TDP004 - Objektorienterad programmering

Templates och smartpekare

Pontus Haglund & Rasmus Jonsson

Institutionen för datavetenskap

LINKÖPING UNIVERSITY

1. Mål med föreläsningen
2. Kompilatorns steg
3. Funktionsmallar
4. Klassmallar
5. Deducera typ
6. Smartpekare
7. Funktioner som parametrar

1 Mål med föreläsningen
2 Kompilatorns steg
3 Funktionsmallar
4 Klassmallar
5 Deducera typ
6 Smartpekare
7 Funktioner som parametrar

Mål med föreläsningen

Efter föreläsningen skall studenten kunna:

- Kompilatorns olika steg
- Använda funktionsmallar
- Använda klassmallar
- Använda smartpekare

1 Mål med föreläsningen
2 Kompilatorns steg
3 Funktionsmallar
4 Klassmallar
5 Deducera typ
6 Smartpekare
7 Funktioner som parametrar

Kompilatorns steg

1. Preprocessor
2. Kompilering
3. Länkning

1 Mål med föreläsningen
2 Kompilatorns steg
3 Funktionsmallar
4 Klassmallar
5 Deducera typ
6 Smartpekare
7 Funktioner som parametrar

Funktionsmallar

Vad är en mall

```
int main()
{
  max(5, 6);
  max(5.0, 6.0);
}
```

Funktionsmallar

Vad är en mall

```
int max(int a, int b)
{
  if( a < b )
  {
    return b;
  }
  else
  {
    return a;
  }
}
```

```
double max(double a, double b)
{
  if( a < b )
  {
    return b;
  }
  else
  {
    return a;
  }
}
```

Funktionsmallar

DRY

Don’t Repeat Yourself

Funktionsmallar

Enkel mall

```
template <typename T>
T max(T a, T b)
{
  if( a < b )
  {
    return b;
  }
  else
  {
    return a;
  }
}
```

Funktionsmallar

Olika anrop

```
int main()
{
  int i{2};
  int j{3};
  my_max(i, j);
  my_max<int>(i, j);
}
```

1 Mål med föreläsningen
2 Kompilatorns steg
3 Funktionsmallar
4 Klassmallar
5 Deducera typ
6 Smartpekare
7 Funktioner som parametrar

Klassmallar

Vad är en klassmall

```
int main()
{
  Stack my_int_stack{1, 2, 3};
  Stack my_float_stack{1.0, 2.0, 3.0};
}
```

Klassmallar

Hur ser en klassmall ut

```
int main()
{
  Stack my_int_stack{1, 2, 3};
  Stack my_float_stack{1.0, 2.0, 3.0};
}
```

Klassmallar

Hur ser en klassmall ut

```
template <typename T>
class Stack
{
public:
  //...
private:
  struct Node
  {
    Node* next;
    T data;
  };
  //...
};
```

Klassmallar

tcc-filer

```
//h-fil
template <typename T>
class Stack
{
  Stack();
  void push(T data);
  T pop();
  //...
};

#include "Stack.tcc"
```

```
//tcc-fil
template <typename T>
void Stack<T>::push(T data)
{
  //...
}

template <typename T>
T Stack<T>::pop()
{
  //...
}
```

Klassmallar

Flera mallparametrar

```
template <typename Value_type, typename Key_type = int>
class Stack
{
public:
  //...
private:
  //...
  class Node
  {
    Node(Node* n, Key_type k, Value_type v);
    Node* next;
    Key_type key;
    Value_type value;
  };
};
```

Klassmallar

tcc-filer

```
//h-fil
template <typename Value_type, typename Key_type = int>
class Stack
{
  Stack();
  void push(Key_type key, Value_type value);
  pair<Key_type, Value_type> pop();
  //...
};

#include "Stack.tcc"
```

Klassmallar

tcc-filer

```
//tcc-fil
template <typename Value_type, typename Key_type>
void Stack<Value_type, Key_type>::push(Key_type key,
                                       Value_type value)
{
  //...
}

template <typename Value_type, typename Key_type>
pair<Key_type, Value_type> Stack<Value_type, Key_type>::pop()
{
  //...
}
```

1 Mål med föreläsningen
2 Kompilatorns steg
3 Funktionsmallar
4 Klassmallar
5 Deducera typ
6 Smartpekare
7 Funktioner som parametrar

Deducera typer - value type

Är ett sätt att **beräkna** typen av ett objekt.

```
template<typename T>
typename T::value_type first(T const& container)
{
  return *begin(container); //iterator
}
```

```
vector<int> vi{1, 2, 3};
vector<string> vs{"hej", "svej"};

int i{first(vi)};
int s{first(vs)};
```

1 Mål med föreläsningen
2 Kompilatorns steg
3 Funktionsmallar
4 Klassmallar
5 Deducera typ
6 Smartpekare
7 Funktioner som parametrar

LINKÖPING UNIVERSITY

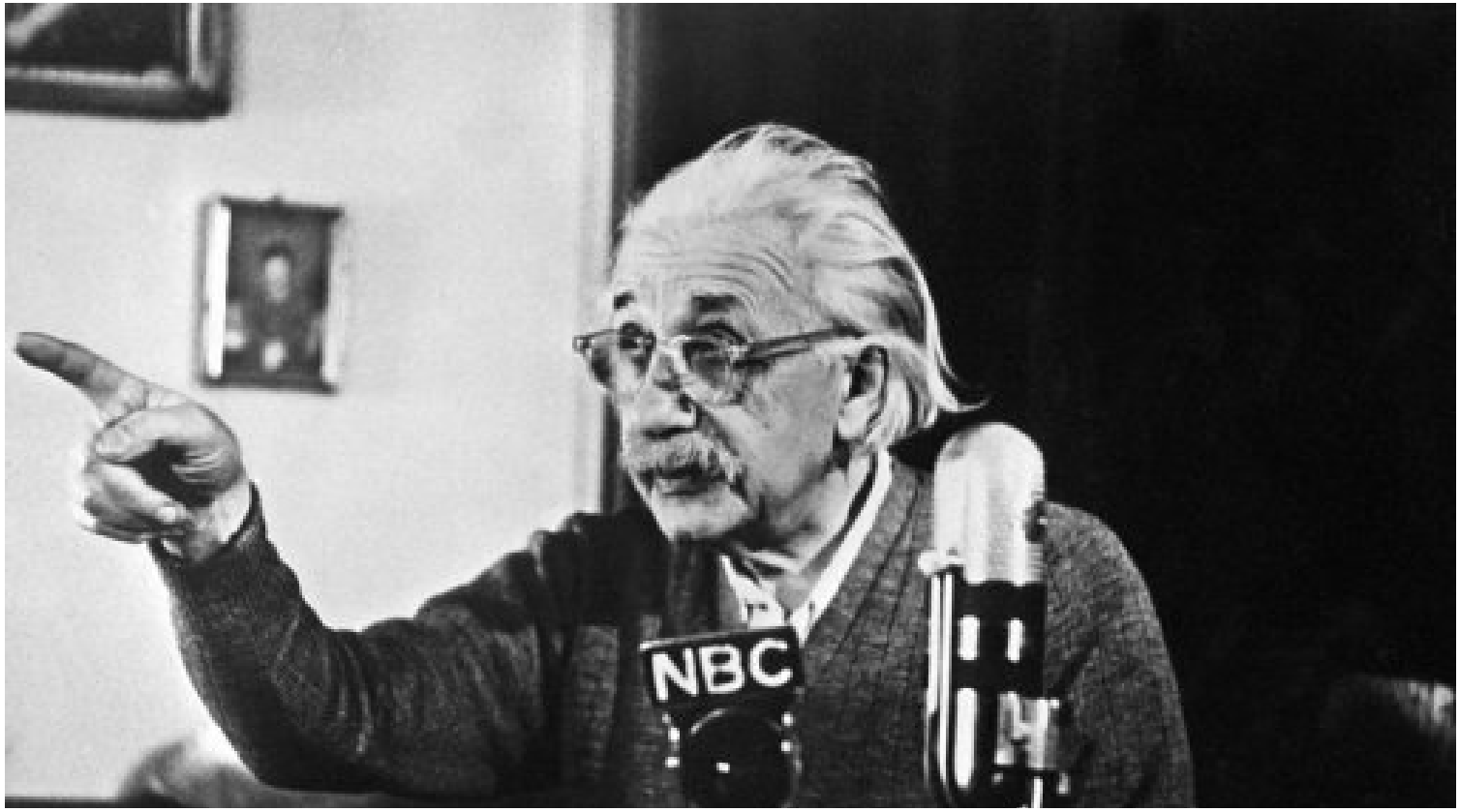
Smartpekare

Vad är fel?

```
int main()
{
  int* my_ptr { new int{5} };
}
```

Smartpekare

Smartpekare



Smartpekare

unique_ptr

```
int main()
{
  unique_ptr<int> my_uptr{ new int{5} };
}
```

Smartpekare

Kan INTE kopieras

```
void print(unique_ptr<int> up)
{
  cout << *up << endl;
}

int main()
{
  unique_ptr<int> uptr{ new int{5} };
  print( move(uptr) );
}
```

Smartpekare

shared_ptr

```
int main()
{
  shared_ptr<int> my_sptr1{ new int{5} };
  {
    shared_ptr<int> my_sptr2{ my_sptr1 };
  }
}
```

1 Mål med föreläsningen
2 Kompilatorns steg
3 Funktionsmallar
4 Klassmallar
5 Deducera typ
6 Smartpekare
7 Funktioner som parametrar

LINKÖPING UNIVERSITY

Funktioner som parametrar

```
void bar(int x){ cout << x << endl; }

template <typename F>
void callback(F function, int i)
{
  function(i);
}

int main()
{
  callback(bar, 4);
  callback([](int x) {cout << x+5 << endl;}, 3);
}
```

Template fel

```
passfun.cc: In function ‘int main’():
... no matching function for call to‘
        callback(void (&)(int), int’)
   15 |   callback(bar, 4);
      |                    ^
... candidate:‘
template<class F, class T> void callback(F, int’)
    8 | void callback(F function, int i)
      |      ^~~~~~~~
... template argument deduction/substitute failed:
... ’couldnt deduce template parameter ‘’T
   15 |   callback(bar, 4);
```

www.liu.se

LINKÖPING UNIVERSITY