F03A - Organisation av C-kod 5DV149 Datastrukturer och algoritmer Programmeringsbok i C

Niclas Börlin niclas.borlin@cs.umu.se

2024-01-18 Tor

Funktionsdeklarationer och -definitioner (1)

- ► En funktions-deklaration berättar för kompilator att en funktion finns.
- Den berättar vad funktionen har för:
 - 1. Namn.
 - 2. Typ på returvärde.
 - 3. Parametrar (antal, ordning och typ men behöver ej namn).
- **Exempel**:

```
list_pos list_next(const list *1, 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 också funktionens kropp

```
code/list.c.
        /**
112
         * list next() - Return the next position in a list.
113
114
         * @l: List to inspect.
115
         * Op: 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
         */
        list_pos list_next(const list *1, const list_pos p)
120
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.

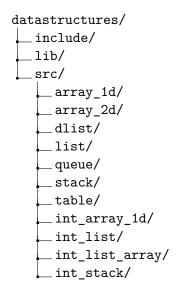
Organisation av C-kod

- 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 f\u00f6r en modul.
 - ► I vårt fall motsvarar modulen datatypen Lista.
- Dessutom skapas en s.k. header-fil (.h-fil) där varje publik funktion deklareras, t.ex. list.h.
- Header-filerna kan sägas specificera modulens (datatypens) gränssnitt.

Mappstruktur

- Det är vanligt att strukturera koden i mappar på följande sätt:
 - ▶ include/ innehåller .h-filerna.
 - ► lib/ innehåller biblioteksfiler.
 - bin/ kompilerade filer läggs hit.
 - 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/.

Mappstruktur för kodbasen



```
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 hjälptext 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
        * Op: 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

// List type.
typedef struct list list;

// List position type.
typedef struct cell *list_pos;
```

32

33

34 35

36

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

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

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 tist.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 *1 = list_empty(NULL);
```

- Kompilatorn kontrollerar ungefär detta:
 - list_empty() tar en parameter av typen Z.
 - 2. NULL går att konvertera till typen Z.
 - 3. list_empty() returnerar ett v\u00e4rde av typen list *.
 - 4. list * går att konvertera till den typ som variabeln 1 har.
- Dessutom genererar kompilatorn anropskod till funktionen, ungefär
 - 1. Konvertera NULL till typen Z och lägg på stacken.
 - Anropa funktionen list_empty().
 - 3. Konvertera returvärdet till list * och stoppa i variabeln 1.

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:

► 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
 - Anropa funktionen list_empty().

till

2. Anropa funktionen på adress 0x4800ef34 (typ).

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

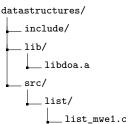
```
cd .../src/list
gcc -I ../../include/ -o list_mwe1 list_mwe1.c list.c
```

► 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/



Biblioteksfiler (2)

 Vid kompileringen används då flaggan –L för att tala om sökvägen till
 biblioteksmappen och –1 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
```