Funktioner

- Vad är det och hur definierar vi en
- Top-down-programmering
- lokala – globala variabler
- void och flera inparametrar

OBS! Till nästa gång läs igenom problemlösning 1 och skissa på lösningar!

Problemlösning

När man skall lösa ett komplicerat problem gör man det lättast genom att dela upp problemet i mindre delar. Det gäller för programmering, matematik och faktiskt all problemlösning.

I programmering använder vi för detta **funktioner**.

Funktion

- En funktion är en liten programsnutt som utför en speciell uppgift.
- Ofta kan uppgiften bestå i att den får något värde och spottar ut ett annat värde. I sådana fall påminner den mycket om en matematisk funktion:

  3 → (dubblar talet) → 6

Exempel

En funktion räknar ut arean av en cirkel

- Den behöver veta radien. Radien kallas för en in-parameter, input, eller bara parameter.
- Med hjälp av radien räknar den ut arean. Arean kallas för returvärde eller output.

2.00 → (arean beräknas) → 12.57

Funktionsdefinition -exempel

```
float cirkelarea(float radie)
{
    float area;
    area = radie*radie*M_PI;
    return area;
}
```

…med kod

```
#include <stdio.h>
#include <math.h>

float cirkelarea(float radie){
      float area;
      area = radie*radie*M_PI;
      return area;
}

int main(void)
{
   printf("En cirkel med radien 2 har arean: %.2f\n",cirkelarea(2));
   float r=4;
   printf("En cirkel med radien %.2f har arean: %.2f",r,cirkelarea(r));
}
```

Resultat:
En cirkel med radien 2 har arean: 12.57
En cirkel med radien 4.00 har arean: 50.27

main- är en funktion, programmets huvudfunktion

funktionsdefinitionen sist

```
#include <stdio.h>
#include <math.h>

float cirkelarea(float radie);

int main(void)
{
   printf("En cirkel med radien 2 har arean: %.2f\n",cirkelarea(2));
   float r=4;
   printf("En cirkel med radien %.2f har arean: %.2f",r,cirkelarea(r));
}

float cirkelarea(float radie){
      float area;
      area = radie*radie*M_PI;
      return area;
}
```

Funktionsdeklaration!

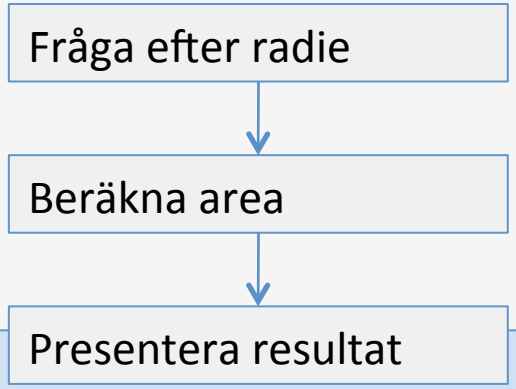
Varför funktioner?

- Enklare att dela upp problem i delar
- Tydligare kod som är lättare att läsa och lättare att felsöka
- Vi kan använda samma kod flera gånger
- Vi kan använda samma kod i olika program (bl.a. via bibliotek)
- Vi kan använda kod utan att förstå hur den gör. Vi behöver bara förstå hur man använder metoden, dvs vad stoppar vi in, vad får vi ut.

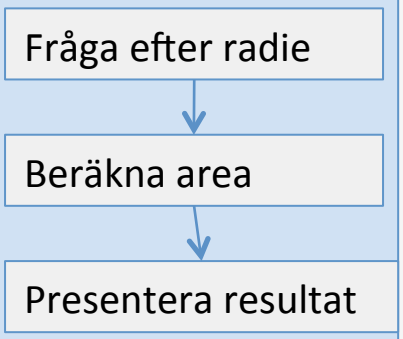
Exempel:

Beräkningarna görs i en funktion och hur man gör behöver vi inte fundera på när vi löser det övergripande problemet:

Top-down-programmering



Top-down




```
#include <stdio.h>
#include <math.h>

float cirkelarea(float radie);

int main(void)
{
   float area,radie;

   printf("Ange radien:");
   scanf("%f",&radie);

   area=cirkelarea(radie);

   printf("En cirkel med radien %.3f har arean: %.3f",radie,area);
}

float cirkelarea(float radie){
        float area;
        area = radie*radie*M_PI;
        return area;
}
```

Detaljerna kan man vänta med. Framförallt kan man tänka på funktionen som en svart låda när man skriver huvudprogrammet.

Med loop

```
#include <stdio.h>
#include <math.h>

float cirkelarea(float radie);

int main(void)
{
   float area,radie;
   int fortsatt = 1;
   while(fortsatt==1)
   {
      printf("Ange radien:");
      scanf("%f",&radie);

      area=cirkelarea(radie);

      printf("En cirkel med radien %.3f har arean: %.3f\n",radie,area);
      printf("Vill du fortsatta nej-0, ja-1?");
      scanf("%d",&fortsatt);
   }
}
```

Variabler

```
#include <stdio.h>

int dubbla(int tal)
{
   tal=tal*2;
   return tal;
}

int main(void)
{
      int i=1;
      int j=dubbla(i);
      printf("i: %d, j %d",i,j);

}
```

Resultat: 1,2

Alla variabler som vi definierar i en funktion finns bara inuti funktionen vi säger att de är **lokala variabler**.

De in-parametrar vi skickar in i en metod kopieras till en ny minnesplats så att parametervariabeln är en lokal variabel. **Ändrar vi värdet på en inparameter ändras inte värdet i main!**

Globala variabler deklareras utanför funktionerna och de finns då i alla funktioner nedan deklarationen.

Om en ny variabel med samma namn deklareras i en funktion *skuggas* den globala variabeln.

Undvik globala variabler utom i undantagsfall. De förstör funktionen som svart låda.

```
#include <stdio.h>
int a=1;//global!
void funk2(void){
   printf("%d,",a);
}
void funk1(void){
   int a=4;
   printf("%d,",a);
}
int main(void){
   int a=2;
   printf("%d,",a);
   {
      int a=3;
      printf("%d,",a);
   }
   funk1();
   funk2();
}
```

Resultat: 2,3,4,1,

Varianter

- En funktion behöver inte ha någon returtyp
- En funktion behöver inte ha inparametrar

```
void stars()
{
        printf("***************\n");
}
```

- En funktion kan ha flera inparametrar

```
int summa(int a, int b, int c)
{
        return a+b+c;
}
```

Exempel top-down-programmering

Vi vill skriva ett program som låter användaren få testa om ett visst tal är ett primtal.

antag att vi har en funktion som kan bestämma om ett tal är ett primtal

```
int main(void)
{
   int tal;
   printf("skriv in ett heltal:");
   scanf("%d",&tal);

   if(primtal(tal))
   {
      printf("Det ar ett primtal");
   }
   else
   {
      printf("Det ar inte ett primtal");
   }

}
```

enkelt! –tyvärr kan vi inte testkompilera ☹

```
#include <stdio.h>

int primtal(int tal)
{
   return 1;
}

int main(void)
{
   int tal;
   printf("skriv in ett heltal:");
   scanf("%d",&tal);

   if(primtal(tal))
   {
      printf("Det ar ett primtal");
   }
   else
   {
      printf("Det ar inte ett primtal");
   }

}
```

lägg till en enkel funktion som levererar ett svar om än fel så kan du testa resten av koden

då är det bara att skriva en funktion som bestämmer om ett tal är ett heltal…

```
int primtal(int tal)
{
   int i;
   for(i=2;i<=tak(tal);i++)
   {
      if(delbart(tal,i))
      {
         return 0;
      }
   }
   return 1;
}
```

och så funktioner som räknar ut högsta talet man behöver testa och som tar reda på om ett tal är delbart i ett annat

```
//beräknar högsta tal som man behöver testa med för
//att bestämma om ett tal är ett primtal
int tak(int tal)
{
   return tal-1;
}

int delbart(int taljare, int namnare)
{
   return !(taljare%namnare);
}
```

Nu måste man först testa att dessa fungerar som tänkt innan man testar hela programmet. För detta får man skriva testkod. Sedan testar man sig baklänges upp tills man testat hela programmet.

```c
#include <stdio.h>

//beräknar högsta tal som man behöver testa med för
//att bestämma om ett tal är ett primtal
int tak(int tal)
{
   return tal-1;
}

int delbart(int taljare, int namnare)
{
   return !(taljare%namnare);
}

int primtal(int tal)
{
   int i;
   for(i=2;i<=tak(tal);i++)
   {
      if(delbart(tal,i))
      {
         return 0;
      }
   }
   return 1;
}

int main(void)
{
   int tal;
   printf("skriv in ett heltal:");
   scanf("%d",&tal);
   if(primtal(tal))
   {
      printf("Det ar ett primtal");
   }
   else
   {
      printf("Det ar inte ett primtal");
   }
}
```

Kan vi förbättra programmet?

lägg märke till att när vi förbättrar tak behöver vi inte ändra något annat den är som en svart låda!
(förrutom #include <math.h>)

```
int tak(int tal)
{
    return sqrt(tal);
}
```

```c
#include <stdio.h>
#include <math.h>
//beräknar högsta tal som man behöver testa med för
//att bestämma om ett tal är ett primtal
int tak(int tal)
{
   return sqrt(tal);
}

int delbart(int taljare, int namnare)
{
   return !(taljare%namnare);
}

int primtal(int tal)
{
   int i;
   for(i=2;i<=tak(tal);i++)
   {
      if(delbart(tal,i))
      {
         return 0;
      }
   }
   return 1;
}

int main(void)
{
   int tal;
   printf("skriv in ett heltal:");
   scanf("%d",&tal);
   if(primtal(tal))
   {
      printf("Det ar ett primtal");
   }
   else
   {
      printf("Det ar inte ett primtal");
   }
}
```

- En annan metodik för att dela upp problem med hjälp av funktioner är bottom-up
- Då börjar man med att skapa enkla verktyg man tror sig behöva och sätter sedan ihop dem till allt mer avancerade verktyg
- fördelar/nackdelar?