CS3157: Advanced Programming

Lecture #8
June 20
Shlomo Hershkop
shlomo@cs.columbia.edu
Overview

- More Classes and hierarchies
- Basic DS
- Intro perl…
- Next time:
  - Search engine Technology
  - Programming Patterns
Reality check

- int *p = new int (1024);

- int *q = new int [1024];

- int (*r)[1024] = new int [4][1024];
Dynamic allocation

- Local variables have local life and scope
- If you want to dynamically create and manage memory, use the new and delete
- Using pointers

- Have to be careful from dangling pointers…
- Ideas?
base class constructors

- need to launch base class constructor in derived class if you don’t want the default to be called

- destructors are reversed

- lets see this in action..code a simple inheritance chain…
issue

- one issue with overriding, is that if the derived class doesn’t provide a function, we will use the base class definition

- this doesn’t always make sense

- Example I want a function MPG for any type of vehicle, but doesn’t make sense of base class
virtual functions

solution:

- declare the function to be virtual

- virtual double MPG();

- allow you to use a base class pointer to call at runtime the correct function (polymorphism)
abstract class

- sometimes it's even useful to have a base class which can't be instantiated
- if any virtual function is declared pure virtual:

```cpp
virtual int MPG() = 0;
```
note

- constructors cannot be virtual

- need virtual destructors to make everything work if you are going to have destructors in any of your classes (do it anyway)
lets look at 20.1 code
Linkage directions

- If you want to call a function in another programming language, the compiler must be told that different rules apply.

- Linkage directive
  - Single statement
  - Compound form

- Declared outside of functions
Single form

- extern “C” void something(int);

- Keyword
- String
- Function
- Compiler will type check any function calls
Compound form

- extern "C" {
  int printf(const char * …);
  int scanf (const char * …);
}

- extern "C" {
  #include <cmath>
}

```c
extern “C” {
  int printf(const char * …);
  int scanf (const char * …);
}
```
Other languages

- Depends on the compiler
- For example many support
- FORTRAN
Abstraction and member functions

- How are objects internally manipulated by C++? Let's take a look at a complex example.
Rect

class Rect {

    // ...
    private:
    int top, left;
    int width, height;
    ..
};
Color

class Color{
    // ..
    private:
    int data;
};
class TextBox: public Rect{
    //...
    private:
    Color txtColor;
    int frameThick;
    char *text;
};
main

main() {
    TextBox source, dest;
    //...
    dest = source;

    How to get this to work?
Overloading operator =

class TextBox : public Rect{
  public:
    void operator=(TextBox &source);
  ..
Equivalent

main()
{
    TextBox source, dest;

    // ...

    dest.operator=(source);
}
Inside

```cpp
void TextBox::operator=(TextBox &source) {
    if(this == &source)
        return;

    Rect::operator=(source);

    txtColor = source.txtColor;
    frameThick = source.frameThick;

    delete []text;
    if(source.text != 0) {
        text = new char[strlen(source.text+1)];
        strcpy(text,source.text);
    } else {
        text = 0;
    }
}
```
Implicit assignment

- If you don’t define an assignment operator
  - Will try to figure out how do to it
  - By looking at each field member variable
  - Works with primitives
  - Pointers will get shallow copied
    - Difference between
      - DEEP COPY
      - SHALLOW COPY
Copy constructor

- TextBox t2 = t1;

- Looks like assignment
- Really a constructor call with object as argument
- Called copy constructor
- Combination of constructor and assignment
Defining it

- Just overload the constructor
- `TextBox(TextBox &source);`

Be careful:
- When you overload the copy constructor you throw out a default constructor
- Which means you need to explicitly define a default constructor (no arg)
code

TextBox::TextBox(TextBox &source) {

Rect::operator=(source);

frameThick = source.frameThick;
textColor = source.textColor;

etc
Chaining

- If you want to be able to say
  
  ```c
  Textbox a, b, c;
  
  // ...
  
  a = b = c ;
  ```

- how would the operator overloaded be different ??
Exception

- Like in java, CPP allows you to throw and catch exceptions
  
  - Compiler time exceptions
  - Run time exceptions
Template programming

- Allows you to specify a type to pass in to your class, so can create a collection class to handle many different types, without having the problem if limited casting in the code

- Allows you to move errors from run time to compiler time
Templates

template<typename X>
void foo(X &first, X second)
{
    first += second;
}

Exceptions..

- try
- catch

- similar to java
  - except won’t escalate if not caught unless specifically enclose in try->catch blocks
  - general exception class (<exception> and std::exception)
23_2
virtual functions

- in C++ virtual functions allow you to define a specific function in the base class, which is undefined, and each of the subclasses need to override (implement a definition)

- virtual char * md5sum();
- so if we use a base class pointer at a derived class object, calling md5sum will call the correct one

- compile time resolution
  - static binding
Abstract

- virtual char * md5sum() =0;

- any ideas on what error will be thrown if you instantiate it?
non virtual base functions

- if you have a parent class A.foo()
- derived class B defines B.foo()

- A *a_ptr = B_object

- a_ptr.foo()
  - which foo will be triggered?
  - why?
abstract classes II

- remember that making a pointer doesn’t instantiate anything
- can create pointers of type abstract classes
- used to enable polymorphic behavior

Example: Operating system device
  - read/write behavior
destructors

- when creating and manipulating objects in a polymorphic context, destructors will only be called on base class
solution

- define a virtual base class destructor
- will correct destructor will be called
Virtual functions

- Allows you to declare a function in the base class without a definition
- Each of the derived class provide a definition unique to their implementation
- At runtime will allow all derived class object instances to be manipulated uniformly
Next

How do search engines work...
First

- Background

- What is data structures?
Idea

- Data Structure
- ADT

Example:
- List ADT
Operations

- Want to be able to add things to list
- And locate things in the list
- Maybe even remove things from the list
Simple list

- Array

- Issues:
  - What do to when need to grow the list
Alternative

- Links

- Idea: each item can link to new item

- Will allocate new space per need
Question

- So which is better?

- Which is faster?
Problems

- Let's do an example of putting things in the list and finding them

- Try to see how and why things might take a while....

- Can we do better?
Sketch

- Hash table idea

- If we don’t care about order of insert can get truly fast organization of list items
Search engine technology

- Generally search engines work in the following way:
  - collect webpages
  - index information
- wait for user to search
  - Return relevant results
So how can we quickly find results?

That is match the terms to the websites pages?
Vector Model

- Each document is a vector in an n-dimensional vector space of search terms.
- Take query and find closest points.
- Sparse (very).
- If one word token, order will be ignored.
Need to generate a master word list
  - Can run through a bunch of examples for this

- can strip out stop words
  - Std lists exist

- can also calculate related words i.e. runs and run worry and worrying through “stemming”
Example: master word list

- cat
- dog
- fine
- good
- got
- hat
- make
- pet

# A cat is a fine pet
$vec = [ 1, 0, 1, 0, 0, 0, 1 ];
many ways of getting similarity between two documents

Cosine distance
General issues

- Better parsing
- Non-English Collections
  - stemming
  - stop words
- Similarity Search
  - can combine a few docs to find similarity
- Term Weighting
- Incorporating Metadata
- Exact Phrase Matching
- hash table of search words

- inverted index table
  - Table of words and then list of documents which they appear in..
  - Can add freq and locations
Something cool......
Now that we’ve covered C/C++

- Let's do something much nicer...

- Imagine if the programming language tried to figure out what you want...and did it for you
  - Welcome to scripting languages...
Perl

- Perl
  - History
  - Version 5.6+
  - Rumblings of version 6

- What is it?
  - Scripting language
  - Aims to be a USEFUL language
  - Base + tons of libraries
  - Both a compiler and byte code executable

- Where to get it?
  - cpan.org
  - www.activestate.com/Products/ActivePerl/
Compiler/interpreter

- Perl is interpreted
- The script needs to tell the system where the interpreter is sitting
- Accomplished by special command on the first line of your program:

```bash
#!/usr/bin/perl
or
#!c:\perl\bin
```
- Comments start the line with a hash, will continue to end of line mark

- In addition, on unix/linux need to tell system to execute your perl script
  
  ```
  chmod +x test.pl
  ./test.pl
  ```

- The other way is to call perl directly
  
  ```
  perl test.pl
  ```
Built in functions

- Can call tons of built in functions to do stuff in perl
- Can define your own (later today)
- One is the print command

```perl
print "something\n";
```
test.pl

#!/usr/bin/perl

#your first perl program

print "hello everyone\n";
Technical details

- By default the start of your code is the equivalent of “main”
- Will run each line in turn, execute and then next line
- Will end when reach end of code
Structure

- **Whitespace**
  - only needed to separate terms
  - all whitespace (spaces, tabs, newlines) are treated the same
  - Use them to make the code look nice, easier to look over

- **Semicolons**
  - every simple statement must end with one
  - except compound statements enclosed in braces (i.e., no semicolon needed after the brace)
  - except final statements within braces
  - Advice: ALWAYS use semicolons for commands
Something NEW!

- Most languages you know already, variables need to be declared ahead of time, what type they will deal with

- By default perl, will try to figure out what you mean

- Which means as soon as you use a variable for the first time, perl will assign it a type then

- So initialization and declaration/assignment happen at once
Data types

The basic data types are as follows, we will go through each in turn

- scalars ($)
- arrays (@)
- hashes (%)
- subroutine (&)
- typeglob(*)
Scalars

- This type of variable starts with a $:
  - $first
  - $course

- Can hold: int, real, string:
  - 234
  - -89
  - 36.34
  - “hello world”

- Context dependant:
  - $name = “shlomo”;  #perl sees this is a string
  - $n = 123;  #perl sees this as a number
Arrays

- Starts with @
- Order list of scalars
  - @class3157 = ("shlomo","weijen","edward");
- The scaler type in the array can be anything a scalar can hold
  - So can mix numbers and strings etc
- To reference elements, use the variable name with a dollar in front and subscript
  - $class3157[0]; #is shlomo
- Since perl tries to be useful what do you think this should be:
  - $class3157[-1];
  - $class3157[14];
Related to basic arrays

- Can get the length easily by:
  - $a = @class3157;
  - or
  - print @class3157
- Elements in the array
  - $class3157[i]
- Referencing an array through a scaler
  - $ref = \@class3157;
- De-refrencing an array
  - $$ref[0]

- This can be done with any perl type
- Will print \texttt{ARRAY(0x18328cc)} when printing a referenced array
Question?

- Why would you want to use a reference to an array?
Answer

- Say you want to pass in 2 arrays into functions, you will need to reference the arrays.....
Java question:

- Say you have a list of numbers, and you want the max and average value ....

- Can you want to return 2 values from a method ?

- So what do you actually do in Java ?
Java solutions:

- 2 separate methods
  - i.e. run through a million numbers twice
- Global variables
- Wrap in array, i.e. return an array of results
- Wrap in a class, i.e. return some class with the values inside
Anonymous arrays

- Really cool perl feature:
  - Can create the equivalent of an anonymous array by simply putting parenthesis around scalars
  - return ($max,$average);

- Example 2:
  - ($first,$second) = @number
  - This will grab the first and second scalars from the number array
Hashes

- Very useful built in type:
- Curly braces say is a hash type
- name/values pairs can be defined in one of two ways:
  - %phonelist = {adam=>718, barry=>345};
  - or
  - %phonelist = {“adam”,718,“barry”,345};

- Use the name to find the value
  - $phonelist{“adam”}  #is 718
Modifiers allow you to differentiate between variable types.

Local – a local variable exists within the local section, either main or a subroutine or code block (between braces)
- my $name

Global – can be seen by any section of the code
- our $name
- local $name

Special – these can change where a variable can be accessed
- ALL
- LEX
- RO
- PKG
Next

- Now that we covered the basic types of perl lets start to get to the logic/rules of the code
Programming statements

- simple statements are expressions that get evaluated
- they end with a semicolon (;)
- a sequence of statements can be contained in a block, delimited by braces ({ and })
- the last statement in a block does not need a semicolon
- blocks can be given labels:

```plaintext
myblock: {
  print "hello class\n";
}
```
Conditional Statements

1. simple if
   if (expression) {block} else {block}

2. unless
   unless (expression) {block} else {block}

3. compound if
   if (expression1) {block}
   elsif (expression2) {block}
   ... 
   elsif (expressionN) {block}
   else {block}
Loops

- while
- for
- foreach
while

syntax:
   while (expression) {block}

elementary

#!/usr/bin/perl
@b = (2,4,6,8);
$a = @b;
$i=0;

while ( $i < $a ) {
   print "i=",$i," b["i"]=",$b[$i],"\n";
   $i++;
}


for

syntax:
  for ( expression1; expression2; expression3 ) {block}

eample:

#!/usr/bin/perl
@b = (2, 4, 6, 8);
$a = @b;
for ( $i=0; $i<$a; $i++ ) {
    print "$i=", $i, " b[", $b[$i], ",\n";
}
foreach

The foreach statement allows you to quickly cycle through array values
syntax:
   foreach var (list) {block}

example:

#!/usr/bin/perl
@b = (2,4,6,8);
$a = @b;

foreach $e (@b) {
   print "e=",$e,"\n";
}
Question

- So if foreach allows you to cycle through array
- How would you cycle through hash, since its composed of key->value pairs?
keys

- Built in command
- Allows you to fetch all the keys of the hash type
- Use each one to access the individual value pair
Side note

Side note:

to look up perl command can use the perldoc command:

perldoc -f keys

keys HASH

Returns a list consisting of all the keys of the named hash. (In scalar context, returns the number of keys.) The keys are returned in an apparently random order. The actual random order is subject to change in future versions of perl, but it is guaranteed to be the same order as either the "values" or "each" function produces (given that the hash has not been modified).
Example:

```
foreach $k (keys $emailcount) {
    print "$k = 
    print $emailcount{$k};
    print "\n";
```
Controlling loops

- **next**
  within a loop allows you to skip the current loop iteration

- **last**
  allows you to end the loop
Modifiers

- Should be careful about assuming a line will execute, read it carefully.
- You can follow a simple statement by an `if`, `unless`, `while`, or `until` modifier:
  - `if` expression; statement
  - `unless` expression; statement
  - `while` expression; statement
  - `until` expression;

- Example:
  ```perl
  #!/usr/bin/perl
  @b = (2, 4, 6, 8);
  $a = @b;

  print "hello world!\n" if ($a < 10);
  print "hello world!\n" unless ($a < 10);
  #print "hello world!\n" while ($a < 10);
  print "hello world!\n" until ($a < 10);
  ```
Reserved variables

there’s a (long) list of global special variables...
a few important ones:

$_ = default input and pattern-searching string, its usually the last scalar you touched

example:

#!/usr/bin/perl
@b = (2,4,6,8);

foreach (@b) {
    print $_,"\n";
}

Reserved II

- \$/ = input record separator (default is newline)
- $$ = process id of the perl process running the script
- $< = real user id of the process running the script
- $0 = (0=zero) name of the perl script
- @ARGV = list of command-line arguments
- %ENV = hash containing current environment
- STDIN = standard input
- STDOUT = standard output
- STDERR = standard error
Operators

- unary:
  1. ! : logical negation
  2. - : arithmetic negation
  3. ~ : bitwise negation

- arithmetic
  1. +,-,*,/,% : as you would expect
  2. ** : exponentiation

- relational
  1. >, <=, <=, <= : as you would expect

- equality
  1. ==, != : as you would expect
  2. <--> : comparison, with signed result:
  3. returns -1 if the left operand is less than the right;
  4. returns 0 if they are equal;
  5. returns +1 if the left operand is greater than the right
Operators II

assignment, increment, decrement

- =
- +=, ++
- -=, --
- *=, **=, /=, %= 
- &=, ||=

just like in C/Java
Ok so far

- We have variable types and some logic

- Write your own functions, which Perl calls subroutines
Subroutine

- syntax for defining:
  sub name {block}
  sub name (proto) {block}

- where proto is like a prototype, where you put in sample arguments

- syntax for calling from your code:
  name(arg list);
  name arglist;

  (the & sign used to be required when calling functions, this has been changed in the latest perl: it is optional if you use parenthesis in the method call)
code

- Usually the sub definitions are placed at the bottom of the file
- No reason, but makes code easier to read
strings

- Double quoted strings are interpreted, so you can have a scalar in it and it will be translated
  “hello: $name\n”;

- Use a period to combine strings
  “hello” . $name . “\n”;

- Single quote strings are not interpreted, they are read literally
Simple example

print "welcome to the program";
testsub();

sub testsub()
{
    print "hi everyone\n";
}

Passing in values

- All arguments to a subroutine come in on the @_ array
- That is argument list is packaged as an array and can be accessed in the sub
- 3 different ways to grab arguments:
  - $a = @_[0];
  - ($a) = @_;  
  - $a = shift;
    - Shift moves out the top of an array, if we don’t specify it (and its first) we are talking about the default array (nice and confusing).
So...what will be printed here?

$n = 45;

print "n is now $n\n";
testsub($n);
print "n is now $n\n";

sub testsub{
    $a = shift;
    print "in testsub 1 = $a\n";
    $a++;
    print "in testsub 2 = $a\n";
}

Any idea?

- By default pass by value
- Which means a copy is sent in
- Any changed will be only local to subroutine
Pass by reverence

- The way to be able to pass in a scalar and have its changes stay, is to pass in a reference to the scalar...i.e. you are really manipulating the original one

- Use a backslash to say it’s a scalar
- Use extra dollar to say dereference
Pass by reference

$n = 45;

print "n is now $n\n";
testsub(\$n);
print "n is now $n\n";

sub testsub{
    $a = shift;
    print "in testsub $a\n";
    $$a++;
}

Working with files 1

- Since perl was made to be useful, many things which are complicated in other language become super easy in perl

- To go through a file 3 step process
  1. Open
  2. Read
  3. Close
When you open, you need to say what type of operations you will be doing:

- `open( FILEHANDLE, filename );`
  - # to open a file for reading
- `open( FILEHANDLE, >filename );`
  - # to open a file for writing
- `open( FILEHANDLE, >>filename );`
  - # to open a file for appending
Errors

- So what if the open command fails:
- easy trick:
  - Can use double pipe to do something
  - `|| warn print "message";`
    - or if you want to fail:
  - `|| die print "message";`
Printing with files

- Once you’ve opened the file, can send something to it by putting it after the print command
- The filehandle is the variable you specify with the open command

- print FILEHANDLE, “….”;  
- Don’t forget to close the file when done writing
Write example

example:
#!/usr/bin/perl
open( MYFILE,">a.dat" );
print MYFILE "hi there!\n";
print MYFILE "bye-bye\n";
close( MYFILE );
File handles

- Once you open a file handle, can get stuff from it using the pointy brackets
- `<MYFILE>`
- This basically reads a line of text from the file
- Since destination isn’t specified its read to the `$_`
- You can also say:
  - `$line = <MYFILE>;
    which is like doing a readline in other languages`