VELDI KOMPETENS

Course outline week 5

- **Week 1**
  - Introduction
  - First program ”Hello world”
  - Integer Datatype
  - ”if” and ”else” statement
  - IDE
- **Week 2**
  - Datatypes continued
  - Namespace
  - For and while loops
  - Switch and jump statements
  - Arrays
- **Week 3**
  - Functions
  - Pointers
  - Exceptions
  - Lists
- **Week 4**
  - Preprocessor
  - Classes and Objects
  - Constructor and Destructor
  - Class methods
  - Class inheritance
- **Week 5**
  - Dynamic memory
  - File handling
  - Multiple files and headers
  - Libraries

VELDI

Introduction to C++
Part 1
-
Dynamic memory

Dynamic memory

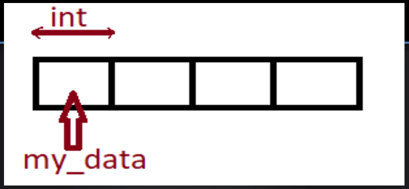
- Dynamic memory are memory that can be allocated and released during runtime
- Example of situations when this might be good
  - You do not know when the program starts, how much memory you need for “data”
  - You have a limited memory and need to reuse the memory by allocating and release many times
- To allocate memory, C++ uses the *new* operand
- To free the allocated memory, C++ uses the *delete* operand
- C++ does not have any automatic memory cleaner, so it is up to the programmer to release the memory, when no longer used

Operator *new* and *new[]*

- **Dynamic memory is allocated using operator *new*. *new* is followed by a data type specifier and, if a sequence of more than one element is required, the number of these within brackets []. It returns a pointer to the beginning of the *new* block of memory allocated. Its syntax is:**
  - pointer = new type
    pointer = new type [number_of_elements]

```
int *my_data;             //Pointer not set

my_data = new int [4];  //Now it points to a location where we have place for 4 "int":s
```



Operator *delete* and *delete[]*

- In most cases, memory allocated dynamically is only needed during specific periods of time within a program; once it is no longer needed, it can be freed so that the memory becomes available again for other requests of dynamic memory. This is the purpose of operator delete, whose syntax is:
  - delete pointer
  - delete [] pointer

```
int *my_data;            //Pointer not set
my_data = new int [4];   //Now it points to a location where we have place for 4 "int":s
int *single;             //Pointer not set
single = new int(19);    //Now it points to a location with single one int with value 19

delete[] my_data;        //delete array my_data
delete single;/          /delete the single int
```

VELDI

Operator *delete* and *delete[]*..

- Incorrect usage of the [] when delete memory might lead to memory leaks
  - If we allocate an array “kalle=new int[10]” and only use “delete kalle” we will tell the compiler to delete a single object and not an array
  - This will normally not be detected by the compiler, so those mistakes can be hard to detect
  - One good tool to use for detecting memory leaks like this is Valgrind (not a part of this course (or bonus) but will give output simular to below)

```
==3477== Mismatched free() / delete / delete []
==3477==    at 0x483D1CF: operator delete(void*, unsigned long) (in /usr/lib/x86_64-linux-
gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==3477==    by 0x10920C: main (a.cpp:11)
==3477==  Address 0x4db4c80 is 0 bytes inside a block of size 16 alloc'd
==3477==    at 0x483C583: operator new[](unsigned long) (in /usr/lib/x86_64-linux-
gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==3477==    by 0x1091DE: main (a.cpp:7)
```

Can be used to allocate classes also

- The new and delete operand can also be used with classes
- Note that we then uses "->" instead of "." on objects

```cpp
#include <iostream>

class Rectangle {
private:
	int width, height;
public:
	void set_height(int x){height = x;};
	void set_width(int x){width = x;};
	int get_area(){return width * height;};
};

int main() {
	Rectangle m_rect;
	Rectangle *my_rect;
	my_rect = new Rectangle;
	m_rect.set_height(5);
	m_rect.set_width(11);
	std::cout << "area: " << m_rect.get_area() << std::endl;
	my_rect->set_height(12);
	my_rect->set_width(10);
	std::cout << "area: " << my_rect->get_area() << std::endl;
	delete my_rect;		//we shall not use [] here, due to not used at new
}
```

VELDI

Can be used to allocate classes also..

- **BUT: If we miss the new allocation the program here will CRASH at runtime when trying to call the first method in the class**

```cpp
#include <iostream>

class Rectangle {
private:
int width, height;
public:
void set_height(int x){height = x;};
void set_width(int x){width = x;};
int get_area(){return width * height;};
};

int main() {
        Rectangle m_rect;
        Rectangle *my_rect;
        m_rect.set_height(5);
        m_rect.set_width(11);
        std::cout << "area: " << m_rect.get_area() << std::endl;
        my_rect->set_height(12);         //!!!!!!will crash here, we have not allocated memory!!!!!
        my_rect->set_width(10);
        std::cout << "area: " << my_rect->get_area() << std::endl;
        delete my_rect;                  //we shall not use [] here, due to not used at new
}
```

Introduction to C++

Part 2

-

File handling

VELDI

Using files

- With C++ you can write to a file, and read from a file quite easily
- This examples will use the <fstream> variant to read and write to files.
- The open function needs the filename as the first argument and optional also the opening mode flag/flags
- To make it possible to separate read and write operations the following streams exists
    - ofstream: Stream class for writing on files (Write only)
    - ifstream: Stream class for reading from files (Read only)
    - fstream: Stream class for both read and write from/to files (Both)

File opening mode

- **The file can be opened with the following modes**

ios::in	Open for input operations.
ios::out	Open for output operations.
ios::binary	Open in binary mode.
ios::ate	Set the initial position at the end of the file. If this flag is not set, the initial position is the beginning of the file.
ios::app	All output operations are performed at the end of the file, appending the content to the current content of the file.
ios::trunc	If the file is opened for output operations and it already existed, its previous content is deleted and replaced by the new one.

- **The default opening modes are:**

ofstream	ios::out (Will create file also)
ifstream	ios::in
fstream	ios::in | ios::out (Note can not be used at same time and will not create file)

VELDI

Writing to a file

- Let the file name end with *.txt* to create a simple text-file
- We can take either “ofstream” or “fstream” for writing.

```
#include <iostream>
#include <fstream>

int main () {
  std::ofstream myfile;           //uses ofstream (write only)
  myfile.open ("example.txt");    //open stream
  myfile << "Hello world!\n";     //write to stream
  myfile.close();                 //close stream
}
```

- Output file “example.txt” will have the following content:

```
Hello world!
```

Reading example.txt file

- We can take either “ifstream” or “fstream” for reading.
- To be sure we are in “read only” mode we use “ifstream”
- We use “getline” to read a line in the file as text

```
#include <iostream>
#include <fstream>

int main () {
  std::ifstream myfile;                    //uses ifstream (read only)
  std::string str;                         //temporary string
  myfile.open ("example.txt");             //open stream
  std::getline(myfile,str);                //read a line from file, store in str
  std::cout << str;                        //print the data from str
  myfile.close();                          //close stream
}
```

- If file “example.txt” exist, the following output will appear

```
Output:   Hello world!
```

Reading example.txt using “>>”

- When we wrote the ”file” we used the << to write directly on the stream
- This can also be done on reading operand but with “>>”, BUT then we will only get the first data until a space appear

```cpp
#include <iostream>
#include <fstream>

int main () {
  std::ifstream myfile;          //uses ifstream (read only)
  std::string str;               //temporary string
  myfile.open ("example.txt");   //open stream
  myfile >> str;                 //will read until a space appear
  std::cout << str;              //print the data from str
  myfile.close();                //close stream
}
```

- If previously file “example.txt” exist, the following output will appear

```
Output:  Hello
```

Writing multiple lines to a file

- To write multiple lines to a file you can use a for-loop

```cpp
#include <iostream>
#include <fstream>
#include <string>

int main() {
      std::string kalle[] = {"pineapple", "bananas","apples","lemons"};
      std::ofstream outfile;
      outfile.open("example.txt");
      for (std::string e : kalle) {
            outfile << e << std::endl;
      }
      outfile.close();
      return 0;
}
```

```
pineapple
bananas
apples
lemons
```

Reading multiple lines from a text file

- **Reading a text file is done by using ifstream and looping *getline()* function**

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

int main() {
	std::string kalle;
	std::ifstream infile;		//ifstream = input
	infile.open("example.txt");

	while (getline(infile,kalle)) {  //Loop until returns false=no more lines
		std::cout << kalle << std::endl;
	}
	infile.close();
	return 0;
}
```

```
Output:    pineapple
           bananas
           apples
           lemons
```

VELDI

Problem accessing file

- Sometimes the file is read-only, or you have an incorrect search path to the file, or wrong filename
- To solve this always check with *is_open()* function

```
#include <iostream>
#include <fstream>

int main() {
        std::string kalle;
        std::ifstream infile;
        infile.open("No_example.txt");
        if (infile.is_open()) {
                std::cout << "file opened OK";
        }
        else {
                std::cout << "Error opening file";
        }
        infile.close();
        return 0;
}
```

```
Output:  Error opening file
```

Dealing with a file located elsewhere

- If you want to read and write to a text file located somewhere else on your hard drive, you just write the entire file path instead of just the file name
- Note that you need to use double “\\” in path

```
#include <iostream>
#include <fstream>

int main() {
      std::string kalle;
      std::ifstream infile;
      infile.open("c:\\My_path\\example.txt");
      if (infile.is_open()) {
            std::cout << "file opened OK";
      }
      else {
            std::cout << "Error opening file";
      }
      infile.close();
      return 0;
}
```

Reading and writing to file

- **First do the write then close and reopen as read. Below write timestamp to file and then print using fstream**

```
#include <iostream>
#include <fstream>
#include <ctime>

int main() {
	std::string line;
	std::fstream file;
	file.open("example.txt", std::ios::out | std::ios::app);          //open as append output
	std::time_t t = std::time(0);                                     //get time now
	std::tm* now = std::localtime(&t);
	//write date tag to file
	file << (now->tm_year + 1900) << '-'  << (now->tm_mon + 1) << '-'  <<  now->tm_mday << " " <<
now->tm_hour << ":" << now->tm_min << ":" << now->tm_sec  << std::endl;
	file.close();
	file.open("example.txt",std::ios::in);       //reopen as input
	while (getline(file,line)) {                 //Loop until returns false=no more lines
		std::cout << line << std::endl;
	}
	file.close();
	return 0;
}
```

```
Output:           2021-10-19 13:26:52         //first time
                  2021-10-19 13:26:55         //second time also this
```

VELDI

Introduction to C++

Part 3

-

Multiple files and header files

VELDI

Multiple files and header files

- Big programs that only contain one file is not a good idea to create
- Instead, you can create multiple files that for example can be split into different type of behavior
- To get the “content” of those files, a header file will be used to expose the functions that shall be reachable outside that file
- When you want to use functions from that file just include the corresponding header file
- To illustrate this, we will create *add.cpp, add.h* and *main.cpp* in the coming slides

Example: add.cpp (source file)

- Here we create a file and save it as *add.cpp*
- This file just returns the value of 2 argument
- Note that this file do not need any include files to work

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

VELDI

Example: add.h (header file)

- Here we create a file (header file)and save it as *add.h*
- This will describe the “public” (add) function in add.cpp
- Note the “#ifndef, #define and #endif of ADD_H, those are there to make sure that only one instance of this will appear if this file will be used by more than one file

```
#ifndef ADD_H
#define ADD_H

int add(int a, int b); //our function from add.cpp

#endif  // ADD_H
```

Example: main.cpp (main file)

- Here we create a file and save it as *main.cpp*
- Here we include both <iostream> and “add.h” file

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

int main()
{
    int a = 2;
    int b = 1;
    std::cout << a << " + " << b << " = " << add(a,b);
    return 0;
}
```

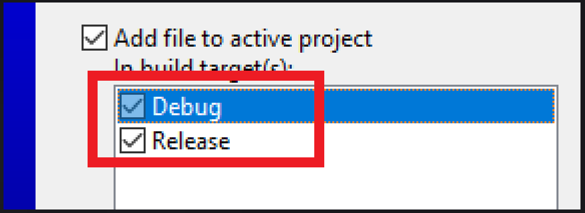
- Now we are done and can compile the program

Include paths, <X> or “X”

- You might have noticed that we used “add.h” instead of <add.h> when we added the include files in the previous slide
- If we just add “add.h” we refer to the current directory
- If we write <add.h> we refer to the current *include path* directory
- So, if we tell the compiler the correct path to include files the best way is to use the <add.h> instead of the “add.h” that we used in previous slide

Example in codeblocks

- In codeblocks you add files with File->New->File
- Note that you need to add them also to both release and debug targets while adding the files

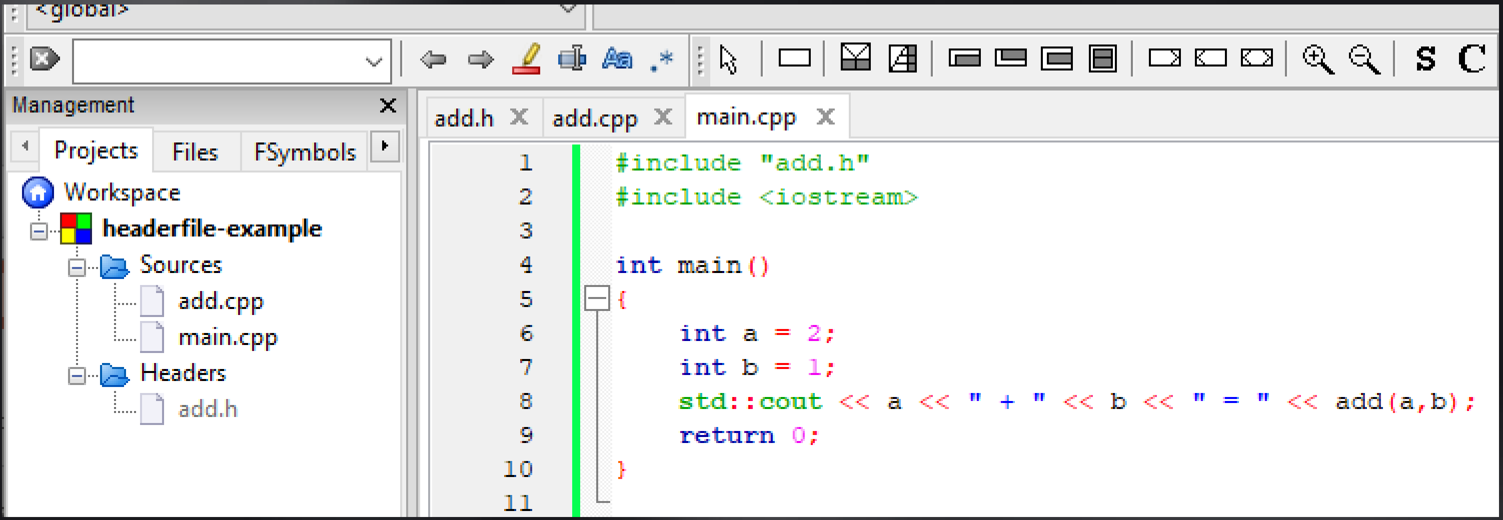


☑ Add file to active project

In build target(s):

☑ Debug

☑ Release



```cpp
#include "add.h"
#include <iostream>

int main()
{
    int a = 2;
    int b = 1;
    std::cout << a << " + " << b << " = " << add(a,b);
    return 0;
}
```

Introduction to C++

Part 4

-

Creating libraries

VELDI

What is a library ?

- In Windows you might have heard about DLL-files
- In Linux you might have heard about “libs” and “so” files
- This type of files exist in Linux, Windows, Mac etc. and are binary pre-compiled files
- When to use ?
    - If you have done a nice function and don´t want to share the source code but want other to be able to use the function a library is a perfect choice
    - To let high level program, use low level hardware
    - For example: writing a library in C++ and use it in python, C# or other languages
- Note that you need to deliver different pre-compiled libraries for each platform you want to support

Creating a DLL in Codeblocks

- File->New->Project->Dynamic Link Library (choose default and name project to "dll")
- You will be given a basic dll with one function named "SomeFunction" that takes a string as argument
- You can add more function in the same way as that function with the prefix "DLL_EXPORT" in the beginning.

```
void DLL_EXPORT SomeFunction(const LPCSTR sometext)
```

- We now add a new function to the *main.cpp* file:

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

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

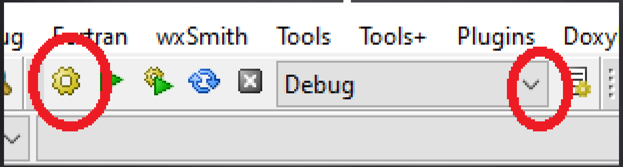
- And the prototype to the header file *main.h*

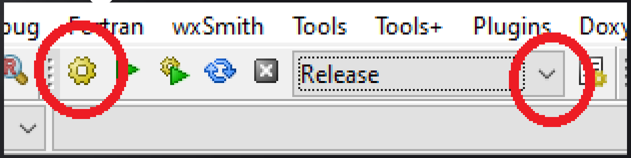
```
int DLL_EXPORT add(int a, int b);
```

```
int DLL_EXPORT add(int a, int b);
```

Creating a DLL in Codeblocks..

- Then we compile both the debug and the release version

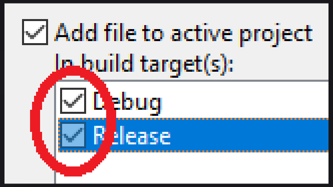




- The DLL shall now be complete, and you can find it under the projects debug and release folder
    - Project-folder/dll/bin/Release/dll.dll
    - Project-folder/dll/bin/Debug/dll.dll
- You will notice that the release binary is smaller than the debug binary, this is due to the debug version includes debug symbols
- You will notice some other code the *main.cpp* file and some defines in *main.h*, those are for creating a DLL and we will not go into those in this course

Using our DLL in Codeblocks..

- File->New->Project->Console application (Select default with C++ and name project "use-dll")
- On our new project: File->New->File..->C/C++ header and crate a new *main.h* file and add for both debug and release by clicking them both.



- Copy the complete content from the *main.h* file in the "dll" project and paste into our new *main.h* file
- Replace *main.cpp* file with the following content:

```
#include <iostream>
#include "main.h"
int main()
{
    int result = add(1,3);
    std::cout << "Dll usade:" << result << std::endl;
    return 0;
}
```

Using our DLL in Codeblocks..

- Select “Build options…” on the project and on the “Linker” tab, add the *dll.dll* file created in the DLL example (you need to select *.* to see the DLL)

  (select keep this as relative path)
- Now we can build both release and debug version of our project
- To be able to run and debug the “use-dll” project we also need to copy the *dll.dll* to the project in the respective release and debug folder
  (or better on the project select Properties->Build targets and change “Execution working dir” to the directory where the dll is located)
- Now we can also run and debug the project in codeblocks

Example: Using our DLL python (Bonus)

- The created dll.dll can also be used from other programming languages, below is an example how to use it from python
- (This is just an example how to use it from python, not a part of this course)

```
import ctypes
dll = ctypes.WinDLL('c:\\temp\\dll.dll')
print(dll.add(12,12))
```

```
Output:    24
```

Conclusion's libraries

- C++ is a perfect tool to create hardware related libraries or libraries that needs to be executed fast
- Libraries can then also be used by other programming languages
- A normal windows installation consists of thousands of dll files that can be used in the same way
- We will soon go through how to use libraries to create GUI (graphical user interface) application instead of the console applications that has been used so far in this course
- You need to have some header-file or documentation of the DLL to know the available functions inside it and how to use them

Introduction to C++

Bonus

-

GUI libraries

VELDI

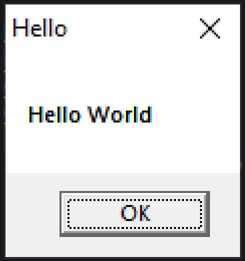
GUI (Graphical User Interface)

- So far in this course, we have only covered “console” based application
- We will now go through some examples of how to make program that uses GUI libraries in the operative system
- We will first start with the lowest library to create windows applications
- This is the win32 API

(and the “32” is more a name and exists as both 32-bit and 64-bit versions of windows so this is more a name that has lived for a long time since the earliest days of windows)

Win32 API

- Low level in Win32 GUI programming
- Will only work on Windows operative systems


Hello

Hello World

```
#include <windows.h>

int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine,
                   int nCmdShow) {

  MessageBox(NULL, "Hello World", "Hello", MB_OK);

  return 0;
}
```

- This is just an example you do not need to know anything about the syntax

VELDI

Win32 API Good and bad

- Bad
  - Will only run, on Windows operative systems
  - Complicated to create programs in
    - Takes longer time to create programs

- Good
  - Will produce small files with extremely low dependencies
  - Will follow windows low level “look and field”

C++ GUI libraries

- QT
  - This is probably one of the best and mostly used cross platform GUI libraries right now. Good for both embedded and PC applications.
- FLTK
  - Small light toolkit to create GUI application, cross platform
- wxWidgets
  - Quite small but still powerful cross platform GUI toolkit
- GTK+
  - Has been around for many years and uses c, cross platform
- MFC
  - Microsoft Foundation Class, used in visual studio, Windows only
- EFL (Enlightenment Foundation Libraries)
  - Started as a window manager for a long time ago, using c, cross platform

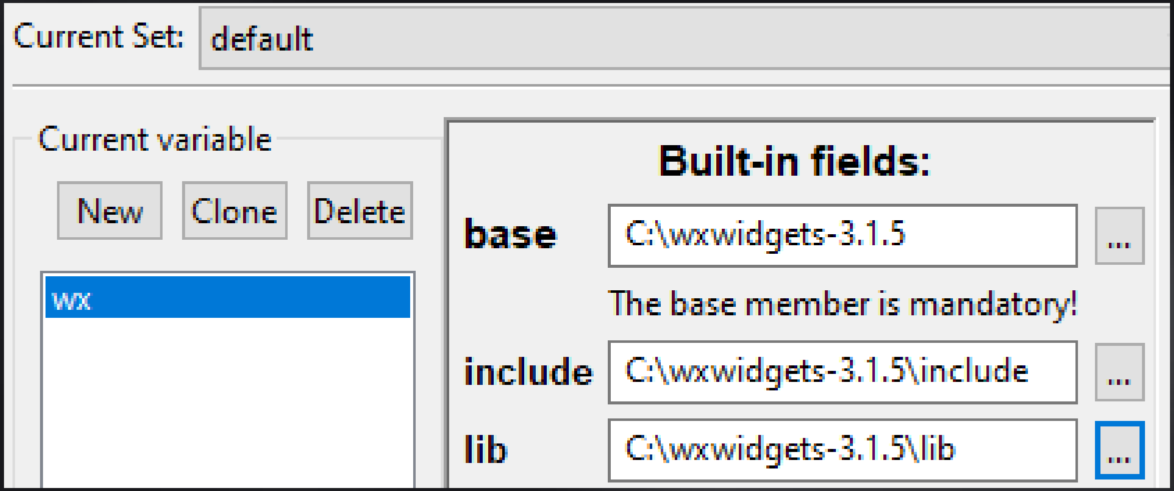
wxWidgets

- Setup

VELDI

wxWidgets in Codeblocks

- We could have chosen any GUI libraries, but we took this one mostly because it is both simple and small to install
- Decompress the “wxWidgets-3.1.5-x86_64” file using 7-zip (www.7-zip.org)
- Copy the 2 files in _missed_cd_ in wxWidget package to your codeblock IDE in folder MinGw/Lib/
- Start Codeblocks, Settings->Global variables create a new wx global variable with the following settings



Current Set: default

Current variable: wx

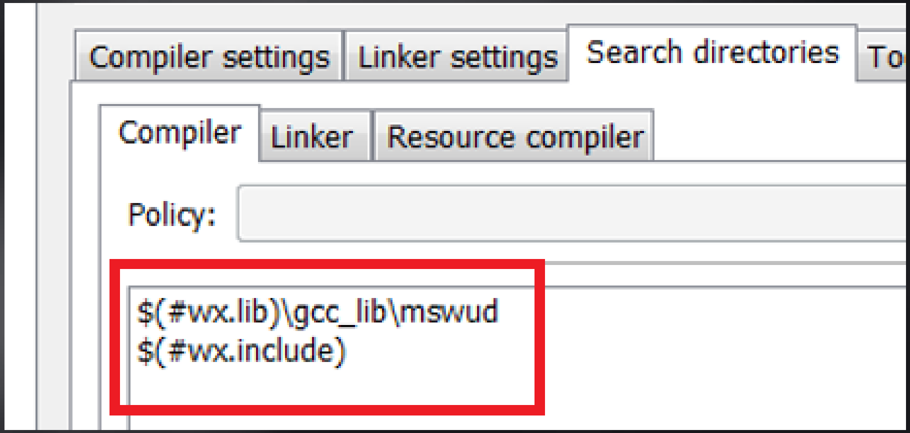
Built-in fields:

base	C:\wxwidgets-3.1.5
	The base member is mandatory!
include	C:\wxwidgets-3.1.5\include
lib	C:\wxwidgets-3.1.5\lib

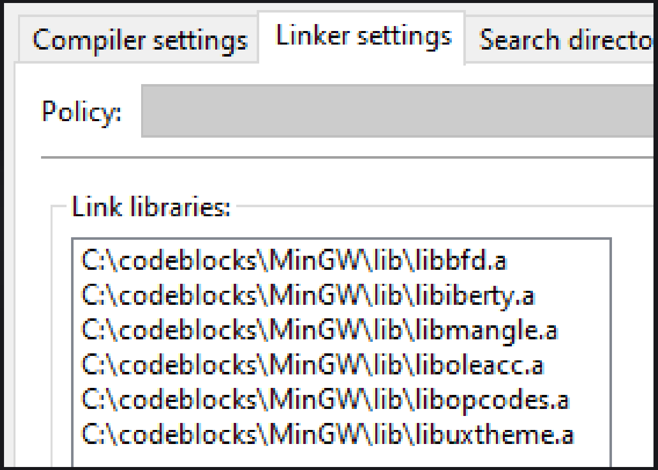
VEIDI

wxWidgets in Codeblocks..

- Settings->Compiler->Search Directories



```
$(#wx.lib)\gcc_lib\mswud
$(#wx.include)
```



```
C:\codeblocks\MinGW\lib\libbfd.a
C:\codeblocks\MinGW\lib\libiberty.a
C:\codeblocks\MinGW\lib\libmangle.a
C:\codeblocks\MinGW\lib\liboleacc.a
C:\codeblocks\MinGW\lib\libopcodes.a
C:\codeblocks\MinGW\lib\libuxtheme.a
```

- Under tab “linker settings” add all libraries in codeblocks/MinGw/lib
- Now you can create wxWidgets 3.1 application in codeblocks

VELDI

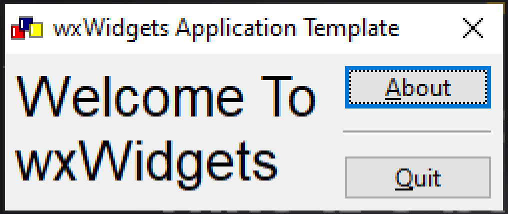
wxWidgets

- Example

VELDI

wxWidgets Example in Codeblocks

- File->new->Project->wxWidgets project and create the project for wxwidget version 3.1
- Give it a name and fill in author data
- Don´t forget to select the “Enable Unicode” option in project wizard
- An example project might look like below:



Introduction to C++
Bonus
-
Misc libraries

VELDI

Working with Excel sheet

- Download and install LibXL from www.libxl.com
- “g++ -Lbin64 -Iinclude_cpp -lxl main.cpp”

```cpp
#include "libxl.h"
using namespace libxl;

int main()
{
    Book* book = xlCreateBook(); // xlCreateXMLBook() for xlsx
    if(book)
    {
        Sheet* sheet = book->addSheet("Sheet1");
        if(sheet)
        {
            sheet->writeStr(2, 1, "Hello, World !");
            sheet->writeNum(3, 1, 1000);
        }
        book->save("example.xls");
        book->release();
    }
    return 0;
}
```

- If you want an open-source variant, you can use OpenXLSX

What you have learned in this course

- Week 1
  - Introduction
  - First program ”Hello world”
  - Integer Datatype
  - ”if” and ”else” statement
  - IDE
- Week 2
  - Datatypes continued
  - Namespace
  - For and while loops
  - Switch and jump statements
  - Arrays
- Week 3
  - Functions
  - Pointers
  - Exceptions
  - Lists
- Week 4
  - Preprocessor
  - Classes and Objects
  - Constructor and Destructor
  - Class methods
  - Class inheritance
- Week 5
  - Dynamic memory
  - File handling
  - Multiple files and headers
  - Libraries

VELDI

veldikompetens.se

Thank you!

VELDI KOMPETENS