További lehetőségek

1.) tagváltozók inicializálása

```
class Point
{
        unsigned x;
        unsigned y;
public:
        Point(unsigned px, unsigned py)
        {
                x = px;
                y = py;
        }
};

class Point
{
        …
        Point(unsigned px, unsigned py) : x(px), y(py) {}
        …
};

class Circle
{
        unsigned r;
public:
        Point &c;
        Circle(unsigned r, Point c) : r(r), c(c){ }
        Circle(unsigned r, unsigned x, unsigned y): r(r), c(x,y) { }

};
```

2.) Konstans tagok

```
class Person
{
        public:
        const unsigned birthYear;
        private:
        string name; //már irtunk egy string osztályt
        public:

        Person(unsigned birthYear, string name): birthYear(birthYear)
        {
                //name ellenőrzés
                this->name = name;
        }
```

```
        string getName()
        {
                return name;
        }

        …

};

void fv(const Person &p)
{
        …
        p.getName(); //compiler HIBA!, mert a getName-ről nem tudja, hogy változtat-e p-n
}
```

kiküszöbölés: getName() **const**

```
                {
                        return name;
                }
```

Statikus tagok

C-ben:

static *globális változó*; //extern-nel nem lehet másik c-ből linekelni ha static, mert nem látja a fordító

C++ (ez elavult):

```
int fv()
{
        static unsigned count = 0;    //inicializálás 1x történik meg, NEM a stackre megy le! de
                                      //nem használunk globális változót (olllé)
        count++;
}

class Person
{
        […]

        static unsigned votingYear; //osztály változó

        […]
};
```

.cpp:

unsigned Person :: votingYear = 18; **//így kell inicializálni (CSAK így lehet!!!)**
// Person :: votingYear-rel lehethozzáférni mert static

3.) **Friend**

a.) **függvény hozzáférését engedélyezzük**

```
class Point
{
        unsigned x;
        unsigned y;

        friend void printPoint(Point p);
};
```

cpp:

```
void printPoint(Point p)
{
        printf(„%u, %u”, p.x, p.y); //a friend miatt engedi a hozzáférést
}
```

b.) **másik osztály hozzáférését engedélyezzük**

```
class Point
{
        unsigned x;
        unsigned y;

        friend class PrintPoint;
};


class PrintPoint
{
        …
public:
        void printPoint(Point p) { …};
};
```

4.) **Névterek**

a.)

String.h:

```
namespace myLib
{
	class String
	{
		[…]
		char* get();
	};
}
```

string.cpp:

```
char* myLib:: String::get() {…}
```

b.)

```
namespace myLib
{
	char* String::get() {…}
}
```

```
//felhasználás

myLib:: String str;
```

= =

```
using namespace myLib;
string str;
```

→ a névtelen névterek olyanok mintha a tagok bennük static lennének

5.) **C++ I/O**

a.) állománykezelés

```
#include <fstream>
using namespace std;

ifstream in(„c:\\adat.dat”);
ofstring out(„c:\\adat.dat”);

if(!in) {//hiba}
if(!out) {//hiba}
```

```
char ch;
while(in.get(ch))
{
        out.put(ch);
}

//hibaellenőrzés!
```

```
ofstream out(„…”, ios::binary | iosport);
out.put(11);
out.write((const char*)&i, sizeof(int));

ifstream in(„…”, ios::binary|iob::in);
char c;
in.get(c);

int i;
in.read((char *)&i, sizeof(int));
```

Név	C	C++
be	stdin	cin (obj.)
ki	stdout	cout (obj.)
hiba	stden	cen (obj.)
napló	--	clog (bufferelt)

---

pl.:

```
#include <iostream>
using namespace std;

cout << „Enter a number”;
int i;
cin >> i;

cout << „What you have entered is” << i << endl;   // endl = „\n”
```

```
//C
printf(„%8.2lf, % 8.2lf\n”, x, y);

//C++
cout << setprecision(2) << setiosflex(ios::fixed) << setw(8) <<x << setw(8) << y << endl;
```

I/O manipulátorok

#include <iomanip>

int n = 12;
cout << hex << n;
cout << dec << n;
cout << oct << n;

manipulátor	*tagfüggvény*
cout<<setprecision(2)	cout.precision(2);
setw	width
setiosflags	setf
resetiosflags	unsetf
dec, hex, oct	dec, hex, oct
showpos/noshowpos	- || -
showupper/noshowupper	- || -
showbase/noshowbase	- || -
boolalpha/noboolalpha	- || -

left/right/internal (rendezi a helyét)

showpost
showupper