Screencast

Funktioner

Nyttan med funktioner[1]

- Delar upp stora uppgifter i mindre
- Förenklar underhåll och lokala förbättringar
- Förenklar (möjliggör) återanvändning av kod
- Ökar abstraktionsnivån genom att gömma detaljer bakom ett beskrivande namn
- Ett C-program består i regel av många funktioner

[1] Ett funktionsanrop fungerar litet som en fotnot.

– Syntaxen för en funktion i C är denna:

```
1   ReturTyp namn(ParameterTyp1 pnamn1 ... ParameterTypN pnamnN) {
2     Funktionskropp
3   }
```

Där:

`ReturTyp` avser typen på det värde som funktionen returnerar, t.ex. **`int`**

`ParameterTyp1 pnamn1` avser typ och namn på en parameter, t.ex. **`char minit`**, och

Funktionskropp är C-uttryck separerade med **`;`**

Man kan ersätta både returtypen och parametrarna med **`void`** som betyder ”ingenting” alltså avsaknad av returvärde respektive parametrar.

– Exempel på en funktion som tar två heltal och returnerar det största:

```
1   short max(short a, short b) {
2     int result = a;
3     if (b > a) { result = b; }
4     return result;
5   }
```

– En funktions parametertyper och returtyp kallas för dess *signatur*. Signaturen för `max` är **`short`** $\times$ **`short`** $\rightarrow$ **`short`**, dvs. den tar emot två **`short`**:ar och returnerar en

– Vi kan skriva om föregående funktion på följande (mer läsbara) sätt:

```
1     short max(short a, short b) {
2       if (a > b) {
3         return a;
4       } else {
5         return b;
6       }
7     }
```

– . . . eller. . .

```
1     short max(short a, short b) {
2       if (a > b) {
3         return a;
4       }
5       return b;
6     }
```

Dessa är alla ekvivalenta. Den sistnämna ger prov på hur en funktions exekvering avbryts av **return**. Om villkoret på rad 2 är sant utförs rad 3 och sedan ”hoppar vi ur” funktionen och utför alltså inte rad 4 eller 5.

– Nedanstående kod visar funktionerna `min` och `max` där den första (för att illustrera en poäng, gör inte om detta) är implementerad i termer av den andra:

```
1   short min(short a, short b) {
2     return max(a,b) == a ? b : a; // notera anropet till en funktion!
3   }
4
5   short max(short a, short b) {
6     if (a > b) {
7       return a;
8     } else {
9       return b;
10    }
11  }
```

– Om vi kompilerar denna kod (i föreliggande fall med gcc på OS X 10.8) får vi ett fel:

```
foo> gcc test.c
test.c:7: error: conflicting types for 'max'
test.c:4: error: previous implicit declaration of 'max' was here
```

Varför får vi detta fel? (Pröva också med `gcc -std=c99 test.c`. Skillnad?)

– Felmeddelandet är tyvärr litet missvisade och beror på två saker. Vi tar dem en i taget:

1. Felet på föregående sida beror på att C vid anropet till `max` på rad 2 *ännu inte hade sett någon definition av funktionen*. Denna definition kom först på rad 5!

   Eftersom C-kompilatorn inte har sett rad 5 ännu vet den inte något om `max`:s signatur. Den kommer av pragmatiska och förhistoriska skäl att *gissa* att `max` har signaturen **`int`** $\times$ **`int`** $\rightarrow$ **`int`**.

   På så vis blir funktionen `max` *implicit* deklarerad.[2]

2. När C-kompilatorn når rad 5 stöter den på den *explicita* deklarationen av `max` och ser att dess signatur faktiskt är **`short`** $\times$ **`short`** $\rightarrow$ **`short`**, vilket strider mot det tidigare antagandet.

   Nu generas felet.

– Ett dåligt sätt att komma runt felet är att byta *deklarationsordning* på funktionerna.

  Detta fungerar dock enbart om det inte finns någon ömsesidig rekursion, och kan försämra läsbarheten om semantisk gruppering bryts upp för att tillfredsställa kompilatorn.

– Ett bättre sätt är att använda en s.k. *funktionsprototyp*.

[2] I vissa C-standarder (t.ex. C 99) bör detta ge en varning även om man inte har `-Wall` påslaget.

– En funktionsprototyp i C är en funktionsdefinition minus funktionskroppen, t.ex.:

```
short max(short a, short b);
```

– En funktionsprototyp registrerar en funktions signatur hos kompilatorn så att beroenden av det tidigare slaget kan lösas ut.

– Koden skrivs alltså med fördel:

```
1   short min(short a, short b);
2   short max(short a, short b);
3
4   short min(short a, short b) {
5     return max(a,b) == a ? b : a;
6   }
7
8   short max(short a, short b) {
9     if (a > b) {
10      return a;
11    } else {
12      return b;
13    }
14  }
```

- Funktionsprototyperna är ett sätt att deklarera ett programs design
- Funktionsprototyper deklareras i regel i en s.k. header-fil (täcks av annan screencast)
- Om man ändrar en funktions signatur måste man uppdatera på alla (båda) ställen där signaturen anges
- Ta för vana att alltid deklarera funktionsprototyper för varje funktion