Dag 4

Pedher Johansson (pedher@cs.umu.se)
Thomas Johansson (thomasj@cs.umu.se)
Mattias Åsander (mattiasa@cs.umu.se)

Dag 4

- Introduktion till mallar (templates)
- Introduktion till arrayer och listor i standardbiblioteket

Skydd av data

- En styrka med objekt är att vi kan skydda och kontrollera data
- Utmärkt för att kapsla in data, oavsett struktur
  - garantera konsistens och validitet

Arrayer som exempel

- Arrayer har flera inbyggda problem

Krävs pekare och new för att dynamiskt bestämma storlek (krångligt och riskerar pekarfel och minnesläckor)

Måste bestämma storlek vid initiering

Ingen kontroll av index

Kan ej garantera initiala värden

Vet ej storlek (allokerad eller antal inlagda element)

```
int v[5];
cout << v[0] << endl;
cout << v[5] << endl;
for (int i = 0; i < 5; i++)
    cout << v[i] << endl;
```


Array som eget objekt

Exempel på funktionalitet

- Ange initial storlek i konstruktorn
- objektet garanterar startvärde
- objektet håller reda på dynamisk allokering och avallokering
- objektet håller reda på storleken
- objektet kan ev. garantera index

Array som eget objekt

```
class MyArray
{
private:
    int* array;
    int length;
public:
    MyArray (int size);
    ~MyArray ();
    int size();
    void set(int index, int value);
    int get(int index);
}
```

```
MyArray v(5);
cout << v.get(0) << endl;
cout << v.get(5) << endl;
for (int i = 0; i < v.size(); i++)
    cout << v.get(i) << endl;
```

Generisk datatyper

- Funktionella objekt, men vi kan inte ange datatyp → problem!!!
  - Måste skapa en klass per datatyp?
- I C++ finns ett sätt att skapa generiska datatyper där datatyp anges när vi definierar variabel och allokerar objekt.

Mallar

Går bara in på hur de används

```
template <class T>
class MyArray
{
private:
    T* array;
    int length;
public:
    MyArray (int size);
    ~MyArray ();
    int size();
    void set(int index, T value);
    T get(int index);
}
```

```
MyArray<int> v(5);
cout << v.get(0) << endl;
cout << v.get(5) << endl;;
for (int i = 0; i < v.size(); i++)
    cout << v.get(i) << endl;
```

- Kompilatorn skapar en instans per datatyp som är använd med en mall

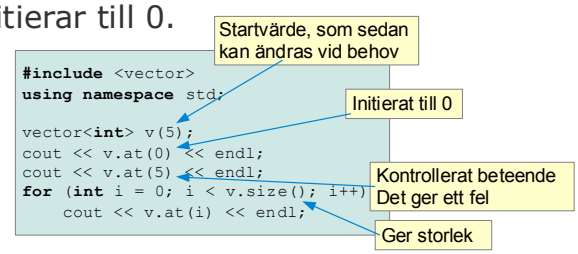
Standardbiblioteket

- I standardbiblioteket finns förutom iostream (utskrifter), även flera datatyper färdiga som mallar
  - `array` (från C++ 11), `vector`, `list`, etc.

```
#include <array>
#include <vector>
#include <list>
using namespace std;
```


Exemplet <vector>

- <vector> till skillnad från <array> kan ändra storlek dynamiskt, samt initierar till 0.



Sekvenser

- <array> array av fix storlek

```
array<int, 3> arr();
```

  - [] eller at() kan användas, den senare kollar om indexet är giltigt

```
cout << arr[0];
cout << arr.at(0);
```

- <list> dubbellänkad lista
  - kan växa och krympa efter behov
  - kan infoga element mitt i listan
  - kan ej komma åt enskilda element
- <vector> array, vissa metoder som i en lista
  - kräver ibland kopiering

```
vector<double> v(5);
v.insert(0, 10, 2);
```


Vanliga metoder

- <array>
  - fill, size, at eller []
- <vector>
  - size, at eller [], insert, erase, clear, push_back, pop_back
- <list>
  - size, front, back, insert, erase, clear, push_front, push_back, pop_front, pop_back

Iteratorer

- Itertorer är små hjälpobjekt vid iterering av listor och arrayer

```
list<int> mylist;

for (int i=1; i<=10; i++)
    mylist.push_back(i);

cout << "mylist contains:";
for (list<int>::iterator it=mylist.begin(); it != mylist.end(); ++it)
    cout << ' ' << *it;
cout << '\n';
```


Varför (standard)bibliotek?

- Enhetligt och välkänt
- Vältestat och optimerat
- Behöver inte återuppfinna hjulet
- Finns inte på alla system

Förenklad for-loop

- Finns i C++ 11
- Fungerar med sekvenser i standardbiblioteket

```
vector<double> values;

// Listar samtliga element i values
for(double e: values)
   cout << e << endl;
```