CS3157: Advanced Programming

Lecture #6
June 14
Shlomo Hershkop
shlomo@cs.columbia.edu
Overview

Today:

- Starting C++
- Language basics: identifiers, data types, operators, type conversions, branching and looping, program structure
- data structures: arrays, structures
- pointers and references differences
- I/O: writing to the screen, reading from the keyboard, iostream library
- classes: defining, scope, ctors and dtors
- Bunch of code
Announcements

- Labs
- Project
- Make up class
  - Need to makeup class, can we do this Friday? 3 - 6pm?
  - Maybe Sunday?
- Another choice
  - take home final and class last session....
Four main OOP concepts

- **abstraction**
  - creation of well-defined interface for an object, separate from its implementation
  - e.g., Vector in Java
  - e.g., key functionalities (init, add, delete, count, print) which can be called independently of knowing how an object is implemented

- **encapsulation**
  - keeping implementation details “private”, i.e., inside the implementation

- **hierarchy**
  - an object is defined in terms of other objects
  - Composition => larger objects out of smaller ones
  - Inheritance => properties of smaller objects are “inherited” by larger objects

- **polymorphism**
  - use code “transparently” for all types of same class of object
  - i.e., “morph” one object into another object within same hierarchy
Main difference between c and cpp

- C’s power is driven by functions. You define a set of function which operate in a specific sequence to implement some algorithm
  - Top down
- CPP is an object oriented language
  - Bottom up
Compatible

- Cpp is backwards compatible with c
- Cpp is bottom up approach
- Cpp compilers will compile c code
- As with any other language….
  - ✓ THINK
  - ✓ PROGRAM
Advantages

- Can create new programs faster because we can reuse code
- Easier to create new data types
- Easier memory management
- Programs should be less bug-prone, as it uses a stricter syntax and type checking.
- `Data hiding', the usage of data by one program part while other program parts cannot access the data
- Will whiten your teeth
C++ vs. Java

advantages of C++ over Java:
- C++ is very powerful
- C++ is very fast
- C++ is much more efficient in terms of memory
- Compiled directly for specific machines (instead of bytecode layer, which could also be seen as a portability advantage of Java over C++...)

disadvantages of C++ over Java:
- Java protects you from making mistakes that C/C++ don’t, as you’ve learned now from working with C
- C++ has many concepts and possibilities so it has a steep learning curve
- Extensive use of operator overloading, function overloading and virtual functions can very quickly make C++ programs very complicated
- Shortcuts offered in C++ can often make it completely unreadable, just like in C
Sample Hello.cpp

```cpp
#include <iostream.h>
#include <stdio.h>

using namespace std;

main() {
    cout << "hello world\n";
    cout << "hello" << " world" << endl;
}

- compile using:
g++ hello.cpp -o hello

- like gcc (default output file is a.out)
main()

- In C main is the first thing to run
- C++ allows things to run before main, through global variables
  - What is the implications?
- Variable which are declared outside of main, have global scope (will cover limits).
- Can have function calls here
File conventions

- No one convention
  - .C
  - .cc
  - .cp
  - .cpp  ← I prefer this
  - .cxx
  - .c++
Keywords c++

- asm
- catch
- class
- friend
- delete
- inline
- new
- operator

- private
- protected
- public
- this
- throw
- template
- try
- virtual
Identifiers

- i.e., valid names for variables, methods, classes, etc
- just like C:
  - names consist of letters, digits and underscores
  - names cannot begin with a digit
  - names cannot be a C++ keyword

- literals are just like in C with a few extras:
  - numbers, e.g.: 5, 5u, 5L, 0x5, true
  - characters, e.g., 'A'
  - strings, e.g., "you" which is stored in 4 bytes as 'y', 'o', 'u', '\0'
data types

- simple native data types: bool, int, double, char, wchar_t
- bool is like boolean in Java
- wchar_t is “wide char” for representing data from character sets with more than 255 characters
- modifiers: short, long, signed, unsigned, e.g., short int
- floating point types: float, double, long double
- enum and typedef just like C
Operators

- same as C, with some additions

- if you recognize it from C, then it’s pretty safe to assume it is doing the same thing in C++

- Operator overloading…
  - What it is
Type conversions

- all integer math is done using int datatypes, so all types (bool, char, short, enum) are promoted to int before any arithmetic operations are performed on them

- mixed expressions of integer / floating types promote the lower type to the higher type according to the following hierarchy:

  int < unsigned < long < unsigned long < float < double < long double
Conversions II

- you can do explicit conversions like in C using cast
  - (int)something
- you can also do explicit conversions using C++ operators:
  - static_cast
    - safe and portable; e.g. c = static_cast<char>(i);
  - reinterpret_cast
    - system dependent, not good to use
  - const_cast
    - lets you change a const into a modifiable variable
  - dynamic_cast
    - used at run-time for casting objects from one class to another (within inheritance hierarchy); this is sort of like Java but can get really messy and is really a more advanced topic...
Branching and Looping

- if, if/else just like C and Java
- while and for and do/while just like C and Java
- break and continue just like C and Java
- switch just like C and Java
- goto just like C (but don’t use it!!)
Program structure

- just like in C

- program is a collection of functions and declarations

- language is block-structured

- declarations are made at the beginning of a block; allocated on entry to the block and freed when exiting the block

- parameters are call-by-value unless otherwise specified
arrays

- similar to C
- dynamic memory allocation handled using new and delete instead of malloc (and family) and free

- examples:
  ```
  int a[5];
  char b[3] = { 'a', 'b', 'c' };
  double c[4][5];
  int *p = new int(5); // space allocated and *p set to 5
  int **q = new int[10]; // space allocated and q = &q[0]
  int *r = new int; // space allocated but not initialized
  ```
Starting

- Let's talk about how functions are different in C++
Defining c++ functions

- a function’s “signature” is its name plus number and type of arguments
- you can have multiple functions with same name, as long as the signatures are different
- example:
  
  ```
  void foo( int a, char b );
  void foo( int a, int b );
  void foo( int a );
  void foo( double f );
  main() {
    foo( 1, 'x' );
    foo( 1, 2 );
    foo( 3 );
    foo( 5.79 );
  }
  ```
- OVERLOADING – when function name is used by more than one function
C++ Function II

- Foo() or Foo(void) for void arguments
  - Different than c
- Foo(…) for unchecked parameters
  - Look up va_list and va_start
  - A cleaner approach is to pass in an array

New Trick:
- Foo(int a, int b, int c=10)
  - Foo(4,5,2)
  - Foo(4,5)
Function III

- Inline functions
- Function overloading:
  - void foo(int a, char c)
  - void foo(char c)

- Not allowed are changes to return with same args
  - void foo(int a)
  - int foo(int a)
Other additions

- C++ includes many compiler side additions to help the programmer (yes that is you) to write better code
- Other technical changes (will be pointing them out as we pass them)
Void pointers

- C allows you to assign and convert void pointers without casting
- C++ needs a cast

```c
void * V;
...
Foo *f = (Foo)V;
```
NULL

- null pointer (0)
- in C, it’s a language macro:
  #define NULL (void *) 0
- in C++, it’s user defined because otherwise an explicit cast is needed!
  #define NULL 0

- Some books recommends using 0 instead of NULL
enums

- Are treated a little differently in c++

enum day {Sunday, Monday, .. }

day X = 1;  //only works in c

day X = Sunday;
Structures

- struct keyword like in C (but you don’t need typedef) (last class)

- use dot operator or -> to access members (fields) of a struct or struct *

- C++ allows **functions** to be members, whereas C only allows data members (i.e., fields)

- example

```cpp
struct point {
    public:
    void print() const { cout << "(" << x << "," << y << ")"; }
    void set( double u, double v ) { x=u; y=v; }
    private:
    double x, y;
}
```
Pointers and References

- pointers are like C:
  - int *p means “pointer to int”
  - p = &i means p gets the address of object i. references are not like C!! they are basically aliases – alternative names – for the values stored at the indicated memory locations, e.g.:
    
    int n;
    int &nn = n;
    double a[10];
    double &last = a[9];

- The difference between them:
  
  int a = 5; // declare and define a
  int *p = &a; // p points to a
  int &refa = a; // alias (reference) for a
  *p = 7; // *p points to a, so a is assigned 7
  refa = *p + 1; // a is assigned value of *p=7 plus 1
I/O Screen

// hello world in C++
#include <iostream>
using namespace std;
int main() {
    cout << "hello world" << endl;
}

- comment characters are // or /* ... */ , just like Java
- using namespace is sort of like importing a package in Java; it is used in conjunction with the header declaration
- you could also say #include <iostream.h> and leave out the using namespace std; line; this is an older style of C++ but it still works
- cout << is like System.out.print in Java or like printf() in C
- endl outputs a newline; saying cout << "\n"; does the same thing
  - Advantage is its system dependant
it’s preferred not to use C’s stdio (though you can), because it’s not “type safe” (i.e., compiler can’t tell if you’re passing data of the wrong type, as you know from getting run-time errors...)

- stdio functions are not extensible
- note << is left-shift operator, which iostream “overloads”
- you can string multiple <<’s together, e.g.:
  - cout << "hello" << " world" << "\n";
- cout is like stdout
- cerr is like stderr
I/O keyboard

- read from the keyboard using cin >>, which is like scanf() in C

- example:
  ```cpp
  #include <iostream>
  using namespace std;
  int main() {
    int i;
    cout << "enter a number: ";
    cin >> i;
    cout << "you entered " << i << "\n";
  }
  ```
C++ iostream

- two bit-shift operators:
  - `<<` meaning “put to” output stream (“left shift”)
  - `>>` meaning “get from” input stream (“right shift”)

- three standard streams:
  - `cout` is standard out
  - `cin` is standard in
  - `cerr` is standard error

- the iostream library is “type safe”, so you don’t have to use formatting statements:
  variables are input/output based on their datatype
ostream and istream

- ostream
  - cout is an ostream, << is an operator
  - use cout.put( char c ) to write a single char
  - use cout.write( const char *p, int n ) to write n chars
  - use cout.flush() to flush the stream

- istream
  - cin is an istream, >> is an operator
  - use cin.get( char &c ) to read a single char
  - use cin.get( char *s, int n, char c='\n' ) to read a line (inputs into string s at most n-1 characters, up to the specified delimiter c or an EOF; a terminating 0 is placed at the end of the input string s)
  - also cin.getline( char *s, int n, char c='\n' )
  - use cin.read( char *s, int n ) to read a string
Formatted output

- in `<iomanip>` header file, the following are defined:
  - scientific – prints using scientific notation
  - left – fills characters to right of value
  - right – fills characters to left of value
  - internal – fills characters between sign and value
  - setfill( int ) – sets fill character
  - setw( int ) – sets field width
  - setprecision( int ) – sets floating point precision
Example

- `cout << setprecision(3) << 2.34563;`
Declaring Class

- Almost like struct, the default privacy specification is private whereas with struct, the default privacy specification is public
- **example**

```cpp
class point {
    double x, y; // implicitly private
public:
    void print();
    void set( double u, double v );
};
```

- classes can be nested (like java)
- static is like in Java, with some weird subtleties
Create some code

- Lets create a simple c++ example printing out something
- Create makefile
- Run it

- Now lets add the point class example..
Using

```c
point x;
x.set(3,4);
x.print();

point *pptr = &x;

pptr->set(3,2);
pptr->print();
```
Classes: function overloading and overriding

- overloading:
  - when you use the same name for functions with different signatures
  - functions in derived class supercede any functions in base class with the same name

- overriding:
  - when you change the behavior of base-class function in a derived class
  - DON'T OVERRIDE BASE-CLASS FUNCTIONS!!

- because compiler can invoke wrong version by mistake
Access specifiers

- In class declaration can have:
  - Public
    - Anyone can access
  - Private
    - Only class members and friends can access
Access specifiers

- **public**
  - public members
  - can be accessed from any function

- **private members**
  - can only be accessed by class’s own members
  - and by “friends” (see ahead)

- **Protected**
  - Class members, derived, and friends.

- “access violations” when you don’t obey the rules...
- can be listed in any order
- can be repeated
Class scope

- `::`
- `example:
  ::i // refers to external scope
  point::x // refers to class scope
  std::count // refers to namespace scope

- given previous definition of point, we could do:
  ```
  point p;
  p.print();
  p.point::print(); // redundant but legal
  ```
Defining functions

```cpp
void point::print()
{
    cout << "(" << x << "," << y << ")";
}

void point::set( double u, double v )
{ x=u; y=v; }
```
Constructors and destructors

- Constructors are called ctors in C++
  - take the same name as the class in which they are defined, like in Java

- Destructors are called dtors in C++
  - take the same name as the class in which they are defined, preceded by a tilde (~)
  - sort of like finalize in Java

- Ctors can be overloaded and can take arguments
  - Dtors can not

- Default constructor has no arguments
More coding...

class point {
  double x, y;
  public:
  point() { x=0; y=0; } // default
  point( double u ) { x = u; y = 0; }
  // conversion
  point( double u, double v )
  { x = u; y = v; }

  //...
}
usage

point p;
CPP pass by reference

- Another way of passing by reference
  int count = 10;
  int &rcount = count;
references

void foo2(int &);

void foo(int &refint){
    refint *= refint;
}

Variable scope

- CPP allows you to specify scope through unary scope operator (::)
- So can differentiate between local and global variables
int count = 10;

int main()
{
    int count = 5;

    // count is local
    // ::count is global
    // std::count is the same as 2
Inline functions

- What are they

- Where do you code them?
Functions organization

- You’ve programmed classes in Java
- In general, what kind of functions exist with well designed classes
Function types

- Accessor
- Mutator
- Helper
- Predicate
CPP classes

- A class if a collection of functions and variables
- In CPP we have constructors and destructors to help manage classes
Order of running program

- In C we saw that the program always starts from main

- This is different in C++
What can go wrong

- The good thing about cpp is that your program can now crash many times even before reaching main 😊
Ordering and where to look for problems

- Global variables
  - Assignments and constructors
  - What else ?
- Main
- Local variables
- End local variables
- End main
- Global destructors
I’d like to cover a bunch of code examples now illustrating the power of classes.

Will start from simple array and work out a complex class.

We can do it in lab with string example…
Code simple version of point class

Add extensions with specific stuff
- Member functions
- Coding
- Pointers
Class friends

- allows two or more classes to share private members
- e.g., container and iterator classes
- friendship is not transitive
hierarchy

- composition:
  - creating objects with other objects as members
  - example: array4.cpp

- derivation:
  - defining classes by expanding other classes
  - like “extends” in java
  - example:
    ```cpp
    class SortIntArray : public IntArray {
    public:
    void sort();
    private:
    int *sortBuf;
    }; // end of class SortIntArray
    ```
- “base class” (IntArray) and “derived class” (SortIntArray)
- derived class can only access public members of base class
Class derivation

- encapsulation
  - derivation maintains encapsulation
  - i.e., it is better to expand IntArray and add sort() than to modify your own version of IntArray

- friendship
  - not the same as derivation!!
  - example:

  - is a friend of
  - B2 is a friend of B1
  - D1 is derived from B1
  - D2 is derived from B2
  - B2 has special access to private members of B1 as a friend
  - But D2 does not inherit this special access
  - nor does B2 get special access to D1 (derived from friend B1)
Derivation and pointer conversion

- derived-class instance is treated like a base-class instance
- but you can’t go the other way
- example:
  ```c
  main() {
    IntArray ia, *pia;
    // base-class object and pointer
    StatsIntArray sia, *psia;
    // derived-class object and pointer
    pia = &sia; // okay: base pointer -> derived object
    psia = pia; // no: derived pointer = base pointer
    psia = (StatsIntArray *)pia; // sort of okay now since:
    // 1. there’s a cast
    // 2. pia is really pointing to sia,
    // but if it were pointing to ia, then
    // this wouldn’t work (as below)
    psia = (StatsIntArray *)&ia; // no: because ia isn’t a StatsIntArray
  }
  ```
danger:

- don’t point a base class pointer to an array of derived objects!
- they aren’t the same size!
Const variables

- Can have const variables in a class

- Any ideas for this?
Operator overloading

- Most operators can be overloaded in cpp
- Treated as functions
- But it's important to understand how they really work
Look up list
Operators which cant be overloaded

- .
- *
- ::
- ?:
- sizeof
- \( X = X + Y \)
- Need to overload
  - +
  - =
- But this doesn’t overload +=
Functions can be member or non-member
Non-member as friends
If its member, can use this
(), [], -> or any assignments must be class members

When overloading need to follow set function signature
Code from fig18_03 (c book)
unary

- Y += Z
- Y.operator+=( Z )

- ++D
- member
  - D.operator++()
- Non member
  - operator++(D)
coding

- Lets overload the point + operator
- Now the =

- See it in action…