


CSCI 1730 Systems Programming

C++

Crash Tutorial

C++: Motivation?

Working for Google:

» http://www.forbes.com/sites/quora/2013/06/05/are-programmers-in-cc-more-preferred-at-google-than-programmers-in-java/

Ranking of Languages:

- http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
- http://langpop.com/
- http://spectrum.ieee.org/computing/software/top-10-programming-languages
- http://redmonk.com/sogrady/2014/06/13/language-rankings-6-14/

What you should learn?

- http://mashable.com/2014/01/21/learn-programming-languages/
- http://tech.pro/blog/1885/top-10-programming-languages-to-learn-in-2014

C++ and C

C:

- **Low level (close to hardware)**
- **No runtime type info (it is compile time)**
- **Easy implementation**

C++:

- **Originally to add some OO functionality to C**
- **Attempt to be a higher-level language**
- **Now it's a totally different language**

C/C++ and Java

- **Similar syntax**
- **Basic Constructs Similar!**
  - » **If,**
  - » **Loops**
  - » **Function**
  - » **Switch**
  - » **recursion**

First the Trusty First Program

```
# include <iostream>

int main() // like plain.
{
    // Hey this is different! Stream like
    cout << "Hello world!" << endl;
    return 0;  // 0 is normal
}
```

```
{nike:maria:77} make -f Makefile.cpp
g++ -g -c Hello.cpp -o Hello.o
g++ -g -o Hello.out Hello.o
```

I/O C++ Style: iostream

- **Basic Classes:**
  - » iostream (cout,cin,cerr)
  - » ostringstream, istringstream.

I/O C++ Style
<< : The Output Operator

- **Overloaded, works with any type (built-in)**
  - » (so different from C, but java-ish)

```
int k = 2;
      double d = 4/5;
      char c = 'x';

      cout << k << endl;  // write an int
      cout << d << endl;  // write a double
      cout << c << endl;  // write a char
```

- **Chaining:**

```
x = y = z = 55;
      cout << "we are awesome: "  << y << z;
```


Namespace: and example input

```
// i/o example

#include <iostream>
using namespace std;

int main ()
{
  int i;
  cout << "Please enter an integer value: ";
  cin >> i;
  cout << "The value you entered is " << i;
  cout << " and its double is " << i*2 << ".\n";
  return 0;
}
```

- **using namespace std;**

namespace

- ***Namespaces group functions and variables under a prefix***
- **Analogous to Java Packages**
  - » (w/o access modification, or path restrictions)
- **Used to avoid name collisions.**
- **Declared by:**

```
namespace <name> {
}
```

- **All symbols (functions and variables) are under the prefix <name>.**
- **Symbols accessed by `<name>::symbol`**
- **Namespaces can be nested.**
- **You can omit the <name>:: when referring symbols from the same namespace, or a containing namespace.**

using

- **Analogous to Java import statements**
- **There are 2 forms:**
  - » `using <name>::symbol;`
  - » `using namespace <name>;`
- **The first form tells the compiler that `symbol` means `<name>::symbol`.**
- **The second form tells the compiler to look for `<name>::symbol` if it cannot find symbol in the current namespace.**
- **NEVER put using directives in header files!!! Bad Form!!**
- **All standard library symbols are in the namespace `std`**

I/O C++ Style or C Style?

Hmmm….

- `printf("%.3f rounded to 2 decimals is %.2f\n", 2.325, 2.325:`

- `cout << setprecision(3) << 2.325 << " rounded to 2 decimals is " << setprecision(2) << 2.325 << endl;`

C and C++ I/O compared

- C-style I/O:
  - » No type safety: printf("%d", 'c');?
  - » Conversion specifications have a high learning curve.
  - » Almost all the state of the I/O is contained in the function call.
- C++ style I/O:
  - » Manipulators are very verbose/annoying
  - » Global state gets changed.
    - – When you do "cout << 2.4555", what precision are you set at? You don't know. It's worse with threads.
- You get more customizability since C++ I/O is classed based.
- Should not really mix the two Styles (buffers are not synchronized)

Dynamic Memory

- Dynamically sized memory in both C and C++ is manually managed (allocated and freed)
- Allocate:
- Free:
  - » Do not free memory twice (double free).
  - » Do not free memory that has not been
- Manual memory management allows for finer grained control of your program

new, delete, delete[]

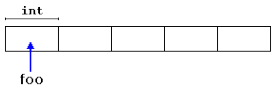
- the new operator allocates new memory, initializes it and returns a pointer to it.
- the delete operator deallocates memory allocated by new
- If you allocate a new array, you must delete it with delete[] and not delete

```
Point2D *p = new Point;
delete p;
p = NULL;

int *ar = new int[50];
delete[] ar;
ar = NULL;
```


new, delete, delete[]

```
int * foo;
foo = new int [5];
```



- Returns a pointer to the first element
- No guarantee memory will be granted, What to do:
  - » 1) Exception handling. (similar to java)
  - » 2) No throw
    - – Return a null pointer (check it).
    - – foo = new (nothrow) int [5];

Example: remember-o-matrix.cpp

```
int main ()
{
  int i,n;
  int * p;
  cout << "How many numbers would you like to type? ";
  cin >> i;
  p= new (nothrow) int[i];
  if (p == NULL)
    cout << "Error: memory could not be allocated";
  else
  {
    for (n=0; n<i; n++)
    {
      cout << "Enter number: ";
      cin >> p[n];
    }
    cout << "You have entered: ";
    for (n=0; n<i; n++)
      cout << p[n] << ", ";
    delete[] p;
  }
  return 0;
}
```


What about Alloc, malloc recalloc?

- Defined in <cstdlib> (stdlib.h in C)
  - » As with printf better not to mix new, and C style memory allocation
  - » http://www.cplusplus.com/reference/cstdlib/
- Not typesafe returns (void *) so need to cast.
- May be faster, more efficient.

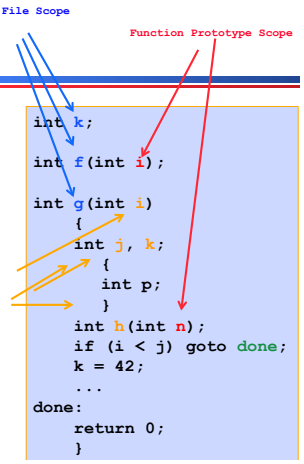
Declaration and Definition

- **Declaration**
  - » Name of variable / structure
- **Definition**
  - » Name, and
  - » How to store the data structure/variable
    - – Function and objects need this to be allocate
    - – Includes the body of the function
  - » typedef and enumeration constants

```
// declaration only
struct wasp;
```

```
// a definition
// declaration and definition
struct wasp
{       int head;
        int legs;
        int eyes;
};
```


C/C++ Scope (standard)



1. **Function Prototype**
   Declared in list but doesn't include its definition)then it is block scope)
2. **Function**
   Statement labels before it is declared) .. '
   Ex. **Done** (visible before it is declared)
3. **File** (sometimes called global scope)
   Outside blocks; functions, structures,
4. **Block** (sometimes called local scope)
   - » Declared within function definition/blocks
   - » Includes the parameter list

C++ (1&2) above, + the below:

- Local : C's Block scope
- Name Space : C's File Scope
- Class : New

C++/C Standards explained by DR Dobbs
http://www.drdobbs.com/cpp/scope-regions-in-c/240002006

http://www.drdobbs.com/cpp/scope-regions-in-c/240002006?pgno=2

Scope (cont)

- Begins at the end of its declaration, and before it is initialized
  - » More Complex structures closely after its 'tag'
- C++ additions (plain C):
  - » Declarations within loop definitions (block scope)
  - » Namespace (global scope, one word).
    - – Declared either in namespace or in the
    - – C's equivalent of file scope (imports it into scope)
  - » Generalizes the rules of file scope to include names.

```
long int *p = NULL, x[N];

struct stag     enum etag
       {               {
      ...             ...
       };              };


namespace identifier
     {
     ...
     }
```


Scope (cont)

- **Class scope**
  - » Declared within class definition
  - » Classes in C++ includes Structures & Unions
- **Translational units**
  - » A file after the preprocessor processing (includes the includes).
  - » Globals are available within the 'translational units'
  - » 'Globals' can extend its scope by 'external linking'

http://www.learncpp.com/cpp-tutorial/42-global-variables/

Focus on Global Variables

- **Global Variables are not evil!**
  - » Allocated at program start.
  - » De-allocated at program end.
- **By default (should) initialized to bit-wise zero**
- **Next: Modifiers:**
  - » `extern`,
  - » `static`, and
  - » `const`

Global Variables - Gotcha I

What is wrong with this code?

```
/* util.c */

int g_numCalls = 0;

void someFunc(void) {
  fprintf(stderr,
    "Num Calls to %s: %d\n",
    __func__, g_numCalls);
g_numCalls++;
  ...
}
```

```
/* test.c */
void someFunc(void);
int g_numCalls = 0;

int main(void) {
   fprintf(stderr,
     "Num Calls to %s: %d\n",
     __func__, g_numCalls);

   someFunc(); someFunc();
   ...
}
```

compile line: `gcc -Wall util.c test.c -o test`

Static

- On a global variable or a function:

```
static int g_someGlobalVariable;
static void myFunction(void);
```

Tells the linker not to export the variable or function.

» Ensures the identifier remains in "file scope,"

– The linker will not use it fulfill dependencies from other files.

- On a local (function) variable:

```
void someFunc(void) {
    static int array[4000];
}
```

» Places the variable off the stack.

– This has the side-effect that it retains it value across calls. It is often used when a variable is too large to be put on the stack.

- On class member (later)

Global Variables - Gotcha I fix A.

```
/* util.c */

static int g_numCalls = 0;

void someFunc(void) {
  fprintf(stderr,
    "Num Calls to %s: %d\n",
    __func__, g_numCalls);
g_numCalls++;
  ...
}
```

```
/* test.c */
void someFunc(void);
static int g_numCalls = 0;

int main(void) {
   fprintf(stderr,
     "Num Calls to %s: %d\n",
     __func__, g_numCalls);

   someFunc(); someFunc();
   ...
}
```

Static // Now the two variables "g_numCalls" have no relation. Think of them as private to each file.

Global Variables - Gotcha II

What is wrong with this code?

```
/* debug.h */

int debug_level;
…
```

```
/* test.c */

#include "debug.h"
…
```

```
/* debug.c */

#include "debug.h"
…
```

```
/* otherfile.c */

#include "debug.h"
…
```

compile line: `gcc -Wall -ansi *.c -o test`

(this actually works with the latest gnu compiler)

Global Variables - Gotcha II (bad fix)

static with fix the compile [error]…

```
/* debug.h */

static int debug_level;
…
```

```
/* test.c */

#include "debug.h"
…
```

```
/* debug.c */

#include "debug.h"
…
```

```
/* otherfile.c */

#include "debug.h"
…
```

compile line: `gcc -Wall -ansi *.c -o test`

…but get 3 distinct copies of debug_level …

extern

- Avoids the extra allocation of variables
  - » Declare
    - – `int debug_level`
  - » and avoid allocate space for it every time
- `extern int debug_level;`

  "there exists an int called debug_level, but the storage is elsewhere, Go and find it linker."

- Function Prototypes are
  - » implicitly declared as `extern`
- As with prototypes, you must remember to actually declare the 'real' variable

const

Operator Overloading (covered in Hackathon)

- Enables you to define operators (`+,-,*,/`) for new types.
- Operators are usually binary, or unary functions.
  - » Examples of unary operators:
    `!false, -i`
  - » Examples of binary operators
    `a + b, isBig || isRed`
- The syntax for referring to the operator function is `operator<symbol>`
  - » Example: prototype for integer addition (+):
    `int operator+(int lhs, int rhs);`

Example: Operator Overloading

```
Point2D
operator+(const Point2d &lhs, const Point2d &rhs)
{
        Point2D p;
        p.x = lhs.x + rhs.x;
        p.y = lhs.y + rhs.y;

        return p;
}
```


friends

- Classes and functions can be declared as friends by writing
  - » `friend <classname>;`
  - » `friend <function prototype>;`
- Friends have access to a class's private members.
- *Only friends can touch each other's private variables*

Classes

- Roots come from C-style structs
- Fairly similar to Java classes in concept.
- Used to group related data and functions.
- Can be used to write OO code

Great Resources

- *Programming -- Principles and Practice Using C++ (Second Edition) Bjarne Stroustrup (2014)*
- *C++ for Java Programmers, by Timothy Budd*
- *I think well of Thinking in C++ by Bruce Eckels.*
- *Accelerated C++: Practical Programming by Example, Andrew Koenig and Barabra Moo (2000)*

- *http://www.cs.washington.edu/orgs/acm/tutorials* *(Links)*
- http://pages.cs.wisc.edu/~cs368-2/CppTutorial/ (Today)
- http://cs.brown.edu/courses/cs123/docs/java_to_cpp.shtml' http://www.cprogramming.com/java/c-and-c++-for-java-programmers.html
- Wikipedia/References:

http://en.wikipedia.org/wiki/Comparison_of_Java_and_C%2B%2B

Quiz: Last Name, First Name - LAB

1. With regard to UNIX what what was the most interesting topic
2. With regard to UNIX which was the least interesting.
3. Did we miss a Topic in UNIX that you were interested in?
4. What grading issue do you have not been addressed

Struct vs. Class in C++
Class is more private than Struct

- Default scope of members:
  - » Class : private by default and
  - » Struct: are public by default.
- Default access specifier when Deriving IT.
  - » when deriving a class, default access specifier is private.
  - » When deriving a struct, default access specifier is public.

Java vs. C Parameter Passing

- Review: Pass by Value – copy of the parameter is passed
- Review: Pass by Reference – pass the 'address; of the variable (still copies but there reference may not be copied).
  - » In Java copy of the address for large structures, but it refers to the same address as the original reference.
  - » In C (large) structures are automatically copied if passed by reference.
    - – In java need to use & to send it to the method get the same effect (use * within function).
    - – IN C++ use & in parameter list to indicate it is passed by reference. (in plain C you would use a *).

- See example. (pass by value .cpp)

Templates and STL

- http://msdn.microsoft.com/en-us/magazine/cc163754.aspx
- Template examples (see schedule page)
  - » template <typename T>
  - » Function/method template (square.cpp, squareT.cpp)
  - » Class template (MVector.cpp)
- Review STL here (on your own). http://www.mochima.com/tutorials/STL.html

Closing C++ vs. Java

- Review / Read:
- http://en.wikibooks.org/wiki/C%2B%2B_Programming/Programming_Languages/Comparisons/Java

Schedule

- We: Hackathon 11 (optional)
  - » C++/C
  - » Grade Concerns (record).
- Th: Showcase, more grade concerns?