TDDC76 – Programmering och datastrukturer

Kursintroduktion och grunder i C++

Eric Ekström
Klas Arvidsson
Institutionen för datavetenskap

LINKÖPINGS UNIVERSITET

Agenda

1 Allmän info
2 Namespace
3 Fler strömmar
4 Operatorer
5 Scope
6 Funktioner
7 Vektorer
8 Upprepning
9 Aggregat
0 Inkludering

LINKÖPINGS UNIVERSITET

Allmän info

Anmälan till labgrupp

- Kom ihåg att anmäla er i WebReg
- Det som benämns “LAB2” i WebReg (och LADOK) motsvarar hela labserien i HT1 (lab 0 – 3)
- Om du inte hittat en labparter, då finns det en grupp för det – vi löser resten på labpasset

Programmering i C++

Vad ska vi göra idag?

Hur lata får vi egentligen vara som programmerare?

Using namespace std

Konsten att slippa skriva std::

- En av de mest std:: intensiva exemplen igår

program.cc

```
#include <iostream>
#include <iomanip>

int main()
{
  std::cout << std::setw(6)
            << std::setfill('.')
            << 10 << std::endl;
  return 0;
}
```

Using namespace std

Konsten att slippa skriva std::

- En av de mest std:: intensiva exemplen igår

program.cc

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
  cout << setw(6)
       << setfill('.')
       << 10 << endl;
  return 0;
}
```

Namespace

Vad är det bra för?

- Samlar funktioner och variabler under ett namn
  - Som att samla medlemmar i en förening
- Kan särskilja mellan grupper som har samma namn på variabler, funktioner
- `using namespace [namn]` använder vi då det inte finns en krock mellan namn

Andra typer av strömmar

Koncept

- En ström är en datakälla vi kan hämta tecken från eller ett datamål vi kan skicka tecken till
- Tecken kommer i en viss ordning som inte kan ändras
- Tecken mellanlagras automatiskt i strömmen för effektivare hantering
- Fungerar enhetligt oberoende av datakälla/datamål

Andra typer av strömmar

Sammanfattning av strömtyper

```
#include <iostream>
istream cin;                    // cin har tangentbordet som datakälla
ostream cout;                   // cout har terminalen som datamål

#include <fstream>
ifstream infile{"config.txt"}; // infile har angiven fil som datakälla
ofstream outfile{"save.txt"};  // outfile har angiven fil som datamål

#include <sstream>
istringstream iss{teststring}; // iss har angiven sträng som datakälla
ostringstream oss{};           // oss samlar på sig all utdata
oss.str();                     // och kan ge tillbaka den som en sträng
```

Inläsning från fil

```
string line {};
ifstream ifs {"fil.txt"};

while ( getline(ifs, line) )
{
  cout << line << endl;
}

ifs.close();
```

Smidig inläsning rad för rad

Med flera data per rad!

```
string line{};
while ( getline(ifs, line) )
{
  istringstream iss{line};
  string name{};
  int age;

  if ( iss >> name >> age )
  {
     cout << name << " är " << age << " år ung!" << endl;
  }
}
```

Operatorer vi kan använda i C++

Strömoperatorer

- Vi har använt två operatorer flitigt, strömoperatorerna << och >>
- De är tillämpningsbara på alla typer av strömmar

- (Notera att för andra datatyper betyder << och >> något annat… det blir alltså inte kompileringsfel)

Operatorer vi kan använda i C++

Hur många operatorer finns det egentligen?

Operatorer vi kan använda i C++

Hur många operatorer finns det egentligen?

- Aritmetiska operatorer
  - Ex: +, -, *, /, % (+=, -=, *=, /=, ++, --)
- Jämförelseoperatorer
  - Ex: (==, !=, <, >, <=, >=)
- Logiska operatorer
  - Ex: (||, &&, !)
- Strömoperatorer
  - Ex: (<<, >>)
- Bitoperatorer
  - Ex: (<<, >>, |, &, ^)
- Minnesoperatorer
  - Ex: (*, &, ->)

Operatorer vi kan använda i C++

Aritmetiska operatorer

- C++ har de vanligaste matematiska operatorerna inbyggt
  - +, -, *, /
  - Fungerar som ”vanligt”: `x = 4 + 2;`
  - Beräkningsordning densamma som matematiken, kan ändras med parenteser

Operatorer vi kan använda i C++

Aritmetiska operatorer

- Finns ett undantag – heltalsdivision!
- Två heltal (int) dividerade ger ett heltal
  - Matematiskt: a/b = c + d/b
  - C++: a/b = c
  - Resten (d) ges av modulo-operatorn %

- Ex : `x = 7/3` `y = 7%3`

  x : 2 y: 1

Operatorer vi kan använda i C++

Sammansatt tilldelning

- Vi har en heltalsvariabel x och vill lägga till 7
  - Jobbigt att skriva:

```
int x{1};
x = x + 7;
```

  - Skriver istället

```
int x{1};
x += 7;
```

- Fungerar för alla aritmetiska operatorer (och några fler)

Operatorer vi kan använda i C++

Stegning

- Öka (++) eller minska (--) en variabel med värdet ett
- Finns prefix- (++x) och postfix-versioner (x++)

```
int x{1};
++x;
--x;
x++;
x--;
```

Operatorer vi kan använda i C++

Stegning

- Öka (++) eller minska (--) en variabel med värdet ett
- Finns prefix- (++x) och postfix-versioner (x++)

```
int x{1};
++x;
--x;
x++;
x--;
```

- Vad är skillnaden mellan ++x och x++?

Operatorer vi kan använda i C++

Stegning - prefix

- Prefix-stegning används oftast
- Ökar värdet på variabeln och resultatet är det nya värdet

```
int x{1};
cout << x << endl;
cout << ++x << endl;
cout << x << endl;
```

```
1
2
2
```

Operatorer vi kan använda i C++

Stegning - postfix

- Ökar värdet på variabeln och resultatet är det gamla värdet
- Anropet x++ motsvarar följande beräkning (resultatet är temp):

```
int temp{x};
x = x + 1;
```

Operatorer vi kan använda i C++

Stegning - postfix

- Ökar värdet på variabeln och resultatet är det gamla värdet
- Anropet x++ motsvarar följande beräkning (resultatet är temp):

```
int temp{x};
x = x + 1;
```

```
int x{1};
cout << x << endl;
cout << x++ << endl;
cout << x << endl;
```

```
1
1
2
```

Operatorer vi kan använda i C++

Postfix och prefix

- Vad skrivs ut?

```
int x{1};
++x;
cout << x << endl;
cout << x++ << endl;
cout << ++x << endl;
cout << x++ << endl;
cout << x << endl;
```

Operatorer vi kan använda i C++

Postfix och prefix

- Vad skrivs ut?

```
int x{1};
++x;                    //Resultatet 2, x: 2
cout << x << endl;      //Skriver ut 2
cout << x++ << endl;    //Resultatet 2, x: 3
cout << ++x << endl;    //Resultatet 4, x: 4
cout << x++ << endl;    //Resultatet 4, x: 5
cout << x << endl;      //Skriver ut 5
```

Operatorer vi kan använda i C++

Jämförelseoperatorer

- Vi vill ofta jämföra värden
- Alla logiska operatorer ger ett logisk värde som svar
  - `bool` (`true`, `false`)
  - `==` (Likhet)
  - `!=` (Olikhet)
  - `<` (Mindre än)
  - `>` (Större än)
  - `>=` (Större än eller lika)
  - `<=` (Mindre än eller lika)

```
int x{1};
if (x < 5)
{
  //instruktioner
}
```

Operatorer vi kan använda i C++

Jämförelseoperatorer

- Hur många av de andra jämförelserna kan du skapa genom att skriva om dem till ett uttryck som bara använder Mindre än?
  - == (Likhet)
  - != (Olikhet)
  - < (Mindre än)
  - > (Större än)
  - >= (Större än eller lika)
  - <= (Mindre än eller lika)
- Tänk igenom inför lab 1

Operatorer vi kan använda i C++

Logiska operatorer

- || - logiskt ELLER
- && - logiskt OCH
- ! – logiskt ICKE

```
int x{1}
if (x < 5 || x == 7)
{
  //instruktioner
}
if (x <= 5 && x >= 3)
{
  //instruktioner
}
if (!(x==4))
{
  //instruktioner
}
```

Operatorer vi kan använda i C++

Beräkningsordning

- Operatorer i sammansatta uttryck utvärderas i bestämd ordning
- Om du inte intuitivt vet vilken operator som utvärderas först, använd parenteser!
- Se Operator Precedence på [cpp-reference]

```
int x{ 1 + 2 * 3 - 4}

int x{ 1 + (2 * 3) - 4}

bool b{ true || false && true }

bool b{ true || (false && true) }
```

Operatorer vi kan använda i C++

Logiska operatorer

- Short-circuit evaluation används för || och &&
- Idé - Utvärdera först villkoret till vänster om logiska operatorn:
  För &&: Om villkor false -> resultatet false
  För ||: Om villkor true -> resultatet true

```
int x{1}
if (x < 5 || x++)
{
  //instruktioner
}
```

```
int x{1}
if (x >= 5 && x >= 3)
{
  //instruktioner
}
```

Scope

En egen liten värld

- Ett program består av satser
- Satser som omsluts av { } ligger i ett block
  - Exempelvis satserna i int main():

```
int main()
{
  return 0;
}
```

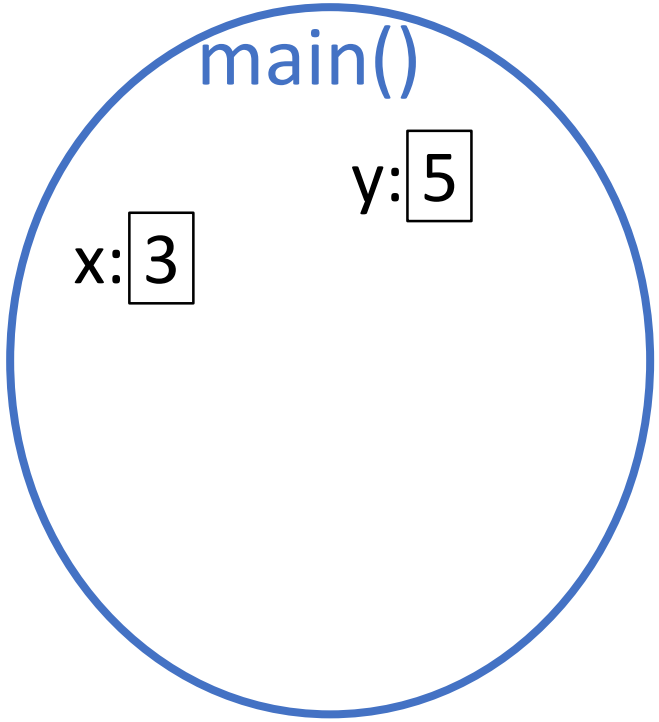
- I ett block har vi ett scope

Scope

En egen liten värld

- Vi kan skapa block var vi vill i koden
- De deklarationer som görs inom ett scope (ett blocks område) gäller endast där

```
int main()
{
> int x{3}, y{5};
  {
    int x{4}; //Ej fel, x skapas lokalt
    int z{};
    cout << x << ' ' << y; //skriver ut "4 5"
  }
  z = 6; //Fel, z finns inte här
}
```

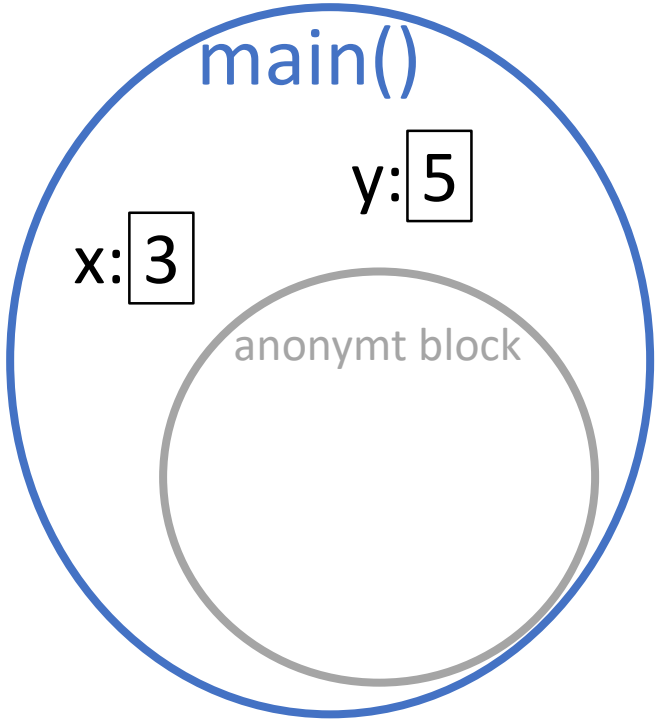


Scope

En egen liten värld

- Vi kan skapa block var vi vill i koden
- De deklarationer som görs inom ett scope (ett blocks område) gäller endast där

```
int main()
{
  int x{3}, y{5};
> {
    int x{4}; //Ej fel, x skapas lokalt
    int z{};
    cout << x << ' ' << y; //skriver ut "4 5"
  }
  z = 6; //Fel, z finns inte här
}
```

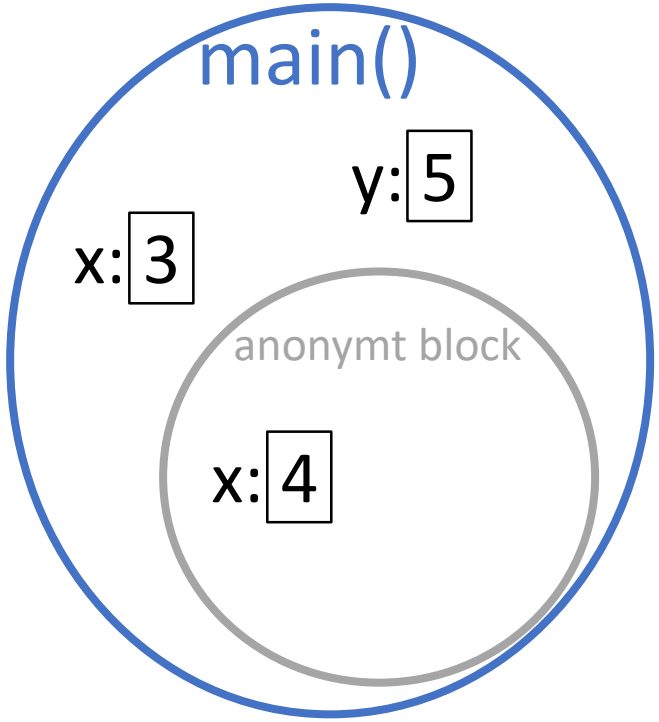


Scope

En egen liten värld

- Vi kan skapa block var vi vill i koden
- De deklarationer som görs inom ett scope (ett blocks område) gäller endast där

```
int main()
{
  int x{3}, y{5};
  {
>   int x{4}; //Ej fel, x skapas lokalt
    int z{};
    cout << x << ' ' << y; //skriver ut "4 5"
  }
  z = 6; //Fel, z finns inte här
}
```

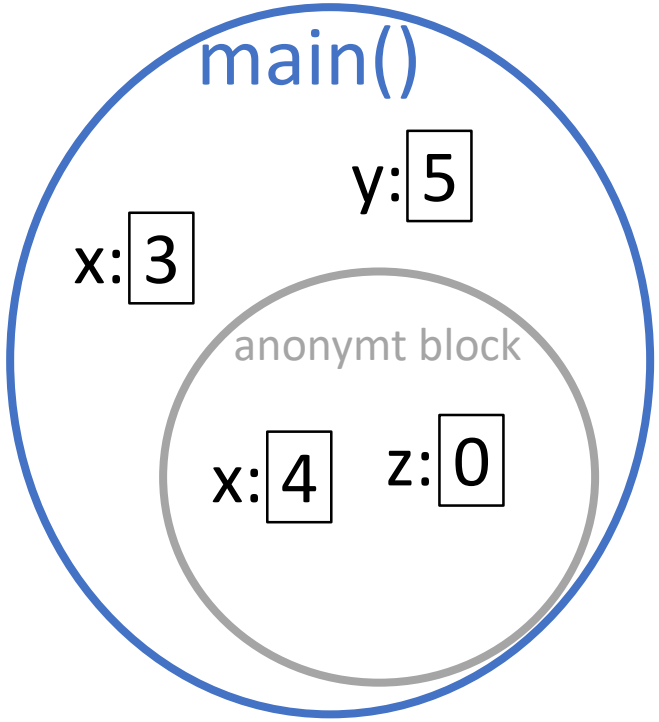


Scope

En egen liten värld

- Vi kan skapa block var vi vill i koden
- De deklarationer som görs inom ett scope (ett blocks område) gäller endast där

```
int main()
{
  int x{3}, y{5};
  {
    int x{4}; //Ej fel, x skapas lokalt
>   int z{};
    cout << x << ' ' << y; //skriver ut "4 5"
  }
  z = 6; //Fel, z finns inte här
}
```

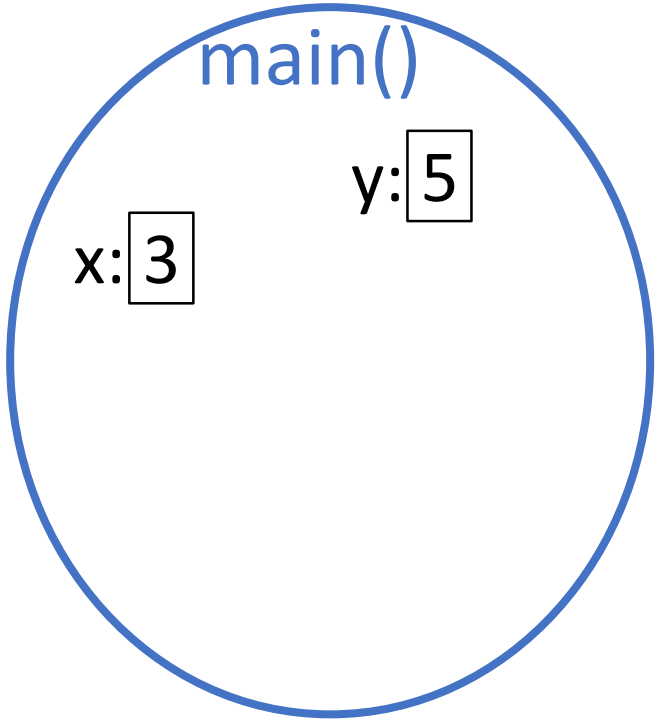


Scope

En egen liten värld

- Vi kan skapa block var vi vill i koden
- De deklarationer som görs inom ett scope (ett blocks område) gäller endast där

```
int main()
{
  int x{3}, y{5};
  {
    int x{4}; //Ej fel, x skapas lokalt
    int z{};
    cout << x << ' ' << y; //skriver ut "4 5"
> }
  z = 6; //Fel, z finns inte här
}
```



Funktioner

Vad, hur och varför använder vi dem?

- En funktion är en samling instruktioner
  - Ett block med ett namn
- Kan anropas och då utförs funktionens instruktioner
- Vi behöver inte veta exakt hur funktionen är uppbyggd för att använda den
- Går att återanvända om och om igen
- Ska ha exakt ett syfte och inga sidoeffekter

Funktioner

Vad, hur och varför använder vi dem?

```
#include <iostream>
using namespace std;

void hello_world(); // Deklaration

int main()
{
  hello_world();     // Anrop
  cout << "-------------" << endl;
  hello_world();     // Anrop
}

void hello_world()   // Defintiion
{
  cout << "Hello, World!" << endl;
}
```

Funktioner

Hur skriver vi en funktionsdefinition?

```
[returtyp]  funktion_namn([möjliga parametrar])
{
  //Instruktioner
  [möjlig retursats];
}
```

Funktioner

Vad händer om jag glömmer returnera?

```
int slarver()
{
    // return 9;
}
int main()
{
    int x{0};
    x = slarver();
}
```

Funktioner

Vad händer om jag glömmer returnera?

- Komplettering på labben! Men du kan be kompilatorn rädda dig!

```
$ g++ slarver.cc
$
```

```
$ g++ -std=c++17 -Wall -Wextra -Weffc++ slarver.cc
slarver.cc: In function ‘int slarver()’:
slarver.cc:4:1: warning: no return statement in function
returning non-void [-Wreturn-type]
$
```

Funktioner

Vad händer om jag returnerar på flera ställen?

```
string day(int i){
    if ( i == 1 ) {
      return “Mån”; // avsluta funktionen när
                    // villkoret är uppfyllt
    }
    if ( i == 2 ) {
      return “Tis”;
    }
    return “Inte Mån eller Tis”;
}
```

Funktioner

Hur skriver vi en funktionsdefinition?

- En funktion som inte returnerar ett värde har returtyp `void`

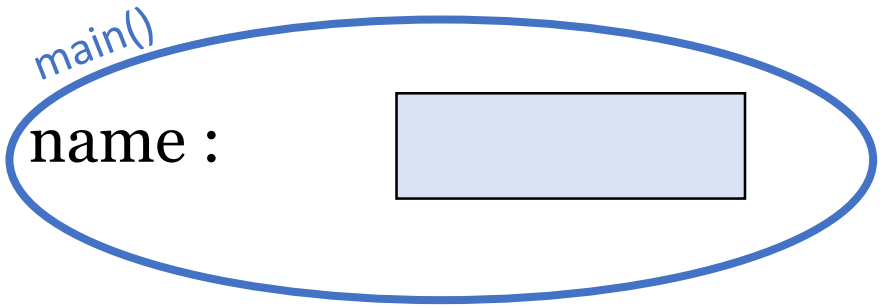
```
void hello_world()
{
    cout << "Hello, World!" << endl;
}
```

Funktioner

Hur skriver vi en funktionsdefinition?

```
string get_username()
{
    string username{};
    cout << "Skriv ditt användarnamn: " << endl;
    cin >> username;
    return username;
}
```

```
int main()
{
    string name{};
    name = get_username();
    return 0;
}
```

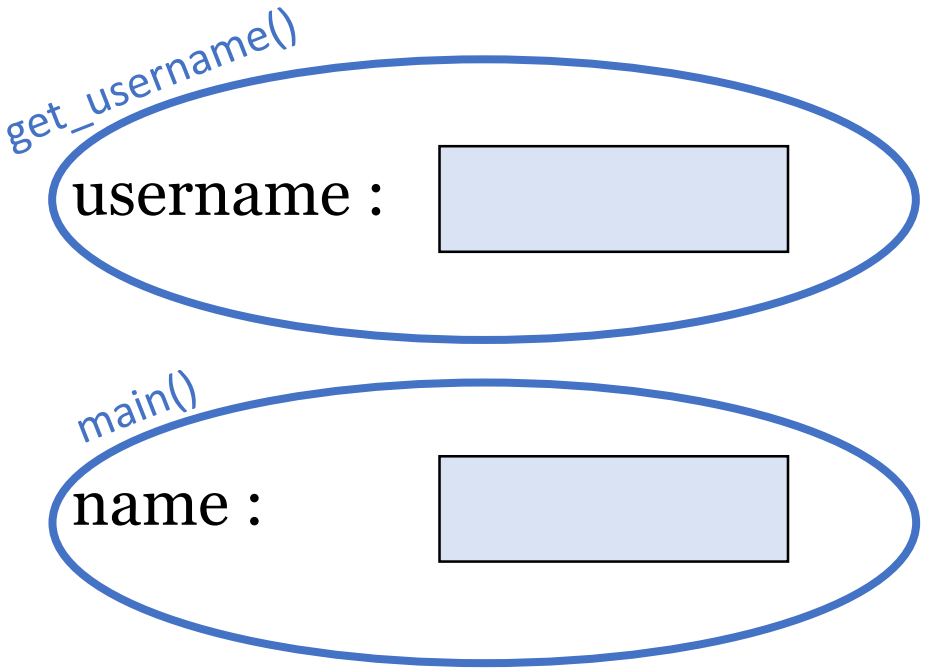


Funktioner

Hur skriver vi en funktionsdefinition?

```
string get_username()
{
    string username{};
    cout << "Skriv ditt användarnamn: " << endl;
    cin >> username;
    return username;
}
```

```
int main()
{
    string name{};
    name = get_username();
    return 0;
}
```

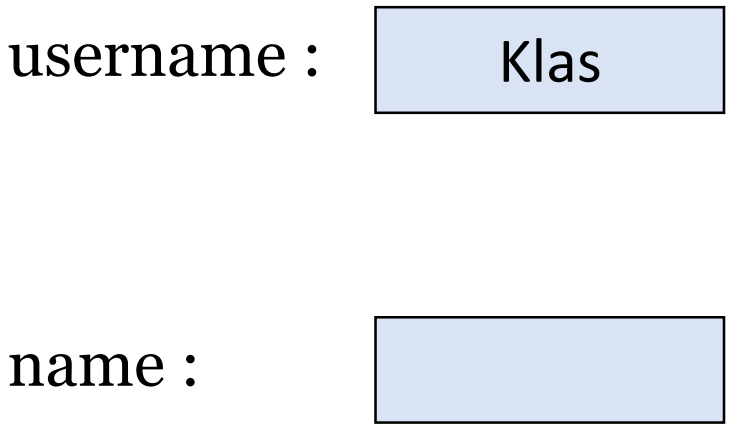


Funktioner

Hur skriver vi en funktionsdefinition?

```
string get_username()
{
    string username{};
    cout << ”Skriv ditt användarnamn: ” << endl;
    cin >> username;
    return username;
}
```

```
int main()
{
    string name{};
    name = get_username();
    return 0;
}
```

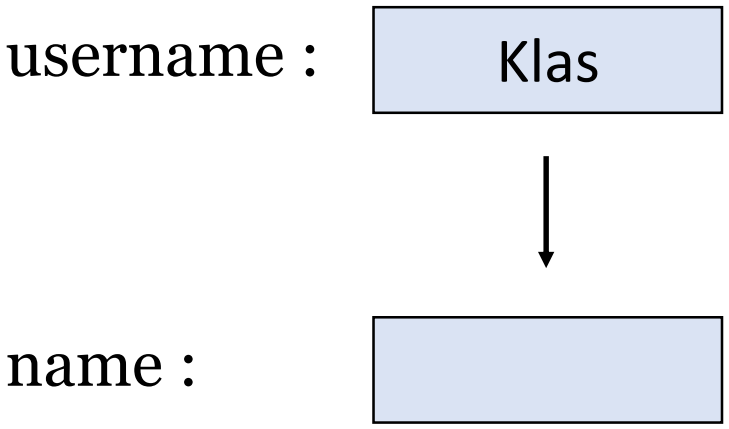


Funktioner

Hur skriver vi en funktionsdefinition?

```
string get_username()
{
    string username{};
    cout << "Skriv ditt användarnamn: " << endl;
    cin >> username;
    return username;
}
```

```
int main()
{
    string name{};
    name = get_username();
    return 0;
}
```



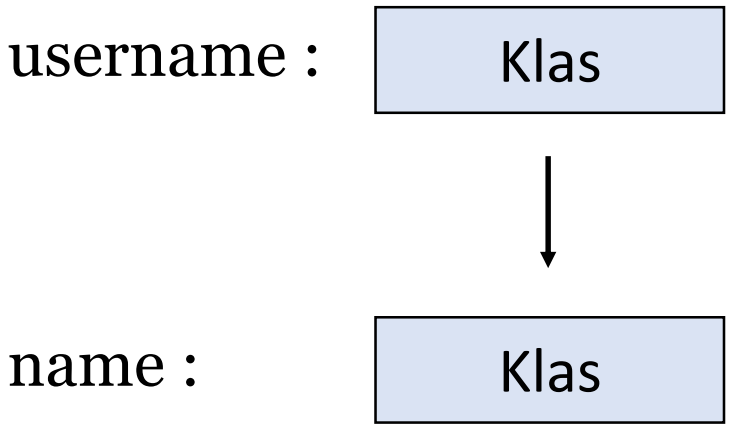
Funktioner

Hur skriver vi en funktionsdefinition?

- **Returvärdet kopieras ut.**

```
string get_username()
{
    string username{};
    cout << "Skriv ditt användarnamn: " << endl;
    cin >> username;
    return username;
}
```

```
int main()
{
    string name{};
    name = get_username();
    return 0;
}
```



Funktioner

Hur skriver vi en funktionsdefinition?

```
string get_username()
{
    string username{};
    cout << ”Skriv ditt användarnamn: ” << endl;
    cin >> username;
    return username;
}
```

```
int main()
{
    string name{};
    name = get_username();
    return 0;
}
```

name :

Klas

Funktioner

Hur fungerar parameteröverföring?

- Anroparen skickar ett argument (en variabel eller värde)
- En funktion tar emot argumentet i parametrar (variabler)
- Vid flera argument avgör ***ordningen*** vilket argument som sparas i respektive parameter.
  - Argument 1 -> Parameter 1
  - Argument 2 -> Parameter 2
  - osv…

Funktioner

Hur fungerar parameteröverföring?

- Exempel på parameteröverföring (värdet kopieras över)

Parameter (Kopia)

```
void introduceYourself(string first_name)
{
    cout << "Jag heter"
         << first_name
         << endl;
}
```

```
int main()
{
    string name{"Oskar"};
    introduceYourself(name);
    return 0;
}
```

Argument

name : Oskar

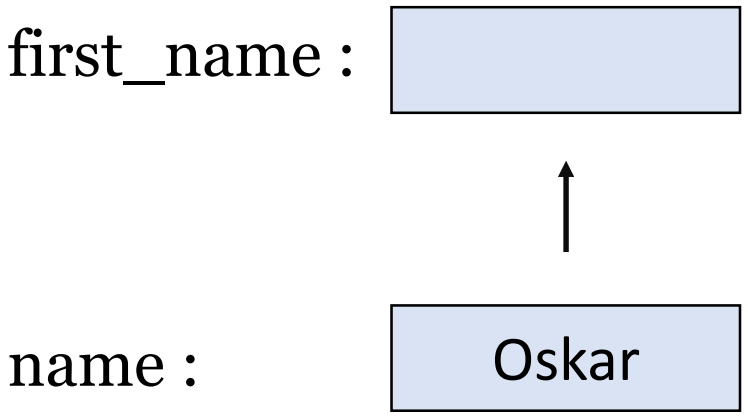
Funktioner

Hur fungerar parameteröverföring?

- Exempel på parameteröverföring (värdet kopieras över)

```
void introduceYourself(string first_name)
{
    cout << "Jag heter"
         << first_name
         << endl;
}
```

```
int main()
{
    string name{"Oskar"};
    introduceYourself(name);
    return 0;
}
```



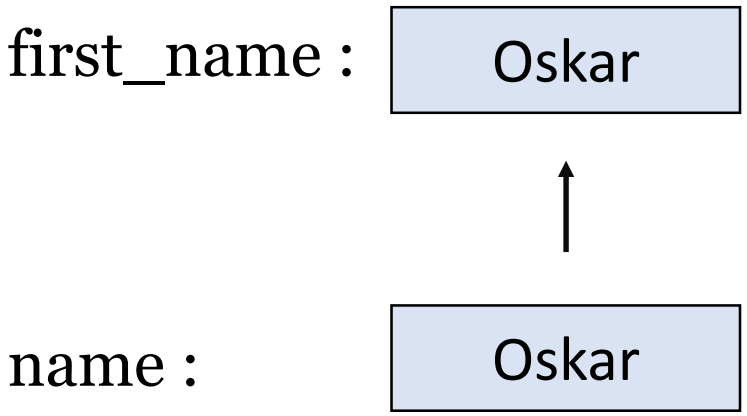
Funktioner

Hur fungerar parameteröverföring?

- Exempel på parameteröverföring (värdet kopieras över)

```
void introduceYourself(string first_name)
{
    cout << "Jag heter"
         << first_name
         << endl;
}
```

```
int main()
{
    string name{"Oskar"};
    introduceYourself(name);
    return 0;
}
```



Funktioner

Hur fungerar parameteröverföring?

- Argument kan överföras på andra sätt
  - const – parametern får inte ändras (ex: `string const name`)
  - & - anroparen låter funktionen låna sin variabel (ex: `string & name`)
  - const & - får låna, fast inte ändra! (ex: `string const & name`)

Referenser

Vad är en referens?

- Alla variabler är sparade någonstans
- Istället för att ge värdet i variabeln så ger vi själva lådan till variabeln
- En annan person kan då använda och ändra värdet i den
- Kan referera till variabler i ett annat anropande scope
  - Referens till variabler i funktion som gör funktionsanrop: OK – Ett anropande scope existerar hela tiden medan det aktuella körs klart
  - Referens (via returvärde) till variabler i funktion som vi anropat – FEL: Ett scope existerar endast medan funktionen exekverar

Funktioner

Hur fungerar parameteröverföring?

- Exempel på parameteröverföring (värdet kopieras över men får ej ändras)

```
void introduceYourself(string const first_name)
{
    cout << "Jag heter"
         << first_name
         << endl;
}
```

```
int main()
{
    string name{"Oskar"};
    introduceYourself(name);
    return 0;
}
```

name : Oskar

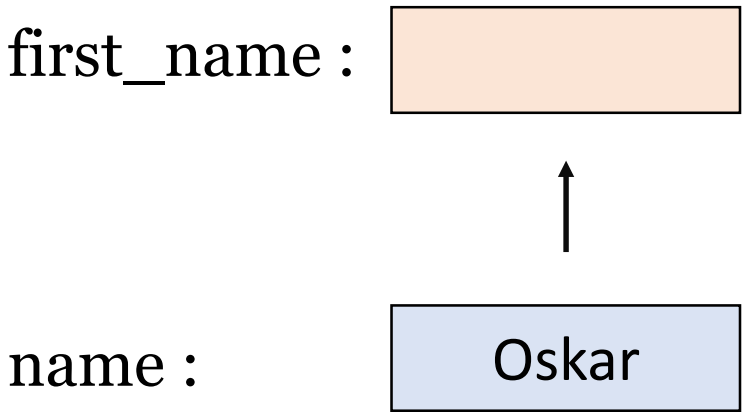
Funktioner

Hur fungerar parameteröverföring?

- Exempel på parameteröverföring (värdet kopieras över men får ej ändras)

```
void introduceYourself(string const first_name)
{
    cout << "Jag heter"
         << first_name
         << endl;
}
```

```
int main()
{
    string name{"Oskar"};
    introduceYourself(name);
    return 0;
}
```



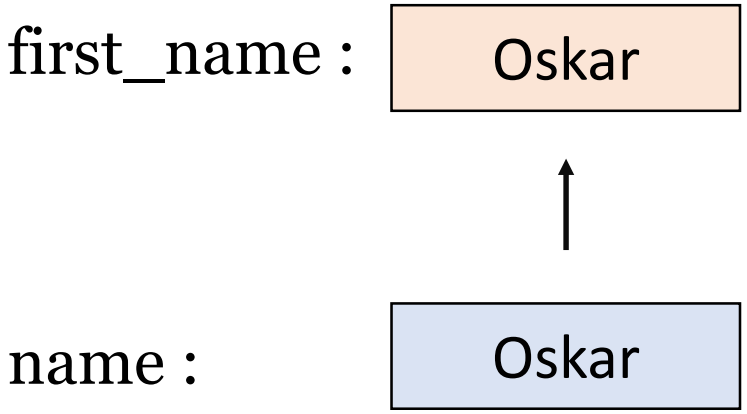
Funktioner

Hur fungerar parameteröverföring?

- Exempel på parameteröverföring (värdet kopieras över men får ej ändras)

```
void introduceYourself(string const first_name)
{
    cout << "Jag heter"
         << first_name
         << endl;
}
```

```
int main()
{
    string name{"Oskar"};
    introduceYourself(name);
    return 0;
}
```



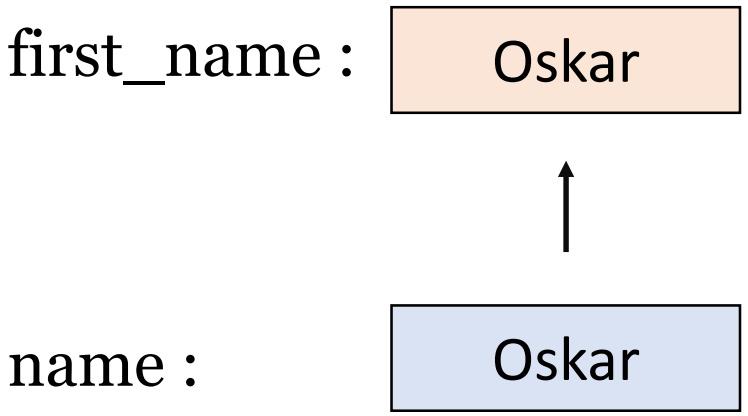
Funktioner

Hur fungerar parameteröverföring?

- Exempel på parameteröverföring (värdet kopieras över men får ej ändras)

```
void introduceYourself(string const first_name)
{
    first_name = "Klas";
    cout << "Jag heter"
         << first_name
         << endl;
}
```

```
int main()
{
    string name{"Oskar"};
    introduceYourself(name);
    return 0;
}
```



Funktioner

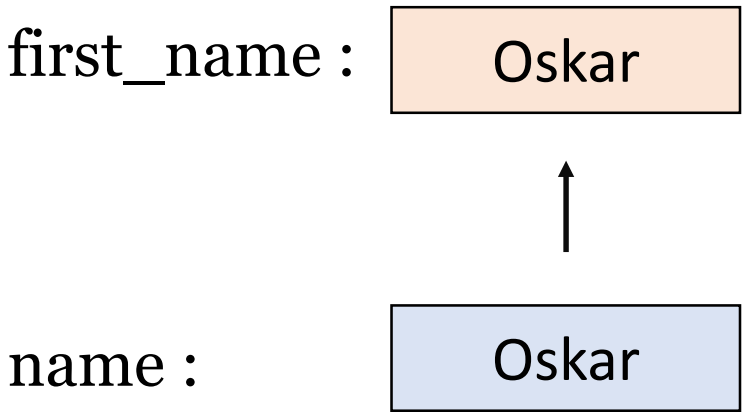
Hur fungerar parameteröverföring?

- Exempel på parameteröverföring (värdet kopieras över men får ej ändras)

```
void introduceYourself(string const first_name)
{
    first_name = "Klas";
    cout << "Jag heter"
         << first_name
         << endl;
}
```

KOMPILERAR INTE

```
int main()
{
    string name{"Oskar"};
    introduceYourself(name);
    return 0;
}
```



Funktioner

Hur fungerar parameteröverföring?

- Exempel på parameteröverföring
  (referens, variabeln från anropande kod görs tillgänglig)

```
string get_username(string & username)
{
    cout << ”Skriv ditt användarnamn: ” << endl;
    cin >> username;
    return username;
}
```

```
int main()
{
    string name{};
    get_username(name);
    return 0;
}
```

name :

Funktioner

Hur fungerar parameteröverföring?

- Exempel på parameteröverföring
  (referens, variabeln från anropande kod görs tillgänglig)

```
string get_username(string & username)
{
    cout << ”Skriv ditt användarnamn: ” << endl;
    cin >> username;
    return username;
}
```

```
int main()
{
    string name{};
    get_username(name);
    return 0;
}
```

username :

name :

Funktioner

Hur fungerar parameteröverföring?

- Exempel på parameteröverföring
  (referens, variabeln från anropande kod görs tillgänglig)

```
string get_username(string & username)
{
    cout << ”Skriv ditt användarnamn: ” << endl;
    cin >> username;
    return username;
}
```

```
int main()
{
    string name{};
    get_username(name);
    return 0;
}
```

username :

name :

Klas

Funktioner

Hur fungerar parameteröverföring?

- Exempel på parameteröverföring (const-referens, variabeln från anropande kod görs tillgänglig read-only)

```
void introduceYourself(string const & first_name)
{
    cout << "Jag heter"
         << first_name
         << endl;
}
```

```
int main()
{
    string name{"Oskar"};
    introduceYourself(name);
    return 0;
}
```

name :

Oskar

Funktioner

Hur fungerar parameteröverföring?

- Exempel på parameteröverföring (const-referens, variabeln från anropande kod görs tillgänglig read-only)

```
void introduceYourself(string const & first_name)
{
    cout << "Jag heter"
         << first_name
         << endl;
}
```

```
int main()
{
    string name{"Oskar"};
    introduceYourself(name);
    return 0;
}
```

first_name :

name :

Oskar

Funktioner

Hur fungerar parameteröverföring?

- Exempel på parameteröverföring (const-referens, variabeln från anropande kod görs tillgänglig read-only)

```
void introduceYourself(string const & first_name)
{
    first_name = "Klas";
    cout << "Jag heter"
         << first_name
         << endl;
}
```

```
int main()
{
    string name{"Oskar"};
    introduceYourself(name);
    return 0;
}
```

first_name :
name :

Oskar

Funktioner

Hur fungerar parameteröverföring?

- Exempel på parameteröverföring (const-referens, variabeln från anropande kod görs tillgänglig read-only)

```
void introduceYourself(string const & first_name)
{
    first_name = "Klas";
    cout << "Jag heter"
         << first_name
         << endl;
}
```

KOMPILERAR INTE

```
int main()
{
    string name{"Oskar"};
    introduceYourself(name);
    return 0;
}
```

first_name :
name : Oskar

Funktioner

Vilka parametrar ska jag välja? Enligt konvention:

Viktigt!

1. Vill jag ändra på argumentet?
   ➡ Referens (&)
2. Är det en inbyggd datatyp (int, double, char, bool)?
   ➡ Kopia (const Kopia valfritt)
3. Är det en sammansatt datatyp (string, vector, stack, egna klasser, …)?
   ➡ Ej ändringsbar Referens (const &)

Vektor

- En vektor lagrar en sekvens värden (jämför lista i python)
- Vi måste ange datatypen för värden som ska lagras i sekvensen
- Alla värden måste ha samma datatyp

```
#include <vector>

int main()
{
  std::vector<int> v{ 1, 5, 2 };
  std::cout << v.at(0) << v.at(1) << v.at(2);
  return 0;
}
```

Vektor

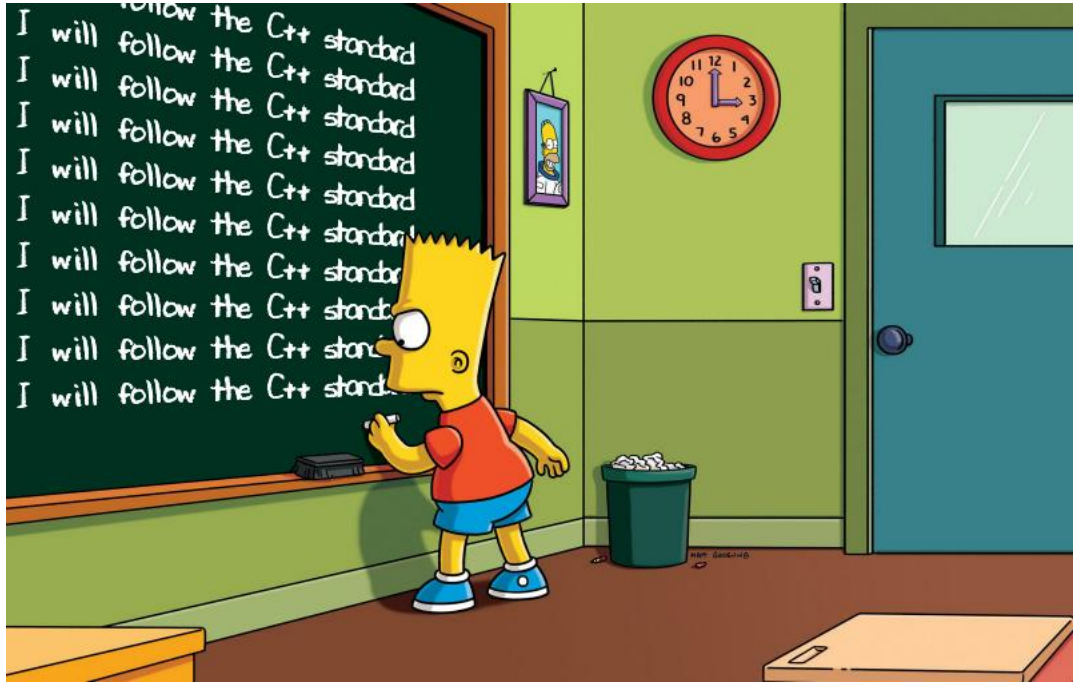
- Vi kan lägga till värden sist i en vektor

```
#include <vector>

int main()
{
  std::vector<int> v{};

  v.push_back(1);
  v.push_back(5);
  v.push_back(2);

  std::cout << v.at(0) << v.at(1) << v.at(2);
  return 0;
}
```

Upprepning

for, while och do-while

- Ibland vill få att något ska upprepas flera gånger
- Tråkigt för oss – inget problem för datorer



Upprepning

for, while, do-while

- På vilka olika sätt kan vi repetera?
  - Ett specifikt antal gånger
  - Tills något händer

Upprepning

for-loop

- for – Använd när du vet eller enkelt kan räkna ut exakt hur många gånger något ska upprepas

```
for (init; villkor; end)
{
    //Block med instruktioner
}
```

Upprepning

for-loop

```
#include <iostream>
using namespace std;
int main()
{
>    int x{0};
     for (int i{1}; i <= 3; ++i)
     {
          x += i;
     }
     cout << x << endl;
     return 0;
}
```

x:

0

Upprepning

for-loop

```
#include <iostream>
using namespace std;
int main()
{
    int x{0};
>   for (int i{1}; i <= 3; ++i)
    {
        x += i;
    }
    cout << x << endl;
    return 0;
}
```

x: 0

i: 1

Upprepning

for-loop

```
#include <iostream>
using namespace std;
int main()
{
    int x{0};
    for (int i{1}; i <= 3; ++i)
    {
>       x += i;
    }
    cout << x << endl;
    return 0;
}
```

x: 1

i: 1

Upprepning

for-loop

```
#include <iostream>
using namespace std;
int main()
{
    int x{0};
>   for (int i{1}; i <= 3; ++i)
    {
        x += i;
    }
    cout << x << endl;
    return 0;
}
```

x: 1

i: 2

Upprepning

for-loop

```
#include <iostream>
using namespace std;
int main()
{
    int x{0};
    for (int i{1}; i <= 3; ++i)
    {
>       x += i;
    }
    cout << x << endl;
    return 0;
}
```

x: 3

i: 2

Upprepning

for-loop

```
#include <iostream>
using namespace std;
int main()
{
    int x{0};
>   for (int i{1}; i <= 3; ++i)
    {
        x += i;
    }
    cout << x << endl;
    return 0;
}
```

x: 3

i: 3

Upprepning

for-loop

```
#include <iostream>
using namespace std;
int main()
{
    int x{0};
    for (int i{1}; i <= 3; ++i)
    {
>       x += i;
    }
    cout << x << endl;
    return 0;
}
```

x: 6

i: 3

Upprepning

for-loop

```
#include <iostream>
using namespace std;
int main()
{
    int x{0};
>   for (int i{1}; i <= 3; ++i)
    {
        x += i;
    }
    cout << x << endl;
    return 0;
}
```

x: 6

i: 4

Upprepning

for-loop

```
#include <iostream>
using namespace std;
int main()
{
    int x{0};
    for (int i{1}; i <= 3; ++i)
    {
        x += i;
    }
>   cout << x << endl;
    return 0;
}
```

x: 6

Upprepning

while-loop

- while – Använd när ett villkor kan styra hur länge ett upprepning ska fortsätta (Noll eller fler gånger)

```
while (villkor)
{
    //Block med instruktioner
}
```

Upprepning

while-loop

```
string name{};
cout << ”Vad heter du?” << endl;
cin >> name;

while (name != ”Klas”)
{
    cout << ”Bättre namn kan du. Vad heter du?” << endl;
    cin >> name;
}
cout << ”Nämen! Det var ett fint namn.” << endl;
```

Upprepning

do-while-loop

- do-while – Använd när ett villkor kan styra hur länge ett upprepning ska fortsätta (En eller fler gånger)

```
do
{
    //Block med instruktioner
}
while (villkor);
```

Upprepning

När ska vi använda det?

```
int num{};

do
{
  cout << ”Mata in ett tal över 4:” << endl;
  cin >> num;
{
while (num <= 4);
```

Upprepning

När ska vi använda det

- Skriver vi snarlik kod i följd som följer ett mönster så är det en signal att upprepning kan vara bra att använda

```
x += 2;
x += 3;
x += 4;
x += 5;
x += 6;
x += 7;
```

```
for (int i{0}; i < 6; ++i)
{
  x += 2 + i;
}
```

Upprepning

När ska vi använda det

- Ibland måste vi förstärka mönster för att kunna använda upprepning (vad är skillnaden mellan kodrutorna?):

```
cout << ”1: Måndag” << endl;
cout << ”2: Tisdag” << endl;
cout << ”3: Onsdag” << endl;
cout << ”4: Torsdag” << endl;
cout << ”5: Fredag” << endl;
```

```
vector<string> days{
  ”Mån”, ”Tis”, ”Ons”, ”Tors”, ”Fre”
};

for (int i{0}; i < days.size(); ++i)
{
  cout << i << ”: ” << days.at(i)
       << ”dag” << endl;
}
```

Upprepning

När ska vi använda det

- Ibland behöver vi upprepa samma typ av kod på flera ställen och upprepning(for,while) räcker inte alltid till:

```
cout << ”Mjölk kostar 5kr” << endl
     << ”Vill du köpa Mjölk? << endl;
…
cout << ”Saft kostar 3kr” << endl
     << ”Vill du köpa Saft? << endl;
…
cout << ”Vatten kostar 15kr” << endl
     << ”Vill du köpa Vatten? << endl;
```

- Då ska vi använda funktioner (fet och kursiv blir parametrar)!

Aggregat

- Vi kan samla data som tillsammans skapar en större betydelse
- Skapas som en ny datatyp med ett namn och de data som ska ingå

```
struct Person
{
  string name;
  int shoe_size;
  double length;
};
```

- En struct beskriver bara den nya datatypen. Ingen variabel skapas!

Aggregat

```
struct Person
{
  string name;
  int shoe_size;
  double length;
};
```

- Nu kan vi skapa variabler av den nya datatypen i våra program.

```
void print(Person const& p)
{
   std::cout << ”Namn: ”       << p.name      << std::endl
             << ”Skostorlek: ” << p.shoe_size << std::endl
             << ”Längd: ”      << p.length    << std::endl;
}

int main()
{
   Person examinator{ ”Klas”, 42, 1.89 };
   print(examinator);
   return 0;
}
```

Programmering i C++

Vad ska vi göra idag?

Hur lata får vi egentligen vara som programmerare?

Inkludering

Varför behöver vi inkludera? Vad händer?

- Varför?
  – Inte uppfinna hjulet igen eller kopiera in kod varje användning
- Vad?
  – koden vi inkluderar kopieras automatiskt in i filen

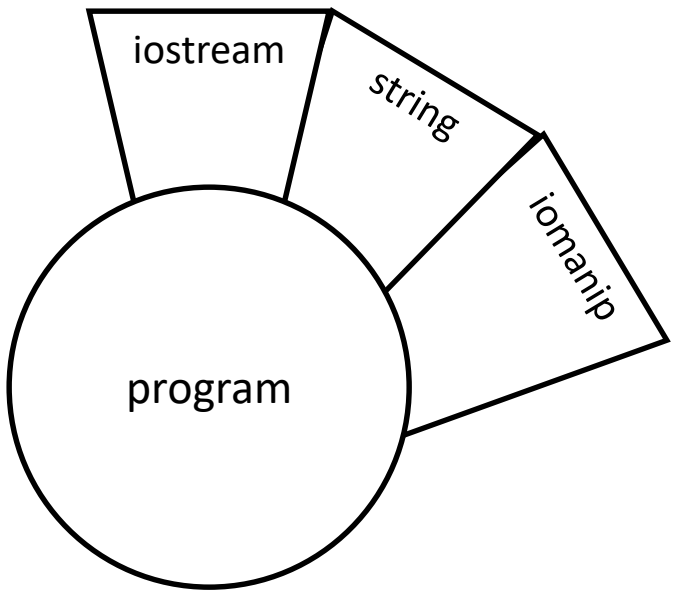
helloworld.cc

```
#include <iostream>
int main()
{
  std::cout << "Hello, World"
            << std::endl;
  return 0;
}
```

Inkludering

Varför behöver vi inkludera? Vad händer?

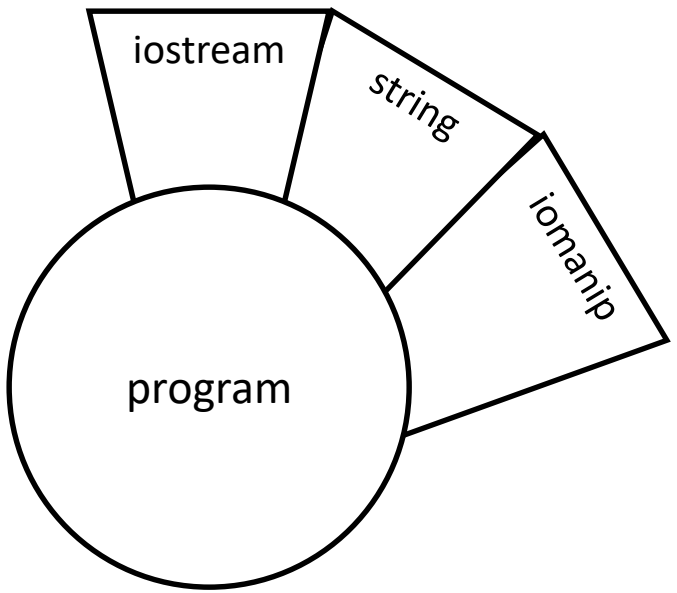
- Varför inkluderar vi bara inte alla bibliotek?



Inkludering

Varför behöver vi inkludera? Vad händer?

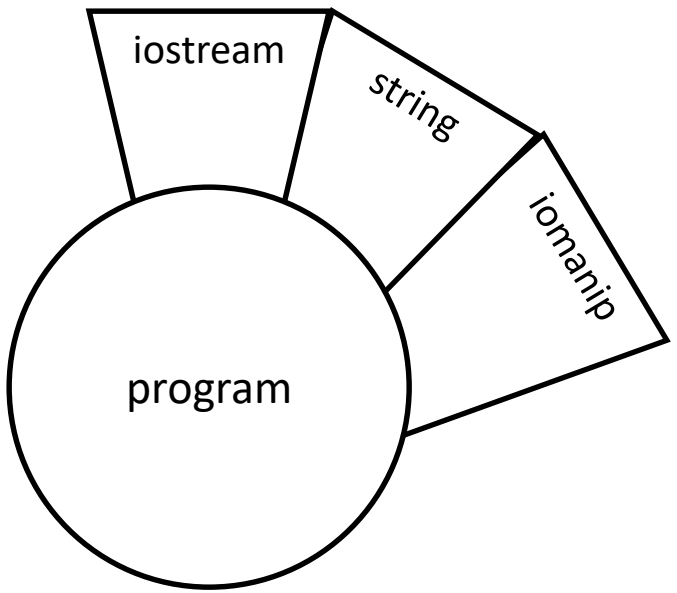
- Varför inkluderar vi bara inte alla bibliotek?
- För varje tillägg så blir programmet större och tyngre



Inkludering

Varför behöver vi inkludera? Vad händer?

- Varför inkluderar vi bara inte alla bibliotek?
- För varje tillägg så blir programmet större och tyngre
- Inkludera bara det vi behöver just nu



Registrera er i kursen och anmäl er i Webreg!

www.liu.se

LINKÖPINGS UNIVERSITET