Comp 170 Object Oriented Programming

A FIRST SERIOUS PROGRAMMING CLASS
DR. WILLIAM L HONIG

Two Level Thinking for Programmers

<table>
<thead>
<tr>
<th>CONCEPTS</th>
<th>DETAILS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables</td>
<td>All those funny symbols</td>
</tr>
<tr>
<td>Boolean Logic</td>
<td>- ( ) [ ] , ;</td>
</tr>
<tr>
<td>Statements and Expressions</td>
<td>- + - / * %</td>
</tr>
<tr>
<td>Types and Declarations (and Type Checking)</td>
<td>-</td>
</tr>
<tr>
<td>Control Flow; Control Structures (Conditionals, Loops)</td>
<td>All the gibberish</td>
</tr>
<tr>
<td>Functions / Methods / Procedures</td>
<td>&lt; void</td>
</tr>
<tr>
<td>Class and Object (separate and related)</td>
<td>public, private</td>
</tr>
<tr>
<td>Memory Allocation and Usage</td>
<td>- if while do return break</td>
</tr>
</tbody>
</table>

A Lot of Things We Won’t Do (at first)

1. Use GUI such as Windows
2. Do OOP, except in passing
3. Write large programs

So programs may be a bit BORING, for a while
My Program Won’t Work !?#$?

Three distinct kinds of errors while programming:

1. **Syntax Error (Compile Error)**
   - Red error messages in Xamarin; may give exact information or may be misleading, confusing. Often only the first one counts.
   - Approach: Compile often (at least every method); fix the first one; look above the error message too.
   - Have some templates (like, click of a button with parameters and that returns value, etc.)

2. **Runtime Error (Bomb, Exception…)**
   - Approach: What was the program doing? Is there a chance it always happens, or only on some inputs?
   - Add debugging statements to see how far the program ran before the error.

3. **Logic Error (Runs and does the wrong thing)**
   - Approach: Play computer and follow through the program. Where did it go wrong (parameters, logic…)?

KEY Ability to learn = What Kind of Error and How To Fix It!

Why C# ??

1. Microsoft creation
2. In the C++, Java evolution
3. A real language!
4. Real IDE tools

What we learn will apply to C++, Java, many other languages

Some Links to Microsoft Documentation of Interest

2. C# Keywords: [http://msdn.microsoft.com/en-us/library/x53a06bb.aspx](http://msdn.microsoft.com/en-us/library/x53a06bb.aspx)

Good to be able to see the reference to augment the text!
Truth Tables = Basis for EVERYTHING

Why George Boole is famous
True / False On / Off 1 / 0 Hi / Lo
Any calculation can be done with And, Or, Not
Any calculation can be done with only NAND
Transistors are either On or Off
Computers use lots of transistors (to build logic gates)

Conjunctive Disjunctive Normal Form
ANY TRUTH TABLE CAN BE TURNED INTO A LOGIC FORMULA

Mechanical way to convert any truth table into a logic formula using AND, OR and NOT
Conjunction = AND Disjunction = OR

How to do it: and together each row that has a one in the result. If the input is True, use it by itself, if the input is False, put a NOT in front. OR them all together. You look only at the rows of the truth table with a 1 in the result. Ignore those with a 0 in the result.

Example: (A and NOT B and C) or (NOT A and B and NOT C) or (NOT A and B and C)

How Many Truth Tables Are There with 2 Inputs?
GIVEN EVERY POSSIBLE COMBINATION OF INPUTS
Draw a truth table with two inputs A and B
Add every possible output you can think of...
(a bit like program testing, thinking of all the possible values a variable or a return value can take)

HOW MANY CAN YOU WRITE IN C#?
Try it:
Given variables a and b
(What would their type be???)
Make each one a function, give it a name
(Return the result, what type???)
Why worry about programming style?

**MOTIVATION**

Programming languages are big, flexible
- Many options; alternative ways to express concepts
- Not very prescriptive on appearance of program

Without direction, programmers deal with this complexity on their own
- Beginners develop their personal style
- Changes over time as programmer "learns"
- Trial and error process

Opportunity to speed up the learning

Programming is seldom a One Person Task
- Sooner or later, others must read, use, review your code
- Team programming, software reuse, grading…

A shared or common style eases the next person's task

**WHAT DOES IT DO?**

Standard Rules for:
- identifiers
- indenting
- Naming different things to look differently
- Ways to write control structures

Goal: Familiar, easy to read, code that avoids common errors

Use Good Mnemonic Names (1 of 2)

Use names that imply their role in the program
- Use names that are reasonably long and descriptive
- Better than adding separate comments to the code!
- Identical in code size, run time, ...
- It takes time to devise and make use of informative names

Aids readability (self and others)

```
x1 = x2 * x3 + x4 * x5;
x6 = x7 * x1;
x = x1 - x6;
grossPay = (wage * hours) + (overtimeWage * extraHours);
tax = taxRate * grossPay;
netPay = grossPay - tax;
```

Use Good Mnemonic Names (2 of 2)

Use style that helps to identify What is What
- No real cost if done consistently and from the beginning of development
  - Prevents compile errors
  - Reduces need to "look things up"

```
date my_birthday;
if (my_birthday.year() > too_old_year)
  cout << "Too old for this event."
```

```
date myBirthday;
if (myBirthday.Year() > TOO_OLD_YEAR)
  cout << "Too old for this event."
```
Why impose a style guide?
Three Reasons

1. Help programmer avoid common errors
   - Common syntax or semantic errors
   - Experience shows programmers often make this error
2. Make programs more readable or reusable
   - All programmers to do things in similar ways
   - Standard ways to find things in a program
   - Helps original programmer and later "users"
3. Limit flexibility or options to a few well proven ones
   - Select among many ways to do the same thing
   - Experience in real world shows the best approach

Examples
Avoid errors and common problems

```
// Most acceptable forms of for loops
for (i = 0; i <= max_index; ++i)
for (i = 0; i < sizeof(array); ++i)
for (i = max_index; i >= 0; --i)
for (i = max_index; i > 0; --i)
```

You need a really good reason to write anything different!

Top – Down Development

```
PROBLEM APPROACH: MAKE IT EASY TO
SOLVE THE WHOLE THING AT ONCE!

System Thinking:
1. What is everything I need to do?
2. Divide into key chunks
   - Part-name the "hard" parts, e.g., "reverse string"
3. Order the chunks (full program)
   - Create the proper control flow for the whole app
4. Create stubs for the underlying parts
   - They can do stupid things, e.g., always return 1
5. Order the chunks
   - Create the proper control flow for the whole app
```

```
STEPWISE REFINEMENT

Main (Facade pattern)
```

```
Stub
Subroutine
Stub
Subroutine
```

Can the whole program run? Keep it running!
```
```
The 170 Top Ten

Maybe not everything you need to know to ace 170, but key things you must know for success.

We will come back often to this list.

Keep your own checklist of how you are doing!

How can I help?

Comp 170 is a hard course, but there will be lots of help!

You don’t know how to do something until you can teach it!