4. fejezet

Objektumok

Haladó Alkalmazott Programozás
ELTE, 2019

Objektumok

A C++ objektum orientált eszköztárának alapvető részei az objektumok:

- Az objektumok valamilyen szempont szerint közös használatú adatokat és függvényeket foglalnak össze egy új típussá.

Objektumok

A C++-ban objektumnak nevezzük a `struct`-okat és a `class`-okat is, és majdnem minden tekintetben ekvivalens a két nyelvi elem.

Amit kifejeznek: a `struct` csak egy gyűjtemény, kollekció, amelyben nincsnek olyan implicit összefüggések, amelyek sérülhetnének az elemek önkényes módosításával

A `class` esetében vannak ilyenek, ezért közvetlenül nem módosítandóak az adatok, hanem a `class` által biztosított interfészen keresztül adott műveletek elvégzésére van lehetőség.

Objektumok

Példák:

Az `std::pair` egy `struct`: két elemet tárol, de semmilyen logikai megkötés nincsen a kettő kapcsolatára, de fontos a flexibilitás, hogy könnyen el lehessen érni és módosítani az elemeket

Az `std::vector` egy `class`: van belső állapot (a heap memória pointer, és a méretek), amit ha valaki kívülről megpiszkál, teljesen elronthatja a memóriakezelési logikát.

Objektumok

Példa: írjunk egy objektumot:
egy 2 elemű matematikai vektort reprezentáló osztályt.

Elvárt funkcionalitás:

- Tárolja a két elemet
- Legyenek meg a vektoroktól elvárható matematikai műveletek
- Legyen néhány C++-os használathoz illeszkedő operátor

Objektumok

```
struct Vector2d
{
  double x, y;
};
```

Objektumok


```
struct Vector2d
{
  double x, y;
};
```

Ezt most így tudjuk használni:

```
Vector2d v = {1.0, 2.0};
v.x = 5.0;
std::cout << v.x << " " << v.y << "\n";
```

Objektumok - operátorok

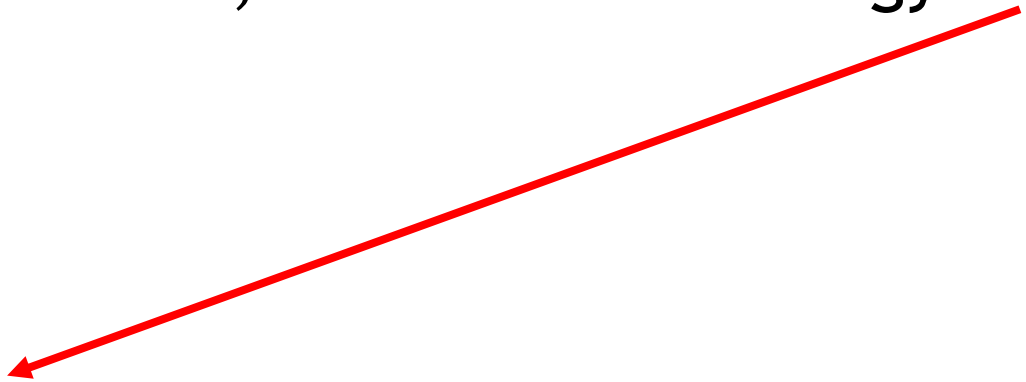
Vektorok összeadása: szeretnénk azt írni, hogy pl.: u+v

```
struct Vector2d
{
  double x, y;
};

Vector2d operator+( Vector2d const& a, Vector2d const& b )
{
  return Vector2d{ a.x + b.x, a.y + b.y };
}
```

Objektumok - operátorok

Az összeadás operátor, ugyan olyan függvény, mint bármelyik másik, csak a neve az hogy: operator+



```
Vector2d operator+( Vector2d const& a, Vector2d const& b )
{
  return Vector2d{ a.x + b.x, a.y + b.y };
}
```

Objektumok - operátorok

Most már írhatunk ilyet:

```
int main()
{
  Vector2d v = {1.0, 2.0};
  Vector2d u = {4.0, -2.0};
  auto w = v + u;
  std::cout << w.x << " " << w.y << "\n";
  return 0;
}
```

Objektumok - operátorok

Viszont ez még nem megy:

```
int main()
{
  Vector2d v = {1.0, 2.0};
  Vector2d u = {4.0, -2.0};
  v += u;
  std::cout << w.x << " " << w.y << "\n";
  return 0;
}
```

Objektumok - operátorok

Az összevont (aritmetikai művelet ÉS értékadás) operátorokat másképp kell bevezetni: az objektumon belül, és 1 argumentumos változatban (a másik argumentum maga az objektum!):

```
struct Vector2d
{
  double x, y;

  Vector2d& operator+=( Vector2d const& v )
  {
    x += v.x;  y += v.y;
    return *this;
  }
};
```

Objektumok - operátorok

Mivel bent vagyunk az objektum scope-jában, látjuk a változóit, x-et és y-t.

Ezekhez hozzáadjuk a másik vektor elemeit.

```
struct Vector2d
{
  double x, y;

  Vector2d& operator+=( Vector2d const& v )
  {
    x += v.x;  y += v.y;
    return *this;
  }
};
```

Objektumok - operátorok

A művelet eredménye „saját magunk”;
A `this` saját magunkra ad vissza egy pointert, amit azonnal dereferálunk és referenciaként adjuk vissza.

```
struct Vector2d
{
  double x, y;

  Vector2d& operator+=( Vector2d const& v )
  {
    x += v.x;  y += v.y;
    return *this;
  }
};
```

Objektumok - operátorok


```
struct Vector2d
{
  double x, y;

  Vector2d& operator+=( Vector2d const& v )
  {
    x += v.x;  y += v.y;
    return *this;
  }
};
```

Ha nem referenciát adunk vissza, hanem simán `Vector2d`-t, akkor nem írhatunk ilyet:

```
u += v += w
```

Objektumok - operátorok

Az előzőekhez hasonlóan definiálhatóak a további műveletek is:

- Az osztályon belül:
  - `operator-=`
  - `operator*=` (skalárral szorzás)
  - `operator/=` (skalárral osztás)
- Az osztályon kívül pedig:
  - `operator-`
  - `operator*` (skalárral szorzás, jobbról és balról külön-külön be kell vezetni!)
  - `operator/` (skalárral osztás, csak jobbról van értelme)

Objektumok - operátorok

A további hasznos műveleteket érdemes beszédes függvény nevekkel és nem operátorokkal bevezetni.

Néhány példa, csak a szignatúrákkal:

```
double dot(Vector2d const&, Vector2d const&);
double length(Vector2d const&);
double sqlength(Vector2d const&);
Vector2d normalize(Vector2d const&);
```

Objektumok - operátorok

C++-ban nagyon hasznos, ha tudjuk szöveggé alakítani, illetve szövegből beolvasni az objektumunkat!

Ezeket csinálják a << és >> operátorok!

Objektumok - operátorok

```
std::ostream& operator<<( std::ostream& o, Vector2d const& v )
{
  o << v.x << "   " << v.y;
  return o;
}

std::istream& operator>>( std::istream& i, Vector2d& v )
{
  i >> v.x;
  i >> v.y;
  return i;
}
```

Objektumok - operátorok

A szöveggé alakító operátor kötelezően egy std::ostream&-et vár (a stream módosul kiírás közben!), a kiírandó objektum nem változik, ezért const, a visszatérési érték ugyan az a stream, ami bejött, hogy lehessen láncolni (pl.: std::cout << u << v << w).

```
std::ostream& operator<<( std::ostream& o, Vector2d const& v )
{
  o << v.x << "   " << v.y;
  return o;
}
```

Objektumok - operátorok

A beolvasó operátor kötelezően egy std::istream&-et vár (a stream módosul olvasás közben!), a kiírandó objektum IS változik, ezért nincs const, a visszatérési érték ugyan az a stream, ami bejött, hogy lehessen láncolni (pl.: std::cin >> u >> v >> w).

```
std::istream& operator>>( std::istream& i, Vector2d& v )
{
  i >> v.x;
  i >> v.y;
  return i;
}
```

Objektumok - operátorok

Példa: program, ami bekér két vektort és összeadja őket:

```
int main()
{
  Vector2d v1{}, v2{};
  std::cout << "Enter vector 1:\n";
  std::cin >> v1;
  std::cout << "Enter vector 2:\n";
  std::cin >> v2;
  std::cout << "The sum of the two vectors is:\n" << v1+v2;
  return 0;
}
```

Objektum templatek

Objektum templatek

A valóságban a vektorunkban sokféle elemtípust szeretnénk használni, de nem szeretnénk mindegyikre újra implementálni az összes műveletet.

Például: szükségünk lehet egész elemű vektorokra, `float`, `double` elemekre, de akár komplex elemű vektorokra is.

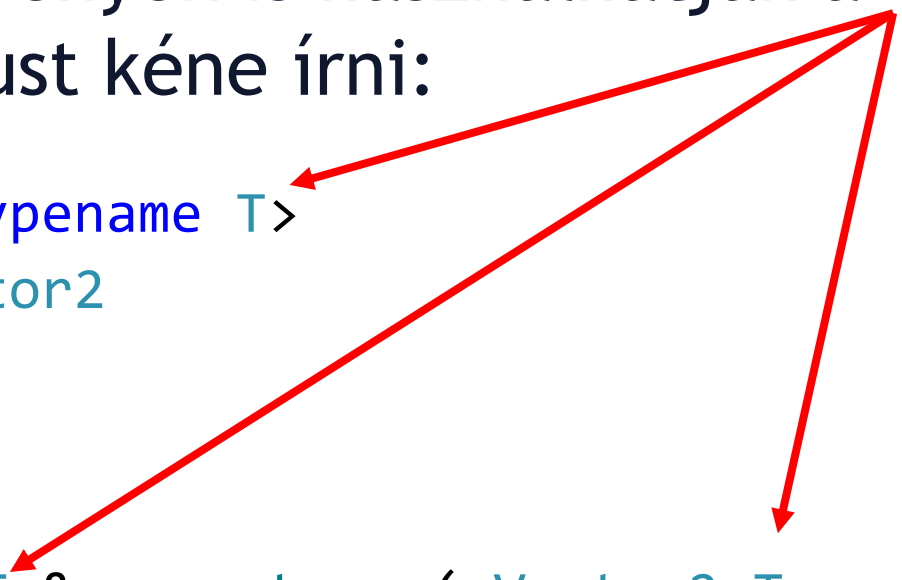
Objektum templatek

Minimális változtatással megvalósíthatjuk ezt a vector2 objektumunkon:

```
template<typename T>
struct Vector2
{
  T x, y;
};
```

Objektum templatek

A tagfüggvények is használhatják a T típust minden helyen, ahova típust kéne írni:



```
template<typename T>
struct Vector2
{
  T x, y;

  Vector2<T>& operator+=( Vector2<T> const& v )
  {
    x += v.x;  y += v.y;
    return *this;
  }
};
```

Objektum templatek

A globális függvényeknél/operátoroknál újra be kell vezetni a `T` típust minden egyes definíciónál külön-külön:

```
template<typename T>
Vector2<T> operator+( Vector2<T> const& a, Vector2<T> const& b )
{
  return Vector2<T>{ a.x + b.x, a.y + b.y };
}
```

Vektor osztály

Feladat / házi feladat:

Készítsük el a template-s 2 elemű vektor osztályt egy külön header-be (vector2.h) a teljes funkcionalitással, ami a diasorban említésre került
(vagy amit még úgy gondolunk, hogy hasznos lehet).

Minden funkcióra legyen egy ellenőrzés a main.cpp-ben!