Review

Why do we use protection levels?

Why do we use constructors?

How can we tell if a function is a constructor?

What is a default constructor?

How do we pass arguments to a constructor?

Unix Environment and Compiling

Computer Science Accounts

All your programs should compile and run correctly on the departmental programming server linprog.cs.fsu.edu.

- Actually 4 different computers (linprog1, linprog2, ...)

If you do not have an account, the CS Systems Group has instructions for obtaining one.

To access the departmental server (Windows Users) should download an Secure Shell Client (SSH) from http://its.fsu.edu/Software/SoftwareLicensing/SSH

Mac and Linux users should have SSH built in. Just open a terminal (Terminal-app on Mac) and type:

- ssh -X youraccountname@linprog.cs.fsu.edu

Basic Unix Commands

ls – list files and directories in current directory

cd <directory> - change current directory to <directory>

cd .. – change directory to next up in hierarchy

cd ~ - change directory to your home directory

mkdir <dir> - create sub-directory <dir> in current directory

rm <filename> – remove file <filename>

rm –R <directory> – remove directory and all its subdirectories

cat <filename> - output text of <filename> to screen

chmod +x filename

more <filename>- similar to cat, allows scrolling

man <keyword> - displays manual page for various unix and programming concepts

Basic Unix Commands

After you login to linprog, do the following

```
$ mkdir  cop3330                  // create a directory for cop3330
$ls
…                                 // you should see cop3330 among other files
$cd  cop3330                      // change directory to cop3330
$mkdir  proj1                     //  make a proj1 directory
$ls
proj1/                            // this is what you should see
$cd  proj1                        // change directory to proj1
$mkdir  sample                    // create a directory called sample
$mkdir  myproj1                   // create a directory called myproj1
$ls
myproj1/    sample/               // this is what you should see
$cd ..                            // go up one level, 'cd ~' goes to the home directory
$ls
proj1/
$cd  proj1
$ls
myproj1/    sample/
```

Transferring Files To/From

In Windows:

The SSH client has a built in file transfer utility

- Connect to linprog, click the File Transfer Utility button on the SSH client, your machines files are on the left, linprog's file system is on the right, you may drag and drop files back and forth

In OS X/Linux:

Use the scp/sftp tools

- To copy files to linprog type 'scp <localfilepath> yourusername@linprog.cs.fsu.edu:~/where/you/want/it'
- To copy files from linprog type 'sftp linprog.cs.fsu.edu' and then 'get <filename>'

Transferring Files To/From

Exercise with Windows SSH client

- Download all project 1 sample files to the local machine and store them somewhere
- Start SSH client and connect to linprog.cs.fsu.edu
- Open the file transfer utility, you should see cop3330 in the right window.
- Click on cop3330, you should see proj1,click on it, you should see sample and myproj1, click on sample
- You should now see an empty right window
- Drag all four project file1 files to the empty window one by one
- You should now see four files in the window

Basic UNIX Commands (cont)

```
$ls
myproj1/    sample/                    // this is what you should see
$cd sample
$ls                                    // check what is in there
proj1.docx  proj1_linprog.a   testcase0_out.txt   testcase0.txt
$ cat  testcase0.txt                   // look what is in the testcase0.txt
S13S12S11S10S1
D5D4D3D2D1
….
$ cat  testcase0_out.txt               // look what is in testcase0_out.txt, you can replace cat with more
Loyal flush: S13S12S11S10S1
Straight flush: D5D4D3D2D1
…
$ chmod +x proj1_linprog.a      // make file proj1_linprog.a executable, this is important
$./proj1_linprog.a              // now run the program, important to type "./" before the file
S13S12S11S10S1                  // you have type in this line and type return
Loyal flush: S13S12S11S10S1
                                // type control-D here to finish the program
```

Basic UNIX Commands (cont)

```
$ ./proj1_linprog.a < testcase0.txt          // redirect the input from a file
Loyal flush: S13S12S11S10S1
Straight flush: D5D4D3D2D1
Straight flush: C8C7C6C5C4
Four-of-a-kind: S11D11C11H11D8
Full house: S13J13S12D12H12
Flush: C13C4C10C2C5
Straight: C12H11S10D9D8
……..
```

UNIX Editors

Many editors are available in UNIX

- emacs, vi, vim, pico, etc

- Just pick one and be good at it.
  - Some key functions you would need.
    - Go to line (and/or display line number)
    - Search for a string
    - Search and replace

Multiple File Projects

Although any program can be written in a single file, we often separate a program into several files:

Header File – contain class declaration, ends in .h (circle.h)

Implementation File – contains class definition (class member function implementations), ends in .cpp (circle.cpp)

Driver File – contains main() and possibly other subroutines used in the program, ends in .cpp (driver.cpp)

Many of the programs you submit in this class will have the above format.

Multiple File Projects continued

Reasons for using multiple files:

- Allows changes to the class implementation without affecting the class interface/structure
- Allows a class to be used in multiple driver programs
- Increases the opportunities for code reuse
- Code becomes more *modularized.*
- It is infeasible to code a large project in a single file

Compiling

Compiling is the process of converting a C++ language file into a language the computer understands (machine language)

Compilation takes place in two primary phases:

- Compile Phase – Responsible for language translation.
- Linking Phase – Responsible for resolving references across files.

Compile Phase

Performs **preprocessor directives** (#include, #define, etc.)

Checks syntax of program (e.g. missing semicolon).

Checks if functions/variables declared before use.

Does **NOT** check if functions that are used are defined.

Creates machine code representation of file known as an object file (term not associated with ‘object-oriented’, customarily ends with .o extension).

An object file is not executable and may have functions that are used but have no definition (implementation).

Linking Phase

Links the object code/file(s) into a single executable program.

The linking stage is the time when function calls are matched up with their definitions, and the compiler checks to make sure it has one, and only one, definition for every function that is called.

All class member functions must also be defined during this phase.

There must be a main() function (executable must know where to start).

Example: Compiling multi-file project

See Fraction example again.

g++ -c Fraction.cpp

- performs compile stage on Fraction.cpp and creates object file Fraction.o.

g++ -c main.cpp

- performs compile stage on main.cpp and creates object file main.o.

g++ main.o Fraction.o –o main.x

- links object files, code in main.o which references member functions of the Fraction class are **linked** to the definitions compiled in Fraction.o.

Shortcut:

g++ main.cpp Fraction.cpp -o main.x

- Performs compile and link phase in one step.

Exercise

make a directory called lect1 under cop3330
Put sample1.cpp in our first lecture in the directory
Change directory to cop3330/lect1
$ls
sample1.cpp
$more sample1.cpp
…… // display the content of the file
// you need to type a key to see the whole thing
$g++ sample1.cpp // compile the program
$ls
a.out sample1.cpp // a new file is there, a.out is the executable for sample1.cpp
$./a.out
…. // see the output
$ g++ sample1.cpp –o myprog
$ls
a.out myprog sample1.cpp // you just create a new executable called myprog
$./myprog
… // see the output
$ rm a.out // remove a.out
$rm myprog // remove myprog
$ls
sample1.cpp

In the project, you basically need to create a proj1.cpp file that has not errors.

Exercise

After you complete the project with all files in under cop3330/proj1/myproj1
Assume you just login
$cd cop3330/proj1 // change directory to cop3330/proj1
$tar cvf smith_j_proj1.tar myproj1 // create a tar file name smith_j_proj1.tar

Use the ftp window to drag smith_j_proj1.tar to your local machine
Upload smith_j_proj1.tar from your local machine to the blackboard

Make sure that you remove all unnecessary files in myproj1 directory before tar

If you have a tar file (xxx.tar) in UNIX, you can run the following command to expand the file

$ tar xvf xxx.tar // this extract the content of the tar file.