

F03A - Organisation av C-kod

5DV149 Datastrukturer och algoritmer

Niclas Börlin
niclas.borlin@cs.umu.se

2024-03-20 Ons

Funktionsdeklarationer och -definitioner (1)

- ▶ En funktions-**deklaration** berättar för kompilator att en funktion **existerar**.
- ▶ Den berättar vad funktionen har för:
 1. Namn.
 2. Typ på returvärde.
 3. Parametrar: antal, ordning, typ (namnet är ointressant).
- ▶ Exempel:

```
list_pos list_next(const list *l, const list_pos p);
```

- ▶ list_next är en funktion som tar två parametrar:
 - ▶ den första parametern är av typen **const list ***.
 - ▶ den andra parametern är av typen **const list_pos**.
- ▶ list_next returnerar ett värde av typen list_pos.
- ▶ Notera det **avslutande semikolonet!**

Funktionsdeklarationer och -definitioner (2)

- ▶ En funktions-**definition** berättar vad funktionen **gör**.
- ▶ Den innehåller funktionens kropp

```
code/list.c
112 /**
113  * list_next() - Return the next position in a list.
114  * @l: List to inspect.
115  * @p: Any valid position except the last in the list.
116  *
117  * Returns: The position in the list after the given position.
118  *         NOTE: The return value is undefined for the last position.
119  */
120 list_pos list_next(const list *l, const list_pos p)
121 {
122     return p->next;
123 }
```

Funktionsdeklarationer och -definitioner (3)

- ▶ Funktionsdeklarationer får upprepas.
 - ▶ Måste komma före användande av (anrop till) funktionen.
- ▶ Funktionsdefinitionen får bara förekomma **en gång**.

Organisation av C-kod

- ▶ För projekt som implementeras i språket C är det vanligt att definitioner av funktioner som hör ihop samlas i en kodfil, t.ex. `list.c`.
 - ▶ Det kallas ibland för en **modul**.
 - ▶ I vårt fall motsvarar modulen **datatypen Lista**.
- ▶ Dessutom skapas en s.k. **header-fil** (`.h`-fil) där varje **publik** funktion **deklarerar**, t.ex. `list.h`.
- ▶ Header-filerna kan sägas specificera modulens (datatypens) **gränssnitt**.

Header-filer som dokumentation

- ▶ Det är vanligt att header-filer innehåller viktiga **implementationsdetaljer**.
- ▶ Så också på denna kurs:
 - ▶ Delar av specifikationen till respektive laboration kommer att återfinnas i respektive header-fil.
- ▶ Om ni ska lära er använda ny, okänd kod — kolla i första hand den officiella dokumentationen, i andra hand header-filerna (motsv.)!

Mapstruktur

- ▶ Det är vanligt att strukturera koden i mappar på följande sätt:
 - ▶ `include/` innehåller `.h`-filerna.
 - ▶ `lib/` innehåller biblioteksfiler.
 - ▶ `bin/` kompillerade, exekverbara filer hamnar här.
 - ▶ `src/` toppkatalog för källkoden
 - ▶ `list/` kod för list-modulen
 - ▶ `dlist/` kod för dlist-modulen
 - ▶ `stack/` kod för stack-modulen
 - ▶ ...
- ▶ För små projekt kan all kod samlas i en katalog `src/`.

Mapstruktur för kodbasen

```
datastructures/  
├── include/  
├── lib/  
└── src/  
    ├── array_1d/  
    ├── array_2d/  
    ├── dlist/  
    ├── list/  
    ├── queue/  
    ├── stack/  
    ├── table/  
    ├── int_array_1d/  
    ├── int_list/  
    ├── int_list_array/  
    └── int_stack/
```

```
datastructures/  
├── include/  
│   └── list.h  
├── ...  
└── src/  
    └── list/  
        ├── list.c  
        ├── list_mwe1.c  
        ├── list_mwe2.c  
        ├── list_test1.c  
        └── list_test2.c
```

Header-filer (.h-filer) (1)

- ▶ Välskrivna headerfiler innehåller förutom deklarationerna av funktioner också en **hjälp**text som förklarar det användaren behöver veta för att kunna använda funktionen på rätt sätt.

```
code/list.h
75 /**
76  * list_next() - Return the next position in a list.
77  * @l: List to inspect.
78  * @p: Any valid position except the last in the list.
79  *
80  * Returns: The position in the list after the given position.
81  * NOTE: The return value is undefined for the last position.
82  */
```

- ▶ Dessutom innehåller .h-filerna vanligen definitioner av publika **typer** och **konstanter** som hör ihop med modulen.

```
code/list.h
32 // List type.
33 typedef struct list list;
34
35 // List position type.
36 typedef struct cell *list_pos;
```

Header-filer (.h-filer) (2)

- ▶ För att undvika att samma definitioner inkluderas flera gånger innehåller .h-filer vanligen **#ifndef**-direktiv (*if not defined*):

```
code/list.h
1  #ifndef __LIST_H
2  #define __LIST_H
```

```
code/list.h
177 #endif
```

Kod som använder modulen (1)

- ▶ Källkodsfiler som ska **använda** modulens funktioner/konstanter/typer använder kompilatordirektivet

```
#include <list.h>
```

eller

```
#include "list.h"
```

till att inkludera innehållet i headerfilen vid kompileringen.

- ▶ Ett include-direktiv
 - ▶ på formen **#include <list.h>** söker efter .h-filen på standardställen.
 - ▶ på formen **#include "list.h"** söker dessutom i aktuell katalog.
- ▶ Vanligen använder man den senare versionen bara på kod man skrivit själv och som ligger i samma katalog som C-filen.

Kod som använder modulen (2)

- ▶ Källkodsfilerna kan vara med ett **huvudprogram** med en **main()**-funktion eller en **annan modul** som använder Lista.
- ▶ Efter include-direktivet kommer alla funktionsdeklarationer, definierade konstanter och typer som finns i header-filen bli tillgängliga.

Kod som använder modulen (3)

- ▶ När kompilatorn hittar **anrop** till funktionerna som deklarerats i `list.h` så kan den **kontrollera** att parametertyper, ordning och antal samt returtyp är korrekt.

```
list *l = list_empty(NULL);
```

- ▶ Kompilatorn kontrollerar ungefär detta:
 1. `list_empty()` tar en parameter av typen `Z`.
 2. `NULL` går att konvertera till typen `Z`.
 3. `list_empty()` returnerar ett värde av typen `list *`.
 4. `list *` går att konvertera till den typ som variabeln `l` har.
- ▶ Dessutom genererar kompilatorn **anropskod** till funktionen, ungefär
 1. Konvertera `NULL` till typen `Z` och lägg på stacken.
 2. Anropa funktionen `list_empty()`.
 3. Konvertera returvärdet till `list *` och stoppa i variabeln `l`.

Kod som använder modulen (4)

- ▶ Glömmer man bort att inkludera header-filen brukar man få fel av typen **unknown type name** eller varningar av typen **implicit declaration of function**:

```
list_mwe1.c: In function 'main':
list_mwe1.c:26:2: error: unknown type name 'list'
 26 | list *l = list_empty(NULL);
    |      ^~~~
list_mwe1.c:26:12: warning: implicit declaration of function 'list_empty' [-Wimplicit-function-declaration]
 26 | list *l = list_empty(NULL);
    |             ^~~~~~
```

- ▶ Kom ihåg att kontrollera det **första** felmeddelandet/varningen!

Koden som definierar funktionerna i modulen

- ▶ Källkodsfilen (här: `list.c`) som innehåller **definitionerna** till funktionerna i modulen ska också innehålla

```
#include <list.h>
```

- ▶ Det är inte ett krav från kompilatorn men riskerar att skapa svårupptäckta **buggar** om det utelämnas.
- ▶ I ett senare skede av kompileringen så "länkas" den kompilerade koden av funktionen `in` (den exakta adressen i minnet på funktionen läggs till).
- ▶ Länkaren byter ut anropskoden från
 2. Anropa funktionen `list_empty()`.till
 2. Anropa funktionen på adress `0x4800ef34` (eller annan adress).

Kompilering av flera filer

- ▶ Vid kompilering av flera filer så måste **alla** C-filer som ska kompileras anges.

```
gcc file1.c file2.c ...
```

- ▶ Header-filer ska inte inkluderas i listan.
 - ▶ I stället ska sökvägen till mapparna som innehåller include-filerna anges med flaggan `-I`
- ▶ Om vi till exempel vill kompilera exempelfilen `list_mwe1.c` så måste vi lägga till `list.c`

```
cd ../src/list
gcc -I ../../include/ -o list_mwe1 list_mwe1.c list.c
```

- ▶ Glömmer vi att lägga till `list.c` så kommer länkaren att ge oss felmeddelanden av typen **undefined reference**:

```
/usr/bin/ld: /tmp/ccKrG8Zg.o: in function `main':
list_mwe1.c:(.text+0x84): undefined reference to `list_empty'
/usr/bin/ld: list_mwe1.c:(.text+0xac): undefined reference to `list_first'
/usr/bin/ld: list_mwe1.c:(.text+0xc2): undefined reference to `list_insert'
```

Biblioteksfiler (1)

- ▶ Om vi använder moduler som använder andra moduler så blir det snabbt jobbigt att hålla reda på vilka källkodsfiler vi behöver.
- ▶ Då finns det sätt att använda **biblioteksfiler** som innehåller halvkompilerade versioner av källkodsfilerna (s.k. objektfiler).
- ▶ Dessa biblioteksfiler lagras vanligen i en mapp `lib/`

```
datastructures/  
├── include/  
├── lib/  
│   └── libdoa.a  
└── src/  
    └── list/  
        └── list_mwe1.c
```

Biblioteksfiler (2)

- ▶ Vid kompileringen används då flaggan `-L` för att tala om sökvägen till biblioteksmappen och `-l` för att ange vilka bibliotek man vill ha.

```
datastructures/  
├── include/  
├── lib/  
│   └── libdoa.a  
└── src/  
    └── list/  
        └── list_mwe1.c
```

```
cd ../src/list  
gcc -I ../../include/ -o list_mwe1 -L ../../lib list_mwe1.c -ldoa
```