McCormick

Northwestern Engineering

Department of Electrical Engineering and Computer Science

EECS 211(B) – Object-Oriented Programming with C++

Winter 2012 Quarter

Instructor: Goce Trajcevski

Contact: g-trajcevski@northwestern.edu

Class meets: Mon. (Tue.) Wed., Fri. 3:00-3:50PM – at LR2, Tech.

Office hours: Mon/Wed/Fri 2:00-3:00 (or, by appointment)

Detailed TA info: TBA

I. **Course description**: The main objective of this course is to expose the students to the foundations of the Object-Oriented programming model and make them proficient in the C++ language, as a particular example of the languages from that paradigm.
The course will attempt to strike a balance between the two fundamental aspects of studying any field: **– *depth*,** via studying both the syntax and semantics of the important programming concepts, along with providing programming assignments and tests related to them; and **– *breadth***, bringing to awareness the "bigger picture" of good programming style and tying it with problem-solving, along with designing relevant test-cases and developing a set of broader-skills to come handy in the subsequent courses focusing on software design/engineering.
Part of the course will provide a brief introduction to C/C++ programming in Unix/Linux environments, thereby introducing the students to some aspects that will be subsequently studied in greater detail in courses like introduction to systems programming, operating systems, networking, etc.

II. **Linguistic aspects:** Building upon C as its foundation, C++ offered two major improvements in terms of the syntax, the second which also affects the semantics (i.e., the meaning) of the programs:
   a. Syntactic constructs which "bettered" the C language (e.g., providing operators instead of traditional functions), while staying in the realm of the imperative/procedural programming paradigm.
   b. Syntactic constructs which provided capabilities for writing a program code in which the entities behave in accordance with the Object-Oriented paradigm. Under this paradigm, the main "players" during the execution of a particular program are the *objects*, each of which must be an *instance* of a particular *class* – the encoding of which captures the *state* (captured via *data members*) and the *behavior* (enabled by the *member functions*) of its instances. Programming with classes provides the opportunity to build *highly cohesive* "modules" (which was the abstraction typically used before classes were formally introduced) and a *low coupling* across them. The (objects from different) classes communicate with each other via well-defined collection of "messages" – which amounts to invocation of the respective *member-functions*. One of the main advantages of this paradigm is that structuring the program code in this manner conforms more closely to the entities appearing in many real-world applications which, in turn, makes the overall product development (spanning from requirements specifications, through design, to coding/testing/deployment) much faster. A specific feature that was introduced in C++ to better cater to the Object-Oriented paradigm, and will be covered in this course, is the *inheritance* between classes.

III. Prerequisites: CS 110 or knowledge of any programming language. The course begins with a review of data types and basic flow-of-control, so your prior knowledge of a programming language need not be deep and need not be specific to C/C++.

IV. Required text: "Big *C++"*, by Cay Horstmann and Timothy Budd (publisher: Wiley)

V. Recommended text and/or other materials:

a. "*C++: How to Program",* by Deitel&Deitel (publisher: Pearson (Prentice Hall), any edition after the 5th one);
b. Handouts that will be provided as part of the course.
c. There is a plethora of textbooks covering the major topics which qualify as an introduction to C++ programming. The reasons for selecting the textbooks above include:
   i. The required text is pedagogically easier to follow for a first-time C++ programmer, and can be used as a reference in the subsequent stages of the education;
   ii. The recommended text, with various editions being around since the early 1990s, has an extensive collection of example-codes. Hence, after some initial familiarization, the students should not have any problems using the recommended text as a "quick-lookup" source.

NOTE: There are plenty of examples available on (various forums on) the web that you should feel free to consult.

VI. Course Outcomes: After finishing the course, the students should be able to:

a. Use (most of) the syntactic elements of the C++ programming language in a fluent manner;
b. Develop well organized object-oriented code in C++ code for solving problems of interest in various applications' settings;
c. Provide their code with various test-cases and "guards" for the purpose of preventing unwanted behavior;
d. Understand code of existing applications and be able to add new features, correct existing errors and incorporate their novel solutions. An implicit benefit is increasing the broad flexibility in terms of being comfortable with adjusting to new programming environments.

VII. Tentative Course Outline (NOTE for HRL – this is just a template from my 230 – will change it tomorrow in accordance with the topics listed in the .pdf file of your syllabus!!!!!)

Week1	- Introduction/Motivation/History of programming; - The basic stages of a program "lifetime"; • Visual C++ IDE (Integrated Development Environment); - Basic Data Types and Variables ("touch" of classes);
Week2	- Program Flow • Comparisons, Relational Operators, **`if-then-else`** branching; - Nested Branching; - Boolean type and Operators; - Looping constructs • `while`; `for` and `do` loops
Week 3	- Introduction to Functions and parameters/arguments; - Functions (cont.) • Scopes of Variables and References; - Recursion;

Week4	- Arrays and Vectors; • Strings; - Pointers and Dynamic Memory Allocation; - • **Quiz**
Week5	- Arrays and Pointers; - Streams (Files I/O vs. command line arguments) - Introduction to Classes - Classes and encapsulation; • Object-Oriented paradigm and C++
Week 6	- Data Members and Member Functions - Life-span of objects • constructors and destructors; - Access privileges and friendship - Overloading (functions vs. operators) **Midterm**
Week 7	Nested classes - Inheritance (derived classes) • Inheriting state (data members + access) and behavior (overriding member functions) - Polymorphism and virtual functions - Abstract classes and pure virtual functions
Week 8	- Multiple inheritance in C++ - Templates (functions vs. classes) - Namespaces
Week 9	- Special topics: Exceptions handling (C model vs. C++) - Introduction to Unix/Linux OS (file system; C-based I/O) - The "hidden" arguments of *main* – revisited
Week 10	- OS-issues (cont.) - Porpourri: • Basic features of other Object-Oriented and Object-Based languages: (Java; C#)
Week 11	- **Final exam (material after the midterm only)**

VIII. Grading

Your grades will be based on:

- 2-3 homeworks – 2.8%
- 1 Quiz – 7.2%
- 8 Projects – a 40% (see the corresponding paragraph below)
- Midterm – 25%

- Final – 25%

NOTE: the distribution given above is approximate and may be subject to some *very minor* changes. However, the firm-policy will be announced during the last week of classes (Week #10 of the Winter quarter).

Practice Sessions: Starting with week#3, Tuesday lectures will be specifically allocated to a practice which can have a two-fold nature: (1) going over a collection of exercises that will illustrate the concepts and abstractions introduced in a certain topic of the course; (2) dedicated lab-demonstrations (towards the end of the quarter).

Programming Assignments: There will be eight programming assignments. The first two introduce the students to basic C++ programming and the use of a modern, integrated development environment (IDE). The remaining six assignments form a single medium scale application. Each assignment adds a piece to the overall project. The various assignments in the project cover common programming applications such as character processing, linked data structures, classes and inheritance, etc. Sample applications from previous quarters include internet message passing, computer memory management system, sensor network simulator, and XML parser.

Students generally do the programming assignments on their laptops, although the course does include demonstrations and lectures on using UNIX for programming. A separate document contains information on how to obtain free software for C++ software development.

Awareness, Academic Responsibilities and Closing Remarks:

Please be advised that it is each student's *individual responsibility* to keep him/herself up-to-date with the announcements *made in class*, *distributed via email*, or *otherwise posted*. Although you are encouraged to always discuss class-related issues with your classmates, it is your individual responsibility to ensure that the work is done individually. Example: you are encouraged to discuss the algorithmic aspects of solving a particular programming assignment, and even the high-level design approach. The source code that you will submit, however, must be typed *individually* in its entirety. The policies for cheating are well-defined and there will be no exceptions made for any excuse whatsoever – if caught cheating (e.g., copying on an exam, borrowing someone else's code as well as allowing someone to borrow your code), you will automatically fail the class and face a possible expulsion from the University. In addition, notwithstanding our willingness to be understanding for the students' commitments and time-constraints, please do not attempt to obtain an incomplete grade for the course, based solely on your poor performance – it is against the University regulations.

Lastly, please note that a substantial part of your grade is based on the programming assignments. Hence, in addition to keeping yourself up-to-date with the materials lectured, you should also make it a habit to start working on the programs as early as possible. You should not allow yourself to fall behind with the topics, as the new ones will be building incrementally upon the older ones, and it will be very hard to catch up. Plan your time wisely and good luck.