

Lecture 3: String Theory

Bart Iver van Blokland

PSA (reminder)

- We want to see whether we should adjust how many student assistants are available throughout the week
- Please let us know which times you would like to drop by to get help from our student assistants
 - Note: this is not a binding registration!



<https://nettskjema.no/a/585338>

Previously on...

Season 48

What On Earth Was That Lecturer Talking About

Do you remember from yesterday?

- What value will be stored in the following variables?

```
int pi = 3.1415927;
```

```
double half = 5.0 / 2;
```

```
double average = (9 + 2) / 2;
```

- What is a potential issue with the following line?

```
double angle;
```

- What is the difference between a signed and unsigned integer?

Today

Basics of the C++ Language

- **Text input**
- If statements
- Loops
- Switch statement
- Functions
- Strings
- Basics of graphics
- Some tips for writing code

There's a way cin..

- Just like cout, there's a cin
- Uses >> to read a value into a variable
- The variable you read into can be of any of the common data types.

```
cout << "What is your favourite food? " << endl;  
string food;  
cin >> food;  
cout << "That's awesome, " << food << " is also my favourite!" << endl;
```

There's a way cin..

- cin only reads one word or one number at a time!
For example: entering "1.5 2.3 3.8 4.4" here will print "1.5":

```
double value;  
cout << "Write some numbers: ";  
cin >> value;  
cout << "Value: " << value << endl;
```

If you use cin again in this example, "value" becomes 2.3

- You can use the getline() function to read a line:

```
string message;  
cout << "Write a message: ";  
getline(cin, message);  
cout << "The message was: " << message << endl;
```

Today

Basics of the C++ Language

- Text input
- **If statements**
- Loops
- Switch statement
- Functions
- Strings
- Basics of graphics
- Some tips for writing code

If statements

```
bool condition = true;

if(condition) {
    cout << "condition is true!" << endl;
} else {
    cout << "condition is false." << endl;
}
```

- If statements allow you to run a piece of code based on a boolean true/false condition

If statements

```
bool condition = true;
bool alternateCondition = true;

if(condition) {
    cout << "condition is true!" << endl;
} else if(alternateCondition) {
    cout << "condition is false." << endl;
    cout << "alternateCondition is true!" << endl;
} else {
    cout << "condition is false." << endl;
    cout << "alternateCondition is also false." << endl;
}
```

- Multiple statements can be chained

Today

Basics of the C++ Language

- Text input
- If statements
- **Loops**
- Switch statement
- Functions
- Strings
- Basics of graphics
- Some tips for writing code

Loops

- There are three different loop variations in C++

- The for loop:

```
for(int i = 0; i < 10; i++) {  
  
}
```

- The while loop:

```
int i = 0;  
while(i < 10) {  
    i++;  
}
```

- The do while loop:

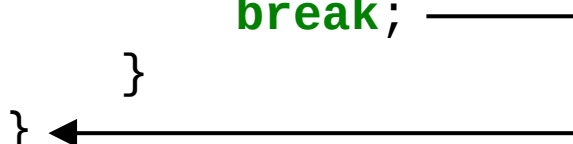
```
int i = 0;  
do {  
    i++;  
} while(i < 10);
```

- Loops are interchangeable: each can be rewritten as another type.
 - For example, all the shown loops repeat their loop body 10 times!

Loops

- There are also two execution flow statements:
 - Break: stop the innermost loop immediately

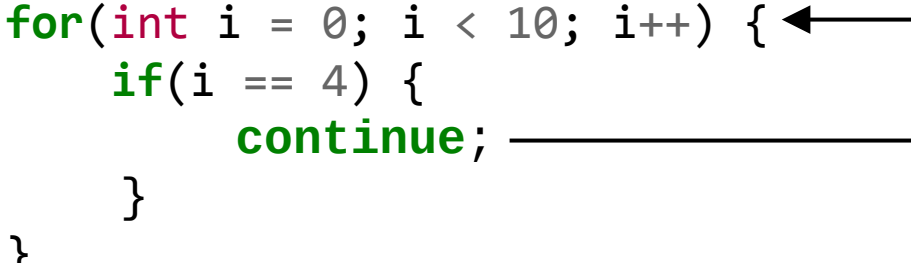
```
for(int i = 0; i < 10; i++) {  
    if(i >= 3) {  
        break;  
    }  
}
```



A diagram illustrating the effect of the `break` statement. A horizontal line extends from the `break;` statement to the right, then turns 90 degrees down, then 90 degrees left, ending with an arrow pointing to the closing curly brace of the `for` loop, indicating an immediate exit from the loop.

- Continue: stop the current iteration and move to the next one

```
for(int i = 0; i < 10; i++) {  
    if(i == 4) {  
        continue;  
    }  
}
```



A diagram illustrating the effect of the `continue` statement. A horizontal line extends from the `continue;` statement to the right, then turns 90 degrees up, then 90 degrees left, ending with an arrow pointing to the opening curly brace of the `for` loop, indicating that the current iteration is skipped and the next one begins.

Loops

- Where should you use each loop type?
 - The for loop:
When you know in advance how many iterations you need
 - The while loop:
When you don't know in advance how many iterations you need
 - The do while loop:
When you need the loop body to be executed at least once

Loops

for (**int** **i** = 0; **while** (**i** < 10) { **i**++;



In practice, the for loop is the one used most (by far)

Task: Loops

- Rewrite the following loop such that it counts from 10 to 0
 - The final line printed by the program should be “The value of i is 0”
- No, rewriting the println() call to print 10 - i does not count ;)
- Done? Rewrite it into a while loop

```
for(unsigned int i = 0; i < 10; i++) {  
    println("The value of i is {}", i);  
}
```


Today

Basics of the C++ Language

- Text input
- If statements
- Loops
- **Switch statement**
- Functions
- Strings
- Basics of graphics
- Some tips for writing code

The switch statement

- Has very specific use cases (such as a menu system...)
- Code will be executed from the case that matches the value being 'switched' on exactly

```
int number = 0;
std::cin >> number;
switch(number) {
    case 42:
        println("The ultimate answer!");
    case 1:
        println("We are number one!");
    case 1337:
        println("H4ck0rz!");
}
```

The switch statement

- Use break statements to ensure your program does not execute multiple case statements!

```
int number = 0;
std::cin >> number;
switch(number) {
    case 42:
        println("The ultimate answer!");
        break;
    case 1:
        println("We are number one!");
        break;
    case 1337:
        println("H4ck0rz!");
        break;
}
```

Today

Basics of the C++ Language

- Text input
- If statements
- Loops
- Switch statement
- **Functions**
- Strings
- Basics of graphics
- Some tips for writing code

Functions

- Functions are pieces of code that can be run on demand
- Functions allow reusing functionality across your program
 - And if you fix a bug in a function used in multiple places, you have fixed it everywhere!
- Functions allow abstraction: as long as the function behaves the same way, you can change how it works.

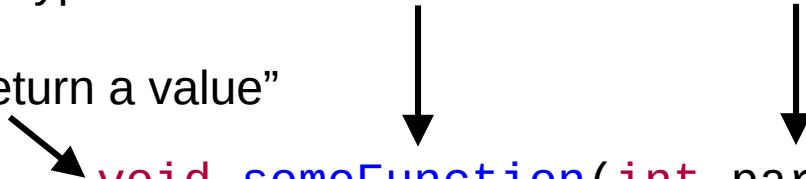
Functions

Return data type
void means
“does not return a value”

Function name

Parameter list
(can be empty)

```
void someFunction(int parameter, bool another) {  
    }  
}
```



Functions

Functions can «result in» values by returning them. Values are returned using a return statement.

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

Functions are called by writing the name, followed by any parameter values separated using commas.

```
void printAnswer() {  
    cout << add(21, 21) << endl;  
}
```

Returned values can be used directly

Functions

- Declaration versus Definition:

- Declaration: what does the function look like?
 - Does not have code surrounded by { }

```
void functionA();
```

- Definition: what does the function do?
 - Contains code surrounded by { }

```
void functionA() {  
}
```


Functions

Functions must be declared before they can be used.

Order matters: the program is analysed top to bottom

```
#include "std_lib_facilities.h"

void functionA() {
    cout << "I am function A" << endl;
}

void functionB();

int main() {
    functionA();
    functionB();
    return 0;
}

void functionB() {
    cout << "I am function B" << endl;
}
```

The call stack

- When a function is being executed, it may itself call other functions.
 - Which in turn can call other functions
 - Which in turn can call other functions
 - Which in turn can call other functions
 - Which in turn can call other functions
 - Which in turn can call other functions
- The sequence of functions that are «in progress» at any point in time is called the *call stack*.

Demonstration: the call stack

Functions: Recursion

- In order to understand recursion, one must first understand recursion
- Functions can call themselves to break down a problem into smaller parts
 - Fibonacci in assignment 1

```
void count(int countFrom) {  
    std::cout << countFrom << std::endl;  
    count(countFrom - 1);  
}
```

Functions: Recursion

- The risk: stack overflow!
 - Happens when you recurse too deeply

```
void count(int countFrom) {  
    std::cout << countFrom << std::endl;  
    count(countFrom - 1);  
}
```

91 void count(int countFrom) {

Exception has occurred: Exception ✕

EXC_BAD_ACCESS (code=2, address=0x16f603ffc)

Functions: Recursion


- Solution: remember to add an exit clause!

```
void countDown(int countFrom) {  
    if(countTo == 0) {  
        return;  
    }  
    std::cout << countFrom << std::endl;  
    count(countFrom - 1);  
}
```

Functions: Default parameters

It is possible to give function parameters a default value.

If the parameter is omitted, the default value is used.



```
int increment(int value, int incrementBy = 1) {  
    return value + incrementBy;  
}  
  
int main() {  
    cout << increment(10) << endl; ← Prints 11  
    cout << increment(10, 5) << endl; ← Prints 15  
    return 0;  
}
```

Functions: Function overloading



Functions: Function overloading

Function overloading: defining multiple functions with the same name.

The compiler determines based on the data type of the parameters which function to call.

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

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

```
int main() {  
    cout << "5 + 6 = " << add(5, 6) << endl;  
    cout << "5.0 + 6.0 = " << add(5.0, 6.0) << endl;  
    cout << "5.0 + 6.0 = " << add(5.0, 6) << endl;  
    return 0;  
}
```


Two integers,
calls the top variant



Two doubles,
calls the bottom variant



Error: not sure
which one to use



Debugging



Debugging

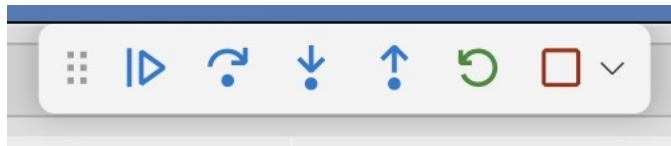
- Bug: a discrepancy between what a program *should do*, and what it *does*
 - Fixing a bug requires understanding what exactly is going on that results in the incorrect behaviour.
- Worst possible approach: debugging by permutation
 - While bug not fixed, make a change that feels like it should have a desired effect. Repeat as needed.

Debugging

- Examples of debugging techniques:
 - Print debugging
Using `cout` or `println` to write values to the terminal
 - Using a debugger
Allows running your code line by line
 - Using tests
Run functions with different parameters and check if their output matches the expected results
- It is common to use a combination of these!

Debugging

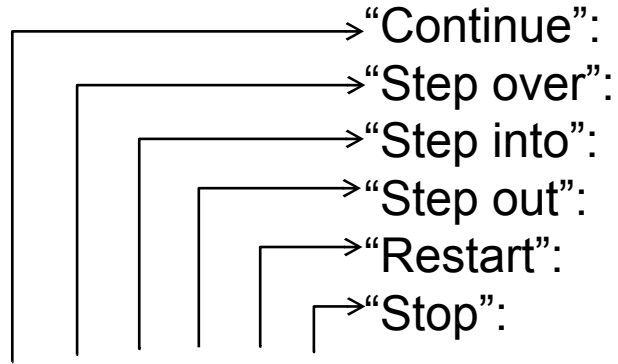
- Using a debugger
 - Use Run > Start with debugging in VS Code
- Set a breakpoint at the start of a piece of code you believe might have a problem (click in left hand margin in code editor)
- Step through the program using the controls that pop up:



```
25  int main() {  
26  
27      for(int i = 0; i <  
28          int computedBl  
29          println("Using  
31  
32      return 0;  
33  }  
34  
35  
36
```

Click to add a breakpoint

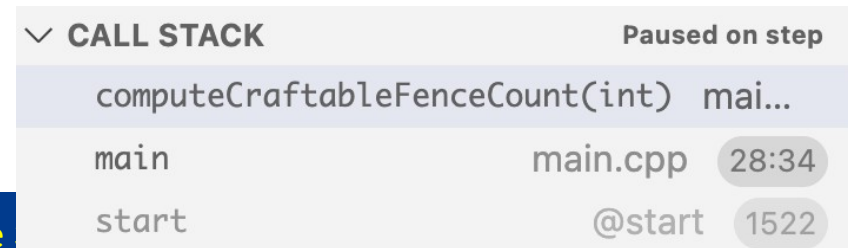
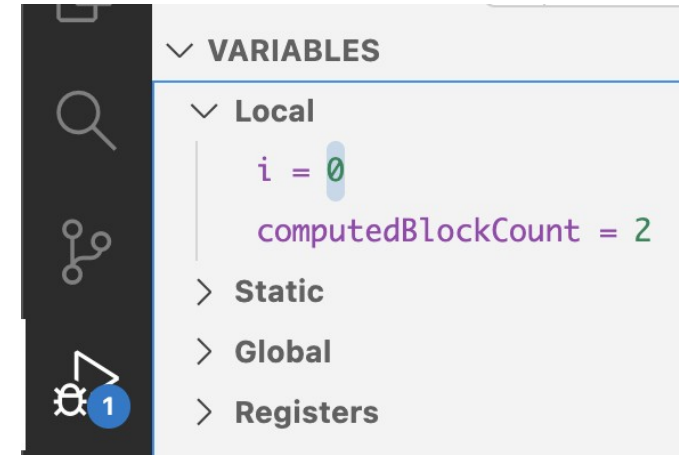
Debugging



Run until another line with a breakpoint is encountered
Stop at the next line in this function
Stop at the next line of code
Stop at the next line of code after current the function exits
Restart the program
Stop the program

The debugger shows you all variables and their values while you step through your program in the “run and debug” tab

As well as the call stack further down



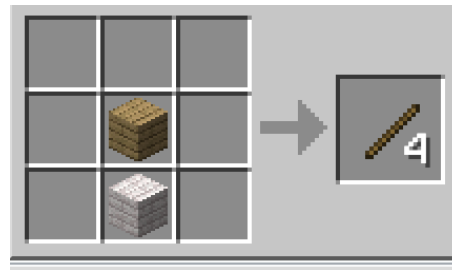
Task: Debugging

Task: Debugging

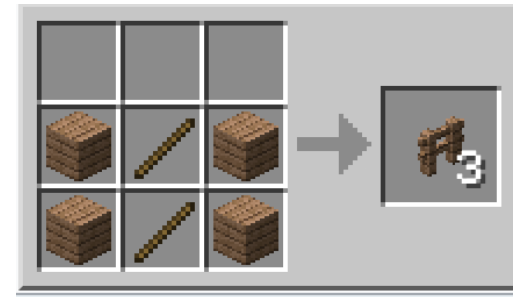
- I have written a function that computes how many oak fences can be crafted from a given number of oak logs, but it has a bug. Use the debugger to find it.
- In VS Code, use Create new project > Lecture 3 > Task - debugging



1 log -> 4 planks



2 planks -> 4 sticks



4 planks + 2 sticks -> 3 fences

Task: Debugging - Conclusions

- In order to fix bugs you need to understand exactly what your program does
 - It is often not enough to read through your code
 - Remember to use the debugger! It seems to be a tool that is often forgotten when people run into problems.

Today

Basics of the C++ Language

- Text input
- If statements
- Loops
- Switch statement
- Functions
- **Strings**
- Basics of graphics
- Some tips for writing code

Strings: sometimes weird

Windows

```
1  #include "std_lib_facilities.h"
2
3  int main() {
4      cout << "Går det an å bruke Norske tegn?" << endl;
5      return 0;
6  }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Går det an å bruke Norske tegn?

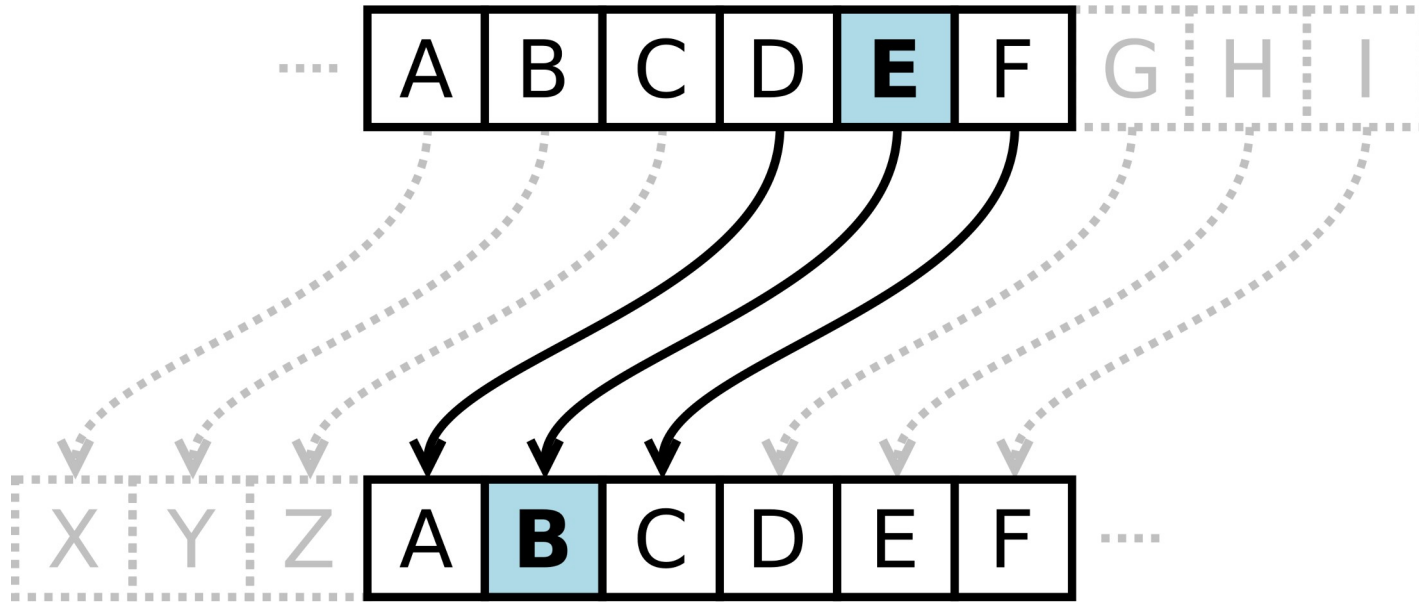
MacOS
and Linux

```
1  #include "std_lib_facilities.h"
2
3  int main() {
4      cout << "Går det an å bruke Norske tegn?" << endl;
5      return 0;
6  }
```

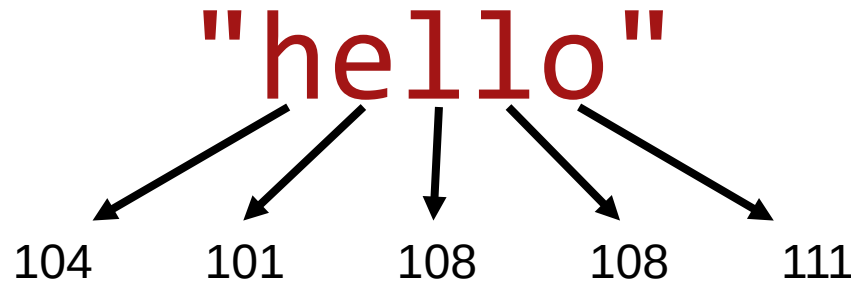
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Går det an å bruke Norske tegn?

Strings: the Caesar cypher



Fundamental problem: computers can only store numbers



Solution: we assign a number to each character

How numbers are interpreted as characters determines how they are shown on screen.

Windows by default uses an encoding that does not support Norwegian characters.

Mac and Linux use Unicode (UTF-8), which does.

Strings

```
string text;
```

```
text += 'H';  
text += 'e';  
text += 'y';
```

```
text += " there!";
```

```
text += "\n";
```

```
text += "Here is a number: "  
      + to_string(42);
```

```
cout << text << endl;
```

Unlike numeric data types, strings are automatically initialised to an empty string.

You can append single characters to a string using single quotes (')

Or append another string

Want a fresh row? Use the special `\n` character to create a new line.

Use the `to_string()` function to convert numeric types to strings before appending them.

You can print a string using `cout`, just like any other basic type.

Converting to and from string

Category	Data Type	Convert from string	Convert to string
Integers	char	stoi(value)	to_string(int(value))
	unsigned char	stoul(value)	to_string(int(value))
	short	stoi(value)	to_string(value)
	unsigned short	stoul(value)	to_string(value)
	int	stoi(value)	to_string(value)
	unsigned int	stoul(value)	to_string(value)
	long long	stoll(value)	to_string(value)
	unsigned long long	stoull(value)	to_string(value)
Real numbers	float	stof(value)	to_string(value)
	double	stod(value)	to_string(value)
Miscellaneous	bool	not applicable	to_string(value)
	enum	not applicable	to_string(value)

Examples:

```
string stringValue = "15";           double score = 1337.3;
int intValue = stoi(stringValue);    string converted = to_string(score);
```

Today

Basics of the C++ Language

- Text input
- If statements
- Loops
- Switch statement
- Functions
- Strings
- **Basics of graphics**
- Some tips for writing code

Graphics: AnimationWindow

- Used in most assignments
- Comes preinstalled with the VS Code extension
- A library we have written, not part of standard C++
- Detailed documentation (linked from course website):

<https://tdt4102.idi.ntnu.no/documentation/>

Graphics: AnimationWindow

- Why a graphics library?
 - Modern software development is to a large extent using other people's code. It's good to get some experience with what that is like.
 - We think drawing things is more fun than having some text appear on screen
 - Real graphics libraries are too complicated to include in a course like this, so we have made our own abstraction layer

Create a window

```
#include "std_lib_facilities.h"
#include "AnimationWindow.h"
```

AnimationWindow must be imported before it can be used

```
int main() {
    AnimationWindow window;
```

Creating a variable of type AnimationWindow opens a window

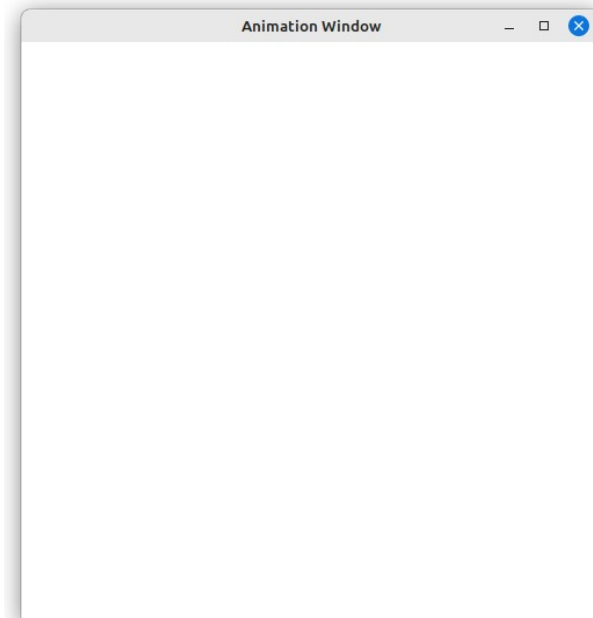
```
// Draw things here
```

We'll replace this with functions that draw shapes

```
window.wait_for_close();
return 0;
```

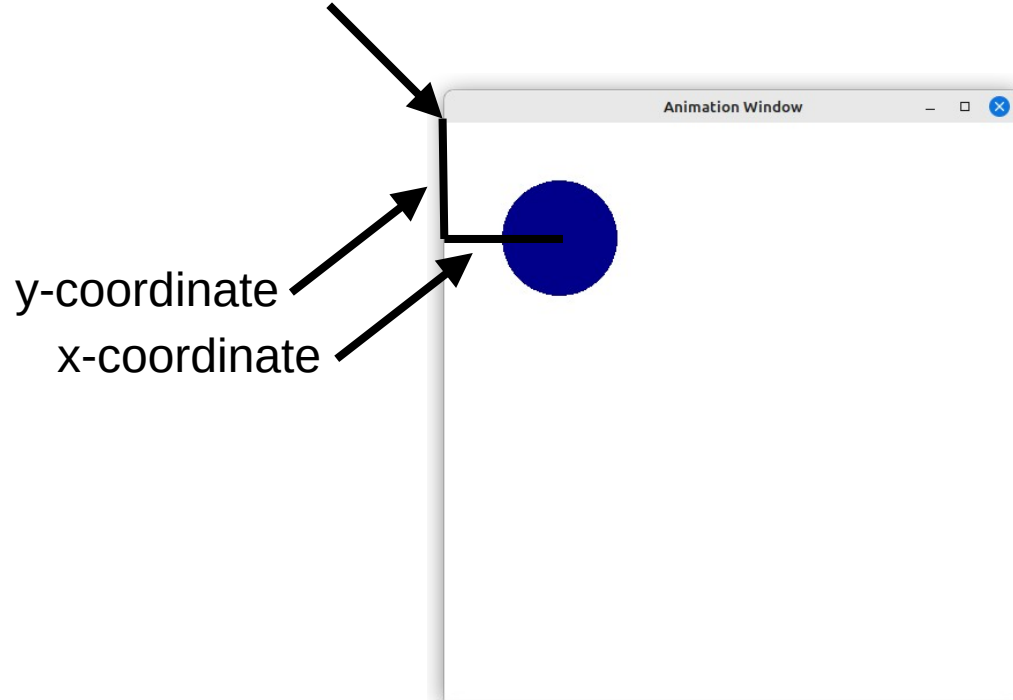
```
}
```

Without using wait_for_close() the program will exit immediately and in the process close the window.



Drawing: Coordinates

The origin (0, 0) is in the top left corner



All coordinates are measured in pixels!

Drawing: Circles

```
#include "std_lib_facilities.h"
#include "AnimationWindow.h"
```

```
int main() {
    AnimationWindow window;

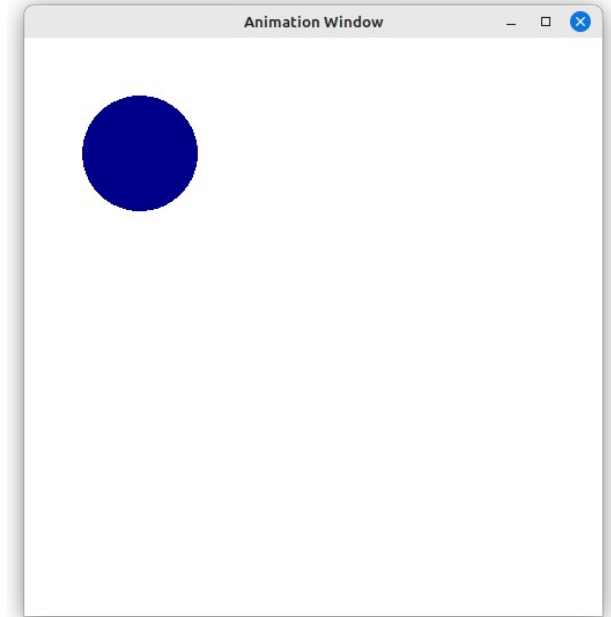
    window.draw_circle({100, 100}, 50);

    window.wait_for_close();
    return 0;
}
```

Radius of the circle



- Location where the circle should be drawn
- The function requires that we group the x and y coordinates using {}



Drawing: Line

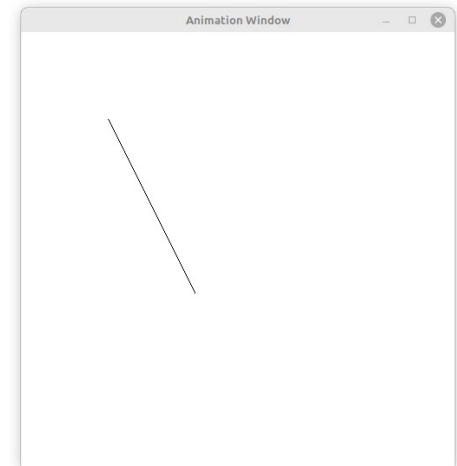

```
#include "std_lib_facilities.h"
#include "AnimationWindow.h"

int main() {
    AnimationWindow window;

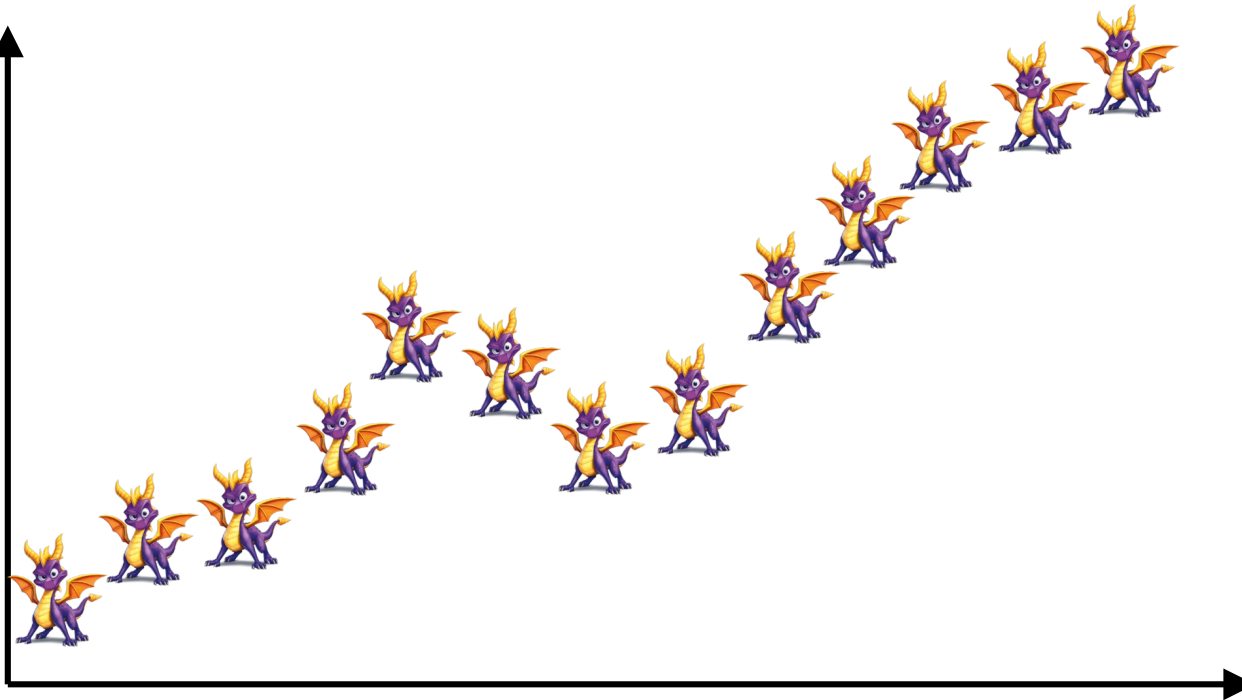
    window.draw_line({100, 100}, {200, 300});

    window.wait_for_close();
    return 0;
}
```

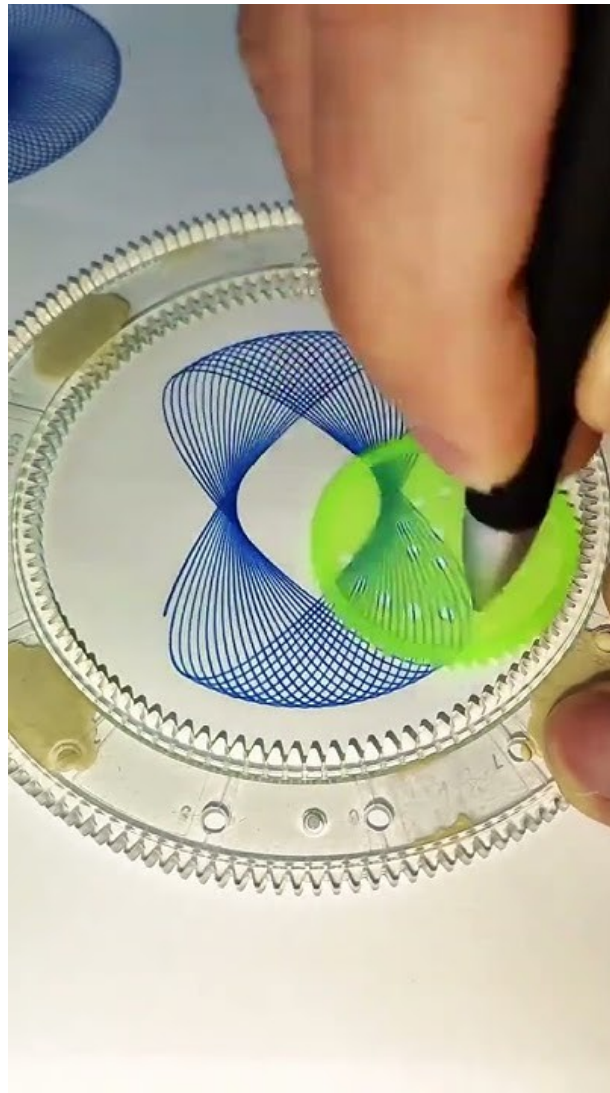
Start point End point



Demonstration



Demonstration



Drawing: Rectangles

```
#include "std_lib_facilities.h"
#include "AnimationWindow.h"
```

```
int main() {
    AnimationWindow window;

    window.draw_rectangle({100, 100}, 300, 160);

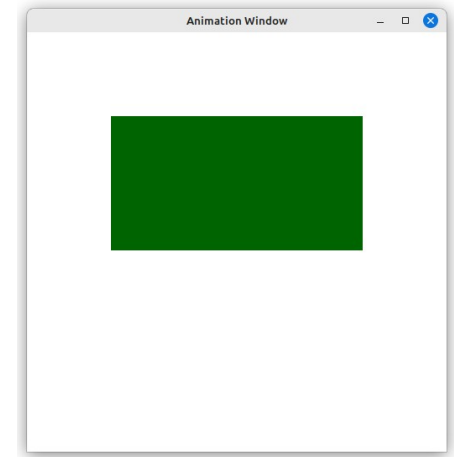
    window.wait_for_close();
    return 0;
}
```

Width Height

↓ ↓



Location where the top left
corner of the rectangle
should be positioned



Drawing: Quads

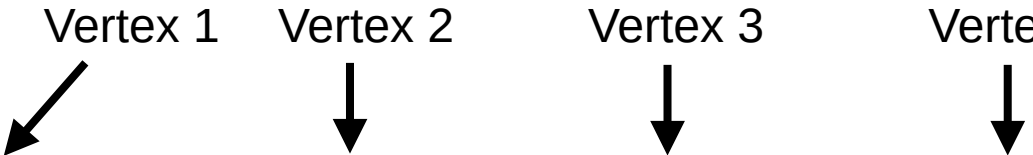
```
#include "std_lib_facilities.h"
#include "AnimationWindow.h"
```

```
int main() {
    AnimationWindow window;

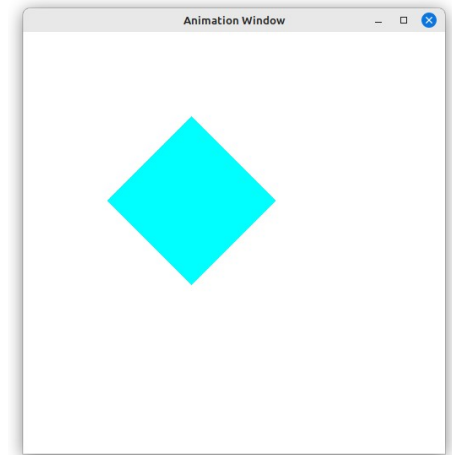
    window.draw_quad({200, 100}, {100, 200}, {200, 300}, {300, 200});

    window.wait_for_close();
    return 0;
}
```

Vertex 1 Vertex 2 Vertex 3 Vertex 4



The diagram shows four labels: 'Vertex 1', 'Vertex 2', 'Vertex 3', and 'Vertex 4'. Below each label is a black arrow pointing downwards to a specific coordinate pair in the function call `draw_quad({200, 100}, {100, 200}, {200, 300}, {300, 200})`. The first arrow points to `{200, 100}`, the second to `{100, 200}`, the third to `{200, 300}`, and the fourth to `{300, 200}`.



Drawing: Triangle

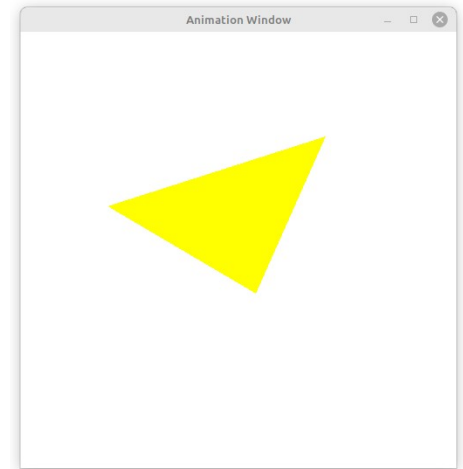
```
#include "std_lib_facilities.h"
#include "AnimationWindow.h"
```

```
int main() {
    AnimationWindow window;
    window.draw_triangle({350, 120}, {100, 200}, {270, 300});
    window.wait_for_close();
    return 0;
}
```

Vertex 1
↓

Vertex 2
↓

Vertex 3
↓



Drawing: Arc

```
#include "std_lib_facilities.h"
#include "AnimationWindow.h"
```

```
int main() {
    AnimationWindow window;

    window.draw_arc({250, 200}, 150, 150, 30, 200);

    window.wait_for_close();
    return 0;
}
```

Centre point Start angle End angle

Width Height



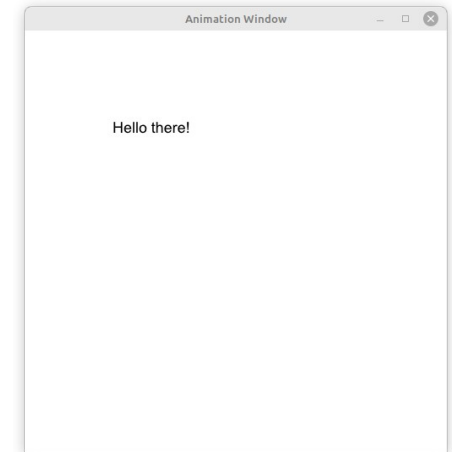
Drawing: Text

```
#include "std_lib_facilities.h"
#include "AnimationWindow.h"
```

```
int main() {
    AnimationWindow window;
    window.draw_text({100, 100}, "Hello there!");
    window.wait_for_close();
    return 0;
}
```

Starting point

Text to show



Drawing: Images

```
#include "std_lib_facilities.h"
#include "AnimationWindow.h"
```

```
int main() {
    AnimationWindow window;
```

```
    Image image("thisisfine.png");
    window.draw_image({100, 100}, image);
```

```
    window.wait_for_close();
    return 0;
```

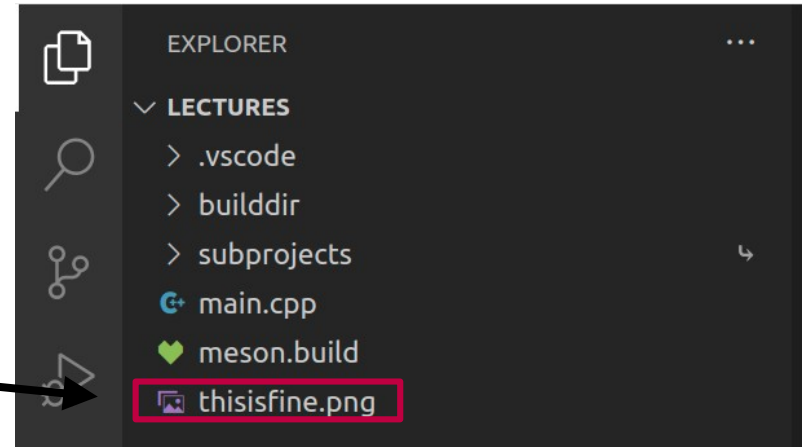
```
}
```

Name of the image file.
Must match exactly!

Where to draw
the image

image to
draw

Put images in
your project
directory



Drawing: optional parameters

```
#include "std_lib_facilities.h"  
#include "AnimationWindow.h"
```

```
int main() {  
    AnimationWindow window;
```

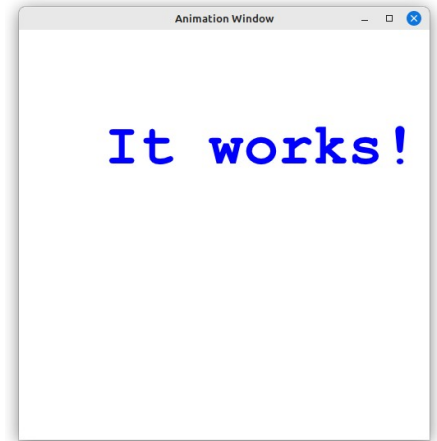
```
    window.draw_text({100, 100}, "It works!", Color::blue, 80,  
                      Font::courier_bold);
```

```
    window.wait_for_close();  
    return 0;
```

```
}
```

All shown drawing functions have optional parameters for changing things like colour. See the documentation for more information:

<https://tdt4102.pages.stud.idi.ntnu.no/documentation/>



Today

Basics of the C++ Language

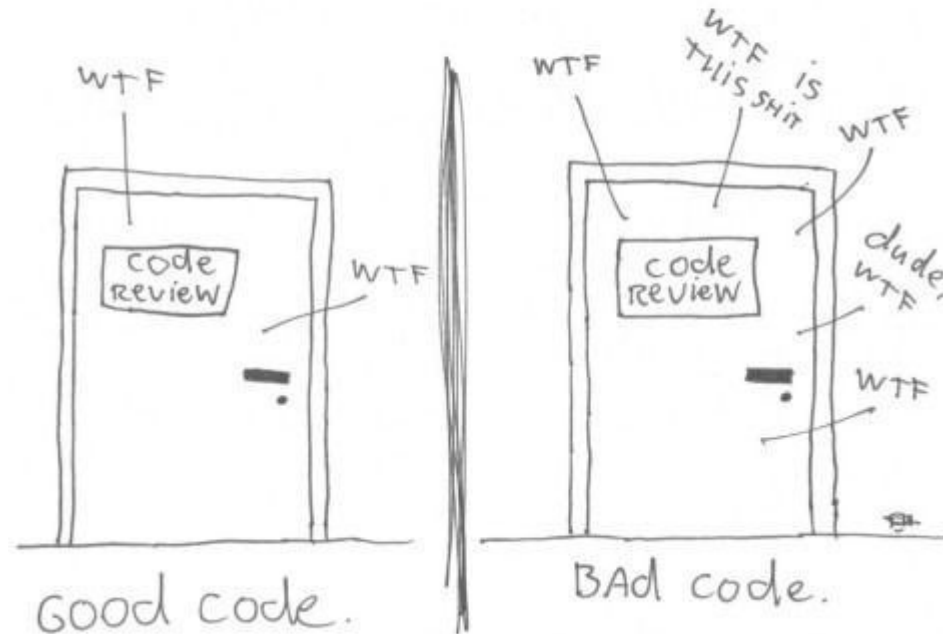
- Text input
- If statements
- Loops
- Switch statement
- Functions
- Strings
- Basics of graphics
- **Some tips for writing code**

Our code might work, but how well is it built?



Code is not always easy to understand

The ONLY valid measurement
of code quality: WTFs/minute



(c) 2008 Focus Shift/OSNews/Thom Holwerda - <http://www.osnews.com/comics>

What does this function do?

```
int r(int x, int y) {  
    int b = 0;  
    for(int q = x; q < y; q++) {  
        if(q % 2 == 1) {  
            b++;  
        }  
    }  
    return b;  
}
```

What does this function do?

```
int countOddNumbersInRange(int start, int end) {  
    int count = 0;  
    for(int i = start; i < end; i++) {  
        bool numberIsOdd = i % 2 == 1;  
        if(numberIsOdd) {  
            count++;  
        }  
    }  
    return count;  
}
```

Names help figure out what code does!

Names: Things you can do

- Communicate intent

```
int timeInDays;  
int visitedPageCount;  
bool buttonWasClicked;
```

- Names give meaning to values

```
if(distance > 42.195) {  
  
}
```

```
double marathonDistanceKm = 42.195;  
bool hasCompletedMarathon = distance > marathonDistanceKm;  
if(hasCompletedMarathon) {  
  
}
```

Names: Things you can do

- Don't hide information

// hard to see the difference

```
double dailyTemperatureMeasureIncrementInDegreesCelcius;  
double dailyTemperatureMeasureDecrementInDegreesCelcius;
```

// possible solution: shorten name

```
double temperatureIncrement_Celcius;  
double temperatureDecrement_Celcius;
```

// what can you expect to find in one of these?

```
string productInfo;  
string measuredData;  
int value;
```

Mars Probe Lost Due to Simple Math Error

BY ROBERT LEE HOTZ

OCT. 1, 1999 12 AM PT



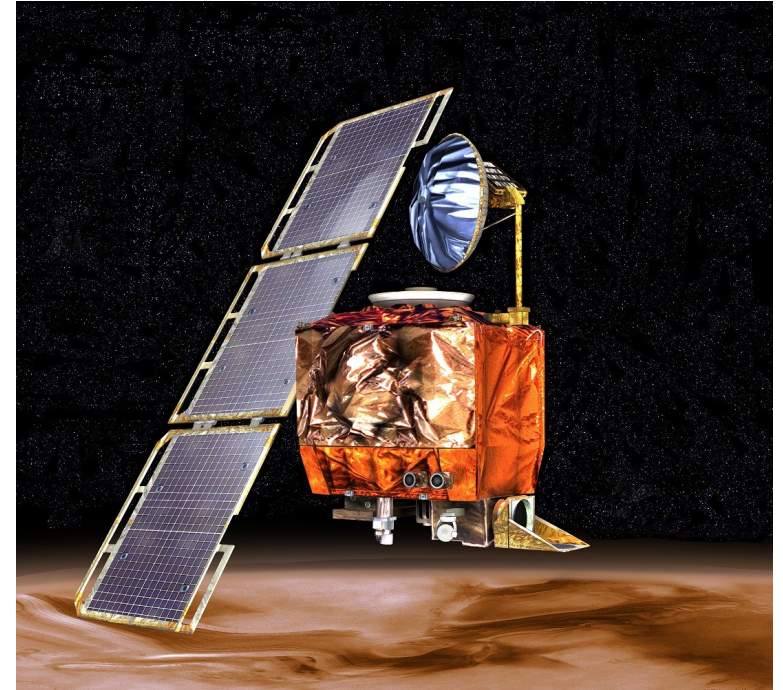
TIMES SCIENCE WRITER

NASA lost its \$125-million Mars Climate Orbiter because spacecraft engineers failed to convert from English to metric measurements when exchanging vital data before the craft was launched, space agency officials said Thursday.

A navigation team at the Jet Propulsion Laboratory used the metric system of millimeters and meters in its calculations, while Lockheed Martin Astronautics in Denver, which designed and built the spacecraft, provided crucial acceleration data in the English system of inches, feet and pounds.

As a result, JPL engineers mistook acceleration readings measured in English units of pound-seconds for a metric measure of force called newton-seconds.

In a sense, the spacecraft was lost in translation.



```
double distance = getDistance();
```