CSE 1325

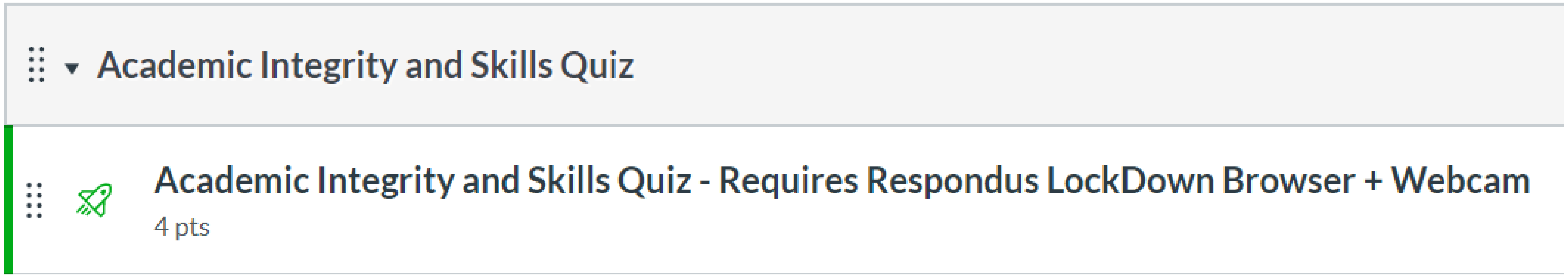
Week of 08/31/2020

Instructor : Donna French

Academic Integrity and Skills Quiz

- Quiz needs to be taken BEFORE the first OLQ which is next Tuesday.

- The number of points that are scored on the quiz will be added to your first OLQ score – up to 4 bonus points.



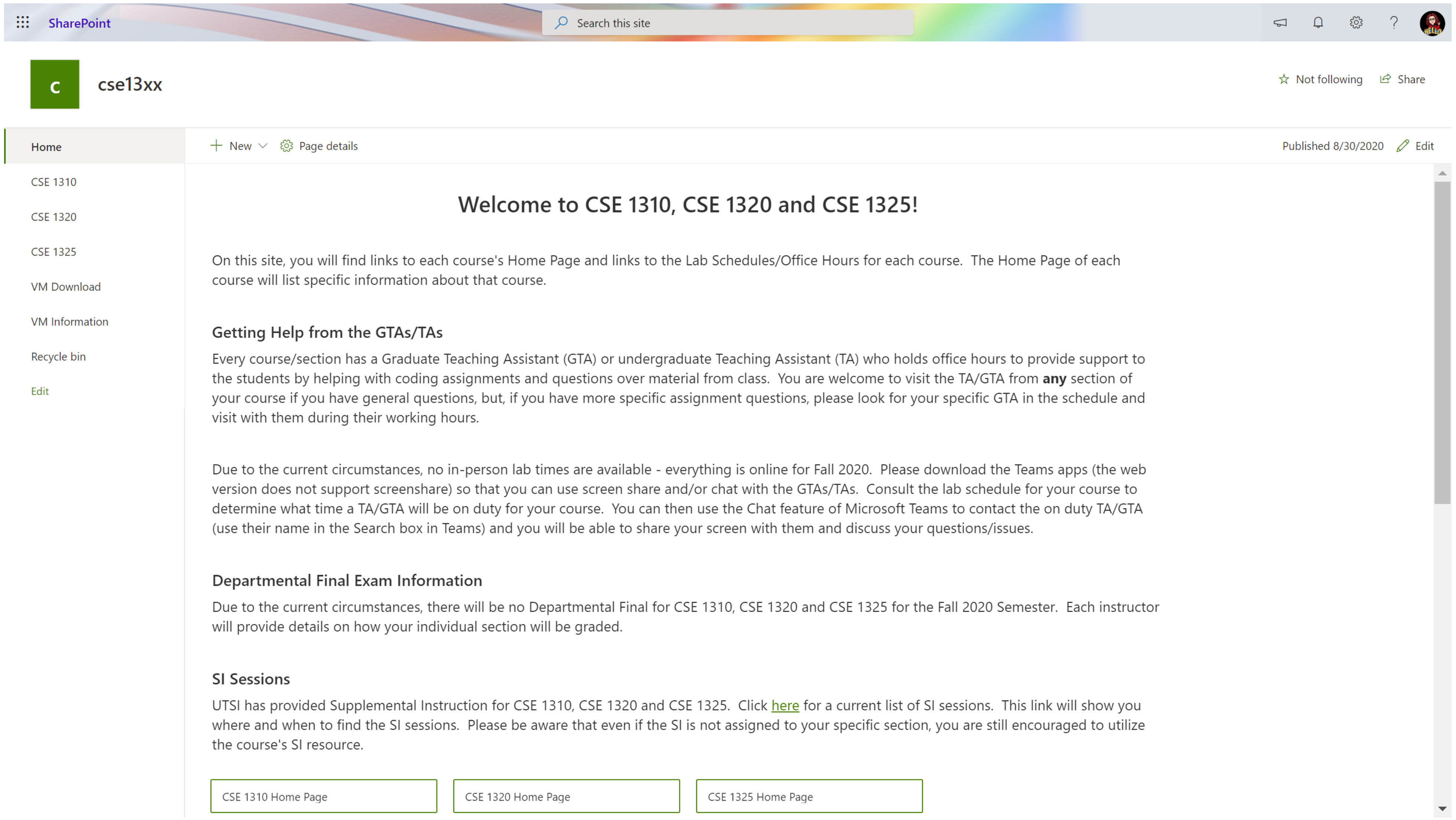
Academic Integrity and Skills Quiz

Academic Integrity and Skills Quiz - Requires Respondus LockDown Browser + Webcam

4 pts

- If you do not complete the Academic Integrity and Skills Quiz, 10 points will be deducted from your first OLQ.

https://mavsuta.sharepoint.com/sites/cse13xx



Welcome to CSE 1310, CSE 1320 and CSE 1325!

On this site, you will find links to each course's Home Page and links to the Lab Schedules/Office Hours for each course. The Home Page of each course will list specific information about that course.

Getting Help from the GTAs/TAs

Every course/section has a Graduate Teaching Assistant (GTA) or undergraduate Teaching Assistant (TA) who holds office hours to provide support to the students by helping with coding assignments and questions over material from class. You are welcome to visit the TA/GTA from **any** section of your course if you have general questions, but, if you have more specific assignment questions, please look for your specific GTA in the schedule and visit with them during their working hours.

Due to the current circumstances, no in-person lab times are available - everything is online for Fall 2020. Please download the Teams apps (the web version does not support screenshare) so that you can use screen share and/or chat with the GTAs/TAs. Consult the lab schedule for your course to determine what time a TA/GTA will be on duty for your course. You can then use the Chat feature of Microsoft Teams to contact the on duty TA/GTA (use their name in the Search box in Teams) and you will be able to share your screen with them and discuss your questions/issues.

Departmental Final Exam Information

Due to the current circumstances, there will be no Departmental Final for CSE 1310, CSE 1320 and CSE 1325 for the Fall 2020 Semester. Each instructor will provide details on how your individual section will be graded.

SI Sessions

UTSI has provided Supplemental Instruction for CSE 1310, CSE 1320 and CSE 1325. Click here for a current list of SI sessions. This link will show you where and when to find the SI sessions. Please be aware that even if the SI is not assigned to your specific section, you are still encouraged to utilize the course's SI resource.

What is bash?

Bash is a Unix shell and command language written by Brian Fox for the GNU Project as a free software replacement for the Bourne shell.

Bash is a command processor that typically runs in a text window where the user types commands that cause actions. This is called the Command Line Interface (CLI). It can be easier for programmers to just type out a command rather than digging through menus. Bash can also read and execute commands from a file, called a shell script.

The shell's name is an acronym for Bourne-again shell, a pun on the name of the Bourne shell that it replaces and on the common term "born again".

Ubuntu

What is it?

- Ubuntu is a free and open source operating system and Linux distribution.
- Ubuntu is produced by Canonical.
- Ubuntu is named after the Southern African philosophy of ubuntu (literally, 'human-ness'), which Canonical suggests can be loosely translated as "humanity to others" or "I am what I am because of who we all are".
- Ubuntu is the most popular operating system for the cloud.

Hello World

In C

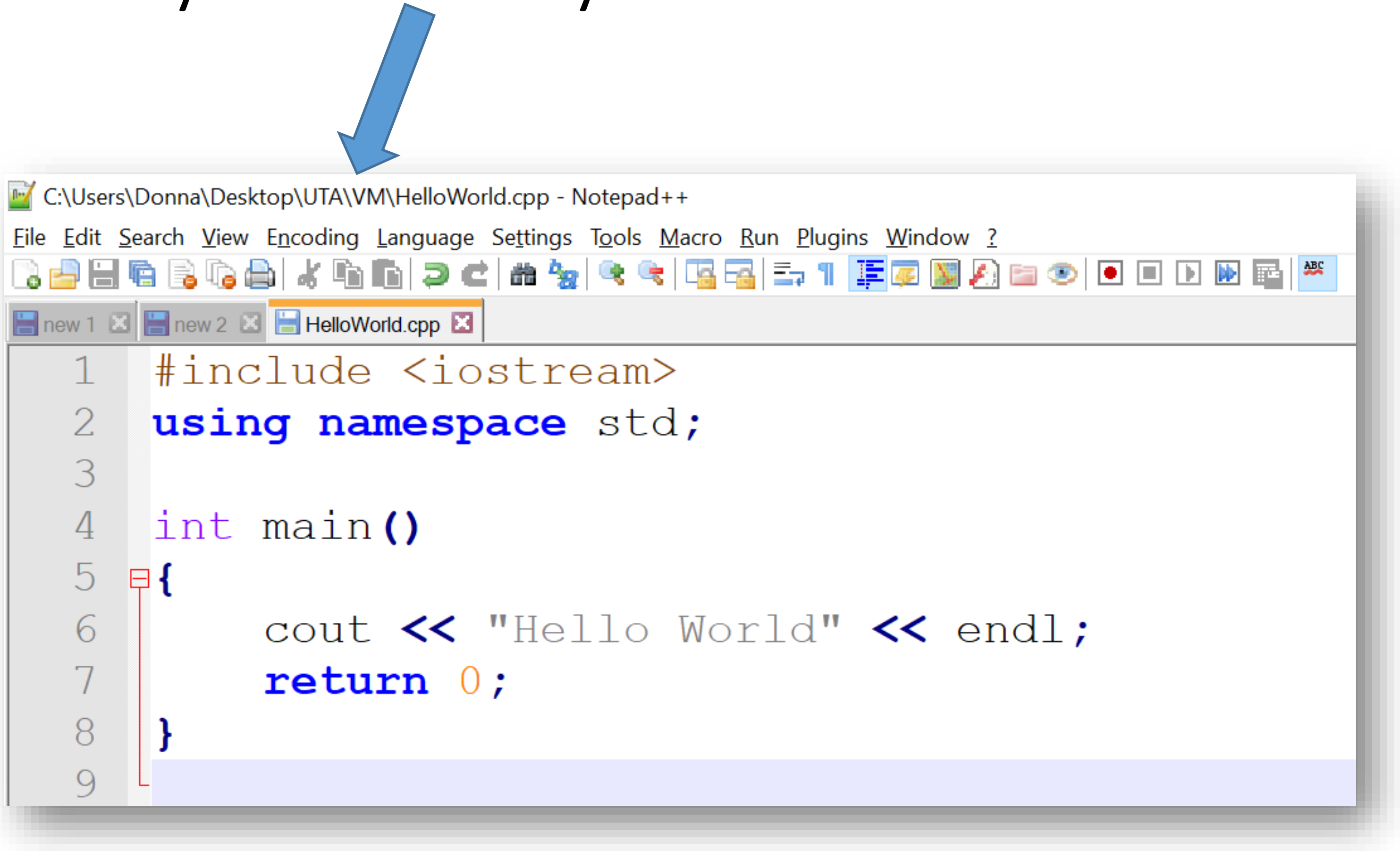
```
#include <stdio.h>
int main(void)
{
   printf("Hello World\n");
   return 0;
}
```

In C++

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

int main()
{
   cout << "Hello World" << endl;
   return 0;
}
```

Hello World

Use your favorite editor (I use Notepad++) to write HelloWorld.cpp.
Save to the folder you shared in your VM.


C:\Users\Donna\Desktop\UTA\VM\HelloWorld.cpp - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
new 1
new 2
HelloWorld.cpp

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

int main()
{
    cout << "Hello World" << endl;
    return 0;
}

```

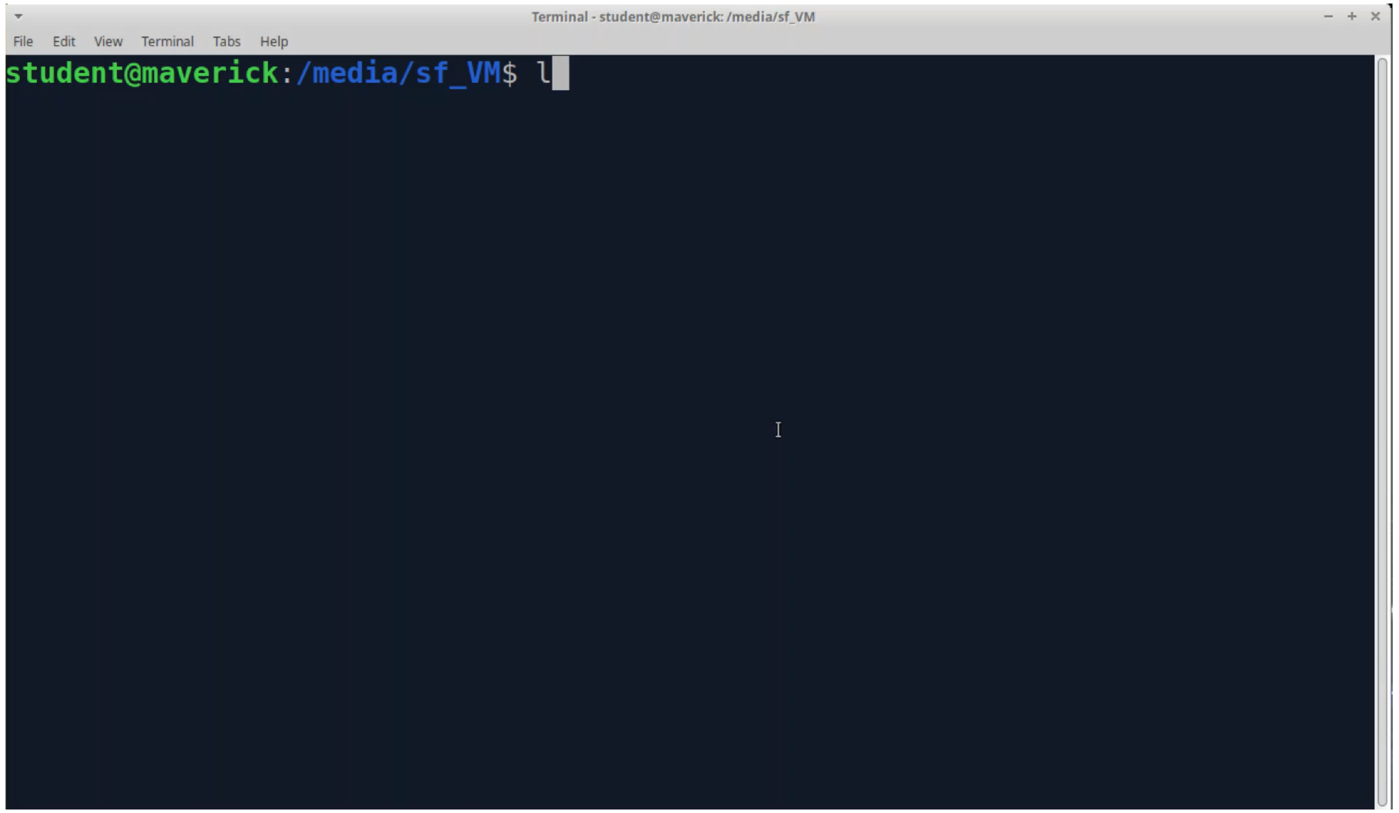
Hello World

You should be able to see it now in your VM when you open your shared folder with the terminal.

```
g++ HelloWorld.cpp
```

Should produce an a.out file.
Run your executable with

```
./a.out
```


Terminal - student@maverick: /media/sf_VM
File Edit View Terminal Tabs Help

```
student@maverick:/media/sf_VM$ l
```

Hello World

`#include <iostream>`

`iostream` is the header file which contains the functions for formatted input and output including `cout`, `cin`, `cerr` and `clog`.

C++ standard library packages don't need a .h to reference them.

```
#include <iostream>

using namespace std;

int main()
{
   cout << "Hello World" << endl;
   return 0;
}
```

Hello World

`using namespace std`

The built in C++ library routines are kept in the standard namespace which includes `cout`, `cin`, `string`, `vector`, `map`, etc.

Because these tools are used so commonly, it's useful to add "`using namespace std`" at the top of your source code so that you won't have to type the **std**:: prefix constantly.

We use just

```
cout
```

instead of

```
std::cout
```

```
#include <iostream>

using namespace std;

int main()
{
   cout << "Hello World" << endl;
   return 0;
}
```

Hello World

What is a `namespace`?

`namespace` is a language mechanism for grouping declarations. Used to organize classes, functions, data and types.

Namespaces are used to organize code into logical groups and to prevent name collisions that can occur especially when your code base includes multiple libraries.

```
#include <iostream>

using namespace std;

int main()
{
   cout << "Hello World" << endl;
   return 0;
}
```

I could create a function with the same name and define its own namespace and use the :: scope resolution operator to refer to my version.

We'll get into this more later…

Hello World

```
#include <iostream>

using namespace std;

int main()
{
   cout << "Hello World" << endl;
   return 0;
}
```

`cout` and `<<` and `endl`

`cout` is an abbreviation of **c**haracter **out**put stream.

`<<` is the output operator

`endl` puts `'\n'` into the stream and flushes it

So the line

```
cout << "Hello World" << endl;
```

puts the string “Hello World” into the character output stream and flushes it to the screen

Hello World Plus

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

int main()
{
   string first_name;
   cout << "Hello World" << endl;
   cout << "What is your name?" << endl;
   cin >> first_name;
   cout << "Hello " << first_name << endl;
   return 0;
}
```

`string` is a variable type that can hold character data

`cin` is an abbreviation of **c**haracter **in**put stream.

`>>` is the input operator

Hello World Plus

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

int main()
{
   string first_name;
   cout << "Hello World" << endl;
   cout << "What is your name?" << endl;
   cin >> first_name;
   cout << "Hello " << first_name << endl;
   return 0;
}
```

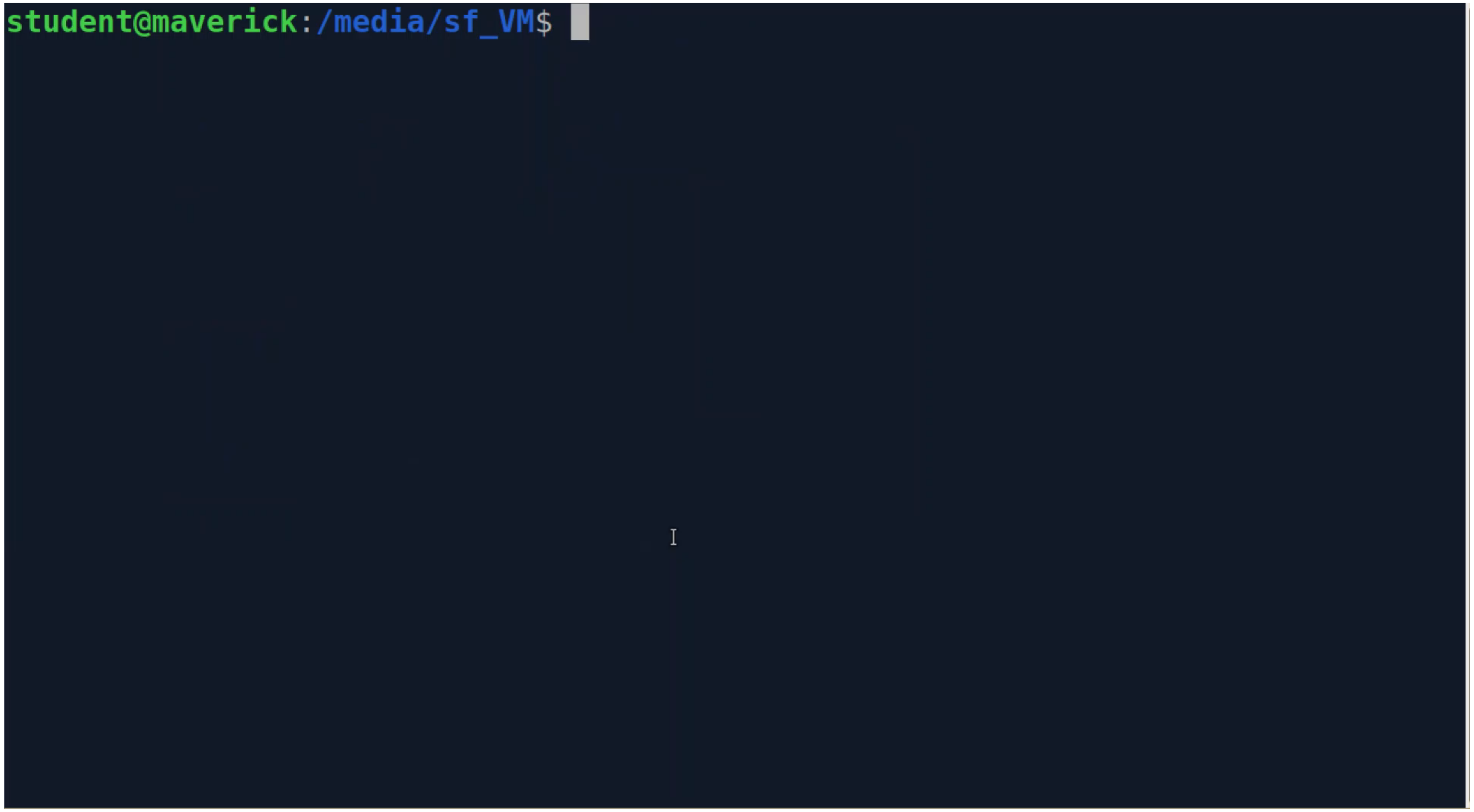
This line

```
cin >> first_name;
```

puts whatever you type at the terminal (up to the first whitespace) into the string variable `first_name`

Note that the <ENTER> key (newline) is not stored in `first_name`

Hello World Plus



```
student@maverick:/media/sf_VM$ 
```

makefile

We are going to start out using a simple makefile.

We will expand on this later as it becomes more necessary.

Please download this template from Canvas and use it with all of your coding assignments.

Course Materials ->

C++ makefile

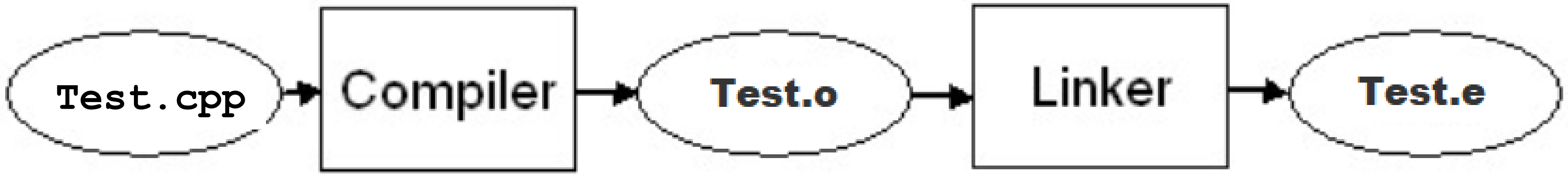
Add your name and student id as the first line

```
#FirstName LastName StudentID
```

```
#makefile for C++ program
SRC = HelloWorld.cpp
OBJ = $(SRC:.cpp=.o)
EXE = $(SRC:.cpp=.e)

CFLAGS = -g -std=c++11

all : $(EXE)

$(EXE): $(OBJ)
	g++ $(CFLAGS) $(OBJ) -o $(EXE)

$(OBJ) : $(SRC)
	g++ -c $(CFLAGS) $(SRC) -o $(OBJ)
```

makefile

```
g++ -c Test.cpp        g++ -g Test.o -o Test.e
```



The source file that you type into the editor. This is just a text file, anybody can read.

The object file is an intermediate file. It is only readable by the compiler and the linker.

The executable is the final product. It is a binary file that the operating system can run.

`makefile`

What is a `makefile`?

`make` is UNIX utility that is designed to start execution of a `makefile`.

A `makefile` is a special file, containing shell commands, that you create and name `makefile`.

While in the directory containing your `makefile`, you will type `make` and the commands in the `makefile` will be executed.

If you create more than one `makefile`, be certain you are in the correct directory before typing `make`.

`makefile`

`make` keeps track of the last time files (normally object files) were updated and only updates those files which are required (ones containing changes) to keep the sourcefile up-to-date.

If you have a large program with many source and/or header files, when you change a file on which others depend, you must recompile all the dependent files.

Without a `makefile`, this is an extremely time-consuming task.

`makefile`

As a `makefile` is a list of shell commands, it must be written for the shell which will process the `makefile`. A `makefile` that works well in one shell may not execute properly in another shell.

The `makefile` contains a list of rules. These rules tell the system what commands you want to be executed. Most times, these rules are commands to compile(or recompile) a series of files.

The rules, **which must begin in column 1**, are in two parts. The first line is called a dependency line and the subsequent line(s) are system commands or recipes which must be indented with a tab.

makefile

```
RULE : DEPENDENCIES
[tab]SYSTEM COMMANDS (RECIPE)
```

A **rule** is usually the name of a file that is generated by a program; examples of rules are executable or object files. A rule can also be the name of an action to carry out, such as "clean". Multiple rules must be separated by a space

A **dependency** (also called *prerequisite*) is a file that is used as input to create the rule. A rule often depends on several files.

The **system command(s)** (also called *recipe*) is an action that make carries out. A recipe may have more than one command, either on the same line or each on its own line. Recipe lines must be indented using a single <tab> character.

`makefile`

After the `makefile` has been created, a program can be (re)compiled by typing `make` in the correct directory.

`make` then reads the `makefile` and creates a dependency tree and takes whatever action is necessary. It will not necessarily do all the rules in the `makefile` as all dependencies may not need updated. It will rebuild target files if they are missing or older than the dependency files.

Unless directed otherwise, `make` will stop when it encounters an error during the construction process.

makefile

rule dependency

```
all : Code1_100074079.e
```

rule dependency

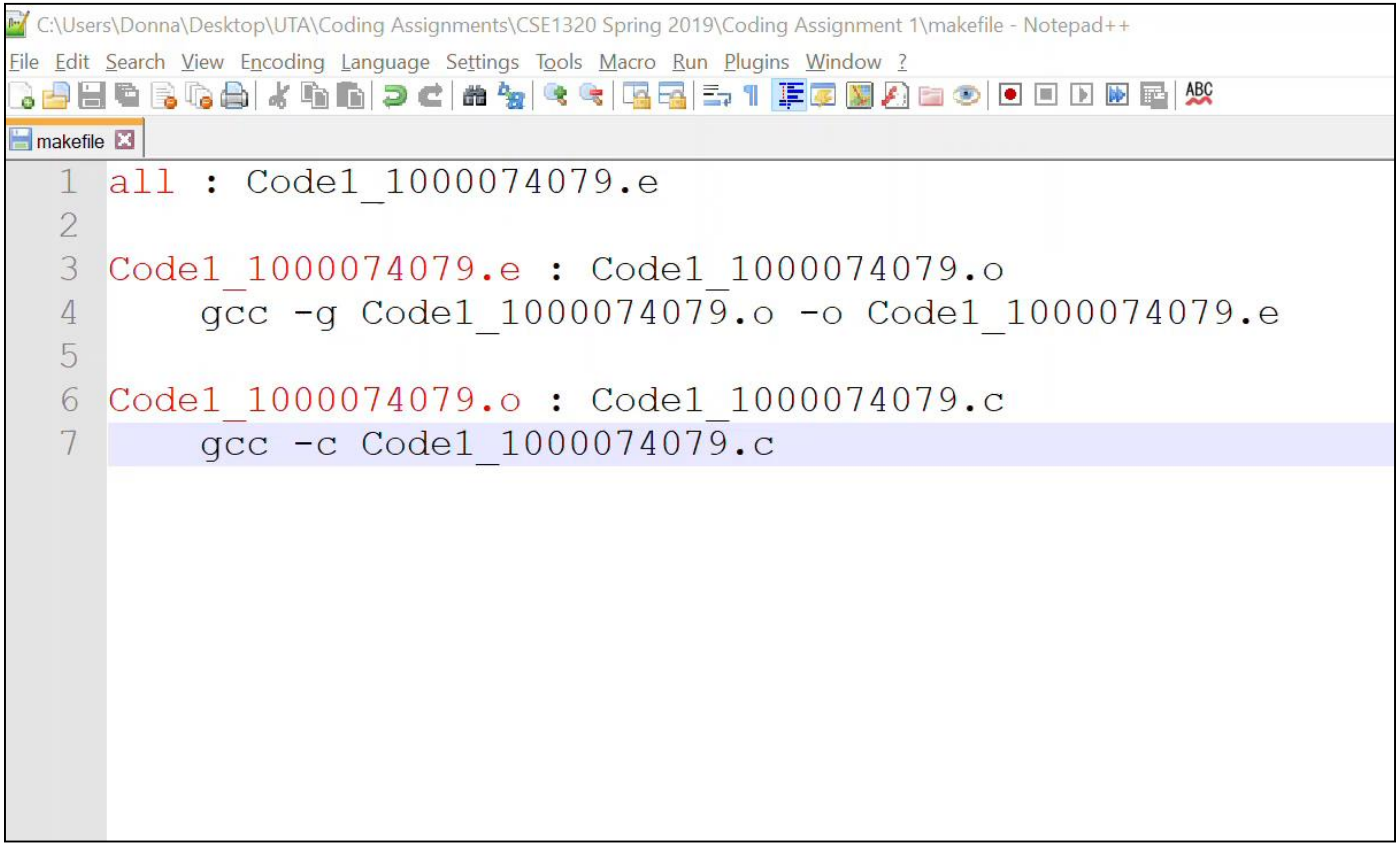
```
Code1_100074079.e : Code1_100074079.o
	g++ -g Code1_100074079.o -o Code1_100074079.e
```

recipe

rule dependency

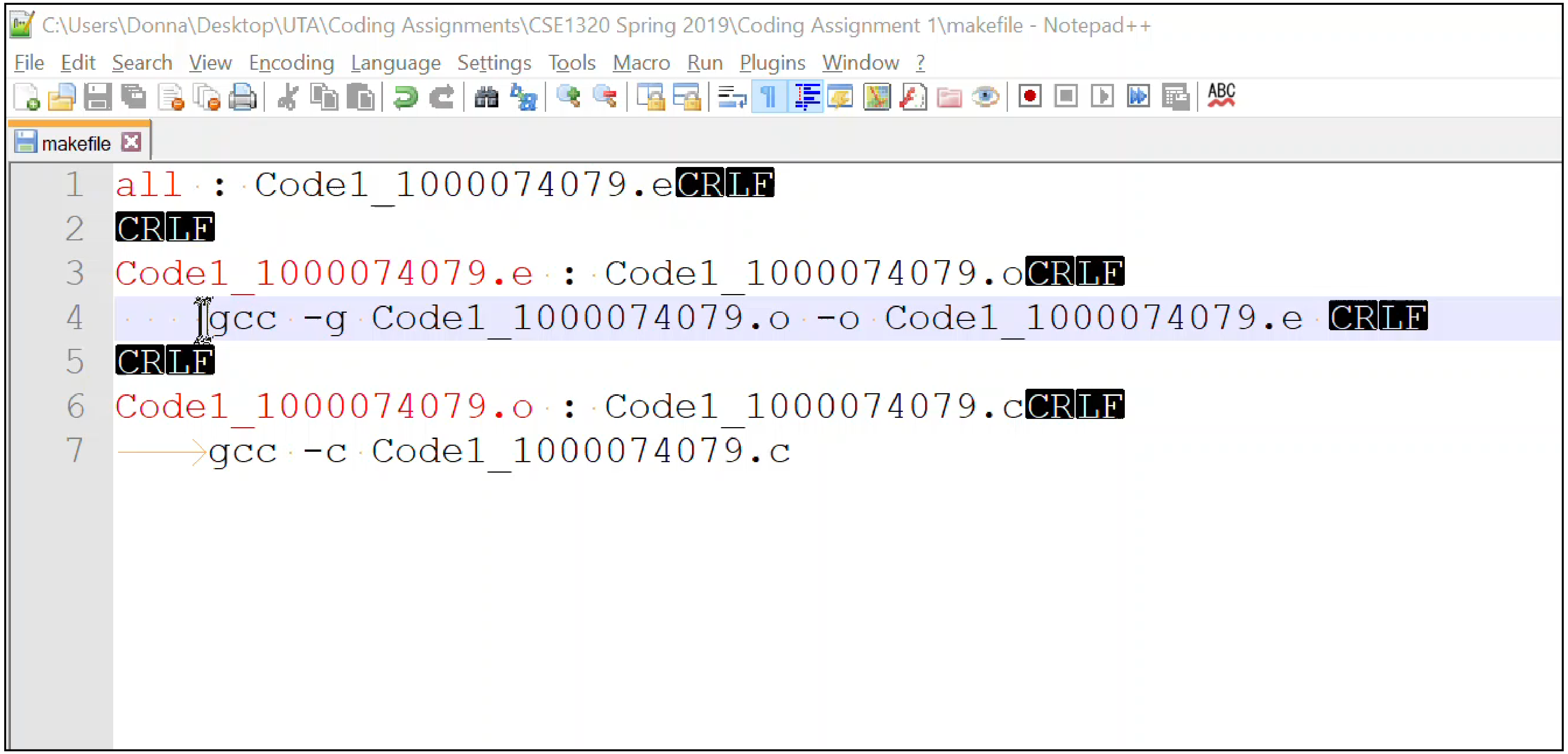
```
Code1_100074079.o : Code1_100074079.cpp
	g++ -c Code1_100074079.cpp
```

recipe


C:\Users\Donna\Desktop\UTA\Coding Assignments\CSE1320 Spring 2019\Coding Assignment 1\makefile - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
makefile

```
all : Code1_1000074079.e

Code1_1000074079.e : Code1_1000074079.o
	gcc -g Code1_1000074079.o -o Code1_1000074079.e

Code1_1000074079.o : Code1_1000074079.c
	gcc -c Code1_1000074079.c
```

```
[frenchdm@omega CA1]$ make
makefile:4: *** missing separator.  Stop.
[frenchdm@omega CA1]$
```



C:\Users\Donna\Desktop\UTA\Coding Assignments\CSE1320 Spring 2019\Coding Assignment 1\makefile - Notepad++

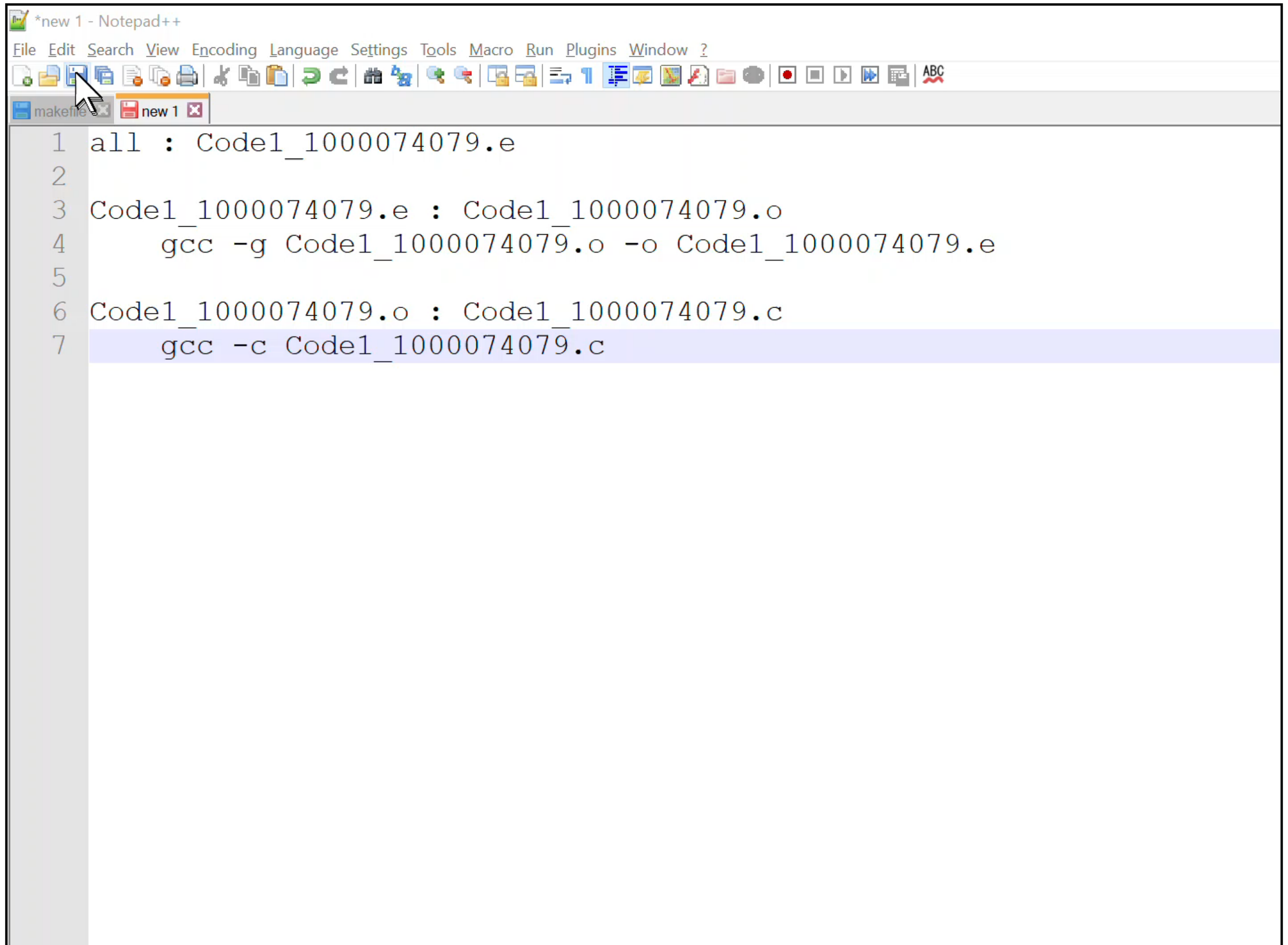
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
makefile

```
all : Code1_1000074079.eCRLF
CRLF
Code1_1000074079.e : Code1_1000074079.oCRLF
    gcc -g Code1_1000074079.o -o Code1_1000074079.e CRLF
CRLF
Code1_1000074079.o : Code1_1000074079.cCRLF
	gcc -c Code1_1000074079.c
```

The name of the `makefile` MUST BE

`makefile`

Save in Notepad++ with a dot on the end to force Notepad++ to not add an extension.


*new 1 - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
makefile
new 1

```
all : Code1_1000074079.e

Code1_1000074079.e : Code1_1000074079.o
	gcc -g Code1_1000074079.o -o Code1_1000074079.e

Code1_1000074079.o : Code1_1000074079.c
	gcc -c Code1_1000074079.c
```

```
[frenchdm@omega CA1]$ ls
Code1_1000074079.cpp  makefile.txt
[frenchdm@omega CA1]$ make
make: *** No targets specified and no makefile found.  Stop.

[frenchdm@omega CA1]$ mv makefile.txt makefile.mak
[frenchdm@omega CA1]$ ls
Code1_1000074079.cpp  makefile.mak

[frenchdm@omega CA1]$ make
make: *** No targets specified and no makefile found.  Stop.

[frenchdm@omega CA1]$ mv makefile.mak makefile
[frenchdm@omega CA1]$ make
g++ -c Code1_1000074079.cpp
g++ -g Code1_1000074079.o -o Code1_1000074079.e
[frenchdm@omega CA1]$
```

`makefile`

```
all : HelloWorld.e

HelloWorld.e : HelloWorld.o
	g++ -g -std=c++11 HelloWorld.o -o HelloWorld.e

HelloWorld.o : HelloWorld.cpp
	g++ -c -g -std=c++11 HelloWorld.cpp -o HelloWorld.o
```

With this explicit `makefile`, calling just “`make`” causes execution to start at rule `all`

Calling “`make HelloWorld.e`” causes execution to start at rule `HelloWorld.e`

Calling “`make HelloWorld.o`” causes execution to start at rule `HelloWorld.o`

```
student@cse1325:/media/sf_VM$ more makefile
all : HelloWorld.e

HelloWorld.e : HelloWorld.o
	g++ -g -std=c++11 HelloWorld.o -o HelloWorld.e

HelloWorld.o : HelloWorld.cpp
	g++ -c -g -std=c++11 HelloWorld.cpp -o HelloWorld.o
student@cse1325:/media/sf_VM$ make HelloWorld.o
g++ -c -g -std=c++11 HelloWorld.cpp -o HelloWorld.o
student@cse1325:/media/sf_VM$ ls
HelloWorld.cpp  HelloWorld.o  makefile
student@cse1325:/media/sf_VM$ make HelloWorld.e
g++ -g -std=c++11 HelloWorld.o -o HelloWorld.e
student@cse1325:/media/sf_VM$ ls
HelloWorld.cpp  HelloWorld.e  HelloWorld.o  makefile
```

makefile

```
SRC = Code1_100074079.cpp
OBJ = $(SRC:.cpp=.o)
EXE = $(SRC:.cpp=.e)

CFLAGS = -g -std=c++11

all : $(EXE)

$(EXE): $(OBJ)
      g++ $(CFLAGS) $(OBJ) -o $(EXE)


$(OBJ) : $(SRC)
      g++ -c $(SRC)
```

```
all : Code1_1000074079.e

Code1_1000074079.e : Code1_1000074079.o
        g++ -g Code1_1000074079.o -o
Code1_1000074079.e

Code1_1000074079.o : Code1_1000074079.cpp
        g++ -c Code1_1000074079.cpp
```

makefile

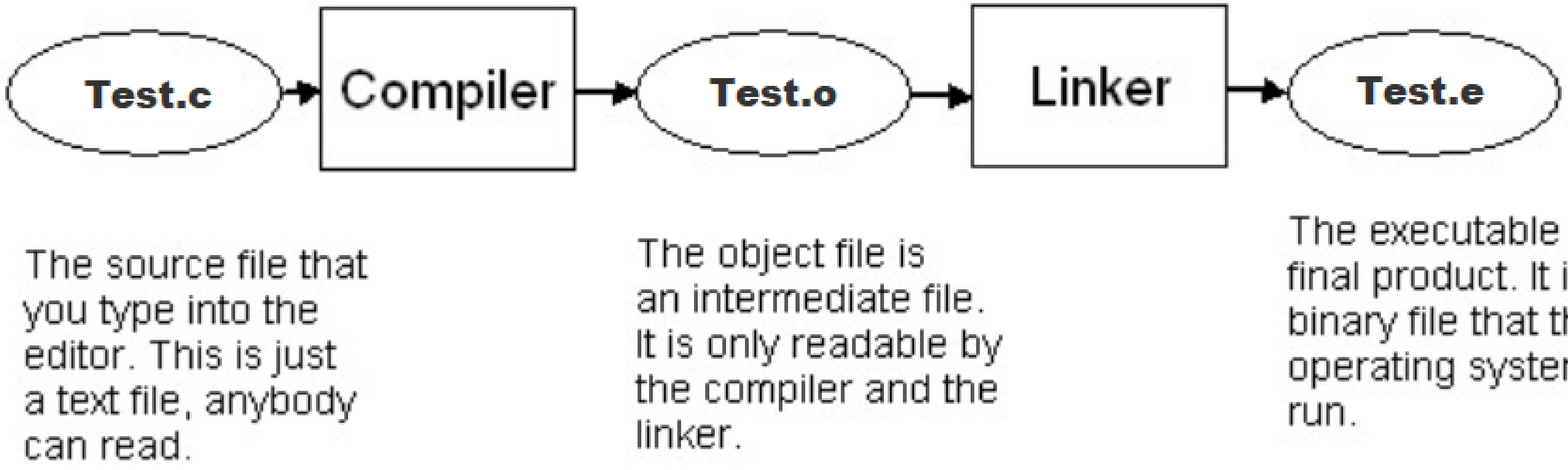
```
SRC = Test.cpp
OBJ = Test.o
EXE = Test.e

CFLAGS = -g -std=c++11

all : Test.e

Test.e  Test.o

      g++ -g -std=c++11 Test.o -o Test.e

Test.o : Test.cpp
      g++ -c Test.cpp
```

You **DO NOT** do these substitutions yourself – you let `make` do its job.

makefile

```
g++ -c Test.c
```

```
g++ -g Test.o MyLib.o -o Test.e
```



The source file that you type into the editor. This is just a text file, anybody can read.

The object file is an intermediate file. It is only readable by the compiler and the linker.

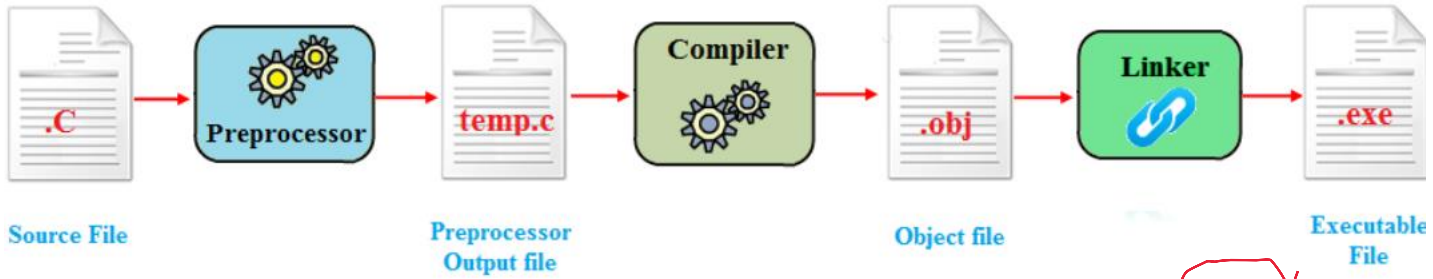
The executable is the final product. It is a binary file that the operating system can run.

makefile

```
SRC1 = Code1_1000074079.cpp
SRC2 = MyLib.cpp
OBJ1 = $(SRC1:.cpp=.o)
OBJ2 = $(SRC2:.cpp=.o)
EXE = $(SRC1:.cpp=.e)

CFLAGS = -g -std=c++11

all : $(EXE)

$(EXE): $(OBJ1) $(OBJ2)
	g++ $(CFLAGS) $(OBJ1) $(OBJ2) -o $(EXE)

$(OBJ1) : $(SRC1)
	g++ -c $(CFLAGS) $(SRC1) -o $(OBJ1)


$(OBJ2) : $(SRC2)
	g++ -c $(CFLAGS) $(SRC2) -o $(OBJ2)
```

```
SRC = Code1_100074079.cpp
OBJ = $(SRC:.cpp=.o)
EXE = $(SRC:.cpp=.e)

CFLAGS = -g –std=c++11

all : $(EXE)

$(EXE): $(OBJ)
	g++ $(CFLAGS) $(OBJ) -o $(EXE)

$(OBJ) : $(SRC)
	g++ -c $(CFLAGS) $(SRC) -o $(OBJ)
```

```
#Donna French 1000074079 Coding Assigment 6
SRC1 = Code6_1000074079.cpp
SRC2 = TrickOrTreater.cpp
SRC3 = House.cpp
SRC4 = CandyHouse.cpp
SRC5 = ToothbrushHouse.cpp
OBJ1 = $(SRC1:.cpp=.o)
OBJ2 = $(SRC2:.cpp=.o)
OBJ3 = $(SRC3:.cpp=.o)
OBJ4 = $(SRC4:.cpp=.o)
OBJ5 = $(SRC5:.cpp=.o)
EXE = $(SRC1:.cpp=.e)

CFLAGS = -g -std=c++11 -pthread

all : $(EXE)

$(EXE): $(OBJ1) $(OBJ2) $(OBJ3) $(OBJ4) $(OBJ5)
          g++ $(CFLAGS) $(OBJ1) $(OBJ2) $(OBJ3) $(OBJ4) $(OBJ5) -o $(EXE)

$(OBJ1) : $(SRC1)
          g++ -c $(CFLAGS) $(SRC1) -o $(OBJ1)

$(OBJ2) : $(SRC2)
          g++ -c $(CFLAGS) $(SRC2) -o $(OBJ2)

$(OBJ3) : $(SRC3)
          g++ -c $(CFLAGS) $(SRC3) -o $(OBJ3)

$(OBJ4) : $(SRC4)
          g++ -c $(CFLAGS) $(SRC4) -o $(OBJ4)

$(OBJ5) : $(SRC5)
          g++ -c $(CFLAGS) $(SRC5) -o $(OBJ5)
```



Shell Scripting

compiler

creates an object file

linker

takes in object files and produces an executable file

```
SRC1  = Code2_1000074079.cpp
SRC2  = MyLib.cpp
OBJ1  = $(SRC1:.cpp=.o)
OBJ2  = $(SRC2:.cpp=.o)
EXE  = $(SRC1:.cpp=.e)



CFLAGS  = -g -std=c++11

all  : $(EXE)

$(EXE): $(OBJ1) $(OBJ2)
        g++ $(CFLAGS) $(OBJ1) $(OBJ2) -o $(EXE)

$(OBJ1)  : $(SRC1)
        g++ -c $(CFLAGS) $(SRC1) -o $(OBJ1)

$(OBJ2)  : $(SRC2)
        g++ -c $(CFLAGS) $(SRC2) -o $(OBJ2)
```

makefile

************ IMPORTANT INFORMATION************

After you successfully compile a program, running `make` again will result in

```
student@cse1325:/media/sf_VM$ make
make: Nothing to be done for 'all'.
```

To force a recompile, use `–B` after **make**

```
student@cse1325:/media/sf_VM$ make -B
g++ -c -g -std=c++11 HelloWorld.cpp -o HelloWorld.o
g++ -g -std=c++11 HelloWorld.o -o HelloWorld.e
student@cse1325:/media/sf_VM$ make
make: Nothing to be done for 'all'.
student@cse1325:/media/sf_VM$ make -B
g++ -c -g -std=c++11 HelloWorld.cpp -o HelloWorld.o
g++ -g -std=c++11 HelloWorld.o -o HelloWorld.e
```

OLQ1

OLQ1 will be over `make` and `makefile`.

1. Given a `makefile` for a one module program, can you expand it to three modules?
2. Can you explain how to fix the `make` error

```
*** missing separator.
```

3. Can you identify which parts of a `makefile` are the compiler and which parts are the linker?

Variables in C++

Familiar variable types from C carry over to C++

```
    char
    short
    int
    float
    double
    void
    long
    unsigned
    signed
```

These are built-in types

A new built-in type in C++

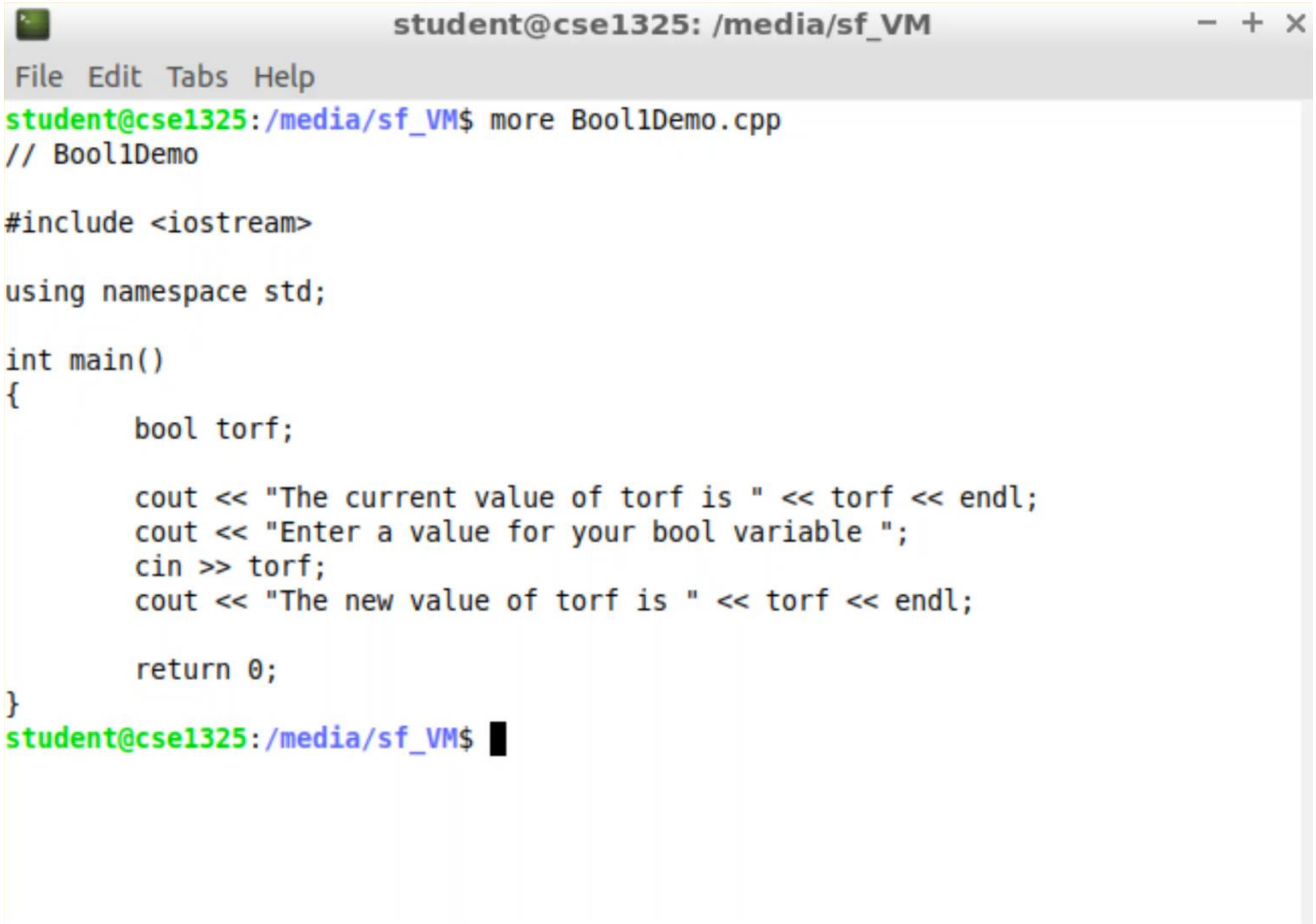
```
bool x
```

`x` is a Boolean which can have a value of true(1) or false(0)

New types defined in the standard library

```
string xxxx
```

`xxxx` is stream of characters



```
student@cse1325:/media/sf_VM$ more Bool1Demo.cpp
// Bool1Demo

#include <iostream>

using namespace std;

int main()
{
        bool torf;

        cout << "The current value of torf is " << torf << endl;
        cout << "Enter a value for your bool variable ";
        cin >> torf;
        cout << "The new value of torf is " << torf << endl;

        return 0;
}
student@cse1325:/media/sf_VM$ 
```

String Operators in C++

Initialization

```
string MyString4 = "How are you today?";
```

Assignment

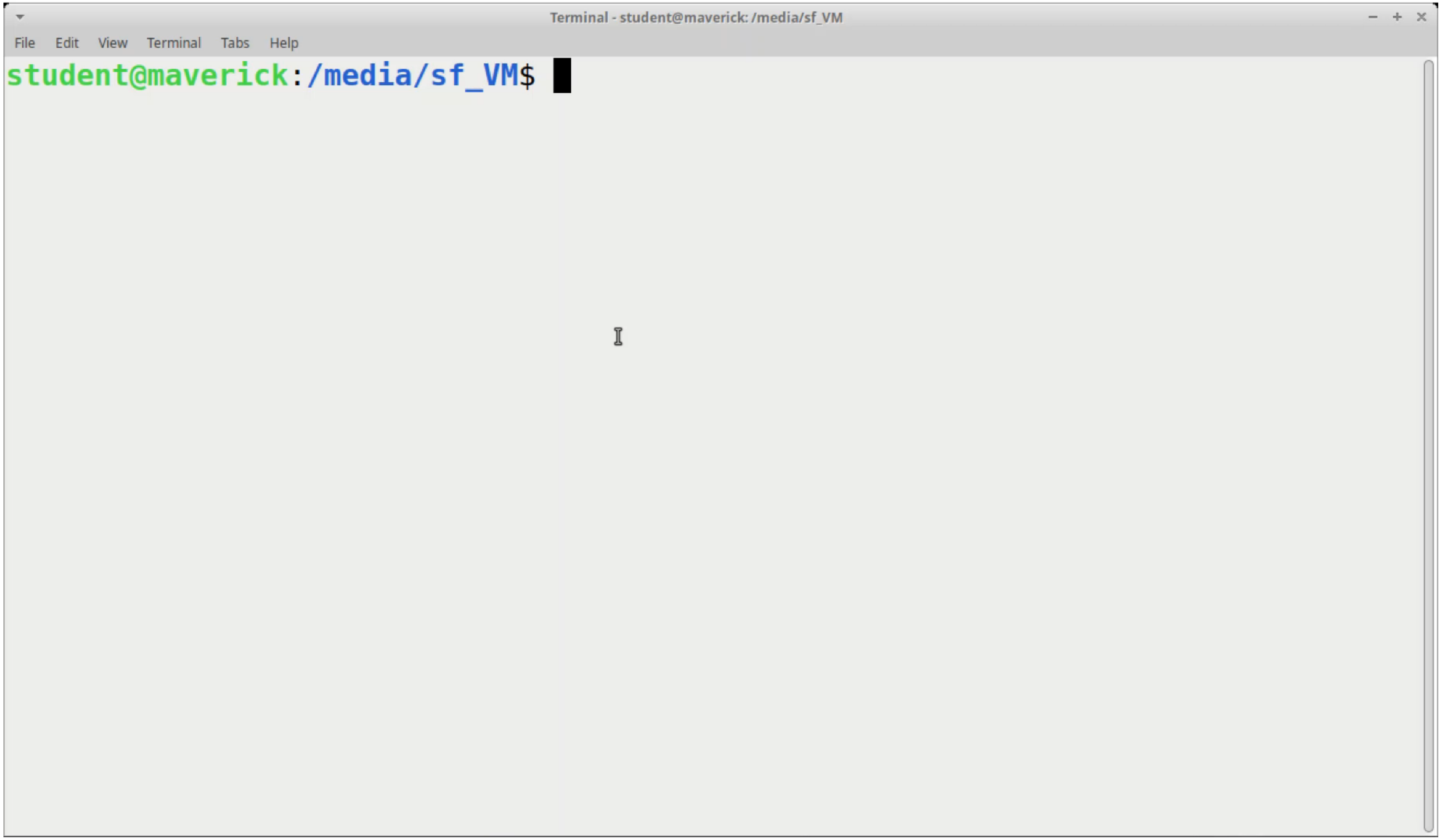
```
MyString2 = MyString1;
```

Concatenation

```
MyString2 = MyString1 + MyString3;
```

Comparison

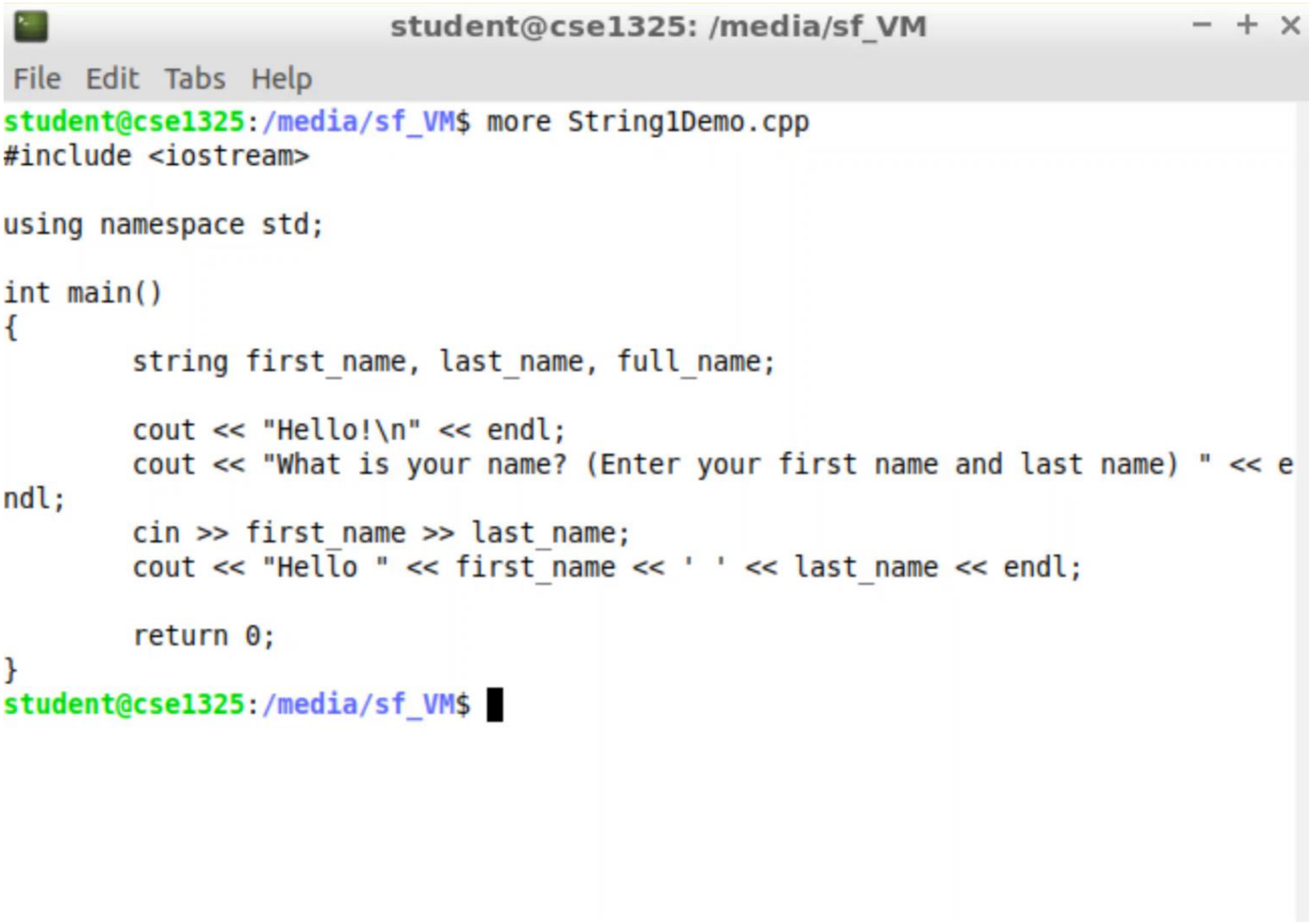
```
MyString1 <  MyString3
MyString1 <= MyString3
MyString1 >  MyString3
MyString1 >= MyString3
MyString1 == MyString3
MyString1 != MyString3
```



```
student@maverick:/media/sf_VM$ 
```

String Operators in C++

```
string MyString1, MyString2, MyString3;
string MyString4 = "How are you today?";

MyString2 = MyString1;

MyString2 = MyString1 + MyString3;

if (MyString1 < MyString3)
   cout << MyString1 << " is alphabetically before " << MyString3 << endl;
else if (MyString1 > MyString3)
   cout << MyString1 << " is alphabetically after " << MyString3 << endl;
else if (MyString1 == MyString3)
   cout << MyString1 << " is alphabetically equal to " << MyString3 << endl;
```



```
student@cse1325:/media/sf_VM$ more String1Demo.cpp
#include <iostream>

using namespace std;

int main()
{
        string first_name, last_name, full_name;

        cout << "Hello!\n" << endl;
        cout << "What is your name? (Enter your first name and last name) " << e
ndl;
        cin >> first_name >> last_name;
        cout << "Hello " << first_name << ' ' << last_name << endl;

        return 0;
}
student@cse1325:/media/sf_VM$ 
```

cin

```
cin >> CreamPuff;
```

cout

```
cout << "Happy Birthday";
```

stream insertion
vs
stream extraction

<<

stream insertion operator

>>

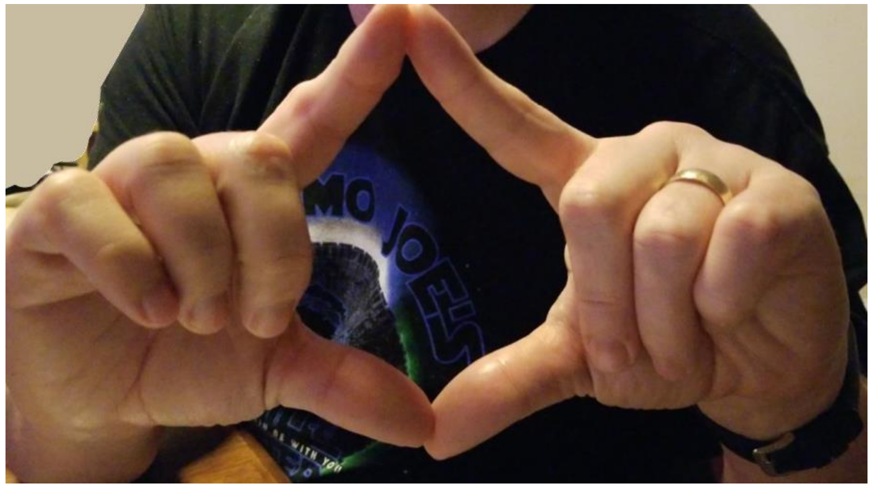
stream extraction operator

Remember the rule in English of "i before e except after c"?

i before e

insertion extraction

<< >>



Uniform Initialization

Unsafe Conversions

C++ allows for (implicit) unsafe conversions.

unsafe = a value can be implicitly turned into a value of another type that does not equal the original value

```
int IntVar1 = 32112;
char CharVarA = IntVar1;
int IntVar2 = CharVarA;
```

Uniform Initialization

Unsafe Conversions

To be warned against these unsafe conversions, use the uniform initialization format

```
int IntVar1 {32112};
char CharVarA {IntVar1};
int IntVar2 {CharVarA};
```

```
student@maverick:/media/sf_VM$ 
```

Uniform Initialization

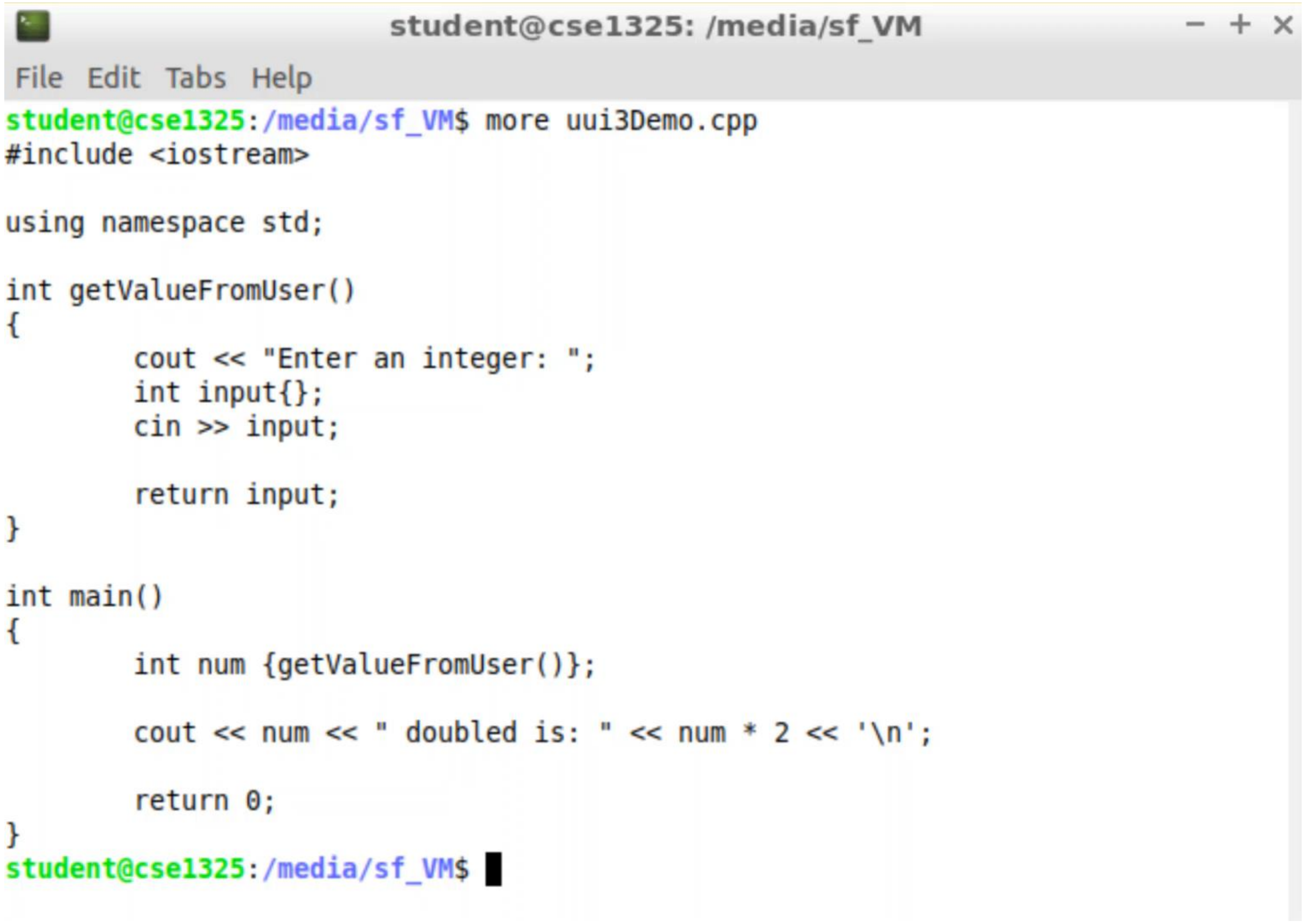
Additional Notes about Uniform Initialization

Type bool can be initialized with UI

```
bool b1 {true};
bool b2 {false};
bool b3 {!true};
bool b4 {!false};
```

A function can be called that returns a value inside the {}

Use empty braces {} to initialize a variable to 0.



```
student@cse1325:/media/sf_VM$ more uui3Demo.cpp
#include <iostream>

using namespace std;

int getValueFromUser()
{
        cout << "Enter an integer: ";
        int input{};
        cin >> input;

        return input;
}

int main()
{
        int num {getValueFromUser()};

        cout << num << " doubled is: " << num * 2 << '\n';

        return 0;
}
student@cse1325:/media/sf_VM$ 
```

DRY vs WET Coding

DRY

Don’t Repeat Yourself

Advantages

- Maintainability
- Readability
- Reuse
- Cost
- Testing

WET

Write Everything Twice

We Enjoy Typing

Advantages

NONE

Abstraction

In order to use a function, you only need to know its name, inputs, outputs, and where it lives.

You don’t need to know how it works, or what other code it’s dependent upon to use it.

This lowers the amount of knowledge required to use other people’s code (including everything in the standard library).

Required Formatting of Code

The opening brace for a function should be given its own line and the closing brace should line up with the opening brace. Any code lines within the braces should be indented the same amount which must be between 3 and 5 spaces.

```
int main()
{
    my first line
    my second line
    my third line
}
```

`std::string`

Just like the Java and C, a string is a collection of sequential characters.

C++ has a `string` type. Just like the Java, `string` is actually an object; therefore, knows things and can do things. This will make more sense once we start talking about classes and member functions.

To use `string` include the `string` header file.

```
    #include <string>
```

`std::string`

As we did with `cin` and `cout`, we can either put

```
     using namespace std
```

in our .cpp file and not need to preface `string` with `std::` or we can not use the `std` namespace and need to use `std::string`.

```
std::string MyString;
string MyString;
```

std::string

Declaring and initializing a string in one line

```
string MyString("Silly");
```

constructing

A string can also be declared and then assigned a value

```
string MyString;
MyString = "Silly";
```

assignment

`std::string`

We've already seen the example where `cin` stops reading at whitespace (just like `scanf()`).

```
string first_name, last_name, full_name;

cout << "Hello!\n" << endl;
cout << "What is your name? (Enter your first name and last name) " << endl;
cin >> first_name >> last_name;
cout << "Hello " << first_name << ' ' << last_name << endl;
```

std::string

What if we need to read a line of input including the whitespace into a single variable?

For example, what if I wanted to take whatever name was entered and only store it in one variable?

```
string full_name;

cout << "Hello!\n" << endl;
cout << "What is your name? " << endl;
cin >> full_name;
cout << "Hello " << full_name << endl;
```

If I type

```
Fred Flintstone
```

at the prompt, what will print?

`std::string`

`getline()` is the C++ version of `fgets()` from C. It takes two parameters just like `fgets()`.

The first parameter is the stream to read from – when reading from the screen use `cin`.

The second parameter is `string` variable where you want to store the input.

```
string full_name;

cout << "Hello!\n" << endl;
cout << "What is your name? " << endl;
getline(cin, full_name);
cout << "Hello " << full_name << endl;
```

```
Hello!

What is your name?
Fred Flintstone
Hello Fred Flintstone
```

`std::string`

Mixing `cin` with `getline()` can cause issues

`cin` leaves the newline (`\n`) in the standard input buffer.

```
10          cin >> dog_name;
(gdb)
What is your dog's name? Dino
11          cout << "Hi " << dog_name << endl;
(gdb) p *stdin
$1 = {_flags = -72539512, _IO_read_ptr = 0x555555769284 "\n",
```

`std::string`

Which `getline()` then reads and uses; therefore, not prompting for more input.

We can use

```
cin.ignore(50, ‘\n’);
```

This function discards the specified number of characters or fewer characters if the delimiter is encountered in the input stream.

Puts a null at the end of the buffer and throws out the newline

New keywords in C++

`const`

Used to inform the compiler that the value of a particular variable should not be modified.

If a value does not (or should not) change in the body of a function to which it's passed, the parameter should be declared `const`.

```
const int counter = 1;
```

`counter` is an integer constant

New keywords in C++

`const`

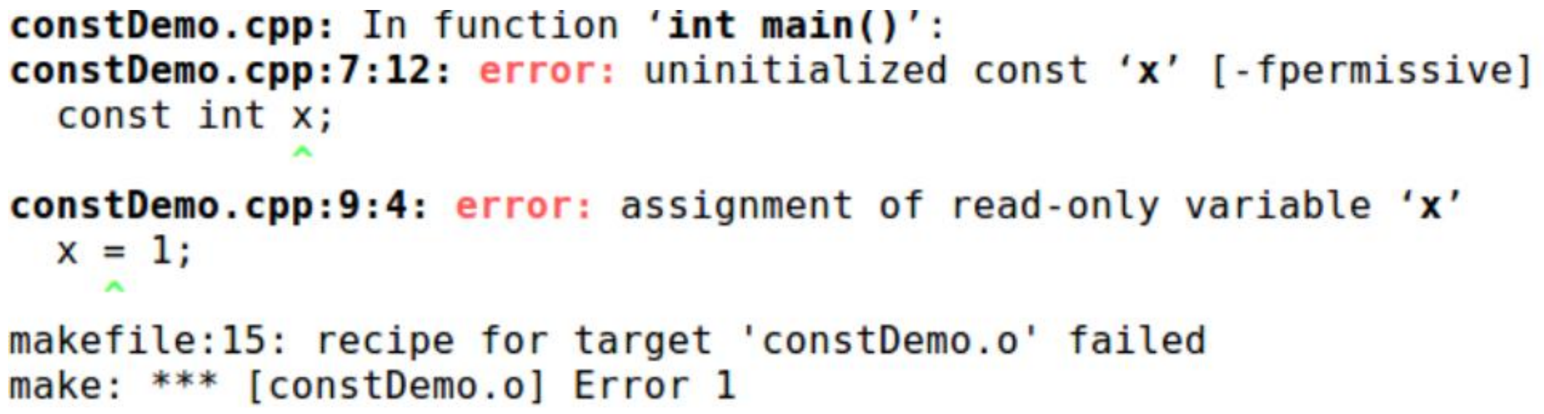
`const` variables must be initialized when you define them and then that value can not be changed via assignment.

`const` variables can be initialized from other variables (including non-const ones).

We will use `const` with function parameters when we learn about passing by value in C++.

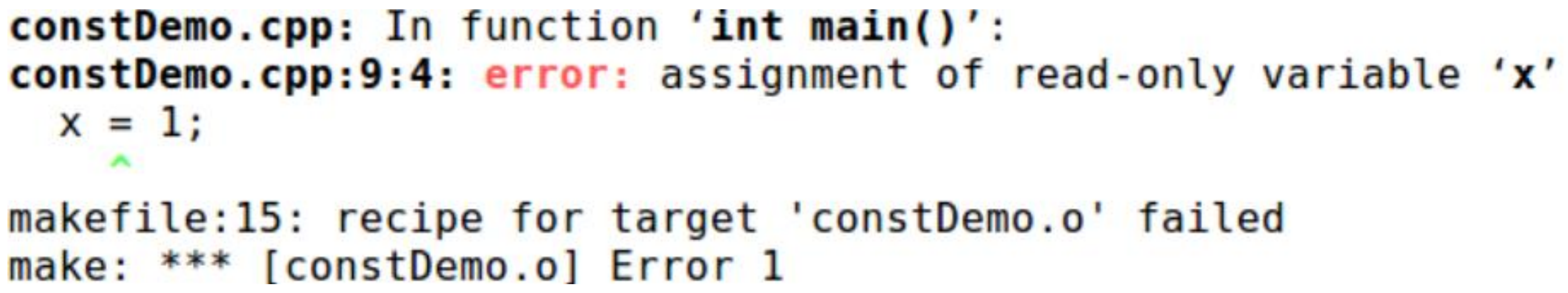
```
#include <iostream>

using namespace std;

int main()
{
  int x;

  x = 1;

  return 0;
}
```

```
#include <iostream>

using namespace std;

int main()
{
  const int x;

  x = 1;

  return 0;
}
```



```
constDemo.cpp: In function ‘int main()’:
constDemo.cpp:7:12: error: uninitialized const ‘x’ [-fpermissive]
  const int x;
            ^
constDemo.cpp:9:4: error: assignment of read-only variable ‘x’
  x = 1;
    ^
makefile:15: recipe for target 'constDemo.o' failed
make: *** [constDemo.o] Error 1
```

```cpp
#include <iostream>

using namespace std;

int main()
{
  const int x = 1;

  x = 1;

  return 0;
}
```



```
constDemo.cpp: In function 'int main()':
constDemo.cpp:9:4: error: assignment of read-only variable 'x'
  x = 1;
    ^
makefile:15: recipe for target 'constDemo.o' failed
make: *** [constDemo.o] Error 1
```