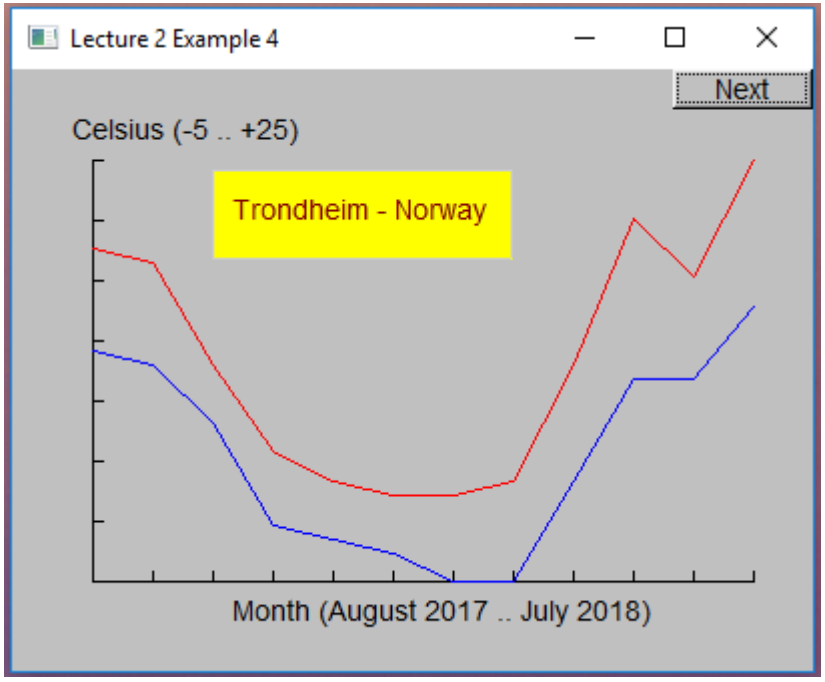
TDT4102 - Procedural and Object-Oriented Programming – Lecture 2

More C++ fundamentals and function parameters

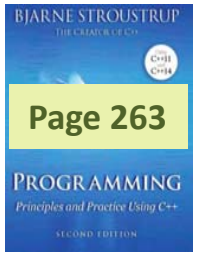


Norwegian University of Science and Technology

Overview Lecture 2

- Graphics example - continued
  - Plotting data stored in vector
  - Data dependant scaling of graphics
- Switch, string, input
- Symbolic constants and constexpr
- Declarations & scope
- Namespace and using directive
- Functions and parameter mechanisms

Initialization of variables



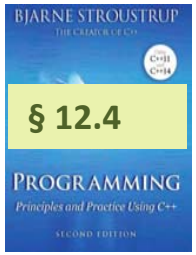
- Do it!
- C++ has several alternative syntaxes
  - `int a = 1;`
  - `int b(2);`
  - `int c{ 3 };`
- We recommend the { } initializer syntax
  - most general
  - most explicitly say initializer
  - is called ***universal and uniform intialization*** (PPP p. 83)
  - (PPP side 83, 263 og 311 m.fl.)

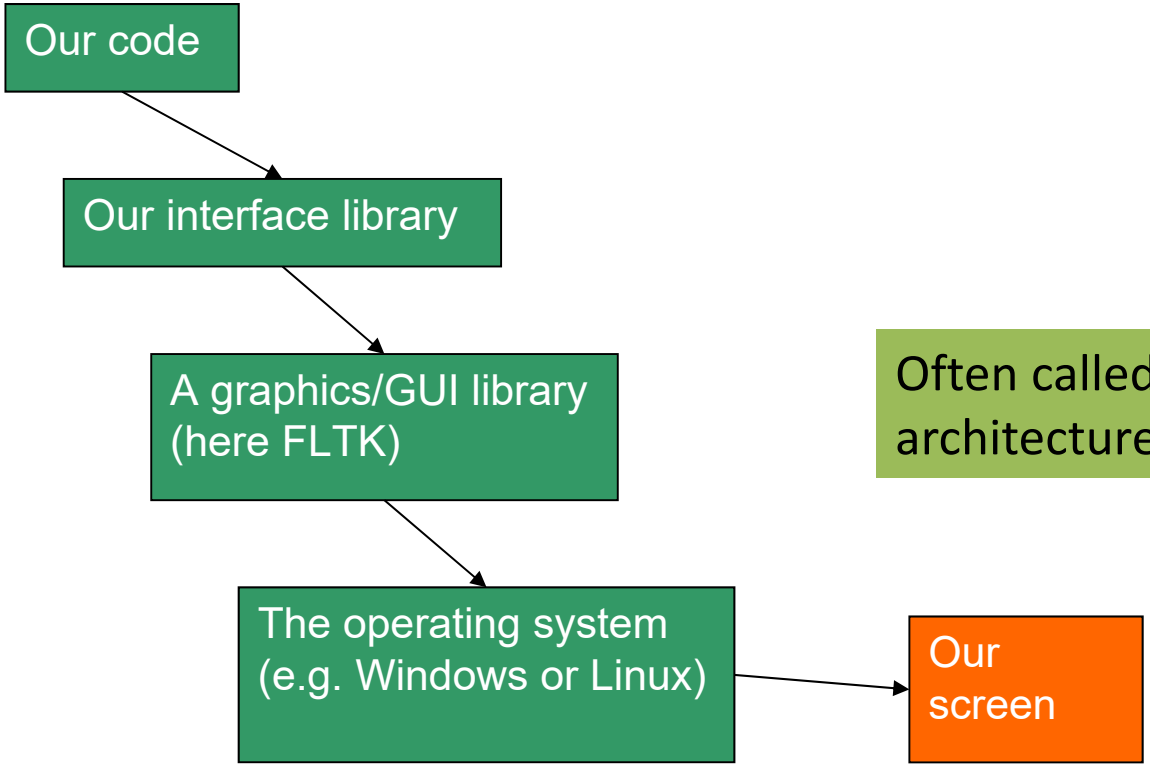
Open_polyline, example

- A series of connected line segments, between Points
  - Points can be added dynamically (during execution) by member function add
  - (More details in PPP § 13.6, covered later)
- Example:

```
17      vector<int> maxTemp{ 17, 16, 9, 3, 1, 0, 0, 1, 9, 19, 15, 23 };
18      vector<int> minTemp{ 10, 9, 5, -2, -3, -4, -6, -6, 1, 8, 8, 13 };
19
20      Open_polyline oplMax;
21      for (int i = 0; i < maxTemp.size(); i++) {
22          oplMax.add(Point{ origo.x + i, origo.y - maxTemp[i]});
23      }
24      oplMax.set_color(Color::red);
25      win.attach(oplMax);
```

Using a GUI library, «behind the scenes»





Often called "a layered architecture"

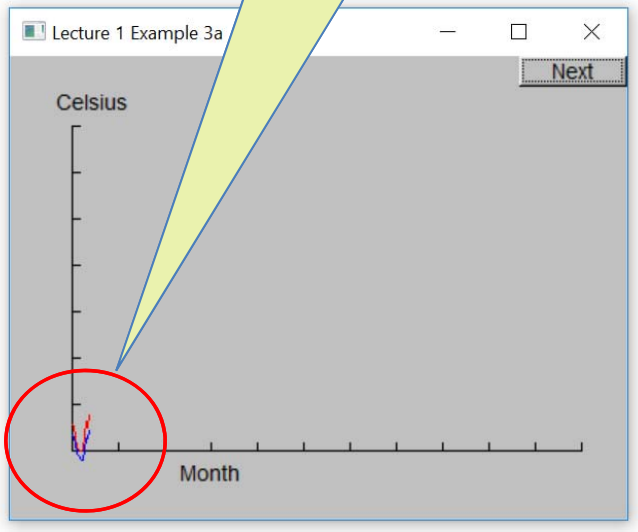
Example program Lec2Ex1a.cpp – part 1

```
1   // Lec2Ex1a.cpp (a first step, code will be improved!)
2   #include "Graph.h"
3   #include "Simple_window.h"
4   using namespace Graph_lib;
5   int main() {
6       Point tl{ 100, 100 }; // tl is Top-Left corner of our window
7       Point origo{ 40, 255 };
8       Simple_window win(tl, 400, 300, "Lecture 2 Example 5a");
9       Axis xa(Axis::x, origo, 330, 11, "Month");
10      win.attach(xa); // attach xa to the window, win
11      xa.set_color(Color::black);
12      Axis ya(Axis::y, origo, 210, 7, "Celsius");
13      win.attach(ya); // attach ya
14      ya.set_color(Color::black);
15
16      // two vectors of max and min temperatures in Trondheim for august 2017 to july 2018
17      vector<int> maxTemp{ 17, 16, 9, 3, 1, 0, 0, 1, 9, 19, 15, 23 };
18      vector<int> minTemp{ 10, 9, 5, -2, -3, -4, -6, -6, 1, 8, 8, 13 };
```

Example program Lec2Ex1a.cpp – part 2

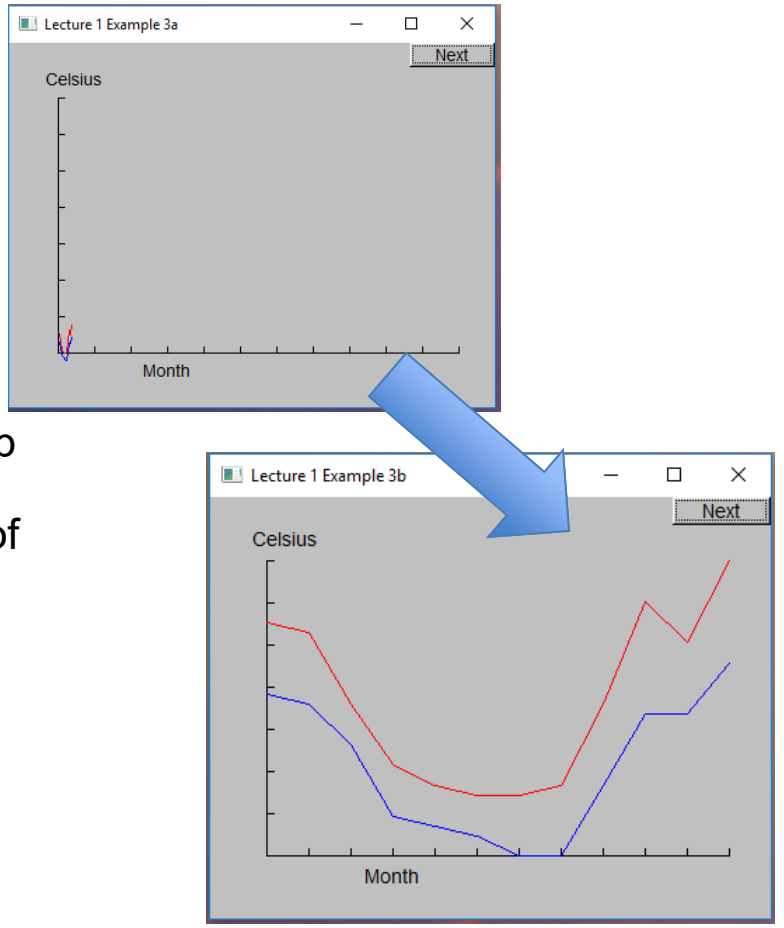
```
16      // two vectors of max and min temperatures in Trondheim for august 2017 to july 2018
17      vector<int> maxTemp{ 17, 16, 9, 3, 1, 0, 0, 1, 9, 19, 15, 23 };
18      vector<int> minTemp{ 10, 9, 5, -2, -3, -4, -6, -6, 1, 8, 8, 13 };
19
20      Open_polyline oplMax;
21      for (int i = 0; i < maxTemp.size(); i++) {
22          oplMax.add(Point{ origo.x + i, origo.y - maxTemp[i]});
23      }
24      oplMax.set_color(Color::red);
25      win.attach(oplMax);
26
27      Open_polyline oplMin;
28      for (int i = 0; i < minTemp.size(); i++) {
29          oplMin.add(Point{ origo.x + i, origo.y - minTemp[i]});
30      }
31      oplMin.set_color(Color::blue);
32      win.attach(oplMin);
33
34      win.wait_for_button();
35  };
```

We need scaling of the plotted data in our graph



Data dependant scaling

- X-axis
  - Plot lowest x-value at x = 0
  - Plot largest x-value at end of x-axis
  - (12 months numbered 0..11)
- Y-axis
  - Plot lowest y-value (from both vectors maxTemp and minTemp) at y = 0
  - Plot largest y-value (from both vectors) at end of y-axis
    - Max always larger than min
    - Calculate span along yAxis
    - span = max - min
- **Lec2Ex1b.cpp**
  - With functions min and max
  - Calculate the correct placement of each point as (x,y)



Example program Lec2Ex1b.cpp

```
                                                int totalMax = max(maxTemp); // h
                                                int totalMin = min(minTemp);
Open_polyline oplMin;                           int ySpan = totalMax - totalMin;
for (int i = 0; i < minTemp.size(); i++) {
    int temp = minTemp[i];
    int xCoord = origo.x + ((i * 330) / 11);
    int yCoord = origo.y - (210 * (temp - totalMin)) / ySpan;
    oplMin.add(Point{ xCoord, yCoord});
}
```

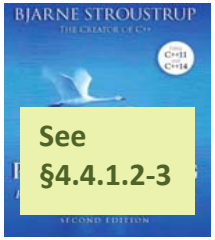
330 = length of x-axis,
11 = no of months -1,
210 = length of y-axis
These are all «magic numbers» --- that should be avoided if possible

Can be improved!

Switch (selection)

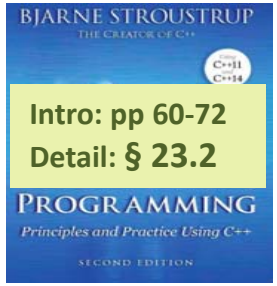
- Example (very simple “menu”):

```
int choice = -1;
cin >> choice;
switch (choice) {
case 1:
     inputIntegersAndPrintSum();
     break;
case 2:
     testIsOdd();
     break;
case 3:
     comparePlans();
     break;
default:
     cout << "Ugyldig valg\n";
}
```



A break “breaks out of” the nearest enclosing switch-statement, while-statement, do-statement, or for-statement; that is, the next statement executed will be the statement following that enclosing statement” § A.6

Introduction to <string>



- (<string> is included by our std_lib_facilities.h)
- `string s2 = "Hello";`
  - Declares a variable named `s2` to be a string-object and initializes it to a textstring with the five letters Hello.
- Characters in a string are numbered (indexed) 0, 1, …
- It has dynamic size!
- A variable of type string is an object, and you can call string-functions «on behalf of» that object:
  - `s2.length(); // returns the size of s2 in caracters`
  - `s2.insert(pos, str); // inserts the string object str`
    `// into s2 at position pos`

Converting between values and strings

- Convert values to string, examples:
  - `string s1 = to_string(12.333);`
  - `string s2 = to_string(1+5*6-99/7);`
  - The value of s1 is now "12.333" and the value of s2 is "17".;

  - ```
    string win_label{"Overview" + to_string(samples)
    + "per year"};
    ```
- There are many functions to convert string to values:
  - `double d = stod("17.18");`
  - `int i = stoi("190");`
  - `long l = stol("171845679");`
  - See also http://www.cplusplus.com/reference/string/
  - (Textbook (PPP) presents from_string)
    ```

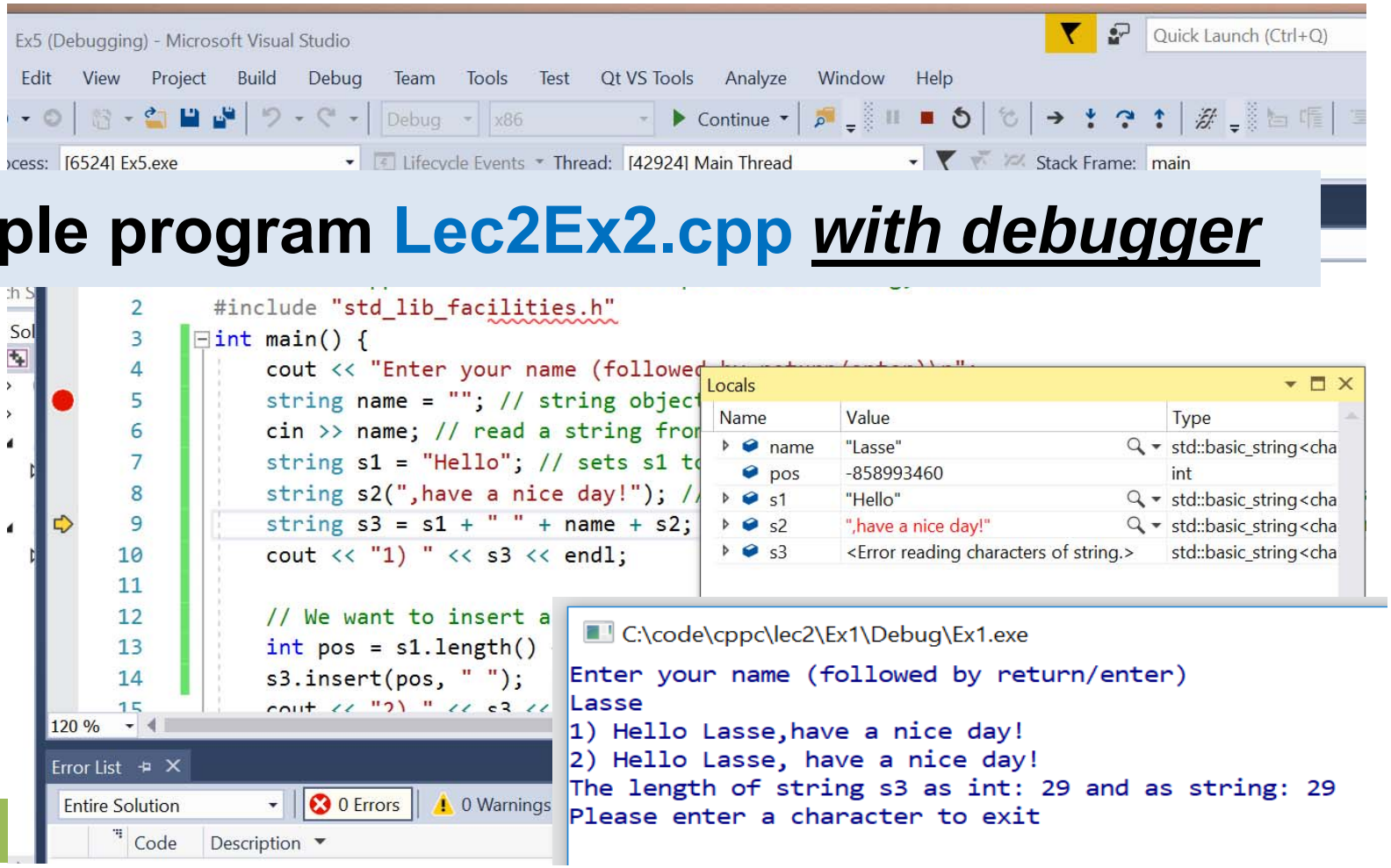
More on input

- cin will do its best to interpret the typed input as the type of the «receiving variable(s)»
  - Eg. 1 can be read as char, int or string
  - cin >> string_var
    - will read up to the next white_space character
- Line-oriented input (PPP § 11.5)

```
string name;
getline(cin, name); // input: Dennis Ritchie
cout << name << '\n'; // output: Dennis Ritchie
```

- cin.ignore() can be used to skip (rest of linje including) '\n'

Example program Lec2Ex2.cpp *with debugger*

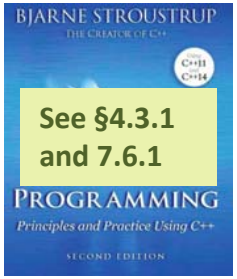


Debugger

TDT4102: --- Øving 2 – skal være «dekket» ---

Symbolic constants – const and constexpr

- **Avoid «magic constants»**
  - Like 3.141592653589793238, 12, -1, 365, 24, 2.718281828, 299792458 etc.
- A name should increase code readability, descriptive, not too long
- If a "constant" could change (during program maintenance) or if someone might not recognize it, use a symbolic constant
  - change in precision is often a significant change; 3.14 != 3.14159265
- 0 and 1 are usually fine without explanation, -1 and 2 sometimes (but rarely)
- If a constant is used twice, it should probably be symbolic
  - that way, you can change it in one place
- const
  - a modifier used in front of variable definition, says it cannot be changed



Symbolic constants – const

- Use symbolic constants
  - Examples;
    - `constexpr double pi = 3.14159;`
    - `constexpr double gravity = 9.80665;`
  - make your program more readable
- (Older C++ code use const for the same purpose)
- Improve Lec2Ex5b.cpp:

330 = length of x-axis,
11 = no of months -1,
210 = length of y-axis
These are all «magic numbers»

```
constexpr int xAxisSize = 330;
constexpr int yAxisSize = 210;
constexpr int maxMonthNo = 11;
```

```
Open_polyline oplMin;
for (int i = 0; i < minTemp.size(); ++i) {
    int temp = minTemp[i];
    int xCoord = origo.x + ((i * 330) / 11);
    int yCoord = origo.y - (210 * (temp - totalMin)) / ySpan;
    oplMin.add(Point{ xCoord, yCoord});
}
```

```
int xCoord = origo.x + ((i * xAxisSize) / maxMonthNo);
int yCoord = origo.y - (yAxisSize * (temp - totalMin)) / ySpan;
```

Declarations

- A declaration
  - introduces a name into a scope
  - specifies a type for the named object
  - sometimes includes an initializer
- A name must be declared before it can be used
- Examples:
  - `int a = 7; // an int variable named ‘a’`
  - `const double cd = 8.7;// a double-precision floating-point constant`
  - `double sqrt(double); // a function taking a double argument and`
    `// returning a double result`
  - `vector<Element> v; // a vector variable of Elements (variable)`

Scope

- A scope is a region of program text
  - Local scope (between { … } braces), called a block
  - Statement scope (e.g. in a for-statement)
  - Class scope (within a class) // Covered later
- A name in a scope can be seen from within its scope and within scopes nested within that scope
  - Only after the declaration of the name ("can't look ahead" rule)
- A scope keeps "things" local
  - Prevents my variables, functions, etc., from interfering with yours
  - Remember: real programs have **many** thousands of entities
  - **Locality is good!**
    - **Keep names as local as possible**

Scopes nest

```
int x;  // global variable – avoid those where you can
int y;  // another global variable

int f() {
   int x;   // local variable (Note – now there are two x's)
   x = 7;   // local x, not the global x
   {
    int x = y; // another local x, initialized by the global y
               // (Now there are three x's)
    ++x;       // increment the local x in this scope
   }
}

// avoid such complicated nesting and hiding: keep it simple!
```

Declarations and header files

- Declarations are frequently introduced into a program through header files
  - interface to other parts of a program
  - **allows for abstraction!**
    - **you don't have to know the implementation details of a function** like **cout** in order to use it.
- When you add **#include "std_lib_facilities.h"** to your code, the declarations in that header file become available for use (including **cout**, etc.).

Organizing code, namespace

- Use blocks { … } to organize code (§8.4, scope)
- Use functions (and *classes*, introduced later) to organize code
  - Can define variables without worrying that their names will clash with other names in our program
- Use namespace to organize/group functions, data, types and classes
  - a named scope
  - the scope operator :: specifies namespace
    - Examples:

      std::cout << x;

      Graph_lib::Text …

using Declarations and Directives

- To avoid typing
  - **std::cout << "...**

  you could write a “using declaration”
  - **using std::cout;** *// when I say **cout**, I mean **std::cout***
  - **cout << "...** *// ok: std::cout*
  - **cin >> x;** *// error: cin not in scope*

- or you could write a “using directive”
  - **using namespace std;** *// “make all names from namespace std available”*
  - **cout << "...** *// ok: std::cout*
  - **cin >> x;** *// ok: std::cin*

is included in std_lib_facilities.h

More on functions

- Chop a program into manageable pieces
- Let each function name one logical operation that it does well
- Ease testing, distribution of labor, and maintenance
- Keep functions small
  - Easier to understand, specify, and debug

Functions and parameter passing

- Returning a value // 8.5.2
- Pass by value // 8.5.3
- Pass by reference // 8.5.5
- Pass by const reference // 8.5.4

Functions – returnvalue

- Returning a value
  - (Introduced in Lecture 1)
  - The return value can be any expression
  - Many return statements in one function is possible
    - Function willl end its execution after first return-statement reached

```
int square(int x) {
    return x * x;
}
```

```
int max(int a, int b) {
    if (a<b)
        return b;
    else
        return a;
}
```

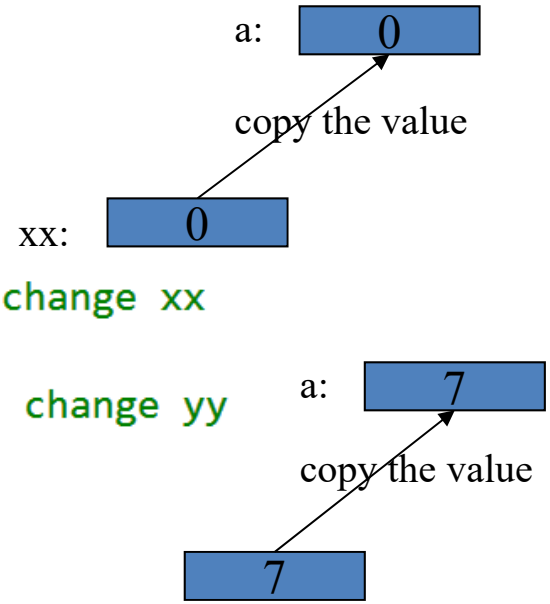
  - If no value should be returned, specify type of return value as void
    - return; // will terminate a void function
  - (More than one single value can be returned using a user defined type, <pair> or <tuple> // covered later)

Functions – Pass-by-value

```
// call-by-value (send the function a copy of the argument's value)

int f(int a) { a = a+1; return a; }

int main()
{
    int xx = 0;
    cout << f(xx) << '\n';  // writes 1
    cout << xx << '\n';     // writes 0; f() doesn't change xx
    int yy = 7;
    cout << f(yy) << '\n';  // writes 8;  f() doesn't change yy
    cout << yy << '\n';     // writes 7
}
```



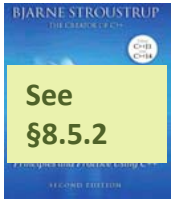
Example program Lec2Ex3.cpp

```
1   // Lec2Ex2.cpp, demonstrates two functions with a bug, both can
2   // "fall through the end of the function" and that can give undefined results
3    #include "../../std_lib_facilities.h"
4   int maxOfTwo(int a, int b) { ... }
10  double maxOfTwo(double a, double b) {
11      if (a > b)
12          return a;
13      else if (b > a)
14          return b;
15  }
16  int main() {
17      cout << maxOfTwo(2.0, 1.99999999) << endl;
18      cout << maxOfTwo(2.0, 3.0) << endl;
19      cout << maxOfTwo(2.0, 2.0) << endl;
20      cout << maxOfTwo(2.0, 1.9999999999999999) << endl;
```

```
C:\code\cppc\lec2\Ex2\Debug\Ex2.exe
2
3
-nan(ind)
-nan(ind)
```

Avoid «fall through»!
(PPP page 275)
Ensure that the function always will return a well defined value

What is a reference? – motivation



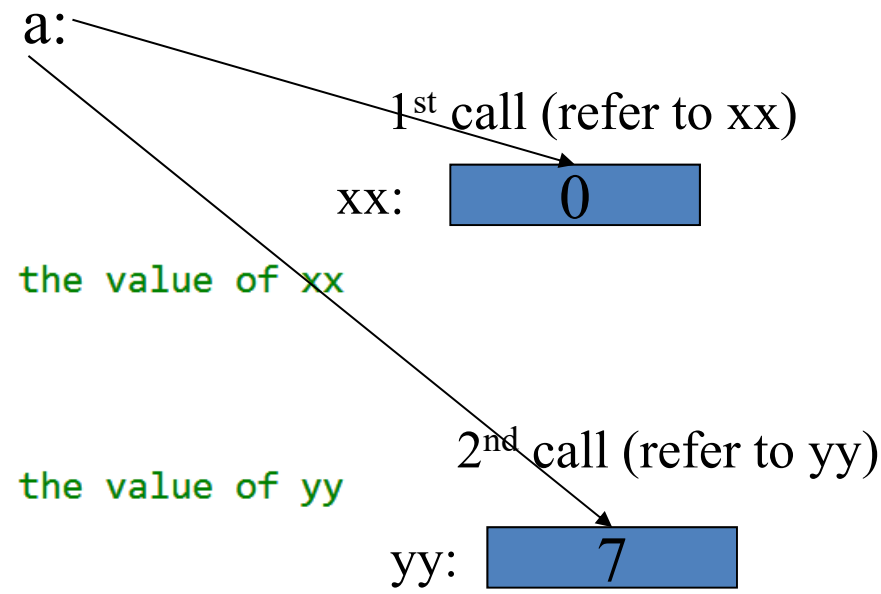
- Consider passing an image (eg. a large vector of pixel values) to a function displaying the image
  - Copy by value will copy every element
    - Takes extra time (and battery)
  - Better send the address of the vector to the function
    - Gives the function access to the vector using **reference**
    - Saves time and energy!!
- Think of a reference as an alternative name (alias)
- A const reference can be read but protects the data "referred to" against changes
  - ➔ More readable code, safer code, and potentially faster code

Example program
Lec2Ex4.cpp

Functions – Pass-by-reference

```
// call-by-reference (pass a reference to the argument)
int f(int& a) { a = a + 1; return a; }

int main()
{
    int xx = 0;
    cout << f(xx) << '\n'; // writes 1
                           // f() changed the value of xx
    cout << xx << '\n';    // writes 1
    int yy = 7;
    cout << f(yy) << '\n'; // writes 8
                           // f() changes the value of yy
    cout << yy << '\n';    // writes 8
}
```



Functions –
Call by value/by reference/by const-reference

```
//void f(int a, int& r, const int& cr) {
    //++a; ++r; ++cr; // error: 'cr': you cannot assign to a variable that is const
//}
void g(int a, int& r, const int& cr) {
    ++a; ++r; int x = cr; ++x;
}
int main() {
    int x = 0;
    int y = 0;
    int z = 0;
    g(x, y, z); // x==0; y==1; z==0
    // g(1, 2, 3);  // error: 'void g(int... cannot convert argument 2 from 'int' to 'int &'
    g(1, y, 3); // ok: since cr is const we can pass “a temporary”
}
```

Functions – parameter passing and performance

```
void searchVectorCopy(vector<int> v, int num) { // call by value (copy)
    for (unsigned int i = 0; i < v.size(); ++i) { ... }
}
void searchVectorRef(vector<int>& v_r, int num) { // call by reference
    for (unsigned int i = 0; i < v_r.size(); ++i) { ... }
}
void searchVectorConstRef(const vector<int>& v_cr, int num) { // call by const reference
    for (unsigned int i = 0; i < v_cr.size(); ++i) { ... }
}

int main() {
    clock_t b
    vector<in
    reportTim

    cout << e
    before =
    searchVec
    reportTim

    cout << e
    before =
    searchVec
    reportTim
```

```
C:\code\cppc\lec2\Ex3\Release\Ex3.exe
Num: 9999 found at index 258363
Num: 9999 found at index 11535563
Num: 9999 found at index 14420246
Num: 9999 found at index 30905431
Num: 9999 found at index 31049740
copy clock_ticks used:256

Num: 9999 found at index 258363
Num: 9999 found at index 11535563
Num: 9999 found at index 14420246
Num: 9999 found at index 30905431
Num: 9999 found at index 31049740
reference clock_ticks used:72

Num: 9999 found at index 258363
Num: 9999 found at index 11535563
Num: 9999 found at index 14420246
Num: 9999 found at index 30905431
Num: 9999 found at index 31049740
const reference clock_ticks used:69
```

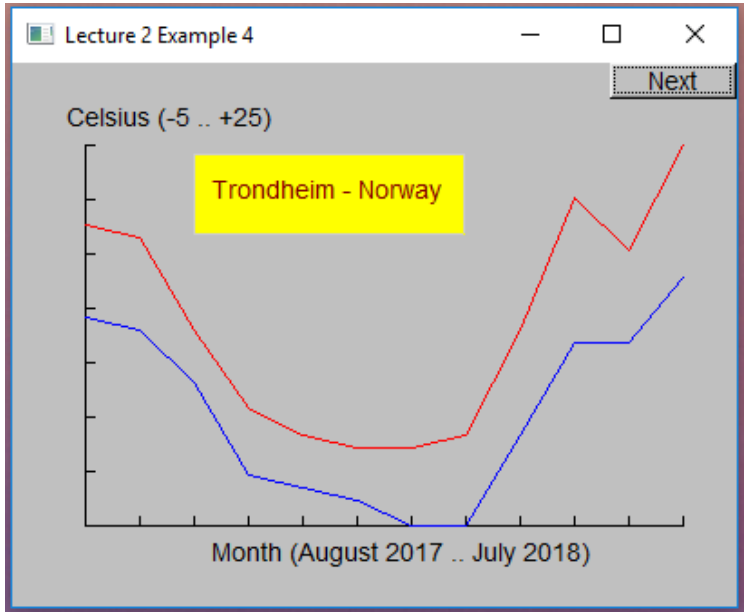
Example program
Lec2Ex4.cpp

Inspired by example at PPP page 277.
The example also demonstrates:

- Generating random numbers
- Measuring execution time

Lec2Ex1c.cpp

```
// an improvement of Lec2Ex1b.cpp
```



- Less magic numbers
- Repeated code for calculating xCoord and yCoord moved into functions
- Graphics: filled Rectangle and text
- Nested object declaration and initialization (Rectangle object)
- (Improved legend on xAxis and yAxis (still a naive solution))

TDT4102: --- Øving 3 – skal være «dekket» ---

TDT4102: --- Stopp forelesning 2 ---

- Regner ikke med å komme så langt som dette
- Reserve
  - Spørsmål fra salen?
    - C++
    - Administrativt?
  - Mer tid på noen av eksempelprogrammer?