CSE 384

Introduction to C/C++

Spring 2020

C Language (Background)

- C Programming Language
  - Originally developed in 1972 by Dennis Ritchie
    - AT&T Bell Labs as a systems programming language
      - Unix (and Linux) operating system written in C.
  - high-level (imperative/procedure) language, designed to give the programmer a lot of control while still emphasizing portability
    - Before C, operating systems were often written in assembly languages tied specific machine architectures, greatly limiting portability.
  - 1973: Ritchie and Ken Thompson rewrote Unix in C
    - Unix and C Language became extremely successful.
      - There are used everywhere today in 2020!

C Language: Background (cont.)

- 1978: Brian Kernighan and Dennis Ritchie published “The C Programming Language” (2nd edition 1988)
- American National Standards Institute (ANSI) C89 standard: ANSI C
- 1990: the International Organization for Standardization (ISO) adopted ANSI C (with a few minor modifications): C90
- 1999: the ANSI released C99.
  - adopted many features which had already made their way into compilers as extensions (or had been implemented in C++)
- 2011: C11
- 2018: C18 (latest C standard)

Source: https://www.learncpp.com/cpp-tutorial/introduction-to-cplusplus/

C++ Programming Language (background)

- 1979: C++ (C plus plus)
  - developed by Bjarne Stroustrup, AT&T Bell Labs as an extension to C
  - 1998: standardized by ISO committee, with major updates C++11, C++14, C++17
    - C++20 in works
- Modern C++ is a very powerful and expressive (federation) of language: a multi-paradigm programming language
  - Imperative/procedural (its C Language roots)
  - Objected-oriented - classes (encapsulation and information hiding, inheritance, aggregation, composition, dynamic type binding and polymorphism)
  - Generic – templates (specialization, custom processing policies), functional template metaprogramming
  - Standard Template Library (STL) - Data structure containers, algorithms, functions, and iterators

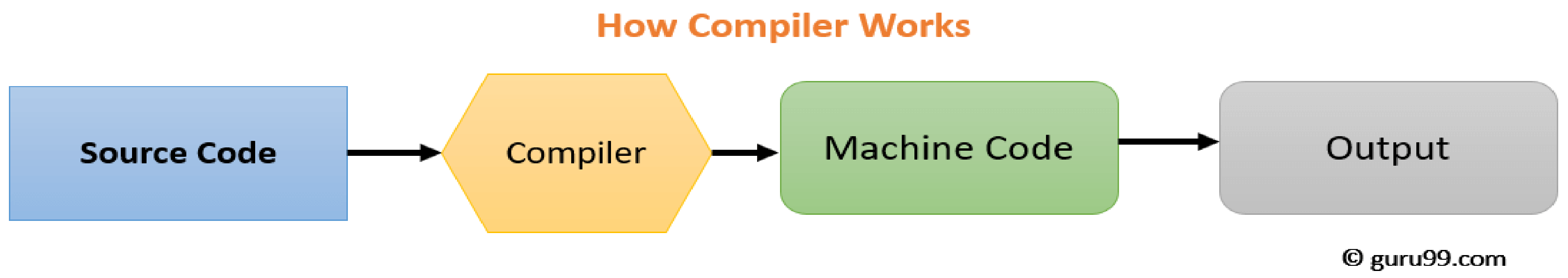
C++ Programming Language (background)

- Although C and C++ have evolved independently, there remains a high degree of compatibility.
  - C is *essentially* a subset of the C++
- In this class we intend to use both C and C++.
- The intent is to use aspects of C++ to facilitate the use of C libraries and functions needed for system programming on Linux.
  - The challenge with using C++ is that is a complex language. As mentioned earlier, although it is very expressive, but you have to know concepts about the object model to use it effectively, which could be an entire course unto itself (see Dr. Fawcett's previous SU graduate course on Object-oriented Design using C++: https://ecs.syr.edu/faculty/fawcett/handouts/webpages/cse687.htm)
  - And his recent github site: Modern C++ story here: https://jimfawcett.github.io/CppStory_Prologue.html
- The intent is to help you gain a foundational working model, and the proficiency to perform system programming tasks in C and C++.

Nice reference https://www.learncpp.com/cpp-tutorial/introduction-to-cplusplus/

C/C++ Programming: First things first…

- C and C++ are compiled languages (unlike interpreted languages such as Python, JavaScript, and BASH scripting etc.)
  - Compiled means that source code is (translated/compiled) into native machine code which can be executed directly on the hardware.
  - Scripting languages are NOT compiled, but rather depend on require an interpreter (another program) to run the instructions.



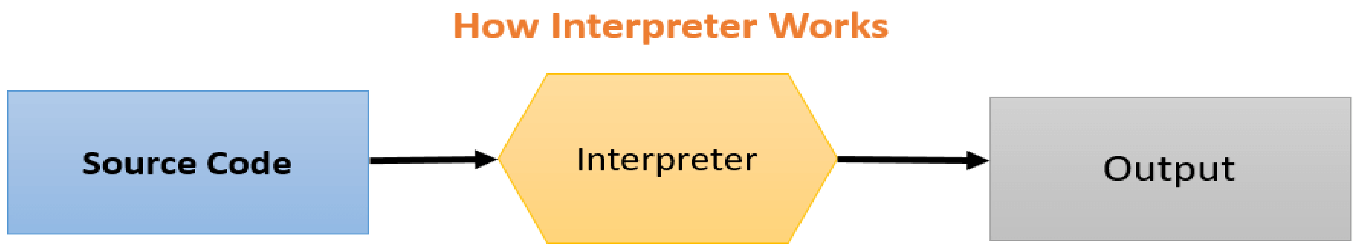
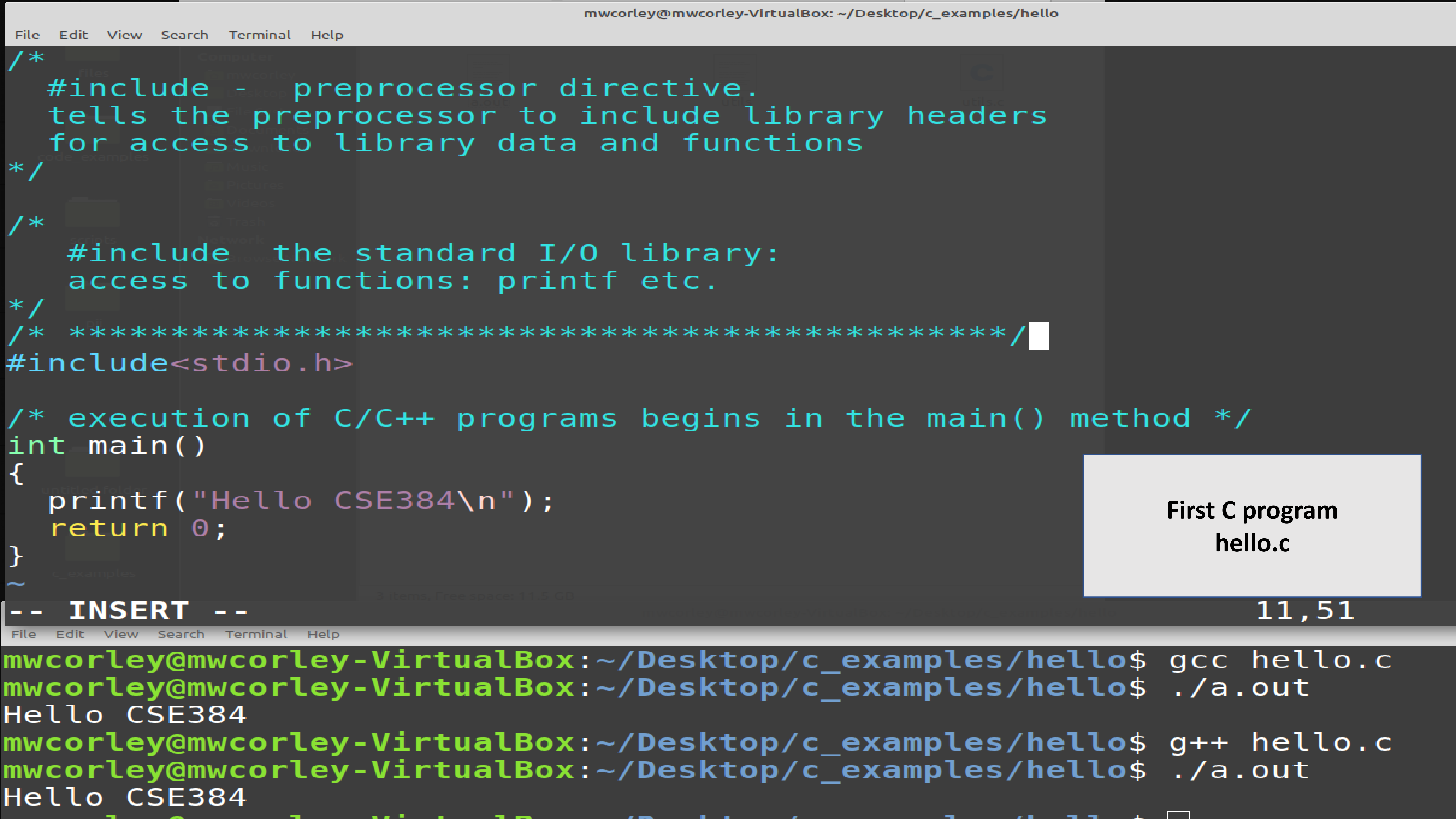


Image Source: https://www.guru99.com/difference-compiler-vs-interpreter.html

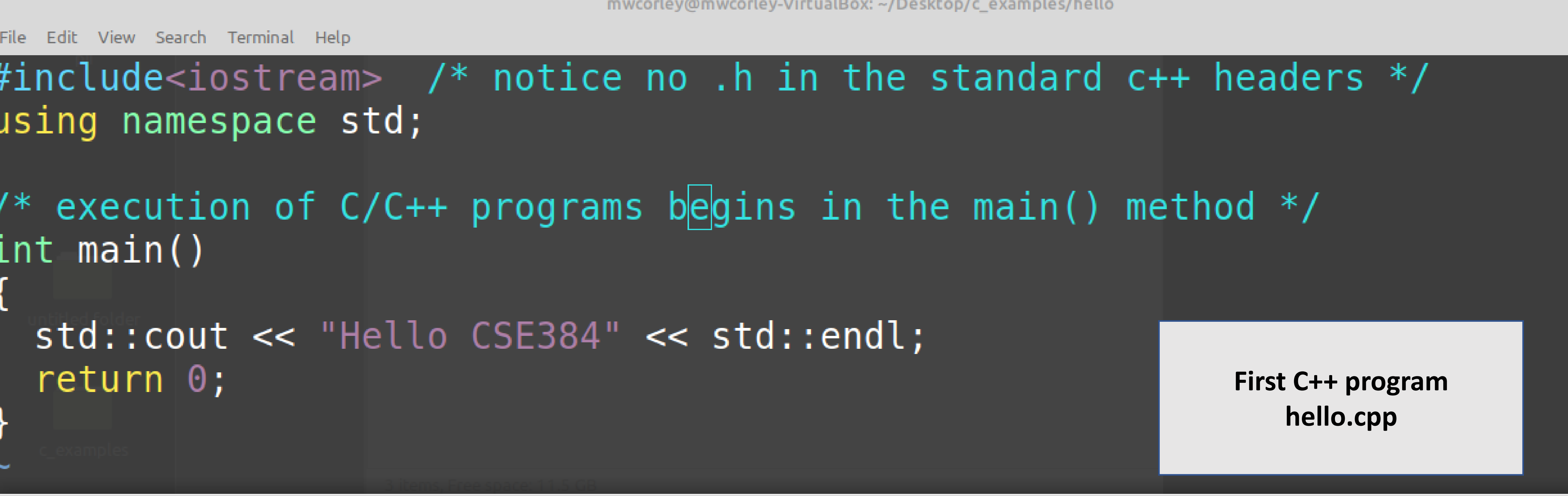

mwcorley@mwcorley-VirtualBox: ~/Desktop/c_examples/hello
File Edit View Search Terminal Help

```
/*
   #include -  preprocessor directive.
   tells the preprocessor to include library headers
   for access to library data and functions
*/


/*
    #include  the standard I/O library:
    access to functions: printf etc.
*/
/* ***********************************************/
#include<stdio.h>

/* execution of C/C++ programs begins in the main() method */
int main()
{
  printf("Hello CSE384\n");
  return 0;
}
~
-- INSERT --                                              11,51
```

First C program
hello.c

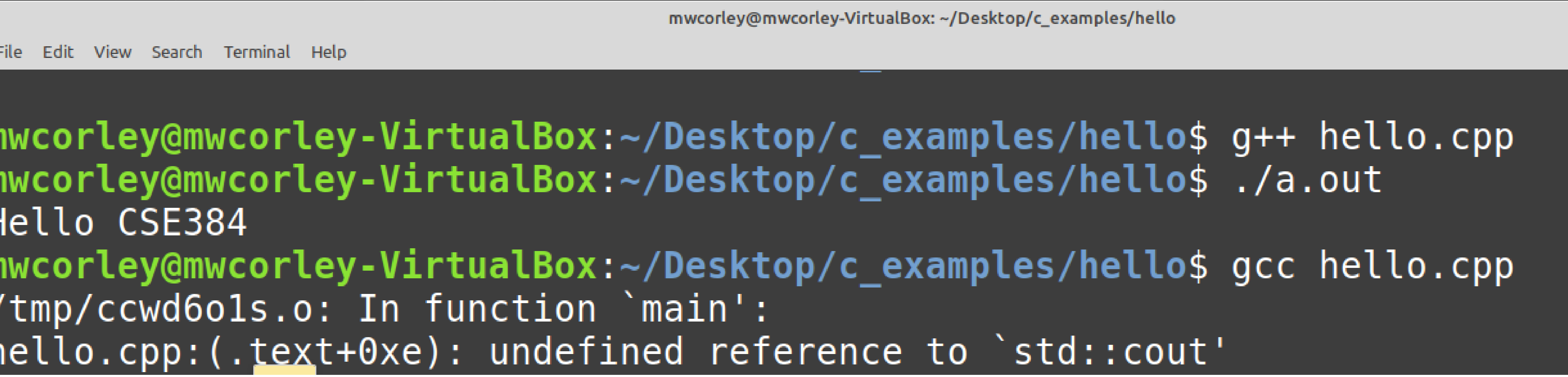
File Edit View Search Terminal Help

```
mwcorley@mwcorley-VirtualBox:~/Desktop/c_examples/hello$ gcc hello.c
mwcorley@mwcorley-VirtualBox:~/Desktop/c_examples/hello$ ./a.out
Hello CSE384
mwcorley@mwcorley-VirtualBox:~/Desktop/c_examples/hello$ g++ hello.c
mwcorley@mwcorley-VirtualBox:~/Desktop/c_examples/hello$ ./a.out
Hello CSE384
```


mwcorley@mwcorley-VirtualBox: ~/Desktop/c_examples/hello
File Edit View Search Terminal Help

```
#include<iostream>  /* notice no .h in the standard c++ headers */
using namespace std;

/* execution of C/C++ programs begins in the main() method */
int main()
{
  std::cout << "Hello CSE384" << std::endl;
  return 0;
}
~
```

First C++ program
hello.cpp


mwcorley@mwcorley-VirtualBox: ~/Desktop/c_examples/hello
File Edit View Search Terminal Help

```
mwcorley@mwcorley-VirtualBox:~/Desktop/c_examples/hello$ g++ hello.cpp
mwcorley@mwcorley-VirtualBox:~/Desktop/c_examples/hello$ ./a.out
Hello CSE384
mwcorley@mwcorley-VirtualBox:~/Desktop/c_examples/hello$ gcc hello.cpp
/tmp/ccwd6o1s.o: In function `main':
hello.cpp:(.text+0xe): undefined reference to `std::cout'
```

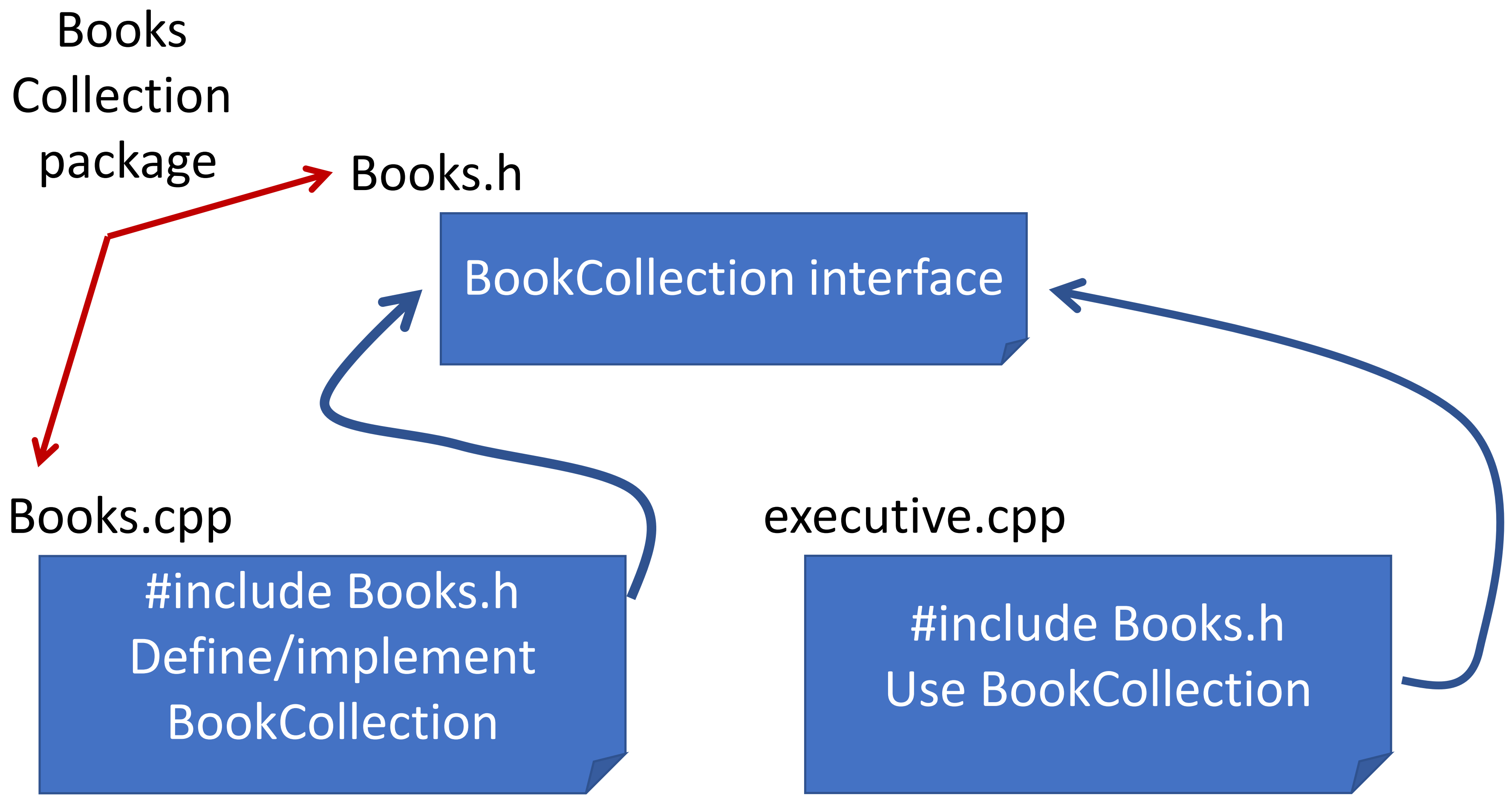
#include …

- A processor directive for including library (headers) in your program
    - think of import in python: same (conceptual) idea
  - Basically the same process in C and C++
    - In C, all headers end in .h (header):
      - e.g.   #include <stdio.h>, #include “stack.h”
    - In C++ only “user include” headers end in .h:
      - #include <iostream>, #include “stack.h”
        - Notice: standard C++ headers omit the “.h” in the include directive
  - Standard includes: <iostream>, <stdio.h>, etc.
    - Compiler (preprocessor) searches standard include paths: /usr/include etc.
  - User includes
    - search the user specified header path: include “./myheaders/string.h”

Separate compilation

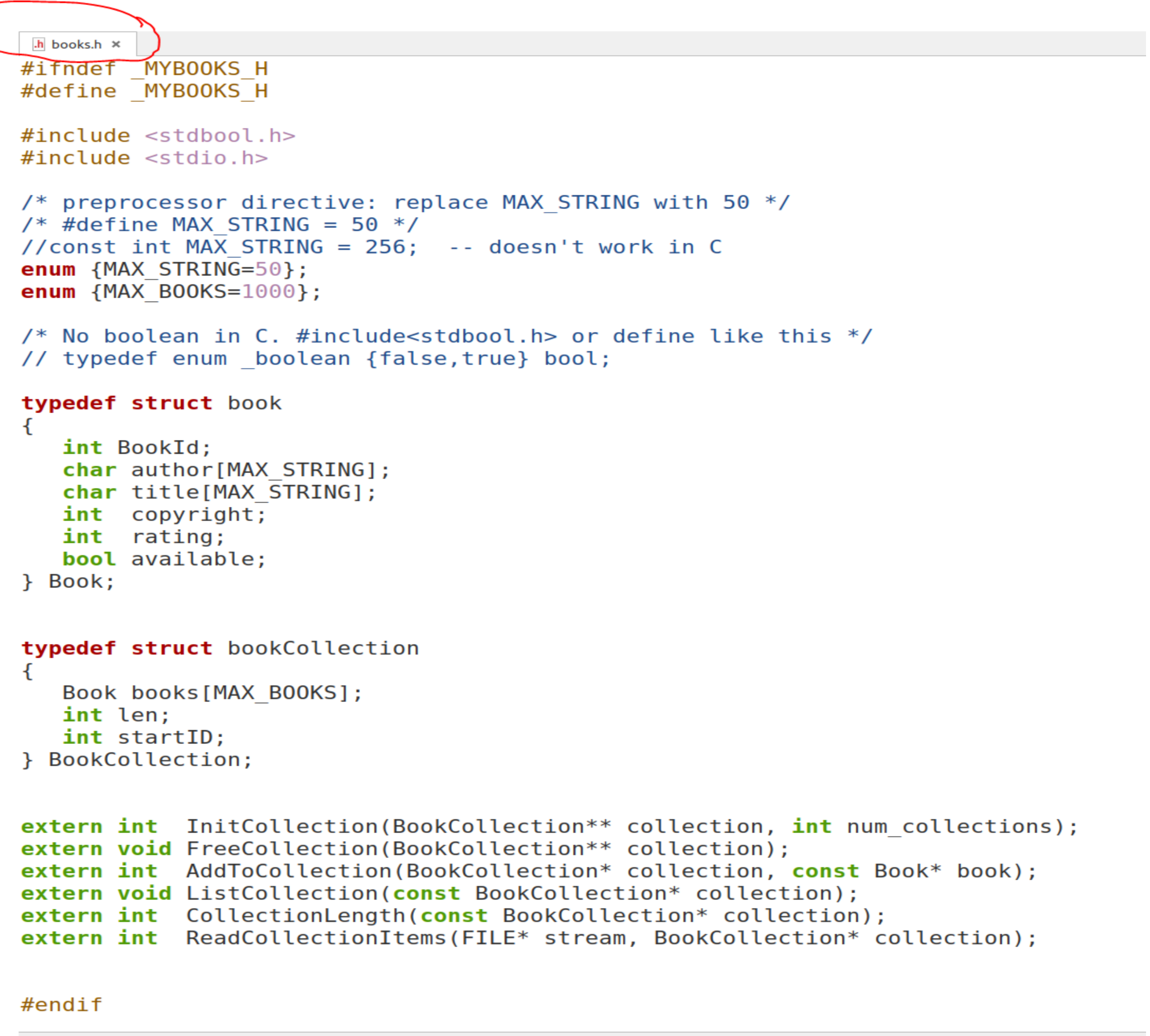
- Library interface:
  - “declarations” of types of functions go into a header file (.h file)
  - This provides interface to the library
  - “definitions” (implementation) of functions and types go into separate .c/.cpp file.
  - This is a essentially the definition of a package (sometimes in older speak its called a module)
  - Packages give us a necessary strategy for organizing and managing system structure.
    - This a basic construct for partitioning the system and achieving reuse
    - Is it imperative to use good package structure in your design for your system to be successful.   We will see examples of this throughout the semester. See the ***Package*** example included in the ***lecture8 c_examples***

BooksCollection Package (structure example)

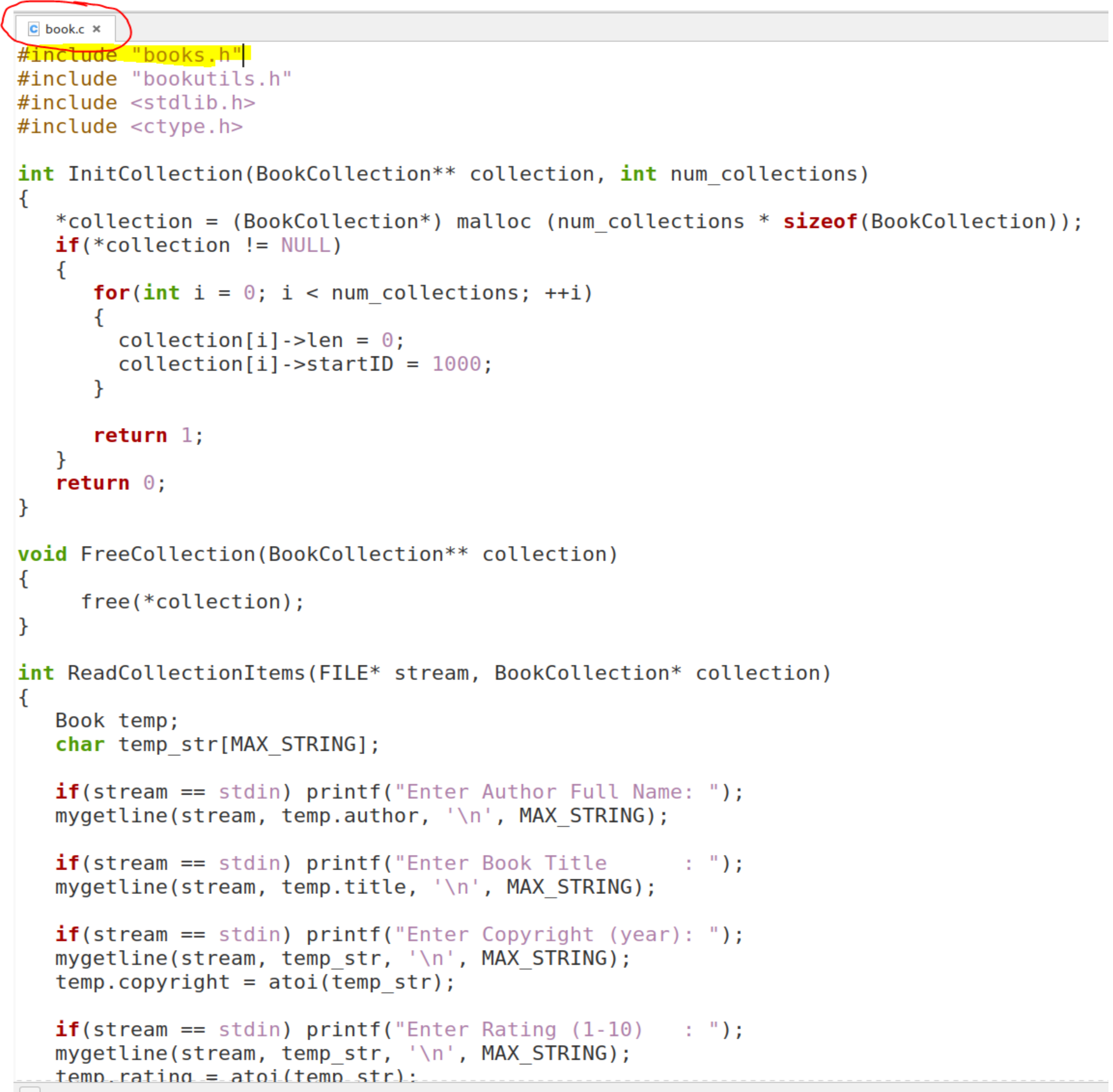


* The executive (client) simply uses the package to achieve requirements

Books.h Header file (interface) ***declares*** Books data and functions

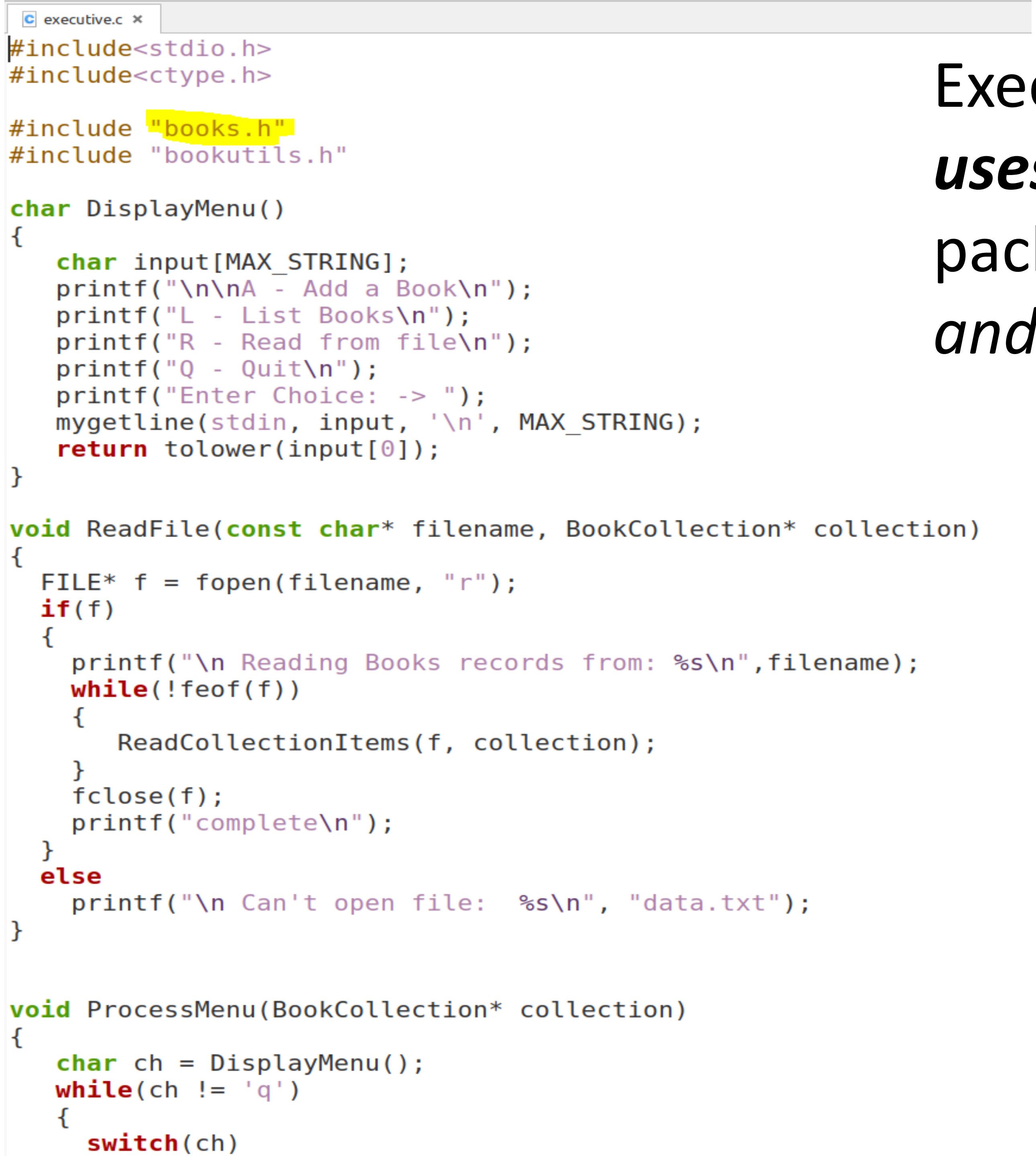

books.h

```c
#ifndef _MYBOOKS_H
#define _MYBOOKS_H

#include <stdbool.h>
#include <stdio.h>

/* preprocessor directive: replace MAX_STRING with 50 */
/* #define MAX_STRING = 50 */
//const int MAX_STRING = 256;  -- doesn't work in C
enum {MAX_STRING=50};
enum {MAX_BOOKS=1000};

/* No boolean in C. #include<stdbool.h> or define like this */
// typedef enum _boolean {false,true} bool;

typedef struct book
{
   int BookId;
   char author[MAX_STRING];
   char title[MAX_STRING];
   int  copyright;
   int  rating;
   bool available;
} Book;


typedef struct bookCollection
{
   Book books[MAX_BOOKS];
   int len;
   int startID;
} BookCollection;


extern int  InitCollection(BookCollection** collection, int num_collections);
extern void FreeCollection(BookCollection** collection);
extern int  AddToCollection(BookCollection* collection, const Book* book);
extern void ListCollection(const BookCollection* collection);
extern int  CollectionLength(const BookCollection* collection);
extern int  ReadCollectionItems(FILE* stream, BookCollection* collection);


#endif
```



```
#include "books.h"
#include "bookutils.h"
#include <stdlib.h>
#include <ctype.h>

int InitCollection(BookCollection** collection, int num_collections)
{
   *collection = (BookCollection*) malloc (num_collections * sizeof(BookCollection));
   if(*collection != NULL)
   {
      for(int i = 0; i < num_collections; ++i)
      {
        collection[i]->len = 0;
        collection[i]->startID = 1000;
      }

      return 1;
   }
   return 0;
}

void FreeCollection(BookCollection** collection)
{
     free(*collection);
}

int ReadCollectionItems(FILE* stream, BookCollection* collection)
{
   Book temp;
   char temp_str[MAX_STRING];

   if(stream == stdin) printf("Enter Author Full Name: ");
   mygetline(stream, temp.author, '\n', MAX_STRING);

   if(stream == stdin) printf("Enter Book Title       : ");
   mygetline(stream, temp.title, '\n', MAX_STRING);

   if(stream == stdin) printf("Enter Copyright (year): ");
   mygetline(stream, temp_str, '\n', MAX_STRING);
   temp.copyright = atoi(temp_str);

   if(stream == stdin) printf("Enter Rating (1-10)    : ");
   mygetline(stream, temp_str, '\n', MAX_STRING);
   temp.rating = atoi(temp_str);
```

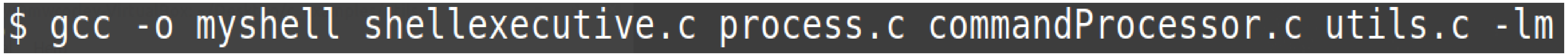
Books.c ***defines*** (implements) *Books data and functions*



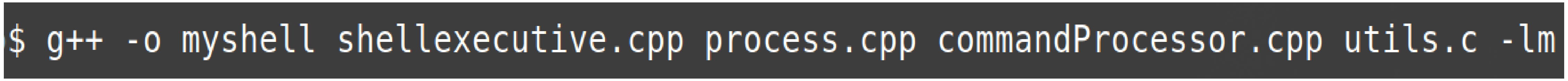
```
#include<stdio.h>
#include<ctype.h>

#include "books.h"
#include "bookutils.h"

char DisplayMenu()
{
   char input[MAX_STRING];
   printf("\n\nA - Add a Book\n");
   printf("L - List Books\n");
   printf("R - Read from file\n");
   printf("Q - Quit\n");
   printf("Enter Choice: -> ");
   mygetline(stdin, input, '\n', MAX_STRING);
   return tolower(input[0]);
}

void ReadFile(const char* filename, BookCollection* collection)
{
  FILE* f = fopen(filename, "r");
  if(f)
  {
    printf("\n Reading Books records from: %s\n",filename);
    while(!feof(f))
    {
       ReadCollectionItems(f, collection);
    }
    fclose(f);
    printf("complete\n");
  }
  else
    printf("\n Can't open file:  %s\n", "data.txt");
}


void ProcessMenu(BookCollection* collection)
{
   char ch = DisplayMenu();
   while(ch != 'q')
   {
     switch(ch)
```

Executive.c (the client application) ***uses*** the services of the Books package (implements) *Books data and functions*

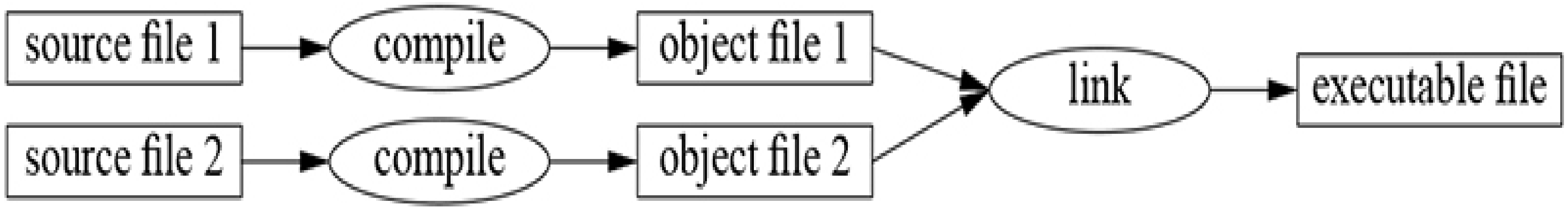
C/C++ Compilation and compute/memory models



```
$ gcc -o myshell shellexecutive.c process.c commandProcessor.c utils.c -lm
```



```
$ g++ -o myshell shellexecutive.cpp process.cpp commandProcessor.cpp utils.c -lm
```



- C and C++ Language Comparison
  - Dr. Fawcett's notes from CSE 687: Object-oriented Design (Spring 2019)
- Compilation, computation, and memory models
  - Dr. Fawcett's notes from CSE 687: Object-oriented Design (Spring 2019)
- Program Memory Layout

Examples

- lecture 8 (tarball): lecture8_examples.tar.gz
  - C Examples: see folder: c_examples
  - C++ Examples: see folder: cpp_examples

- Note: fort the C++ examples you will get a C++ compiler.
  - For Mint (Unbunu), open terminal (bash shell) and type the command -> *sudo apt-get install g++*