Lesson 1: Introduction
Introduction

- Programming in C/C++ (EE3490E)
- Lectures + Assignments + Handouts (0.3) + Final exam (0.7)
- Compiler: MS Visual C++
- Textbooks:
- Website:
  - [http://mica.edu.vn/perso/kiendt/EE3490E](http://mica.edu.vn/perso/kiendt/EE3490E)
Contents

- Introduction (1 week)
  - The development of programming languages
  - Introduction to C and C++

- Structural programming (7 weeks)
  - Program analysis and design
  - Variables, data, types, operators
  - Control structures
  - Memory management
  - Functions, libraries
  - Data structures
  - Input/output programming
Contents (cont.)

- Object-oriented programming - OOP (3 weeks)
  - Introduction to OOP
  - Classes and objects
  - Inheritance
  - Data structures with OOP
  - Input/output programming with OOP

- Generic programming (2 weeks)
  - Function and class templates
  - Standard template library (STL)
Overview
Programs and programming

- Program: a set of instructions executed by computer
- Software: a collection of programs
- Operating systems (OS):
  - is a software
  - helps users to manipulate the computer
  - manages resources (CPU, memory, soft/hardware,…)
  - runs programs
- Algorithm: an ordered chain of instructions that solves a certain problem
- Compiler: a special program that converts source code into computer-executable binary code
Instruction code and data

- Computers understand only binary codes (bytes, bits)
  - An instruction can have the code of 01000001
  - Code of letter ‘A’ is also 01000001
  - Code of number 65 is 01000001 too

- How can computers distinguishes what 01000001 represents:
  - Depending on what the instruction being executed is
  - Programmers must know what the memory he is working on contains

- Internal memory of computer contains both data and programs’ machine codes
Programming languages

- Are languages designed by human, and used to create computer programs

Development history:
- Machine code: write binary codes directly, no need to compile, and code depending on the host CPU
- 2nd generation (assembly): need to compile, more comprehensible, still depending on CPU but less
- 3rd generation (structural): with control structures, rich data type, encapsulation. Ex: Fortran, C, C++, Basic, Pascal, COBOL,…
- 4th generation: increased effectiveness but less errors, syntax closed to speaking languages. Ex: SQL, LabVIEW, ColdFusion,…
Software development process

- Software life cycle:
  - Problem definition and analysis
  - Software design (functions and algorithms)
  - Implementation (coding)
  - Maintenance, upgrade
  - Obsolete

- Test: examination process to verify the correctness of program functioning

- Debugging: a process to find the origins of program errors and to fix them
Program errors

- Syntax errors (or compile errors):
  - Produced when programmer do not follow the syntax of the PL
  - Detected by the compiler
  - Attn: sometimes not detected but misinterpreted as another syntax

- Runtime errors:
  - Produced when program violates certain conditions
  - Detected at runtime

- Logical errors:
  - Originated from wrong algorithms
  - Undetectable by computer, but by users
Introduction to C and C++
History of C language

- Born in 1970s, related closely to the development of Unix
- Author: Dennis Ritchie
- Goals:
  - Efficiency
  - Ability to access hardware at low level
  - Structural language (to replace programming in assembly)
- C is in the middle of low level languages…
  - Able to access memory directly and naturally
  - Condensed syntax, few keywords
- … and high level languages
  - Independent of CPU and hardware
  - Control structures, functions, encapsulation
  - Type checking
History of C++ language

- Born in 1979 by extending C language
- Author: Bjarne Stroustrup
- Goals:
  - Add more functionality
  - Overcome some drawbacks of C language
- Added features compared to C:
  - Object-oriented programming (OOP)
  - Generic programming (template)
  - Many small features which improve the flexibility (bool type, variables declared everywhere, strong typed, function overload, namespace, exception handling,...)
Compiling a C/C++ program

- Compilation: is the process of converting a source code (written by human) into a program in the form of machine codes that can be executed.

Source file: file1.c
Compiler
Object code: file1.obj

Source file: file2.c
Compiler
Object code: file2.obj

Libraries
Linker
Executable program: a.exe
Compiling a C/C++ program (cont.)

- Benefits of compiling each source file individually:
  - Easy to divide and manage parts of the program
  - Only need to modify the involved files when something need to change
    → shorten maintenance, update time
  - Only need to recompile modified files instead of everything
    → shorten compilation time

- Modern compilers are able to optimize compiled codes (both data and instructions)

- Some popular compilers: MS Visual C++, gcc/g++, clang, Intel C++ Compiler, Watcom C/C++,...
Why C/C++?

Advantages:
- Efficiency
- Flexibility, highly customizable
- Widely supported
- by many environments and platforms
- with many libraries and tools

Drawbacks:
- The language is [too] complicated
- Difficulty in controlling errors, compared to high level languages (Java, .NET, script,…), especially in using pointers
Some points about C/C++ languages

- Are case sensitive: int, Int, INT are totally different
- Use ; to separate simple instructions
- Use { … } to define one block of instructions
- Do not allow using keywords (void, int, char, struct, const,…) to name variables/constants/functions…
- Within a block without control structures (if, for, while,…), instructions are executed from top to bottom
- Comments:
  - in C: with /* … */
  - in C++: additionally with // for commenting one lines
Introduction to MS Visual C++

- Is an integrated development environment (IDE), which helps:
  - Writing programs
  - Compiling
  - Debugging

- Do examples…