TDDE18 & 726G77

Course Introduction

Christoffer Holm

Department of Computer and information science

LINKÖPING UNIVERSITY

LINKÖPING UNIVERSITY

Course Information

Personnel

- Examiner: Klas Arvidsson
- Course leader: Christoffer Holm
- Course assistant: Mladen Nikic
- Assistant: August Karlsson
- Assistant: David Ångström
- Assistant: Elin Frankell
- Assistant: Emil Erkgärds
- Assistant: Jesper Jonsson

Course Information

Aim (syllabus)

- Prerequisites: Skills in one programming language
- C++
- Usage of standard Linux/UNIX systems
- Problem solving

Course Information

Content

- Basic constructs
- Pointers and memory
- Object-oriented programming
- Inheritance and polymorphism
- Standard library
- Templates

Course Information

Examination

- Labs
- Exam

Course Information

Examination

- Labs
  - 6 lab assignments
  - Soft deadlines (1 per lab)
  - Demonstrate your work to the assistant
  - Complementary work
  - Bonus for exam
- Exam

Course Information

Examination

- Labs
  - Work in pairs
  - Two time slots, group A or B
  - **Register on WebReg before first lab!**
  - the two groups must be of equal size due to resources, so try to register to the group with fewer students
- Exam

Course Information

Examination

- Labs
- Exam
  - Computer exam
  - 5 assignments
  - Grading
  - Complementary work

Course Information

Organization

- Lectures
- Lab sessions
- Teaching session

Course Information

Online resources

- http://ida.liu.se/~TDDE18
- http://cppreference.com
- The library part of cppreference will be available during the exam!

Course Information

Register to the lab

Register to the labs on WebReg:

https://www.ida.liu.se/webreg-beta/TDDE18-2020-1/LAB1

LINKÖPING UNIVERSITY

C++ basics

What is C++?

- Programming language
- Is based on C
- Defined by a committee

C++ basics

What is C++?

- Gives programmer control

C++ basics

What is C++?

- Gives programmer control
- Broad application area

C++ basics

What is C++?

- Gives programmer control
- Broad application area
- Highly optimized

C++ basics

What is C++?

- C++ is **not** a specific set of programs
- C++ is **not** an editor
- C++ is **not** a compiler
- It is simply a language that can be passed to a compiler

C++ basics

A first program

program.cc

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

int main()
{
  cout << "A C++ program" << endl;
  return 0;
}
```

C++ basics

A first program

- `main` is the start point
- When the program starts, every line of code in `main` will be executed in order
- We signal the end of a line with `;`
- `cout` prints text to the console
- `return 0` tells the program to exit
- `#include <iostream>` and `using namespace std` makes `cout` available

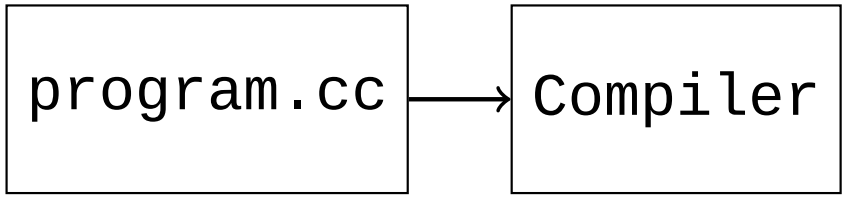
C++ basics

Compiling

```
program.cc
```

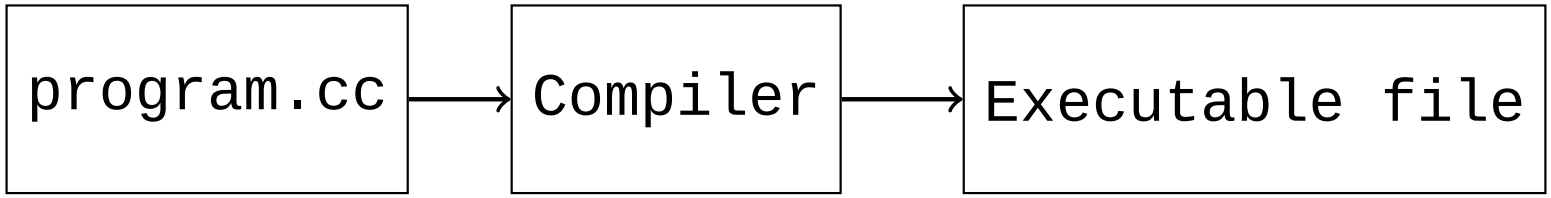
C++ basics

Compiling



C++ basics

Compiling



C++ basics

Compiling

- A compiler is a special program
- It converts *source code files* into *executable files*
- A source code file is a text file that contains your code
- An executable file is a file that contains machine code which the computer can run
- There are many compilers for C++, in this course we use one called g++

C++ basics

Compiling

```
$ ls
```

C++ basics

Compiling

```
$ ls
program.cc
```

C++ basics

Compiling

```
$ ls
program.cc
$ g++ program.cc
```

C++ basics

Compiling

```
$ ls
program.cc
$ g++ program.cc
$
```

C++ basics

Compiling

```
$ ls
program.cc
$ g++ program.cc
$
$ ls
```

C++ basics

Compiling

```
$ ls
program.cc
$ g++ program.cc
$
$ ls
a.out  program.cc
```

C++ basics

Compiling

```
$ ls
program.cc
$ g++ program.cc
$
$ ls
a.out  program.cc
$ ./a.out
```

C++ basics

Compiling

```
$ ls
program.cc
$ g++ program.cc
$
$ ls
a.out  program.cc
$ ./a.out
A C++ program
```

C++ basics

Compiling

- To compile your source file `program.cc` run the following in the console: `g++ program.cc`
- If nothing is printed then the compilation was successful
- This will produce an executable file called `a.out`
- To run your program you write: `./a.out` in the console

C++ basics

Compiler flags

```
g++ -Wall -Wextra -Wpedantic -std=c++17 program.cc
```

C++ basics

Compiler flags

- Flags can be used to enable or configure certain features in the compiler
- `-Wall -Wextra -Wpedantic` will add more warnings from the compiler which helps us write better programs
- `-std=c++17`, `-std=c++14` or `-std=c++11` allows us to pick a certain version of C++, default should be `C++17`
- **Recommended:** Create an alias

C++ basics

Creating alias

```
echo "alias w++17='g++ -std=c++17 -Wall -Wextra -Wpedantic'" >> ~/.bashrc
```

C++ basics

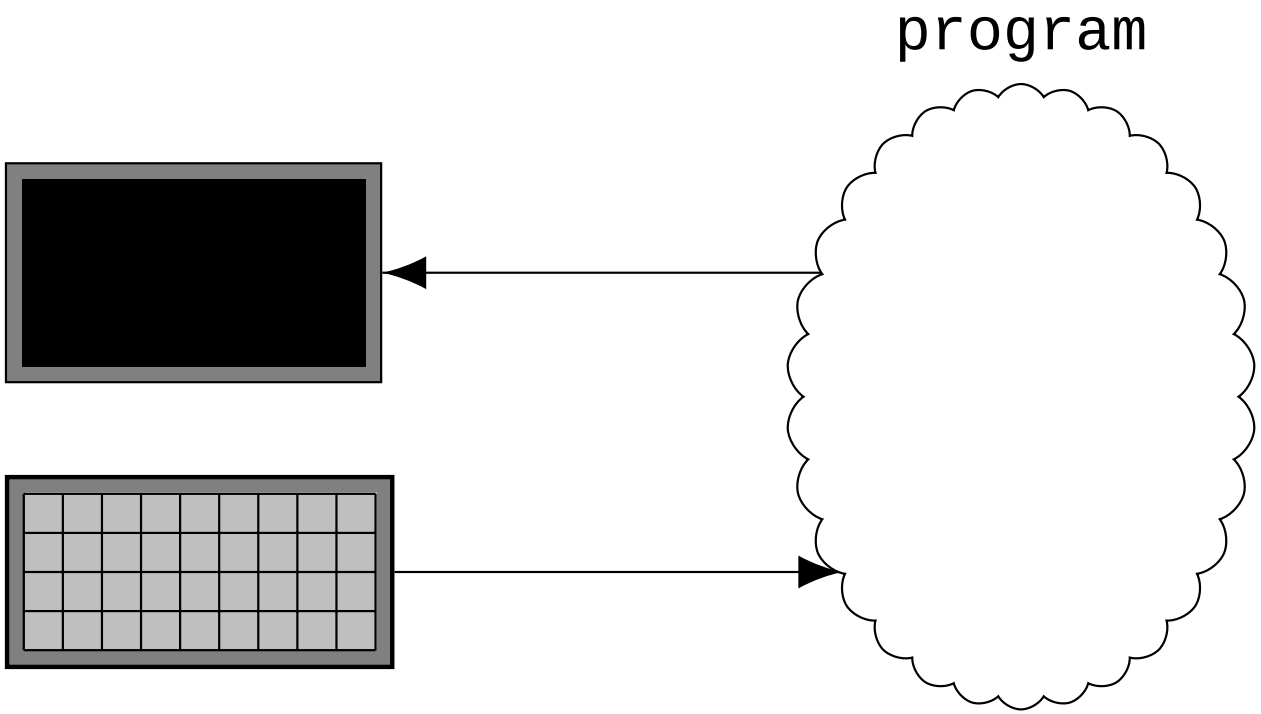
Creating alias

- This will add an *alias* to your system.
- Allows us to use `w++17` as our compiler to automatically get all the flags
- **Example:** `w++17 program.cc` will now be the same as `g++ -std=c++17 -Wall -Wextra -Wpedantic program.cc`.

LINKÖPING UNIVERSITY

IO

Idea



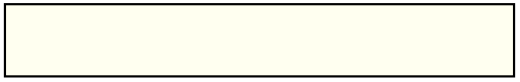
IO

Idea

- most (if not all) programs requires some kind of interaction with the user
- modern computer systems involves GUI (Graphical User Interface)
- but before GUI was a thing, everything was done through a terminal
- this is where we begin; printing to and reading input from a terminal

IO

Printing

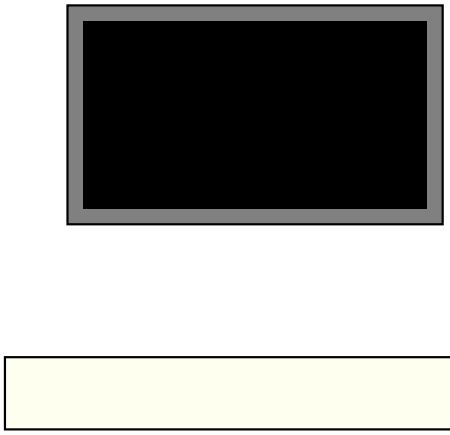


program.cc

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

int main()
{
  cout << "Text!" << endl;
  cout << "Some text!\n";
  cout << "More!" << endl;
  cout << "The " << flush;
  cout << "End!" << endl;
}
```

IO

Printing



program.cc

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

int main()
{
  cout << "Text!" << endl;
  cout << "Some text!\n";
  cout << "More!" << endl;
  cout << "The " << flush;
  cout << "End!" << endl;
}
```

IO

Printing

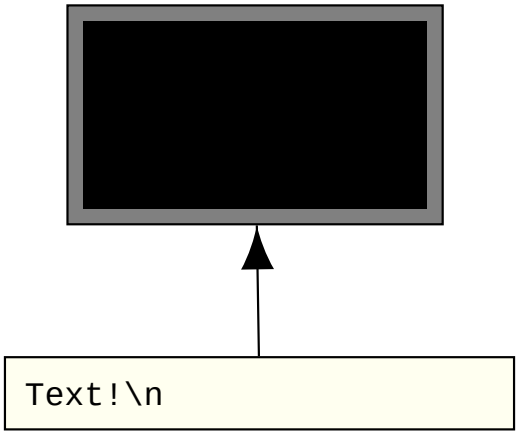
```
Text!\n
```

program.cc

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

int main()
{
  cout << "Text!" << endl;
  cout << "Some text!\n";
  cout << "More!" << endl;
  cout << "The " << flush;
  cout << "End!" << endl;
}
```

IO

Printing

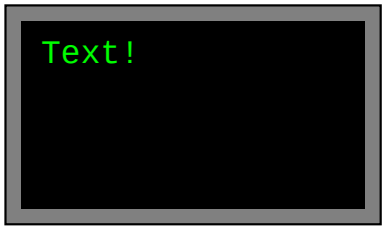


program.cc

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

int main()
{
  cout << "Text!" << endl;
  cout << "Some text!\n";
  cout << "More!" << endl;
  cout << "The " << flush;
  cout << "End!" << endl;
}
```

IO

Printing

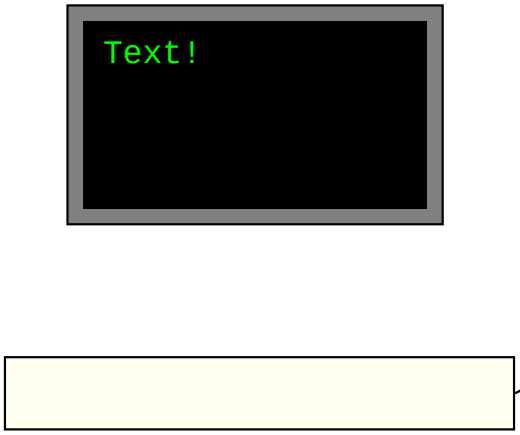


program.cc

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

int main()
{
  cout << "Text!" << endl;
  cout << "Some text!\n";
  cout << "More!" << endl;
  cout << "The " << flush;
  cout << "End!" << endl;
}
```

IO

Printing

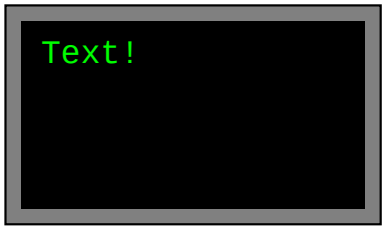


program.cc

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

int main()
{
  cout << "Text!" << endl;
  cout << "Some text!\n";
  cout << "More!" << endl;
  cout << "The " << flush;
  cout << "End!" << endl;
}
```

IO

Printing




```
Some text!\n
```

program.cc

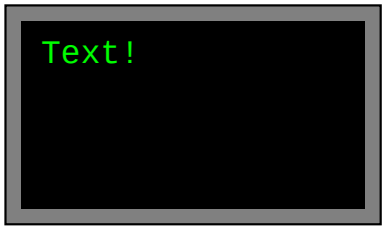
```cpp
#include <iostream>
using namespace std;

int main()
{
  cout << "Text!" << endl;
  cout << "Some text!\n";
  cout << "More!" << endl;
  cout << "The " << flush;
  cout << "End!" << endl;
}
```

IO

Printing

program.cc

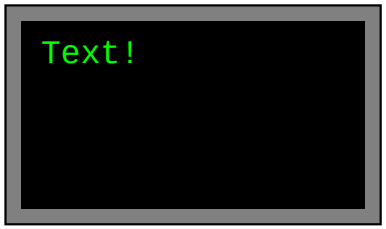
```
#include <iostream>
using namespace std;

int main()
{
  cout << "Text!" << endl;
  cout << "Some text!\n";
  cout << "More!" << endl;
  cout << "The " << flush;
  cout << "End!" << endl;
}
```




```
Some text!\n
```

IO

Printing



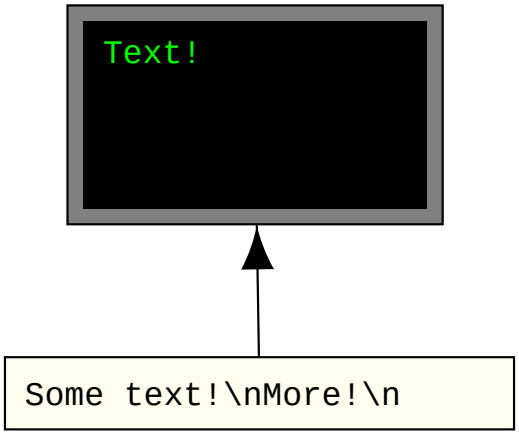

```
Some text!\nMore!\n
```

program.cc

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

int main()
{
  cout << "Text!" << endl;
  cout << "Some text!\n";
  cout << "More!" << endl;
  cout << "The " << flush;
  cout << "End!" << endl;
}
```

IO

Printing



program.cc

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

int main()
{
  cout << "Text!" << endl;
  cout << "Some text!\n";
  cout << "More!" << endl;
  cout << "The " << flush;
  cout << "End!" << endl;
}
```

IO

Printing

```
Text!
Some text!
More!
```

program.cc

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

int main()
{
  cout << "Text!" << endl;
  cout << "Some text!\n";
  cout << "More!" << endl;
  cout << "The " << flush;
  cout << "End!" << endl;
}
```

IO

Printing

program.cc

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

int main()
{
  cout << "Text!" << endl;
  cout << "Some text!\n";
  cout << "More!" << endl;
  cout << "The " << flush;
  cout << "End!" << endl;
}
```

```
Text!
Some text!
More!
```

IO

Printing

```
Text!
Some text!
More!
```

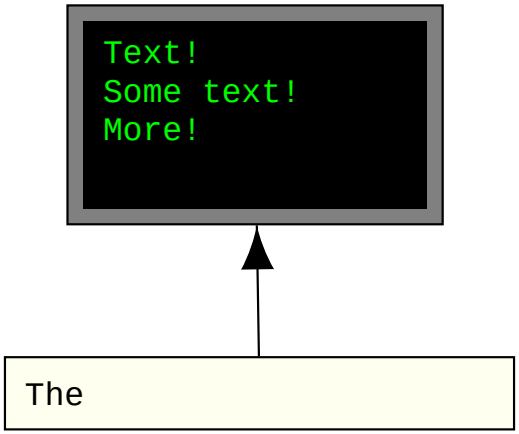
```
The
```

program.cc

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

int main()
{
  cout << "Text!" << endl;
  cout << "Some text!\n";
  cout << "More!" << endl;
  cout << "The " << flush;
  cout << "End!" << endl;
}
```

IO

Printing



program.cc

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

int main()
{
  cout << "Text!" << endl;
  cout << "Some text!\n";
  cout << "More!" << endl;
  cout << "The " << flush;
  cout << "End!" << endl;
}
```

IO

Printing

```
Text!
Some text!
More!
The
```

program.cc

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

int main()
{
  cout << "Text!" << endl;
  cout << "Some text!\n";
  cout << "More!" << endl;
  cout << "The " << flush;
  cout << "End!" << endl;
}
```

IO

Printing

program.cc

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

int main()
{
  cout << "Text!" << endl;
  cout << "Some text!\n";
  cout << "More!" << endl;
  cout << "The " << flush;
  cout << "End!" << endl;
}
```

```
Text!
Some text!
More!
The
```

IO

Printing

```
Text!
Some text!
More!
The
```

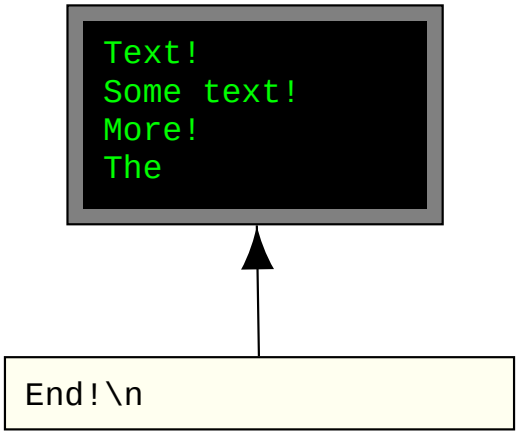
```
End!\n
```

program.cc

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

int main()
{
  cout << "Text!" << endl;
  cout << "Some text!\n";
  cout << "More!" << endl;
  cout << "The " << flush;
  cout << "End!" << endl;
}
```

IO

Printing



program.cc

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

int main()
{
  cout << "Text!" << endl;
  cout << "Some text!\n";
  cout << "More!" << endl;
  cout << "The " << flush;
  cout << "End!" << endl;
}
```

IO

Printing

```
Text!
Some text!
More!
The End!
```

program.cc

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

int main()
{
  cout << "Text!" << endl;
  cout << "Some text!\n";
  cout << "More!" << endl;
  cout << "The " << flush;
  cout << "End!" << endl;
}
```

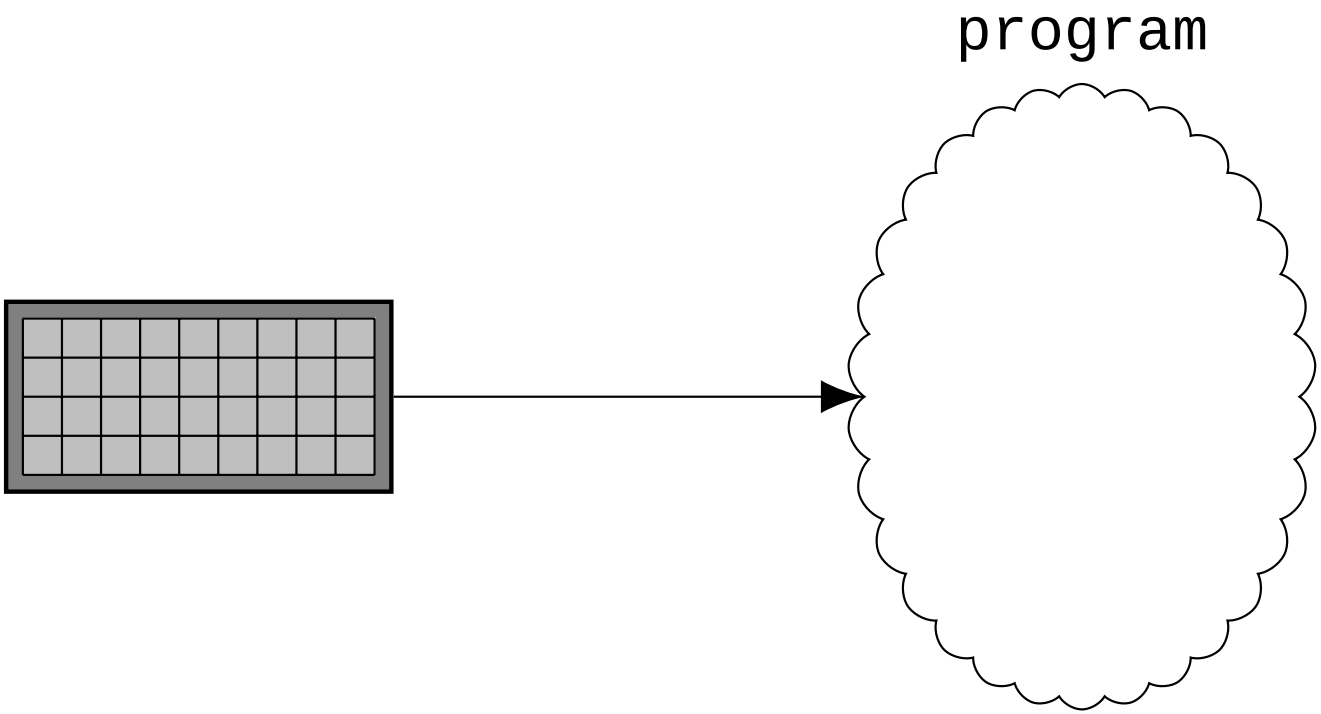
IO

Printing

- `cout` prints to a *buffer*
- the buffer is printed when it is `flushed`
- usually the buffer is flushed when a newline (`\n`) is printed (not guaranteed though)
- however to guarantee a flush we can use `endl` instead (which also inserts a newline)
- to flush without adding a newline we can use `flush`

IO

What about reading?



IO

Storing things

- When reading things from the terminal we must be able to store these things in our program
- Everything entered into a terminal is text.
- But computers work with numbers, how do we read numbers?
- We can specify how the computer should interpret the things read by specifying a so called *data type*

LINKÖPING UNIVERSITY

Variables

Basics

```
int main()
{
  int x{3};
  double y{3.14};
  char z{'c'};
}
```

Variables

Basics

```
int main()
{
  int x{3};
  double y{3.14};
  char z{'c'};
}
```

```
int main()
{
  int x{3};
  cout << "x = "
       << x << endl;
}
```

Variables

Basics

- Variables are used to store and access data
- Have different types; integers, decimal numbers, characters etc.
- Types determines what kind of values can be stored inside the variables.
- A variable can never change type
- Most variables can be printed to `cout`

Variables

string

```
#include <iostream>
#include <string>

using namespace std;
int main()
{
  string str {"hello"};
  cout << str << endl
       << str.size() << endl
       << str.front() << endl;
}
```

Variables

string

```
#include <iostream>
#include <string>

using namespace std;
int main()
{
  string str {"hello"};
  cout << str << endl
       << str.size() << endl
       << str.front() << endl;
}
```

```
$ ./a.out
hello
5
h
```

Variables

`string`

- `string` is defined in `#include <string>`
- can either be accessed with `using namespace std;` or by calling it `std::string` instead of just `string`
- Represents text (a sequence of characters).
- Has alot of builtin functionality that other types do not.

Variables

const

```
int x{5};
x = 7;

int const y{7};
y = 9; // will not compile

const int z{9};
```

Variables

`const`

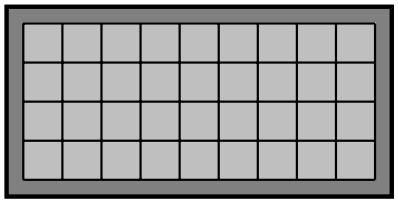
- Variables can be marked as *read-only* by adding the keyword `const`
- This means that the value of the variable can never change.
- You can place the `const` before or after the data type.
- I recommend that you place it after the type, why will become apparent in later lectures.

LINKÖPING UNIVERSITY

More IO

Reading

```
word = ""
number = 0
letter = '\0'
```



program.cc

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

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```

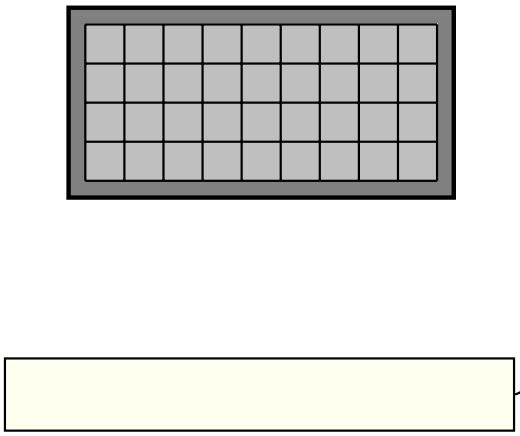
More IO

Reading

```
word = ""
number = 0
letter = '\0'
```

program.cc

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

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```



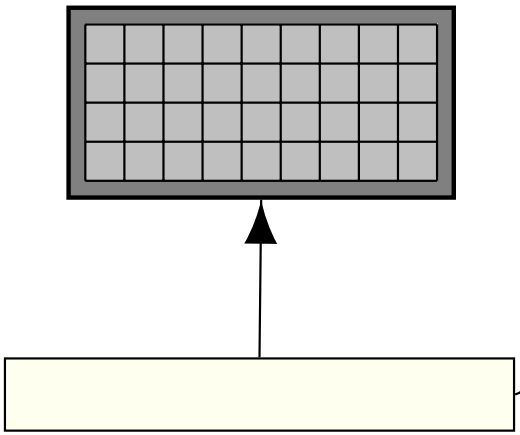
More IO

Reading

```
word = ""
number = 0
letter = '\0'
```

program.cc

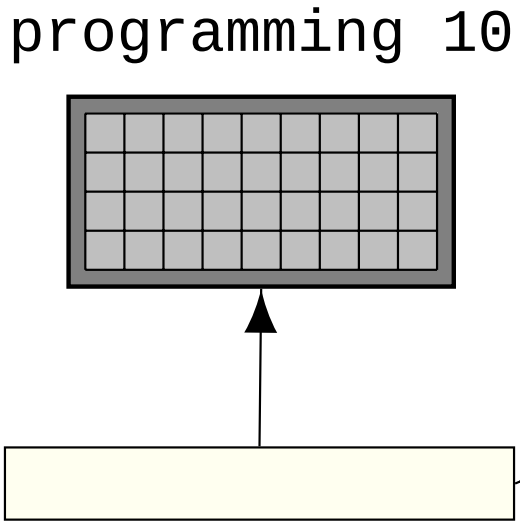
```cpp
#include <iostream>
#include <string>
using namespace std;

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```



More IO

Reading

```
word = ""
number = 0
letter = '\0'
```



program.cc

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

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```

More IO

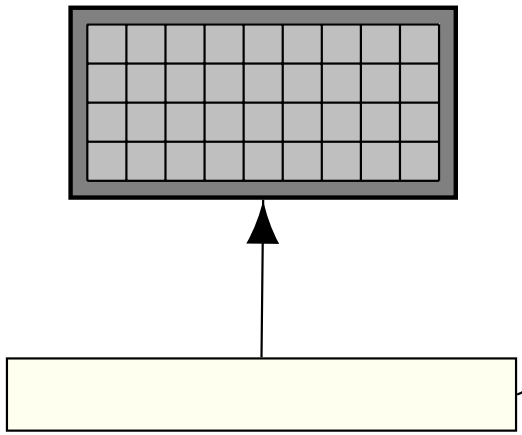
Reading

```
word = ""
number = 0
letter = '\0'
```

program.cc

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

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```

programming 10↵



More IO

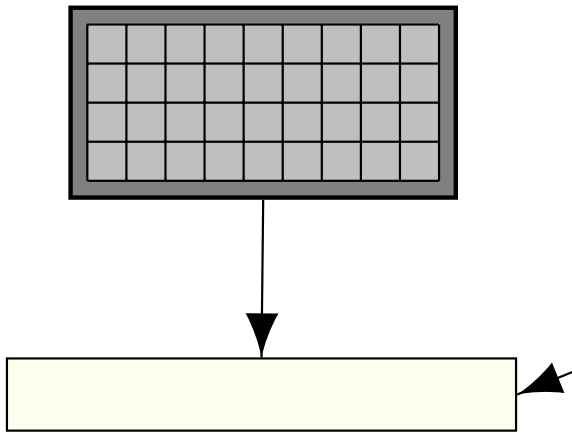
Reading

```
word = ""
number = 0
letter = '\0'
```

program.cc

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

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```

programming 10↵



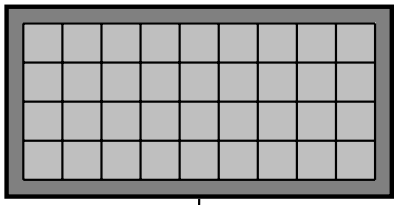
More IO

Reading

```
word   = ""
number = 0
letter = '\0'
```

program.cc

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

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```



```
Programming 10\n
```

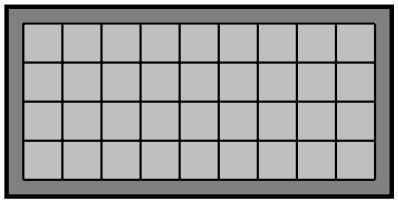
More IO

Reading

```
word = ""
number = 0
letter = '\0'
```

program.cc

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

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```



```
Programming 10\n
```

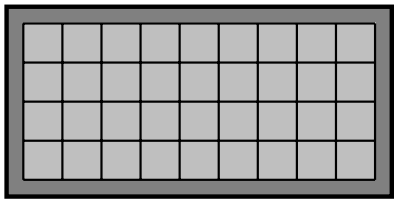
More IO

Reading

```
word = "programming"
number = 0
letter = '\0'
```

program.cc

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

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```



```
10\n
```

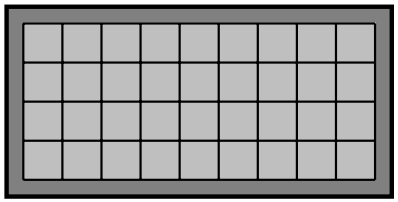
More IO

Reading

```
word = "programming"
number = 0
letter = '\0'
```

program.cc

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

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```



```
10\n
```

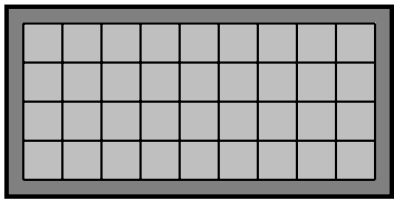
More IO

Reading

```
word = "programming"
number = 0
letter = '\0'
```

program.cc

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

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```



```
10\n
```

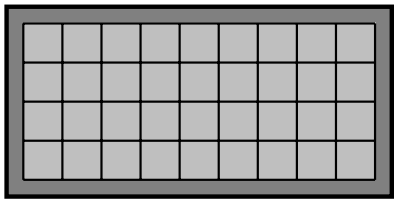
More IO

Reading

```
word = "programming"
number = 0
letter = '\0'
```

program.cc

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

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```



```
10\n
```

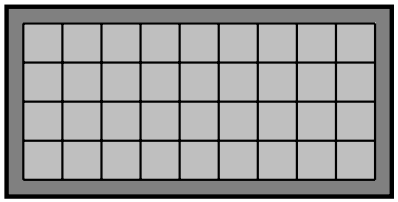
More IO

Reading

```
word = "programming"
number = 10
letter = '\0'
```

program.cc

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

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```

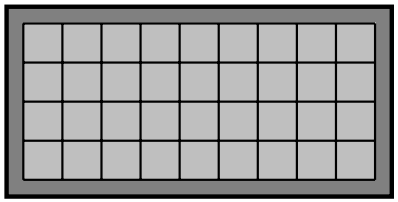


```
\n
```

More IO

Reading

```
word = "programming"
number = 10
letter = '\0'
```



```
\n
```

program.cc

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

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```

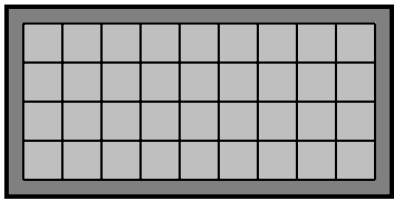
More IO

Reading

```
word   = "programming"
number = 10
letter = '\0'
```

program.cc

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

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```



```
\n
```

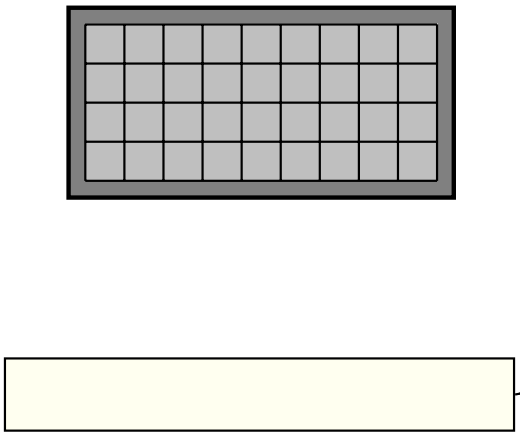
More IO

Reading

```
word   = "programming"
number = 10
letter = '\0'
```

program.cc

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

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```



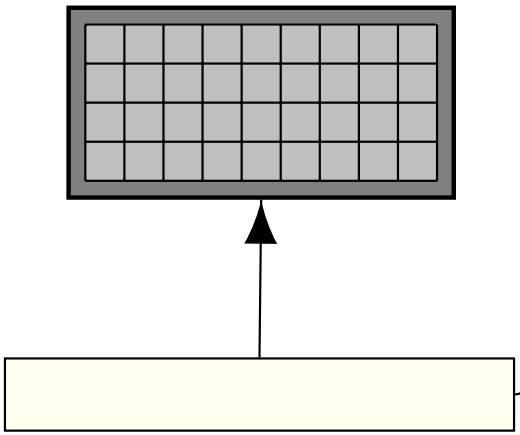
More IO

Reading

```
word = "programming"
number = 10
letter = '\0'
```

program.cc

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

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```



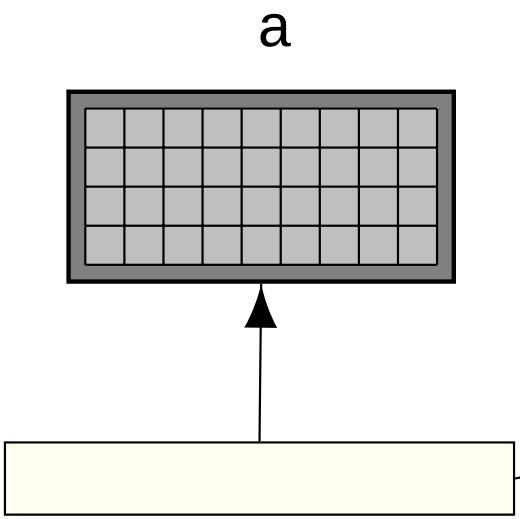
More IO

Reading

```
word = "programming"
number = 10
letter = '\0'
```

program.cc

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

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```



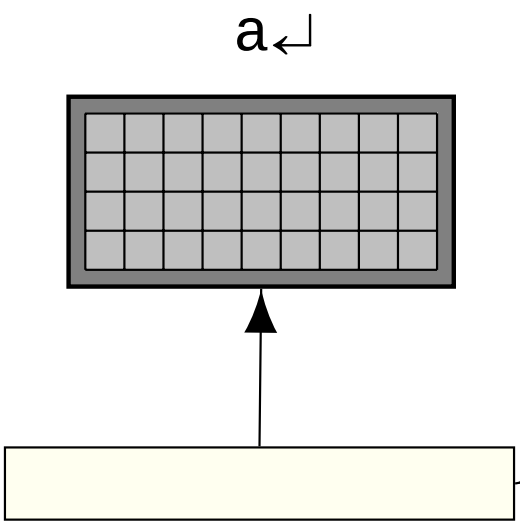
More IO

Reading

```
word = "programming"
number = 10
letter = '\0'
```

program.cc

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

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```



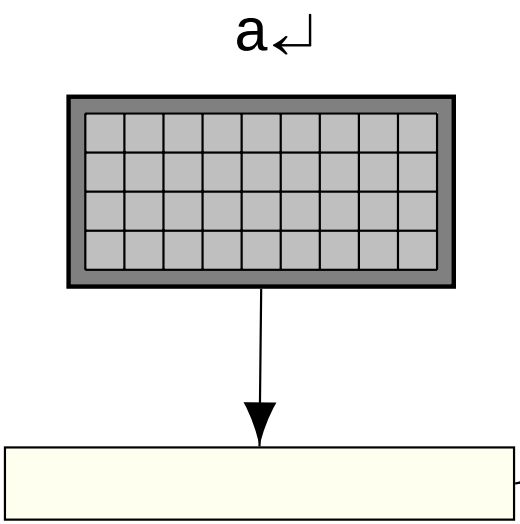
More IO

Reading

```
word   = "programming"
number = 10
letter = '\0'
```

program.cc

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

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```



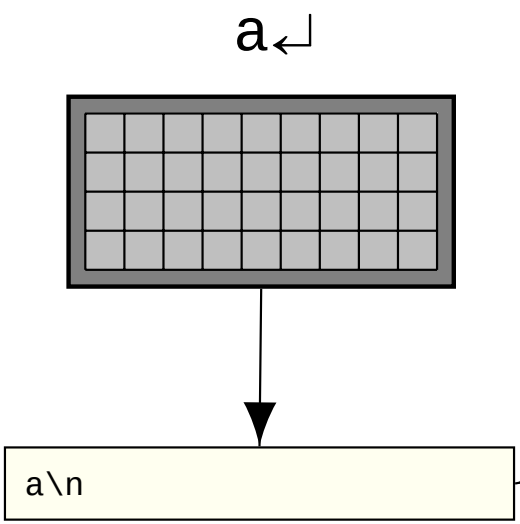
More IO

Reading

```
word = "programming"
number = 10
letter = '\0'
```

program.cc

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

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```



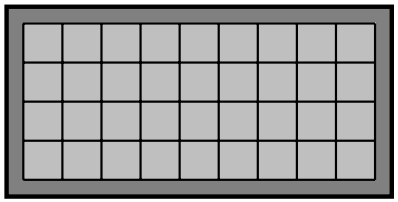
More IO

Reading

```
word   = "programming"
number = 10
letter = '\0'
```

program.cc

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

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```



```
a\n
```

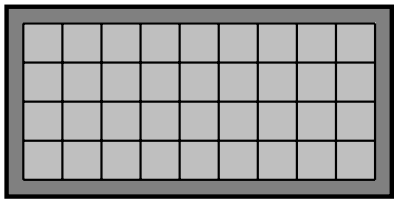
More IO

Reading

```
word = "programming"
number = 10
letter = '\0'
```

program.cc

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

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```



```
a\n
```

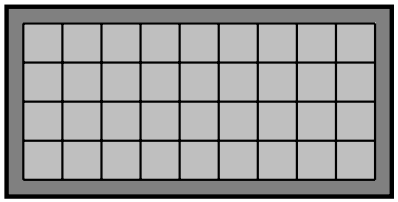
More IO

Reading

```
word = "programming"
number = 10
letter = 'a'
```

program.cc

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

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```

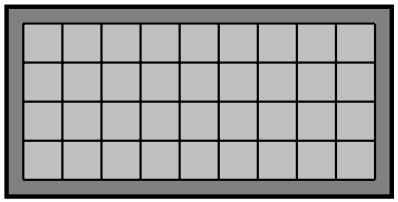


```
\n
```

More IO

Reading

```
word = "programming"
number = 10
letter = 'a'
```



```
\n
```

program.cc

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

int main()
{
  cout << "Enter a word and number: ";
  string word{};
  int number{};
  char letter{};
  cin >> word;
  cin >> number;
  cin >> letter;
}
```

More IO

Reading

- Reading values from the terminal into variables is done using `cin`.
- There is a buffer which the reading will be done from in first-hand.
- If the buffer is empty then, and only then will a prompt appear in the terminal.
- `cin` will read from this buffer until it is empty again.
- Most data types can be read from `cin`.

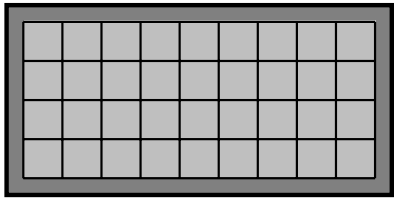
More IO

Reading

- Whitespace characters are such characters as space, newline and other characters that doesn't have a glyph.
- While reading from the buffer, `cin` will ignore all whitespaces until it reaches a non-whitespace character.
- If `cin` finds a whitespace character (or any character that is nonsensical for the data type) while reading a value, the reading is done.

More IO

getline



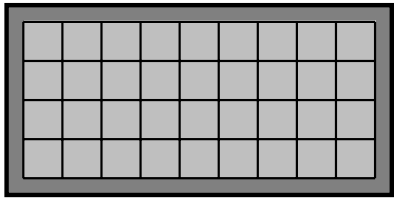
```
This is a line\nAnother line\n
```

line = ""

```
string line;
getline(cin, line);
cin.ignore(1000, '\n');
```

More IO

getline



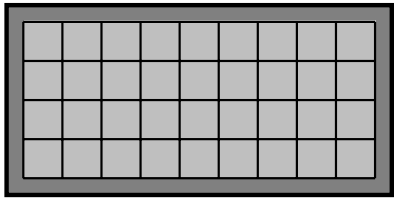
line = ""

```
string line;
getline(cin, line);
cin.ignore(1000, '\n');
```

```
This is a line\nAnother line\n
```

More IO

getline



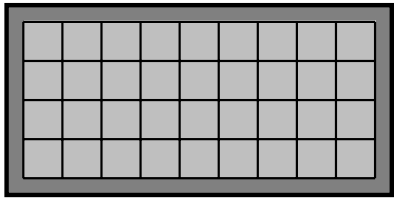
```
This is a line\nAnother line\n
```

line = ""

```
string line;
getline(cin, line);
cin.ignore(1000, '\n');
```

More IO

getline



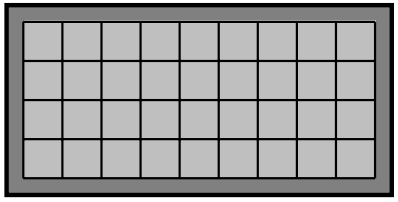
```
\nAnother line\n
```

line = "this is a line"

```
string line;
getline(cin, line);
cin.ignore(1000, '\n');
```

More IO

getline



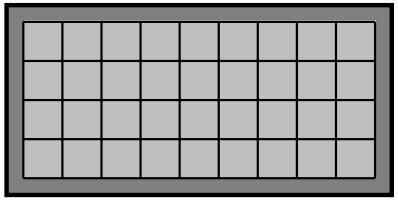
```
Another line\n
```

line = "this is a line"

```
string line;
getline(cin, line);
cin.ignore(1000, '\n');
```

More IO

getline



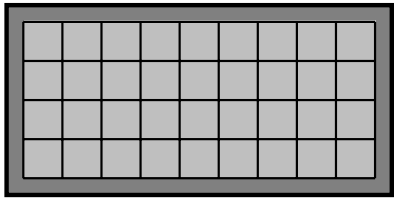
```
Another line\n
```

line = "this is a line"

```
string line;
getline(cin, line);
cin.ignore(1000, '\n');
```

More IO

getline



line = "this is a line"

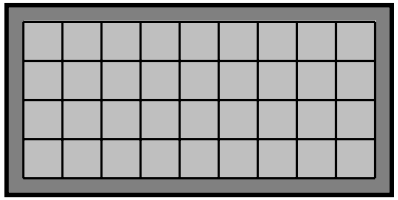
```
string line;
getline(cin, line);
cin.ignore(1000, '\n');
```

```
Another line\n
```

More IO

getline

line = "this is a line"

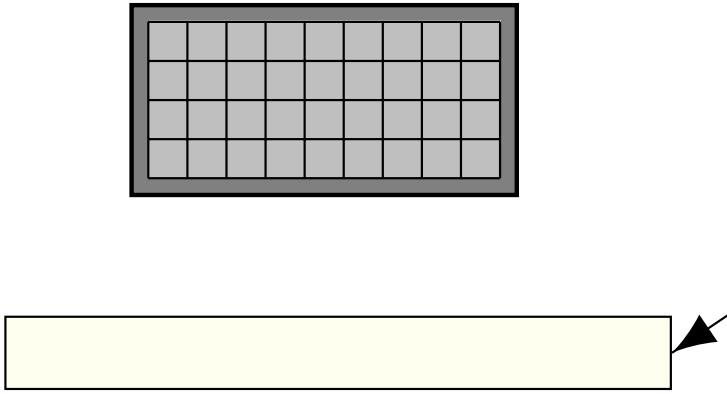


```
string line;
getline(cin, line);
cin.ignore(1000, '\n');
```

```
\n
```

More IO

getline

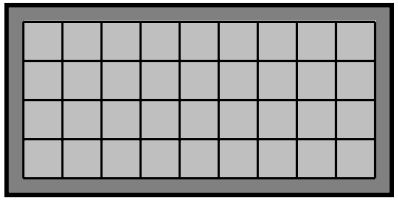


line = "this is a line"

```
string line;
getline(cin, line);
cin.ignore(1000, '\n');
```

More IO

getline



line = "this is a line"

```
string line;
getline(cin, line);
cin.ignore(1000, '\n');
```

More IO

`getline`

```
#include <iostream>
#include <string>

using namespace std;
int main()
{
  string line;
  cout << "Enter a line: ";
  getline(cin, line);
  cout << "Your line was: "
       << line << endl;
}
```

More IO

`getline`

- `getline` is how we read entire lines instead of words.
- We give it `cin` and a string we want to read into.
- It will read until it finds a newline character (`\\n`) and store it into the string.
- Then it will remove the newline from the buffer.

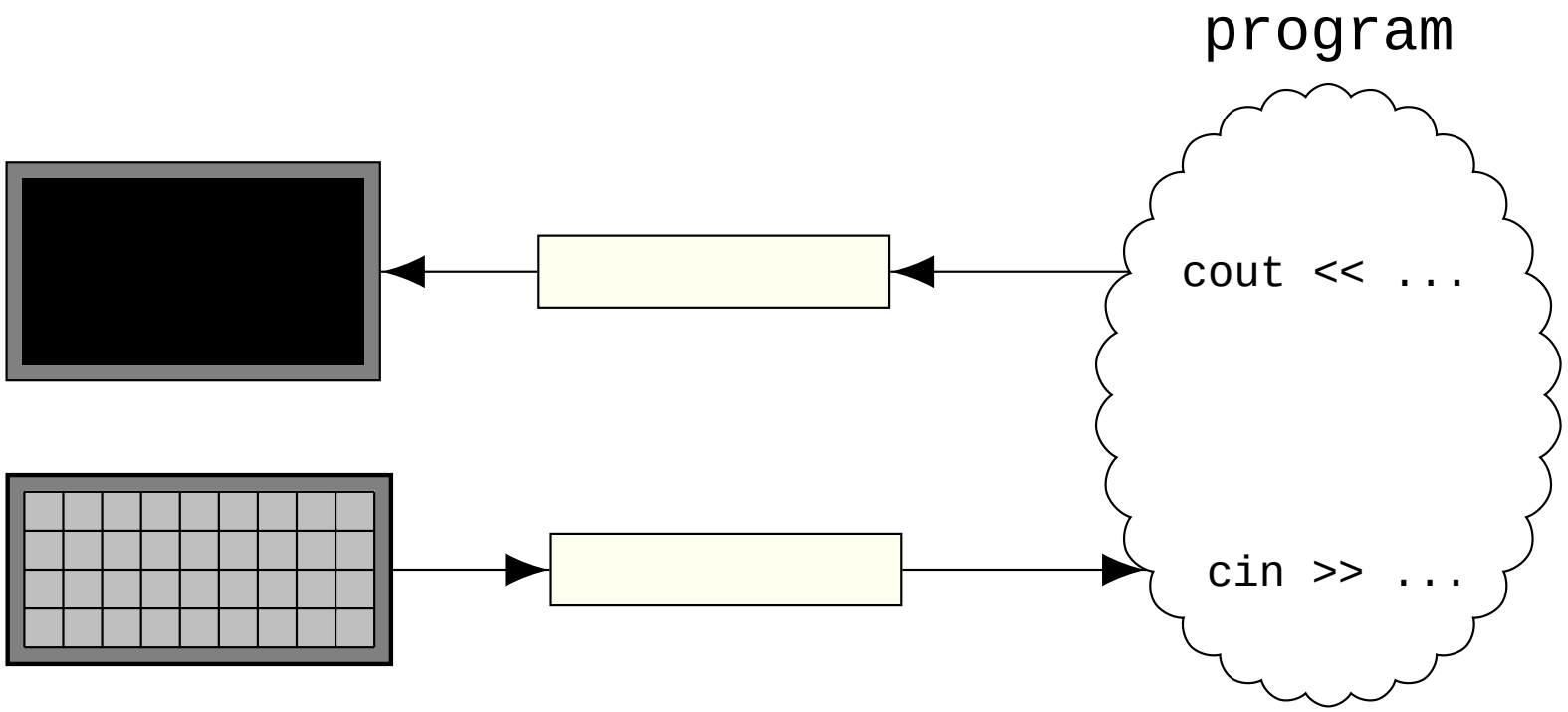
More IO

`ignore`

- `cin.ignore` will remove things from the stream
- we give `cin.ignore` two things; how many characters to ignore and what the *delimiter* is
- `cin.ignore` will ignore either the specified amount of character or until it finds the *delimiter* character, whichever occurs first!

More IO

The complete picture



LINKÖPING UNIVERSITY

Streams

Reading from files

```
#include <fstream>
#include <string>
using namespace std;

int main()
{
  ifstream in{"data.txt"};
  string line;
  int x;
  in >> x;
  getline(in, line);
  in.ignore(1000, '\n');
}
```

Streams

What are streams?

- Streams are how we communicate with external resources
- `cout` and `cin` are streams that communicate with the terminal
- But there are other external resources such as files, memory, network etc.
- C++ lets us communicate with files through `ifstream` (reading from) and `ofstream` (writing to)

Streams

What are streams?

- `ifstream` and `ofstream` are defined in
  `#include <fstream>`
- All streams work according to the same principles we learned from `cin` and `cout`
- so all operations we have talked about work exactly the same way for all streams

Streams

Formatting output streams

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
  cout << setw(10) << "hello"
       << '|' << right
       << setw(10) << "world"
       << endl;
  cout << setfill('-')
       << setw(21) << "The end!"
       << endl;
}
```

```
$ ./a.out
hello     |     world
-------------The end!
```

Streams

Formatting output streams

- For more advanced formatting we can include
  `#include <iomanip>`
- `setw(10)` will ensure that the next item printed will print *at least* 10 characters.
- If the printed item prints less than 10 characters the rest will be filled with spaces after the item until it is 10 characters.

Streams

Formatting output streams

- We can modify properties of `setw`
- `right` places the spaces before the item instead of after.
- `setfill` will replace the spaces with some other character.
- Both `right` and `setfill` are *sticky* meaning they will stay in effect until they are replaced.
- There are a lot more features in `#include <iomanip>`. Look at cppreference: https://en.cppreference.com/w/cpp/io/manip

LINKÖPING UNIVERSITY

Basic constructs

Conditional statements

```
if (some logical statement)
{
  // do this
}
else if (some other logical statement)
{
  // do this instead
}
else
{
  // when all else fails, do this
}
```

Basic constructs

bool

```
int main()
{
  bool statement{false};
  if (statement)
  {
    // will not run
  }
  else
  {
    // will run
  }
}
```

Basic constructs

bool

```
int main()
{
  bool statement{false};
  if (!statement)
  {
    // will run
  }
  else
  {
    // will not run
  }
}
```

Basic constructs

`bool`

- `bool` is a data type that represents logical results
- Can be `true` or `false`.
- Represents the results of conditional statements.
- Can be *inverted* with `!` (or the keyword `not`).

Basic constructs

Comparison and logical operators

- `a == b`
- `a != b`
- `a < b`
- `a <= b`
- `a > b`
- `a >= b`

Basic constructs

Comparison and logical operators

- `a == b`
- `a != b`
- `a < b`
- `a <= b`
- `a > b`
- `a >= b`

- `a == b and c != b`
- `a == b or a == c`

Basic constructs

Comparison and logical operators

- `a == b`
- `a != b`
- `a < b`
- `a <= b`
- `a > b`
- `a >= b`

- `a == b && c != b`
- `a == b || a == c`

Basic constructs

Loops

```
#include <iostream>
using namespace std;
int main()
{
  int x{};
  cout << "Enter number (1-10): ";
  cin >> x;
  while (x < 1 || x > 10)
  {
    cout << "Enter number (1-10): ";
    cin >> x;
  }
}
```

Basic constructs

Loops

```
#include <iostream>
using namespace std;
int main()
{
  int x{};
  do
  {
    cout << "Enter number (1-10): ";
    cin >> x;
  }
  while (x < 1 || x > 10);
}
```

Basic constructs

Loops

- Two ways to loop based on a condition
- `while`-loops will run 0 or more times
- `do`-`while`-loops will run 1 or more times

Basic constructs

`for`-loop

```
#include <iostream>
using namespace std;
int main()
{
  for (int i{0}; i < 10; ++i)
  {
    cout << "Iteration #" << i << endl;
  }
}
```

Basic constructs

`for`-loop

- `for`-loops are the third way to loop
- This is used whenever we know how many times we are going to loop

Basic constructs

Arithmetic operations

- `a + b` (addition)
- `a - b` (subtraction)
- `a * b` (multiplication)
- `a / b` (division)
- `a % b` (modulus)

Basic constructs

Arithmetic operations

- `a + b` (addition)
- `a - b` (subtraction)
- `a * b` (multiplication)
- `a / b` (division)
- `a % b` (modulus)
- `-a` (negation)
- `++a` (prefix increment)
- `a++` (postfix increment)
- `--a` (prefix decrement)
- `a--` (postfix decrement)

Basic constructs

Prefix vs. Postfix

```
int a{0};

a += 2; // a = a + 2
++a;    // a = a + 1
a++;    // a = a + 1

int b{++a};
int c{a++};

// what is a, b and c?
```

Basic constructs

Prefix vs. Postfix

- `++a` will increment `a` and then give back the new value of `a`.
- `a++` will increment `a` and then give back the old value of `a`.
- **Example:**

```
int a{0};   // a = 0
int b{++a}; // a = 1 and b = 1
int c{a++}; // a = 2 and c = 1
```

Basic constructs

Prefix vs. Postfix

Rule of thumb: the placement of ++ determines when the increment is performed (before or after we read the value)

Basic constructs

Type casting

- `3 / 2 = 1`

Basic constructs

Type casting

- `3 / 2 = 1`
- `3 / 2.0 = 1.5`

Basic constructs

Type casting

- `3 / 2 = 1`
- `3 / 2.0 = 1.5`
- `3.0 / 2 = 1.5`

Basic constructs

Type casting

- `3 / 2 = 1`
- `3 / 2.0 = 1.5`
- `3.0 / 2 = 1.5`
- `3.0 / 2.0 = 1.5`

Basic constructs

Type casting

- When performing operations on values C++ will always make sure that the result is the same result as the operands
- So all operations on integers will give us integers
- If there are two different operands it will always convert the less accurate one to the more accurate type and then perform the operation

Basic constructs

Type casting

```
int a{3};
int b{2};

cout << a / b << endl;
// will output 1

cout << static_cast<double>(a) / b << endl;
// will output 1.5
```

Basic constructs

Type casting

- `static_cast` can be used to convert an expression into another type
- Will only work if the conversion is sensical
- Should be used as little as possible
- But sometimes it is unavoidable

LINKÖPING UNIVERSITY

Files

Reading all of a file

```
ifstream ifs{"data.txt"};
string s{};
while (...)
{
   ...
}
```

Files

Reading all of a file

```
ifstream ifs{"data.txt"};
string s{};
while (ifs >> s)
{
   ...
}
```

Files

Reading all of a file

- Once the end of the file is reached `cin >> s` will return false
- This allows us to read word by word until the end of the file
- Works with `getline` as well!

Files

Reading all of a file

```
ifstream ifs{"data.txt"};
string line{};
while (getline(ifs, line))
{
   ...
}
```

Files

UNIX console

- `cd` to change directory
- `mkdir` to create a directory
- `ls` view files in current directory
- `rm` remove a file
- `rm -r` remove a directory
- `mv` to rename a file or directory
- `cp` to copy a file

Register on WebReg!

www.liu.se

LINKÖPING
UNIVERSITY