



Assignment 4 - metaprogramming

Generiske multidimensionale arrays

Sune Sloth Simonsen

Dirk Hasselbalch



Målet ...

Eksempel:

```
int main() {  
    Array<int,3> array3D(4,3,5);  
    int x;  
    array3D[2][2][1] = 42;  
    x = array3D[2][2][1];  
    Array<int,2> array2D(array[2]);  
    array2D = array3D[3];  
    // ...  
}
```



Overblik

```
template<typename T, unsigned int N>  
class Array {  
    ...  
    template <typename U>  
    Array(Array<U,N>& array);  
    ...  
};
```

```
template<typename T>  
class Array {  
    ...  
    template <typename U>  
    Array(Array<U,1>& array);  
    ...  
};
```



Generel template Constructor

```
template<typename T,unsigned int N>  
m_array::Array<T,N>::Array(unsigned int length, ...) {  
  
    unsigned int lengths[N];  
    lengths[0] = length;  
  
    va_list paramlist;  
    va_start(paramlist, length);  
    for (unsigned int i=1; i<N; i++) {  
        lengths[i] = va_arg(paramlist, unsigned int);  
    }  
    va_end(paramlist);  
  
    Initialize(lengths);  
}
```



Generel template Initialiseringsfunktion

```
template<typename T,unsigned int N>  
void m_array::Array<T,N>::Initialize(unsigned int* lengths) {  
  
    length = *lengths;  
    data = new Array<T,N-1>[length];  
    ++lengths;  
  
    for (int i=0; i<length; i++) {  
        try {  
            data[i].Initialize(lengths);  
        }  
        catch(const std::bad_alloc& ex) {  
            delete[] data;  
            throw ex;  
        }  
    }  
}
```



Partielt specialiseret template Constructor og initialiseringsfunktion

```
template<typename T>  
m_array::Array<T,1>::Array(unsigned int length) {  
    Initialize(&length);  
}
```

```
template<typename T>  
void m_array::Array<T,1>::Initialize(unsigned int* lengths) {  
    length = *lengths;  
    data = new T[length];  
    for (int i=0; i<length; ++i)  
        data[i] = T();  
}
```



Copy constructor

```
template<typename T,unsigned int N>
template <typename U>
m_array::Array<T,N>::Array(Array<U,N>& array) {
    Copy(array);
}
```

```
template<typename T>
template <typename U>
m_array::Array<T,1>::Array(Array<U,1>& array) {
    Copy(array);
}
```



Generel template Kopieringsfunktion

```
template<typename T,unsigned int N>
template<typename U>
void m_array::Array<T,N>::Copy(Array<U,N>& from) {
    length = from.getLength();
    data = new Array<T,N-1>[length];
    for(int i=0; i < length; ++i) {
        try {
            data[i].Copy(from[i]);
        }
        catch(std::bad_alloc& ex) {
            delete[] data;
            throw ex;
        }
    }
}
```



Partielt specialiseret template Kopieringsfunktion

```
template<typename T>
template<typename U>
void m_array::Array<T,1>::Copy(Array<U,1>& from) {
    length = from.getLength();
    data = new T[length];
    for(int i=0; i < length; ++i) {
        data[i] = from[i];
    }
}
```



Tildelingsoperator

```
template<typename T,unsigned int N>
m_array::Array<T,N>&
m_array::Array<T,N>::operator=(Array<T,N> const& that)
throw(std::out_of_range) {

    if(this == &that)
        return *this;

    if(length != that.length)
        throw std::out_of_range("Index out of range!");

    for (int i=0; i<length; ++i)
        data[i] = that.data[i];

    return *this;
}
```



Subscript-operator

```
Array<T,N-1>& operator[](unsigned int i) {  
    if(i < 0 || length <= i)  
        throw std::out_of_range(  
            "Index out of range in subscript operator");  
    return data[i];  
}
```

```
T& operator[](unsigned int i) {  
    if(i < 0 || length <= i)  
        throw std::out_of_range(  
            "Index out of range in subscript operator");  
    return data[i];  
}
```



Subscript-operator

```
Array<T,N-1> const& operator[](unsigned int i) const {  
    if(i < 0 || length <= i)  
        throw std::out_of_range(  
            "Index out of range in subscript operator");  
    return data[i];  
}
```

```
T const& operator[](unsigned int i) const {  
    if(i < 0 || length <= i)  
        throw std::out_of_range(  
            "Index out of range in subscript operator");  
    return data[i];  
}
```