CSE 1325

Week of 11/07/2022

Instructor : Donna French

Generic Collections

`ArrayList` is a familiar construct now

`ArrayList` is a ***generic collection***.

Generic collection?

Dynamically resizable array-like data structure that stores references to objects of a type that you specify when you create it

Generic Collections

A collection is a data structure (actually an object) that can hold references to other objects.

Usually, collections contain references to objects of any type that has the ***is-a*** relationship with the type stored in the collection.

We stored `Circle`, `Square`, `Rectangle` and `Triangle` in an `ArrayList` of type `Shape`.

We stored `Fish`, `Frog` and `Birds` in an `ArrayList` (`Zoo`) of type `Animal`.

Interface	Description
`Collection`	The root interface in the collections hierarchy from which interfaces `Set`, `Queue` and `List` are derived.

Generic Collections

Type-Wrapper Classes

```
ArrayList<Integer> IList = new ArrayList<>();
ArrayList<Character> CList = new ArrayList<>();
ArrayList<Double> DList = new ArrayList<>();
```

`Integer, Character, Double, Boolean, Byte, Float, Long, Short`

These type-wrapper classes lets us treat primitive-type values as objects and store them in Collections like `ArrayList`.

Generic Collections

Type-Wrapper Classes

`Byte, Short, Integer, Long, Float, Double`

The numeric type-wrapper classes `extends` class `Number`.

Primitive types do not have methods, so the methods related to a primitive type are in the type-wrapper classes.

`parseInt` (converts a `String` to an `int` value) is part of class `Integer`

Generic Collections

Type-Wrapper Classes

Type-wrapper classes are `final` classes

If a class is `final`, then you cannot extend it.

```
cannot inherit from final Integer

Missing javadoc.

Create Test Class
----
(Alt-Enter shows hints)
```

```
ch 1000074079

ylistdemo;

public class TypeWrapper extends Integer
{

}
```

```
public class Rectangle extends Shape
```

```
public final class Rectangle extends Shape
```

So what happens to `Square`?

```
Square extends Rectangle
```

Is that inheritance allowed if `Rectangle` is `final`?

```
4   cannot inherit from final Rectangle
    ----
    (Alt-Enter shows hints)
5   demo;
    public class Square extends Rectangle
7   {
8       public Square(String name, double size)
9       {
10          super(name, size, size);
11          System.out.println("Constructing Square");
12      }
13  }
14
```

Generic Collections

Autoboxing and Auto-Unboxing

Boxing and unboxing conversions automatically convert between primitive-type values and type-wrapper objects.

A *boxing* conversion converts a value of a primitive type to an object of the corresponding type-wrapper class.

An *unboxing* conversion converts an object of a type-wrapper class to a value of the corresponding primitive type.

```
int intArray[] = {1,2,3,4,5,6,7,8,9};
ArrayList<Integer>IntegerAL = new ArrayList<>();

IntegerAL.add(intArray[1]);
IntegerAL.add(intArray[3]);
IntegerAL.add(intArray[5]);
IntegerAL.add(intArray[7]);
```

autoboxing

```
System.out.println("Printing the even values from the ArrayList");

for (Integer it : IntegerAL)
{
    System.out.printf("%d", it);
}
```

auto-unboxing

```
Printing the even values from the ArrayList
2468
```

Generic Collections

`List`

A `List` – sometimes called a sequence – is an ordered `Collection` that can contain duplicates.

Interface `List` is implemented by several classes including

`ArrayList`

`LinkedList`

`Vector`

`List` is an interface…

What does that tell you about `List`??

Generic Collections

`List`

`ArrayList` and `Vector` are resizable-array implementations of `List`.

Inserting an element between existing elements is an inefficient operation.

A `LinkedList` is more efficient for insertions and removals because it is not based on an array.

Knowing about the different `Collection` classes can help you pick the right one for the job.

Generic Collections

ArrayList vs Vector

ArrayList and Vector have nearly identical behavior

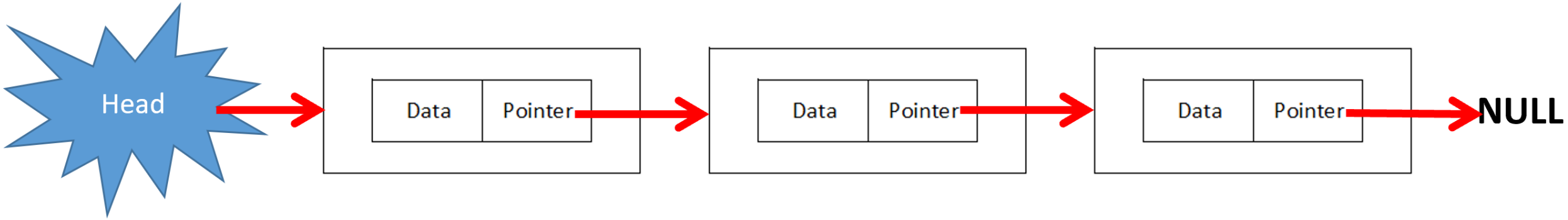
Vector existed before Collections was added to Java so it has some methods that are not part of the List interface; therefore, not found in ArrayList.

ArrayLists behave like Vectors but are faster because they do not have the overhead of thread synchronization.

Generic Collections

`LinkedList`

Remember those from CSE1320?



```
String[] colors = {"red", "orange", "yellow", "blue"};

LinkedList<String> links = new LinkedList<>(Arrays.asList(colors));

for (String it :  links)
    System.out.printf("%s ", it);
```

```
links.addLast("green");
links.add(2, "indigo");
links.add("violet");
links.addFirst("Rainbow : ");

for (String it :  links)
    System.out.printf("%s ", it);
```

```
red orange yellow blue

red orange yellow blue green

red orange indigo yellow blue green

red orange indigo yellow blue green violet
```

```
Rainbow : red orange indigo yellow blue green violet
```

Generic Collections

asList

```
LinkedList<String> links = new LinkedList<>(Arrays.asList(colors));
```

`asList` is a method in `Arrays`.

It returns a `List` view of the array.

That `List` view of the array is then passed to `LinkedList`'s constructor.

`LinkedList` is able to accept a `Collection` as an argument.

Generic Collections

asList

```
String[] colors = {"red", "orange", "yellow", "blue"};

ArrayList<String> Yarn = new ArrayList<>(Arrays.asList(colors));

for (String it :  Yarn)
    System.out.printf("%s ", it);
```

```
red orange yellow blue
```

Generic Collections

Class `Collections` provides algorithms for manipulating collection elements.

These algorithms are implemented as `static` methods.

Method	Description
`sort`	Sorts the elements of a `List`.
`binarySearch`	Locates an object in a `List`, using binary search
`reverse`	Reverses the elements of a `List`.
`shuffle`	Randomly orders a `List`'s elements.
`fill`	Sets every `List` element to refer to a specified object.
`copy`	Copies references from one `List` into another.

```
String[] colors = {"red", "orange", "yellow", "blue", "green"};

ArrayList<String>rainbow = new ArrayList<>(Arrays.asList(colors));

Collections.sort(rainbow);
System.out.println(rainbow);
```

[blue, green, orange, red, yellow]

```
Collections.shuffle(rainbow);
System.out.println(rainbow);
```

[orange, red, yellow, green, blue]

```
Collections.shuffle(rainbow);
System.out.println(rainbow);
```

[red, green, orange, blue, yellow]

```
Collections.reverse(rainbow);
System.out.println(rainbow);
```

[yellow, blue, orange, green, red]

```
Collections.fill(rainbow, "Skittles");
System.out.println(rainbow);
```

[Skittles, Skittles, Skittles, Skittles, Skittles]

Generic Collections

```
String[] colors = {"red", "orange", "yellow", "blue", "green"};
String[] copyColors = {"magenta", "teal", "pink"};

ArrayList<String>rainbow = new ArrayList<>(Arrays.asList(colors));
ArrayList<String>copyRainbow = new ArrayList<>(Arrays.asList(copyColors));

System.out.println(rainbow);
System.out.println(copyRainbow);
```

```
[red, orange, yellow, blue, green]
[magenta, teal, pink]
```

```
Collections.copy(rainbow, copyRainbow);

System.out.println(rainbow);
System.out.println(copyRainbow);
```

```
[magenta, teal, pink, blue, green]
[magenta, teal, pink]
```

Generic Collections

```
String[] colors = {"red", "orange", "yellow", "blue", "green"};

ArrayList<String>rainbow = new ArrayList<>(Arrays.asList(colors));
ArrayList<String>copyRainbow = new ArrayList<>();

System.out.println(rainbow);
System.out.println(copyRainbow);

Collections.copy(copyRainbow, rainbow);

System.out.println(rainbow);
System.out.println(copyRainbow);
```

Generic Collections

```
[red, orange, yellow, blue, green]
[]
```

```
Collections.copy(copyRainbow, rainbow);
```

```
Exception in thread "main" java.lang.IndexOutOfBoundsException: Source does not fit in dest
at java.base/java.util.Collections.copy(Collections.java:561)
at collectalgorithms.CollectionAlgorithms.main(CollectionAlgorithms.java:22)
C:\Users\frenc\Documents\NetBeansProjects\CollectAlgorithms\nbproject\build-impl.xml:1355: The following error occurred while executing this line:
C:\Users\frenc\Documents\NetBeansProjects\CollectAlgorithms\nbproject\build-impl.xml:961: Java returned: 1
BUILD FAILED (total time: 1 second)
```

Generic Collections

To make a copy of an `ArrayList` using `copy()`, the destination `ArrayList` must not only already exist, it must contain at least as many elements as you want to copy.

So, before running the `copy()` method, we need to create an empty array

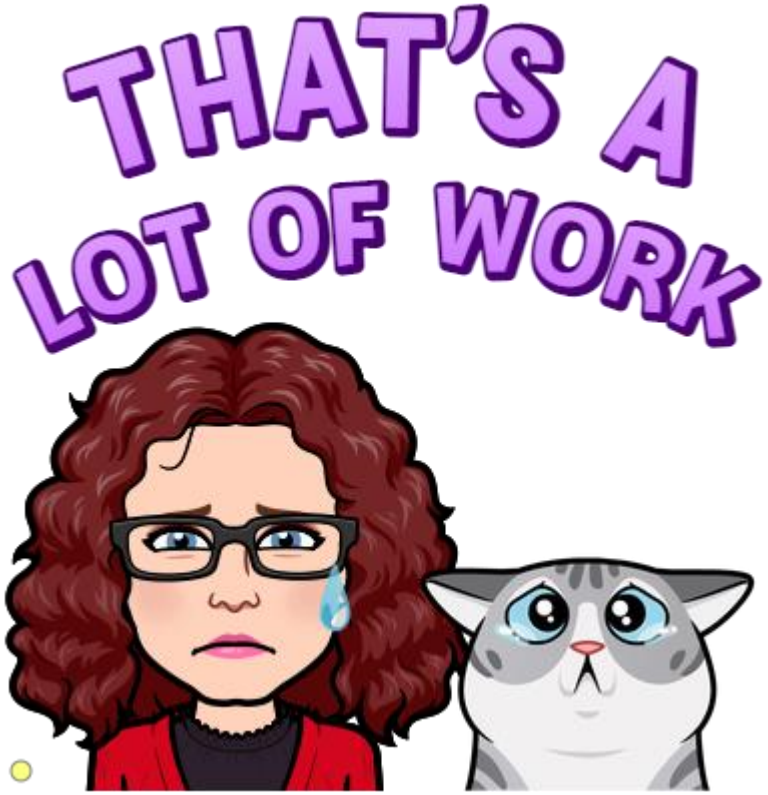
```
String[] it = new String[5];
```

and then construct our "empty" `ArrayList` using the empty array

```
ArrayList<String>copyRainbow = new ArrayList<>(Arrays.asList(it));
```

```
String[] colors = {"red", "orange", "yellow", "blue", "green"};
String[] copyColors = new String[5];

ArrayList<String>rainbow = new ArrayList<>(Arrays.asList(colors));
ArrayList<String>copyRainbow = new ArrayList<>(Arrays.asList(copyColors));

System.out.println(rainbow);
System.out.println(copyRainbow);

Collections.copy(copyRainbow, rainbow);

System.out.println(rainbow);
System.out.println(copyRainbow);
```

```
[red, orange, yellow, blue, green]
[null, null, null, null, null]
[red, orange, yellow, blue, green]
[red, orange, yellow, blue, green]
```



```java
String[] colors = {"red", "orange", "yellow", "blue", "green"};

ArrayList<String>rainbow = new ArrayList<>(Arrays.asList(colors));
ArrayList<String>copyRainbow = new ArrayList<>(rainbow);

System.out.println(rainbow);
System.out.println(copyRainbow);

Collections.copy(copyRainbow, rainbow);

System.out.println(rainbow);
System.out.println(copyRainbow);
rainbow.remove(2);
System.out.println(rainbow);
System.out.println(copyRainbow);
```

```
[red, orange, yellow, blue, green]
[red, orange, yellow, blue, green]
[red, orange, yellow, blue, green]
[red, orange, yellow, blue, green]
[red, orange, blue, green]
[red, orange, yellow, blue, green]
```

Generic Collections

ArrayList

Constructor Summary

Constructors

Constructor and Description
`ArrayList()` Constructs an empty list with an initial capacity of ten.
`ArrayList(Collection<? extends E> c)` Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.
`ArrayList(int initialCapacity)` Constructs an empty list with the specified initial capacity.

`Iterator`

An `Iterator` is an object that is pointing to some element in a range of elements that has the ability to iterate through the elements of that range.

An **`Iterator`** is an object that can traverse (iterate over) a `Collection` class without the user having to know how the `Collection` is implemented. With many `Collection` classes, iterators are the primary way elements of these classes are accessed.

Iterators provide an easy way to step through the elements of a `Collection` class without having to understand how the `Collection` class is implemented.

Iterator

We have been using iterators for a while now in our enhanced for loops.

```
ArrayList<String> Yarn

for (String it :  Yarn)
   System.out.printf("%s ", it);

ArrayList<CokeMachine> CM

for (CokeMachine it : CM)
   System.out.printf("%s", it.getMachineName());
```

Iterator

C
C++
Java
Python

Can we use iterators in other places besides an enhanced `for` loop?

`Iterator` is an `interface` so it has methods.

```
String Filler[] = {"C", "C++", "Java", "Python"};
ArrayList<String> PL = new ArrayList<>(Arrays.asList(Filler));

Iterator<String>it = PL.iterator();

while (it.hasNext())
{
    System.out.println(it.next());
}
```

`iterator()` is a method in `interface Collection` and returns an iterator over the elements in the collection

```
String Filler[] = {"C", "C++", "Java", "Python"};
ArrayList<String> PL = new ArrayList<>(Arrays.asList(Filler));
String Element = null;

Iterator<String>it = PL.iterator();
System.out.println(PL);

while (it.hasNext())
{
    Element = it.next();
    if (Element.contains("a"))
    {
        it.remove();
    }
}

System.out.println(PL);
```

```
[C, C++, Java, Python]
[C, C++, Python]
```

Iterator

```
while (it.hasNext())
{
    Element = it.next();
    if (Element.contains("a"))
    {
        it.remove();
    }
}
```

```
for (String it : PL)
{
    if (it.contains("a"))
    {
        it.remove();
    }
}
```

cannot find symbol
  symbol: method remove()
  location: variable it of type String

Map

Interface `Map` provides operations for manipulating values associated with keys (these values are sometimes referred to as mapped values).

An object that implements `Map` can perform fast storage and retrieval of *unique keys* and *associated values*.

`HashMap` and `Hashtable` implement the `Map` interface.

We are going to use `HashMap`.

Map

A **`HashMap`** is a collection where each element is a pair, called a key/value pair.

The key is used for sorting and indexing the data and must be unique.

The value is the actual data.

Duplicate keys are *not* allowed—a single value can be associated with each key.

This is called a one-to-one mapping.

Map

A map of students where **id number** is the key and **name** is the value can be represented graphically as

1120217 is the key whose value

is Nikhilesh.

This mapping is one to one

One key to one value

1120217	Nikhilesh
1120236	Navneet
1120250	Vikas
1120255	Doodrah

Keys

values

Map

We can use a map to quickly count how many items of something we have.

```
Enter a string : Do you hear an echo echo echo in this room room
room?

The word in appears 1 times
The word this appears 1 times
The word echo appears 3 times
The word do appears 1 times
The word an appears 1 times
The word you appears 1 times
The word hear appears 1 times
The word room appears 2 times
The word room? appears 1 times
```

Map

We can use a map to quickly count how many items of something we have WITHOUT using parsing to count.

```
Enter a string :

I have a cat named Shade and a cat named Sylvester.

You have 2 cats
```

HashMap

```
HashMap<keyType, valueType> myMap = new HashMap<>();
```

keyType and *valueType* can be the same types of values we used for `ArrayList`

`String, Integer, Character, Double, Boolean, Byte, Float, Long, Short`

HashMap

```
HashMap<String, Integer> myMap = new HashMap<>();
```

This creates a mapping of `String` keys who each have an integer value mapped to them.

Key	Value
Cat	2
Dog	3
Bird	1

HashMap

Let's use a `HashMap` to figure out how many times each word appears in a sentence.

```
My cat is named Shade and my cat is black and my cat is 12 years old.
```

```
The word 12 appears 1 times
The word named appears 1 times
The word old. appears 1 times
The word and appears 2 times
The word cat appears 3 times
The word shade appears 1 times
The word black appears 1 times
The word is appears 3 times
The word my appears 3 times
The word years appears 1 times
```

Key	Value
12	1
named	1
old.	1
and	2
cat	3
shade	1
black	1
is	3
my	3
years	1

HashMap

```
public static void main(String[] args)
{
    HashMap<String, Integer> myMap = new HashMap<>();

    createMap(myMap);
    displayMap(myMap);
}
```

```java
private static void createMap(Map<String, Integer> map)
{
    Scanner in = new Scanner(System.in);
    String word = null;
    int count = 0;

    System.out.print("Enter a string : ");
    String input = in.nextLine();

    String[] tokens = input.split(" ");

    for (String token  : tokens)
    {
        word = token.toLowerCase();

        if (map.containsKey(word))
        {
            count = map.get(word);
            map.put(word, ++count);
        }
        else
        {
            map.put(word, 1);
        }
    }
}
```

```
private static void createMap(Map<String, Integer> map)
{
    Scanner in = new Scanner(System.in);
    String word = null;
    int count = 0;

    System.out.print("Enter a string : ");
    String input = in.nextLine();

    String[] tokens = input.split(" ");
```

```
String[] tokens = input.split(" ");

for (String token  : tokens)
{
    word = token.toLowerCase();

    if (map.containsKey(word))
    {
        count = map.get(word);
        map.put(word, ++count);
    }
    else
    {
        map.put(word, 1);
    }
}
```

Checks if `word` is a key in `map`

Gets the value associated with the key `word`

Put the (key, value) pair into the map

My cat is named Shade and my cat is black and my cat is 12 years old.

```
String[] tokens = input.split(" ");

for (String token  : tokens)
{
    word = token.toLowerCase();

    if (map.containsKey(word))
    {
        count = map.get(word);
        map.put(word, ++count);
    }
    else
    {
        map.put(word, 1);
    }
}
```

Key	Value

My cat is named Shade and my cat is black and my cat is 12 years old.

HashMap

Key	Value
my	3
cat	3
is	3
named	1
shade	1
and	2
black	1
12	1
years	1
old.	1

Key	Value
12	1
named	1
old.	1
and	2
cat	3
shade	1
black	1
is	3
my	3
years	1

```
private static void displayMap(Map<String,Integer>map)
{
    String key = null;
    int value = 0;

    for (Map.Entry mapElement : map.entrySet())
    {
        key = (String)mapElement.getKey();


        value = (int)mapElement.getValue();


        System.out.printf("The word %s appears %d times\n", key, value);
    }
}
```

HashMap

```
for (Map.Entry mapElement : map.entrySet())
```

`Map.Entry` is the type of a (key,value) pair from a `HashMap`

`mapElement` is an iterator that is a (key,value) pair

`map.entrySet()` – returns a `Set` of the mappings

HashMap

```
key = (String)mapElement.getKey();
```

`mapElement.getKey()` retrieves the key from the (key,value) pair

Adding `(String)` casts the `key` to a `String`

```
incompatible types: Object cannot be converted to String
----
(Alt-Enter shows hints)
```

HashMap

```
value = (int)mapElement.getValue();
```

`mapElement.getValue()` retrieves the value from the (key,value) pair

Adding `(int)` casts the `key` to a `int`

```
incompatible types: Object cannot be converted to int
----
(Alt-Enter shows hints)
```

```
private static void displayMap(Map<String,Integer>map)
{
    String key = null;
    int value = 0;

    for (Map.Entry mapElement : map.entrySet())
    {
        key = (String)mapElement.getKey();


        value = (int)mapElement.getValue();


        System.out.printf("The word %s appears %d times\n", key, value);
    }
}
```

Key	Value
12	1
named	1
old.	1
and	2
cat	3
shade	1
black	1
is	3
my	3
years	1

HashMap

```
System.out.printf("\nsize : %d\nisEmpty : %b\n",
                  map.size(), map.isEmpty());
```

```
size : 10
isEmpty : false
```

`size()` – number of (key,value) pairs in map

`isEmpty()` – `true` if map is empty,
`false` if not empty

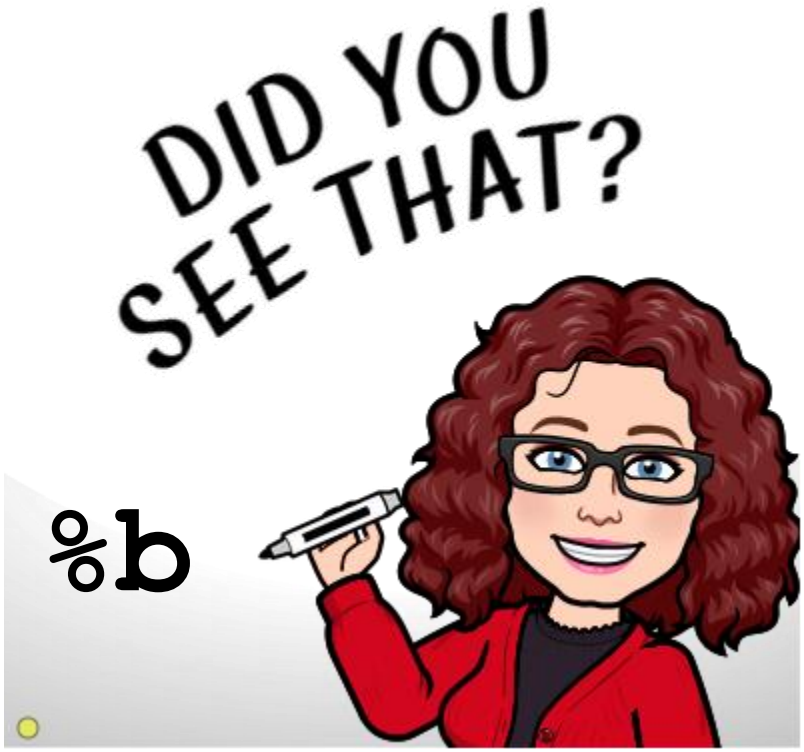
Key	Value
12	1
named	1
old.	1
and	2
cat	3
shade	1
black	1
is	3
my	3
years	1

HashMap

```
System.out.printf("\nsize : %d\nisEmpty : %b\n",
                  map.size(), map.isEmpty());
```

`%b` is the conversion specifier to print a `boolean` value

`isEmpty()` returns `true` or `false`



HashMap

```
System.out.print("Enter a word to search for : ");
String word = in.next();

System.out.printf("The word %s appears %d times\n",
                  word, map.get(word));
```

`get()` takes a key and returns the value mapped to that key

HashMap

```
Enter a string : My cat is named Shade and my cat is black and my cat is 12
years old.

The word 12 appears 1 times
The word named appears 1 times
The word old. appears 1 times
The word and appears 2 times
The word cat appears 3 times
The word shade appears 1 times
The word black appears 1 times
The word is appears 3 times
The word my appears 3 times
The word years appears 1 times
Enter a word to search for : cat
The word cat appears 3 times
```

```
map.get(word));
```

HashMap

```
Enter a string : My cat is named Shade and my cat is black and my cat is 12
years old.

The word 12 appears 1 times
The word named appears 1 times
The word old. appears 1 times
The word and appears 2 times
The word cat appears 3 times
The word shade appears 1 times
The word black appears 1 times
The word is appears 3 times
The word my appears 3 times
The word years appears 1 times
Enter a word to search for : cat
[1, 1, 1, 2, 3, 1, 1, 3, 3, 1]
```

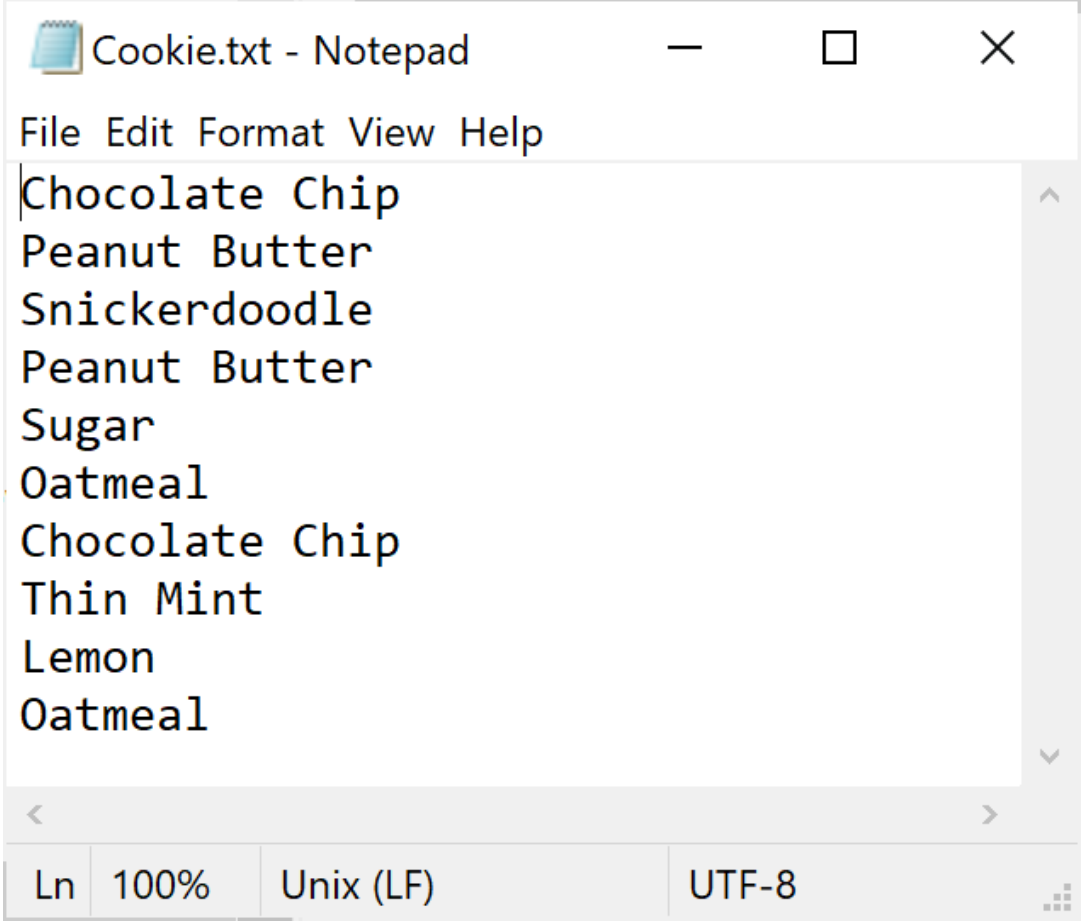
```
System.out.println
(map.values());
```

`values()` returns a listing of the values from the map

In Class Exercise

Create a complete Java program that uses a `HashMap` to count how many cookies of each type are in our Cookie.txt file.

```
Thin Mint - 1
Peanut Butter - 2
Chocolate Chip - 2
Snickerdoodle - 1
Oatmeal - 2
Lemon - 1
Sugar - 1
```



Step 1 – Complete Java program

```
import java.io.File;
import java.util.HashMap;
import java.util.Scanner;
int CookieCount = 0;
package cookiemap;
public class CookieMap
public static void main(String[] args)
String CookieName = null;
HashMap<String, Integer>CookieMap = new HashMap<>();
```

Step 2 – Open File

```
catch (Exception e)
File FH  = new File("Cookie.txt");
FileReader = new Scanner(FH);
Scanner FileReader = null;
String FileLine = null;
System.exit(0);
System.out.printf("Cookie.txt does not exist...exiting\n");
try
```

Step 2 – Open File and Read it

```
FileLine = FileReader.nextLine();
FileReader.close();
while (FileReader.hasNextLine())

// Leave lots room for the map code
```

Step 3 – Create map

```
CookieCount = CookieMap.get(FileLine);

CookieMap.put(FileLine, ++CookieCount);

CookieMap.put(FileLine, 1);

else

if (CookieMap.containsKey(FileLine))
```

Step 4 – Print map

```
CookieCount = (int)mapElement.getValue();

CookieName = (String)mapElement.getKey();

for (HashMap.Entry mapElement : CookieMap.entrySet())
```

Learning C++

- C++ can run differently depending on what machine you are using
- We will be using a standard setup that everyone will be required to use
- We will be using
  - C++ 11
  - Linux Ubuntu 64 bit

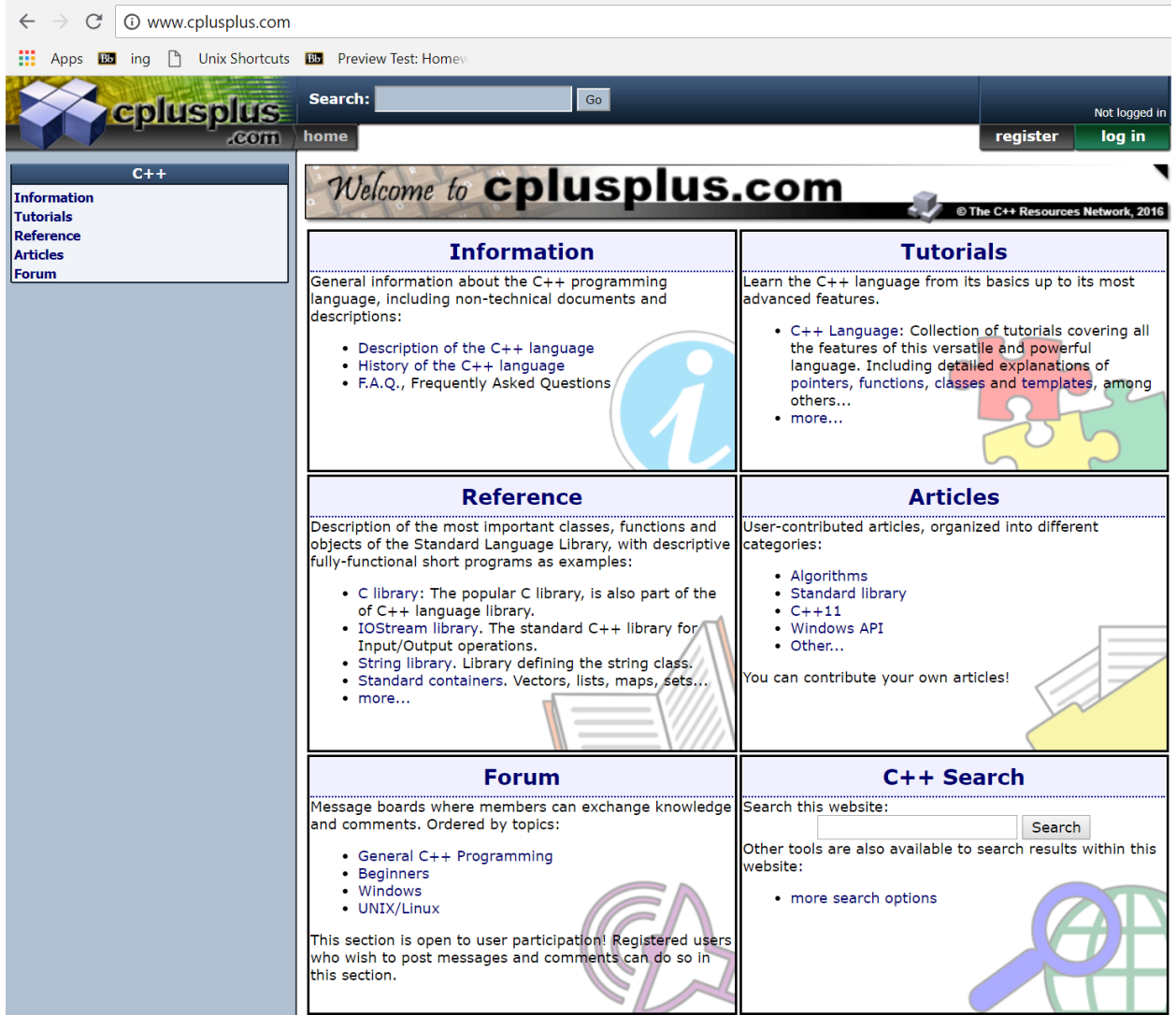
C++ Resources

www.cplusplus.com

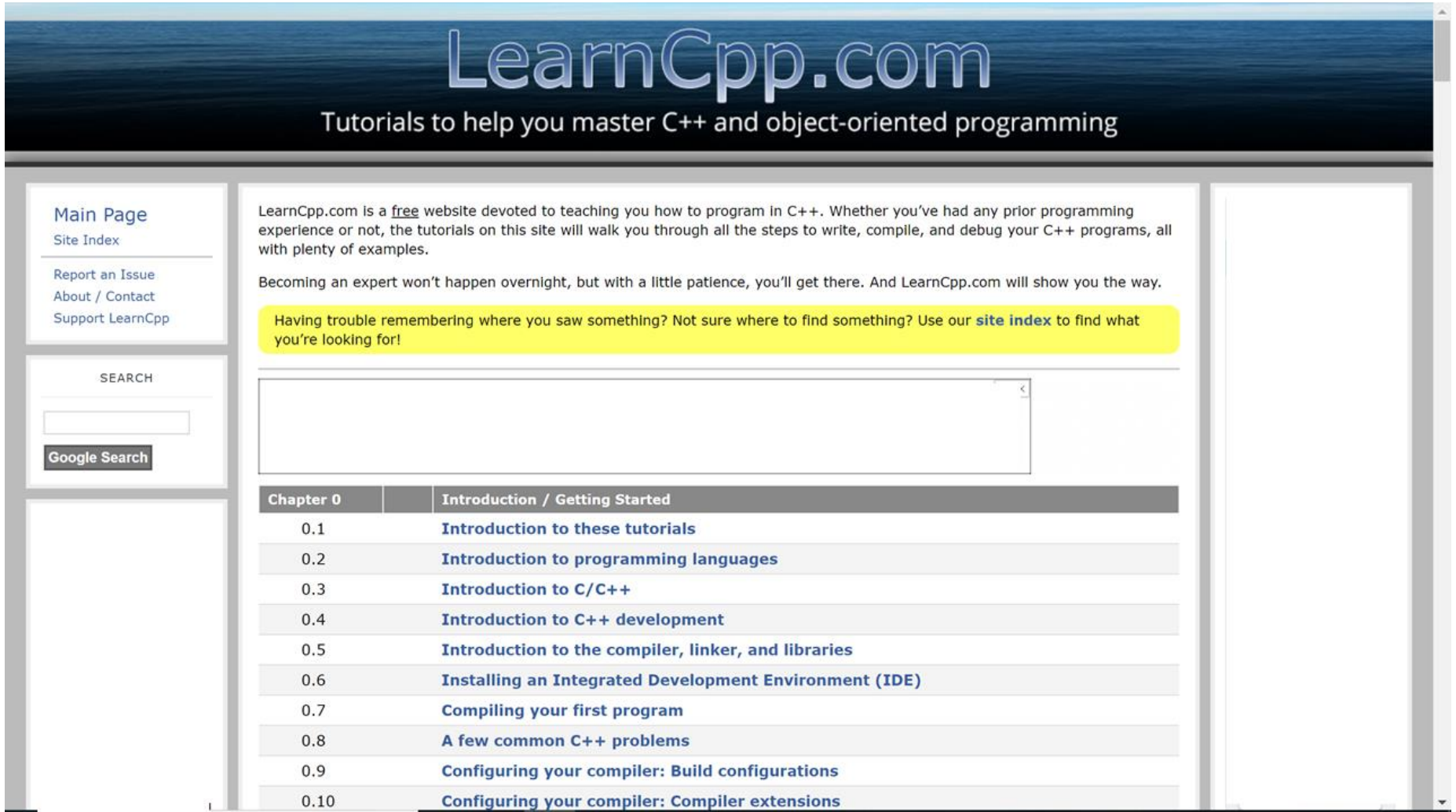
is a good resource

Other resources

- Stack Overflow
- O'Reilly books



Website for CSE 1325
www.learncpp.com



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

makefile

C++ uses a `makefile` just like C.

Change `.c` to `.cpp`

Change `gcc` to `g++`

```
#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)
```

Hello World

In C

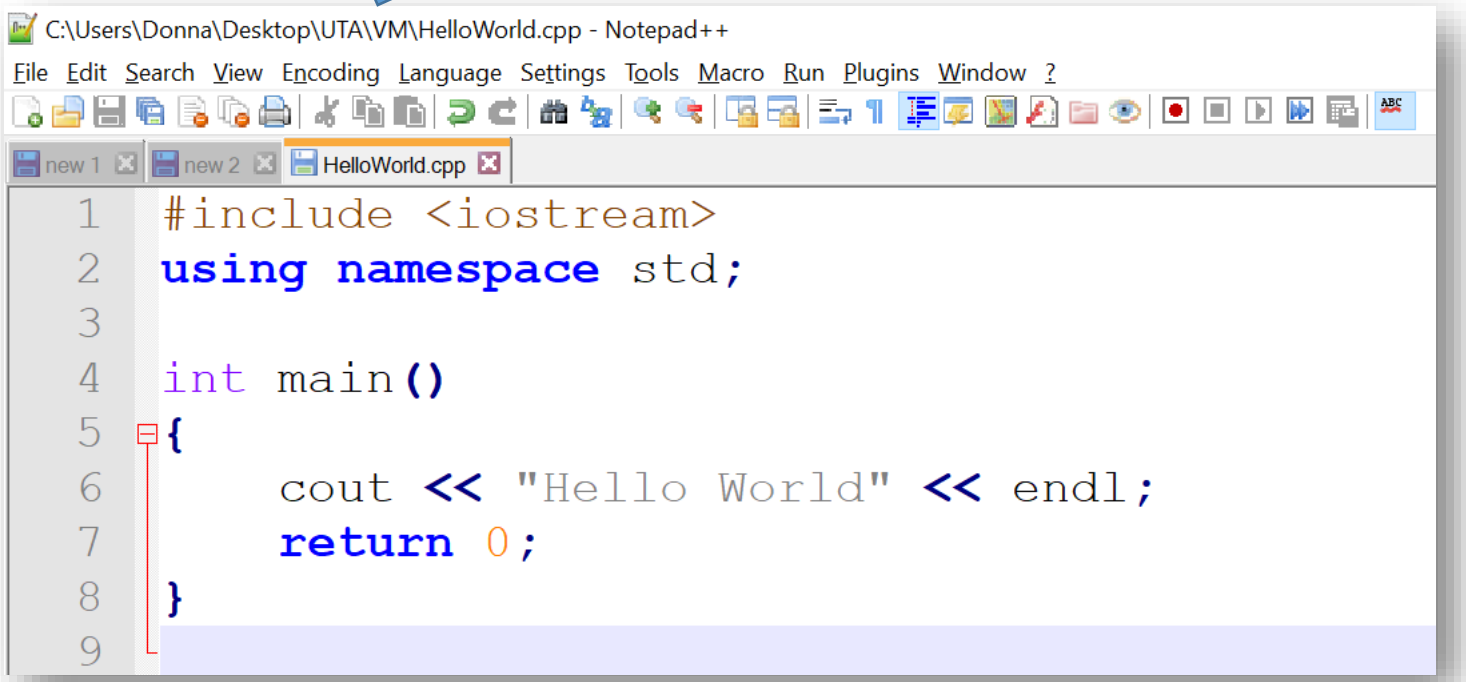
```
#include <stdio.h>
int main(void)
{
   printf("Hello World\n");
   return 0;
}
```

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.



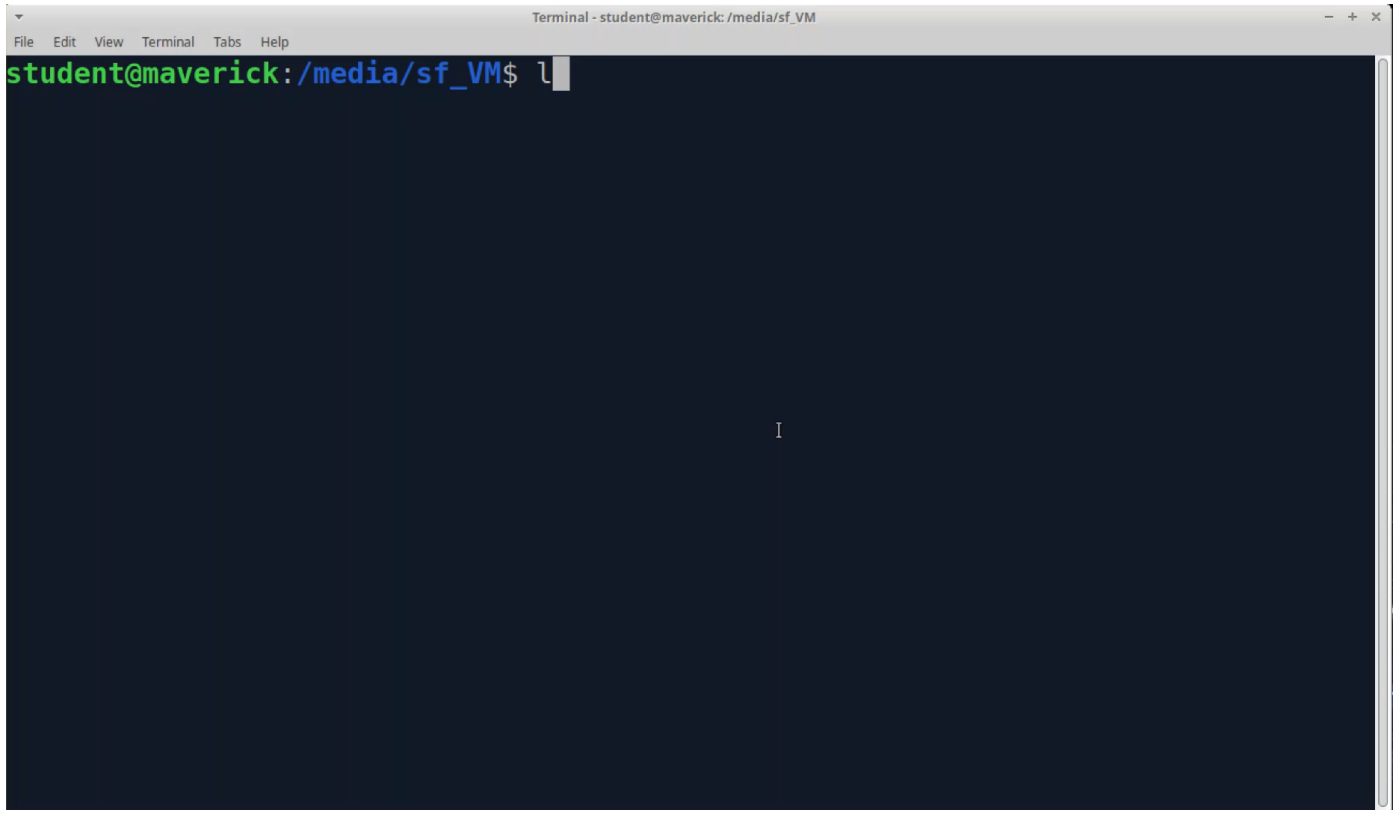
Hello World

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
```



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.

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`

Standard Stream Objects

`cin`

- `istream` object
- "connected to" the standard input device
- uses stream extraction operator >>

```
int grade;
cin >> grade;
```

`cout`

- `ostream` object
- "connected to" the standard output device
- uses stream insertion operator <<

```
cout << grade;
```

Hello World Plus

```
student@maverick:/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
<<	>>