Headerfiler

- Vi har hittills skrivit alla våra funktioner i en enda källkodsfil (och sett till att deklarera funktionerna i rätt ordning!)
- För större program måste funktionerna delas upp på flera filer
- Hur kan vi tillåta en funktion i en fil att anropa en funktion i en annan fil?
- Hur kan vi använda samma **`struct`** i två olika källkodsfiler?
- Lösning: Skriv egna *headerfiler*

Headerfiler

- Vi har redan inkluderat *redan existerande* headerfiler för att använda funktioner från kodbibliotek (**printf**, **scanf**, **pow**...)
- Skapa en *egen* headerfil om du vill
  - deklarera funktioner
  - deklarera datatyper
  - definiera globala variabler och konstanter

  som ska användas i flera källkodsfiler
- För att inkludera en egen headerfil används syntaxen **`#include "myownheaderfile.h"`**

Funktionsdeklarationer

- Består av den första raden i funktionen, följt av ett semikolon
- Löfte till kompilatorn: “Jag lovar att en implementation som motsvarar denna funktionsdeklaration existerar när programmets olika delar skall länkas ihop”
- Om löftet inte uppfylls (funktionsimplementationen saknas helt eller avviker från det utlovade) => *länkningsfel*

```
/tmp/cc0FZaOG.o: In function `main':test.c:
(.text+0x12):
undefined reference to `getMeaningOfLife'
collect2: ld returned 1 exit status
```

=> Vid användning av yttre bibliotek behövs både själva biblioteket (implementationen) **och** en headerfil med deklarationer

Funktionsdeklarationer

```
// calc.h

double calcArea(double radius);
double calcCirc(double radius);
```

```
// calc.c

#include <math.h> // To use M_PI

double calcArea(double radius) {
    return M_PI*radius*radius;
}

double calcCirc(double radius) {
    return 2*M_PI*radius;
}
```

```
// main.c

#include <stdio.h>
#include "calc.h" // to use calcArea

int main(void) {
    printf("Radius %f gives area %f",
                        2.5,
                        calcArea(2.5));
    return 0;
}
```

För att kompilera:
`gcc main.c calc.c`

Implicita funktionsdeklarationer

- Vad händer om **`#include "calc.h"`** utelämnas från main.c?

  => *implicit* funktionsdeklaration av **`calcArea()`**

  - Funktionen antas ha returtypen **`int`**
  - gcc gissar sig till parametertyperna baserat på funktionsanropet

  => programmet kompileras men ger fel resultat

  => använd alltid *explicita* funktionsdeklarationer!

- Kan kompilera med flaggan **`-Wall`** för att få varningar:
  `gcc main.c calc.c -Wall`

Include guards

- Funktioner, datatyper etc. får endast definieras en gång
- Exempelscenario:
  Filen **foo.h** inkluderar filerna **bar.h** och **calc.h**
  Filen **bar.h** inkluderar också **calc.h**
  => **calc.h** inkluderas två gånger
  => alla deklarationer i **calc.h** körs två gånger
  => kompilatorfel!
- Lösning: Använd *include guards*
- Skapar en unik definition för varje fil när den inkluderas första gången
- Kontrollerar om definitionen redan är gjord innan den inkluderas på nytt

Include guards

```
// calc.h

#ifndef __CALC_H__
#define __CALC_H__
float calcArea(float radius);
float calcCirc(float radius);

typedef struct {
    float x;
    float y;
} point;

#endif
```

Om definitionen inte är gjord...

...gör definitionen,
dvs. inkludera filen...

...annars: inkludera ingenting
(ingen #else-del!)

Externa variabler

- Vi har tidigare skapat variabler som varit globalt tillgängliga i **en** fil
- Hur göra variablerna tillgängliga i flera filer?
- Samma princip som för funktioner:
- *Deklarera* den globala variabeln i en headerfil med nyckelordet **`extern`**
- *Definiera* den globala variabeln i **en** av c-filerna
- *Inkludera* headerfilen i de c-filer som använder den globala variabeln

Externa variabler

```
// main.c

#include <stdio.h>
#include "foo.h"

int main(void)
{
    printf("%d\n", global);
    global = 2;
    foo();
    printf("%d\n", global);
    return 0;
}
```

```
// foo.h

// declarations
extern int global;
void foo(void);
```

```
// foo.c

#include <stdio.h>

// definitions
int global = 1;

void foo(void)
{
    printf("%d\n", global);
    global = 3;
    printf("%d\n", global);
}
```