ECE 462
Object Oriented Programming
using C++ and Java
Objects and Classes

Prof. Sam Midkiff
EE 310
smidkiff@purdue.edu
Course Organization

• Prequisites
  – ECE 264. If you have not taken ECE264, please talk to the instructor.
  – Understand the concept of pointers in C
  – Know how to write and compile C programs in UNIX-based (e.g. Linux or Solaris) machines, using tools & commands such as gcc, gdb, and Makefile
  – I will not emphasize syntax. Instead, we will spend more time on language features and concepts how to design and implement non-trivial programs. Because of this, test will be open book and bring your notes
  – Expect to read and write a lot of code -- programming well requires doing it enough that you think algorithmically, and encoding an algorithm in a programming language is second nature
Textbook

• "Programming with Objects" by Kak, John-Wiley
• source code from the book and errata: https://engineering.purdue.edu/kak/programmingwithobjects/ -- provides many executable examples
• Covers a wide range of topics
• Compares and contrasts C++ and Java
Course material, grades, etc.

• I will put course materials, grades, assignments on my web page.
  – At some point I will get a poll out, and if there is strong support for stuff to be on blackboard I may move in that direction
• Go to the bottom of my home page (google midkiff purdue) and you will see the link there
• Grades will be posted to blackboard
Course Outcomes

• Write object-oriented programs of moderate complexity in Java.
• Write object-oriented programs of moderate complexity in C++.
• Understanding of the concepts of inheritance and polymorphism.
• Use template classes and the STL library in C++.
• Overload operators in C++.
• Incorporate exception handling in object-oriented programs.
• Understand the difference between function overloading and function overriding.
• Write programs with multiple threads and use synchronization among threads.
• Passing all outcomes is a necessary (but not sufficient) condition to receive a passing grade (A - D).
Emergency procedures

• Emergency procedures guide: http://www.purdue.edu/ehps/emergency_preparedness/flipchart/index.html

• In general
  – If we have a tornado siren, we'll respond to it
  – If we have a fire, we will evacuate the building
  – If we have an active shooter, we should do what we can to block the door, turn off the lights, shelter where there is no clear shot from the door, and attack if the shooter enters the room
  – If the campus is closed, you will not be penalized and there will be info on the course web page an by email about any assignments, tests, etc. that are due during the closure or shortly thereafter
  – If we leave during a test or quiz, please turn in what you have done at the desk before leaving. I'll leave the room last.
Grading

• Weekly assignments, class participation and homework, 10 points
• Individual and group projects, 45 points
• Midterms and finals, 45 points (15 points each for the final and each midterm.
• All exams are open-book, open-note unless otherwise announced.
  – No collaboration.
  – No electronic devices (including phones)
  – Because of the size of the class, we will be having night exams.
• Bring a photo ID for all exams
• Guaranteed an A if 90 points or higher, B if 80 or higher, C if 70 or higher, ..., after normalizing total points to 100. Note that I may draw the cut-off lines below these cut-offs, e.g., an 87 might get you an A.
• F: Cheat (I reserve the right to have a zero-strike policy), do poorly. You will have at least two chances to pass an outcome.
Exams
(open book, open note, individual)

• multiple choices, short answers, short code segments
• “zero-tolerance” of dishonesty: violations will be reported to the associate head of ECE and the dean of students.
• Cheaters (exams or assignments) will receive F in the class, *not just on the exam.*
• Regrading must be submitted by a written request (or email) within one week after the grade is posted. DO NOT make any marks on your exam after it is returned if you want a regrade. Attempting to change answers will count as cheating.
• You can arrange to take an exam at a different time for good cause. Please contact me as soon as possible.
• Getting permission to take an exam shortly before or after the scheduled exam time will be much harder to get approved.
Collaboration and Submitting work

• You can *discuss* lecture, homework, lab, or programming assignments with anyone. *You can share code* only for the group project and only with your programming partner (if you have one).

• You can have one and at most one partner for each group programming assignment. All other coursework must be done by yourself only. If you discuss with anyone, please document it in your submission.

• The submission strategy will be announced later, but you should ensure your programs run on the MSEE 190/ECN Linux machines.

• Having code similar to another project’s code may lead to an F in the assignment and/or the course.
Programming Assignments

• 3 larger programming assignments:
  – Java, 2 steps. There will likely be three projects to choose from. One will be a game, one will be design your own, with my approval.
  – C++, 1 step, may be a design your own option.
• Java is larger and will be a group project and you may work with one other partner
• You can discuss programming assignments with anyone but you are allowed to share code only with your partner.
• If you work in a team, both students must submit the same files as a group.
• I am happy to discuss other programming projects with you if you want to do something different.
Programming assignment turn-in and grading

• Programs will be turned into blackboard
• You will be asked to demonstrate these to the TA or to me for your grade
• You need to ensure that whatever you upload to Blackboard can be run on an ECN machine. Use of an IDE will make your life easier.
My goal for this class is that you know C++ and Java

• It is a waste of your time and money if you leave this class without knowing C++ and Java
• My interests are aligned with your long-term interests. I am not your enemy.
• Make use of this and ask questions or come to office hours or make an appointment if you are confused.
  – While I was a student I was at times completely confused, blew tests, etc.
  – Use office hours -- don’t be embarrassed!
  – If confused, go over the material, write a test program, and if still confused talk to me -- don’t wait for weeks!
• Teaching and learning are often (we hope) correlated, but are not the same activity
  – You have to internalize what I am saying to learn
  – You may need to do work beyond what I assign to completely learn stuff
My goal for this class is that you know C++ and Java

• I am human and therefore have limited powers. Therefore I:
  – often cannot respond to your email immediately.
  – do not know all of C++ and Java (I'll use as my defense that no one does) but will find out the answers to questions that I cannot answer in class
  – may be in a meeting when you want to talk.
  – have other responsibilities, such as writing recommendation letters, committees, seminars, conferences, papers, theses, ...

• If you make an appointment, make sure we have a set time.
  – When I send you a range of available times we need to decide on one and let me know what you decide
  – you shouldn’t just “show up” at one of the times.
Communication

• Email is the best way to contact me
• Please put 30862 in the subject line of the email for all email -- it will help it get in the right mailbox and for me to see it.
• Try to use the same email address all of the time
• Use your best efforts at English. “when r u giving 1 test?” doesn’t impress me, and it won’t impress your future employer/interviewer.
• Don’t start email with “hey” or “dude”. Use “Prof. Midkiff”, Dr. Midkiff or nothing.
Late Policy

• Late programming submissions will either get a zero or have points taken off for every day late

• **back up your work.** You can lose a job for not doing this and you can lose your grade for not doing this. Losing all of your work at 3AM a day before a project is due is not conducive to a happy life.

• Programming assignments will generally be due Friday or Sunday night

• See the TA for programming assignment questions -- only see me if he doesn’t work out.

• **back up your work!**

• The TA may not be available on Friday or Saturday when programs will typically be due. This is by design. Start early and use the last few days as a insurance, or for big projects, the last week or so as insurance.

• In the real world, being late costs real money and reputation. Get used to it.
The federal government requires Purdue to determine for every student in every class that you attended sometime in the first two weeks.

I take doing the first homework or two as proof of that. So make sure you turn these in.

I may take attendance next week to get better coverage of who is here.
Let's get to some mildly technical stuff
OO languages exist to make programmers more efficient

• The big concepts we will learn that OO languages use to make developing programs easier

• Objects

• Classes
  – function overloading
  – call resolution

• Inheritance
  – function overriding
  – call resolution

• Polymorphism

• Threads

• Containers

• The devil will often be in the details . . .
Why OO languages?

• Earliest languages (Fortran and Cobol) were tailored to specific domains (matrix oriented numerical applications and business processing with record oriented data)
  – Relatively low level
  – Built-in types, and operations on those types
  – Poor support for encapsulation -- a change in the data layout of a data structure could require changes throughout the program. Code making use of the data needed to know how the data was laid out

• Algol 60 made control flow easier, messier function/subroutine semantics
  – good support for *structured programming*
    • no goto’s, computed gotos, labels much rarer
    • supported loops were single entry and exit
    • helped kill spaghetti code
Early, non-OO languages

- Early Basics were syntactically and semantically easier to understand
  - control flow messed up completely with subroutine calls

  ```
  ... more user code
  $a1 = 42
  gosub func1
  func1: if ($a1 > 0) ...
  return
  ... more user code
  more user code
  more user code
  subroutines were inline
  you could `gosub` to them or drop through to them
  ...
  more user code
  ...
  more user code
  ```

- Pascal was a rebellion against unstructured data types and unstructured flow of control -- introduced abstract types (essentially structs and enums in C)

- PL/1 was IBM's attempt at a better language
  - lots and lots of primitive data types and compiler knowledge of those types
  - massive compiler, very complex, combined the best (and worst) of Fortran, Cobol, Pascal, Agol
C and the search for language simplicity

• Algol 60 mutated into Algol 68, also horrifically complicated
• C was a rebellion against both of these and the trend towards complexity
  – access to machine level actions (++, +=, *p, *p( ), <<, >>)
  – access to runtime concepts (malloc, longjump, ...)
  – some attempts at abstraction (struct, enum)
  – programmer had great power, and great responsibility
  – common idioms still rely on knowledge of the underlying data structure

        p = headP;
        while (p != null) {} 
        fprint("%d\n", p->data) // why not p.printData( );
        p = p.next;

– Knowledge of field names and actual datatype necessary to operate on the data
Program simplicity did not follow from language simplicity

```c
main(){printf("\021%six\012\0",unix)["have"]+"fun"-0x60);}

char*m="char*m=\c%s%c;main(){printf(m,34,m,34);}";main(){printf(m,34,m,34);}

main(a,b){while((a=getchar())+1)putchar((b=64^a&223)&&b<27?a&96l(b
  +12)%26+1:a);}

int i;main(){for(;i["" < i;++i){--i;}" ];read('-'-',i+++"hello, world!\n",'//'/'));read(j,i,p)
  {write(j/p+p,i---j,i/i);}

main(v,c)char**c;{for(v[c++]="Hello, world!\n");(!!c)[*c]&&(&v--ll--
c&&execvp(*c,*c,c[]!c[]+!c,!)c));**c=!c)write(!*c,*c,!*c);}

main(Q,O)char**O;{if(--Q){main(Q,O);O[Q][0]^=0x80;for(O[0][0]=0;O[++O[0][0]]!=0;)if(O[O[0][0]][0]>0)puts(O[O[0][0]]);puts("--------");main(Q,O);}}
Still, this all works fine until ...

- You have 5,000 programmers working against the same code base
  - Read Fred Brooks, *The Mythical Man Month*
  - Must program *to interfaces* (black boxes), not to internal logic
  - Must be able to believe specs, cannot hope to understand what 5,000 programmers are doing, or talk to them to figure it out
- You have a code base of 1,000,000+ lines that you are debugging
  - OS/360 had 4 or 5 million lines, almost killed IBM
  - Windows XP had a code base of 45,000,000+ lines
    - Was a year or so late
    - Didn’t really affect Microsoft’s revenue stream
- Object oriented languages are a way to deal with this complexity by programming to specifications using abstractions and specifications -- program to the problem not to the machine or needs of the compiler
What is a Class?

• A class describes the commonalities of similar objects:
  – Person: you, me, David, Mary, Tom, Amy ...
  – A student, which is a subset of all Persons
  – Car: your Toyota Camry, his Ford Explorer, Jennifer's Testarossa ...
  – Classroom: EE170, EE117, EE129 ...
  – Building: EE, MSEE, Purdue Bell Tower, Hovde Hall...

• A class describes both the attributes and the behavior:
  – Person: name, home ... + sleep, eat, speak ...
  – Student: a Person + GPA, major, etc.
  – Car: engine size, year ... + accelerate, brake, turn ...
Relationship among Classes

• A class can be a special case of another class:
  – Student is a special case of Person
  – Laptop is a special case of Computer
• This important relation is (not surprisingly) called the "is a" or (ISA) relationship.
• The ISA relationship often implemented by inheritance.
• The ISA relationship allows a specialized form of a more general class to be created. ISA often implemented using inheritance.
  – things that are appropriate for the specialized class in the general class can be reused
  • Student class may reuse date of birth, address, etc. information from Person class
  • May add major, GPA, etc., to person class to specialize it
Inheritance

• Let’s say we want to have objects for undergraduate students, graduate students and staff
• All of these likely have a *date of birth*, *SSN* or other identifier, *address*, and other attributes.
• Undergraduate students might have a *GPA* and *academic status* (on probation, Dean’s List, etc.), *major field of study*, etc.
• Graduate students might have a *GPA* and *academic status* (on probation or not), *department*, etc.
• Staff might have *position*, *number of years in service*, *salary*, etc.
• Two possible implementations
  – Undergrad, Grad and Staff might all have Person attributes and associated methods or point to/reference something that does
  – Undergrad, Grad and Staff might have the *ISA* relation with Person.
First option

Undergrad
Date of birth
SSN/other id
Address
GPA
Academic status
Major

Grad
Date of birth
SSN/other id
Address
GPA
Academic status
Department

Staff
Date of birth
SSN/other id
Address
Position
Years in service
Salary
Second option -- *inherit* from Person

- Date of birth
- SSN/other id
- Address

- Undergrad
  - GPA
  - Academic status
  - Major

- Grad
  - GPA
  - Academic status
  - Department

- Staff
  - Position
  - Years in service
  - Salary
Second option -- *inherit* from Person

<table>
<thead>
<tr>
<th>Undergrad</th>
<th>Grad</th>
<th>Staff</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date of birth</td>
<td>Date of birth</td>
<td>Date of birth</td>
</tr>
<tr>
<td>SSN/other id</td>
<td>SSN/other id</td>
<td>SSN/other id</td>
</tr>
<tr>
<td>Address</td>
<td>Address</td>
<td>Address</td>
</tr>
<tr>
<td>GPA</td>
<td>GPA</td>
<td>GPA</td>
</tr>
<tr>
<td>Academic status</td>
<td>Academic status</td>
<td>Academic status</td>
</tr>
<tr>
<td>Major</td>
<td>Department</td>
<td></td>
</tr>
</tbody>
</table>

Date of birth, SSN/other id, Address
Why Inherit

• **Save the time** of re-implementing methods in *Person* that manipulate the different *Person* attributes
  – Could do this with an **HASA** relation, e.g., undergrad **HASA** *Person*
  – Somehow, this doesn’t sound right, a student **ISA** person, does not **HASA** a person!

• **Enables** **polymorphism**
  – Because an undergrad, staff and grad object all inherit from *Person* they are all both a *Person* and an *Undergrad/Grad/Staff* object
  – An *Undergrad object* is both a *Person* object and an *Undergrad object*
    • Can be used anywhere either an *Undergrad* or *Person* object can be used
    • Its type is both *Undergrad* and *Person*
  • With polymorphism, e.g., can have an array that contains (or contains *reference to*) a *Person*, Graduate, Undergrad or Staff object!
What is an Object?

• An object can be a “concrete and tangible” entity that can be separated with unique properties:
  – you
  – your book
  – your rat rod
  – my computer
  – Tom's ancient computer
  – Otto’s dog ...

• In programs, an object
  – is an instance of a class
  – is used to represent an entity that the program will operate on
Objects' Three Properties

• Each object is unique and can be identified using a handle, a serial number, a relationship with another object ...
  – In Java and C++ this often accomplished using the address of an object or some function of the address
  – Within a program the programmer may use other criteria (e.g., a serial number or relationship) to identify objects

• Each object has a set of attributes, such as location, speed, size, address, phone number, on/off ...
  – These are often implemented using variables and other data structures within the object

• Each object has behaviors, such as ring (phone), accelerate and move (car), take picture (camera), send email (computer), display caller (pager)
  – These are often implemented using functions or methods associated with the object

• To summarize, each object has three important properties:
  – It has a unique identity
  – It has attributes and holds state
  – It has behaviors (actions) that it can perform
Polymorphism -- an essential OO feature

• We have Student objects that ISA Person object (i.e., the Student class inherits from the Person class.
  – Sometimes we may want to treat the object as if it is a Student object
  – Sometimes we may want to treat the object as if it is a Person object
    • For example, we may want to put Student and Person objects into an array and have each do a Person action
  – Polymorphism allows an object to be treated as many kinds of objects.
Encapsulation - an essential OO feature

• An object can hide information (attributes) from being manipulated by or even visible to other objects:
  – Functions associated with an object may be able to return a Student’s GPA, but functions not associated with the object cannot tell whether the Student object stores a GPA, how it is stored if it is stored, and how it might be computed if it is not.

• An attribute may only be modified via restricted mechanisms or channels to help maintain consistency.
  – A person's address and phone number must be changed together when this person moves.
  – This is enforced by having restricted mechanisms for changing the address and phone number in an object.

Encapsulation is essential for code reuse and to enable OO languages to enhance productivity!