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?

How do we set default parameters for a constructor?

Does this work on other (non-constructor) functions?

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 here.

To access the departmental server (Windows Users) should download an Secure Shell Client (SSH) like this one.

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

more <filename>- similar to cat, allows scrolling

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

Emacs

Emacs is a powerful Unix text editor that you can use to edit your programs on linprog

Type ‘emacs <filename>’ to open/create file <filename> for editing

X11 is a windowing system that allows linprog to display windows on your local machine. **X11** comes with OS X and Linux. Windows users can download here.

If **X11** is running, you may type ‘&’ at the end of the command to retain terminal access while emacs is running (‘&’ is the background modifier for Unix commands)

Emacs Tips

I have posted a file called **.emacs** in the sample directory, you can copy this to your home folder to turn on line numbering and syntax highlighting in emacs

In the windows SSH client, you may want to edit the preferences to send delete when backspace is typed or backspace will not work as intended

Commands:

- You can use ctrl-k to cut multiple lines an ctrl-y to paste them
- ctrl-x-s is used to save the file (Hold ctrl, type x and release x, type s and release s, then release control)
- ctrl-x-c quits emacs
- ctrl-g aborts a command if you find emacs has ‘frozen’
- ctrl-s allows you to search the file for a keyword
- You can find other emacs commands here

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

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.

Doest **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.