CSCI E-225 -- Object-Oriented Programming in C++ (23510)

Syllabus - Spring Term, 2011

Last Updated: January 27, 2011

Instructor:  Dr. Ken Basye
Location:  Science Center 110
Prerequisites:  A working knowledge of C, including structures and pointers, is assumed; substantial experience in C is preferred.
Books:
   Required:  None

Assignments and sections:
The teaching fellows will hold 1-1/2 hour discussion sections each week. These will not present new required material, but they will provide an opportunity to have your questions answered, to learn techniques that go a bit beyond what was done in lecture, and to get hints on the homework. Fill out the accompanying section form and hand it in during the break in the first lecture. Homework is due after the second week, and every two or three weeks thereafter. In addition to the homework, there will be a final project of your own design which will be due May 12th. There are no examinations.

Tentative schedule:
Thursday, Jan. 27 - Introduction, simple additions to C, basic stream I/O

Thursday, Feb. 3 - Introduction to OOP, classes, objects and members, access specifiers.

Thursday, Feb. 10 - Constructors and destructors, new and delete, references

Thursday, Feb. 17 – Const qualifier, simple exceptions.

Thursday, Feb. 24 - Operators, more stream I/O

Thursday, Mar. 3 - Namespaces, Introduction to templates

Thursday, Mar. 10 - More on templates, STL strings

Thursday, Mar. 17 - NO CLASS (Spring break)
Thursday, Mar. 24 - STL collections

Thursday, Mar. 31 - More STL: algorithms

Thursday, Apr. 7 - Inheritance

Thursday, Apr. 14 - More inheritance

Thursday, Apr. 21 – More exception handling

Thursday, Apr. 28 - OOP designs in C++

Thursday, May 5 – More Stream I/O, other libraries

Final projects are due on May 12th.

Assignment 0

Write a C++ program which reads in integers from cin until the user enters a non-integer (a character or EOF), then print out the average.

Submit the source code of your program as well as an execution transcript. (For more information see submission instructions on the website.)

Due midnight Friday, February 4.
CSCI E-225 - Spring 2009

- Introductions
- Course Infrastructure
- Policies and Procedures
- Poll
- Schedule

Course Infrastructure

- Website:
- Lectures and Sections
- Books
- Bulletin board
- Email
Policies and Procedures

• Homework due on Fridays at midnight
• Two one-week extensions, no questions asked…
• …but that’s all there is!
• Final project due May 12th – a Thursday; no extensions without an incomplete.

Policies and Procedures

• Homework to be done individually
• Compilers: MS Dev, G++, your choice.
• Editors: whatever you like
• Homework grading policies
Poll and Schedule

- Poll
- Schedule
  - Basics
  - Standard library
  - Inheritance
  - Design and some remaining topics

Object-Oriented Programming in C (!?)

- Abstraction
  - `fopen`, `fread`, `fprintf`, `fflush`, `fclose`
- Encapsulation
  - Header files and libraries
- Hierarchy
- Polymorphism
  - \((1+2) \text{ vs. } (1.2 + 2.3)\)
Hello World

// My first C++ program
#include <iostream>
int main()
{
    std::cout << "Hello, World" << std::endl;
    return 0;
}

SAME
• Preprocessor
• Program entry point
• Function syntax*

DIFFERENT
• Comments
• I/O mechanism

Comments

• // comments from there to end of line
• /** */ works as in C
• Avoid mixing; neither type respects the other:

// this --> /* is not effective
so this becomes an error */

/* this --> // is not effective */ so this is an error
Stream I/O

• `#include <iostream>`
• `std::cout, std::cin, std::cerr`
  These correspond to stdout, stdin, and stderr
• The `<<` and `>>` operators
  A mixed blessing
• `std::endl`
  It’s just the beginning.

---

Stream output sample

```cpp
// output.cpp : Simple output

#include <iostream>
using std::cout;
using std::endl;
int main(int argc, char* argv[]) {
    cout << "This program is called " << argv[0] << endl;
    cout << "This program was invoked with " << argc << " arguments." << endl;
    int minArgs = 2;
    cout << "This program did " << (argc >= minArgs?"":"not ") << " have enough arguments." << endl;
    cout << "minArgs = " << minArgs << endl;
    cout << "argc = " << argc << endl;
    cout << "Please pay $" << dollars << "." << cents << endl;
    double amount = 14.56;
    cout << "Please pay $" << amount << endl;
    return 0;
}
```

---
Stream input sample

```cpp
// input.cpp : Simple (cooperative) user input
#include <iostream>
using std::cout;
using std::cin;
using std::endl;
int main(int argc, char* argv[]) {
    cout << "Please type in an integer now...";
    int val = 0;
    cin >> val; // This may fail, in which case val will still be 0
    cout << "You typed in " << val << endl;
    cout << "Now enter several integers, separated by whitespace" << endl
        << "and type x when you finish" << endl;
    int sum = 0;
    // This reads until it fails, when user types something not
    // readable as an int (or eof in case of file redirection).
    while ( cin >> val )
    {
        sum += val;
    }
    cout << "That adds up to " << sum << endl;
    return 0;
}
```

C++ as a superset of C

• “A legal C program is a legal C++ program”

Except:

• Type and function declarations required
• New keywords
• Other details (as they arise)
C++ Keywords that are not C keywords

and  and_eq  asm  bitand  bitor  bool  catch  class  compl  const_cast  delete  dynamic_cast  explicit  false  friend  inline  mutable  namespace  new  not  not_eq  operator  or  or_eq  private  protected  public  reinterpret_cast  rest  return  throw  true  try  typeid  typename  using  virtual  wchar_t  xor  xor_eq

C++ you already know

• Control constructs
  for, while, if, switch, function call

• Native types
  int, float, double, char, void
  NEW: bool b = true;

• Type modifiers
  int arr[100], float* fp,
  unsigned char, long int

• Arrays and pointers
  C++ uses a different dynamic allocation syntax
User-defined types

• enum

    enum Suit { CLUB, DIAMOND, HEART, SPADE };
    Suit s; // enum tag not required
    s = CLUB; // of course
    s = (Suit) 1; // Ok, not recommended
    s = 1; // ERROR

    enum PrintFlags {COLOR=1, LANDSCAPE=2, TWOSIDE=4};
    // Range is [0:7]
    PrintFlags pf = COLOR | TWOSIDE;
    pf = (PrintFlags) 6;
    pf = (PrintFlags) 11; // Undefined!!

    switch ( s ) { // use a default label!!


User-defined types

• struct and class

    struct point { int x; int y; };

    point p1; // Note: struct tag not needed
    p1.x = 9; // and so on...

• structs and classes can also have functions
Using existing classes

```cpp
#include <iostream>
#include <string>
using std::string;
int main()
{
    string s1 = "This is a string";
    std::cout << "There are " << s1.size() 
        << " chars in " << s1 << std::endl;
    return 0;
}
```

The string class

- string has over 40 functions
- E.g.:

  bool empty(); // Is s empty?

  // substring
  string substr(int pos, int nchars);

  // inserts cs, returns the new string
  string insert(int pos, char* cs);
Adding functions to structs

```cpp
struct point
{
    int x;
    int y;
    double distance_to_origin()
    {
        return sqrt ((double) x*x + y*y);
    }
};  // don’t forget this semicolon

// Now we can do this:
point p;
p.x = p.y = 4;
cout << "p is " << p.distance_to_origin()
    << " from (0,0)" << endl;
```

A bit of OOP philosophy:

*First, try being a client.*

- Use existing classes where possible
  (this was much harder until fairly recently)
- Design new classes from the client perspective
  (Figure out the interface first)