<v.szoveg; }

void ptr(nagystruktura * p)
{   cout<<'\n'<<p->egesz++;  cout<<'\n'<<p->szoveg; }

void ref(nagystruktura & r)
{   cout<<'\n'<<r.egesz++;    cout<<'\n'<<r.szoveg; }
```



34

## Konklúzió

```
void val(nagystruktura v);

void ptr(nagystruktura * p);

void ref(const nagystruktura & r);
```

- mutató , ha a függvény módosítja a  paramétert
- hivatkozási típus (const !), ha a függvény nem módosítja a  paramétert



35

## Mint függvény visszatérési érték

```
int main(void)
{
    int i = 0;
    int & ref = i;
    i = f();
    f() = 5;
    return(0);
}
int ext_i = 10;

int & f(void)
{
    return(ext_i);
}
```

Hogyan is?



36

## És mi van mögötte?

```
....
PUBLIC          ?ext_i@@3HA             ; ext_i
_DATA SEGMENT
    ?ext_i@@3HA DD  0aH                 ; ext_i
_DATA ENDS

....
_ref$ = -24                             ; size = 4
_i$ = -12                               ; size = 4
....
```



37

## És mi van mögötte?

```
...
?f@@YAAAHXZ PROC                        ; f
; 20  : {
....
; 21  :  return(ext_i);
                    mov     eax, OFFSET ?ext_i@@3HA     ; ext_i
; 22  : }
```



38

## És mi van mögötte?

```
_main   PROC
; 6   : {
....
; 7   :  int i = 0;
                    mov     DWORD PTR _i$[ebp], 0
; 8   :  int & ref = i;
                    lea     eax, DWORD PTR _i$[ebp]
                    mov     DWORD PTR _ref$[ebp], eax
; 10  :  i = f();
                    call    ?f@@YAAAHXZ                 ; f
                    mov     eax, DWORD PTR [eax]
                    mov     DWORD PTR _i$[ebp], eax
; 12  :  f() = 5;
                    call    ?f@@YAAAHXZ                 ; f
                    mov     DWORD PTR [eax], 5
; 14  :  return(0);
                    xor     eax, eax
; 15  : }
```



39

## Melyik működik?

```
int main(void)
{
    int i = 0;
    int & ref = i;
    i = f();
    f() = 5;
    return(0);
}
int ext_i = 10;

int & f(void)
{
    return(ext_i);
}
```

i = ?
extern
auto
static külső
static belső
paraméter
paraméter, ref



40

## Feladat



41

# 2. Rész

## Osztályok

42

## Objektumorientált megközelítés

„Döntsd el, mely típusokra van szükség
és mindegyikhez biztosíts teljes műveletkészletet..”

Bjarne Stroustrup, A C++ programozási nyelv



43

## Új adattípus létrehozása C-ben

Egy lehetséges adattípus, mely a dátumot reprezentálja:

```
struct datum
{
   int nap;
   int honap;
   int ev;
};
struct datum d;

d.nap = 19;          //...változó  inicializálása
d.honap = 9;
d.ev = 1996;
```



44

## És  pl. nyomtatás?

Nyomtatásnál nem lehet a datum struktúrát átadni a printf-nek, mert az csak tagonként tudja kinyomtatni  vagy saját függvényt kell írni :

```
void d_datum( struct  datum * dt )
{
   static char * nev[] =
   { "?",
   "Január",     "Február",     "Március" ,     "Április",
   "Május",      "Június",      "Július",       "Augusztus",
   "Szeptember", "Október" ,    "November" ,    "December"};

   printf( "%d %s %d\n", dt->ev, nev[dt->honap], dt->nap );
}
```



45

## Hátrányok?

- nem garantálja az érvényes dátumot pl.:1985.02.32.
- nem  biztosítja a struktúra  inicializálását
- ha megváltozik a datum típus, akkor nehezen lehet a kódot utólag módosítani pl.:

```
struct datum
{
      int nap;          //...1...365
      int ev;
};
```

Mi ennek a következménye ? Minden programot, ahol a datum típust használják ,  minden kifejezést, ami  használja a hónap adattagot újra kell írni.
Természetesen ezeket a problémákat meg  lehet oldani több programozói munka ráfordítással (pl.:függvényen keresztül lehet **a tagokhoz hozzáférni,....).**



46

## Új adattípus létrehozása C++-ban

**adatok + adatokon manipuláló függvények = osztály**
osztály egy példánya : objektum

```
#include <iostream> -> cpp!
class CDatum
{
    //private:
       //...tagváltozók
       int m_iNap;
       int m_iHonap;
       int m_iEv;
    public:
       CDatum( int ev, int honap, int nap );

       ~CDatum();
       void display();
};
```



47

## Feladat



48

## Osztály

- a class, mint egy típus, a változó pedig objektum
- jelölés: kihangsúlyozza az adat és a függvény kapcsolatát
- tagfüggvényekből csak egy példány
- az objektumok csak a saját tagváltozóikat tartalmazzák



49

## Tagok láthatósága

- private : (def.)csak a tagfüggvények „osztály megvalósítása”
  (struc-> public -> class s{public…} -> class mindig!!!)
- public : bárki „kapcsolat a külvilággal”
- amíg a következő címke nem jön

```
int i;
CDatum d(1985, 12, 3);
i = d.m_iHonap;          //...HIBA
d.m_iNap = 1;            //...HIBA
```



50

## Tagfüggvények

- pl.: display() , megegyezik a C-belivel , de:
  - prototípus a CDatum deklarációjában van
  - definíció CDatum::display() ( class scope )
- függvénynév átdefiniálása
- C: d.nap C++: nap
  - a tagfüggvény „automatikusan” használja az „aktuális” objektum adattagjait, amihez tartozik ( C : d_datum( &dt) )
- tagfüggvény elérése mutatón keresztül
  - CDatum d(1985, 12, 3);
  - CDatum * pd = &d;
  - pd->display();
- tagfüggvény elérése hivatkozási típus segítségével
  - CDatum d(1985, 12, 3);
  - CDatum & rd = d;
  - rd.display();
- private függvényeket csak másik , osztályon belüli tagfüggvényből lehet elérni



51

## Konstruktor, destruktor

- Konstruktor
  - Speciális inicializáló függvény, mely minden objektum létrehozásakor automatikusan meghívódik.
  - biztonságos inicializálás
  - elnevezése megegyezik az osztály nevével
  - nincs visszatérési érték
  - több is lehet (függvénynév átdefiniálás)
  - ha nincs, generálódik egy „üres” – def. Konstruktor
  - Feladata: bázis oszt. konst hívása, virtuális fv
- Destruktor
  - Speciális „takarító” függvény, mely minden objektum megszűnésekor automatikusan meghívódik.
  - takarítás (pl.: memória felszabadítása)
  - elnevezése: tilde(~) + osztály neve
  - nincs paramétere és visszatérési értéke
  - csak egy példány



52

## Objektum létrehozása és megszüntetése

```
#include <iostream>
#include <string.h>

class CDemo
{public:
    CDemo( const char * nm );
    ~CDemo();
private:
    char m_szNev[20];
};

CDemo::CDemo( const char * nm )
{   strncpy( m_szNev, nm, 20 );
    cout << "Konstruktor: " << m_szNev << '\n';}

CDemo::~CDemo()
{   cout << "Destruktor: " << m_szNev << '\n'; }
```



53

## Objektum létrehozása és megszüntetése 2.

```
void f()
{
    CDemo localfObject("localfObject");
    static CDemo staticObject("staticObject");
    cout << "f()-ben\n";
}

CDemo globalObject("globalObject");

void main()
{
    CDemo localMainObject("localMainObject");
    cout << "main(), f() elott\n";
    f();
    cout << "main(), f() utan\n";
}
```



54

## Eredmény

```
/*
Konstruktor: globalObject !!!
Konstruktor: localMainObject
main(), f() elott
Konstruktor: localfObject
Konstruktor: staticObject !!!
f()-ben
Destruktor: localfObject
main(), f() utan
Destruktor: localMainObject
Destruktor: globalObject !!!
Destruktor: staticObject  !!!
*/
```



55

## Header és forrás állományok használata

- H : osztály interface
- CPP : implementáció
- interface, implementáció szétvágása : felhasználhatóság
- egy class -> egy forrás + egy header állomány
- kis, zárt osztályok
- többszörös include-ok elkerülése, gyorsabb fordítás

```
//...datum.h
// #pragma once
#ifndef _DATUM_H_
#define _DATUM_H_
.
.
.
#endif
```



56

## Hozzáférés az adattagokhoz

**FELADAT, három állományba!**
**Módosítsuk a CDatum osztályt, hogy az adattagokhoz hozzá lehessen férni !**



57

## Konstans objektumok és tagfüggvények

```
const int a=2;
const char * p = "eper";
char * const p = "alma";

Konstans objektumok :   „read-only" !
Konstans tagfüggvények:konstans objektumokra is meghivódnak !

class CDatum
{
.
    int GetDay()          const     {return m_iNap;}
    int GetMonth()        const     {return m_iHonap;}
    int GetYear()         const     {return m_iEv;}
    void SetDay(int dy)   const ;   //...HIBA???
.
};
```



58

## CDatum 4

```
CDatum::GetDay() const
{
    return m_iHonap;
}


main()
{
    int i;
    const CDatum d(1990, 11, 23);
    i = d.GetYear();
    d.SetDay( 3 );                  //...HIBA  MS VS 2008
}
```



59

## Osztály, mint tagváltozó

```
class CSzemelyInfo
{
public:
    CSzemelyInfo(char * nev, char * cim, int ev, int honap, int nap);

private:
    char m_szNev[30];
    char m_szCim[60];
    CDatum m_cdSzuletesnap;
};
.
```

m_cdSzuletesnap inicializálása ?



60

## a. változat

```
CSzemelyInfo::CSzemelyInfo(char * nev, char * cim,
int ev, int honap, int nap)
{
    strncpy(m_szNev, nev, 30);
    strncpy(m_szCim, cim, 60);
    m_cdSzuletesnap.SetYear(ev);
    m_cdSzuletesnap.SetMonth(honap);
    m_cdSzuletesnap.SetDay(nap);
}
```

A fordító itt **mindig meghívja a def. konstruktort**, tehát mindig kell egyet készíteni !

Konstans objektumra nem lehet használni a Set...típusú függvények miatt !



61

## b.változat

```
CSzemelyInfo::CSzemelyInfo(char * nev, char * cim, int ev, int honap, int
    nap) : m_cdSzuletesnap(ev, honap, nap)
{
    strncpy(m_szNev, nev, 30);
    strncpy(m_szCim, cim, 60);
}
```

1. CDatum::CDatum(. . . );
2. CSzemelyInfo :: CSzemelyInfo (. . . );

A const objektumra mindig ezt kell használni !

```
void main()
{
    CSzemelyInfo s("Kis Pista", "Nevenics utca", 1980, 12, 13 );
}
```



62

## Kivételkezelés

- Felmerülő hibák / kivételek (pl. ifstream eof) azonnali kezelésére.
- Hibák/kivételek egy helyen történő kezelése. -> külön a „normál” kódtól, azonnal a kezelő kódra kerül a vezérlés
- Hiba kezelése ott, ahol kell:
  - Olyan hibák kezelésére, amiket nem lehet helyben megszüntetni.
  - Máshol észlelt hibák kezelése.
- Ha a hiba helyben kezelhető, ne használjuk.
- Egymásba ágyazhatók



63

## Hogyan?

```
try {
    //...try blokk KÓD
    throw kivétel;      // Függvényben!
    } /

catch(tipus1 arg) //HIBAKEZELÉS
    {//...1.catch blokk
    }
catch(tipus2 arg)
    {//...2.catch blokk
    }
catch(tipusN arg) // Sorrend public származtatás! -> hierarchia: 1.bázis, 2.származtatott ->
                                                    megfogja a származtatott osztályt
    {//...N.catch blokk
    }
catch(...)
    {//...egyéb, ...nem kezelt, kivétel obj-t nem éri el!
    ....
    throw; //tovább lehet dobni!
    }
```

**throw**:
- „visszalép” a hívási fában, még egy blokkban el nem „kapják”!
- lokális változókat felszabadítja (meghívja a destruktort)
- DESTRUKTOR – kivétel a destruktorban? -> csak ha ott kezelődik le,különben abort() !



64

## Ha nincs elkapva…

-> terminate() hívás -> alapértelmezetten abort()-> kilépés VAGY saját rutin!

```
void term_func();

main()
{
    ...
    set_terminate(term_func);
    ...
}

void term_func()
{
    cout << „Saját term_func() hívás.\n";
    // saját „takarítás” !
    exit(-1); //ki kell lépni, nem lehet visszatérni!
}
```

ppt_terminate –> kód!



65

## Kivételkezelés, példa

```
try
{
        cout << "\ntry blokkban";
        throw "hiba";                          //...throw -1
        // throw CMyException2("Ez egy hiba! "); //...vagy CMyException2;
        cout << "\nthrow utan";
}
catch(int i)
{
        cout << "\ncatch(int i)-ben";
}
catch(char * msg)
{
        cout << "\ncatch(char * msg)-ben";
}
catch (CMyException2 e) // VAGY catch (CMyException2)
{
        cout <<  e.msg;
}
```

**Célszerű inkább felhasználói, mint beépített típust használni!**

ppt_exceptions –> kód!



66

## Kivételkezelés, példa2 FELADAT

ppt_exceptions -> kód!



67

## Konstruktor….?

```
Test2::Test2()
{
    Test1 t1("auto");
    Test1 * p = new Test1("new");
    k();                    //MyExp
}
void f(void)
{
    try
    {
        Test2 t2;
    }
    catch (MyExp)
    {
        cout << "\nMyExp";
    }
}
```

p ?



68

## auto_ptr

```
Test2::Test2()
{
    Test1 *pt = new Test1("--auto_ptr--");
    auto_ptr<Test1> pt2(pt);          //...Felszabadítja a memóriát!
```

„Dinamikus” mutatók kivétel biztos kezelésére !

(destruktor csak akkor hívódik meg, ha a konstruktor lefut!)

```
    Test1 *pt4 = pt3.get();           //...Cím lekérése

    k(); //...Kivételt dob
}
```

ppt_auto_ptr -> kód!



69

## Osztályok és a dinamikus memóriakezelés

C-ben:

```
struct datum * datumptr;
datumptr = (struct datum *)malloc( sizeof(struct datum) );
```

C++-ban malloc helyett new operátor !

```
int i;
datumptr = (CDatum *)malloc( sizeof( CDatum) );
i = datumptr-> GetDay();
```

C++ malloc?
m_iNap? i értéke NINCS definiálva !



70

## new

```
CDatum * datumptr1, *datumptr2;
datumptr1 = new CDatum;
int i = datumptr1->GetDay();
datumptr2 = new CDatum(1990, 11, 23);
i = datumptr2->GetDay();          //...i =23
```

- automatikusan mindig meghívja a konstruktort
- a malloc nem ismeri az objektum típusát : void *, sizeof()
- sikertelen allokálás esetén 0 (null pointer, NULL->0)



71

## delete

new párja: delete operátor

```
delete datumptr1;
delete datumptr2;
```

- a destruktor automatikusan meghívódik az objektum törlődése előtt
- észreveszi a NULL pointert !
- delete-new **PÁRBAN**!



72

## Példa a new, delete operátorokra

Alaptípus

```
int * ip = new int;
.
.
delete ip;
```

Tömb

```
int hossz;
char * cp = new char [hossz];
.
.
delete  [] cp;
```

Többdimenziós tömb

```
int (*matrix)[10];
int meret=20;
matrix = new int[meret][10];
.
.
delete [] matrix;
```



73

## Tömb esetén, []

```
CTest * p = new  CTest[5];
p[0].Display();              //!!!!!
delete[] p;                  //delete p;  HIBA!

//NINCS konstruktor!
int i;
CTest * * pp = new CTest*[3];

for (i = 0; i < 3; i++)      pp[i] = new CTest;

for (i = 0; i < 3; i++)      pp[i]->Display();  //!!!!!

for (i = 0; i < 3; i++)      delete pp[i];

delete[] pp;                 //bár nincs destruktor…
```

ppt_objektum_tömb -> kód!



74

## Pointer, mint tagváltozó FELADAT



75

## Teszt

```
void main()
{
try{
    CMyString str("ez egy string.");
    cout << '\n';     str.Display();
    str.SetAt(0, 'E');
    cout << '\n';     str.Display();
    str.Append("ez egy string. ");
}
catch (CMyStringExcp & e)
{
    cout << "\nExcp:" << e.m_Type;
}
catch (...)
{
    cout << "\nIsmeretlen kivétel!"; }
}
```



76

## Teszt

- Minden CMyString objektum két blokkot foglal le (két adattag)
- sizeof( CMyString ) : sizeof(char *) + sizeof( int ), nem a tömb mérete !
- Index ellenőrzés
- Private: m_pchData !!!!
- Destruktur : automatikusan felszabadítja a memóriát
- Append: string címe



77

## str2 = str1 ?

```
.
.
CMyString str1("alma"), str2("körte");
str2 = str1;
.
```

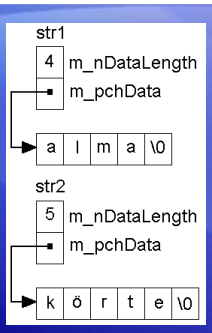
A „végrehajtódó” utasítások (tagonkénti értékadás) :

```
str2.m_pchData      =     str1.m_pchData;
str2.m_nDataLength  =     str1.m_nDataLength;
```



78

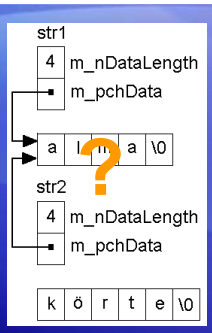
## str2 = str1 előtt






79

## str2 = str1 után




str1 módosítása megváltoztatja a str2 tartalmát is!



80

## Értékadás operator

str2 = str1 -> str2.operator=( str1 )

Értékadó operátor átdefiniálása.

```
class CMyString
{
.
  void operator=( const CMyString & str);
.
}
```

CSERE, FELADAT

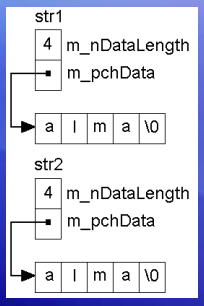
```
void CMyString::operator=( const CMyString & str)
{


}
```



81

## És az eredmény…






82

## str2 = str2 ?

```
str2 = str2;
vagy
CString * strptr = &str1;
.
.
str1 = *strptr;          ?  //...később, valahol

void CMyString::operator=( const CMyString & str)
{


}
```



83

## A this pointer

Minden osztályobjektum az osztály adattagjairól saját másolatot tárol. A tagfüggvényekből azonban csak EGY példány létezik.

Minden egyes tagfüggvény tartalmaz egy mutatót a saját osztálya típusához. Ennek a mutatónak this a neve. A this mutató annak az osztályobjektumnak a címét tartalmazza, amelyen keresztül az adott tagfüggvényt meghívtuk.



84

## Hogyan is működik?

```
void CMyString::SetAt(int nIndex, char ch)
{
   .
   m_pchData[nIndex] = ch;
   .
}

str.SetAt(0, 'E');
```

Az előbbi C megfelelője:

```
                                                    !
void CMyString_SetAt(CString * const this, int nIndex, char ch)
{
   .
   this->m_pchData[nIndex] = ch;
   .
}

CMyString_SetAt( &str,  0,  'E' );
```



85

## Használata

```
void CMyString::SetAt(int nIndex, char ch)
{
   .
   this->m_pchData[nIndex] = ch;
   (*this).m_pchData[nIndex] = ch;
   m_pchData[nIndex] = ch;
   .
}
```



86

## Akkor bővítsük a CMyString-t

FELADAT



87

## str1 = str2 = str3 ?

str1 = str2 = str3; ? (összekapcsolás)

Sorrend jobbról balra: str1 = ( str2 = str3 );
str1.operator=( str2.operator=( str3 ) )
FELADAT



88

## Értékadás vagy inicializálás ?

```
int i;
i=3;

int i=3;
```

Mi a különbség C-ben ? Semmi.

És C++ -ban ?

első eset : értékadás (nx !)
második eset: inicializálás (1x !)

```
CMyString  str1("alma");
CString l = "alma";
CString k = ('a', 21);
CString j = 'a';
CMyString  str2=str1;     //...Mi hívódik meg ?
```



89

## Copy constructor

Mivel az operator = funkció csak LÉTEZŐ objektumra hívható, ezért az un. „copy constructor” hívódik meg az előbbi példánál „ha van”, különben tagonkénti inicializálás ( 4.2.4 ,tömb !) hajtódik végre !

```
class CMyString
{
   .
   CMyString( const CMyString & str);
   .
};
```

CSERE, FELADAT



90

## „Copy constructor” alkalmazásai

- **Inicializálás**: lásd előbb
- **Objektum, mint paraméter (kivétel par. &!)**

```
void f( CMyString str )
{
.
}
main()
{
  CMyString s("banán");
  f( s );
}
```

Az f() egy másolatát kapja meg az s-nek, azaz az str-t a következők szerint inicializálja:
CMyString str( s ) -> **„copy constructor” hívás, ha van!**



91

## És még egy…

- **Objektum, mint visszatérési érték**

```
CMyString f( )
{
  CMyString retStr("banán");
  return retStr;
}
 main()
{
  CMyString s2;
  s2 = f( );
}
```

**Mi történik visszatéréskor ?**

**CString temp(retStr);** -> **„copy constructor” hívás, ha van!**
**s2 = temp;** **(temp : ideiglenes objektum)**



92

## Egy gyorsabb megoldás

```
void f( const CMyString  & str )
{
  .
  .
}
main()
{
  CMyString s("banán");
  f( s );
}
```

**Hívásnál: const CMyString & str = s;**
**azaz új objektum nem jön létre !**

**Az operator= - nál szintén alkalmazható!**
**Visszatérési értéknél csak óvatosan ! (automatikus változó!!! )**



93

## Ami célszerű….

Alapértelmezett konstruktor (Default constructor)

Értékadó operátor

Másoló konstruktor (Copy constructor)



94

## Static tagok

**Közös erőforrások, fontosabb információk tárolása.**
**Osztályhoz tartozik, nem az objektumhoz.**

```
class CMyString
{
  .
  static int m_iCounter;
  .
};
```

**Minden létrejövő CMyString típusú változó látja!**
**Inicializálás mindig blokkon kívül:**

```
int CMyString::m_iCounter = 0;
```

**Hivatkozás a static tagra :**

```
CMyString s;
cout<<s.m_iCounter ;
cout<<CString::m_iCounter;
```



95

## Static tagfüggvények

```
class CMyString
{  .
public:
  static int GetObjectCount() { return m_iCounter; }
  .
};
main()
{
    CMyString  s;
    cout <<s.GetObjectCount();
    cout<< CMyString::GetObjectCount();
}
```

**Csak static tagváltozók és static tagfüggvények érhetők el a static tagfüggvényekből !**
**This nem érhető el!**
**FELADAT: m_iCounter, de csak DEBUG módban!**



96

## Friend mechanizmus

```
class Cp1
{
   private:
      int m_iTopSecret;
};

class Cp2
{
public:
   void change( Cp1 p1)    {p1. m_iTopSecret++;}  //...ERROR !
};
```

**Megoldás?**

```
class Cp1
{
   friend class CP2;          //...függvény is lehet !
private:
   int m_iTopSecret;
};
```

**A hozzáférés nem kölcsönös !**



97

## FELADAT, lista kezelése, iterator



98

## Objektum tömbök

```
void main()
{
   CMyString msg[10] = {    "\nElső",
                            "\nMásodik",
                            CMyString("\nHarmadik\n"),
                            CMyString('x', 10),
                            CMyString() };
   for(int i=0; i<10; i++)
         msg[i].Display();
}

CMyString * str = new CMyString[5];
.
delete [] str;
```

**Ebben az esetben a delete str HIBÁS !** ppt_objektum_tömb -> kód!



99

## set_new_handler függvény

**Sikertelen new használatának ellenőrzése.**

```
#include <iostream>
#include <stdlib.h>
#include <new>

void fullmem(void)
{
   cerr<< "\nElfogyott a memória!\n";
   exit(-1);
}
void main()
{
   set_new_handler( fullmem );
   long total=0;
   while(1)
   {
         char * p = new char[10000];
         total += 10000;
         cout << "\n>>>>>\t" << total;
   }
}
```



100

## A new és delete operátor átdefiniálása

**Memória inicializálása**

```
#include <iostream>
#include <stdlib.h>
#include <string.h>

void *operator new(size_t size)
{
   void * ptrRet = calloc(1,size);
   return ptrRet;
}
 void *operator new(size_t size, char fill)
{
   void * ptrRet;
   if( (ptrRet = malloc(size)) != NULL )
         memset(ptrRet, fill, size);
   return ptrRet;
}
void operator delete( void * ptr ) !!!
{
   free(ptr);
}
```



101

## Átdefiniálás

```
void main()
{
   int * ip = new int [10];
   for(int i=0; i<10; i++)
         cout<<" "<<ip[i];
   delete [] ip;

   char * cp = new( '*') char [10];
   for(i=0; i<10; i++)
         cout << " " << cp[i];
   delete [] cp;
}
```



102

## Osztály specifikus new és delete operátor **FELADAT**



103

## Operátorok átdefiniálása

Átdefiniálható operátorok:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| + | - | * | / | % | ^ | & | \| |
| ~ | ! | , | = | < | > | <= | >= |
| ++ | -- | << | >> | == | != | && | \|\| |
| += | -= | *= | /= | %= | ^= | &= | \|= |
| <<= | >>= | [] | () | -> | new | delete | |



104

## Néhány megszorítás…

Amit nem lehet:

- a nyelvet új operátorokkal kibővíteni (pl.:**)
- az oparátorok operandusainak a számát módosítani
- az operátorok precedenciáját megváltoztatni
- az operátorok asszociativitását megváltoztatni
- az alapadattípusok operátorait felülírni
- a következő operátorokat felülírni:
  - . osztálytag operátor,
  - ::scope,
  - ?: feltételes kifejezés



105

## **FELADAT CSERE**, index operátor



106

## **FELADAT** Példa, << operátor

?



107

## **FELADAT** Példa +=, + operátorok,

```
CMyString y("banán"), m("eper");
CMyString z((const char *)NULL);
CMyString x("012345678"),

x = m;
x += y;
z = y + m; //  (y+m!)
```



108

## Példa +=, + operátorok, FELADAT

ppt_string2 -> kód!

109

109

## Példa += str, += psz, = str1 +str2

„...\Példák\ppt_példák\ppt_string2"

110

110

## Objektumorientált megközelítés

„Döntsd el, mely osztályokra van szükséged, biztosíts mindegyikhez teljes műveletkészletet, az öröklés segítségével pedig határold körül pontosan a **közös tulajdonságokat**."

Bjarne Stroustrup, A C++ programozási nyelv

Koschek Vilmos C++ Osztályok használata 111

111

## Öröklés (inheritance) és többrétűség (polymorphism)

Kapcsolódó adatszerkezetek kezelése C-ben

Feladat: különböző munkabérek kiszámítása, melyek az alábbiak lehetnek

- **dolgozó**: órabér * ledolgozott órák száma
- **ügynök**: (órabér * ledolgozott órák száma)+(jutalék*eladott darabszám)
- **vezető**: heti bér

Koschek Vilmos C++ Osztályok használata 112

112

## Egy lehetséges megoldás …

```
struct wage_pay
{
    float wage;          //...munkabér,órabér
    float hrs;           //...óra
};

struct sales_pay
{
    float wage;
    float hrs ;
    float commission;    //...jutalék
    float sales_made;    //...eladott darabszám
};

struct manager_pay
{
    float weekly_salary; //...heti bér
};
```

Koschek Vilmos C++ Osztályok használata 113

113

## Adatstruktúrák

```
enum { WAGE_EMPLOYEE, SALESPERSON, MANAGER }
    EMPLOYEE_TYPE;

typedef struct employee
{
    char name[30];
    EMPLOYEE_TYPE    type;
    union
        {    struct wage_pay        worker;
             struct sales_pay       seller;
             struct manager_pay     manager;
        };
} EMPLOYEE;
```

Koschek Vilmos C++ Osztályok használata 114

114

## És a kód

```
float Compute_pay(EMPLOYEE * p )
{
    switch(p->type)
    {
        case WAGE_EMPLOYEE:
            return p-> worker.wage * p-> worker.hrs;
                                    break;
        case SALESPERSON:
            return p-> seller.wage * p-> seller.hrs +
             p-> seller.commission * p-> seller.sales_made;
            break;
        case MANAGER:
            return p-> manager.weekly_salary;
                    break;
    //....
    };
}
```



115

## A fenti konstrukció néhány hátránya

- Nehezen olvasható, különösen , hogy több adattípus feldolgozása egy helyen történik
- Nehézkes a kezelése, mert minden új (dolgozó) típusnál a teljes kódot módosítani , újra kell fordítani és tesztelni.

A C nem szolgáltat ☹ egy könnyű és kezelhető utat a fentiek megvalósítására.



116

## Akkor most tegyük meg…

„Döntsd el, mely osztályokra van szükséged, biztosíts mindegyikhez teljes műveletkészletet, az öröklés segítségével pedig határold körül pontosan a **közös tulajdonságokat**.”

Bjarne Stroustrup, A C++ programozási nyelv



117

## Hogyan is?

Feladat: különböző munkabérek kiszámítása, melyek az alábbiak lehetnek

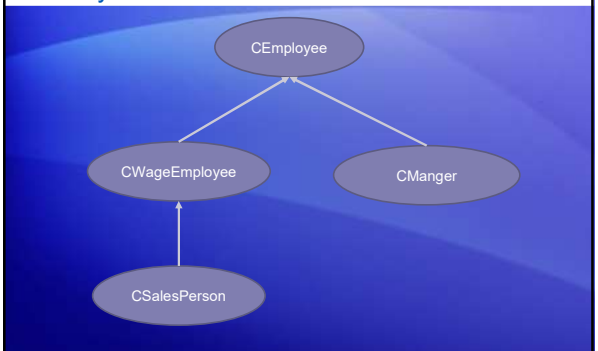
- **dolgozó**: órabér * ledolgozott órák száma
- **ügynök**: (órabér * ledolgozott órák száma)+(jutalék*eladott darabszám)
- **vezető**: heti bér

Hogyan lehet?



118

## Osztályhierarchia






119

## Kapcsolódó adatszerkezetek kezelése C++-ban

Alkalmazottak (dolgozó, ügynök, vezető) **közös** jellemzői :

```
class CEmployee
{
public:
  CEmployee();
  CEmployee(const char *nm);
  const char * GetName() const; //2 x const!!!
private:
  char m_szName[30];  //később CMyString!
//...egyéb jellemzők
...
};
```



120

## WageEmployee ?

Hogyan tudom felhasználni a CEmployee-t a WageEmployee-nél?

- Mint a CWageEmployee-nak egy tagváltozója
- Öröklés

```
class CWageEmployee : public CEmployee
{
public:
   CWageEmployee(const char *nm);
   void SetWage(float wg);
   void SetHours(float hrs);
private:
   float m_fWage;      //órabér
   float m_fHours;     //órák száma
};
```



121

## Öröklés

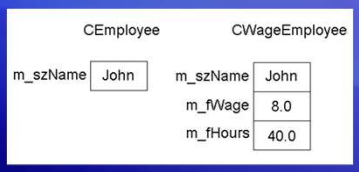
- CWageEmployee: származtatott osztály „derived class”
- CEmployee: bázisosztály „base class”
- public: csak a public ! (interface használata !)
- private: semmihez a gyereknek!
- Újrafelhasználható kód, adatstruktúrák -> duplikálás elkerülése
- Áttekinthetőbb, könnyebben karbantartható kód, mert az adatok + kódok egy helyen
- Osztályhierarchia sok esetben közel áll a való világhoz.



122

## CWageEmployee

```
CWageEmployee  aWorker ( „John” );
char * str;
aWorker.SetHours( 40.0 ); //... CWageEmployee::SetHours()
str = aWorker.GetName();  //... CEmployee::GetName()
```






123

## Többszintű öröklés

```
class CSalesPerson:public CWageEmployee
{
 public:
    CSalesPerson(const char * nm);
    void SetCommision(float comm);
    void SetSales(float sales);

private:
    float m_fCommission; //jutalék
    float m_fSalesMade;   //eladott darabszám
};
```



124

## CManager

```
class CManager:public CEmployee
{
public:
        CManager(const char * nm);
        void SetSalary(float salary);

private:
    float m_fWeeklySalary; //heti bér
};
```



125

## Bázisosztály tagjainak átdefiniálása

Fizetések számítása:

```
float CWageEmployee::ComputePay() const
{
    return m_fWage * m_fHours;
}

float CSalesPerson::ComputePay() const
{
    return CWageEmployee::ComputePay() +
           m_fCommission * m_fSalesMade;
} //...m_fWage,mfHours:ERROR, ComputePay() ?!

float CManager::ComputePay()
{
    return m_fWeeklySalary;
}
```



126

## Használata

```
CSalesPerson aSeller( „John Smith");
aSeller.SetHours(40.0);
aSeller.SetWage(6.0);
aSeller.SetCommission(0.05);
aSeller.SetSales(2000.0);
cout << aSeller.ComputePay() << '\n';
cout << aSeller.CWageEmployee:: ComputePay() << '\n';
```



127

## Származtatott osztály konstruktora

```
CWageEmployee::CWageEmployee(const char  * nm)
   :CEmployee(nm)
{
   m_fWage = 0;
   m_fHours = 0;
 }

CSalesPerson::CSalesPerson(const char * nm):CWageEmployee(nm)
{
   m_fCommission = 0;
   m_fSalesMade = 0;
}

CManager::CManager(const char *nm):CEmployee(nm)
{
   m_fWeeklySalary = 0;
}
```



128

## Konverziók a bázis- és a származtatott osztály között

**Származtatott => Bázis**
**OK!**

Származtatott <= Bázis
**HIBÁS ! (lehet)**



129

## Származtatott => Bázis

```
CEmployee * empPtr;

CManager aBoss( „Mary Brown" );

empPtr = &aBoss;

empPtr-> GetName();          ?

empPtr->SetSalary(10);       ?
```



130

## Származtatott <= Bázis

```
CWageEmployee * wagePtr = &aSeller;
CSalesPerson * salePtr;

//...lehet, de veszélyes !
salePtr = (CSalesPerson *)wagePtr;    ?


CEmployee * empPtr = &aWorker;
CSalesPerson * salePtr;

//...legális, de inkorrekt !
salePtr = (CSalesPerson*)empPtr;      ?

salePtr->SetCommission( 0.05 )        ?
```



131

## Konverziók, kicsit ügyesebben…static_cast

Ha egymásba konvertálhatók, egyébként fordítási hiba.

```
//OK
pEmployee = static_cast<CEmployee*>(pWage);

//OK, de biztos?
pSales = static_cast<CSalesPerson*>(pEmployee);

//Fordítási hiba, nincs a hierarchiában
CTeszt * pTeszt;
pTeszt = static_cast<CTeszt*>(pEmployee);
```



132

## Konverziók, kicsit ügyesebben…dynamic_cast

Pointerek és ref. futásidejű konverziójának ellenőrzése a hierarchiában! ref:bad_cast, ptr:0

```
pSales = &Sales;
pEmployee = pSales;
 //OK
pSales2 = dynamic_cast<CSalesPerson*>(pEmployee);
 //OK
pWage = dynamic_cast<CWageEmployee *> (pEmployee);

pEmployee = &Employee; // egy CEmployee !!!!
 //Null pointer
pSales2 = dynamic_cast<CSalesPerson*>(pEmployee);
 //Null pointer
pTeszt = dynamic_cast<CTeszt*>(pEmployee);
```



133

## Hm…const_cast

const eltávolítása a változóból….

```
class CCTest
{
public:
    void setNumber( int ) { number = num; }
    void printNumber() const;
private:
    int number;
};

void CCTest::printNumber() const
{
    cout << "\nBefore: " << number;
    const_cast< CCTest * >( this )->number--;
    cout << "\nAfter: " << number;
}
```



134

## Egy kérdés…

```
class CEmployeeList
{
.
    Add( CEmployee * newEmp );
.
};
.
CWageEmployee * wagePtr = new CWageEmployee( „Bill Shapiro" );
CSalesPerson * salePtr = new CSalesPerson( „John Smith" );
CManager * mgrPtr = new CManager( „Mary Brown" );
...Add(..)...
.
Employee * person;
while(person = anIter.GetNext())
{   .
    cout << "\n" << person->GetName() ;
.
```

Fizetés számítása ? a megfelelő ComputePay() meghívása
C: ptr típusának meghatározásával, C++: ?



135

## Virtuális függvények

Tagfüggvények, DE amikor egy ilyen függvény meghívódik egy bázisosztály mutatón keresztül, akkor a függvénynek a SZÁRMAZTATOTT osztálybeli verziója hajtódik végre.

Változások a CEmployee-ben

```
class CEmployee
{
.
public:
    virtual ~CEmployee()          {cout << "\n~CEmployee()";}
    virtual float ComputePay()    {return 0.0;}    //...virtual csak egy helyen, bárhol
.
};
```



136

## Még mindig virtuális függvények

```
CWageEmployee aWorker("John");
CSalesPerson aSeller("John Smith");
CManager aManager("Mary Brown");

CEmployee * empPtr;

//...virtual float ComputePay()
empPtr = &aWorker;      cout << '\n' << empPtr->ComputePay();
empPtr = &aSeller;      cout << '\n' << empPtr->ComputePay();
empPtr = &aManager;     cout << '\n' << empPtr->ComputePay();

//...virtual ~CEmployee()
empPtr = new CManager( "Jack" );
delete empPtr;
```

Azt a képességet, hogy egy objektum tagfüggvényét meg tudjuk hívni az objektum típusának ismerete nélkül, többrétűségnek (polymorphism) hívjuk.

FELADAT A KÖVETKEZŐ ÓRÁN!



137

## Hogyan is működik?

```
class CA
{
   public :
   int x;
   virtual void f1(int x) {...};
   virtual void f2(int x) {...};
};

class CB : public CA
{
   public :
   int y;
   virtual void f1(int x) {...};
   virtual void f2(int x) {...};
   virtual void f3(int x) {...};
};
```



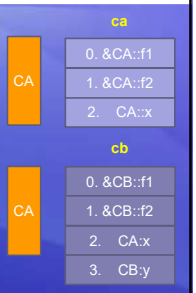
138

## Hívás

```
CA ca;
CB cb;
CA * pCa1 = &ca;
CA * pCa2 = &cb;

pCa1->f1();      -> ca[0] -> &CA::f1

pCa2->f1();      -> cb[0] -> &CB::f1
```




**Konstruktor: v.fv címének kitöltése!**



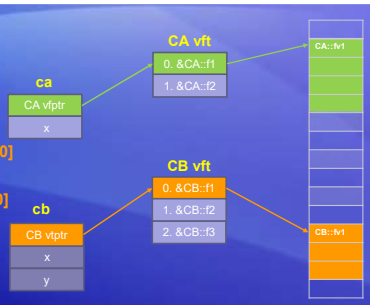
139

## Virtuális fv. tábla

```
CA ca;
CB cb;
CA * pCa1 = &ca;
CA * pCa2 = &cb;

pCa1->f1();
        -> pCa1->vft[0]

pCa2->f1();
        -> pCa2->vft[0]
```




**Konstruktor: vfpt beállítása!**

vfptr: virtual function table pointer



140

## És még egy kis apróság…

```
class CA {
    public:
    virtual void fv(int i=1) { cout << "\n (CA)i: " << i; }
};
class CB : public CA {
    public:
    virtual void fv(int i=5) {cout << "\n (CB)i: " << i; }
};
void main()
{
    CB b;
    CA * p = &b;
    p->fv();        //-> (CB)i: 1 ??????
}
```

**Fordításnál CA (CA *p!!!) az ismert típus !**



141

## Akkor miért is jó ez?

Statikus függvényhívás (early binding, static binding) : C C++
Dinamikus függvényhívás (late binding, dynamic binding): - C++

```
float ComputePayroll( CEmployeeList & l )
{
    float payroll=0;
    CEmployee * person;
    CEmplter anIter( l );
    person = anIter.GetFirst();
    payroll = person->ComputePay();          //...late binding
    while( person=anIter->GetNext() )
    {
        payroll += person->ComputePay();     //...late binding
    }
    return payroll;
}
```

**Mit kell változtatni, ha egy új típussal (pl.:CConsultant) bővítjük a foglalkozásokat ?**
**(EMPUTIL.H + EMPUTIL.OBJ + CONSULT.* !!! )**



142

## Absztrakt osztályok

```
class CEmployee
{
.
    virtual float ComputePay()    const=0; //...pure virtual
.
};
```

DE a származtatott osztályban definiálni kell !

A fentieket tartalmazó osztályokat absztrakt osztályoknak hívjuk.
(legalább egy pure virtual fv van benne)

Nem lehet példányosítani!

Alkalmazás: felület, bázisosztály



143

## Példa

```
class CSortedList
{
.
    AddItem( const CSortableObject &newitem);
.
};

class CSortableObject
{
public:
    virtual int IsEqual(const CSortableObject &other) = 0;
    virtual int IsLessThan(const CSortableObject &other) = 0;
};
```



144

## Példa folytatása…

```
class CSortableName : public CSortableObject
{
public:
    virtual int IsEqual(const CSortableObject &other);
    virtual int IsLessThan(const CSortableObject &other);
private:
    char name[30];
};

int CSortableName :: IsEqual (const CSortableObject &other)
{
    return ( strncmp(name. Other.name, 30)  == 0 );
}
```

A CSortedList minden olyan objektumot képes tárolni ami a CSortableObject -ből származik ! (CSortablePhone,...)



145

## Protected tagok

A protected abban különbözik a private-tól, hogy a származtatott osztályok ezeket a tagokat is láthatják.

```
class CBase
{protected:
    int m_iSecret;
private:
    int m_iTopSecret;
};

class CAclass : public CBase
{public:
    void func();
};
void CAclass::func()
{
    m_iSecret = 1;          //...OK
    m_iTopSecret = 1;       //...ERROR
}
```



146

## Public, private, protected tagok

Public: mindenki látja

Protected: osztályon belül, és a származtatott osztály

Private: osztályon belül



147

## Public, private, protected BÁZISOSZTÁLYOK

public: belülről (fv), a származtatott osztályból a bázisosztály public és protected tagjai érhetők el. Kívülről a származtatott és bázis osztály public tagjai.

protected: belülről, származtatott osztályból a bázisosztály public és protected tagjai érhetők el. Kívülről az adott osztály saját public tagjai.

private: belülről csak a **közvetlen** bázisosztály public és protected tagjai érhetők el. Kívülről az adott osztály saját public tagjai.

A bázisosztály public és protected tagjai a soron következő származtatásokból „kizáródnak”.



148

## Többszörös öröklés

CSalesManager ?

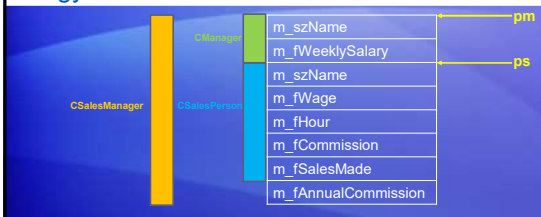
```
class CSalesManager: public CSalesPerson,  public
   CManager
{
   .
   .
   private:
        float m_fAnnualCommission;  //éves jutalék

};
```



149

## Hogyan is néz ez ki?




```
CSalesManager s;
CManager *pm = &s;
CSalesPerson * ps = &s;
```

pm = ps ???

pm = (CManager *)ps; ???

Minden ptr a ptr típusának megfelelő rész elejére mutat!

pm = (CManager*)(CSalesManager *)ps;



150

## És a bázisosztályok?

```
CSalesManager aSellerBoss;

char * str = aSellerBoss.GetName()     ?


str = aSellerBoss.CManager::GetName();
```



151

## A két CEmployee…

```
CEmployee * empPtr;
CSalesManager * salesmgrPtr;

empPtr = salesmgrPtr;          ?


empPtr = (CManager  *)salesmgrPtr;
```



152

## Elegánsabb megoldás

**Virtuális bázisosztály (virtual base class)**

```
class CWageEmployee : public virtual CEmployee
{
   ...
};
class CManager : public virtual CEmployee
{
   ...
};
```

NINCS DUPLIKÁCIÓ, csak egy CEmployee lesz !

```
CSalesManager aSellerBoss;
//...OK
char * str = aSellerBoss.GetName() ;
```

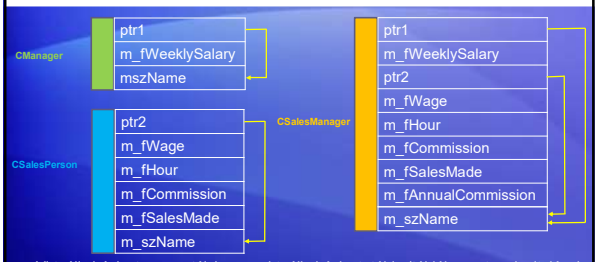
És még valami…

```
float CSalesManager::ComputePay()
{
   return CSalesPerson::ComputePay() + CManager::ComputePay();
}
```



153

## És a memóriában?




Virtuális bázisptr az osztályban -> virtuális bázisptr tábla (táblában a tagok eltolása)

Mindig az objektum végén

A származtatott konst. állítja be a ptr-t!

Nem „növeli" a sebességet -> +indirekció



154

## Névterek

- A program logikailag összetartozó részeinek jelölése.
- Névütközések elkerülése
- Egymásba ágyazhatók
- Tetszőleges névhierarchia, modulok, hivatkozás A::B::
- Külső hivatkozásnál scope operátor
- Alias használata

```
namespace X
{
   int i;
   double j;
}

int main()
{
   X::i++;
}
```

```
namespace a_very_long_namespace_name { ... }

namespace AVLNN = a_very_long_namespace_name;
```



155

## Névterek, hogyan is? CMyString…

**mystring.h-ban**

```
namespace MyString
{
   class CMyStringExcp {…}
   class CMyString {…}
   CMyString operator +(const CMyString& string1, const CMyString& string2);
   std::ostream & operator << (std::ostream& os, CMyString & s); //std? include NE!
} //MyString
```

**mystring.cpp-ben**

```
using namespace std; //  OK !   Ez (preferált) vagy namespace vagy MyString:: fvként!
namespace MyString
{
   CMyStringExcp:: CMyStringExcp() {…}
   …
   CMyString::CMyString() {…}
   ….
   CMyString operator +(const CMyString& string1, const CMyString& string2) {}

   ostream & operator << (ostream& os, CMyString & s) {}
}//MyString
```



156

## Névterek, használatuk

main.cpp-ben
```
#include <iostream>
...
#include "mystring.h"
...
using namespace std;
using namespace MyString;

main()
{
    CMyString str("\nalma");

    cout << str;

    cout << (const char*)str;

}
```

main.cpp-ben
```
#include <iostream>
...
#include "mystring.h„
...
//using namespace std;
using namespace MyString;

main()
{
    MyString::CMyString str("\nalma");

    MyString::operator<<(std::cout, str);

    std::operator<<(std::cout, (const char *)str);

}
```

**std::cout<< (const char *)str ?**



157

## Argumentumfüggő névfeloldás

std::cout << (const char*)str;

?

std::operator<<(std::cout, (const char *)str);

<< operátor fv -> std-ben
cout globális objektum -> std-ben

Elég egyet megadni, másikat már az std-ben keresi a fordító!



158

## Névterek, példa

„...\Példák\ppt_példák\ppt_string3”



159

## FELADAT



160

## Köszönöm a figyelmet!

koschek.vilmos@nik.uni-obuda.hu



161