ISE103 – Programming II

Dr. Sanem Sarıel Talay

General Information

- Course hours: Wed/Fri 09.30am-12.30pm
- Lab hours: Fri 09.30am/01.30pm
- Course web page:
  http://www.ninova.itu.edu.tr
- Lecturers: Dr. Sanem Sariel-Talay and Dr. Cüneyd Tantug

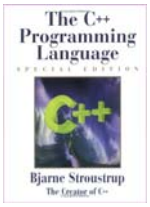
Grading Scheme

- Assignments (20% total)
- 2 Midterm exams (20%, 20%)
- Final Exam (40%)
- You must follow the official Homework Guidelines:
  - http://www.ce.itu.edu.tr/lisans/kilavuz.html

Assignment Submission Policy

Academic dishonesty including but not limited to cheating, plagiarism, collaboration is unacceptable and subject to disciplinary actions. Any student found guilty will have grade F. Assignments are due in class on the due date. Late assignments will generally not be accepted. Any exception must be approved. Approved late assignments are subject to a grade penalty.

References



Most of these lectures are based on

Asst.Prof. Dr. Feza Buzluca's Lecture Notes

Purpose of the course

- To introduce several programming paradigms including **Object-Oriented Programming, Generic Programming, Design Patterns**
- To show how to use these programming schemes with the C++ programming language to build "**good**" programs.

Course Outline

	Topic
1	Introduction to Object Oriented Programming
2	C++
3	Classes and Objects
4	Constructors and Destructors
5	Operator Overloading
6	Inheritance
7	Pointers to Objects
8	Polymorphism
9	The Unified Modeling Language (UML)
10	Exceptions
11	Templates
12	The Standard Template Library – STL

ISE103 – Programming II

Introduction

Software

- Computer Software is the product that software engineers design and build.
- It encompasses
  - programs that execute within a computer of any size and architecture,
  - documents that encompass hard-copy and virtual forms,
  - data that combine numbers and text but also includes representations of pictorial, video and audio information.

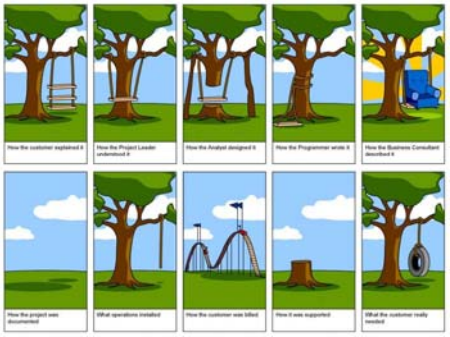
Need for a good programming method

- Common problems:
  - Why does it take so long?
  - Why are development costs so high?
  - Why can't find all faults before delivery?
  - Why can't we measure the development?
- **Financial Times (27/08/2002) P. 12;** The National Institute of Standards and Technology (NIST) recently reported that **software bugs cost American companies approximately $60 billion in 2001**, while Bill Guttmann of Carnegie-Mellon University's Sustainable Computing Consortium pegs the **real cost closer to three or four times that**.

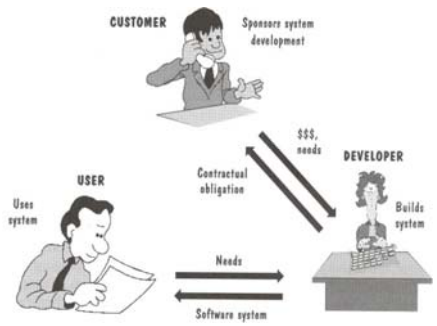
What went wrong?

- We build systems like the Wright brothers built airplanes: build the whole thing, push it off the cliff, let it crash, and start over again.
- But it is not true
- The Wright brothers practiced engineering in its best possible sense: They used theory, experimentation, guesses, and feedback.

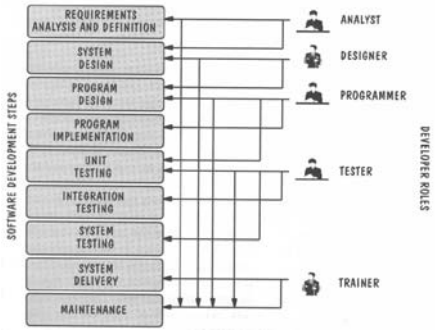
What went wrong? (con't)



Customer-User-Developer



Development Team



Software Life Cycle

- Requirements Phase
- Specification Phase
- Design Phase
- Implementation Phase
- Integration Phase
- Maintenance Phase
- Retirement Phase

Requirements Phase

- Defining constraints
  - Functions
  - Due dates
  - Costs
  - Reliability
  - Size
- Types
  - Functional
  - Non-Functional

Specification Phase

- Documentation of requirements
  - Inputs & Outputs
  - Formal
  - Understandable for user & developer
  - Usually functional requirements. (what to do)
  - Base for testing & maintenance
- The contract between customer & developer

Design Phase

- Defining Internal structure (how to do)
- Has some levels (or types of docs)
  - Architectural design
  - Detailed design
  - ...

- Important
  - To backtrack the aims of decisions
  - To easily maintain

Implementation and Integration Phases

- Implementation phase: Simply coding
  - Unit tests
    - For verification
- Combining modules
- System tests
  - For validation
- Quality tests

Maintenance Phase

- Corrective
- Enhancement
  - Perfective
  - Adaptive
- Usually maintainers are not the same people with developers.
- The only input is (in general) the source code of the software

Retirement Phase

- When the cost of maintenance is not effective.
  - Changes are so drastic, that the software should be redesigned.
  - So many changes may have been made.
  - The update frequency of docs is not enough.
  - The hardware (or OS) will be changed.

Why Object Technology

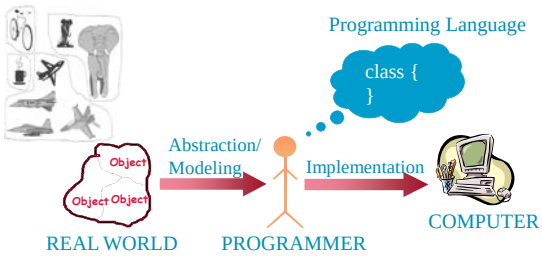
Expectations are,

- Reducing the effort, complexity, and cost of development and maintenance of software systems.
- Reducing the time to adapt an existing system (quicker reaction to changes in the business environment): Flexibility, reusability.
- Increasing the reliability of the system.

What is Programming?

- A programming language provides a way to express concepts.
- Program development involves creating models of real world situations and building computer programs based on these models.
- Computer programs describe the method of implementing the model.
- Computer programs may contain computer world representations of the things that constitute the solutions of real world problems.

What is Programming? (con't)



- If successful, this medium of expression (the object-oriented way) will be significantly easier, more flexible, and efficient than the alternatives as problems grow larger and more complex.

Why C++

- C++ supports writing high quality programs (supports OO)
  - Bjarne Stroustrup, in 1979 at Bell Labs (C with classes)
- It enables both OO and non-OOprogramming (advantage?)
- Compatible with C Programming Language
- Standardized language (1998- ANSI/ISO, 2003- ISO/IEC 14882)
  - C++0x is under development

Why C++, con't

Several Application Domains:

- Systems programming: Operating systems, device drivers. Here, direct manipulation of hardware under real-time constraints are important.
- Banking, trading, insurance: Maintainability, ease of extension, ease of testing and reliability is important.
- Robotic and embedded system programming
- Computer Communication Programs

Learning C++

- Many sytax and grammar rules
- Learning how to write "good programs"
- Focusing on concepts and not get lost in language-technical details
- Paying attention to design techniques rather than details
- Building an effective programming scheme
- Practicing, practicing and practicing!
- Consequently, new and better ways of building systems

Software Quality Metrics – User

- A program must do its job correctly. It must be useful and usable.
- A program must perform as fast as necessary (Real-time constraints).
- A program must not waste system resources (processor time, memory, disk capacity, network capacity) too much.
- It must be reliable.
- It must be easy to update the program.
- A good software must have sufficient documentation (users manual).

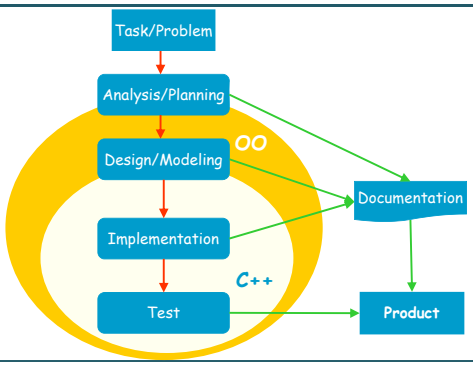
Software Quality Metrics – SW Developer

- Source code must be readable and understandable
- It must be easy to maintain and update the program
- A program must consist of independent modules
- An error may not affect other parts of a program (Locality of errors)
- Modules of the program must be reusable in other projects
- A software project must meet its deadline
- Good software must have sufficient documentation

Software Quality Metrics - OOP

- OOP techniques ensure high-quality programs
- While designing and coding a program, these quality metrics must be always considered

Software Development Process



Software Development Process, con't

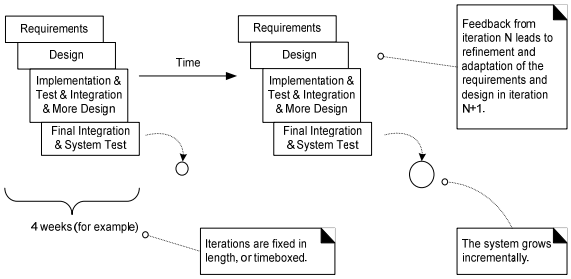
- Analysis: Understanding requirements. They may change during (or after) development of the system! Building the programming team.
- Design: Identifying the key concepts involved in a solution and creation of the models. This stage has a strong effect on the quality of the software. Therefore, before the coding, verification of the created model must be done.
  Design process is connected with the programming scheme. Here, our design style is object-oriented.
- Coding: The solution (model) is expressed in a program.
  In this course we will use C++.
- Documentation: Each phase of a software project must be clearly explained. A users manual should be also written.
- Test: the behavior of the program for possible inputs must be examined.

- These steps are important **design principles** and **design patterns**, which help us developing high-quality software. The Unified Modeling Language (**UML**) is useful to express the model.

Unified Process (UP)

- The UP promotes several best practices.
- Iterative
- Incremental
- Risk-driven

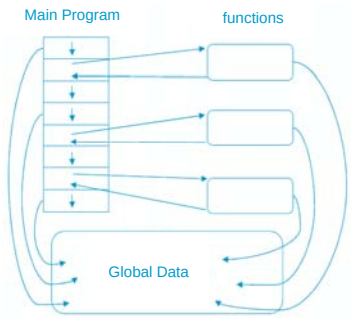
Unified Process (UP), con't



Procedural Programming

- Pascal, C, Basic, Fortran and similar traditional languages are procedural
  - Each statement tells the computer to do something
- The emphasis is on doing things
  - Functions
- A program is divided into functions
  - Each function has a clearly defined purpose and interface

Procedural Programming, con't



Problems with Procedural Programming

- Data Is undervalued
- Data is, after all, the reason for a program's existence. The important parts of a program are not functions that display the data or functions that checks for correct input; they are data
- Procedural programs don't model the real world very well. The real world does not consist of functions
- Global data can be corrupted by functions that have no business changing it
- To add new data items, all the functions that access the data must be modified so that they can also access these new items
- Creating new data types is difficult

Besides...

- It is also possible to write good programs by using procedural programming (C programs).
- But object-oriented programming offers programmers many advantages, to enable them to write high-quality programs.

Object-Oriented Programming

The fundamental idea behind object-oriented programming:

- The real world consists of objects. Computer programs may contain computer world representations of the things (objects) that constitute the solutions of real world problems.
- Real world objects have two parts:
  - *Properties* (or *state*: characteristics that can change),
  - *Behavior* (or *abilities*: things they can do).
- To solve a programming problem in an object-oriented language, the programmer no longer asks *how the problem will be divided into functions*, but **how it will be divided into objects**.
- The emphasis is on **data**

Object-Oriented Programming, con't

- What kinds of things become objects in object-oriented programs?
  - Human entities: Employees, customers, salespeople, worker, manager
  - Graphics program: Point, line, square, circle, ...
  - Mathematics: Complex numbers, matrix
  - Computer user environment: Windows, menus, buttons
  - Data-storage constructs: Customized arrays, stacks, linked lists

OOP: Encapsulation and Data Hiding

- Thinking in terms of objects rather than functions
- Close match between objects in the programming sense and objects in the real world
- both data and the functions that operate on that data are combined into a single program entity
- Data represent the properties (state), and functions represent the behavior of an object. Data and its functions are said to be *encapsulated* into a single entity
- An object's functions, called member functions in C++ typically provide the only way to access its data. The data is *hidden*, so it is safe from accidental alteration.

OOP : Encapsulation and Data Hiding, con't

- ***Encapsulation*** and ***data hiding*** are key terms in the description of object-oriented languages.
- If you want to modify the data in an object, you know exactly what functions interact with it
  - The member functions in the object.
  - No other functions can access the data: This simplifies writing, debugging, and maintaining the program.

Example: A Point on the plane

- A Point on a plane has two properties; x-y coordinates.
- Abilities (behavior) of a Point are, moving on the plane, appearing on the screen and disappearing.
- A model for 2 dimensional points with the following parts:
  - Two integer variables (**x , y**) to represent x and y coordinates
  - A function to move the point: **move**
  - A function to print the point on the screen: **print**
  - A function to hide the point: **hide**

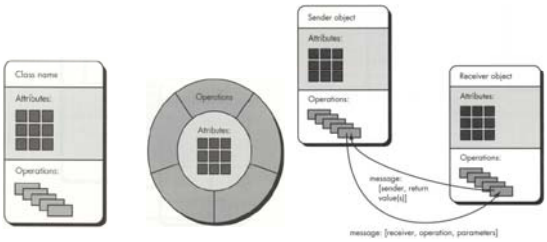
Example: A Point on the plane

- Once the model has been built and tested, it is possible to create many objects of this model, in the main program.

```
Point point1, point2, point3;
    :
point1.move(50,30);
point1.print();
```


Object Model

- A C++ program typically consists of a number of objects that communicate with each other by calling one another's member functions.



OOP vs. Procedural Programming

- Procedural languages still requires you to think in terms of the structure of the computer rather than the structure of the problem you are trying to solve.
- The programmer must establish the association between the machine model and the model of the problem that is actually being solved.
- The effort required to perform this mapping produces programs that are difficult to write and expensive to maintain. Because the real world thing and their models on the computer are quite different

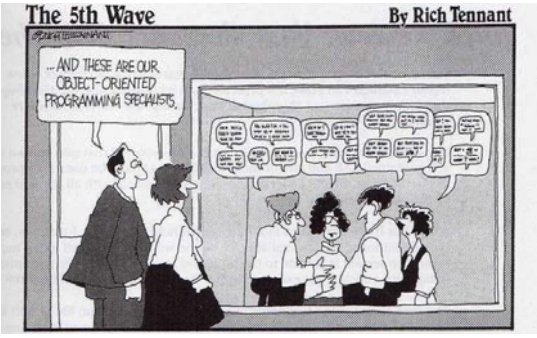
Example: Procedural Programming, con't

- Real world thing: student
- Computer model: char *, int, float ...
- It is said that the C language is closer to the computer than the problem.

OOP *vs*. Procedural Programming, con't

- The OO approach provides tools for the programmer to represent elements in the problem space
- Objects are both in the problem space and the solution
- The OO programs are easy to update by adding new types of objects
- OOP allows you to describe the problem in terms of the problem, rather than in terms of the computer where the solution will run.

OOP *vs*. Procedural Programming, con't

- Benefits of the object-oriented programming:
  - Readability
  - Understandability
  - Low probability of errors
  - Maintenance
  - Reusability
  - Teamwork

OOP everywhere... ☺