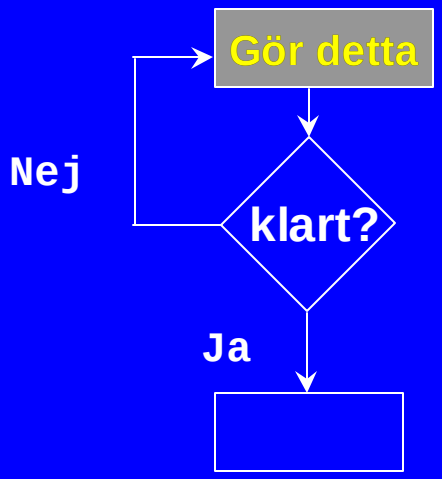
Arbeta med funktioner

Att delegera jobb
- Funktioner och funktionsanrop
- Deklaration och definition
- Globala och lokala variabler

Men först….
- Lite repetition av styrstrukturer (flödeskontroll)

Flödeskontroll - Repitition

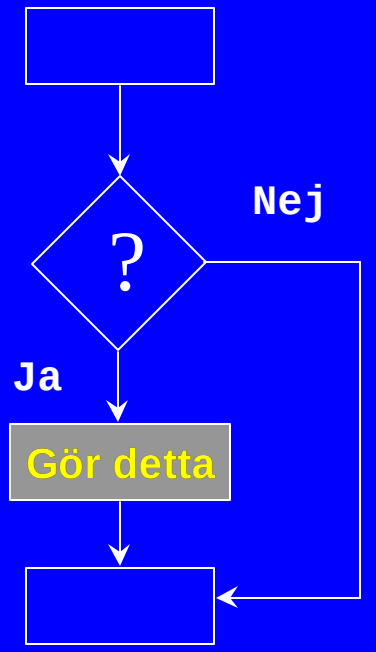
Iteration




```
for(){}

while(){}
do{

} while()
```

Selektion

```
If (){ }
else {}

swith(){
   case 1:
   case 2:
}
```



Flödeskontroll - for

Iteration: for()

Används om antalet gånger något skall utföras är känt

```
for(int antal = 0; antal < amax; antal++){
     summa += lista[i];
     cout << ”summa = ” << summa;
}
```

Flödeskontroll - while

Iteration: while

Används om något skall utföras så länge som…

```
float rest = 500.0;  // villkorsvariabeln
while(rest >= 1){
     rest = rest/3:
     cout << ”\nrest = ” + rest;
}
// Villkorsvariabeln måste ändras i loopen !
```

Flödeskontroll - do … while

Iteration: do …while

Används om något skall utföras så länge som…*minst en gång*

```
do{
     cin  >> val  ;//nytt värde
     cout << ”rest = ” << rest);
} while(val != 6)
```

Flödeskontroll - if

Urval: if ..else

```
if(<villkor>) sats;
if(<villkor>) sats;
else sats;
```

```
if(a < 4) mess = ”För lite”;
else  mess = ”Det räcker”;
cout << mess;
```

Urval - if... else...

```
ex)
if(a == 1)
   funk1();
   else
      if(a == 2)
            funk2();
            else
                  if(a == 3)
                        funk3();
                        else
                              if(a == 4)
                                    funk4();
                                    else
                                          funk5();
```

else ansluter till
närmaste if-sats!

Opraktiskt med många if …else

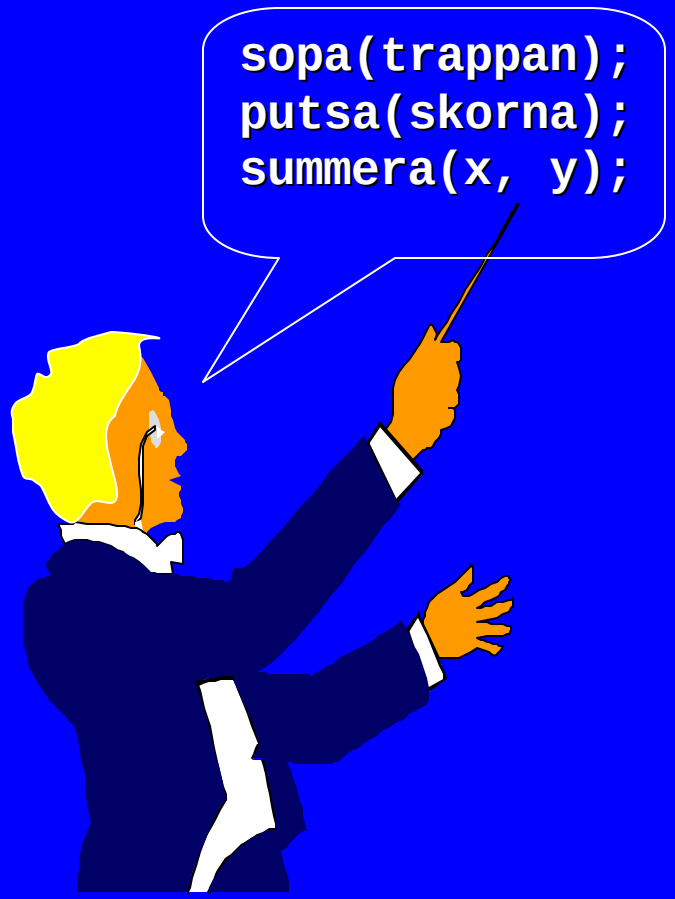
Använd switch!

Flödeskontroll - switch

Urval: switch

```
switch(val){
   case 1 : moveTo(x, y); break;
   case 2 : moveRel(dx, dy); break;
   case 3 : drawTo(x ,y); break;
   default: errmess = ”Ogiltigt val”;
}
```

Att delegera jobb



main( ) är boss
och delegerar jobb till
andra funktioner

sopa(vad)

putsa(vad)

main( )

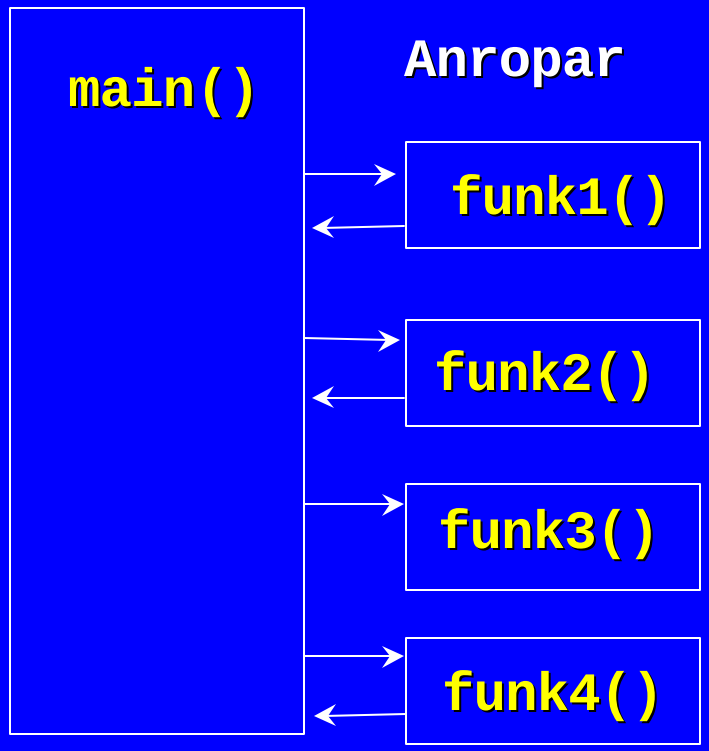
summera(a,b)

funk3( )

Funktioner

- Funktioner används för att få struktur i program.
- Funktioner bör vara specialiserade på sina uppgifter
- main() sköter det administrativa och delegerar jobbet till andra funktioner

Programstruktur



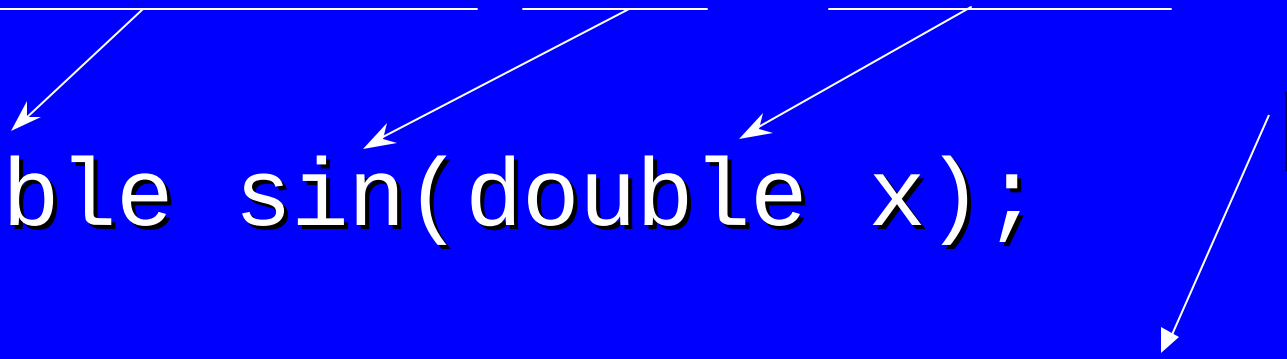
main() anropar funktioner som utför specialiserade uppdrag

Fördefinierade funktioner

```
#include<math.h>
#include<iostream.h>
main(){
  float vinkel, s;
  cout << ”Mata in en vinkel i  radianer-->”
  cin >> vinkel;
  s = sin(vinkel); // anrop
  cout << ”sin(”
       << vinkel << ”) = ” << s;
}
```

Deklaration

returvärdestyp namn argument



Deklareras i math.h - Koden i maths.lib

Inkluderas för att deklarationen av funktionen skall vara känd då den anropas.

Värdeanrop (värdet av x skickas med)

Egen funktion

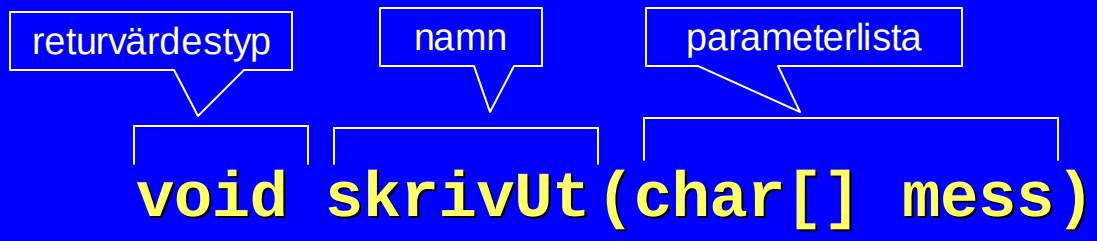
En funktion för utskrift av ett meddelande

funktionshuvud / signatur

```
void skrivUt(char[] mess)
{
     cout << mess;
}
```

kropp / body

Funktionens signatur



Funktioner måste deklareras

Funktionsdeklaration eller prototyp

```
#include <iostream.h>
void skrivUt(char[] mess);
void main(void){
   char message = ”Hej alla glada ):!”;
   skrivUt(message);
}

// Funktionen skriver ut en textsträng
void skrivUt(char[] mess){
   cout << mess;
}
```

deklaration

definition

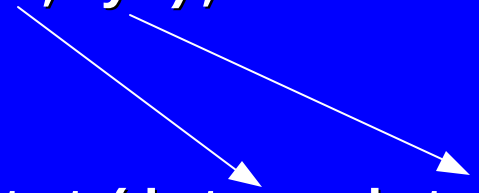
En beräkningsfunktion

```
#include <iostream.h> // skrivut.cpp
void skrivUtResultat(int, int);
void main(void){
   int x, y;
   cout << ”Mata in två tal -->”;
   cin >> x >> y;
   skrivUtResultat(x, y);
}
// Beräkna och skriv ut resultat
void skrivUtResultat(int a, int b){
   int res = a * b;
   cout << ”Resultatet är ” << res;
}
```

Formella parametrar

Funktionsanrop:

```
skrivUtResultat( x, y );
```



```
void skrivUtResultat(int a, int b){
   int res = a * b;
   cout << ”Resultatet är ” << res;
}
```

a och **b** är *formella parametrar*

x och **y** är *aktuella parametrar*

Specialisera funktionerna

```
void skrivUtResultat(int a, int b){
   int res = a * b;
   cout << ”Resultatet är ” << res;
}
```

Funktionen både beräknar och skriver ut resultat
   mindre generell - mindre användbar

Bättre med en beräkningsfunktion
   och en utskriftsfunktion

Funktioner med returvärden

```
#include <iostream.h> // berakna.cpp
int multiplicera(int a, int b);
void main(void){
   int x, y, resultat;
   cout << ”Mata in två tal -->”;
   cin >> x >> y;
   resultat = multiplicera(x, y);
   cout << ”Resultatet är ” << resultat;
}
int multiplicera(int a, int b){
   int res = a * b;
   return res;
}
```

deklaration

anrop

definition

return

Stacken

Funktionen tur och retur

```
resultat =
   multiplicera(x, y);
```

Anropet:

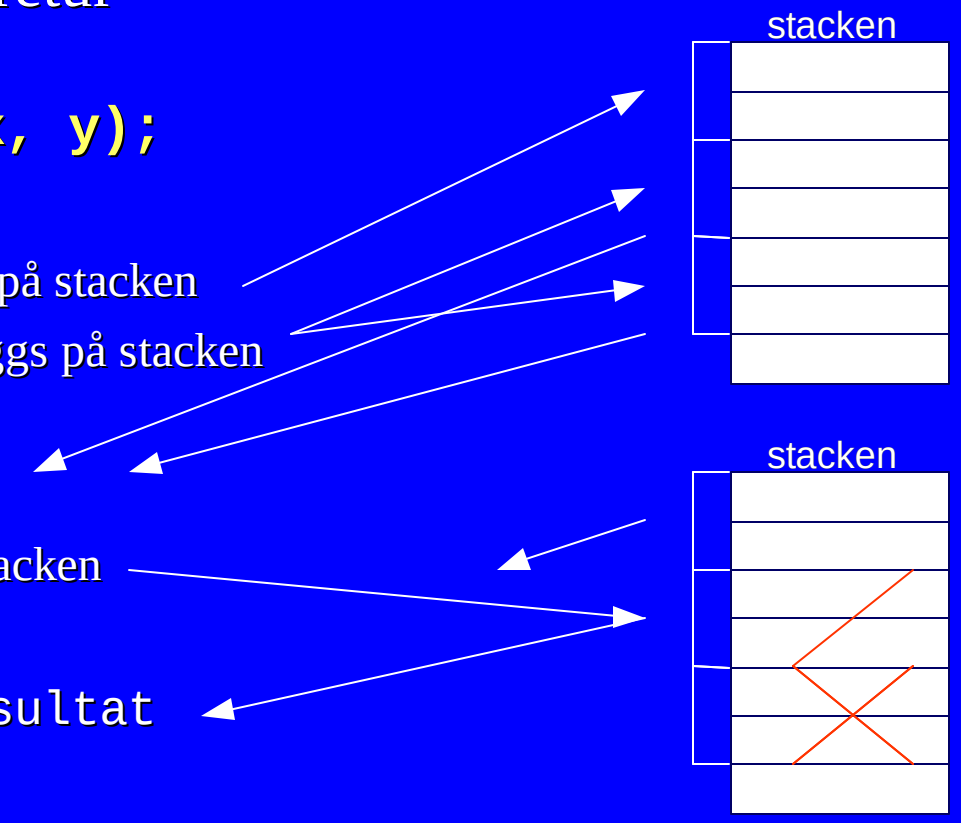
- återhoppsadress läggs på stacken
- värdena för x och y läggs på stacken

Funktionen körs

- värdet x*y beräknas
- returvärdet läggs på stacken

Returnera

- tilldelning sker till `resultat`
- programmet fortsätter efter anropssatsen



Överföring av värden

Vid anrop av en funktion
- läggs återhoppsadress på stacken
- läggs aktuella parametervärden på stacken
- där de hämtas av den anropade funktionen

När funktionen avslutas
- läggs eventuella returvärden på stacken (`return q`)
- återhopp sker till den anropande satsen
  - (genom att returadressen hämtas av processorn)

Åter i den anropande funktionen
- eventuellt returvärde tas om hand (t.ex. tilldelning)
- körning av efterföljande sats...

Globala & lokala variabler

Globala variabler
- deklareras utanför alla funktioner
- är tillgängliga för alla funktioner

Lokala variabler deklareras inom ett funktionsblock
- existerar endast inom blocket
- dör när körningen av blocket avslutas

```
float fglob;
float kub(){
   float ikubik;
   ikubik = fglob*fglob*fglob;
   return ikubik;
}
```

global

lokal

ikubik dör här!

Globala & lokala variabler

```
#include <iostream.h>
// Deklaration av globala variabler [ globloc1. cpp ]
float fglob;
// Deklaration av funktioner
void fixa();

void main(void){
   fglob = 1234.56;
    fixa();
}

// Definition av funktionen fixa
void fixa(){
   cout << "\nfixa säger att fglob = " << fglob;
}
```

Globala & lokala variabler

Globala variabler kan nås från alla funktioner

```
void dona(){ // globloc2.cpp
   fglob = 1000;
    cout << "\ndona säger att fglob = " << fglob;
}
```

Den globala variabelns värde har nu ändrats

```
void lokal(){ // globloc3.cpp
   float fglob = 1300;
    cout << "\nlokal säger att fglob = " << fglob;
}
```

Den globala variabeln döljs av den lokala

Den globala variabeln ändras inte!

Funktioner med returvärden

Funktion för beräkning av n! (n-fakultet)

```
#include <iostream.h>
//deklaration av funktionen
int fakultet( int d );
void main( void ){
   int tal,fak;
   cin >> tal;
   fak = fakultet( tal ); // anrop av funktionen
   cout << tal << ”-fakultet = ” << fak;
}
// definition av funktionen
int fakultet( int d ){
} -->
```

Funktioner med returvärden

```
// definition av funktionen
int fakultet( int d ){
   int fak = 1;
   for ( ; d > 0 ; d-- )
      fak *= d;
   return fak;
}
```

Den lokala variabeln **`fak`** initieras till 1 (0-fakultet är 1)

`for`-loopen behöver ingen initieringssats

`d` räknas ner och **`fak`** multipliceras med **`d`** i loopen

`fak` returneras till anropande funktion **`main`**( )

Arrayer som parametrar

Arrayer (vekorer) som parametrar...

och som returvärden.

Går det?

Svaret: Både Ja och Nej!

Hur hanteras strukturer (struct) som parameter?

Vad händer om man skickar med en pekare?

Kan man returnera en pekare?

Missa inte den spännande forts...