New Jersey's Science &
Technology University

THE EDGE IN KNOWLEDGE

CS 280
Programming Language Concepts

Hello, World

```
//============================================================================
// Name        : HelloWorld.cpp
// Author      : gwryan
// Version     :
// Description : Hello World in C++
//============================================================================
// stuff that begins with // is a comment!

/* this is also a comment */

#include <iostream>
using namespace std;

int main(int argc, char *argv[]) {
        cout << "Hello there, C++ programmers!" << endl;
        return 0;
}
```

classes

- Java programmers: notice that there is no “class” used in this program
- In C++, everything does NOT have to be in a class
- We don’t need a class for this simple example, so we just write some functions

functions

- Defining a function requires you to specify
  - function name
  - type that the function returns when it is done
  - arguments that are passed in to the function
  - the body of code that is executed when the function is called

main

- Every C++ program must have one main function
- The main function is where the program starts, and when main finishes the program is done

```
// the code below says
// main is the name of the function
// the type that it returns is int
// main takes two arguments named argc and argv
// the two statements inside of the { } are run when main gets called

int main(int argc, char *argv[]) {
        cout << "Hello there, C++ programmers!" << endl;
        return 0;
}
```

Preprocessor

- #include is a preprocessor directive
- It tells the compiler to include the contents of a file into the body of the code before compiling
- A filename inside of < and > means “in the standard place for standard files that came with the compiler”
- You can make your own include files and #include them, enclosing filenames in “ and “

Header or Include Files

- C and C++ rely on including files in the preprocessor phase of the compile to make sure that all programs have common definitions of things: variables, constants and different types
- Header files usually go along with library implementations
- C++ comes with a lot of standard libraries and header files

using namespace

- C++ can put functions and definitions into a separate, labeled namespace
- To access something in a namespace, you need to preface the item you are accessing with the name of the namespace and ::
- Items in the C++ standard library are in a namespace named “std”, so cout is actually std::cout
- Saying “using namespace std;” tells the compiler that you want to have everything in the std namespace visible in your program

iostream

- iostream is a C++ standard library that provides a definition and implementation for input streams (istream) and output streams (ostream)
- At runtime every program has:
  - “standard input” or “standard in”
  - “standard output” or “standard out”
  - “standard error” or “standard err”
- To read what a user types in, read the standard in
- To write something for the user to see, write to standard out

iostream

- The iostream library provides a type-safe way to access standard in and standard out.
- cin is standard in
- cout is standard out
- cerr is standard err

- iostream uses the << operator to write to a stream
- iostream uses the >> operator to read from a stream
- There are also methods defined for various operations on streams

```
// the line below performs the << operation on cout
// stream << something
//     causes the something to be written to the stream
//     in this case, the something is a string of characters
// << endl
//     causes and end of line to be written to the stream
//     if you leave off endl you will not skip to the next line
//     writing “\n” or ‘\n’ does the same thing

       cout << "Hello there, C++ programmers!" << endl;
```

operator << and >>

- The implementers of iostream decided to use the << operator (logical bit shift) to mean “shift information onto the stream”
- The >> operator means “shift information in from the stream”
  - So this code reads in an integer:

```
int x;
cin >> x;
```

- Reusing operators in this way is called “operator overloading”. It’s a feature of C++

Errors with streams

- Several errors are possible:
  - "End of file" is reached
  - The input on the stream cannot be converted to the proper type
- You need to check for errors!

Methods for errors

- Stream.good() is true if there are no errors
- Stream.eof() is true if the end of file was reached
- Stream.fail() is true if there was a logical error or a read/write error on the stream
- Stream.bad() is true if there is a read/write error on the stream

Streams without error checking

```
/*
 * streams.cpp
 *      Author: gwryan
 */

#include <iostream>
using namespace std;

int
main(int argc, char *argv[])
{
        int     ival = 0;

        for(;;) {
                cin >> ival;

                cout << "You entered " << ival << endl;

                if( ival == -1 )
                        break;
        }

        cout << "Bye!" << endl;
        return 0;
}
```

Streams with error checking

```
int
main(int argc, char *argv[])
{
        int     ival = 0;

        for(;;) {
                cin >> ival;

                cerr << "good: " << cin.good() << " eof: " << cin.eof()
                << " fail: " << cin.fail() << " bad: " << cin.bad() << endl;

                if( cin.eof() )
                        break;

                if( cin.fail() == false )
                        cout << "You entered " << ival << endl;

                if( ival == -1 || cin.fail() )
                        break;
        }

        cout << "Bye!" << endl;
        return 0;
}
```

Some istream methods

- get - read a single character
- get – read a sequence of characters into an array of characters
- getline - read a line (a sequence of characters terminated by a newline) into an array of characters
- Also a getline function to read from a stream into a “string”

Read input a character at a time

```
#include <iostream>
#include <fstream>
using namespace std;

int main(int argc, char *argv[])
{
        int    ch;

        while( (ch = cin.get()) != EOF ) {
                cout.put(ch);
        }


        return 0;
}
```

Read input a line at a time

```
#include <iostream>
#include <string>
using namespace std;

int main() {
        string inLine;

        for(;;) {
                getline(cin, inLine);

                if( !cin.good() )
                        break;

                cout << "You typed:" << inLine << endl;
        }

        return 0;
}
```

NJIT

THE EDGE IN KNOWLEDGE