Objektum orientált programozás C++ nyelven
1. Bevezetés + Alapok

PEKÁRDY MILÁN – PANNON EGYETEM

PEKARDY@DCS.UNI-PANNON.HU

Források

- http://www.cplusplus.com
- https://isocpp.org/
- http://en.cppreference.com/w/
- Könyv:
  - http://fizweb.elte.hu/download/Fizika-BSc/C-Cpp-programozas/Cpp_Stroustrup.pdf
- Anyagok:
  - https://github.com/pekmil/CppTutorial

Fordítók (compilers)

- A számítógép a gépi nyelvet (machine language) érti
- Forráskódot általában valamilyen magasabb szintű programozási nyelven írunk (pl. C++, Java, C#, stb.)
- Az általunk írt forráskódokat le kell fordítani gépi nyelvre: ezt végzik el a különböző fordítóprogramok (compilers), értelmezők (interpreters), assemblers
- C++ nyelven írt kódokhoz egy fordítóprogram szükséges, ami a gép által közvetlenül értett gépi nyelvre fordítja a programunkat, így a generált program különösen hatékony lesz

Fordítók (compilers)

```
1 int a, b, sum;
2
3 cin >> a;
4 cin >> b;
5
6 sum = a + b;
7 cout << sum << endl;
```

00000	10011110
00001	11110100
00010	10011110
00011	11010100
00100	10111111
00101	00000000

Fejlesztő eszközök

- A mai modern fejlesztéshez elengedhetetlen a megfelelő eszközök használata (akár online eszközök is már)
- Integrált fejlesztő környezetek (IDE):
  - Tartalmaznak különböző szerkesztőket,
  - Fordítóprogramokat,
  - Hibakereső komponenseket (debugger),
  - Rengeteg egyéb hasznos eszközt, ami segíti a fejlesztést (kód kiegészítés, szintaktika ellenőrzés, kód részlet generálás, stb.)
- Az eszközökön kívül a megfelelő fejlesztési ajánlások, irányelvek, tippek, trükkök alkalmazása is előnyös tud lenni

Fejlesztő eszközök

- Qt Creator IDE-t használjuk (létezik jobb is, de a célnak megfelel)
- IDE nélkül, parancssorban is fordíthatunk:

Compiler	Platform	Command
GCC	Linux, among others...	`g++ -std=c++0x example.cpp -o example_program`
Clang	OS X, among others...	`clang++ -std=c++11 -stdlib=libc++ example.cpp -o example_program`

C++ program szerkezete

```cpp
//first C++ program
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello World!" << endl;
    return 0;
}
```

C++ program szerkezete

- 1. sor: megjegyzés (egy soros)
- 2. sor: direktíva (#), ezeket egy előfeldolgozó (preprocessor) dolgozza fel a fordítás előtt
- 3. sor: névtér használatának deklarálása
- 4. sor: main függvény deklarálása, a program belépési pontja
- 5. és 8. sor: blokk megadása zárójelekkel
- 6. sor: C++ utasítás, szekvenciális végrehajtás (alap esetben)
- 7. sor: újabb utasítás, program visszatérési értéke

C++ program szerkezete

- A sortörések, tabulátorok, stb. csak az olvashatóságot segítik, egyéb jelentősége nincs
- Az utasítások végén mindig pontosvessző van
- Egy- illetve több soros megjegyzések szintén segítik a kód megértését, akár másik akár magunk számára is

```
/* first C++ program
   with multi-line comments */
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello World!" << endl; //prints "Hello World!" to the console
    return 0; //successful return from main
}
```

Változók és típusok

- Adatok tárolása különböző típusú változókban (általában valamilyen memória részt reprezentálnak)
- Változókra azonosítójukkal (identifier) hivatkozhatunk
- Fenntartott kulcsszavakkal nem ütközhetnek a nevek
- Kis- és nagybetűk megkülönböztetettek (case sensitive)
- Erősen típusos nyelv (strongly-typed)

```
alignas, alignof, and, and_eq, asm, auto, bitand, bitor, bool, break, case, catch, char, char16_t, char32_t,
class, compl, const, constexpr, const_cast, continue, decltype, default, delete, do, double, dynamic_cast,
else, enum, explicit, export, extern, false, float, for, friend, goto, if, inline, int, long, mutable,
namespace, new, noexcept, not, not_eq, nullptr, operator, or, or_eq, private, protected, public, register,
reinterpret_cast, return, short, signed, sizeof, static, static_assert, static_cast, struct, switch, template,
this, thread_local, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void,
volatile, wchar_t, while, xor, xor_eq
```

Változók és típusok

- Alapvető típusok osztályai (fundamental data types):
  - Karakter típusok,
  - Egész számok,
  - Lebegőpontos számok,
  - Logikai értékek
- Összetett típusok (compound types):
  - Tömbök,
  - Karakter sorozatok,
  - Mutatók,
  - Adatstruktúrák,
  - Dinamikus memória

Változók és típusok

Group	Type names*	Notes on size / precision
Character types	`char`	Exactly one byte in size. At least 8 bits.
	`char16_t`	Not smaller than `char`. At least 16 bits.
	`char32_t`	Not smaller than `char16_t`. At least 32 bits.
	`wchar_t`	Can represent the largest supported character set.
Integer types (signed)	`signed char`	Same size as `char`. At least 8 bits.
	`signed` `short` *`int`*	Not smaller than `char`. At least 16 bits.
	`signed` `int`	Not smaller than `short`. At least 16 bits.
	`signed` `long` *`int`*	Not smaller than `int`. At least 32 bits.
	`signed` `long long` *`int`*	Not smaller than `long`. At least 64 bits.
Integer types (unsigned)	`unsigned char`	(same size as their signed counterparts)
	`unsigned short` *`int`*	
	`unsigned` *`int`*	
	`unsigned long` *`int`*	
	`unsigned long long` *`int`*	
Floating-point types	`float`	
	`double`	Precision not less than `float`
	`long double`	Precision not less than `double`
Boolean type	`bool`	
Void type	`void`	no storage
Null pointer	`decltype(nullptr)`	

Változók és típusok

- A változók deklarálása kötelező használat előtt
- A változók a deklarálásuknak megfelelő hatókörben (scope) érhetők el

```
// operating with variables

#include <iostream>
using namespace std;

int main ()
{
  // declaring variables:
  int a, b;
  int result;

  // process:
  a = 5;
  b = 2;
  a = a + 1;
  result = a - b;

  // print out the result:
  cout << result;

  // terminate the program:
  return 0;
}
```

Változók és típusok

- A változók inicializáláskor kapnak meghatározott kezdő értéket:
  - C stílusú inicializáció
  - Konstruktor inicializáció
  - Uniform inicializáció

```
// initialization of variables

#include <iostream>
using namespace std;

int main ()
{
  int a=5;       // initial value: 5
  int b(3);      // initial value: 3
  int c{2};      // initial value: 2
  int result;    // initial value undetermined

  a = a + b;
  result = a - c;
  cout << result;

  return 0;
}
```

Változók és típusok

- Típus automatikus meghatározása:

```
int foo = 0;
auto bar = foo;  // the same as: int bar = foo;
```

- Nem inicializált változók típusának meghatározása:

```
int foo = 0;
decltype(foo) bar;  // the same as: int bar;
```

Változók és típusok

- Karakterláncok megadása összetett `string` típussal

```
// my first string
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string mystring;
  mystring = "This is a string";
  cout << mystring;
  return 0;
}
```

Változók és típusok

- Karakterláncok megadása összetett `string` típussal

```
// my first string
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string mystring;
  mystring = "This is the initial string content";
  cout << mystring << endl;
  mystring = "This is a different string content";
  cout << mystring << endl;
  return 0;
}
```

Konstansok

- A konstansok fix értékkel rendelkező kifejezések
- Konstansok típusai:
  - Literálok (literals)
    - Egész számok (integer numerals)
    - Lebegő pontos számok (floating point numerals)
    - Karakterek és karakterláncok (character and string literals)
  - Típusos konstans kifejezések (typed constant expressions)
  - Előfeldolgozó definíciók (preprocessor definitions)

Konstansok (egész szám literálok)

- Egész értékeket reprezentáló numerikus konstansok
- Decimális, oktális, hexadecimális kifejezések

```
75;          // decimal
0113;        // octal
0x4b;        // hexadecimal
```

- Alapértelmezetten `int` típusúak, de a típus módosítható

Suffix	Type modifier
u *or* U	unsigned
l *or* L	long
ll *or* LL	long long

```
75;          // int
75u;         // unsigned int
75l;         // long
75ul;        // unsigned long
75lu;        // unsigned long
```

Konstansok (lebegő pontos szám literálok)

- Valós érték kifejezések (tizedes vesszővel, kitevővel megadva)

```
3.14159;     // 3.14159
6.02e23;     // 6.02 x 10^23
1.6e-19;     // 1.6 x 10^-19
3.0;         // 3.0
```

- Alapértelmezetten double típusúak, de a típus módosítható

Suffix	Type
f *or* F	float
l *or* L	long double

```
3.14159L;    // long double
6.02e23f;    // float
```

Konstansok (karakter és karakterlánc literálok)

- Karaktereket és karakterláncokat egyszeres illetve dupla idézőjelekkel jelölünk

```
'z'
'p'
"Hello world"
"How do you do?"
```

- Speciális karaktereket is jelölhetünk a ’\’ segítségével

Escape code	Description
\n	newline
\r	carriage return
\t	tab
\v	vertical tab
\b	backspace
\f	form feed (page feed)
\a	alert (beep)
\'	single quote (')
\"	double quote (")
\?	question mark (?)
\\	backslash (\)

Konstansok (egyéb literálok)

- Az alábbi kulcsszavak (keywords) is literálok:
  - true/false: bool típus lehetséges értékei
  - nullptr: NULL mutató

```
bool foo = true;
bool bar = false;
int* p = nullptr;
```

Konstansok (típusos kifejezések)

- Sokszor kényelmes lehet különböző értékeknek nevet adni, amivel hivatkozhatunk rájuk a kód különböző részein

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

const double pi = 3.14159;
const char newline = '\n';

int main ()
{
  double r=5.0;           // radius
  double circle;

  circle = 2 * pi * r;
  cout << circle;
  cout << newline;
}
```

Konstansok (előfeldolgozó definíciók)

- Másik módszer konstans értékek elnevezésére
- `#define identifier replacement`
- A megadott azonosító helyére a helyettesítő érték az előfeldolgozó által kerül be a kódba a fordítás előtt (előfordulhatnak szintaktikai hibák)

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

#define PI 3.14159
#define NEWLINE '\n'

int main ()
{
  double r=5.0;               // radius
  double circle;

  circle = 2 * PI * r;
  cout << circle;
  cout << NEWLINE;

}
```

Operátorok

- Változókkal és konstansokkal különböző operátorok segítségével végezhetünk műveleteket
- Operátorok típusai:
  - Értékadó operátor (assignment operator)
  - Aritmetikai operátorok (arithmetic operators)
  - Összevont értékadás (compound assignment)
  - Növelés, csökkentés (increment, decrement)
  - Relációs, összehasonlító operátorok (relational, comparison operators)
  - Logikai operátorok (logical operators)
  - Feltételes operátor (conditional ternary operator)
  - Vessző operátor (comma operator)
  - Bitenkénti operátorok (bitwise operators)
  - Explicit típus konverzió operátor (explicit type casting operator)
  - `sizeof`

Operátorok (=)

- A jobb oldalon lévő kifejezés értékét rendeli hozzá a bal oldalhoz

- A jobb oldalon újabb értékadó operátor is állhat, az alábbi kifejezések is értelmesek:

```
y = 2 + (x = 5);
```

```
x = y = z = 5;
```

```
// assignment operator
#include <iostream>
using namespace std;

int main ()
{
  int a, b;           // a:?,  b:?
  a = 10;             // a:10, b:?
  b = 4;              // a:10, b:4
  a = b;              // a:4,  b:4
  b = 7;              // a:4,  b:7

  cout << "a:";
  cout << a;
  cout << " b:";
  cout << b;
}
```

Operátorok

- Aritmetikai operátorok: +, -, *, /, %

```
x = 11 % 3; // x = 2
```

- Összetett értékadó operátorok: +=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=
  - A változó értékét módosítják a reprezentált művelet elvégzésével: a művelet eredményét hozzárendelik a változóhoz

expression	equivalent to...
y += x;	y = y + x;
x -= 5;	x = x - 5;
x /= y;	x = x / y;
price *= units + 1;	price = price * (units+1);

```
// compound assignment operators
#include <iostream>
using namespace std;

int main ()
{
  int a, b=3;
  a = b;
  a+=2;              // equivalent to a=a+2
  cout << a;
}
```

Operátorok (++, --)

- Rövidített jelölések arra, hogy a változó értékét eggyel növeljük illetve csökkentsük
- Az operátorok használhatók előtagként (prefix) és utótagként (postfix) is:
  - Prefix esetben a változó értéke először növekszik és a megnövelt érték szerepel a kifejezés kiértékelésében
  - Postfix esetben a kifejezés kiértékelésekor az eredeti változó érték szerepel, de a változó ennek ellenére ugyanúgy növekszik eggyel

```
x = 3;
y = ++x;
// x contains 4, y contains 4
```

```
x = 3;
y = x++;
// x contains 4, y contains 3
```

Operátorok (==, !=, >, <, >=, <=)

- Relációs és összehasonlító operátorok segítségével kifejezések összehasonlíthatók
- Az összehasonlítás eredménye mindig true vagy false
- Gyakori hiba az = és az == összekeverése

```
(7 == 5)      // evaluates to false
(5 > 4)       // evaluates to true
(3 != 2)      // evaluates to true
(6 >= 6)      // evaluates to true
(5 < 5)       // evaluates to false
```

```
              //a=2, b=3 and c=6
(a == 5)      // evaluates to false, since a is not equal to 5
(a*b >= c)    // evaluates to true, since (2*3 >= 6) is true
(b+4 > a*c)   // evaluates to false, since (3+4 > 2*6) is false
((b=2) == a) // evaluates to true
```

Operátorok (!, &&, ||)

- Logikai operátorok:
  - Logikai tagadás: !
  - Logikai vagy: ||
  - Logikai és: &&

&& OPERATOR (and)		
a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

|| OPERATOR (or)		
a	b	a || b
true	true	true
true	false	true
false	true	true
false	false	false

- Short-circuit evaluation: a kifejezés balról jobbra értékelődik ki, csak addig a pontig, amikor már biztosan eldőlt a kifejezés értéke (igaz vagy hamis)
  - Figyeljünk ha tartalmaz pl. értékadást a kifejezés, nem biztos, hogy mindig meg fog történni!

```
!(5 == 5)   // evaluates to false because the expression at its right (5 == 5) is true
!(6 <= 4)   // evaluates to true because (6 <= 4) would be false
!true       // evaluates to false
!false      // evaluates to true
```

```
( (5 == 5) && (3 > 6) )  // evaluates to false ( true && false )
( (5 == 5) || (3 > 6) )  // evaluates to true ( true || false )

if ( (i<10) && (++i<n) ) { /*...*/ }   // note that the condition increments i
```

Operátorok (?)

- Feltételes operátor: kiértékel egy kifejezést (condition) és ha igaz akkor az első értékkel (result1) egyébként a második értékkel (result2) tér vissza
- `condition ? result1 : result2`

```
7==5 ? 4 : 3      // evaluates to 3, since 7 is not equal to 5.
7==5+2 ? 4 : 3    // evaluates to 4, since 7 is equal to 5+2.
5>3 ? a : b       // evaluates to the value of a, since 5 is greater than 3.
a>b ? a : b       // evaluates to whichever is greater, a or b.
```

```
// conditional operator
#include <iostream>
using namespace std;

  a=2;
  b=7;
  c = (a>b) ? a : b;

  cout << c << '\n';
}
```

Operátorok

- Vessző operátor: kettő vagy több kifejezés szeparálására szolgál oéyan helyen ahol csak egy kifejezés lehetne. Amikor ki kell értékelni a kifejezéseket csak a jobb oldali kifejezés eredménye számít

```
a = (b=3, b+2);     // a = 5, b = 3
```

- Bitenkénti operátorok: az egyes értékeket reprezentáló bitmintán végeznek különböző műveleteket

operator	asm equivalent	description
&	AND	Bitwise AND
|	OR	Bitwise inclusive OR
^	XOR	Bitwise exclusive OR
~	NOT	Unary complement (bit inversion)
<<	SHL	Shift bits left
>>	SHR	Shift bits right

Operátorok

- Explicit típus konverzió operátor: egy értéket a megadott típusáról egy másik típusra konvertál (ha lehetséges). Numerikus értékeknél előfordulhat adatvesztés

```
int i;
float f = 3.14;
i = (int) f;
```

- Sizeof: paraméterként egy típust vagy egy változót vár és visszaadja a típus vagy változó méretét bájtokban. A méret meghatározása mindig fordítási időben történik.

```
x = sizeof (char); // x = 1
```

Operátor precendencia

Level	Precedence group	Operator	Description	Grouping
1	Scope	`::`	scope qualifier	Left-to-right
2	Postfix (unary)	`++ --`	postfix increment / decrement	Left-to-right
		`()`	functional forms	
		`[]`	subscript	
		`. ->`	member access	
3	Prefix (unary)	`++ --`	prefix increment / decrement	Right-to-left
		`~ !`	bitwise NOT / logical NOT	
		`+ -`	unary prefix	
		`& *`	reference / dereference	
		`new delete`	allocation / deallocation	
		`sizeof`	parameter pack	
		`(type)`	C-style type-casting	
4	Pointer-to-member	`.* ->*`	access pointer	Left-to-right
5	Arithmetic: scaling	`* / %`	multiply, divide, modulo	Left-to-right
6	Arithmetic: addition	`+ -`	addition, subtraction	Left-to-right
7	Bitwise shift	`<< >>`	shift left, shift right	Left-to-right
8	Relational	`< > <= >=`	comparison operators	Left-to-right
9	Equality	`== !=`	equality / inequality	Left-to-right
10	And	`&`	bitwise AND	Left-to-right
11	Exclusive or	`^`	bitwise XOR	Left-to-right
12	Inclusive or	`|`	bitwise OR	Left-to-right
13	Conjunction	`&&`	logical AND	Left-to-right
14	Disjunction	`||`	logical OR	Left-to-right
15	Assignment-level expressions	`= *= /= %= += -= >>= <<= &= ^= |=`	assignment / compound assignment	Right-to-left
		`?:`	conditional operator	
16	Sequencing	`,`	comma separator	Left-to-right

Alapvető be-/kimenet (basic input/output)

- A különböző input, output műveletek elvégzésére stream-eket használ (absztrakció különböző források/célok (képernyő, billentyűzet, fájlok, stb.) felett)
- Egy stream olyan entitás amibe a program beszúrhat adatot (pl. karaktereket), illetve kiolvashat adatot. A be- illetve kiolvasás mindig szekvenciálisan történik
- Az absztrakció miatt nem szükséges ismerni, hogy pontosan honnan jön , illetve hova megy az adat, ezek a részletek el vannak rejtve
- Alapvető stream-ek:

stream	description
cin	standard input stream
cout	standard output stream
cerr	standard error (output) stream
clog	standard logging (output) stream

Sztenderd kimenet (standard output)

- A legtöbb esetben a sztenderd kimenet a képernyő. A programban a cout kifejezést és a beszúró operátort (<<) használjuk a kiíratásokhoz.
- A beszúró operátor beszúrja az őt követő adatot az őt megelőző stream-be. Az adat lehet literál, változó és egyéb kiértékelhető kifejezés, aminek az eredménye beszúrható a stream-be

```
cout << "Output sentence"; // prints Output sentence on screen
cout << 120;               // prints number 120 on screen
cout << x;                 // prints the value of x on screen
```

- Egy kifejezésbe egyszerre több beszúró operátor is láncolható
- Sortörést a `\n` vagy `endl` (a buffer ürítése megtörténik) szimbólumokkal szúrhatunk be

```
cout << "First sentence.\n";
```

```
cout << "Second sentence." << endl;
```

Sztenderd bemenet (standard input)

- A legtöbb esetben a sztenderd bemenet a billentyűzet. A programban a cin kifejezést és a kinyerő (extraction) operátort (>>) használjuk a bekérésekhez.
- Beolvasáskor a program várakozik a bementre, ha pl. a felhasználó begépelte az inputot, akkor egy ENTER leütése után küldi azt el a programhoz
- Beolvasáskor az operátor jobb oldalán lévő változó típusa határozza meg az input kezelését: ha pl. a változó típusa egész szám, akkor számokat vár az inputról, stb.
- Amennyiben a felhasználó nem a megfelelő típusú adatot írja be, akkor a kinyerés hibára fut és nem állítja be a változó értékét, ez nem kívánt működéshez vezethet, ezért mindig ellenőrizni kell a felhasználó által produkált bemenetet
- A kinyerő operátorok láncolhatók egy kifejezésen belül

```
cin >> a >> b;
```

```
// i/o example

#include <iostream>
using namespace std;

int main ()
{
  int i;
  cout << "Please enter an integer value: ";
  cin >> i;
  cout << "The value you entered is " << i;
  cout << " and its double is " << i*2 << ".\n";
  return 0;
}
```

Sztenderd bemenet (standard input)

- Bekéréskor a különböző szünet karakterek (whitespaces, tabs, new-line, stb.) jelölik az értékek határait
- Karakterláncok esetén ez problémás, mert így egy szöveget szavanként tudnánk csak beolvasni külön változókba
- Ennek kiküszöbölésére a `getline(instream, str)` metódust használhatjuk

```
// cin with strings
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string mystr;
  cout << "What's your name? ";
  getline (cin, mystr);
  cout << "Hello " << mystr << ".\n";
  cout << "What is your favorite team? ";
  getline (cin, mystr);
  cout << "I like " << mystr << " too!\n";
  return 0;
}
```

Sztenderd bemenet (standard input)

- Karakterláncok kezeléséhez használhatjuk még a `stringstream` típust, ami lehetővé teszi, hogy egy `string` típust stream-ként kezeljünk és így alkalmazhatjuk rá a beszúró/kinyerő operátorokat.
- Hasznos lehet ha pl. `string` értékekből kell numerikus értékeket kinyerni, így a kinyerés előtt ellenőrizni is tudjuk a felhasználói bementet

```
// stringstreams
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

int main ()
{
  string mystr;
  float price=0;
  int quantity=0;

  cout << "Enter price: ";
  getline (cin,mystr);
  stringstream(mystr) >> price;
  cout << "Enter quantity: ";
  getline (cin,mystr);
  stringstream(mystr) >> quantity;
  cout << "Total price: " << price*quantity << endl;
  return 0;
}
```

Feladatok

- Írjunk programot ami végrehajtja az alábbi lépéseket:
  - Bekéri a neved, születési éved, az előző év végi tanulmányi átlagod és egy rövid jellemzést magadról és eltárolja ezeket az adatokat egy-egy megfelelő típusú változóban,
  - A jelenlegi évet tárold el egy CURRENT_YEAR elnevezésű konstansban és ennek segítségével számold ki, hogy hány éves vagy/leszel idén, ezt is tárold el egy változóban,
  - Írasd ki a bekért és kiszámold adatokat a sztenderd kimenetre!
- Adott egy egész szám, ami kettő hatványa. A bitenkénti operátorok felhasználásával számold ki és írasd ki a szám felét illetve dupláját!