

What you should have learnt

Lecture 1

Programming Methodology and Object Oriented Programming (0396A)

We will start the course lightly, just to tell you what we will be doing in the rest of the year.

On the other hand, this is a very rich course, and you will probably be kept rather busy.

In the first semester, we get started doing programming.

We have the basic idea about programming. We know about...

- Programming process, compiler, debugger;
- Constants, variables and expressions;
- Simple statements, conditionals, loops;
- Functions, recursion;
- Pointers, classes, templates;
- Top-down design technique, readability.

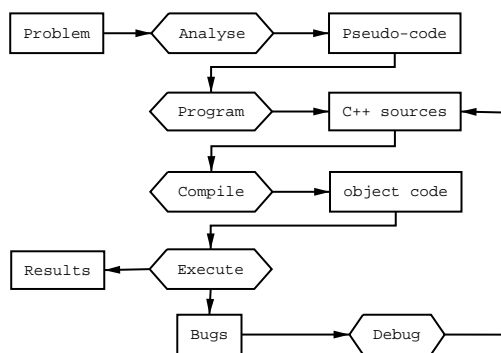
This already allow you to do any sort of programming. You are all experts in programming.

PMOOP(0396A)

PMOOP(0396A)-1.1

Programming model

So far we have been basically doing the following:



PMOOP(0396A)-1.2

PMOOP(0396A)-1.3

Problem of the model

The problem is, when the program becomes larger, the model sucks.

- Program files becomes very big, very difficult to edit.
- Collabrataion is very difficult, or even impossible.
- Any small change of the program lead to complete recompilation.
- To extend the program, it must be edited and recompiled.
- Old versions of the program disappears once a new version is made.
- It is difficult to reuse the code from other projects.
- Some program need to deal with concurrency and other system issues like other users, and pose difficulty when working with pseudo-code.

This course

In this course, you will learn

- Practical techniques in managing a medium size programming project.
- Mechanisms and designs that is needed if you want your program to be easily modifiable, or even better, configurable, to adapt to a rapidly changing environment.
- Mechanisms and designs that enables code of a project to be reused in other projects.
- Interactions of your program with the computing system, precautions that need to be taken for any real programming project.

In a sense, the course is an advanced programming course, extending your knowledge on programming.

What this course is not

It is also important to know what we will not do in this course. It is not to say that they are not important, but they are covered in other courses:

- Algorithmic issues. For such knowledge, take the "Design and Analysis of Algorithms" course. If you're interested, join our programming contest team!
- Software design process. You're doing Dr. Tse's course already. We will focus only on the programming part of the process.
- Java. This is an advanced programming course, not a programming language course. The Principle of Programming Language course will introduce the language. Or just pick a Java book instead.
- Network programming. Instead, take the Network course.
- The compilation process, or code optimization. Take the compiler course, instead.

PMOOP(0396A)-1.4

PMOOP(0396A)-1.5

Course information

Lecturer

Mr. (later Dr.) Isaac K. K. To, kkto@csis.hku.hk, CYC 407
Office hour: Tuesday 4:00–6:00pm

Tutor

Chen Yong, yongchen@csis.hku.hk, xxx
Consultation hour: Thursday 4:00–6:00pm, CYC 419

Course

News: hku.csis.CSIS0396A
Web: <http://www.csis.hku.hk/~c0396a>
E-Mail: c0396a@csis.hku.hk

Reference (All from Addison Wesley)

Design Patterns. Gamma, Helm, Johnson, Vlissides.
Designing and Coding Reusable C++. Carroll, Ellis.
The C++ Programming Language. Bjarne Stroustrup.
A lot of *manpages*, *info pages* and *web sites*.

PMOOP(0396A)-1.6

PMOOP(0396A)-1.7

Course administration

Lectures

Not very formal. But be considerate: if you don't want to listen, just don't attend it. There is no roll-call, anyway.

Tutorials

On demand, usually after assignments are released.

Assessment

3-4 assignments (1 written + 2-3 written/programming mixed). 25%
One 1-hour quiz. 15%.
Exam. 60%.

Plagiarism

Will use the normal departmental policy (i.e., 1st offense—deduct half of everything except exam; 2nd offense—refer to discipline committee). No mercy.

Lecture conduct

Many times students ask: “How much of the lecture should I understand right in the lecture hall?”

- Answer: everything I present to you.
- If you have anything that you don't understand, just ask right away. Don't hesitate: your problem is the problem of most of the class.

I have an annoying tendency to forget pausing for a while after a difficult concept is presented, and thus out-pace everybody.

- If you need some more time to digest, just ask me to pause.
- Remember that if you succeed in slowing me down so that I teach one less topic, there will be one less topic in the exam.

As long as the question is at least remotely reasonable, just ask.

PMOOP(0396A)-1.8

PMOOP(0396A)-1.9

Pre-requisites of the course

- As an advanced programming course, I really expect you to be expert in programming. If you failed 1117, don't bother to take this course. After all, 0396A is not a pre-requisite of any other course.
- This will be a rather heavy course (just like every other courses that I teach). Be sure to schedule enough time for the course, or you will probably get lost in the middle of it.
- The reference books are all **very** interesting. Do read them. Especially the book on patterns.
- The course will assume you have a running Linux system that is reasonably recent. If you don't have it, start doing installation as soon as possible.
- We assume that you know how to read man pages and info pages.

Course outline

- Week 1: Introduction, Audio basics, WAV files.
- Week 2: Encapsulation, Exception handling, Multiple object files.
- Week 3: Makefile and dependencies, OSS library, Abstract Data Types.
- Week 4: External libraries, Libaudio, Proxy.
- Week 5: Static and dynamic library, Extension through module.
- Week 6: Class design.
- Week 7: Decorator, Object Oriented design, Patterns, Composite.
- Week 8: Scripting, Extension through scripting.
- Week 9: Event driven programming, GUI basics and implementation.
- Week 10: Program performance tuning.
- Week 11: Unix concepts: users, files and process, security models.
- Week 12: Security holes and exploits.
- Week 13: Multi-threading.

PMOOP(0396A)-1.10

PMOOP(0396A)

Lecture 2

Audio application: our example project

We will use a very simple-minded application to concretely illustrate concepts taught during the course, which generate sound in various ways.

In this hour we will examine the prerequisites needed to appreciate the example.

Our concrete project: a disclaimer

- We will soon learn a number of ideas that are useful for the development of any software applications.
- To illustrate such ideas, we want a real project that need these ideas.
- Unluckily, such concrete examples tend to be **very large projects**, too large to be presented in first-year lectures.
- As a compromise, we choose to play with a **small project** that is just large enough to make such ideas meaningful, and then **over-design** it.
- Thus the ideas do not really gain a lot for our project. In real situations, we will use simpler design in such small projects.
- Remember that this is an unrealistically small project. Always assess the ideas by imagining what will happen in a larger project.
- Also, imagine that we want to reuse our code in other projects.

PMOOP(0396A)-2.1

Our project: making sound

Basically, what we want is rather simple: we want to generate some sound, given some notes.

- What it mean by to “generate some sound”? Make the computer play out some music? Or to make a WAV (or some other format) file for play later?
- What it mean by “some note”? A text file containing things like 123? Or a MIDI file? Or an interface that allow people to drag some notes into the staff-lines?
- What it mean by “to generate”? Should we just make a perfect sine or square wave? Or try to make wave that sound like a piano or flute?
- What should happen between notes? Silence? Make the note sound softer near the end of note? “Join” the two notes somehow?

PMOOP(0396A)-2.2

The real complication

The answer to each of the questions to the last slide is:

We want all, and even things that we haven't listed.

That single part makes our “small” project “large” enough for some design.

- Realistically, we usually want to answer like this. We usually do not want our program to be limited to particular input format or output format.
- But as we will see, this will add quite a bit of complications into our program.
- In practice, we usually make this answer only after a lot of thoughts that the generality will really be used.

PMOOP(0396A)-2.3

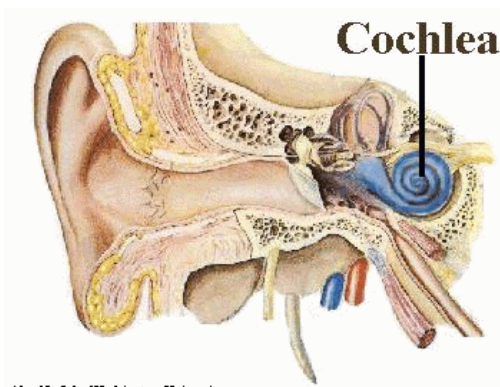
The origin of sound

Here's a small recapitulation of the Physics and Biology of sound.

- A **sound source**, e.g., a piano fork, vibrates for some reason, e.g., somebody hit the key associated to it.
- The vibration push the air molecules around back and forth.
- With an increased/decreased pressure, the air molecules push even further air molecules back and forth.
- Eventually, with an increased/decreased pressure, the air molecules around our ear push the eardrum so that it vibrates.
- The vibration of the eardrum causes resonance in some particular parts of the *cochlea*, depending on the frequency of the vibration.
- Our auditory nerve is activated by such resonance and send signals to the brain to indicate the frequency spectrum we hear.

PMOOP(0396A)-2.4

If you can't remember...



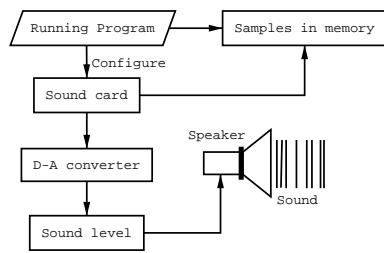
Alec N. Sait, Washington University

PMOOP(0396A)-2.5

PMOOP(0396A)-2.5

How computer makes sound

To make sound, our computer act as a sound source: some part vibrates.



The program stores sound data in RAM and ask the sound card to play it.

The sound card read the data in RAM, convert it to analog signal and put them into the speaker for amplification and sound generation.

PMOOP(0396A)-2.6

What is sound data

It is impossible to represent sound completely accurately:

- There are infinitely many time moments between any interval.
- At each moment we need to represent the position of the drum of the speaker, and again there are infinitely many positions.

So we do the thing that is next-best: to represent sound approximately:

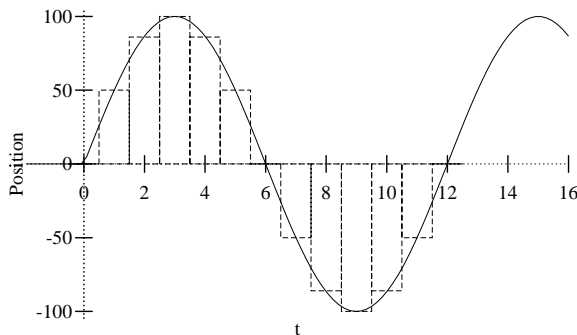
- We represent a sample every some amount of time, say 1ms.
- For each sample, we only allow a fixed number of drum positions.

Therefore, a piece of sound data may look like 0, 50, 86, 100, 86, 50, 0, -50, -86, -100, -86, ... This means some sound if we interpret the numbers as pressure levels at time 0ms, 0.5ms, 1ms, ..., 5ms, ...

A WAV file contains exactly these information to represent sound.

PMOOP(0396A)-2.7

In a picture



PMOOP(0396A)-2.8

Other complications

Not all sound are the same.

- Number of channels: some sound are mono, some are stereo, and yet others contains more channels.
- Sample rate: the frequency at which we take a sample. Typical sampling rates are 8000 Hz (phone line quality), 22050 Hz (tape quality), 44100 Hz (CD quality) and 48000 Hz (DAT quality). This dictates the highest frequency that can be represented.
- Bits per sample: the number of bits we allocate for each sample. Typically either 8-bit or 16-bit. Control the ratio between the loudest sound and the softest sound.

To cater for all these cases seems to be difficult, but is in fact rather easy.

PMOOP(0396A)-2.9

Our first target

However, to begin with, we will first make a very simple program, with the following simplifying assumptions:

- The sound to be generated is always 1 kHz sine wave of fixed amplitude for 1 second, so there is no input.
- The output is a WAV file.
- There would be 2 channels for the sound, the sampling rate will be 22050 Hz, and the samples are of 16-bits.
- No special processing is done at the ends of the sound.
- Running time is not an issue, so we can do whatever complicated things.

We will add more details into the program once we understand adequately how to make simple sound.

PMOOP(0396A)-2.10

Conclusion and Exercise

- For computers, sound is basically a sequence of integer positions to place the speaker drum at.
- Sound quality is controlled by parameters including number of channels, number of bits per sample and sampling rate.
- A WAV file contains exactly these information for sound generation.
- We will write a short program to generate some boring sound in the next lecture.

Exercise:

- Find a formula to convert time in seconds to speaker position for the sound we have specified (1kHz sine wave of unit amplitude), and generalize this to all amplitudes and frequencies.
- Learn how to read info pages (Type ^Hih in Emacs).

PMOOP(0396A)-2.11