TDDE18 & 726G77

Functions & struct

Christoffer Holm

Department of Computer and information science

LINKÖPING UNIVERSITY

LINKÖPING UNIVERSITY

Functions

Blocks

```
{ // start of block

  // body of block

} // end of block
```

Functions

Scope

```
int x{}; // global scope
int main()
{
  int y{}; // local scope
}
```

Functions

Scope & Blocks

```
int x{0};
{
  int x{1};
  {
    cout << x << " ";
    int x{2};
    cout << x << " ";
  }
  cout << x << " ";
}
cout << x << " ";
```

Functions

Scope & Blocks

```
int x{0};
{
  int x{1};
  {
    cout << x << " ";
    int x{2};
    cout << x << " ";
  }
  cout << x << " ";
}
cout << x << " ";
```

```
$ ./a.out
1 2 1 0
```

Functions

Scope & Blocks

```
int x{0}; // global

int main()
{
  int y{1};
  {
    int z{2};
    cout << x << ' ' << y ' ' << z << endl;
  }
}
```

Functions

(Tedious) Example

```
#include <iostream>
using namespace std;
int main()
{
  string name1;
  string name2;
  cout << "Person 1, your name: ";
  cin >> name1;
  cout << "Person 2, your name: ";
  cin >> name2;
}
```

Functions

What are functions?

```
return_type function_name(parameters)
{
  // statements
  return result;
}
```

Functions

Back to our example

```
string read_name(int i)
{
  string result{};
  cout << "Person " << i
       << ", your name: ";
  cin >> result;
  return result;
}
```

```
int main()
{
  string name1;
  string name2;
  name1 = read_name(1);
  name2 = read_name(2);
  return 0;
}
```

Functions

Procedure

```
void foo()
{
  cout << "a procedure" << endl;
}
```

Functions

Declaration and definition

```
void function(); // declaration

// ...

void function()
{
  // ...
}
```

Functions

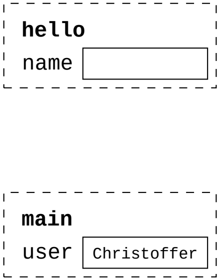
Declaration and definition

```
void hello(); // declaration

int main()
{
  hello();
}

void hello() // definition
{
  cout << "hello" << endl;
}
```

Functions

Parameter passing

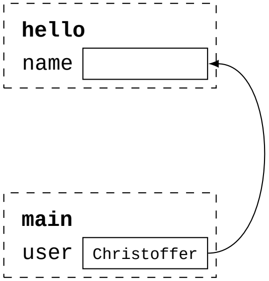
```
void hello(string name)
{
  cout << "hello "
       << name << endl;
}

int main()
{
  string user{"Christoffer"};
  hello(user);
}
```



Functions

Parameter passing

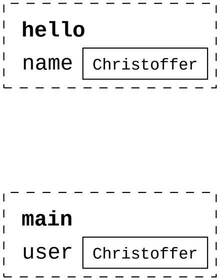
```
void hello(string name)
{
  cout << "hello "
       << name << endl;
}

int main()
{
  string user{"Christoffer"};
  hello(user);
}
```



Functions

Parameter passing

```
void hello(string name)
{
  cout << "hello "
       << name << endl;
}

int main()
{
  string user{"Christoffer"};
  hello(user);
}
```



LINKÖPING UNIVERSITY

More on variables

Data types

- built-in types
- Object types
- Pointers

More on variables

Data types

- built-in types
  - `int`
  - `double`
  - `bool`
  - etc.
- Object types
- Pointers

More on variables

Data types

- built-in types
- Object types
  - `string`
  - `struct` (today!)
  - `class` (later)
- Pointers

More on variables

Data types

- built-in types
- Object types
- Pointers
    - Comes later on!

More on variables

Compound data type

```
string name{};
int age{};
cout << "Enter your name and age: ";
cin >> name >> age;
cout << "Your name is " << name
     << " and you are " << age
     << " years old!" << endl;
```

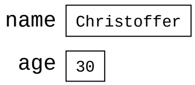
More on variables

Compound data type

```
string name1{};
string name2{};
int age1{};
int age2{};
cout << "Person 1, enter your name and age: ";
cin >> name1 >> age1;
cout << "Person 2, enter your name and age: ";
cin >> name2 >> age2;
```

More on variables

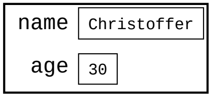
Compound data type



```
string name{};
int age{};

name = "Christoffer";
age  = 30;
```

More on variables

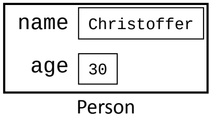
Compound data type



Person

More on variables

Compound data type



```
struct Person
{
  string name{};
  int age{};
};

Person p;

p.name = "Christoffer";
p.age  = 30;
```

More on variables

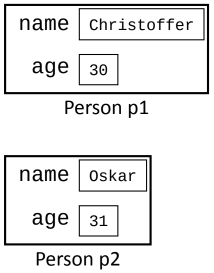
Compound data type

```
Person p1 {"Christoffer", 30};
Person p2 {"Oskar", 31};
```

More on variables

Compound data type

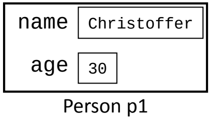
```
Person p1 {"Christoffer", 30};
Person p2 {"Oskar", 31};
```

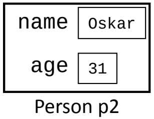


More on variables

Compound data type

```
Person p1 {"Christoffer", 30};
Person p2 {"Oskar", 31};

p1.age++;
```

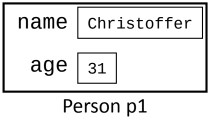


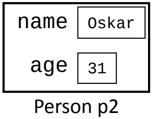


More on variables

Compound data type

```
Person p1 {"Christoffer", 30};
Person p2 {"Oskar", 31};

p1.age++;
```





More on variables

Copy

```
Person teacher{"Christoffer", 30};
Person copied_teacher{teacher};

copied_teacher.age++;

cout << teacher.age << endl;
```

More on variables

References

```
string word{"hello"};
string& greeting{word};

greeting = "hi";

cout << word << endl;
```

More on variables

References

```
string word{"hello"};
string& greeting{word};

greeting = "hi";

cout << word << endl;
```

What will be printed?

More on variables

Constant references

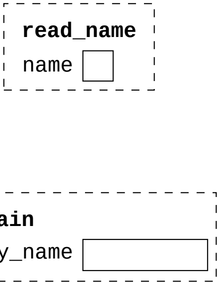
```
string word{"hello"};
string const& greeting{word};

word = "hi"; // works
greeting = "hello"; // Compilation error
```

LINKÖPING UNIVERSITY

More on functions

Parameter Passing

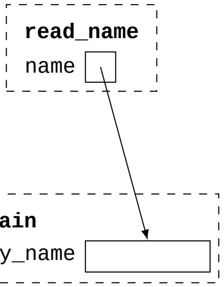
```
void read_name(string& name)
{
  cout << "Your name: ";
  cin >> name;
}

int main()
{
  string my_name;
  read_name(my_name);
  cout << my_name << endl;
}
```



More on functions

Parameter Passing

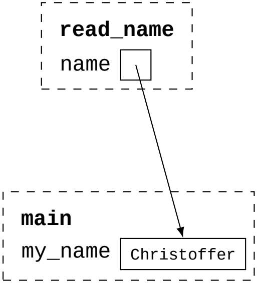
```
void read_name(string& name)
{
  cout << "Your name: ";
  cin >> name;
}

int main()
{
  string my_name;
  read_name(my_name);
  cout << my_name << endl;
}
```



More on functions

Parameter Passing

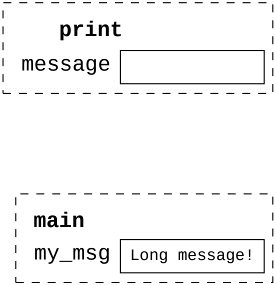
```
void read_name(string& name)
{
  cout << "Your name: ";
  cin >> name;
}

int main()
{
  string my_name;
  read_name(my_name);
  cout << my_name << endl;
}
```



More on functions

Constant Reference

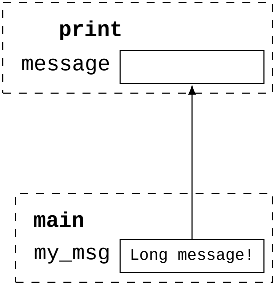
```
void print(string message)
{
  cout << message << endl;
}

int main()
{
  string my_msg{"Long message!"};
  print(my_msg);
}
```



More on functions

Constant Reference

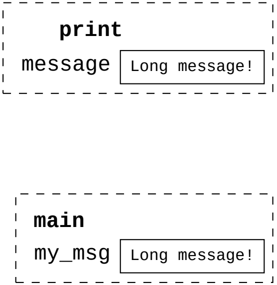
```
void print(string message)
{
  cout << message << endl;
}

int main()
{
  string my_msg{"Long message!"};
  print(my_msg);
}
```



More on functions

Constant Reference

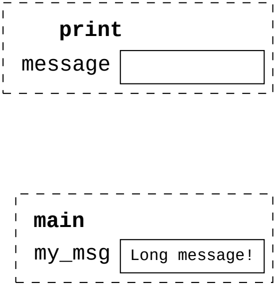
```
void print(string message)
{
  cout << message << endl;
}

int main()
{
  string my_msg{"Long message!"};
  print(my_msg);
}
```



More on functions

Constant Reference

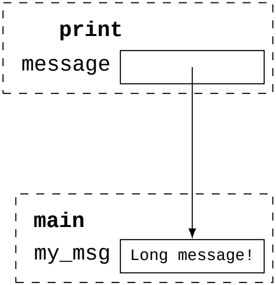
```
void print(string const& message)
{
  cout << message << endl;
}

int main()
{
  string my_msg{"Long message!"};
  print(my_msg);
}
```



More on functions

Constant Reference

```
void print(string const& message)
{
  cout << message << endl;
}

int main()
{
  string my_msg{"Long message!"};
  print(my_msg);
}
```



More on functions

Function overloading

```
// version 1
int add(int a, int b)
{
  return a + b;
}

// version 2
double add(double a,
           double b)
{
  return a + b;
}
```

```
int main()
{
  // will call version 1
  add(1, 2);

  // will call version 2
  add(3.4, 5.6);
}
```

More on functions

Which version?

```
double triangle_area(int base , double height);                 // a
double triangle_area(int side1, int side2    , int side3);      // b
double triangle_area(int side1, int side2    , double angle);   // c
double triangle_area(int side , double angle1, double angle2); // d
```

```
triangle_area(1, 1, 1);
triangle_area(1, 1);
triangle_area(1, 1.0, 1.0);
triangle_area(1, 1, 1.0);
```

More on functions

Which version?

```
double triangle_area(int base , double height);               // a
double triangle_area(int side1, int side2    , int side3);     // b
double triangle_area(int side1, int side2    , double angle);  // c
double triangle_area(int side , double angle1, double angle2); // d
```

```
triangle_area(1, 1, 1);     // b
triangle_area(1, 1);        // a
triangle_area(1, 1.0, 1.0); // d
triangle_area(1, 1, 1.0);   // c
```

More on functions

Default-parameters

```
void ignore(int n, char stop)
{
  cin.ignore(n, stop);
}
```

```
ignore(100, ':');
```

More on functions

Default-parameters

```
void ignore(int n)
{
  ignore(n, '\n');
}
```

```
ignore(100, ':');
ignore(100);
```

More on functions

Default-parameters

```
void ignore()
{
  ignore(1024);
}
```

```
ignore(100, ':');
ignore(100);
ignore();
```

More on functions

Default-parameters

```
void ignore(int n = 1024, char stop = '\n')
{
  cin.ignore(n, stop);
}
```

```
ignore(100, ':');
ignore(100);
ignore();
```

More on functions

Default-parameters

```
void ignore(int n = 1024, char stop = '\n');

int main()
{
  ignore(100, ':');
  ignore(100);
  ignore();
}

void ignore(int n, char stop)
{
  cin.ignore(n, stop);
}
```

LINKÖPING
UNIVERSITY

Operator Overloading

Example

```
struct Person
{
  string first_name;
  string last_name;
};
```

Operator Overloading

Example

```
int main()
{
  Person p1{"Christoffer", "Holm"};
  Person p2{"Klas", "Arvidsson"};

  if (p1.first_name < p2.first_name)
  {
    cout << p1.first_name << " "
         << p1.last_name << endl;
  }
}
```

Operator Overloading

Easier way

```
int main()
{
  Person p1{"Christoffer", "Holm"};
  Person p2{"Klas", "Arvidsson"};

  if (p1 < p2)
  {
    cout << p1 << endl;
  }
}
```

Operator Overloading

To make it work

```
bool operator<(Person const& p1, Person const& p2)
{
  return p1.first_name < p2.first_name;
}
```

Operator Overloading

How does it work?

```
if (p1 < p2)
{
  // ...
}
```

Operator Overloading

How does it work?

```
if (p1 < p2)
{
  // ...
}
```

```
if (operator<(p1, p2))
{
  // ...
}
```

Operator Overloading

Binary operator

```
My_Type a;
My_Type b;
a+b;
a<b;
a==b;
```

Operator Overloading

Binary operator

```
My_Type a;
My_Type b;
a+b;
a<b;
a==b;
```

```
My_Type a;
My_Type b;
operator +(a, b);
operator <(a, b);
operator==(a, b);
```

Operator Overloading

Unary operator

```
My_Type a;
-a;
++a;
a++;
```

Operator Overloading

Unary operator

```
My_Type a;
-a;
++a;
a++;
```

```
My_Type a;
operator-(a);
operator++(a);
operator++(a);
```

Operator Overloading

Unary operator

```
My_Type a;
-a;
++a;
a++;
```

```
My_Type a;
operator-(a);
operator++(a);
operator++(a, 0);
```

Operator Overloading

Unary operator example

```
struct My_Int
{
  int data;
};

My_Int& operator++(My_Int& i);
My_Int  operator++(My_Int& i, int);
```

Operator Overloading

Unary operator example

```
My_Int& operator++(My_Int& i)
{
  ++i.data;
  return i;
}
```

Operator Overloading

Unary operator example

```
My_Int operator++(My_Int& i, int)
{
  My_Int tmp{i};
  ++i;
  return tmp;
}
```

Operator Overloading

Overloading printing operators

```
Person p1{"Christoffer Holm"};
cout << p1 << endl;
```

Operator Overloading

Overloading printing operators

```
Person p1{"Christoffer Holm"};
((cout << p1) << endl);
```

Operator Overloading

Overloading printing operators

```
Person p1{"Christoffer Holm"};
(operator<<(cout, p1)) << endl);
```

Operator Overloading

Overloading printing operators

```
Person p1{"Christoffer Holm"};
operator<<(operator<<(cout, p1), endl);
```

Operator Overloading

Overloading printing operators

```
Person p1{"Christoffer Holm"};
operator<<(operator<<(cout, p1), endl);
```

What should our `operator<<` return to make it work?

Operator Overloading

Overloading printing operators

```
ostream& operator<<(ostream& os, Person const& p)
{
  os << p.first_name << " " << p.last_name;
  return os;
}
```

Operator Overloading

This is called chaining

```
Person p1{"Christoffer Holm"};
cout << p1 << endl;
```

Operator Overloading

This is called chaining

```
Person p1{"Christoffer Holm"};
((cout << p1) << endl);
```

Operator Overloading

This is called chaining

```
Person p1{"Christoffer Holm"};
(operator<<(cout, p1)) << endl);
```

Operator Overloading

This is called chaining

```
Person p1{"Christoffer Holm"};
cout << endl;
```

Operator Overloading

Overloading reading operator

```
Person p;
int x;
cin >> p >> x;
```

Operator Overloading

Overloading reading operator

```
Person p;
int x;
((cin >> p) >> x);
```

Operator Overloading

Overloading reading operator

```
Person p;
int x;
((operator>>(cin, p)) >> x);
```

Operator Overloading

Overloading reading operator

```
Person p;
int x;
operator>>((operator>>(cin, p), x);
```

Operator Overloading

Overloading reading operator

```
istream& operator>>(istream& is, Person& p)
{
  is >> p.first_name >> p.last_name;
  return is;
}
```

LINKÖPING UNIVERSITY

Stream flags

What happens?

```
int x;
string word;
cout << "Enter int: ";
cin >> x;
cout << x << endl;
cout << "Enter word: ";
cin >> word;
cout << word << endl;
```

Stream flags

What happens?

```
int x;
string word;
cout << "Enter int: ";
cin >> x;
cout << x << endl;
cout << "Enter word: ";
cin >> word;
cout << word << endl;
```

```
Enter int: 5
5
Enter word: hello
hello
```

Stream flags

What happens?

```
int x;
string word;
cout << "Enter int: ";
cin >> x;
cout << x << endl;
cout << "Enter word: ";
cin >> word;
cout << word << endl;
```

```
Enter int: a
0
Enter word:
```

Stream flags

What flags are there?

`fail`	Stream operation failed
`eof`	device has reached the end
`bad`	irrecoverable stream error
`good`	no errors

Stream flags

So how do we fix it?

```
int x;
string word;
cin >> x;
cin.clear();
cin >> word;
```

Stream flags

Checking for specific flag

```
if (cin.fail())
{
  // the fail flag
}
if (cin.eof())
{
  // the eof flag
}
if (cin.bad())
{
  // the bad flag
}
```

Stream flags

Setting the flags

```
cin.setstate(ios_base::failbit);
cin.setstate(ios_base::eofbit);
cin.setstate(ios_base::badbit);
cin.setstate(ios_base::goodbit);
```

LINKÖPING UNIVERSITY

File separation

Types of files

- Implementation files (.cc)
- Executable files

File separation

Types of files

- Implementation files (.cc)
- Executable files
- Header files (.h)

File separation

Types of files

- Implementation files (.cc)
- Executable files
- Header files (.h)
- Object file (.o)

File separation

Example

test.h

```
#ifndef TEST_H
#define TEST_H
void test(int x = 0); // declaration
#endif//TEST_H
```

test.cc

```
#include "test.h"
#include <iostream>
using namespace std;
void test(int x) // definition
{
  cout << x << endl;
}
```

main.cc

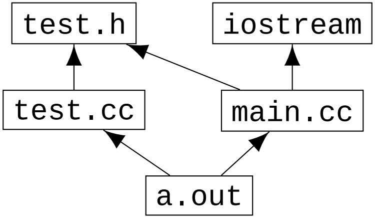
```
#include "test.h"

int main()
{
  test();
  test(1);
}
```

terminal

```
$ g++ test.cc main.cc
$ ./a.out
0
1
```

File separation

Dependency graph


$$\begin{tikzcd}
\text{test.h} & \text{iostream} \\
\text{test.cc} \arrow[u] & \text{main.cc} \arrow[u] \arrow[ul] \\
\text{a.out} \arrow[u] \arrow[ur] &
\end{tikzcd}$$

LINKÖPING
UNIVERSITY

Testing

Testing modules

```
#include "Person.h"
#include <iostream>
using namespace std;
int main()
{
  Person p1{"a", "a"};
  Person p2{"b", "b"};
  Person p3{"a", "a"};
  if (p1 < p2)
  {
    cout << "operator< works!" << endl;
  }

  if (p1 == p3 && p1 != p2)
  {
    cout << "operator== works!" << endl;
  }
}
```

Testing

Testing stream operations

```
#include "Person.h"
#include <iostream>

using namespace std;
int main()
{
  Person ans{"Christoffer", "Holm"};
  Person p;
  cout << "Enter 'Christoffer Holm': ";
  cin >> p;
  if (p == ans)
  {
    cout << "operator>> works!" << endl;
  }
}
```

Testing

Testing stream operations

```
#include "Person.h"
#include <iostream>
#include <sstream>
using namespace std;
int main()
{
  Person ans{"Christoffer", "Holm"};
  Person p;
  istringstream iss{"Christoffer Holm"};
  iss >> p;
  if (p == ans)
  {
    cout << "operator>> works!" << endl;
  }
}
```

Testing

Testing stream operations

```cpp
#include <iostream>
#include <sstream>
using namespace std;
int main()
{
  Person p{"Christoffer", "Holm"};
  ostringstream oss{};
  oss << p;
  if (oss.str() == "Christoffer Holm")
  {
    cout << "operator<< works!" << endl;
  }
}
```

Testing

`cath.hpp`

```
#define CATCH_CONFIG_MAIN
#include "catch.hpp"

TEST_CASE("testing < and ==")
{
  Person p1{"a", "a"};
  Person p2{"b", "b"};
  Person p3{"a", "b"};
  CHECK(p1 == p3);
  CHECK_FALSE(p1 == p2);
  CHECK(p1 < p2);
  CHECK_FALSE(p2 < p1);
}
```

Testing

`cath.hpp`

```
#define CATCH_CONFIG_MAIN
#include "catch.hpp"

TEST_CASE("testing < and ==")
{
  Person p1{"a", "a"};
  Person p2{"b", "b"};
  Person p3{"a", "b"};
  REQUIRE(p1 == p3);
  REQUIRE_FALSE(p1 == p2);
  REQUIRE(p1 < p2);
  REQURE_FALSE(p2 < p1);
}
```

LINKÖPING
UNIVERSITY

Time lab

Labs

- Lab1 Deadline: September 6th
- Lab2 Deadline: September 20th
- Complementary work

Time lab

Teaching session

- First teaching session: September 8th at 08:15-10:00
- There will be one session in English (always given in the highest numbered room in TimeEdit)
- Content will be about lab 2