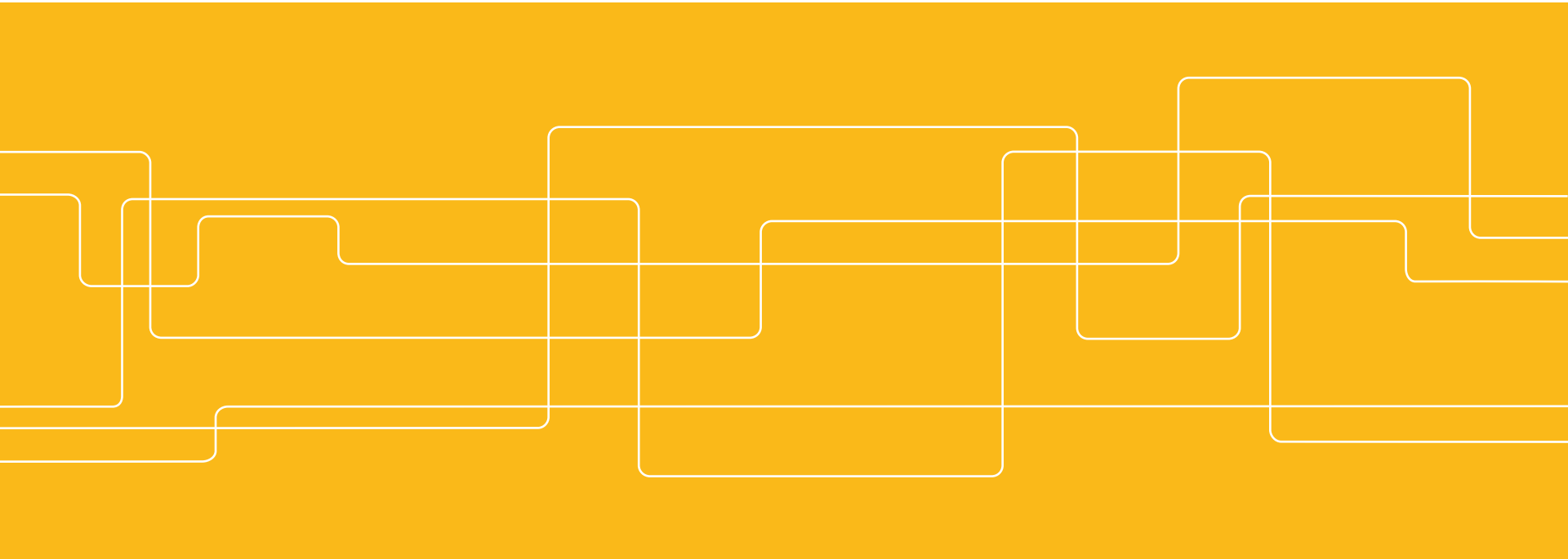
Funktioner och programstruktur

Föreläsning 5



Dagens kluring

```
int v[10]={1,2,3,4,5,6,7,8,9,10};
int i;
for(i=0;i<9;i++)
{
    v[i]=v[i+1];
}
//Hur ser v ut?
```

c89

```
#include <stdio.h>
#define DIM 10

int main(void)
{
    int v[DIM];
    int i;
    for(i=0;i<DIM;i++)
        if(i%2!=0)
            v[i]=1;
        else
            v[i]=0;
    for(i=0;i<DIM;i++)
        printf("%d",v[i]);

    return 0;
}
```

c99

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

int main(void)
{
    int dim=10;
    bool v[dim];
    for(int i=0;i<dim;i++)
        if(i%2!=0)
            v[i]=true;
        else
            v[i]=false;
    for(int i=0;i<dim;i++)
        printf("%d",v[i]);

    return 0;
}
```

Funktioner

- Vad är en funktion och hur gör man
- Varför använder vi funktioner
- Biblioteksfunktioner (math)
- Lokala och globala variabler
- Olika antal in- och ut-parametrar
- Arrays som in-parametrar
- Top-down-programmering

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:

$$3 \rightarrow \text{(dubblar talet)} \rightarrow 6$$

Exempel

En funktion räknar ut arean av en cirkel

Den behöver veta radien. Radien kallas för en in-parameter, argument, input, eller bara parameter.

Med hjälp av radien räknar den ut arean. Arean kallas för returvärde eller output.

$$2.00 \rightarrow \text{(arean beräknas)} \rightarrow 12.57$$

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));
    return 0;
}
```

main- är en funktion, programmets huvudfunktion

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

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));
    return 0;
}

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

Funktionsdeklaration!

Olika antal in- och ut-parametrar

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;
}
```

,men enfunktion kan inte returnera mer än en utparameter

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 och vad får vi ut.

Lokala och globala 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,

Array som in-parameter

```
#include <stdio.h>

int max(int v[],int n)
{
    int i,max=v[0];
    for(i=1;i<n;i++)
    {
        if(max<v[i])
        {
            max = v[i];
        }
    }
    return max;
}

int main(void)
{
    int a[] = {1,3,6,2,8,1};
    printf("Storsta vardet ar %d",max(a,6));
    return 0;
}
```

Biblioteksfunktioner

Att funktioner gör det lätt att använda kod man inte själv skrivit har vi redan använt vid ett flertal tillfällen. Vi har anropat funktioner såsom printf, scanf, rand, time.

Ett mycket användbart bibliotek heter math.h. Där finns

Trigonometri: ***cos, sin, tan, acos, asin, atan***

e och naturliga logaritmen: ***exp, log***

Upphöjt i och roten ur: ***pow,sqrt***

och M_PI som vi tidigare såg (egentligen inte standard och fungerar ej i c99)

Kolla gärna upp math.h och andra C-bibliotek!

Top-down-programmering

-är ett sätt att närma sig problemlösning där vi löser ett problem utgående ifrån att vi redan har funktioner som gör allt vi skulle kunna tänkas behöva göra. Sedan skriver vi dessa fantastiska funktioner på samma sätt. På detta sett delar man naturlig upp ett problem i allt mindre bitar till dess att det blir trivialt att lösa delproblemen.

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 ☹

```c
#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);
}
```


```
#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?

Studieanvisningar F5

- Gör K9 E1, E2, E3. Skriv också ett huvudprogram som testar att dina funktioner fungerar som det är tänkt.
- Läs 9.1-9.5
- Gå tillbaka till bubbelsorteringsprogrammet och försök dela upp det i funktioner på lämpligt sätt. Använd inga globala variabler.
- Läs första delen av 9.6 (sid 204-205)
- Skriv en fakultetsfunktion liknande den i boken (9.6) men lägg till en printf sats som skriver ut värdet på n. Anropa sedan funktionen från main och se hur den anropar sig själv.
- Gör K9 P6, P8
- Läs kapitel 10, jobba igenom programexemplet i 10.5!
- Gör K10 P2
- Svara på instuderingsfrågor
- Gör fler uppgifter om du hinner

E-excercises, P-programming projects