TDP004 - Objektorienterad programmering

Typkonvertering, self assignment

Pontus Haglund & Rasmus Jonsson

Institutionen för datavetenskap

LINKÖPING UNIVERSITY

1. Mål med föreläsningen
2. Typkonvertering
   - Konstruktorer
   - Operatorer
   - explicit vs implicit
3. cast
   - c++ static_cast
   - dynamic_cast
4. c style casts
5. self assignment
6. type aliasing
7. Kompileringsfel

1. Mål med föreläsningen
2. Typkonvertering
   - Konstruktorer
   - Operatorer
   - explicit vs implicit
3. cast
   - c++ static_cast
   - dynamic_cast
4. c style casts
5. self assignment
6. type aliasing
7. Kompileringsfel

Mål med föreläsningen

Efter föreläsningen skall studenten kunna:

- typkonverter
- implementera eget stöd för konvertering
- förstå implicit kontra exlicit konvertering
- repetition av self assignment

1 Mål med föreläsningen
2 Typkonvertering
Konstruktorer
Operatorer
explicit vs implicit
3 cast
c++ static_cast
dynamic_cast
4 c style casts
5 self assignment
6 type aliasing
7 Kompileringsfel

LINKÖPING UNIVERSITY

Konstruktorer

A constructor that is not declared with the specifier explicit (and which can be called with a single parameter (until C++11)) is called a converting constructor.
- cppreference

Konstruktorer

```
class C
{
public:
    C() : i{0} {};                  //Converting
    C(int i) : i{i} {};             //Converting
    C(int i, int j) : i{i+j} {}; //Converting

    void print() { cout << i << endl; }

private:
    int i;
};
```

Konstruktorer

```
void foo(C c)
{
    c.print();
}

int main()
{
    foo(3);
    foo({1, 3});
    foo({});
}
```

Konstruktorer

```
C bar()
{
    return 5;
}

int main()
{
  bar().print();
  C c{bar()};
  c.print()
}
```

Konstruktorer

```
C bar()
{
    return {5, 6};
}

int main()
{
  bar().print();
  C c{bar()};
  c.print()
}
```

Konstruktorer

```
C bar()
{
    return {};
}

int main()
{
  bar().print();
  C c{bar()};
  c.print()
}
```

Konstruktorer

- Detta är alltså inte alltid ett bra beteende.
- Nyckelordet explicit stänger av funktionaliteten

explicit

```
class C
{
public:
    explicit C() : i{0} {};
    explicit C(int i) : i{i} {};
    explicit C(int i, int j) : i{i+j} {};

    void print() { cout << i << endl; }
private:
    int i;
};
```

Ett riktigt exempel...

- Vad hade du för konstruktorer i listlabben?
- Vad hade du för konstruktorer i simulatorn?
- Finns det några tänkbara problem?

Typkonverterande operatorer

Enables implicit conversion or explicit conversion from a class type to another type.
- cppreference

Typkonverterande operatorer

```
class C
{
public:
    explicit C() : i{0} {};
    //...
    operator string()
    {
      return to_string(i);
    }

private:
    int i;
};
```

Typkonverterande operatorer

```
void greet(string const& name)
{
    cout << "Hello " << name << endl;
}

int main()
{
    C c{5};
    greet(c);
}
```

Typkonverterande operatorer

- Liknande problem som konstruktorn (fast omvänt)
- Löses på samma sätt

Typkonverterande operatorer

```
class C
{
public:
    explicit C() : i{0} {};
    //...
    explicit operator string()
    {
      return to_string(i);
    }

private:
    int i;
};
```

Implicit kontra Explicit konvertering

- Nyckelordet explicit hindrar oss inte från att uföra konvertering
- Bara från att göra implicit konvertering
- Så hur gör man explicit konvertering?
  - Antingen genom att explicit anropa konstruktorn
  - Eller genom att **casta** till en annan typ

Time

```
class Time
{
public:
    explicit Time(int s) : s{s} {}

    explicit operator string() const
    {
        stringstream strs{};
        int m{s/60};
        int h{m/60};
        strs << setfill('0')
             << setw(2) << h << ":"
             << setw(2) << (m % 60) << ":"
             << setw(2) << (s % 60);
        return strs.str();
    }
private:
    int s;
};
```

```
int main()
{
    Time t{3665};
    cout << string(t) << endl;
}
```

1 Mål med föreläsningen
2 Typkonvertering
   Konstruktorer
   Operatorer
   explicit vs implicit
3 cast
   c++ static_cast
   dynamic_cast
4 c style casts
5 self assignment
6 type aliasing
7 Kompileringsfel

LINKÖPING UNIVERSITY

casts

Använd dem inte om det inte är nödvändigt.

static_cast

- Compile time
- Konvertera en typ till en annan
- Bara om det är känt hur den konverteringen skall gå till
- Får inte ta bort const-ness

static_cast primitives

```
int main()
{
    char c{'i'};
    int i{static_cast<int> (c)};
    double d{};
    d = static_cast<double> (c);
    cout << "c: " << c << '\n'
         << "i: " << i << '\n'
         << "d: " << d << endl;
}
```

static_cast user-defined

```
int main()
{
  Time t{3665};
  string s{static_cast<string> (t)};
  cout << s << endl;
  // Hur är denna konvertering känd?
}
```

dynamic_cast

Safely converts pointers and references to classes up, down, and sideways along the inheritance hierarchy.
-cppreference

dynamic_cast

- Säker typkonvertering i arvsstrukturer
- Run-time **måste kontrolleras**
- Får ut pekare
- eller nullptr
- eller kastar ett fel
- Ett undantagsfall, fel design?

dynamic_cast

```
int main()
{
    Fighter f{"Janos", 4};
    Archer a{"Pontus", 4, 3};
    vector<Character*> characters{};
    characters.push_back(&f);
    characters.push_back(&a);

    for ( Character* c : characters )
    {
        Archer* ap{dynamic_cast<Archer*> (c)};
        if(ap) //Om inte nullptr
            ap -> shoot();
    }
}
```

Fler casts

- const_cast
  - compile och run
- reinterpret_cast
  - run
- Mycket riskabla, använd inte (utom när man ska...)

1 Mål med föreläsningen
2 Typkonvertering
   Konstruktorer
   Operatorer
   explicit vs implicit
3 cast
   c++ static_cast
   dynamic_cast
4 c style casts
5 self assignment
6 type aliasing
7 Kompileringsfel

LINKÖPING UNIVERSITY

c style

- c style casts stöter man ibland på
- de fungerar men är föråldrade och bör inte användas
- varför är det på detta viset?

c style

```
#include <iostream>

using namespace std;

int main()
{
    char c{'i'}; //105e ascii
    int i = c; //implicit
    double d = (double) c; //explicit
    cout << "c: " << c << '\n'
         << "i: " << i << '\n'
         << "d: " << d << endl;
}
```

c style

```
int main()
{
    Time t{3665};
    string s = (string) t;
    cout << s << endl;
}
```

c style

```
int main()
{
    int i{5};
    int* ip{&i};

    char* cp = (char*) ip;
    cout << *ip << '\n'  // skriver ut: 5
         << *cp << endl; // skriver ut:
}
```

problem i runtime

c style

```
int main()
{
    Fighter f{"Janos", 4};
    Archer a{"Pontus", 4, 3};
    vector<Character*> characters{};
    characters.push_back(&f);
    characters.push_back(&a);

    for ( Character* c : characters )
    {
        // Archer* ap{dynamic_cast<Archer*> (c)};
        Archer* ap = (Archer*) c;
        if(ap)
        {
            ap -> shoot();
        }
    }
}
```

c style

- Inte lika specifikt som c++ stilen
- Inte lika sökbart

1 Mål med föreläsningen
2 Typkonvertering
   Konstruktorer
   Operatorer
   explicit vs implicit
3 cast
   c++ static_cast
   dynamic_cast
4 c style casts
5 self assignment
6 type aliasing
7 Kompileringsfel

LINKÖPING UNIVERSITY

Self assignment

- Ibland så kan ett objekt tilldelas sig själv
- Med andra ord kan ett objekt kopieras till sig själv
- Hur kan detta uppstå och vad kan det få för effekt?

Self assignment, när kan det uppstå?

```
Sorted_List&
select_longest_list(Sorted_List& a, Sorted_List& b)
{
  if ( a.size() > b.size() )
    return a;
  return b;
}

TEST_CASE( "Self assignment" )
{
  List k{1,2,3};
  List l{4,5};
  k = select_longest_list(k, l);
  //...
}
```

1 Mål med föreläsningen
2 Typkonvertering
   Konstruktorer
   Operatorer
   explicit vs implicit
3 cast
   c++ static_cast
   dynamic_cast
4 c style casts
5 self assignment
6 type aliasing
7 Kompileringsfel

type aliasing

```
#include <cstdint>
using hour = uint8_t;
using minute = uint8_t;
using second = uint8_t;

class Time
{
//...
private:
hour h;
minute m;
second s;
};
```

1 Mål med föreläsningen
2 Typkonvertering
   Konstruktorer
   Operatorer
   explicit vs implicit
3 cast
   c++ static_cast
   dynamic_cast
4 c style casts
5 self assignment
6 type aliasing
7 Kompileringsfel

LINKÖPING UNIVERSITY

Kompileringsfel

```
Typografiska //syntax
Kompileringsfel //semantik
Länkningsfel //något saknas i länkningen
Runtime //Något går fel runtime
Minnesfel //Inkorrekt hantering av minnet
```

www.liu.se

LINKÖPING UNIVERSITY