A Programozás Alapjai 2
Objektumorientált szoftverfejlesztés

Dr. Forstner Bertalan
forstner.bertalan@aut.bme.hu

BME AUT
Department of Automation and Applied Informatics

Generikus típusok

Generikus típus

- Generikus vagy paraméterezett típus
  - C++-ban template-nek hívják
- Típus, ami nem teljes, csak ha bizonyos paraméterek meg vannak adva használatkor
- Fogalmi szinten megérteni nehéz, ezért nézünk példákat bőven.

Függvény template-ek

- Írjunk egy max függvényt, ami *tetszőleges típusra* működik
  - pl double-ra, Complex-re, String-re is
  - *Legyen hatékony* és *biztonságos!*
- *3 megoldási ötlet:*

- Írjunk egy max függvényt, ami *tetszőleges típusra* működik
  - pl double-ra, Complex-re, String-re is
  - *Legyen hatékony* és *biztonságos!*
- *3 megoldási ötlet:*
  - *(1)* Makróval sok probléma lehet, láttuk korábban

Inline függvénnyel

- (2) Inline függvény: meg kell írni minden típusra külön-külön

```
inline const Complex& max(const Complex lhs,
const Complex rhs) {
    return lhs>rhs ? lhs: rhs;
}
```

Inline függvénnyel

- (2) Inline függvény: meg kell írni minden típusra külön-külön

```
inline const Complex& max(const Complex lhs,
const Complex rhs) {
    return lhs>rhs ? lhs: rhs;
}
```

- Természetesen csak akkor működik, ha az operator> meg van írva a Complex-re
  - >(pl. a nagyobb abszolút értéket nézi).

- A függvény törzse ugyanígy néz ki minden típusra.
- Jó lenne, ha tudnánk írni egy olyan függvény öntőformát, amiben a típus, amin dolgozik egy paraméter, nincs rögzítve, hanem amikor használjuk, akkor lehetne megadni.
- Erre szolgál a (3) függvény template.
  - Példa

Függvény template

- A ***class*** kulcsszó, lehet **typename**-et is, ~ekvivalens

  template <typename T>

- A T paraméter **nevet mi adtuk neki**
- Amikor használjuk, természetesen a template paraméterek egy adott típussal értéket kapnak
- **Példa** a használatra

A példa

```
template <class T>
inline T max(T lhs, T rhs)
{
   return lhs > rhs ? lhs : rhs;
}

int main(int argc, char* argv[]) {
   int a = 5;
   int b = 6;
   cout << "The winner is " << max<int>(a, b) << endl;
   double c = 3.14159265359;
   double d = 4.28318530618;
   cout << "The winner is " << max<double>(c, d) << endl;

   //A fordito kitalalja a tipust:
   cout << "The winner is " << max(c, d) << endl;
}
```

Feltételek támasztása

- Feltételeket támasztunk a T paraméterre, amit a forráskód nem kényszerít ki!
- Jelen esetben: akkor működik, ha a T típusra létezik illetve meg van írva az operator>.
- **Példa**: Complex és operator>

Konverzió és a template-ek

- Template argumentumok közt nincs minimális konverzió sem
- A nézett példában a

```
cout << "The winner is " << max(a,d) << endl;
```

nem fordul le, mert konverzióra lenne szükség. (a: int, d: double)

- Ha nem változók állnának, akkor sem, ez egy szigorú szabály a template függvényekre.
- A hibaüzenet elég pontosan megmondja a baját (vagy int, vagy double legyen a T).

Konverzió és a template-ek

- Két megoldás:

```
cout << "The winner is " << max((double)a,d) << endl;
cout << "The winner is " << max<double>(a,d) << endl;
```

- Utóbbinál **explicit példányosítjuk** a template függvényt.
  - A max<double> már nem template, hanem egy teljesen közönséges, normál függvény
  - Rá már azok a szabályok érvényesek, melyek a közönséges függvényekre

Code bloat - kódburjánzás

- Bármilyen típussal használjuk, példányosodni fog, nagyon megnőhet a kód mérete.
  - Pl. int és a long is különböző, legenerálódik mindkettőre, hiába van automatikus konverzió.
- Megoldás lehet, ha kiírjuk, hogy melyiket használjuk:

```
max<int>(a, b);
```

  - Itt ha ***a*** és ***b*** long vagy char, akkor sem fog külön generálni neki függvényt, hanem az int-eset használja konverzióval.

Explicit specializáció

- A template függvényeket explicit specializálni (felül lehet definiálni) lehet adott típusokra.
- Akkor kell megírni, ha egy adott típusra az általános verzió nem működne.

- Példa

```
template<>
inline const char* max(const char* lhs, const char* rhs)
{
   return strcmp(lhs, rhs) > 0 ? lhs : rhs;
}
```

```
cout << "Max " << max("Gyula", "Payne") << endl;
```

Argumentum egyeztetés

- Milyen prioritással veszi figyelembe a függvény kiválasztásnál a lehetőségeket?
  1. Ha van olyan függvény, aminek az argumentumai pontosan egyeznek típusban, akkor azt választja ki.
  2. Template-et keres, van-e, ami pontosan egyezik.
  3. Automatikus konverzióval egyezik, de nem lehet template a konvertálandó paraméter.

Több template paraméter

- Lehet több template paramétere is a függvénynek
  - vesszővel elválasztva,
  - olyan, mint a függvények argumentum listája, csak itt típus is lehet paraméter.
  - Függvényen belül fel tudjuk használni ezeket.
- **Példa** árfolyamváltó függvény. Nem tudom, mi lesz a konverziós ráta, illetve az összeg típusa

Több template paraméter

- Lehet több template paramétere is a függvénynek
  - vesszővel elválasztva,
  - olyan, mint a függvények argumentum listája, csak itt típus is lehet paraméter.
  - Függvényen belül fel tudjuk használni ezeket.
- **Példa** árfolyamváltó függvény. Nem tudom, mi lesz a konverziós ráta, illetve az összeg típusa

```
template<typename RATE_TYPE, typename AMOUNT_TYPE>
AMOUNT_TYPE convert(AMOUNT_TYPE sum, RATE_TYPE exchange_rate)
{
        return sum*exchange_rate;
}
```

Parametrizált vagy generikus osztályok

Példa

- Az int-es Stack osztályunk átalakítása

Mi lehet template paraméter?

1. Típus
2. Típusos konstans
3. Template

- Példák

```
template <int DIM>
class Hypercube {
   const int dimensions;
   in origo[DIM];
public:
   Hypercube() : dimensions(DIM) {}
};
```

```
Hypercube<2> square();
const int haromde = 3;
Hypercube<haromde> cube();
```


Intermission

Default template argumentumok

- Ugyanazok a szabályok vonatkoznak rá, mint a függvények default paramétereinél
- Példa

Default template argumentumok

- Ugyanazok a szabályok vonatkoznak rá, mint a függvények default paramétereinél
- Példa

```
template <int DIM=3>
class Hypercube {
   const int dimensions;
   in origo[DIM];
 public:
   Hypercube<>() : dimensions(DIM) {}
};
```

Tagfüggvény template-ek (member templates)

- Olyan függvény template, ami egy tagfüggvény.
  - Annyi verzió generálódik automatikusan, ahány különböző paraméterrel használom.
  - Az osztálynak az adott típus nem paramétere
  - Ezt is lehet specializálni, stb.
  - Természetesen közönséges osztályra is működik.
  - Lehet több ilyen tagfüggvény is, mindben lehet használni ugyanazokat a paraméter neveket
- Példa

```
template <int DIM>
template<class T>
T Hypercube<DIM>::getVerticesCount()
{
    T result = pow(2, dimensions);
    return result;
}
```

Template-ek és öröklés

- Egy szabály: csak osztályból lehet leszármaztatni, template-ből nem.
- Template valami, amíg van legalább egy nem rögzített paramétere. Ha nincs, akkor már osztály.

Template-ek és öröklés

- **Példák** a leszármazási típusokra:
  - Legyen egy termék **sablon** osztályunk:
    - Még nem tudjuk, hogy az őt használó komponens hogyan fogja azonosítani a terméket: **ID_TYPE**
    - Illetve, ohgy a termék lejárati dátuma hogyan kerül bele (timestamp? Date objektum?): **DATE_TYPE**

- **A Példa**

Template-ek és öröklés összefoglalás

1. Az ős template-ből példányosított osztály (tehát nem template!), leszármazott közönséges osztály
2. Egy osztály alapján leszármazott template-et hozunk létre
3. Az ős template-ből példányosított osztály (tehát nem template!), a leszármazott template osztály
4. Template paraméterben megadott ősosztály

Template-ek fordítása

- A template-ek fordítási időben fejtődnek ki
- a template-eket addig le sem fordítja a fordító, amíg nem használjuk valahol
  - > pl. így: vector<int> v;
  - > Amilyen paraméterekkel használjuk, azokkal külön-külön lefordítja.
- A metódusokat is csak akkor fordítja le, ha használjuk őket (bár szintaktikai ellenőrzést végezhet (nem szoktak), ezt nem írja elő a szabvány).
- Emiatt a hibák nem derülnek ki fordításkor, ha nem használjuk
  - > Azt hiszem, hogy jól megírtam, majd ha egyszer használni szeretném, csak akkor derül ki, hogy tele van fordítási hibával.

Template specializáció

- Lehet osztálynál is, a függvényekhez hasonló módon

Explicit példányosítás

- ..hasonló módon, mint a függvény template-eknél

Jótanács

- Ha template-et használunk, nem tudjuk milyen template paraméterrel fogják használni: beépített vagy nem beépített típussal.

```
template<class T>
f(T par) {
   …
}
```

- Ez pl. int-re tökéletes, de pl. ha a T egy nagyobb osztály, akkor meghívódik a másoló konstruktor: lassú lesz (pl. stack objektum...).
- Ehelyett célszerűbb általában:

```
template<class T>
f(const T& par) {
   …
}
```

Öröklés vagy template?

- **Mikor használunk öröklést és mikor template-eket?**

- A template-nél: ugyanaz a viselkedés több típusra.
- Öröklés: a leszármazottakban felüldefiniálható a viselkedés (az egyes függvényekre).