C++ 1 Les 2

Includes

Forward Declaration

Cyclic Dependencies

Leerdoelen

Aan het eind van deze workshop kun je:

- uitleggen wat er problematisch is aan het meerdere keer includen van dezelfde .h file en waarom dat toch werkt in de praktijk
- een forward declaration gebruiken
- omgaan met cyclic dependencies

Header Files

Declaraties komen in de *.h files.

```
class A
{
public:
    A();
    ~A();
    void Fun();

private:
    int* i;
};
```

Constructor en Destructor

Source Files

Definition (Implementatie) komt in de *.cpp files.

```
A::A()
{
    i = new int;
}

A::~A()
{
    delete i;
}

void A::Fun()
{
    *i += 2;
}
```

Includes

- Heb je code nodig in een andere file, geef je dat aan met:

```
#include "file.h"
```

- Dan wordt de betreffende file ingevoegd vóór de rest van de code.

```
//afgeleideklasse.cpp

#include "basisklasse.h"

Afgeleideklasse::afgeleideklasse()
{
};
```

Wordt hier eerst ingevoegd, voor de rest van de code. (omdat hij daar nodig)

`#include ""` of `<>`

- Wanneer je `#include <>` gebruikt dan zoekt de compiler in het system path.
- Wanneer je `#include ""` gebruikt dan zoekt de compiler eerst in het local directory, dan in het system path.

Dus:

- Gebruik `<>` voor includes van system headers
- Gebruik `""` voor je eigen header files

Include Meerdere Malen

- Als je dezelfde include file op meerdere plaatsen zou includen, dan zou hij meerdere malen worden gecompileerd.
- Dit is ongewenst, want dan ziet de compiler twee keer dezelfde declaraties, en dat mag niet.

MyClass.cpp:
```
#include <string>
#include “yourclass.h”
#include “myclass.h”
```

YourClass.h:
```
#include <string>
```

Oplossing (1)

- Gebruik `#pragma once` aan het begin van de header file. (vaak automatisch als je de programmeer omgeving de klasse laat genereren).
- Werkt alleen als de compiler het ondersteunt.

```
//x.h

#pragma once

class X { };
```

Oplossing (2)

- Een `#ifndef` met een unieke identifier aan het begin van de header, zodat wat er na de `#ifndef` komt wordt overgeslagen als de file al is gecompileerd.
- Werkt altijd.

```
//x.h

#ifndef __X_H_INCLUDED__   //unieke identifier, standaard “uit”
#define __X_H_INCLUDED__   //wordt hier “aan” gezet

class X { };

#endif
```

Forward Declaration (1)

- Een afgeleide klasse include altijd zijn basis klasse.
- Als de basis klasse pointers of referenties naar de afgeleide klasse heeft dan is 't mooier als deze forward declared worden.

```
//basis.h

class ding;

Class basis {
public:
    basis();
    ~basis();
private:
    ding* a;
};
```

Forward Declaration

Forward Declaration (2)

- Alles wat een naam heeft kun je forward declaren. Dus:
  - Klassen
  - Functies
  - Enums
  - Etc.

Cyclic Dependencies

```
//a.h
#include “b.h”
class A {
      B* b;
}
```

```
//b.h
#include “a.h”
class B {
      A* a;
}
```

- A heeft een pointer naar een B object en B heeft een pointer naar een A object.
- Dit moet mogelijk zijn.
- Maar wanneer ik dit compileer gaat het fout!

Voorkomen Cyclic Dependencies

```
//a.h
class B;
class A {
      B* b;
}
```

```
//b.h
Class A;
class B {
      A* a;
}
```

- Forward Declaration
- Ik beloof dat class A/class B bestaat
- Compileren gaat nu goed, zolang je je belofte maar waar maakt.

Foute Cyclic Dependency

```
//a.h
class B;
class A {
      B b;
}
```

Geen pointer meer

```
//b.h
Class A;
class B {
      A a;
}
```

- Compileren gaat hier fout want: een A heeft een B heeft een A heeft een B heeft een A…. ….. *ad infinitum*
- Bij pointers bestaat dat probleem niet want: een A heeft geen B, maar wijst naar een B die buiten zijn eigen geheugen ligt (en vice versa voor B)
- Wanneer je dit aantreft heb je waarschijnlijk ergens een ontwerp fout gemaakt. Bekijk nog eens goed wat je wilt doen en pas het ontwerp aan.