University of Duisburg-Essen
Networked Embedded Systems Group
Dept. of Computer Science & Business Information Systems
Schützenbahn 70
45127 Essen, Germany

Programming in C/C++
Exercises Organization

Marcus Handte

Introduction

- The purpose of this exercise
  - Provide practical programming experience in C and C++.
  - Deepen your understanding of the language.
  - Make you familiar with data structures and algorithms.
- The exercises
  - **Are mandatory** for the examination.
    - 60% of all points are required to qualify for exam
  - Are useful for understanding the lecture.
  - Contain additional information that is not covered in the lecture.
- All exercises are programming exercises
  - We expect that you are familiar with Java.

Exercise Instructor

- Marcus Handte
  - Email: marcus.handte@uni-due.de
  - Room: S-A 121
  - Schützenbahn 70, 45127 Essen

- Networked Embedded Systems Group (NES)
  - Institut für Informatik und Wirtschaftsinformatik (ICB)
  - Lehrstuhl für Pervasive Computing

- Group Website: http://www.nes.uni-due.de/

The Course

- Meeting Time:

	Time		Place
Lecture	Wednesday,	12:00 – 14:00	SE 407
Exercises	Thursday,	16:00 – 18:00	SE 407

- Course Website
  - All course materials are on *Moodle2* (http://moodle2.uni-due.de/)
  - Go to our website http://www.nes.uni-due.de and click on the moodle link on the teaching page
    - Or search for “Programmieren in C/C++” directly in Moodle
  - Access Key : ***CPPWT1920*** (**please write it down, now!)**

Important Note: Mail Communication

- Short–notice announcements will be posted via Moodle
- Make sure that you register there with a mail address that you read regularly!

Requirements

- Participation in the exercise is mandatory for the lecture.
- 60% of all points are required to register for the final exam
  - 80-89% you get a 0,3/0,4 bonus on the final grade,
  - >90% you get a 0,6/0,7 bonus on your final grade!

- To successfully finish the exercises, you need to:
  - think and program,
  - be familiar with OOP, and
  - hand in your exercise solution program in time
  - Present your solution to one of the tutors.

Access Key : ***CPPWT1920***

Exam

- It will most likely be a written exam
  - You will be notified if this changes
- You will be expected to code and answer questions on C/C++
  - Use the exercises for practice!
- More details on exam organization will be provided towards end of lectures
  - Stay tuned on Moodle

Assignments

- Assignments will be posted on *Moodle*
  - Total of 11 assignments, starting next week.
  - A new assignment will be issued *every Thursday*.
- Assignment submission
  - Start your assignment early
  - For each assignment submission, you need to:
    1. include your implementation code with comments
    2. zip your files and upload the archive to Moodle
    3. Present your solution during the exercise session
  - Deadline for each assignment is Tuesday, 23:55H
  - Assignments submitted after deadline will not be evaluated

Assignment Evaluation

- The total points for each assignment is 100 points. Divided approx. as follows:
  - *25 points* for the comments/code documentation
  - *75 points* for the implementation code & report

- You are **required** to present your solution to a tutor
  - Will be done during the exercise hour
  - Each presentation will be done individually
  - The correct solution will be presented at the end
  - If you do not present your solution, your submission will not be graded

Assignment Presentation

- In the presentation, the following is expected of you:
  - Explain how you solved the task
  - Why you used any particular code constructs
  - Answer some relevant questions to the exercise

- Presentation and grading will be done individually
  - Presentations will start at 16H and for one hour.

- The correct solution will be demonstrated from 17H
  - Also discussion of next week‘s exercise tasks

Code Comments

- Include a short summary of what each class/method does
  - Brief, one sentence – if I want to know more, I'll read the code
  - Don't try to describe all of what the code is doing

- Include why the code does what it does (and not how)
  - Include special cases / gotchas / or things to watch out for

- Document method input parameters
- Document the method return value
- Name your variables, methods and classes properly
  - They become self-documenting

Code Comments (2)

- For the exercises, use the following guidelines
  - Always use block comments for method/class documentation
    - /* ... */ and not //
  - Doxygen is a tool for generating code documentation from C++ source code http://www.stack.nl/~dimitri/doxygen/index.html
  - Use annotations to give structure to comments, like Java.
    - E.g. @param and @return
  - E.g.:

```
/**
* Registers the text to display in a tool tip. The text
* displays when the cursor lingers over the component.
*
* @param text The string to display.
* @return true if successful, false otherwise
*/
bool setToolTipText(std::string text) {
```

Code Comments (3)

- Remember that comments need to be maintained
  - When code changes, update comments

- Add comments within the method body when necessary
  - Add comments for complicated pieces of code
  - Example: What does this code do?

```
r = n / 2;
while ( abs( r - (n/r) ) > t ) {
        r = 0.5 * ( r + (n/r) );
}

System.out.println( "r = " + r );
```

```
// square root of n with Newton-Raphson
// approximation
r = n / 2;
while ( abs( r - (n/r) ) > t ) {
        r = 0.5 * ( r + (n/r) );
}

System.out.println( "r = " + r );
```

Important Note

- You need to complete the assignment on your own.
- When you have questions about the assignment, you may
  - discuss them with your classmates
  - post them in the *moodle* discussion forum
  - make an appointment for office hours
- Sharing your solutions with others will result in zero points!
- Copying others' solutions will result in zero points!
- **Do submit your solutions in time!**

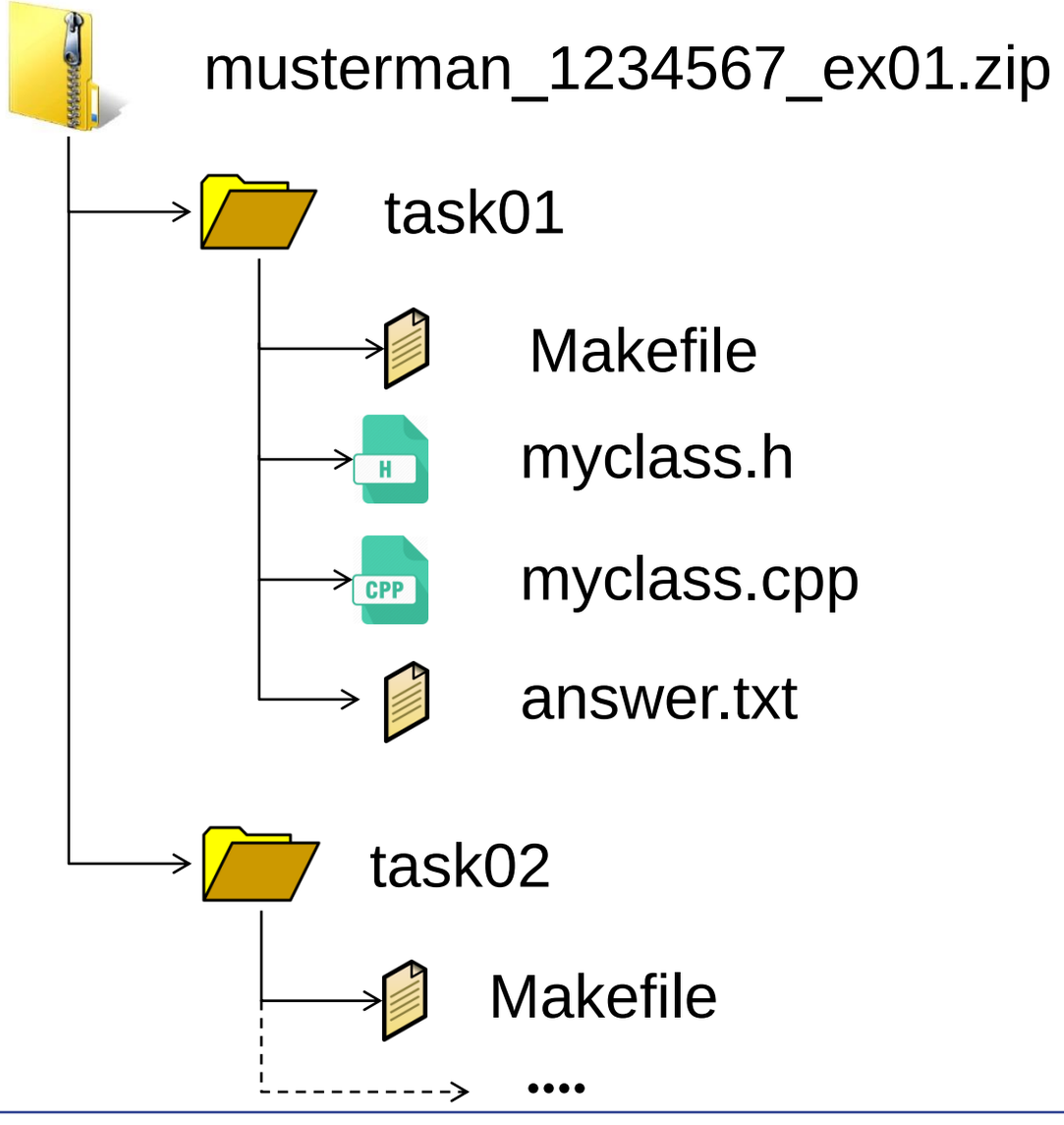
Submission Guidelines!!!

- Your solutions must adhere to the following guidelines
  - You MUST use zip format
    - (not .rar, .tar, .tar.gz or anything else)
  - All solutions must be named: <nachname>_<matrikel>_ex<##>.zip
    - e.g. musterman_2351234_ex03.zip – and not ..._ex3.zip
  - When this folder is unzipped, the output should be folders for the different tasks
    - Create a task folder for each task in the exercise called: task<##>
    - e.g. task01, task02 - and not task1

Submission Guidelines (3)

- All task folders must contain a Makefile called „Makefile“
  - Makefile will generate the executable from your code
  - The name of the executable will be specified for each task
- Text responses should be in an answer.txt file
  - The file should be in the folder of the task
- Your code should be compilable with gcc / g++
  - You can add compiler flags in your Makefile
    - E.g. Flag telling the compiler to use the C++14 standard
  - Submissions will be tested in the Linux environment
    - NOT the system on which you coded it

Submission Guidelines (3)



Compilation requirement

- Your solution MUST compile by running „`make`“ in the task folder

- Recommended environment is Linux (gcc version 5.4 or higher)
  - If you use Windows, you can use a virtual machine
  - You can still install gcc on your machine/OS - as long as your solution compiles it will be accepted

- If for some reason your code does not compile:
  - Explain properly in the summary what you did, what you expected and what actually happens.
  - Give the error output of gcc, explain what you think it means and how you tried to fix it.

Compilation requirement (2)

- In short, prove that you did all you could to get the program to run and failed.
  - Only then, might you get points for the parts of the code which are ok.

- If you only submit code which does not work with no (or an unsatisfactory) explanation, you **get no** points for the exercise!

- You are advised to always test your solution with `make` before submitting

Compilation requirement (3)

- When you prepare your zip for submission, you can test that it works by going to this link:
  - https://project.nes.uni-due.de/teaching/cpp/
  - NOT a debug tool (Debug the program yourself!)
  - Passing the test does not mean you get full points
  - Copy the link!
- Upload your solution zip and run the test
  - Fix any problems and test again
- If all tests pass, then you can submit your zip
  - Even if some tests fail, you can still submit your solution.
  - As long as there are no problems with the zip format

Makefile (1)

- Just like in Java, the compiler takes source code as input to create an executable
- In C/C++, an example using the gcc compiler:
  - `gcc main.cpp hello.cpp factorial.cpp -o hello`
- But with many files, that can become cumbersome
  - Makefiles to the rescue!!
- A Makefile tells the `make` utility how to compile and link your program
  - It describes dependencies between the files

Makefile (2)

- Consists of rules

```
target: dependencies
→	system command(s)
```

- Each rule describes a target and the file dependencies
- Target is a name descriptor used when calling the make utility
- Dependencies are the source code files which are required for building the target
- System commands are instructions for making the target

Makefiles (3)

- Example
- Building our hello program depends on main and hello
- Main and hello also have their own dependencies
- Clean deletes all the build files from the folder

```
all: main.o hello.o
	gcc main.o hello.o -o hello

main.o: main.c
	gcc -c main.c

hello.o: hello.c
	gcc -c hello.c

clean:
	rm *.o hello
```

Makefiles - Targets

- Each target just specifies the commands to be run
- There can be multiple targets in a Makefile
- Targets can depend on other targets
  - Dependent targets will be called first
- “all” is the default target

```
all: main.o hello.o
	gcc main.o hello.o -o hello

main.o: main.c
	gcc -c main.c

hello.o: hello.c
	gcc -c hello.c

clean:
	rm *.o hello
```

Makefiles (5)

- Save the makefile in the task subfolder with the name Makefile
  - The name of the main make task should always be `all`
  - „`all`“ task should create the executable file for the exercise task
    - Name of executable will be specified in the exercise
  - Some exercises may specify multiple build targets

- Always include a clean task in your Makefile
  - Delete the files created by the compilation process
  - Use the linux command `rm`
    - e.g. `rm hello`
    - Windows delete commands will not be supported

Makefiles (6)

- Note that there is a tab before each command in the Makefile
  - `make` will not be happy if the tab is missing

- For the C++ exercises, you might need to use the `g++` compiler instead of gcc
  - You might also need to add some parameters to the commands
  - There will be instructions in the exercise sheets in case complier optimization flags are required
  - By default always use the following standard flags for C/C++
    - gcc : -std=c11
    - g++: -std=c++11

Development Environment

- Linux
  - Installing Linux – Ubuntu is a great choice
    - But feel free to use whatever distro you prefer
  - All tools available or can be easily installed
    - E.g. make, gcc, g++

- Windows
  - Install Linux in a virtual machine e.g. Oracle VirtualBox
    - You can then compile in the Linux environment
    - Set up a shared folder so you can use files in both environments
  - If using Windows 10, you can use Bash for Windows to compile

Development Environment (2)

- Alternative to VM on Windows is to develop fully on Windows
  - Install MinGW tools http://www.mingw.org/
    - By default, installs to C:\MinGW
    - Add C:\MinGW\bin to your path in Environment variables
    - You now have gcc and g++ available to you on the command line
      - However, the make tool has a different name: mingw32-make
- Mac OS X
  - Usually has the build tools installed. If not…
    - Install command line dev tools – you might need to install Xcode
  - Beware: gcc on macOS maybe a clang compiler (which is different)
    - Install and use the actual GNU gcc

Editor

- The exercises are typically small tasks
  - You can use a code editor and compile from command line
  - Some free editors
    - Atom http://atom.io
    - Notepad++ https://notepad-plus-plus.org
    - There are other free/paid editors available

- If however, you prefer an IDE, there are options
  - Qt Creator
  - Visual Studio – free for students
  - JetBrains CLion – free for students

Exercise Sessions

- First hour is presentation of your solutions
  - Be ready to answer some questions about your code and the topic of the exercise
  - Bring along student ID
  - Last 30 minutes – discuss the solution of the previous assignment

- If you cannot make it to exercise
  - Option to present at another time (by appointment)
  - Write me an email **BEFORE** the exercise session requesting appointment
  - Only for special cases, should not be the norm

Exercise Sessions (2)

- Today, exercise 0 will go online
  - Designed to test understanding of the submission guidelines
  - Test setup of programming in environment and VM (if needed)
  - No points will be awarded for this, only pass/fail
  - Deadline: Tuesday, 22.10.2019 @ 23:55H

- Next exercise session on Thursday, 20.10.2019

Access to the Computer Pool

- Your own PC is recommended for doing the assignments
- You can also use the public PC pools of the university

Work after this class

- Add lecture „Programmieren in C/C++” in moodle2
  - access key: ***CPPWT1920***

- For the rest: see lecture slides!

See you next week!