10.2 The string Class

Monday, March 6, 2023 10:08 AM

- string class defines the string type in C++, has many useful functions to manipulate strings
- 2 ways to process strings
    - Treat as arrays of chars, end w/ null terminator ('\0'), these are C-strings
    - Process strings using string class, can use C-string to manipulate, but string is easier
- "string type" , may know how to get string char using the 'at(index)' function, and subscript operator [], and 'size()' and 'length()' functions for number of characters in a string
-

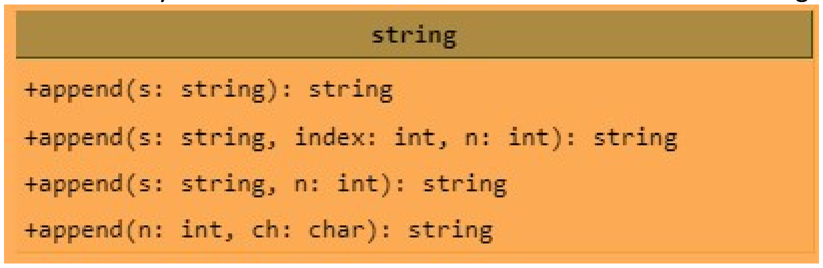
10.2.1 Constructing a String

Monday, March 6, 2023 10:13 AM

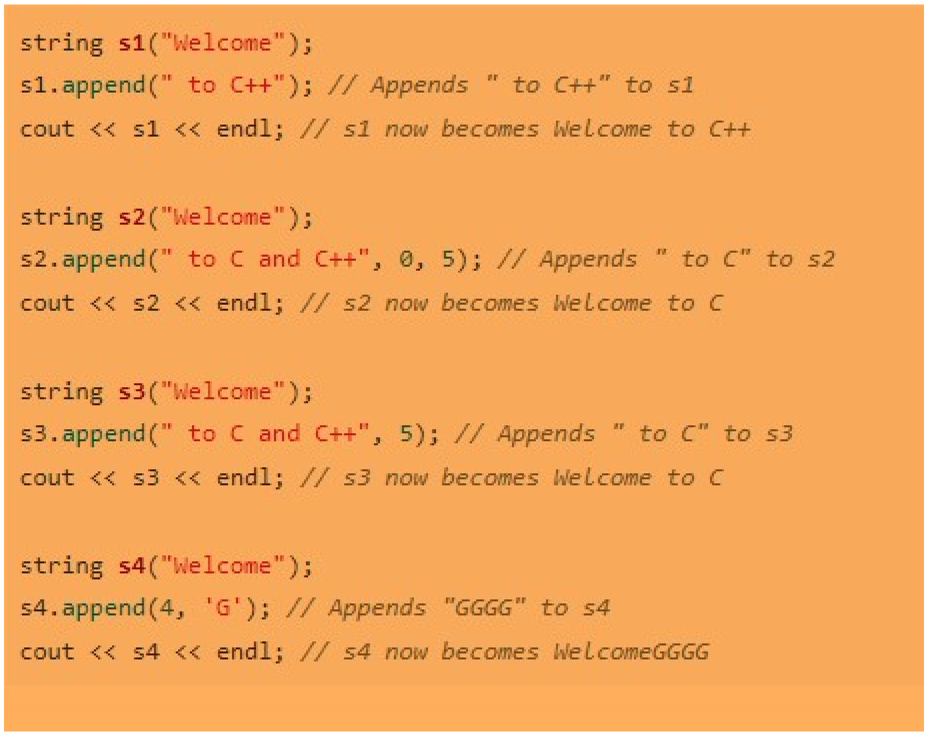
- Use syntax: string s = "Welcome to C++"
- Can also use syntax: string s("Welcome to C++");
- Can also make empty string by no-arg constructor: string s;
- Also can make a string w/ C-string string's constructor:
    - char s1[] = "Good morning";
    - string s(s1);

10.2.2 Appending to a String

Thursday, March 9, 2023 3:14 PM

- Can use many overloaded functions to add new contents to a string
- 

```
string
+append(s: string): string
+append(s: string, index: int, n: int): string
+append(s: string, n: int): string
+append(n: int, ch: char): string
```

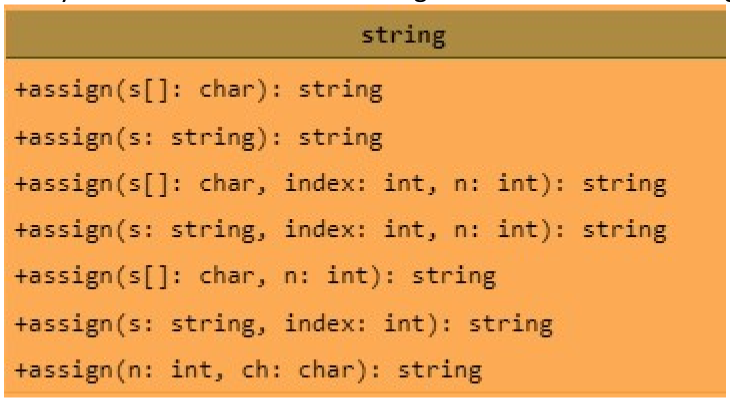
- 

```
string s1("Welcome");
s1.append(" to C++"); // Appends " to C++" to s1
cout << s1 << endl; // s1 now becomes Welcome to C++

string s2("Welcome");
s2.append(" to C and C++", 0, 5); // Appends " to C" to s2
cout << s2 << endl; // s2 now becomes Welcome to C

string s3("Welcome");
s3.append(" to C and C++", 5); // Appends " to C" to s3
cout << s3 << endl; // s3 now becomes Welcome to C

string s4("Welcome");
s4.append(4, 'G'); // Appends "GGGG" to s4
cout << s4 << endl; // s4 now becomes WelcomeGGGG
```

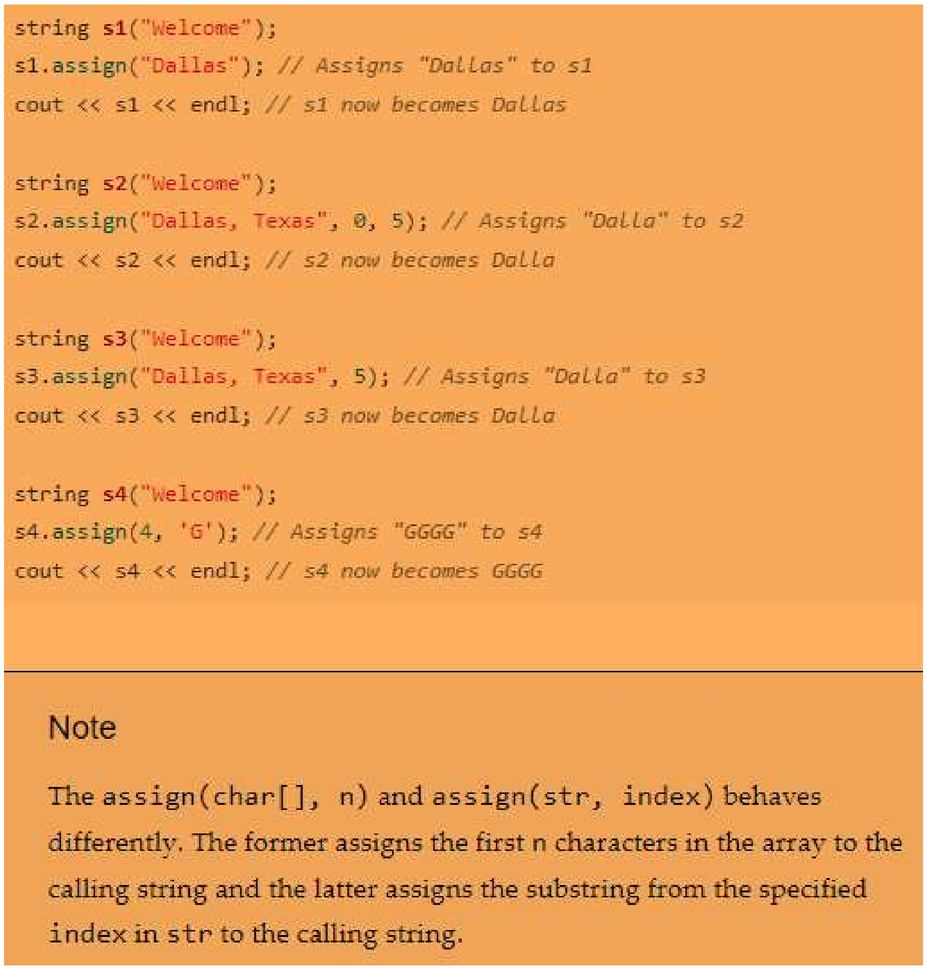
-
-

10.2.3 Assigning a String

Thursday, March 9, 2023 3:18 PM

- Many overloaded functions to assign new contents to a string
- 

```
string
+assign(s[]: char): string
+assign(s: string): string
+assign(s[]: char, index: int, n: int): string
+assign(s: string, index: int, n: int): string
+assign(s[]: char, n: int): string
+assign(s: string, index: int): string
+assign(n: int, ch: char): string
```

-
- 

```
string s1("Welcome");
s1.assign("Dallas"); // Assigns "Dallas" to s1
cout << s1 << endl; // s1 now becomes Dallas

string s2("Welcome");
s2.assign("Dallas, Texas", 0, 5); // Assigns "Dalla" to s2
cout << s2 << endl; // s2 now becomes Dalla

string s3("Welcome");
s3.assign("Dallas, Texas", 5); // Assigns "Dalla" to s3
cout << s3 << endl; // s3 now becomes Dalla

string s4("Welcome");
s4.assign(4, 'G'); // Assigns "GGGG" to s4
cout << s4 << endl; // s4 now becomes GGGG
```

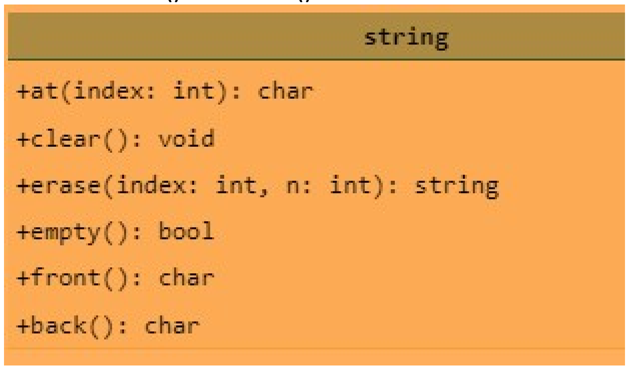
Note

The `assign(char[], n)` and `assign(str, index)` behaves differently. The former assigns the first `n` characters in the array to the calling string and the latter assigns the substring from the specified index in `str` to the calling string.

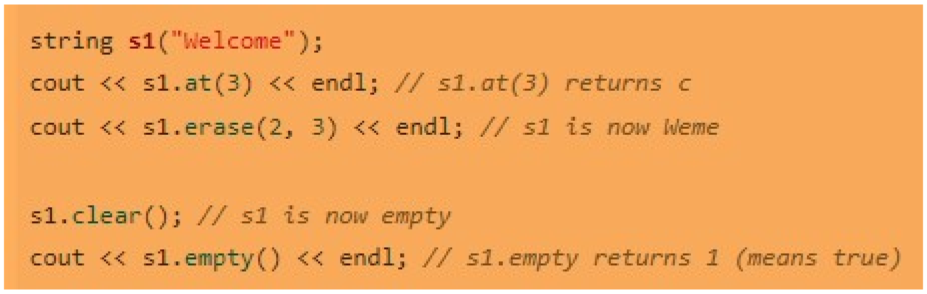
-

10.2.4 Functions at, clear, erase, empty, back, and front

Thursday, March 9, 2023 3:22 PM

- Can use at(index) function to retrieve a character at a specific index
- Can use clear() to clear the string
- Can use erase(index, n) to delete part of the string
- Can use empty() to test if string is empty
- Can use front() and back() to return first and last char from the string
- 

```
string
+at(index: int): char
+clear(): void
+erase(index: int, n: int): string
+empty(): bool
+front(): char
+back(): char
```

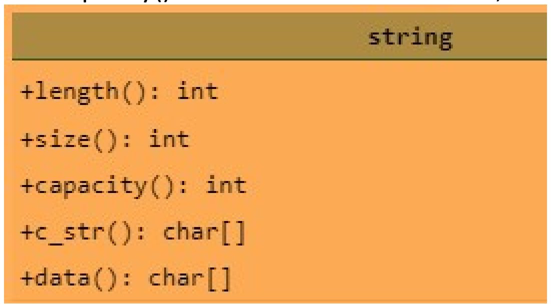
-
- 

```
string s1("Welcome");
cout << s1.at(3) << endl; // s1.at(3) returns c
cout << s1.erase(2, 3) << endl; // s1 is now Weme

s1.clear(); // s1 is now empty
cout << s1.empty() << endl; // s1.empty returns 1 (means true)
```

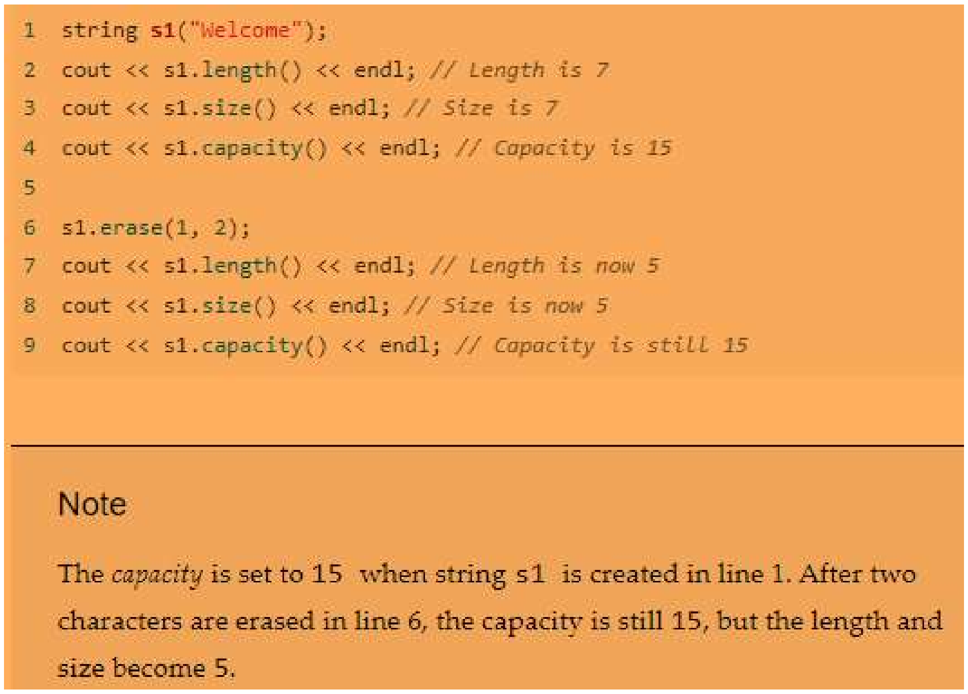
-
-

10.2.5 Functions length, size, capacity, and c_str()

Thursday, March 9, 2023 3:26 PM

- Can use the functions length(), size(), and capacity() to get string's length, size and capacity
- Can usec_str() to return a C-string
- The length() and size() are aliases
- The c_str() and data() are new in C++11
- The capacity() returns internal buffer size, always greater than or equal to actual string size
- 

string
+length(): int
+size(): int
+capacity(): int
+c_str(): char[]
+data(): char[]

- 

```
1  string s1("Welcome");
2  cout << s1.length() << endl; // Length is 7
3  cout << s1.size() << endl; // Size is 7
4  cout << s1.capacity() << endl; // Capacity is 15
5
6  s1.erase(1, 2);
7  cout << s1.length() << endl; // Length is now 5
8  cout << s1.size() << endl; // Size is now 5
9  cout << s1.capacity() << endl; // Capacity is still 15
```

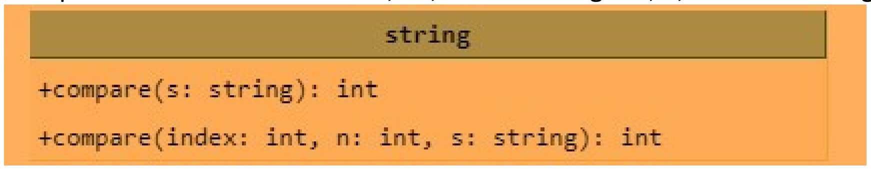
Note

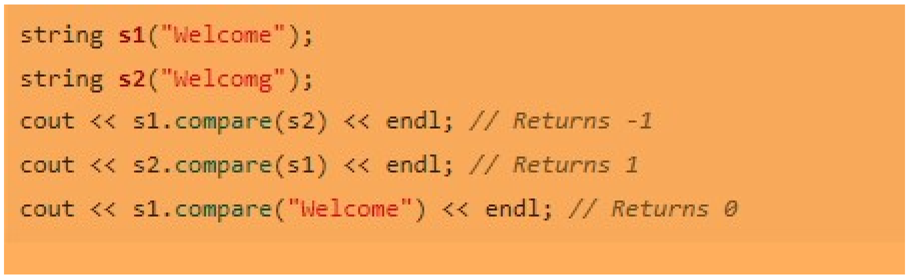
The *capacity* is set to 15 when string s1 is created in line 1. After two characters are erased in line 6, the capacity is still 15, but the length and size become 5.

-

10.2.6 Comparing String

Thursday, March 9, 2023 10:21 PM

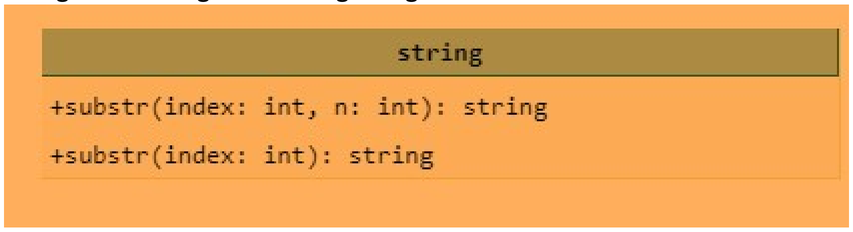
- Usually compare contents of 2 strings, can use compare function
- Compare function returns int >0, =0, or <0 if string is >, =, or < other sting
- 

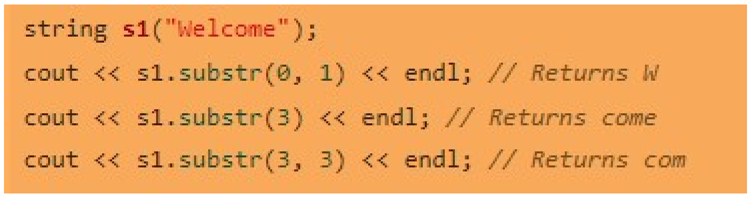
```
string
+compare(s: string): int
+compare(index: int, n: int, s: string): int
```
-
- 

```
string s1("Welcome");
string s2("Welcomg");
cout << s1.compare(s2) << endl; // Returns -1
cout << s2.compare(s1) << endl; // Returns 1
cout << s1.compare("Welcome") << endl; // Returns 0
```
-

10.2.7 Obtaining Substrings

Friday, March 17, 2023 9:58 AM

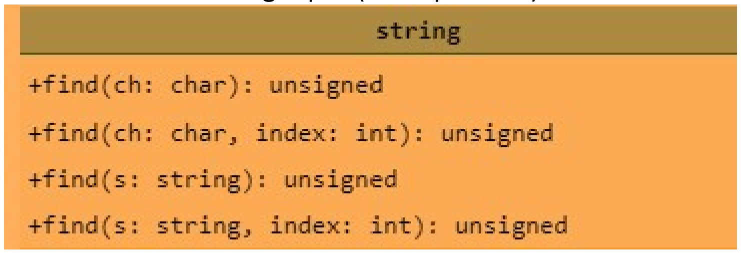
- Can get single char from string using at function
- Can get substring from string using substr function
- 

```
string
+substr(index: int, n: int): string
+substr(index: int): string
```
-
- 

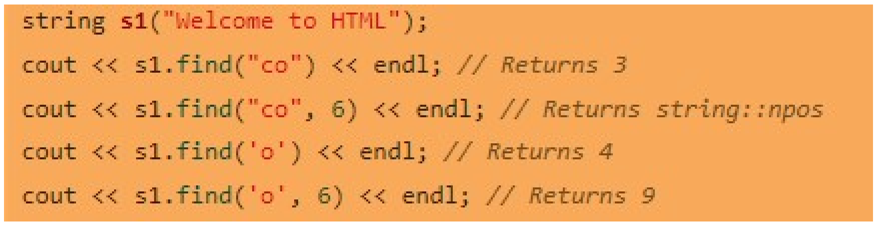
```
string s1("Welcome");
cout << s1.substr(0, 1) << endl; // Returns W
cout << s1.substr(3) << endl; // Returns come
cout << s1.substr(3, 3) << endl; // Returns com
```
-

10.2.8 Searching in a String

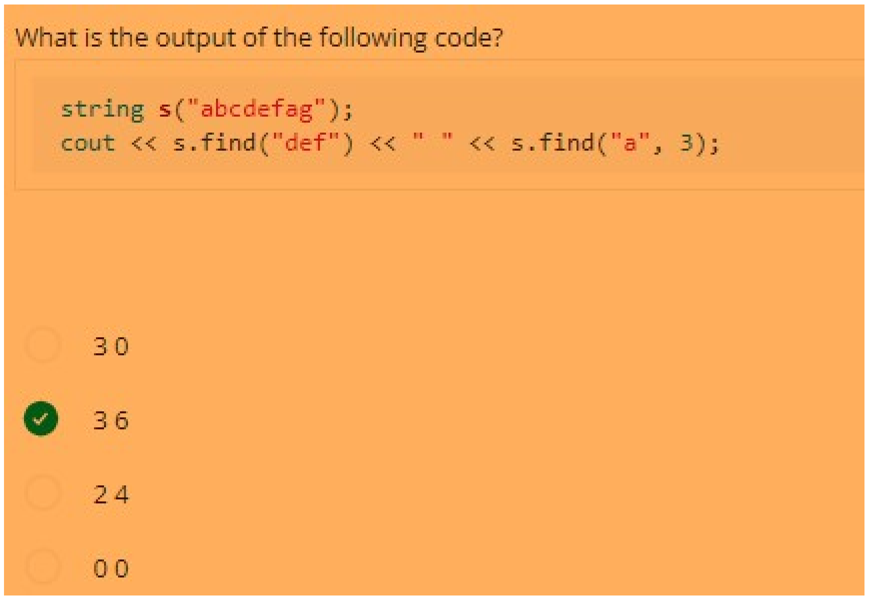
Friday, March 17, 2023 10:00 AM

- Use find function to search for a substring/char in a string
- Function returns string::npos (not a position) if no match found
- 

```
string
+find(ch: char): unsigned
+find(ch: char, index: int): unsigned
+find(s: string): unsigned
+find(s: string, index: int): unsigned
```

- 

```
string s1("Welcome to HTML");
cout << s1.find("co") << endl; // Returns 3
cout << s1.find("co", 6) << endl; // Returns string::npos
cout << s1.find('o') << endl; // Returns 4
cout << s1.find('o', 6) << endl; // Returns 9
```

- 

What is the output of the following code?

```
string s("abcdefag");
cout << s.find("def") << " " << s.find("a", 3);
```

3 0

[✓ 3 6]

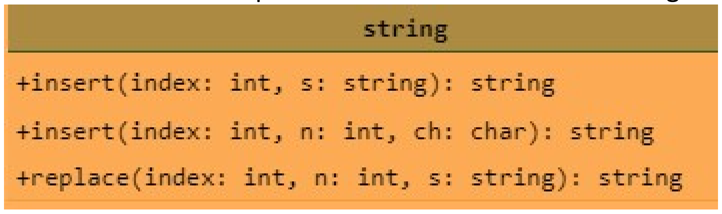
2 4

0 0

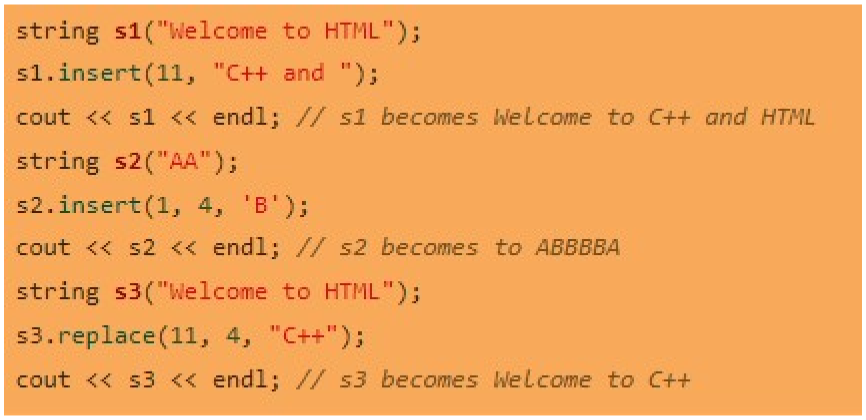
-

10.2.9 Inserting and Replacing Strings

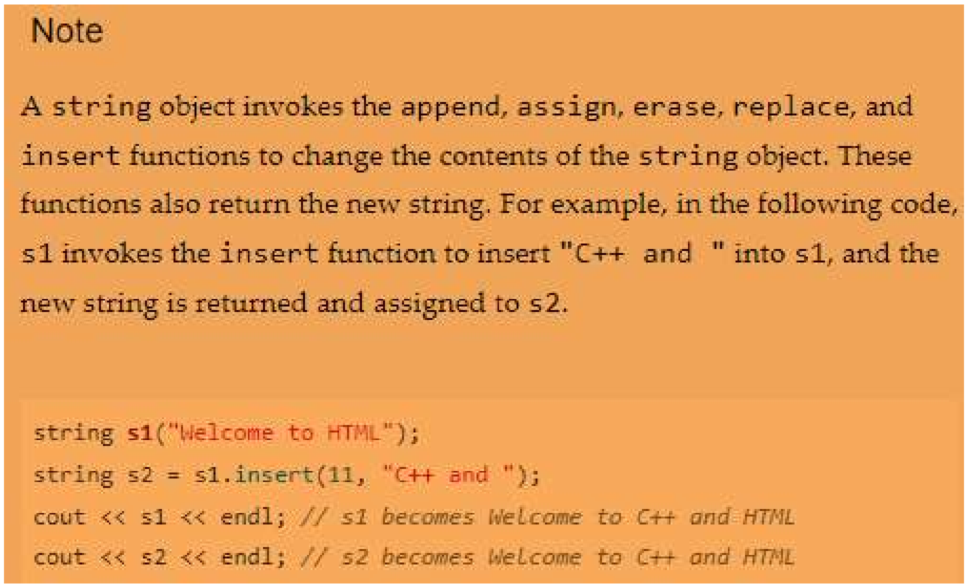
Friday, March 17, 2023 10:04 AM

- Can use insert and replace functions to insert a substring and replace a substring
- 

```
string
+insert(index: int, s: string): string
+insert(index: int, n: int, ch: char): string
+replace(index: int, n: int, s: string): string
```

- 

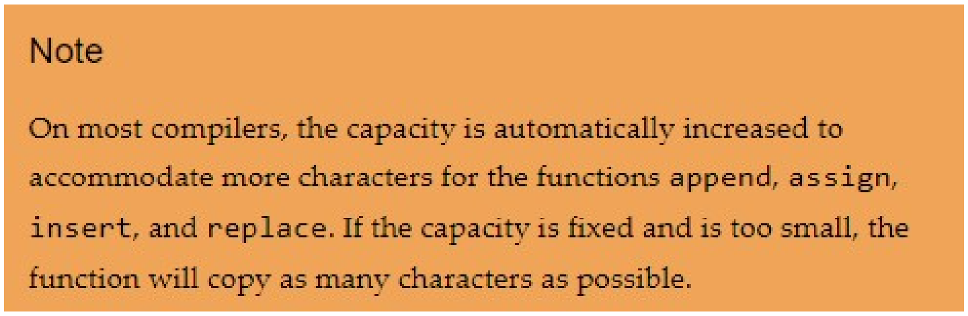
```
string s1("Welcome to HTML");
s1.insert(11, "C++ and ");
cout << s1 << endl; // s1 becomes Welcome to C++ and HTML
string s2("AA");
s2.insert(1, 4, 'B');
cout << s2 << endl; // s2 becomes to ABBBBA
string s3("Welcome to HTML");
s3.replace(11, 4, "C++");
cout << s3 << endl; // s3 becomes Welcome to C++
```

- 

Note

A `string` object invokes the `append`, `assign`, `erase`, `replace`, and `insert` functions to change the contents of the `string` object. These functions also return the new string. For example, in the following code, `s1` invokes the `insert` function to insert `"C++ and "` into `s1`, and the new string is returned and assigned to `s2`.

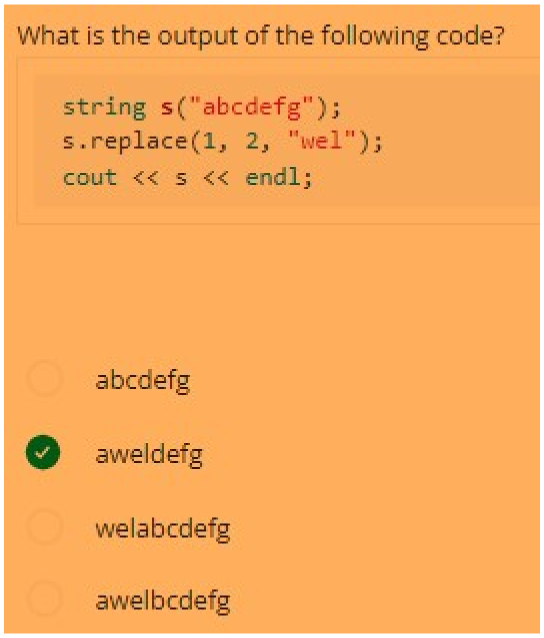
```
string s1("Welcome to HTML");
string s2 = s1.insert(11, "C++ and ");
cout << s1 << endl; // s1 becomes Welcome to C++ and HTML
cout << s2 << endl; // s2 becomes Welcome to C++ and HTML
```

- 

Note

On most compilers, the capacity is automatically increased to accommodate more characters for the functions `append`, `assign`, `insert`, and `replace`. If the capacity is fixed and is too small, the function will copy as many characters as possible.

-

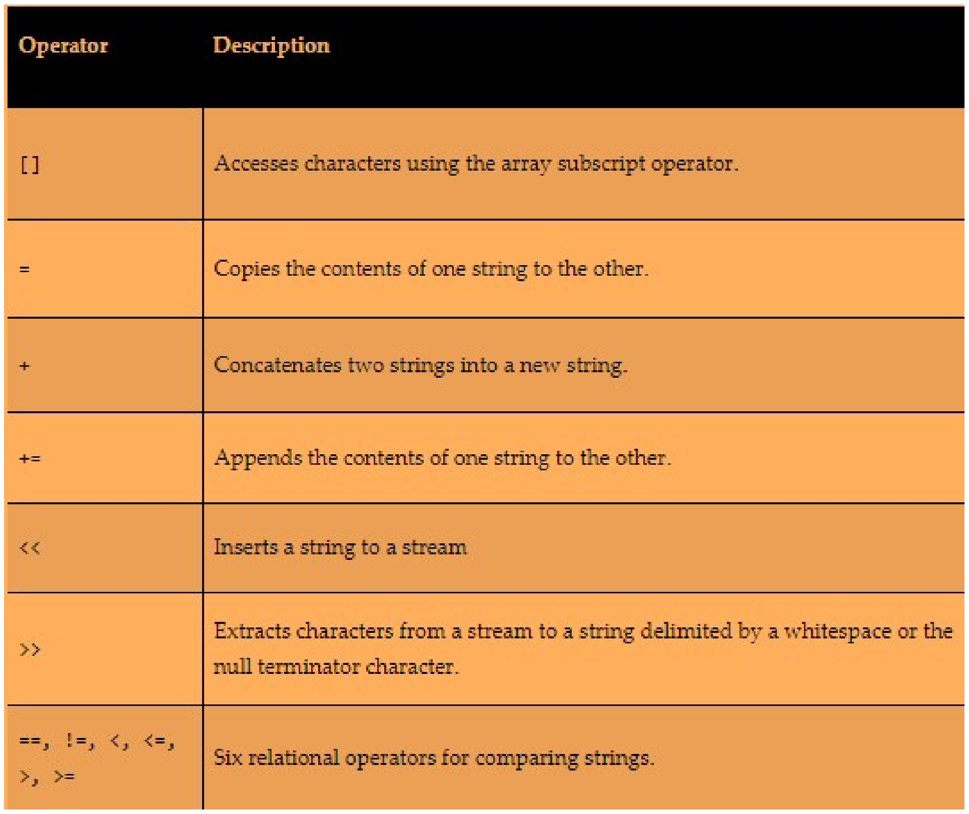

What is the output of the following code?

```
string s("abcdefg");
s.replace(1, 2, "wel");
cout << s << endl;
```

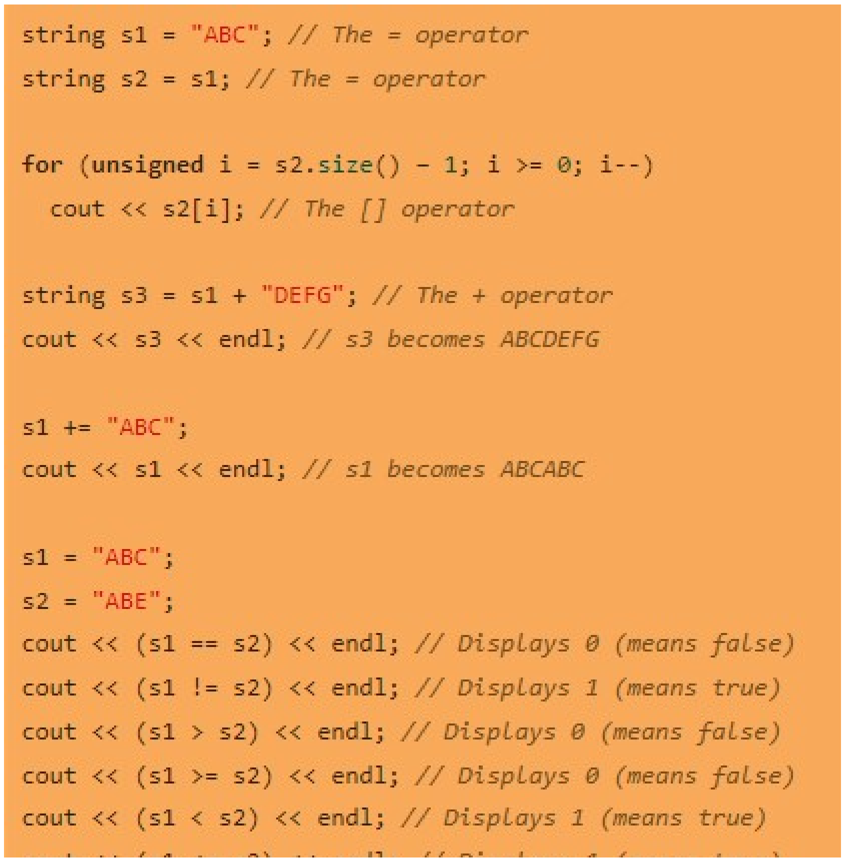
- abcdefg
  [✓ aweldefg]
  welabcdefg
  awelbcdefg
-

10.2.10 String Operators

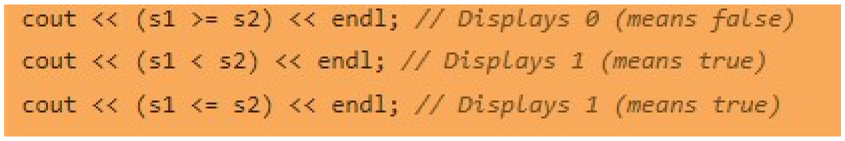
Friday, March 17, 2023 10:09 AM



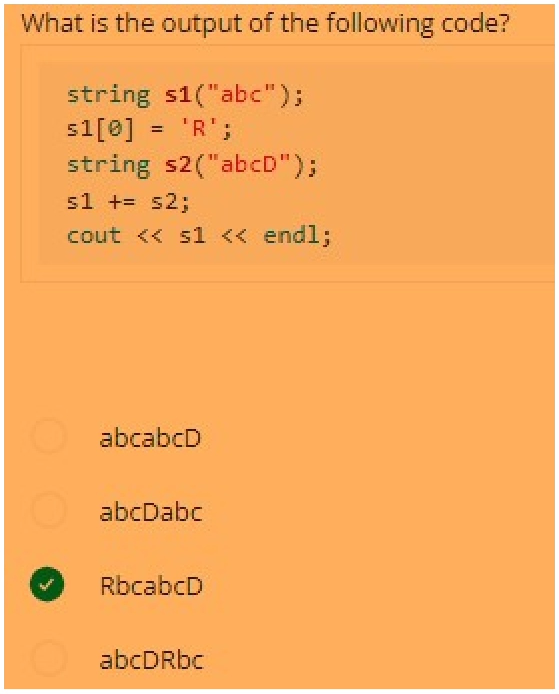
Operator	Description
`[]`	Accesses characters using the array subscript operator.
`=`	Copies the contents of one string to the other.
`+`	Concatenates two strings into a new string.
`+=`	Appends the contents of one string to the other.
`<<`	Inserts a string to a stream
`>>`	Extracts characters from a stream to a string delimited by a whitespace or the null terminator character.
`==, !=, <, <=, >, >=`	Six relational operators for comparing strings.



```
string s1 = "ABC"; // The = operator
string s2 = s1; // The = operator

for (unsigned i = s2.size() - 1; i >= 0; i--)
  cout << s2[i]; // The [] operator

string s3 = s1 + "DEFG"; // The + operator
cout << s3 << endl; // s3 becomes ABCDEFG

s1 += "ABC";
cout << s1 << endl; // s1 becomes ABCABC

s1 = "ABC";
s2 = "ABE";
cout << (s1 == s2) << endl; // Displays 0 (means false)
cout << (s1 != s2) << endl; // Displays 1 (means true)
cout << (s1 > s2) << endl; // Displays 0 (means false)
cout << (s1 >= s2) << endl; // Displays 0 (means false)
cout << (s1 < s2) << endl; // Displays 1 (means true)
```



```
cout << (s1 >= s2) << endl; // Displays 0 (means false)
cout << (s1 < s2) << endl; // Displays 1 (means true)
cout << (s1 <= s2) << endl; // Displays 1 (means true)
```



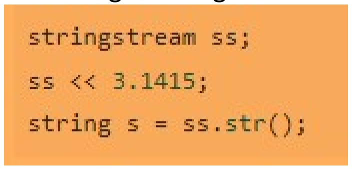
What is the output of the following code?

```
string s1("abc");
s1[0] = 'R';
string s2("abcD");
s1 += s2;
cout << s1 << endl;
```

- abcabcD
- abcDabc
- [✓ RbcabcD]
- abcDRbc

10.2.11 Converting Numbers to Strings Using stringstream

Friday, March 17, 2023 10:12 AM

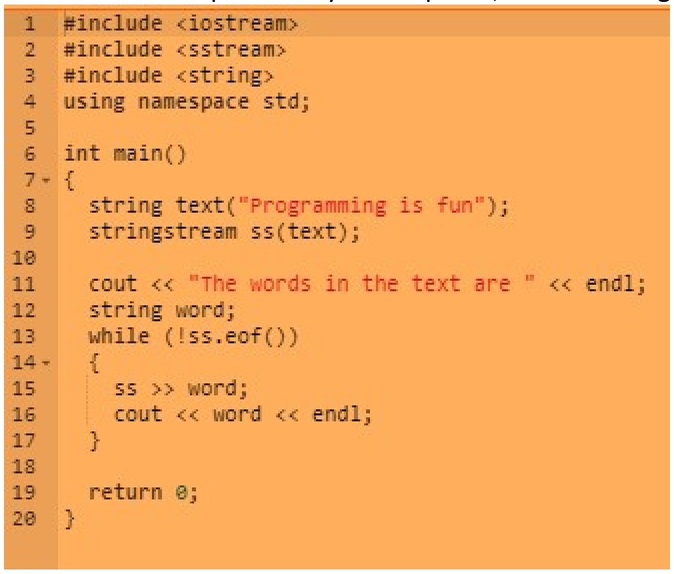
- Into to_string function b4, converting number to a string
- When converting floating-pt numb to string, converted string has 6 digits after decimal
- Trailing 0's appended to the string if there are not enough digits after the decimal point
- Avoid trailing 0s in string by function to convert, or can use stringstream calss in <<sstream>> header file
- The stringstream gives interface to manipulate strings as if they were I/O streams
- 

```
stringstream ss;
ss << 3.1415;
string s = ss.str();
```

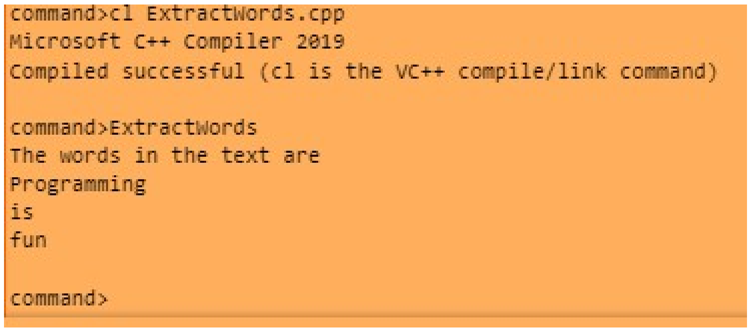
-

10.2.12 Splitting Strings

Friday, March 17, 2023 10:15 AM

- Usually need to extract words from string
- Assume words separated by whitespaces, can use stringstream again
- 

```cpp
#include <iostream>
#include <sstream>
#include <string>
using namespace std;

int main()
{
  string text("Programming is fun");
  stringstream ss(text);

  cout << "The words in the text are " << endl;
  string word;
  while (!ss.eof())
  {
    ss >> word;
    cout << word << endl;
  }

  return 0;
}
```

-
- 

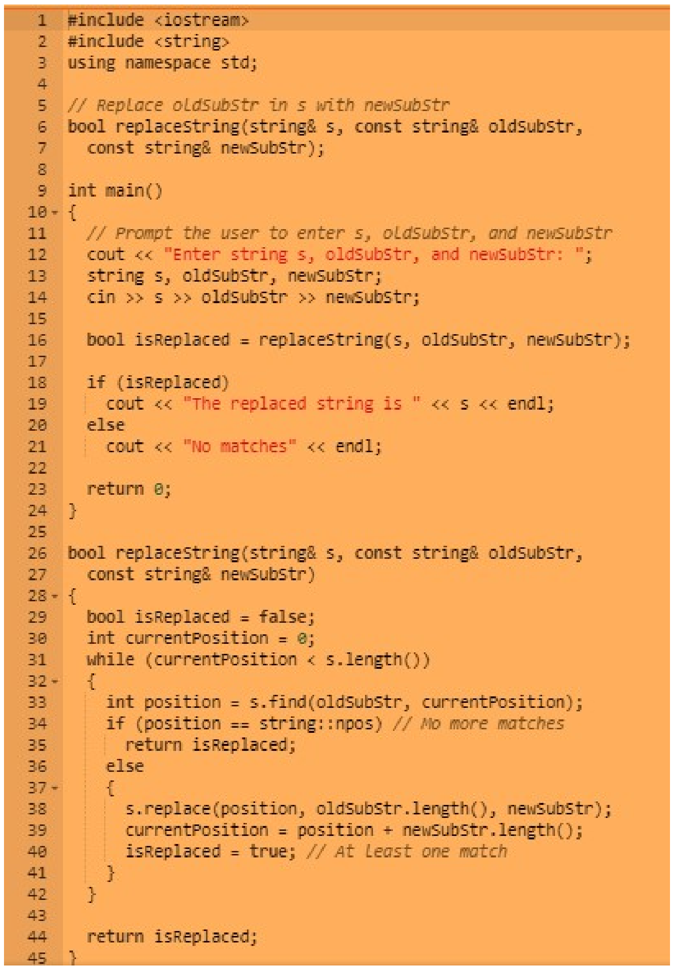
```
command>cl ExtractWords.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>ExtractWords
The words in the text are
Programming
is
fun

command>
```

- Pgrm makes a stringstream object for the text string, this object can be used like an input stream for reading data from console
- It sends data from the string stream to a string object word
- The eof() function in stringstream class returns true when all items in stream are read

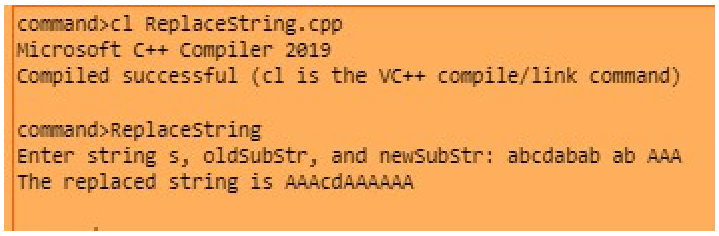
10.2.13 Case Study: Replacing Strings

Friday, March 17, 2023 10:18 AM

- Write following function that replaces the occurrence of a substring oldSubStr w/ a new substring newSubStr in string s

-
  ```
  bool replaceString(string& s, const string& oldSubStr,
    const string& newSubStr)
  ```
- Fn returns true if string s is changed, false if not
-
- 

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

// Replace oldSubStr in s with newSubStr
bool replaceString(string& s, const string& oldSubStr,
  const string& newSubStr);

int main()
{
  // Prompt the user to enter s, oldSubStr, and newSubStr
  cout << "Enter string s, oldSubStr, and newSubStr: ";
  string s, oldSubStr, newSubStr;
  cin >> s >> oldSubStr >> newSubStr;

  bool isReplaced = replaceString(s, oldSubStr, newSubStr);

  if (isReplaced)
    cout << "The replaced string is " << s << endl;
  else
    cout << "No matches" << endl;

  return 0;
}

bool replaceString(string& s, const string& oldSubStr,
  const string& newSubStr)
{
  bool isReplaced = false;
  int currentPosition = 0;
  while (currentPosition < s.length())
  {
    int position = s.find(oldSubStr, currentPosition);
    if (position == string::npos) // No more matches
      return isReplaced;
    else
    {
      s.replace(position, oldSubStr.length(), newSubStr);
      currentPosition = position + newSubStr.length();
      isReplaced = true; // At least one match
    }
  }

  return isReplaced;
}
```

-
- 

```
command>cl ReplaceString.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>ReplaceString
Enter string s, oldSubStr, and newSubStr: abcdabab ab AAA
The replaced string is AAAcdAAAAAA
```

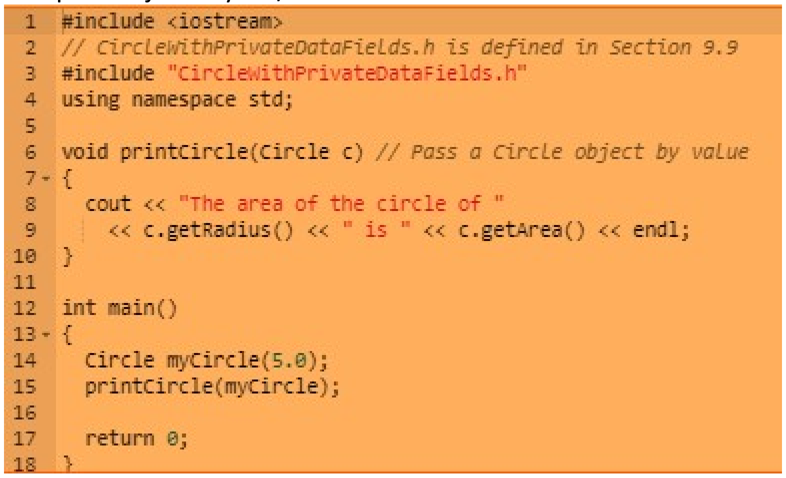
- Pgrm prompts user to enter string, an old substring, and a new substring
- Pgrm invokes replaceString function, so it replaces all occurrences of the old substring w/ the new substring, displays message whether string has been replaced
- ReplaceString function searches for oldSubStr in string s starting from currentPosition starting from 0
- The find function in the string class is used to find a substring in a string
- Returns string::npos if its not found, in which the search ends and the function returns isReplaced

(bool var, initially set to false, true when substring is found)

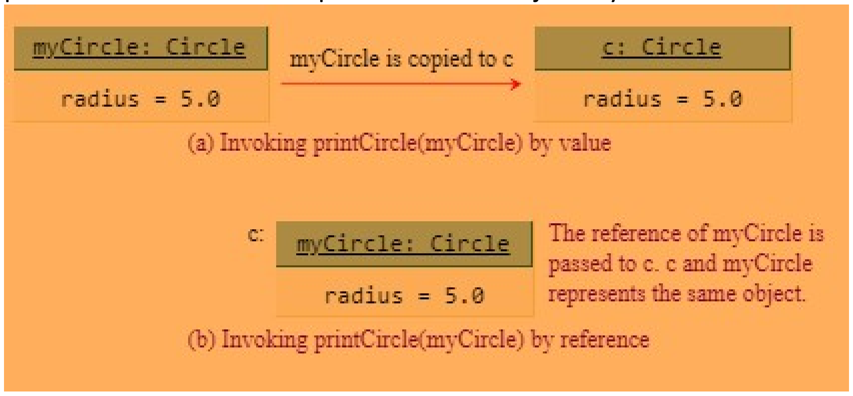
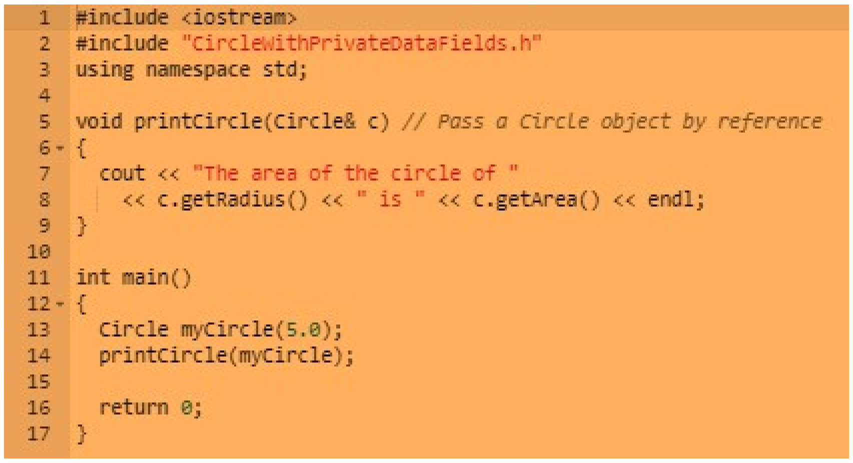
- Function repeatedly finds a substring and replaces it w/ a new substring using the replace function and resets the current search pos to look for a new match in the rest of the string
-

10.3 Passing Objects to Functions

Friday, March 17, 2023 10:49 AM

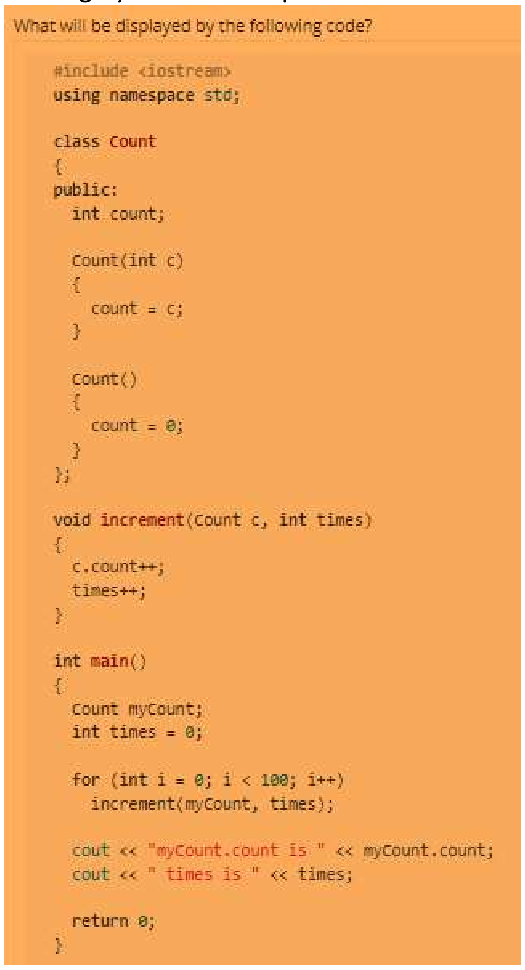
- Objects can be passed to a function by value/reference, but more efficient to pass objects by reference
- Already know how to pass args of primitive types, array types, and string types to functions
- Can pass any types of objects to functions
- Can pass objects by val/reference
- 

```cpp
#include <iostream>
// CircleWithPrivateDataFields.h is defined in Section 9.9
#include "CircleWithPrivateDataFields.h"
using namespace std;

void printCircle(Circle c) // Pass a Circle object by value
{
  cout << "The area of the circle of "
    << c.getRadius() << " is " << c.getArea() << endl;
}

int main()
{
  Circle myCircle(5.0);
  printCircle(myCircle);

  return 0;
}
```

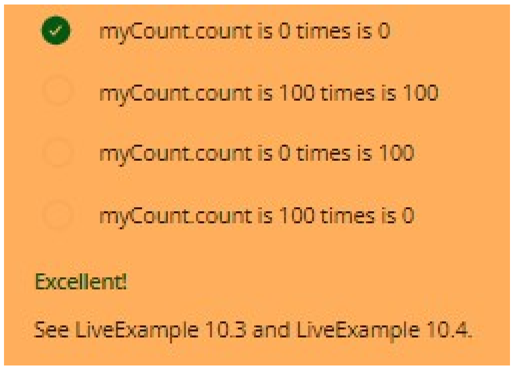
- This is pass by val
- The Circle class defined in past listing, …
- To pass object arg by val is to copy the object to the function parameter, so object c in the printCircle function is independent of the object myCircle in main function
- 
myCircle: Circle
radius = 5.0
myCircle is copied to c
c: Circle
radius = 5.0
(a) Invoking printCircle(myCircle) by value
c:
myCircle: Circle
radius = 5.0
The reference of myCircle is passed to c. c and myCircle represents the same object.
(b) Invoking printCircle(myCircle) by reference
-
- 

```cpp
#include <iostream>
#include "CircleWithPrivateDataFields.h"
using namespace std;

void printCircle(Circle& c) // Pass a Circle object by reference
{
  cout << "The area of the circle of "
    << c.getRadius() << " is " << c.getArea() << endl;
}

int main()
{
  Circle myCircle(5.0);
  printCircle(myCircle);

  return 0;
}
```

- This is pass by reference

- Reference parameter of the Circle type is declared in the printCircle function
- Main fn makes a circle objects myCircle and passes the reference of the object to the printCircle function
- So object c in the the printCircle function basically just alias of the object myCircle in the main fn
- Passing by reference is prefered bc takes time and additional mem space to pass by val
- 

What will be displayed by the following code?

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

class Count
{
public:
  int count;

  Count(int c)
  {
    count = c;
  }

  Count()
  {
    count = 0;
  }
};

void increment(Count c, int times)
{
  c.count++;
  times++;
}

int main()
{
  Count myCount;
  int times = 0;

  for (int i = 0; i < 100; i++)
    increment(myCount, times);

  cout << "myCount.count is " << myCount.count;
  cout << " times is " << times;

  return 0;
}
```
-
- 

[✓ myCount.count is 0 times is 0]

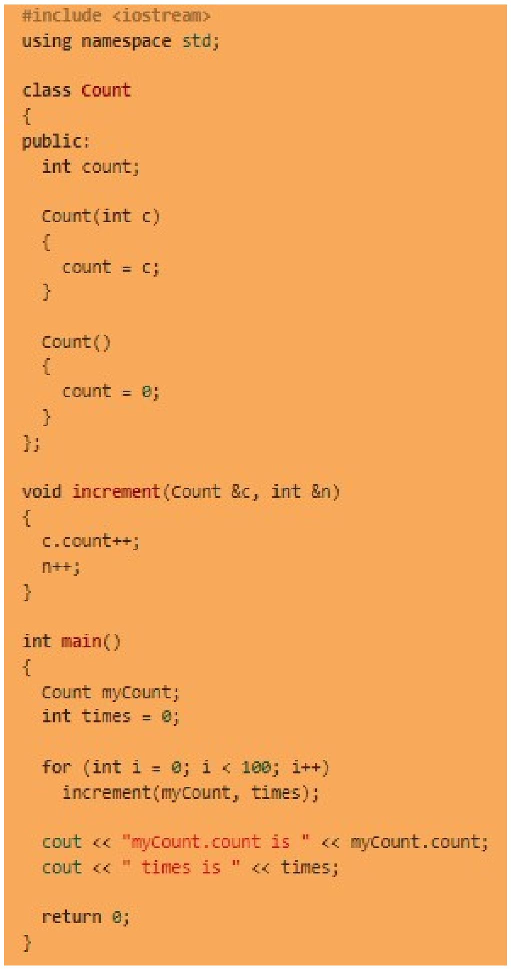
myCount.count is 100 times is 100

myCount.count is 0 times is 100

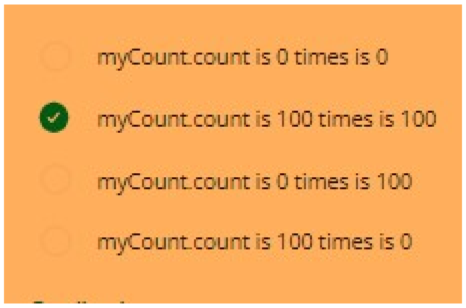
myCount.count is 100 times is 0

Excellent!

See LiveExample 10.3 and LiveExample 10.4.
- [C++ Programming Tutorial 67 - Pass By Reference and Pass By Value]
-

- 

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

class Count
{
public:
  int count;

  Count(int c)
  {
    count = c;
  }

  Count()
  {
    count = 0;
  }
};

void increment(Count &c, int &n)
{
  c.count++;
  n++;
}

int main()
{
  Count myCount;
  int times = 0;

  for (int i = 0; i < 100; i++)
    increment(myCount, times);

  cout << "myCount.count is " << myCount.count;
  cout << " times is " << times;

  return 0;
}
```

- 

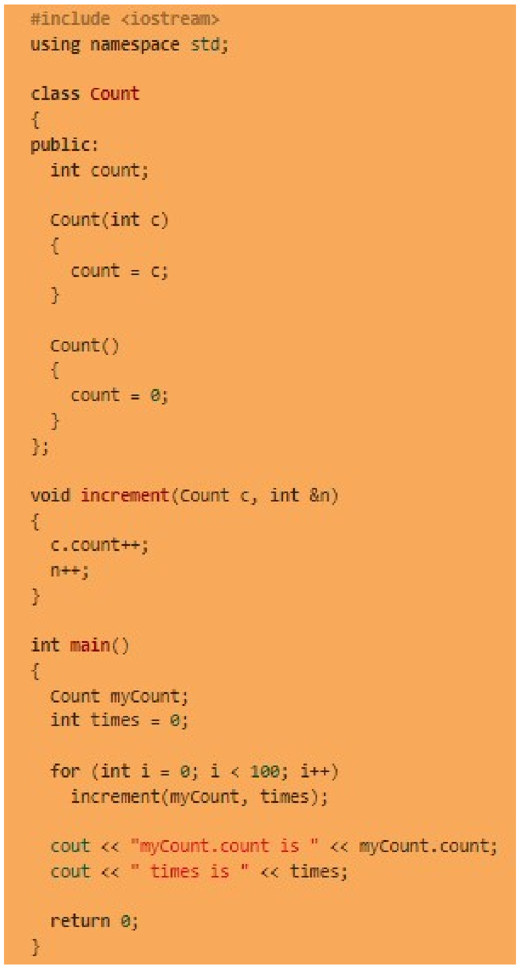
myCount.count is 0 times is 0

[✓ myCount.count is 100 times is 100]

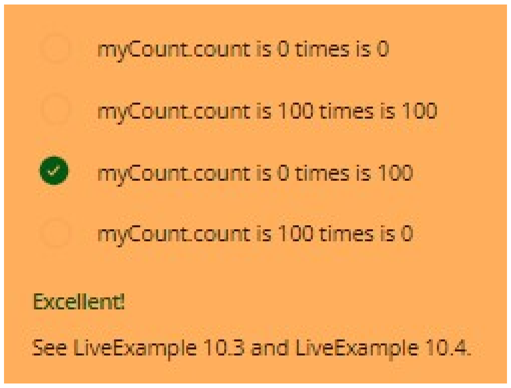
myCount.count is 0 times is 100

myCount.count is 100 times is 0

-



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

class Count
{
public:
  int count;

  Count(int c)
  {
    count = c;
  }

  Count()
  {
    count = 0;
  }
};

void increment(Count c, int &n)
{
  c.count++;
  n++;
}

int main()
{
  Count myCount;
  int times = 0;

  for (int i = 0; i < 100; i++)
    increment(myCount, times);

  cout << "myCount.count is " << myCount.count;
  cout << " times is " << times;

  return 0;
}
```



- myCount.count is 0 times is 0
- myCount.count is 100 times is 100
- [✓ myCount.count is 0 times is 100]
- myCount.count is 100 times is 0

Excellent!

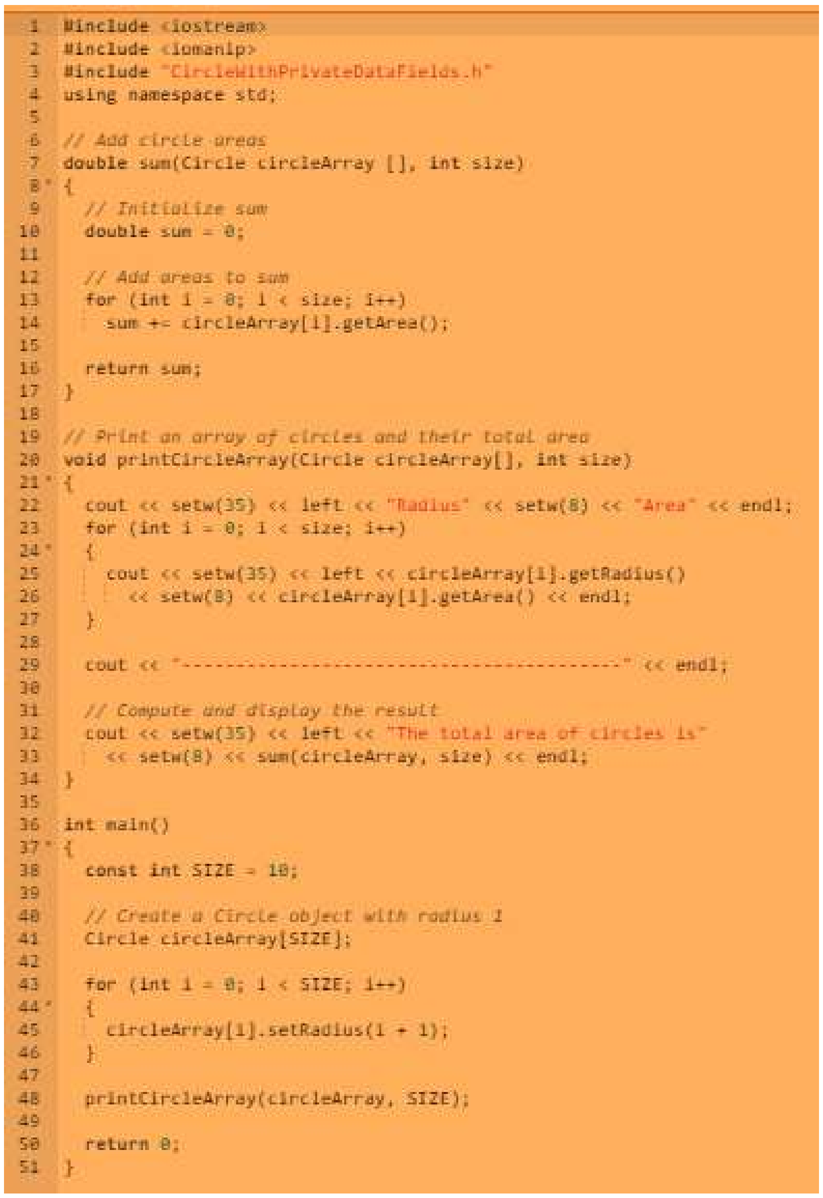
See LiveExample 10.3 and LiveExample 10.4.

10.4 Array of Objects

Friday, March 17, 2023 11:22 AM

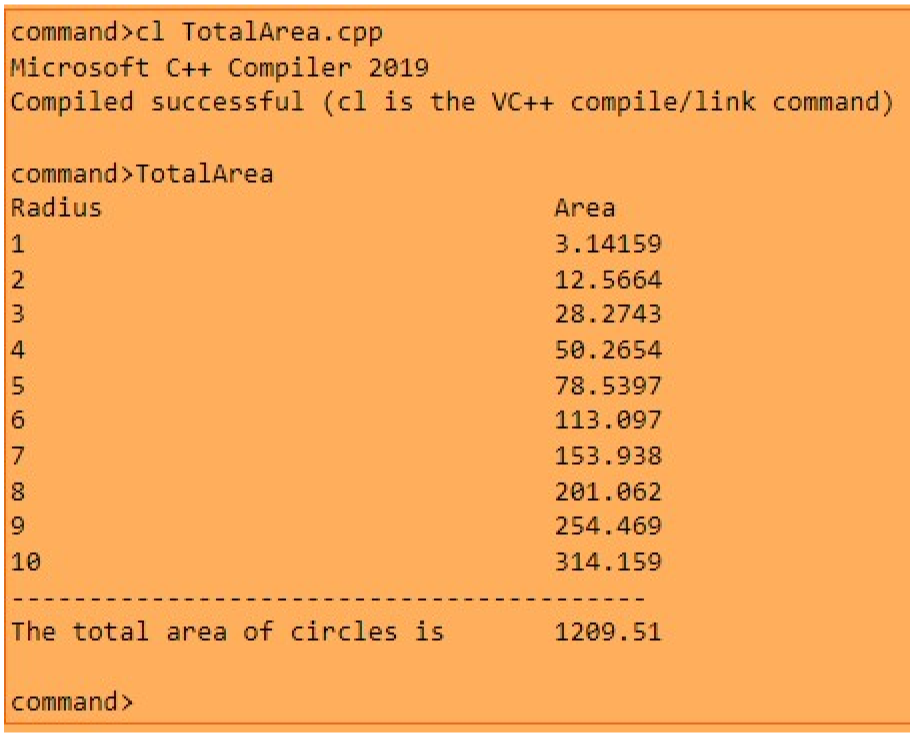
- Can make an array of objects just like an array of primitive vals/stings
- Array of primitive type elements and strings were made, can make arrays of any objects

-
  ```
  Circle circleArray[10]; // Declare an array of ten Circle objects
  ```
- Name of the array is circleArray, and no-arg constructor is called to initialize each element in the aray
- So, circleArray[0].getRadius() returns 1 bc no-arg constructor assigns 1 to radius
- Can also use array initializer to declare and initialize an array using a constructor w/ args, like

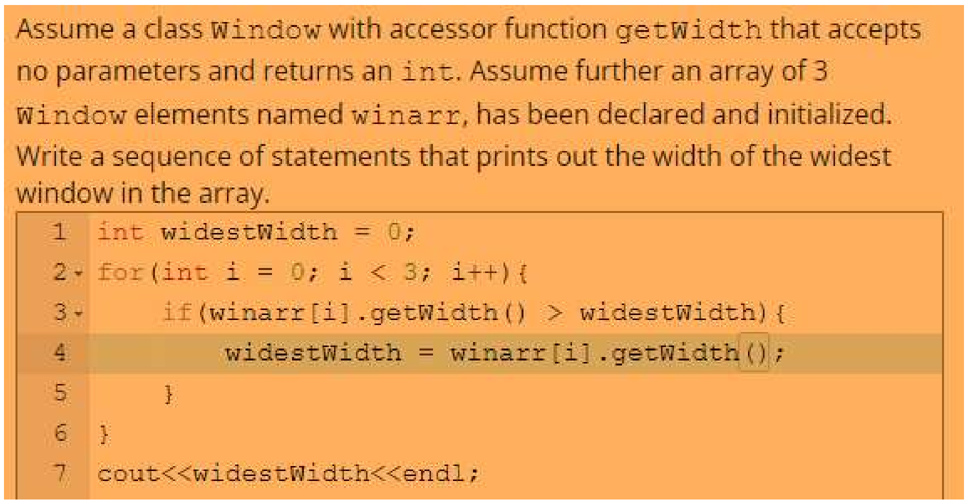
-
  ```
  Circle circleArray[3] = {Circle(3), Circle(4), Circle(5)};
  ```
-
- 

```
#include <iostream>
#include <iomanip>
#include "CircleWithPrivateDataFields.h"
using namespace std;

// Add circle areas
double sum(Circle circleArray [], int size)
{
  // Initialize sum
  double sum = 0;

  // Add areas to sum
  for (int i = 0; i < size; i++)
    sum += circleArray[i].getArea();

  return sum;
}

// Print an array of circles and their total area
void printCircleArray(Circle circleArray[], int size)
{
  cout << setw(35) << left << "Radius" << setw(8) << "Area" << endl;
  for (int i = 0; i < size; i++)
  {
    cout << setw(35) << left << circleArray[i].getRadius()
      << setw(8) << circleArray[i].getArea() << endl;
  }

  cout << "-----------------------------------------" << endl;

  // Compute and display the result
  cout << setw(35) << left << "The total area of circles is"
    << setw(8) << sum(circleArray, size) << endl;
}

int main()
{
  const int SIZE = 10;

  // Create a Circle object with radius 1
  Circle circleArray[SIZE];

  for (int i = 0; i < SIZE; i++)
  {
    circleArray[i].setRadius(i + 1);
  }

  printCircleArray(circleArray, SIZE);

  return 0;
}
```

-

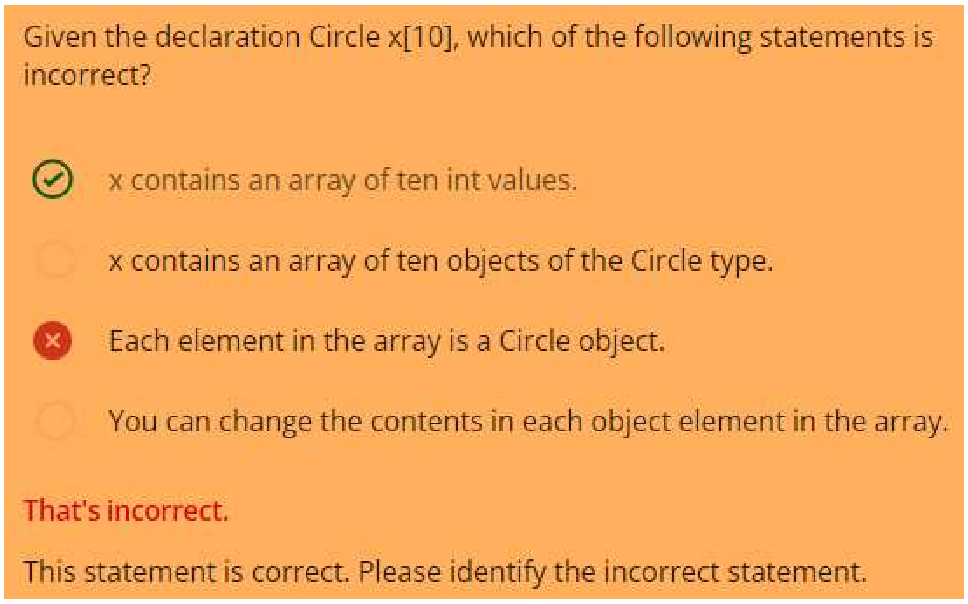
- 

```
command>cl TotalArea.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>TotalArea
Radius                             Area
1                                  3.14159
2                                  12.5664
3                                  28.2743
4                                  50.2654
5                                  78.5397
6                                  113.097
7                                  153.938
8                                  201.062
9                                  254.469
10                                 314.159
-----------------------------------------
The total area of circles is       1209.51

command>
```

-
- 

Assume a class `Window` with accessor function `getWidth` that accepts no parameters and returns an `int`. Assume further an array of 3 `Window` elements named `winarr`, has been declared and initialized. Write a sequence of statements that prints out the width of the widest window in the array.

```
int widestWidth = 0;
for(int i = 0; i < 3; i++){
    if(winarr[i].getWidth() > widestWidth){
        widestWidth = winarr[i].getWidth();
    }
}
cout<<widestWidth<<endl;
```

-
- 

Given the declaration Circle x[10], which of the following statements is incorrect?

✓ x contains an array of ten int values.

x contains an array of ten objects of the Circle type.

✗ Each element in the array is a Circle object.

You can change the contents in each object element in the array.

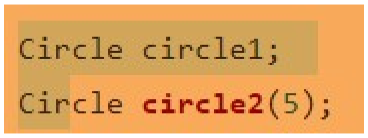
That's incorrect.

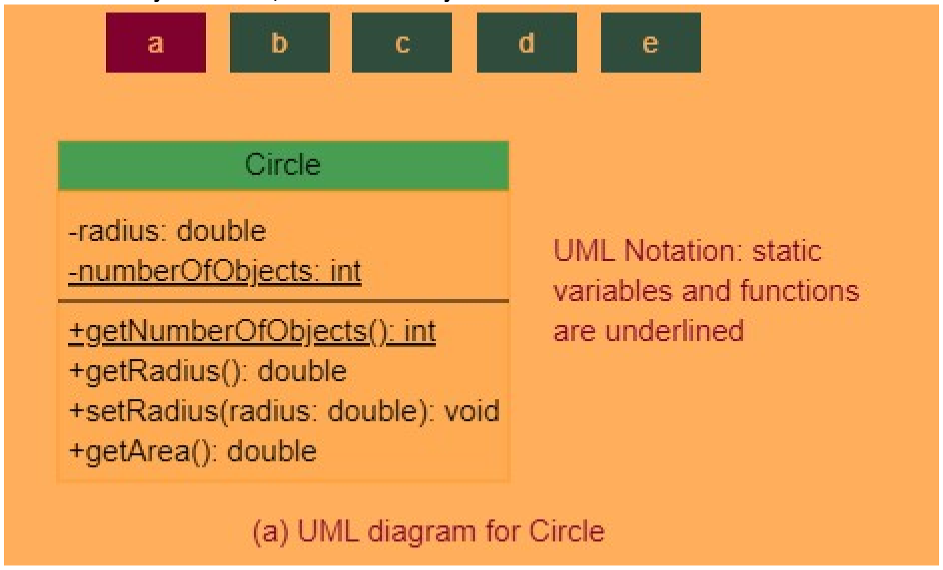
This statement is correct. Please identify the incorrect statement.

- Idk why, I thought it was b initially, then went d then c

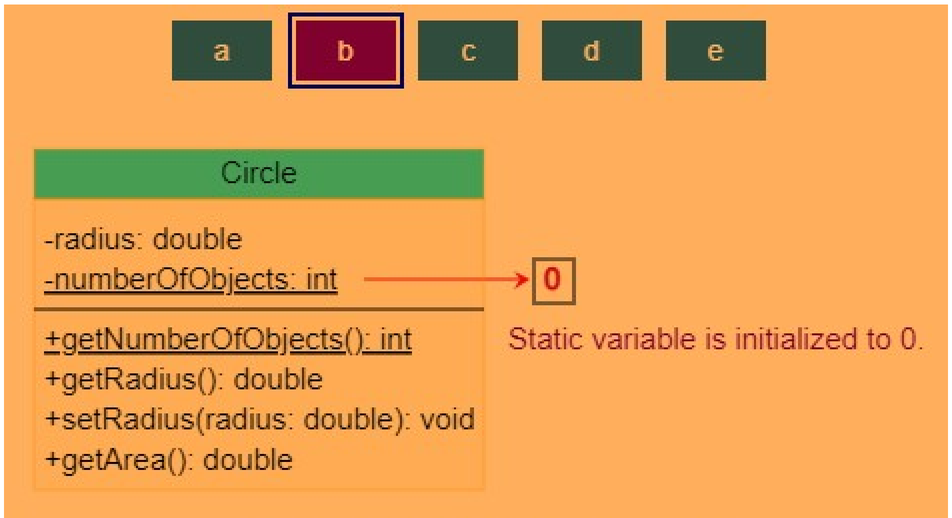
10.5 Instance and Static Members

Friday, March 17, 2023 12:00 PM

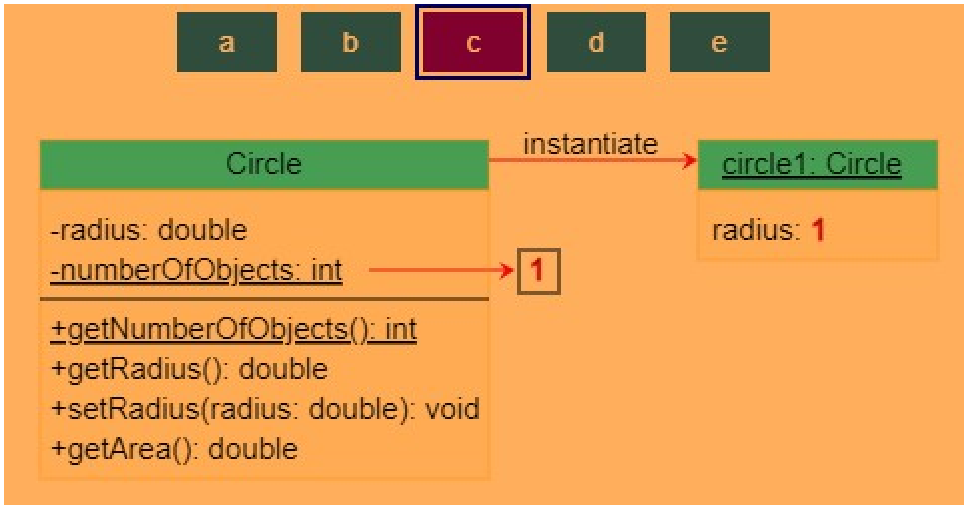
- Static var is shared by all objects of the class, a static function cannot access instance members of the class
- The data fields used in the classes so far aka instance data fields aka instance vars, they tied to a specific instance of the class, not shared among objects of the same class
- EX:
    - 

      ```
      Circle circle1;
      Circle circle2(5);
      ```
    - Radius in circle1 is independent of the radius in circle2 and is stored in a diff mem location, changes made to circle1's radius don't affect circle2's radius, and opp as well
- If want all instances of a class to share data, use static vars aka class vars
- They store vals for the vars in common mem location
- Also, all objects of the same class are affected if one object changes the val of a static var
- C++ supports static functions as well as static vars
- Static fns can be called w/out making an instance of the class (instance fns can only be called from a specific instance)
- Mod the Circle class by adding a static var numberOfObjects to count the number of circle objects created
    - When 2nd object made, numberOfObjects becomes 2
    - 
      a
      b
      c
      d
      e
      Circle
      -radius: double
      -numberOfObjects: int
      +getNumberOfObjects(): int
      +getRadius(): double
      +setRadius(radius: double): void
      +getArea(): double
      UML Notation: static variables and functions are underlined

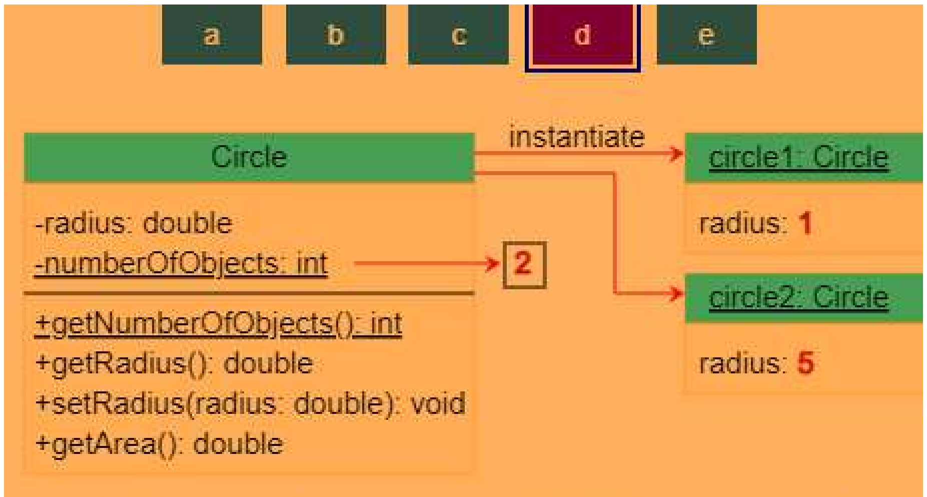
      (a) UML diagram for Circle

- 
a
b
c
d
e
Circle
-radius: double
-numberOfObjects: int
0
+getNumberOfObjects(): int
+getRadius(): double
+setRadius(radius: double): void
+getArea(): double
Static variable is initialized to 0.

(b) Before any Circle object is created.

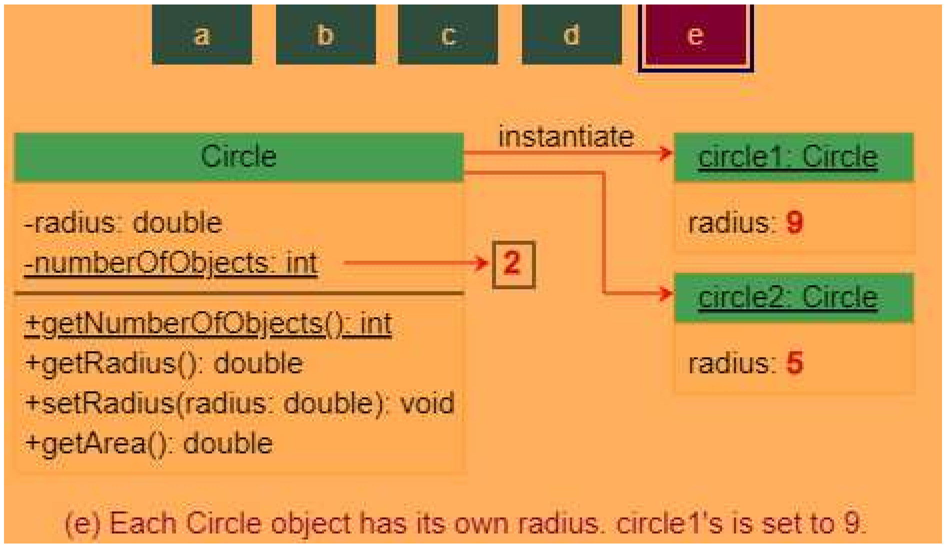
-
- 

(c) Object circle1 is created. numberOfObjects becomes 1.

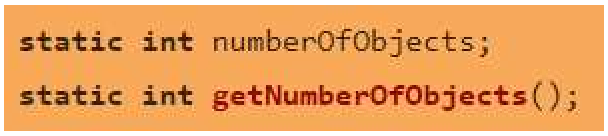
-
- 

(d) Object circle2 is created. numberOfObjects becomes 2.

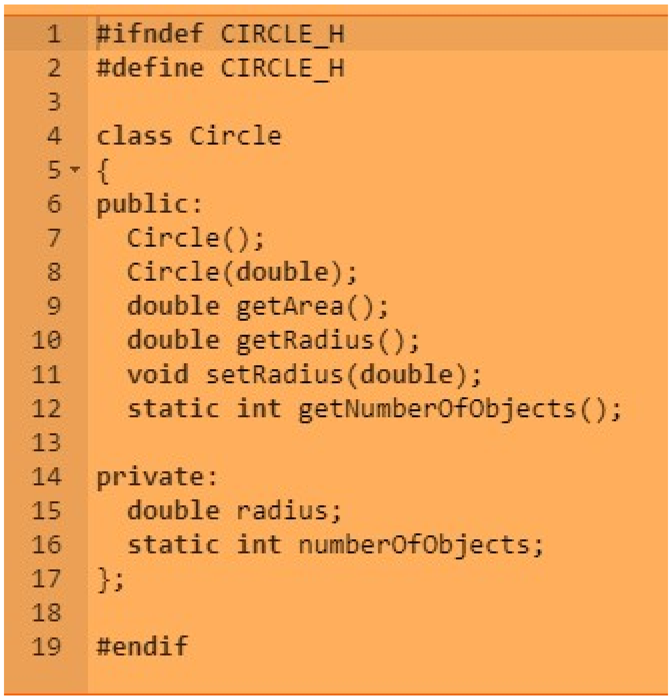
-

- 

(e) Each Circle object has its own radius. circle1's is set to 9.

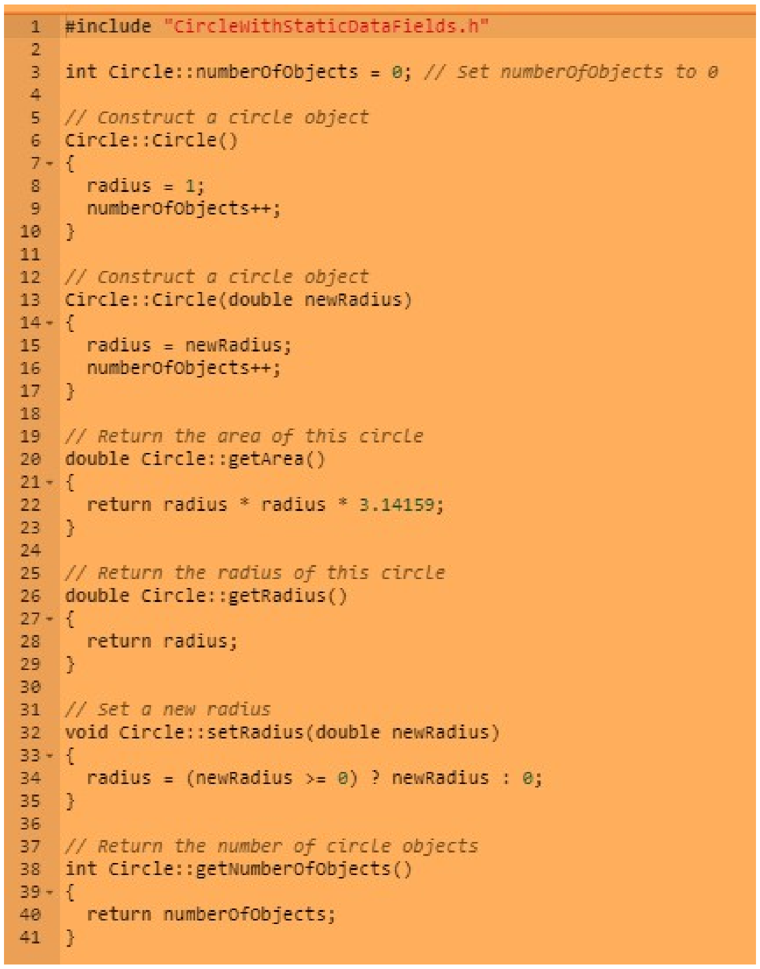
- To declare static var/fn, put mod static in the var/fn dectlared
  -
  - 

```
static int numberOfObjects;
static int getNumberOfObjects();
```

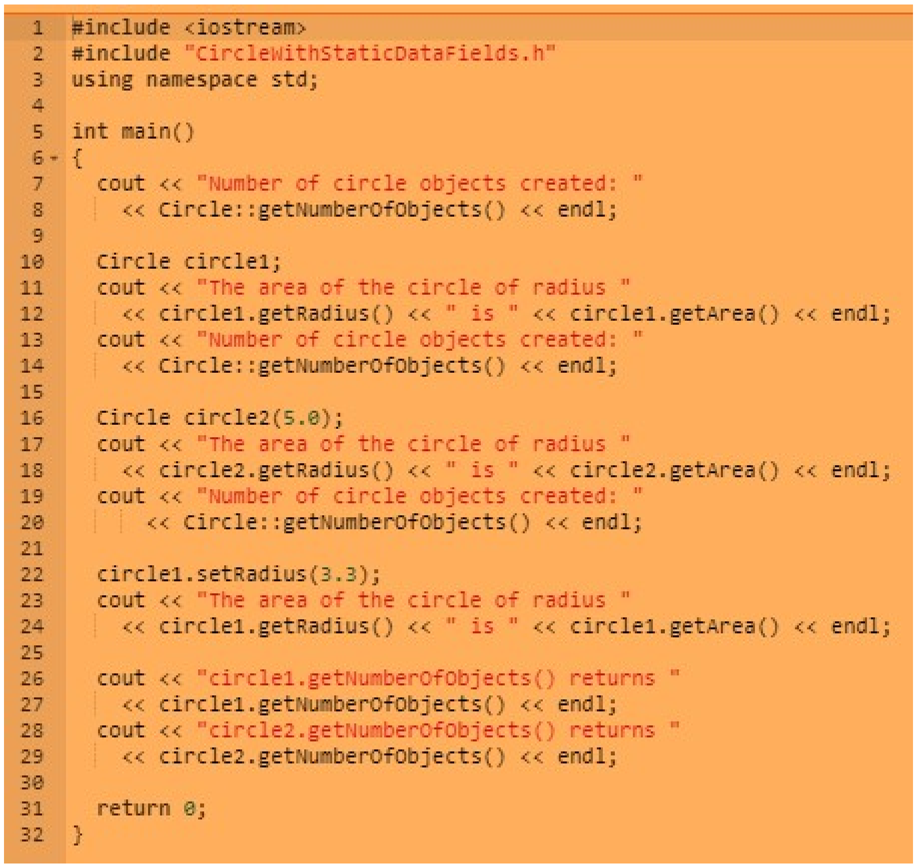
  -
  - 

```
#ifndef CIRCLE_H
#define CIRCLE_H

class Circle
{
public:
  Circle();
  Circle(double);
  double getArea();
  double getRadius();
  void setRadius(double);
  static int getNumberOfObjects();

private:
  double radius;
  static int numberOfObjects;
};

#endif
```

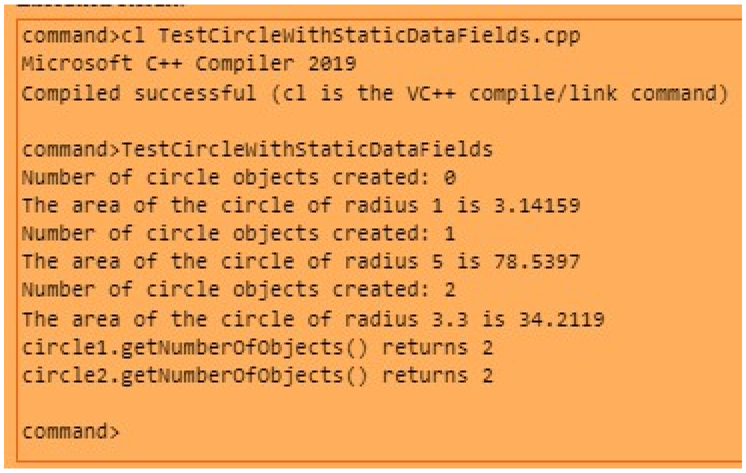
  - Static fn getNumberOfObjects & static var numberOfObjects declared as private data field in the class



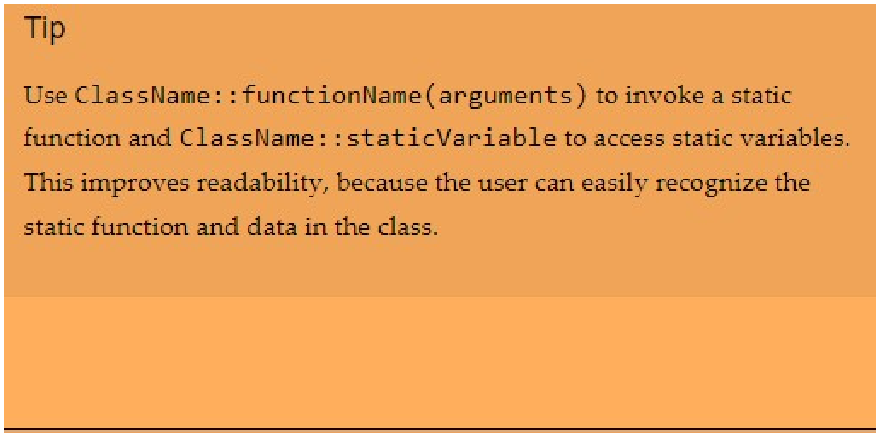
```cpp
#include "CircleWithStaticDataFields.h"

int Circle::numberOfObjects = 0; // Set numberOfObjects to 0

// Construct a circle object
Circle::Circle()
{
  radius = 1;
  numberOfObjects++;
}

// Construct a circle object
Circle::Circle(double newRadius)
{
  radius = newRadius;
  numberOfObjects++;
}

// Return the area of this circle
double Circle::getArea()
{
  return radius * radius * 3.14159;
}

// Return the radius of this circle
double Circle::getRadius()
{
  return radius;
}

// Set a new radius
void Circle::setRadius(double newRadius)
{
  radius = (newRadius >= 0) ? newRadius : 0;
}

// Return the number of circle objects
int Circle::getNumberOfObjects()
{
  return numberOfObjects;
}
```



```cpp
#include <iostream>
#include "CircleWithStaticDataFields.h"
using namespace std;

int main()
{
  cout << "Number of circle objects created: "
    << Circle::getNumberOfObjects() << endl;

  Circle circle1;
  cout << "The area of the circle of radius "
    << circle1.getRadius() << " is " << circle1.getArea() << endl;
  cout << "Number of circle objects created: "
    << Circle::getNumberOfObjects() << endl;

  Circle circle2(5.0);
  cout << "The area of the circle of radius "
    << circle2.getRadius() << " is " << circle2.getArea() << endl;
  cout << "Number of circle objects created: "
      << Circle::getNumberOfObjects() << endl;

  circle1.setRadius(3.3);
  cout << "The area of the circle of radius "
    << circle1.getRadius() << " is " << circle1.getArea() << endl;

  cout << "circle1.getNumberOfObjects() returns "
    << circle1.getNumberOfObjects() << endl;
  cout << "circle2.getNumberOfObjects() returns "
    << circle2.getNumberOfObjects() << endl;

  return 0;
}
```

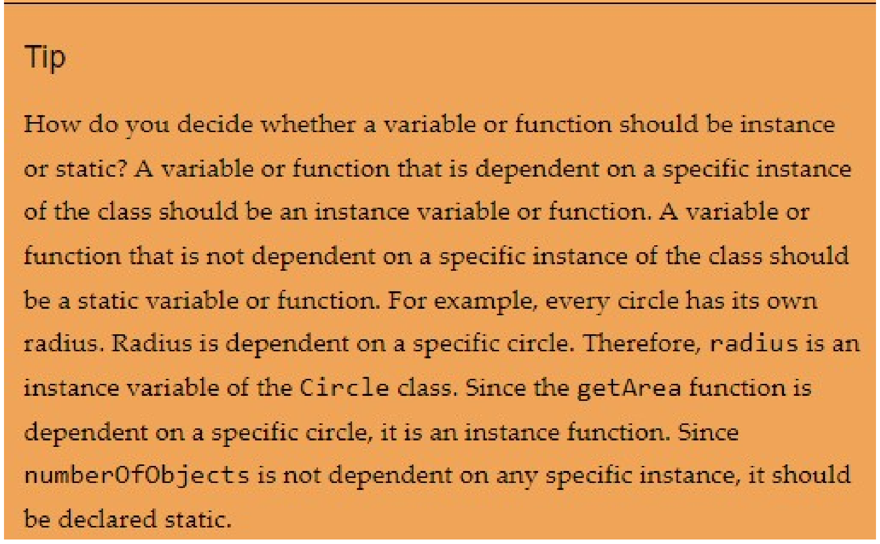
- 

```
command>cl TestCircleWithStaticDataFields.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>TestCircleWithStaticDataFields
Number of circle objects created: 0
The area of the circle of radius 1 is 3.14159
Number of circle objects created: 1
The area of the circle of radius 5 is 78.5397
Number of circle objects created: 2
The area of the circle of radius 3.3 is 34.2119
circle1.getNumberOfObjects() returns 2
circle2.getNumberOfObjects() returns 2

command>
```

- Can access static data fields and fn from instances of the class
- 

Tip

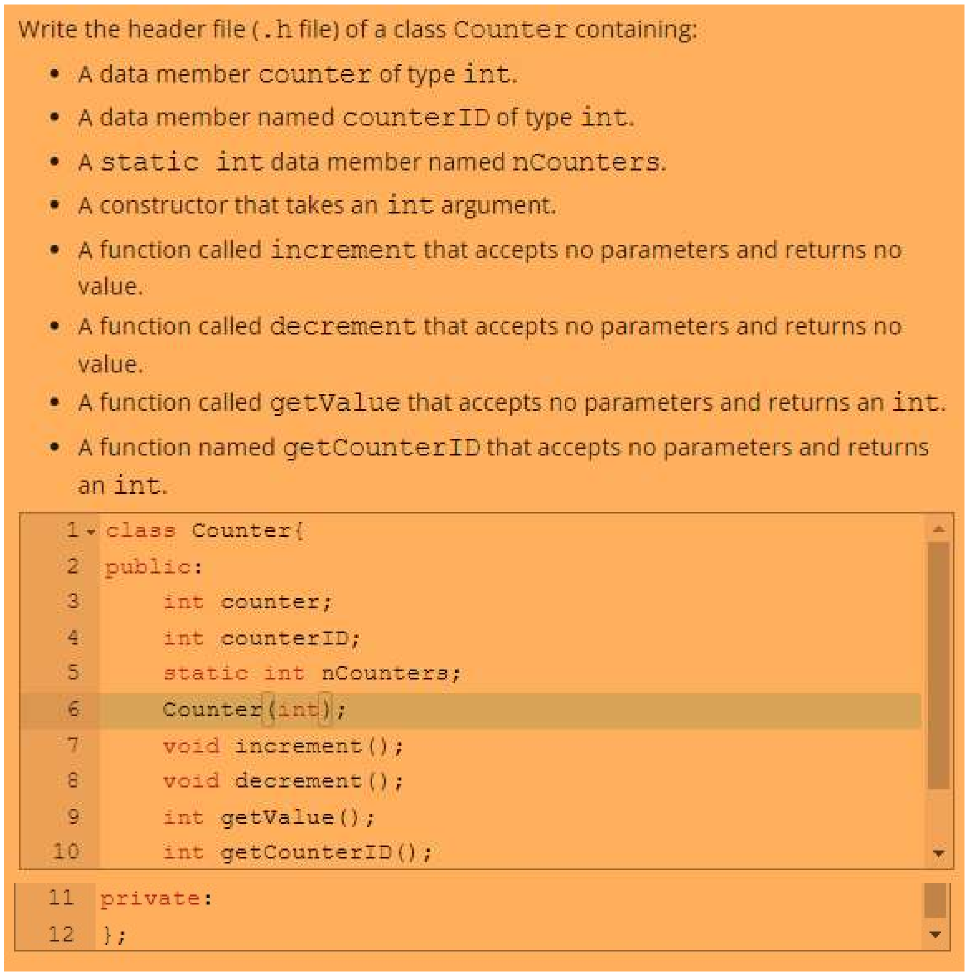
Use `ClassName::functionName(arguments)` to invoke a static function and `ClassName::staticVariable` to access static variables. This improves readability, because the user can easily recognize the static function and data in the class.

- 

Tip

How do you decide whether a variable or function should be instance or static? A variable or function that is dependent on a specific instance of the class should be an instance variable or function. A variable or function that is not dependent on a specific instance of the class should be a static variable or function. For example, every circle has its own radius. Radius is dependent on a specific circle. Therefore, `radius` is an instance variable of the `Circle` class. Since the `getArea` function is dependent on a specific circle, it is an instance function. Since `numberOfObjects` is not dependent on any specific instance, it should be declared static.

-

- 

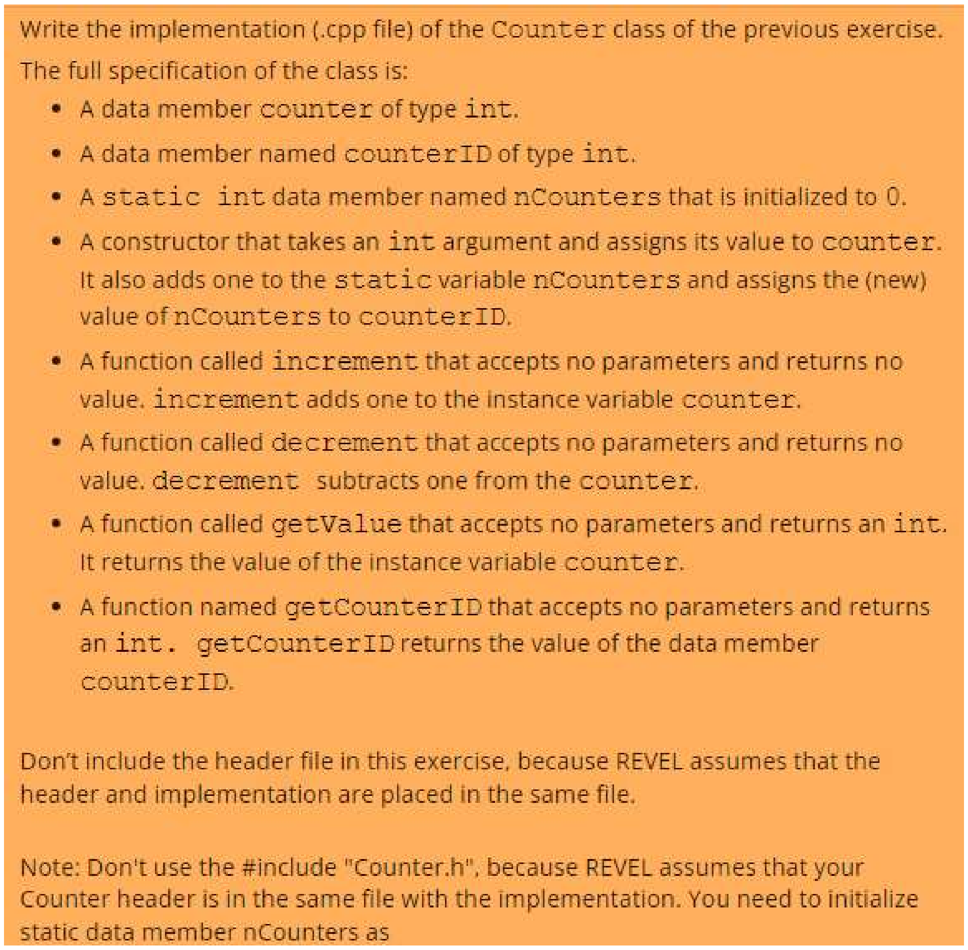
Write the header file (.h file) of a class Counter containing:

- A data member counter of type int.
- A data member named counterID of type int.
- A static int data member named nCounters.
- A constructor that takes an int argument.
- A function called increment that accepts no parameters and returns no value.
- A function called decrement that accepts no parameters and returns no value.
- A function called getValue that accepts no parameters and returns an int.
- A function named getCounterID that accepts no parameters and returns an int.

```
class Counter{
public:
    int counter;
    int counterID;
    static int nCounters;
    Counter(int);
    void increment();
    void decrement();
    int getValue();
    int getCounterID();
```

```
private:
};
```

-
-

- 

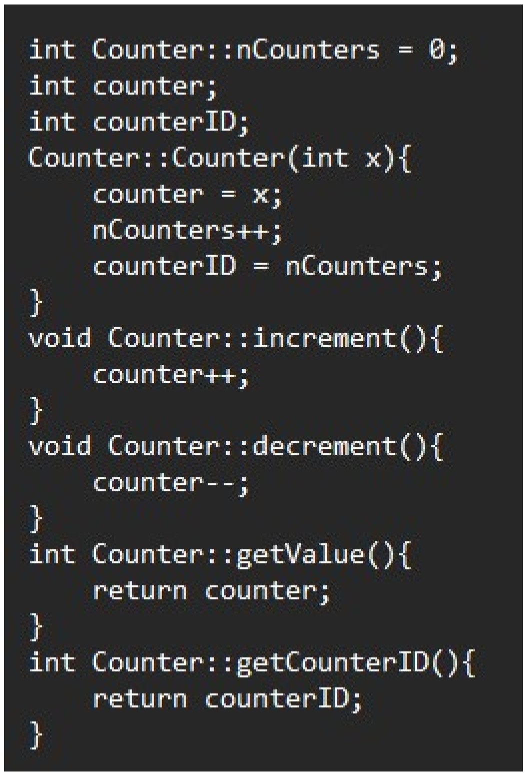
Write the implementation (.cpp file) of the Counter class of the previous exercise.

The full specification of the class is:

- A data member counter of type int.
- A data member named counterID of type int.
- A static int data member named nCounters that is initialized to 0.
- A constructor that takes an int argument and assigns its value to counter. It also adds one to the static variable nCounters and assigns the (new) value of nCounters to counterID.
- A function called increment that accepts no parameters and returns no value. increment adds one to the instance variable counter.
- A function called decrement that accepts no parameters and returns no value. decrement subtracts one from the counter.
- A function called getValue that accepts no parameters and returns an int. It returns the value of the instance variable counter.
- A function named getCounterID that accepts no parameters and returns an int. getCounterID returns the value of the data member counterID.

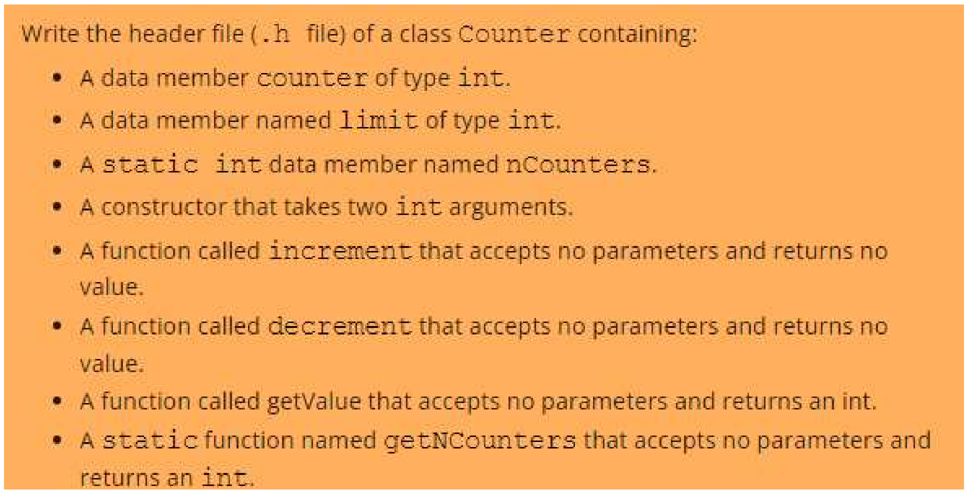
Don't include the header file in this exercise, because REVEL assumes that the header and implementation are placed in the same file.

Note: Don't use the #include "Counter.h", because REVEL assumes that your Counter header is in the same file with the implementation. You need to initialize static data member nCounters as

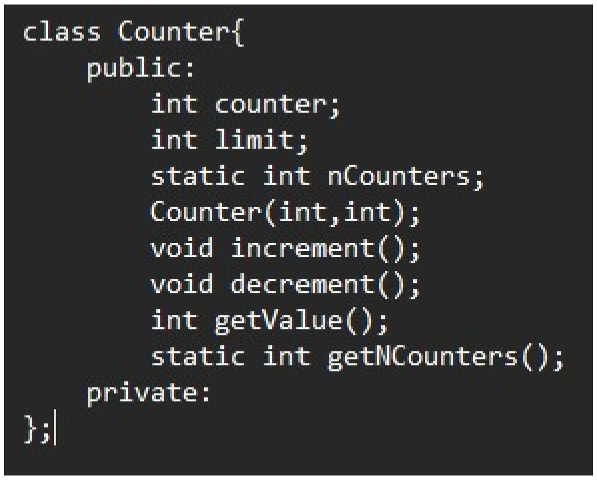
-
- 

```
int Counter::nCounters = 0;
int counter;
int counterID;
Counter::Counter(int x){
    counter = x;
    nCounters++;
    counterID = nCounters;
}
void Counter::increment(){
    counter++;
}
void Counter::decrement(){
    counter--;
}
int Counter::getValue(){
    return counter;
}
int Counter::getCounterID(){
    return counterID;
}
```

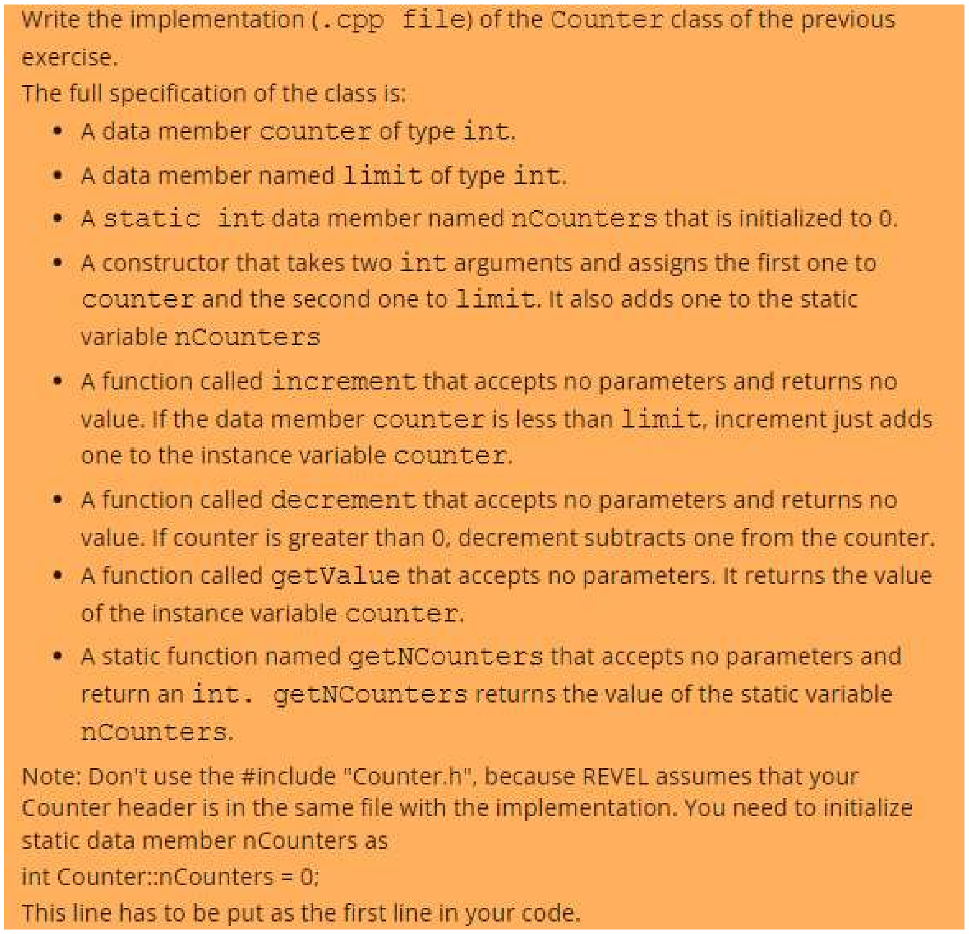
-

- 

  Write the header file (.h file) of a class Counter containing:

  - A data member counter of type int.
  - A data member named limit of type int.
  - A static int data member named nCounters.
  - A constructor that takes two int arguments.
  - A function called increment that accepts no parameters and returns no value.
  - A function called decrement that accepts no parameters and returns no value.
  - A function called getValue that accepts no parameters and returns an int.
  - A static function named getNCounters that accepts no parameters and returns an int.
-
- 

  ```
  class Counter{
      public:
          int counter;
          int limit;
          static int nCounters;
          Counter(int,int);
          void increment();
          void decrement();
          int getValue();
          static int getNCounters();
      private:
  };
  ```
-

- 

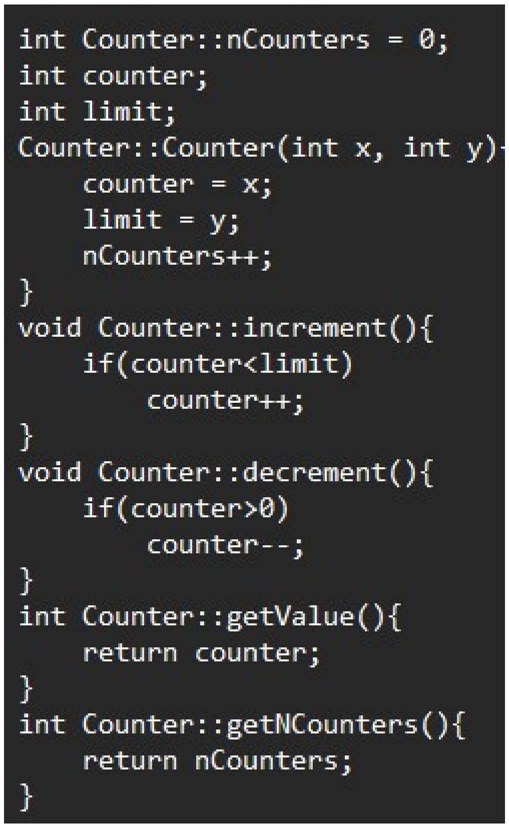
  Write the implementation (.cpp file) of the Counter class of the previous exercise.

  The full specification of the class is:

  - A data member counter of type int.
  - A data member named limit of type int.
  - A static int data member named nCounters that is initialized to 0.
  - A constructor that takes two int arguments and assigns the first one to counter and the second one to limit. It also adds one to the static variable nCounters
  - A function called increment that accepts no parameters and returns no value. If the data member counter is less than limit, increment just adds one to the instance variable counter.
  - A function called decrement that accepts no parameters and returns no value. If counter is greater than 0, decrement subtracts one from the counter.
  - A function called getValue that accepts no parameters. It returns the value of the instance variable counter.
  - A static function named getNCounters that accepts no parameters and return an int. getNCounters returns the value of the static variable nCounters.

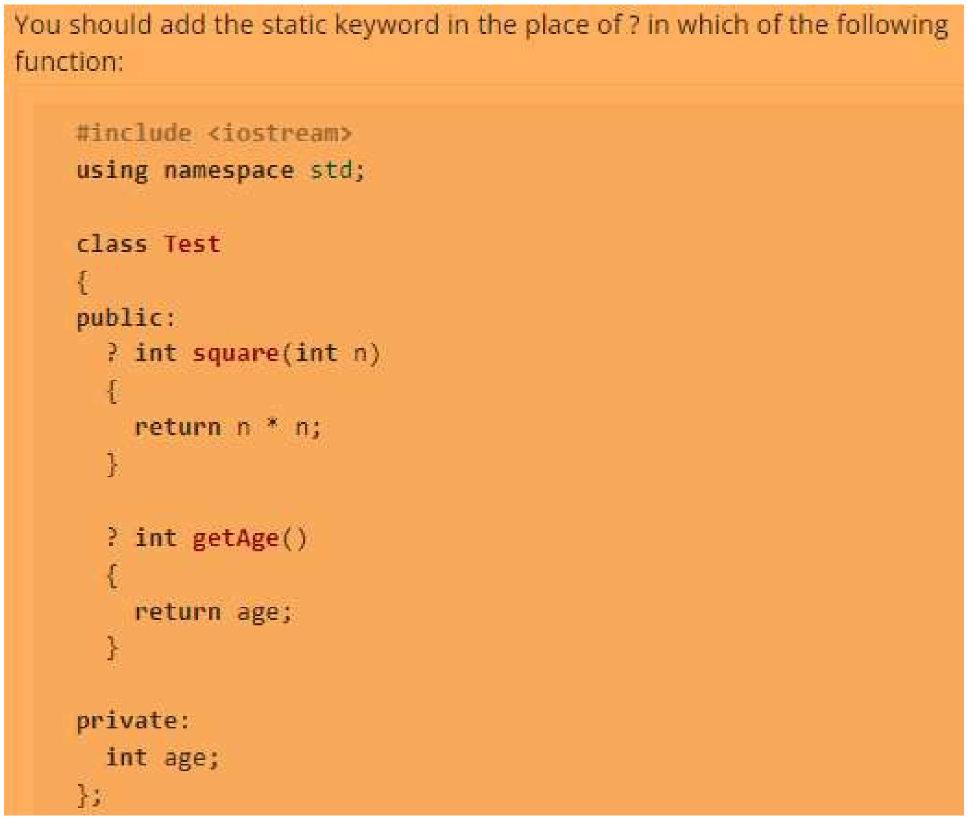
  Note: Don't use the #include "Counter.h", because REVEL assumes that your Counter header is in the same file with the implementation. You need to initialize static data member nCounters as

  int Counter::nCounters = 0;

  This line has to be put as the first line in your code.
-
- 

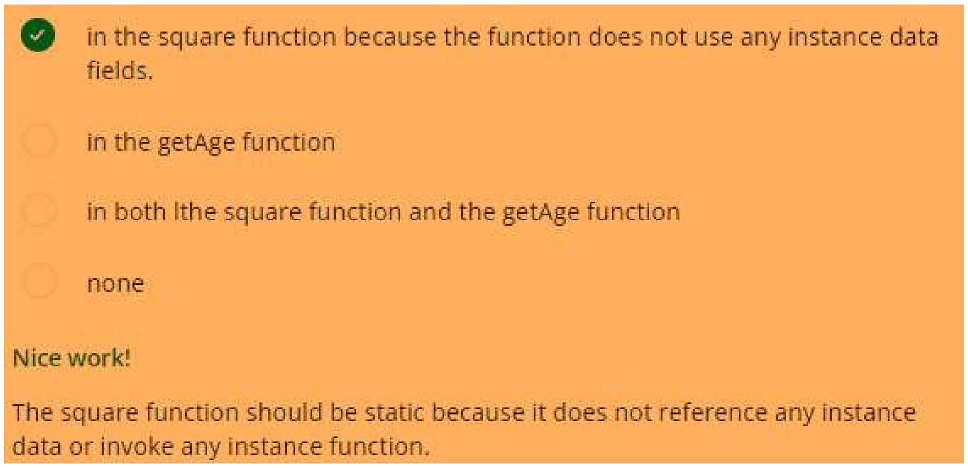
```
int Counter::nCounters = 0;
int counter;
int limit;
Counter::Counter(int x, int y)
    counter = x;
    limit = y;
    nCounters++;
}
void Counter::increment(){
    if(counter<limit)
        counter++;
}
void Counter::decrement(){
    if(counter>0)
        counter--;
}
int Counter::getValue(){
    return counter;
}
int Counter::getNCounters(){
    return nCounters;
}
```

-



You should add the static keyword in the place of ? in which of the following function:

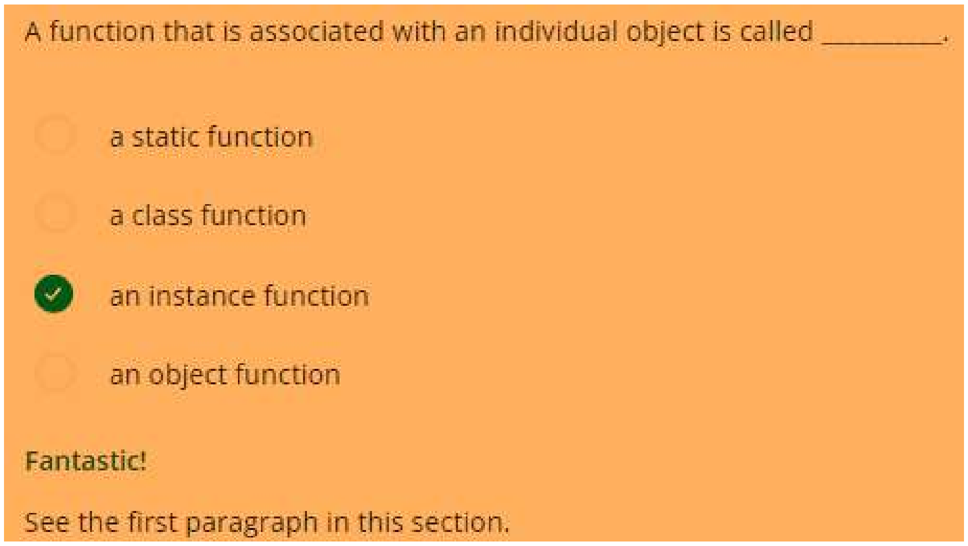
```
#include <iostream>
using namespace std;

class Test
{
public:
  ? int square(int n)
  {
    return n * n;
  }

  ? int getAge()
  {
    return age;
  }

private:
  int age;
};
```



- [✓ in the square function because the function does not use any instance data fields.]
- in the getAge function
- in both lthe square function and the getAge function
- none

Nice work!

The square function should be static because it does not reference any instance data or invoke any instance function.



A function that is associated with an individual object is called ________.

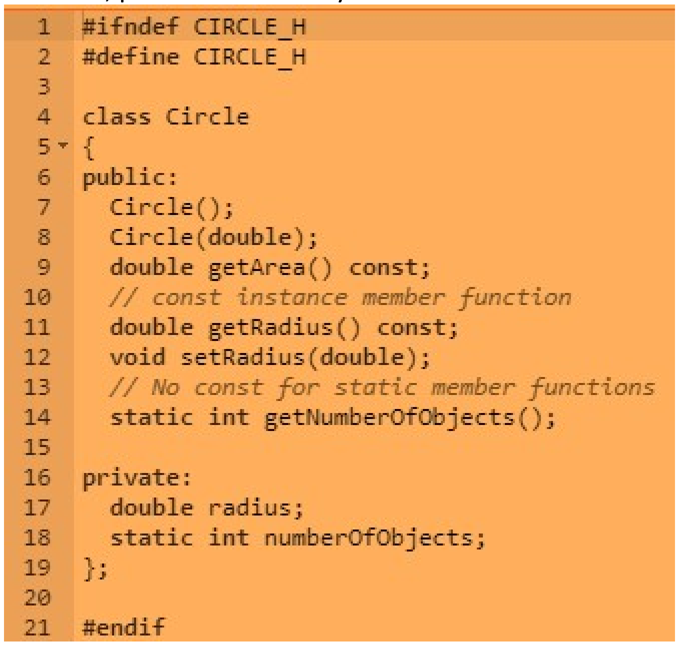
- a static function
- a class function
- [✓ an instance function]
- an object function

Fantastic!

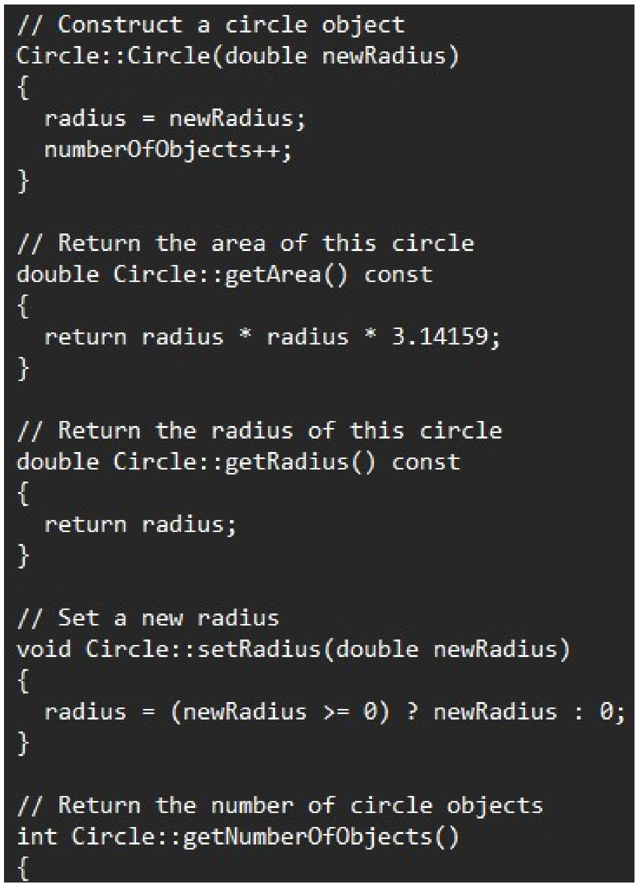
See the first paragraph in this section.

10.6 Constant Member Functions

Friday, March 17, 2023 4:55 PM

- C++ also lets u specify a constant member function to tell compiler that the function should not change the value of any data field in the object
- Can also use const keyword to specify a constant member function (aka constant function) to tell compiler that the function doesn't change the data fields in the object
- To do so, place the const keyword at end of the function header
- 

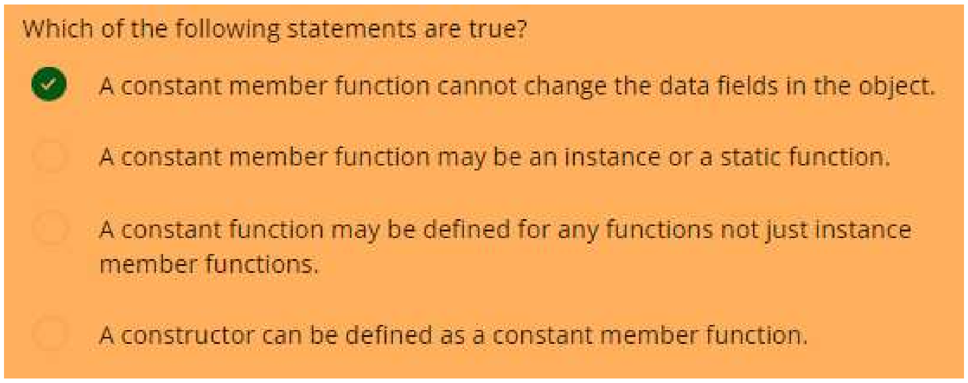
```cpp
#ifndef CIRCLE_H
#define CIRCLE_H

class Circle
{
public:
  Circle();
  Circle(double);
  double getArea() const;
  // const instance member function
  double getRadius() const;
  void setRadius(double);
  // No const for static member functions
  static int getNumberOfObjects();

private:
  double radius;
  static int numberOfObjects;
};

#endif
```

- 

```cpp
// Construct a circle object
Circle::Circle(double newRadius)
{
  radius = newRadius;
  numberOfObjects++;
}

// Return the area of this circle
double Circle::getArea() const
{
  return radius * radius * 3.14159;
}

// Return the radius of this circle
double Circle::getRadius() const
{
  return radius;
}

// Set a new radius
void Circle::setRadius(double newRadius)
{
  radius = (newRadius >= 0) ? newRadius : 0;
}

// Return the number of circle objects
int Circle::getNumberOfObjects()
{
```

```
// Return the number of circle objects
int Circle::getNumberOfObjects()
{
  return numberOfObjects;
}
```

- Only instance memb functions can be defined as constant functions, they for defensive pgrming
    - If fn mistakenly changes the val of data fields in a function, a compile error reported
    - U can define only instant fn constant, not static fn
    - An instance getter fn should always be defined as a constant memb fn, bc doesn't change the contents of the object
- If fn doesn't change object being passed, u should define the parameter constant using the const keyword:
-
```
void printCircle(const Circle& c)
{
  cout << "The area of the circle of "
    << c.getRadius() << " is " << c.getArea() << endl;
}
```
- Note that this code won't compile if the getRadius() or getArea() fn not defined const
- If use the Circle class defined b4, later fn won't compile bc getRadius() and getArea() not defined const
- But if u use the Circle class defined, later fn will compile bc getRadius() and getArea(0 are defined const
- Tip

  You can use the `const` modifier to specify a constant reference parameter or a constant member function. You should use the `const` modifier *consistently* whenever appropriate.
-
- 

  Which of the following statements are true?

  [✓ A constant member function cannot change the data fields in the object.]

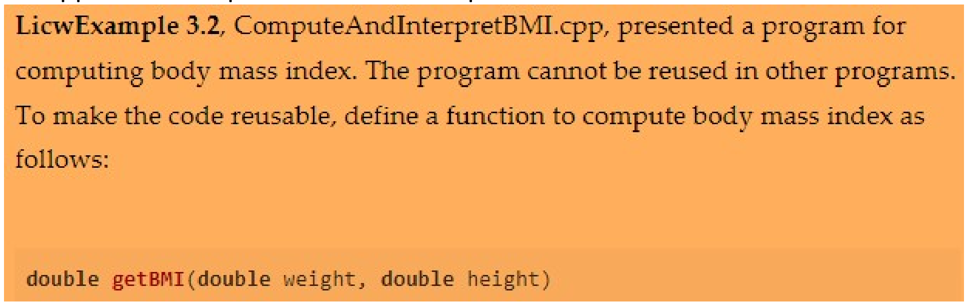
  A constant member function may be an instance or a static function.

  A constant function may be defined for any functions not just instance member functions.

  A constructor can be defined as a constant member function.
-
-

10.7 Thinking in Objects

Friday, March 17, 2023 5:07 PM

- Procedural paradigm focuses on designing functions, oop (- pgrming) paradigm couples data and functions together into objects, software design using the oo paradigm focuses on objects and operations on objects
- In software dev, paradigm describes pattern/model
- Prcedural paradigm solves problems using assignments, selections, loops, fn, and arrays
- Study of procedural paradigm lays foundation for oop
- Oo paradigm gives more flex and modularity for building reusable software
- Oo approach to improve solution for a problem in ch3
- 

  LicwExample 3.2, ComputeAndInterpretBMI.cpp, presented a program for computing body mass index. The program cannot be reused in other programs. To make the code reusable, define a function to compute body mass index as follows:

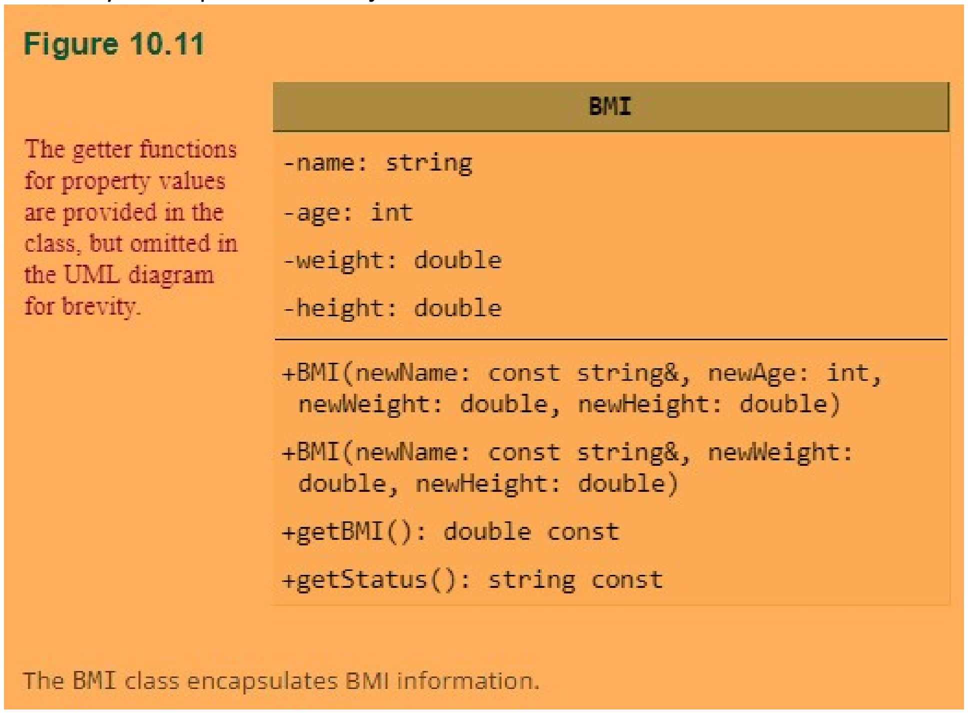
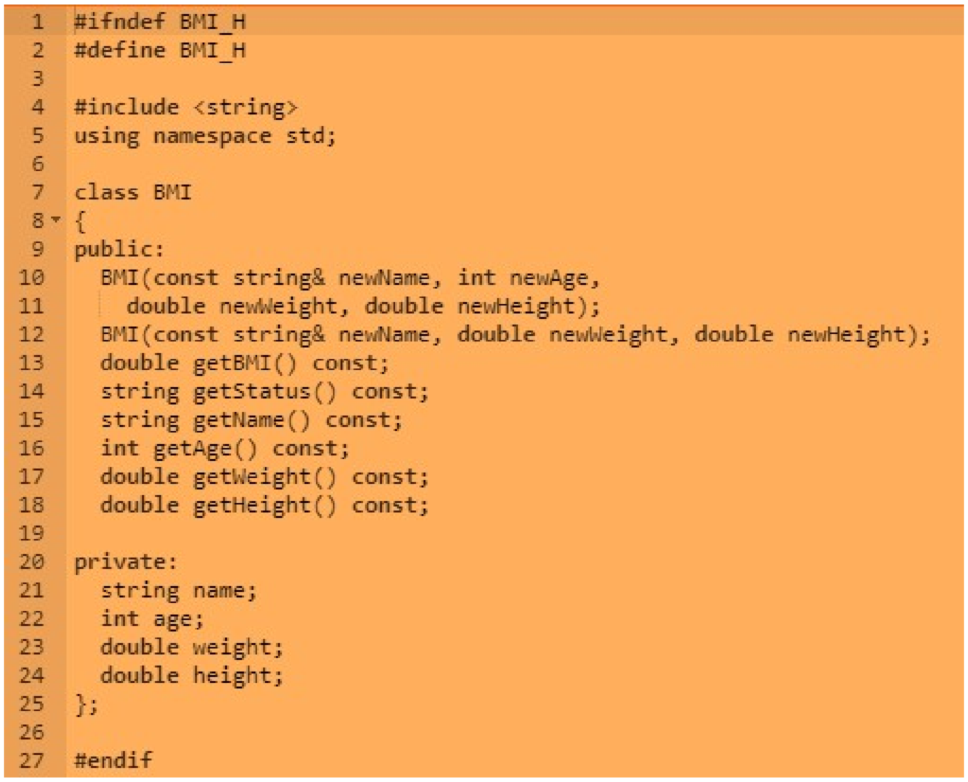
  ```
  double getBMI(double weight, double height)
  ```
- "licw" nice
- Limitations in pgrm
  - Might wanna link weight and height w/ person's name & bday
  - Ideal way to couple is make object that has them
  - 

    Figure 10.11

    The getter functions for property values are provided in the class, but omitted in the UML diagram for brevity.

    ```
    BMI
    -name: string
    -age: int
    -weight: double
    -height: double
    +BMI(newName: const string&, newAge: int,
     newWeight: double, newHeight: double)
    +BMI(newName: const string&, newWeight:
     double, newHeight: double)
    +getBMI(): double const
    +getStatus(): string const
    ```

    The BMI class encapsulates BMI information.
  -



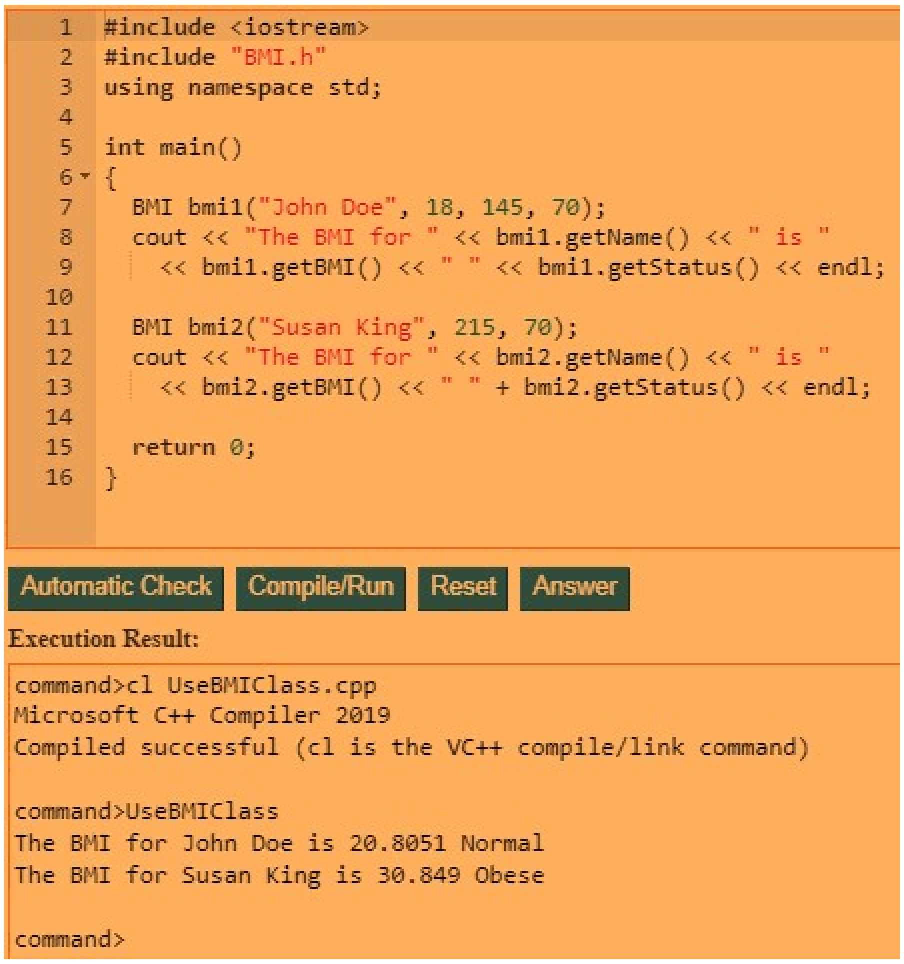
```cpp
#ifndef BMI_H
#define BMI_H

#include <string>
using namespace std;

class BMI
{
public:
  BMI(const string& newName, int newAge,
    double newWeight, double newHeight);
  BMI(const string& newName, double newWeight, double newHeight);
  double getBMI() const;
  string getStatus() const;
  string getName() const;
  int getAge() const;
  double getWeight() const;
  double getHeight() const;

private:
  string name;
  int age;
  double weight;
  double height;
};

#endif
```

Tip

The `string` parameter `newName` is defined as pass-by-reference using the syntax `string& newName`. This improves performance by preventing the compiler from making a copy of the object being passed into the function. Further, the reference is defined `const` to prevent `newName` from being modified accidentally. *You should always pass an object parameter by reference. If the object does not change in the function, define it as a* `const` *reference parameter.*

Tip

If a member function does not change data fields, define it as a `const` *function.* All member functions in the `BMI` class are `const` functions.

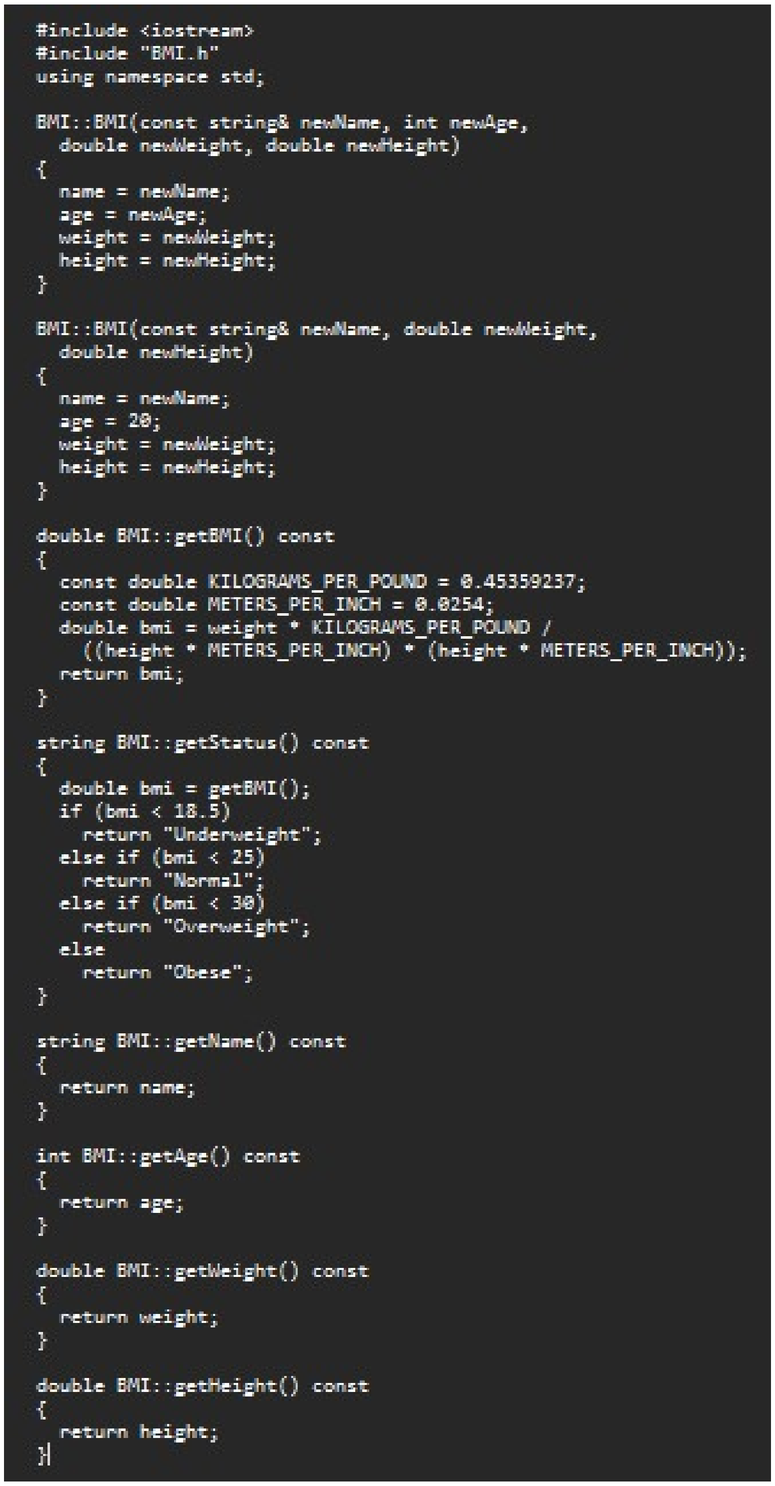
○ 

```
#include <iostream>
#include "BMI.h"
using namespace std;

int main()
{
  BMI bmi1("John Doe", 18, 145, 70);
  cout << "The BMI for " << bmi1.getName() << " is "
    << bmi1.getBMI() << " " << bmi1.getStatus() << endl;

  BMI bmi2("Susan King", 215, 70);
  cout << "The BMI for " << bmi2.getName() << " is "
    << bmi2.getBMI() << " " + bmi2.getStatus() << endl;

  return 0;
}
```

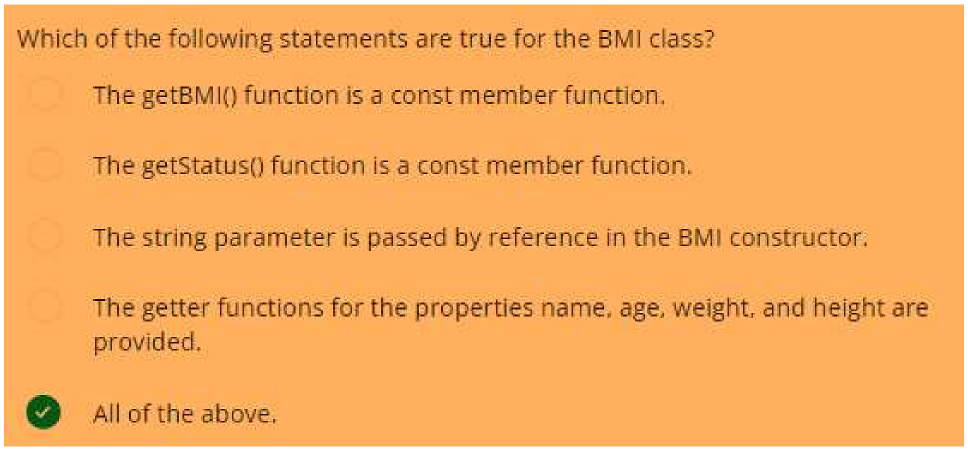
Execution Result:

```
command>cl UseBMIClass.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>UseBMIClass
The BMI for John Doe is 20.8051 Normal
The BMI for Susan King is 30.849 Obese

command>
```

○



```cpp
#include <iostream>
#include "BMI.h"
using namespace std;

BMI::BMI(const string& newName, int newAge,
  double newWeight, double newHeight)
{
  name = newName;
  age = newAge;
  weight = newWeight;
  height = newHeight;
}

BMI::BMI(const string& newName, double newWeight,
  double newHeight)
{
  name = newName;
  age = 20;
  weight = newWeight;
  height = newHeight;
}

double BMI::getBMI() const
{
  const double KILOGRAMS_PER_POUND = 0.45359237;
  const double METERS_PER_INCH = 0.0254;
  double bmi = weight * KILOGRAMS_PER_POUND /
    ((height * METERS_PER_INCH) * (height * METERS_PER_INCH));
  return bmi;
}

string BMI::getStatus() const
{
  double bmi = getBMI();
  if (bmi < 18.5)
    return "Underweight";
  else if (bmi < 25)
    return "Normal";
  else if (bmi < 30)
    return "Overweight";
  else
    return "Obese";
}

string BMI::getName() const
{
  return name;
}

int BMI::getAge() const
{
  return age;
}

double BMI::getWeight() const
{
  return weight;
}

double BMI::getHeight() const
{
  return height;
}
```



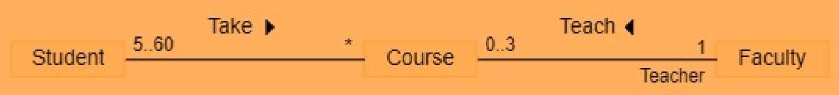
Which of the following statements are true for the BMI class?

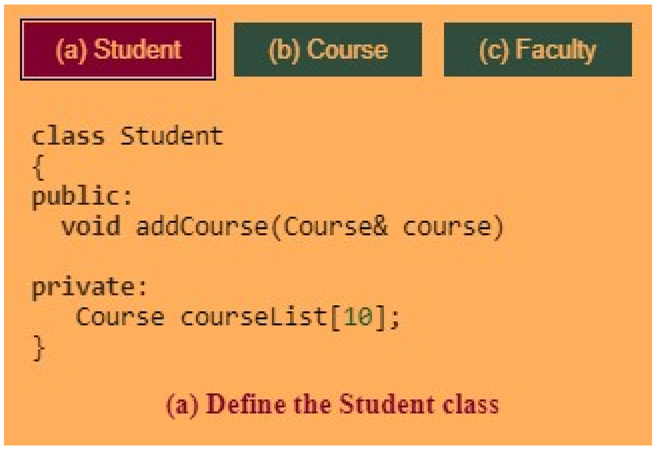
- The getBMI() function is a const member function.
- The getStatus() function is a const member function.
- The string parameter is passed by reference in the BMI constructor.
- The getter functions for the properties name, age, weight, and height are provided.
- [✓ All of the above.]

10.8 Class Relationships

Friday, March 17, 2023 5:21 PM

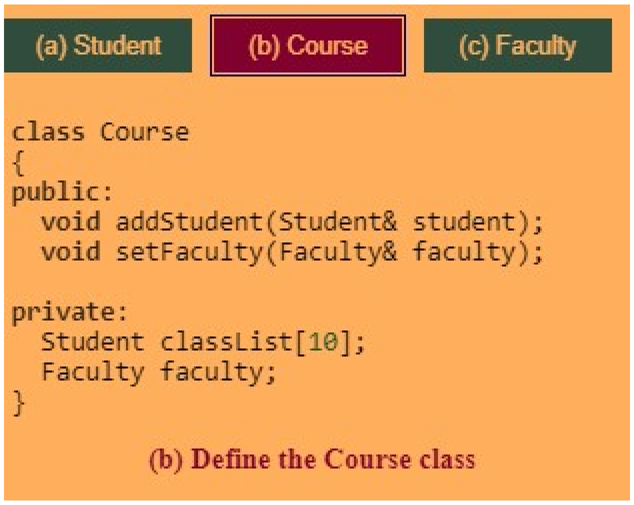
- To design classes, need to explore relationships among classes, common relationships among classes are **association, composition, and inheritance**
- 10.8.1 Association
  - General binary relationship that describes an activity btwn 2 classes
  - Like a student taking a course, association btwn Student class and Course class
  - Faculty member teaching a course, association btwn Faculty class and Course class
  - **Figure 10.12**

    This UML diagram shows that a student may take any number of courses, a faculty member may teach at most three courses, a course may have from five to sixty students, and a course is taught by only one faculty member.
  - Association illustrated by solid line btwn 2 classes w/ optional label that describes relationship
  - Labels are Take and Teach, triangle shows direction of relationship
  - Relationship also have role names, like Student, Course, and Faculty
  - Each class has multiplicity, could be numb/interval that specifies how many of the class's objects are involved in relationship, * means unlimited numb of objects, interval means in btwn the 2 numbs
  - So each student can take infinite numb of couses, each couse must have at least 5 and at most 60 students, each couse taught by 1 faculty memb, each faculty memb can teach 0-3 courses per semester
  - Implement associations by using data fields and fn
  -
  - 

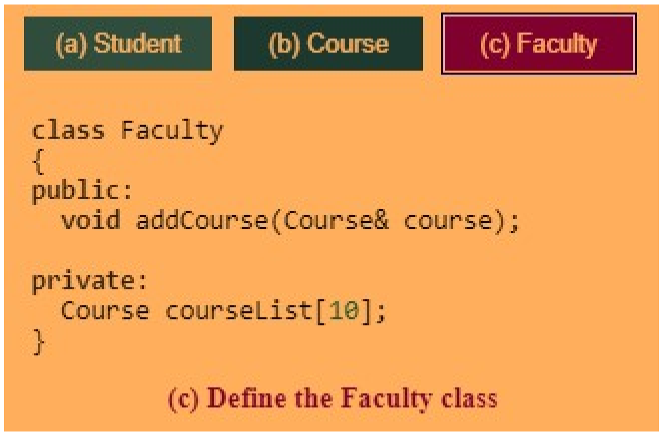
```
class Student
{
public:
  void addCourse(Course& course)

private:
   Course courseList[10];
}
```

    (a) Define the Student class
  -



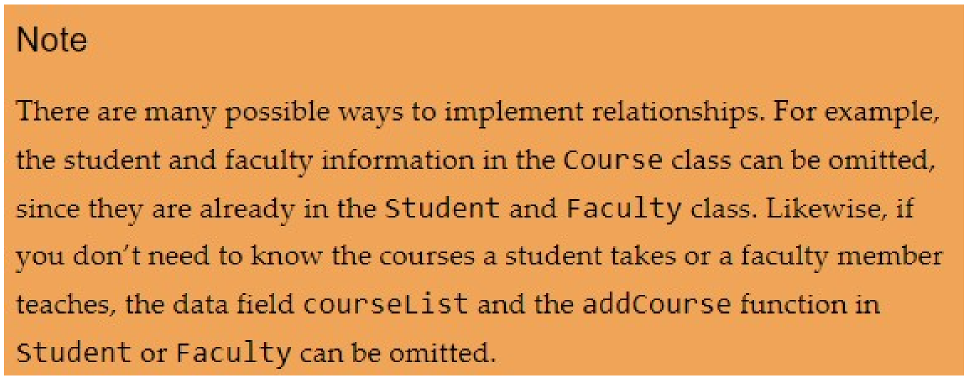
```
class Course
{
public:
  void addStudent(Student& student);
  void setFaculty(Faculty& faculty);

private:
  Student classList[10];
  Faculty faculty;
}
```

(b) Define the Course class



```
class Faculty
{
public:
  void addCourse(Course& course);

private:
  Course courseList[10];
}
```

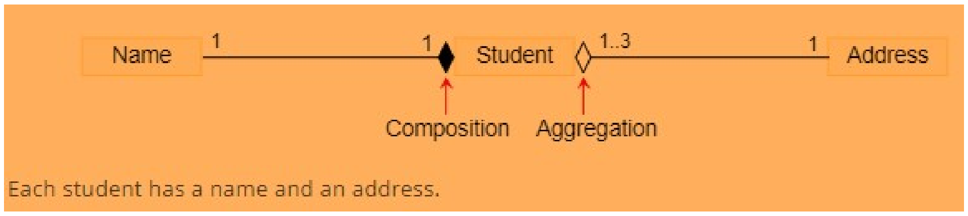
(c) Define the Faculty class



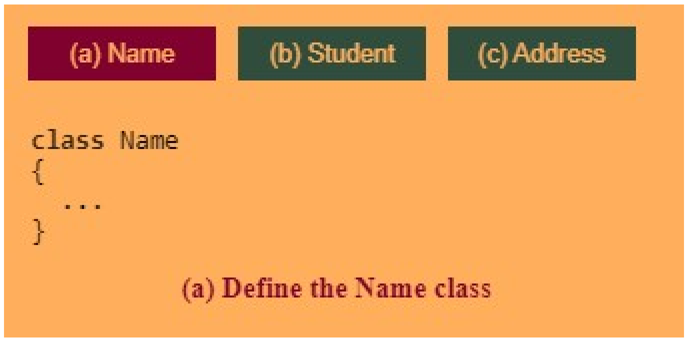
Note

There are many possible ways to implement relationships. For example, the student and faculty information in the `Course` class can be omitted, since they are already in the `Student` and `Faculty` class. Likewise, if you don't need to know the courses a student takes or a faculty member teaches, the data field `courseList` and the `addCourse` function in `Student` or `Faculty` can be omitted.

- 10.8.2 Aggregation and Composition
  - Aggregation = special form of association that reps an ownership relationship btwn 2 objects
  - Aggregation models has-a relationships
  - Owner object called aggregating object, class is called aggregating class
  - Subject object called aggregated object, class is aggregated class
  - Aggregation btwn 2 objects is composition if existance of aggregated object depends on aggregating object
  - If relationship is composition, aggregated object cannot exist on its own
  - "Student has a name" name can't exist on its own, but "Student has an address" address can exist on its own
  - Composition implies exclusive ownership
  - Filled diamond is attached to an aggregating class to show relation w/ aggregated class, empty diamond attached to aggregating class to denote aggregation relation w/ an aggregated class

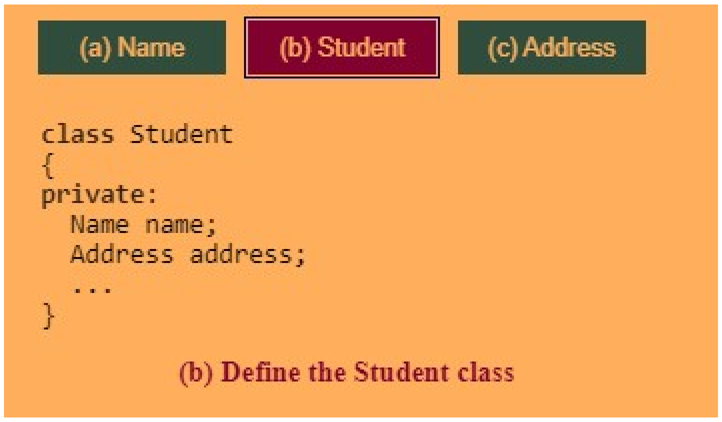


Each student has a name and an address.



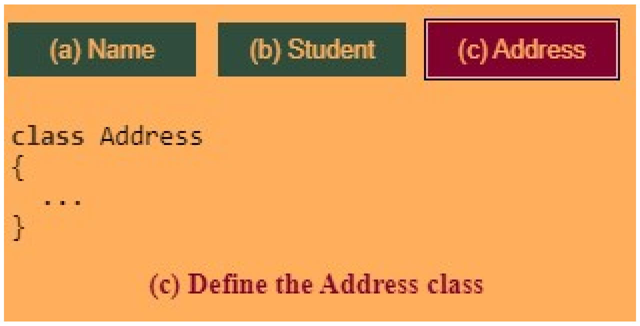
```
class Name
{
  ...
}
```

(a) Define the Name class



```
class Student
{
private:
  Name name;
  Address address;
  ...
}
```

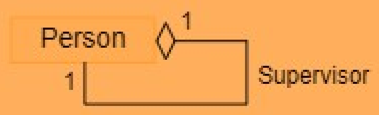
(b) Define the Student class



```
class Address
{
  ...
}
```

(c) Define the Address class

Figure 10.16



A person may have a supervisor.

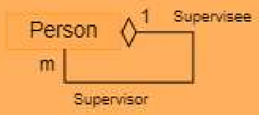
- Aggregation can exist btwn objects of same class

```
class Person
{
private:
```

```
class Person
{
private:
  Person supervisor;  // The type for the data is the class itself
  ...
}
```

If a person may have several supervisors, as shown in **Figure 10.17**, you may use an array to store supervisors (for example, 10 supervisors).

Figure 10.17



```
class Person
{ ...

private:
  Person supervisors[10];
}
```

A person may have several supervisors.

Note

Since aggregation and composition relationships are represented using classes in similar ways, we will not differentiate them and call both compositions for simplicity.

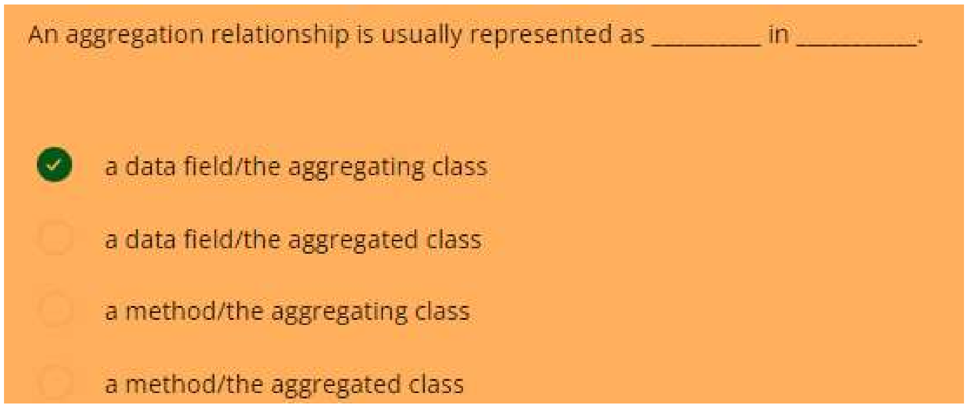
__________ is attached to the class of the composing class to denote the aggregation relationship with the composed object.

[✓ An empty diamond]

A solid diamond

An empty oval

A solid oval


An aggregation relationship is usually represented as _________ in __________.

- [✓ a data field/the aggregating class]
- a data field/the aggregated class
- a method/the aggregating class
- a method/the aggregated class

10.9 Case Study: The StackOfIntegers Class

Friday, March 17, 2023 6:10 PM

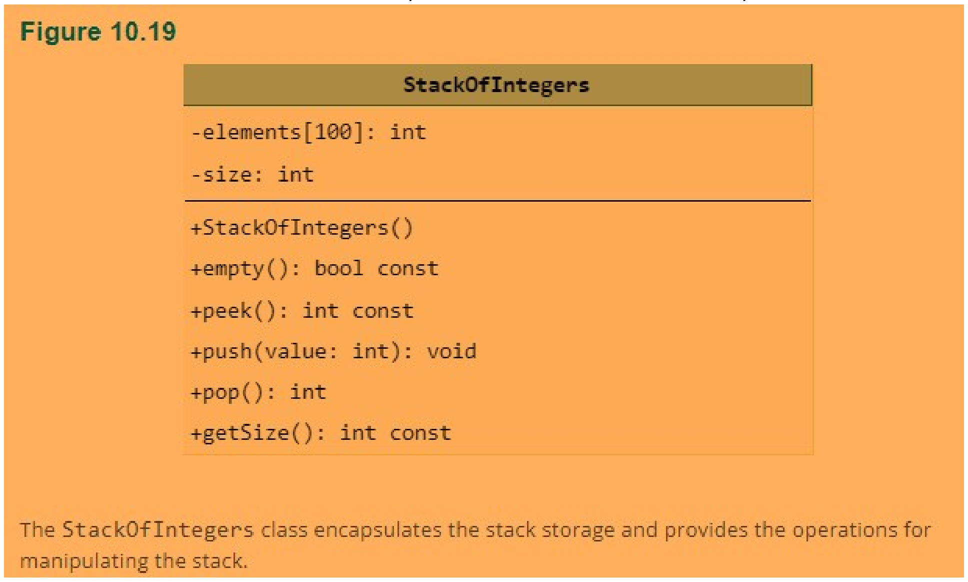
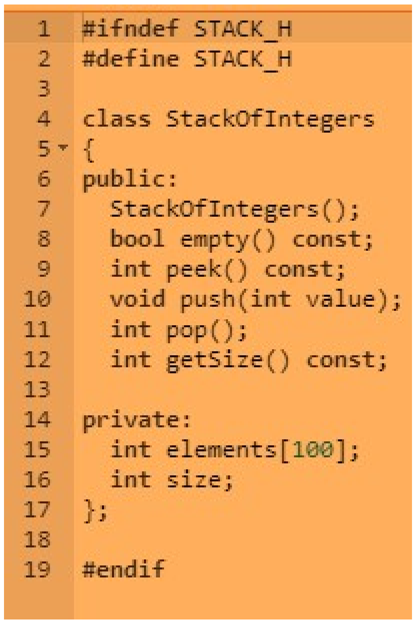
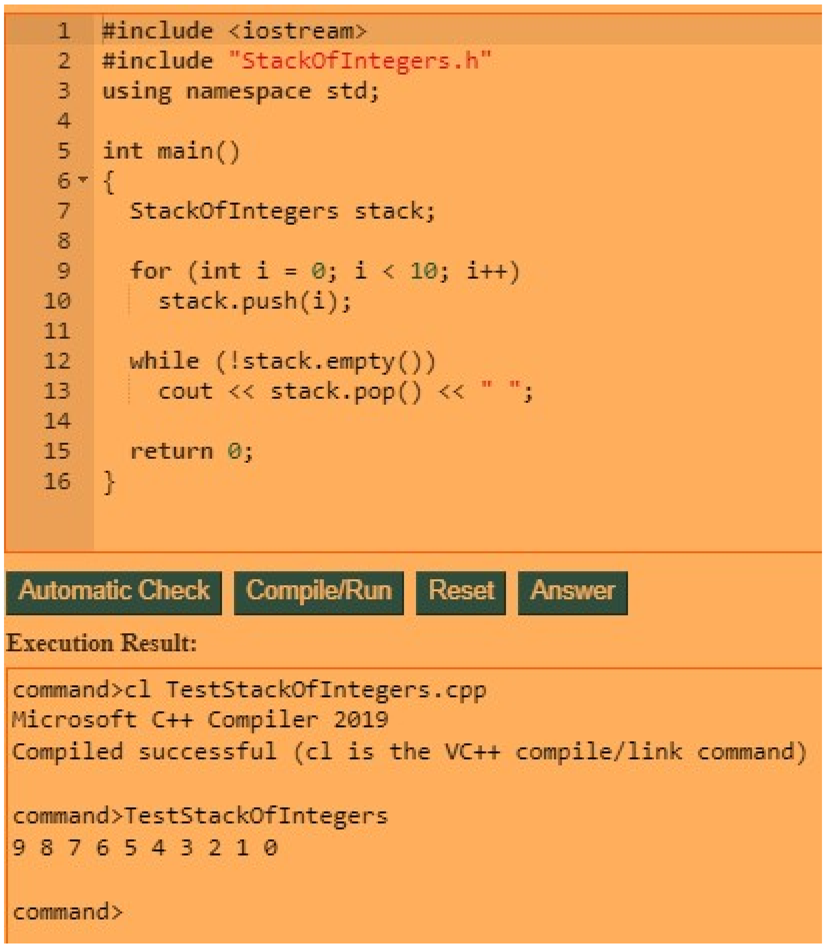
- This sections designs a class for modeling stacks
- Stack is data structure, last-in, first-out
- Many applications, compiler uses stack to process functions
- Can define a class to model stacks, assume stack holds int vals, so name of stack is StackOfIntegers
- 

  Figure 10.19

  ```
  StackOfIntegers
  -elements[100]: int
  -size: int
  +StackOfIntegers()
  +empty(): bool const
  +peek(): int const
  +push(value: int): void
  +pop(): int
  +getSize(): int const
  ```

  The StackOfIntegers class encapsulates the stack storage and provides the operations for manipulating the stack.
- Say class is available, write test pgrm that uses class to make a stack, store 10 ints , then display them in reverse order
- 

  ```
  #ifndef STACK_H
  #define STACK_H

  class StackOfIntegers
  {
  public:
    StackOfIntegers();
    bool empty() const;
    int peek() const;
    void push(int value);
    int pop();
    int getSize() const;

  private:
    int elements[100];
    int size;
  };

  #endif
  ```
-

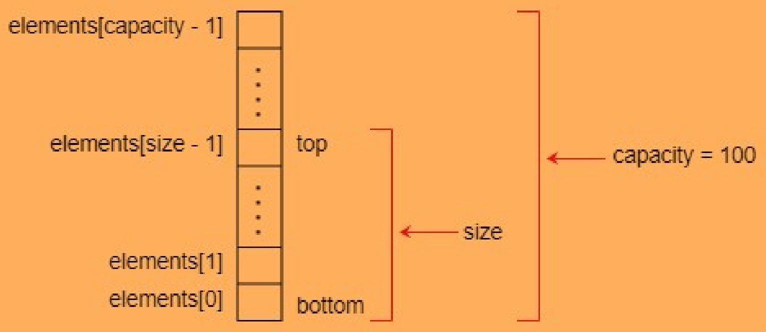
- 

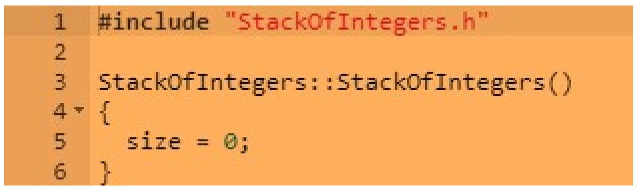
```
#include <iostream>
#include "StackOfIntegers.h"
using namespace std;

int main()
{
  StackOfIntegers stack;

  for (int i = 0; i < 10; i++)
    stack.push(i);

  while (!stack.empty())
    cout << stack.pop() << " ";

  return 0;
}
```

Execution Result:

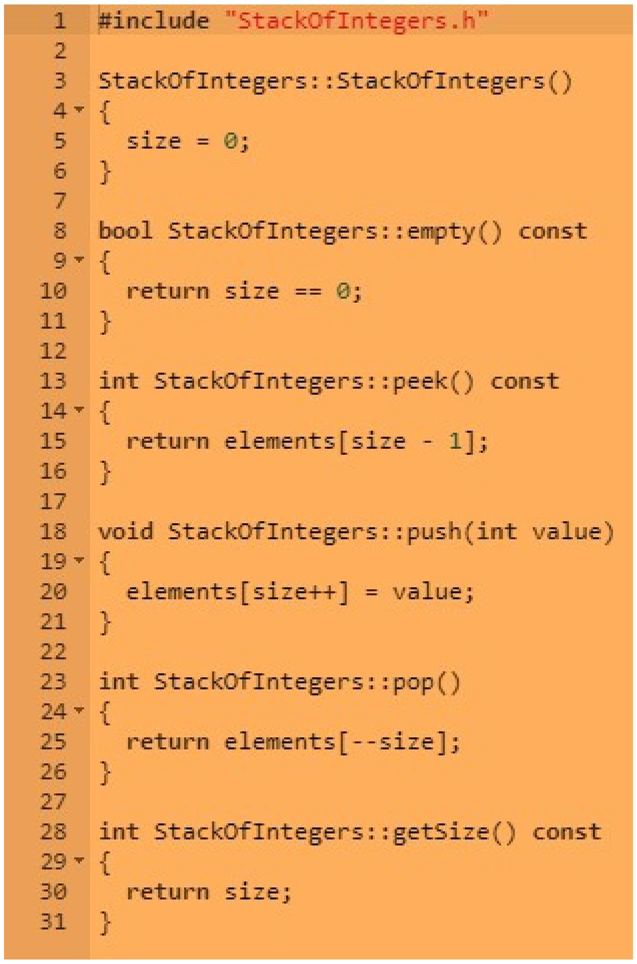
```
command>cl TestStackOfIntegers.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>TestStackOfIntegers
9 8 7 6 5 4 3 2 1 0

command>
```

- Implement StackOfIntegers class by:
  - Elements in stack are stored in an array named elements
  - When make a stack, array also made
  - No-arg constructor initializes size to 0
  - Var size counts numb of elements in stack, and size-1 is index of element @ top of stack
  - For empty stack, size is 0
  - **Figure 10.20**

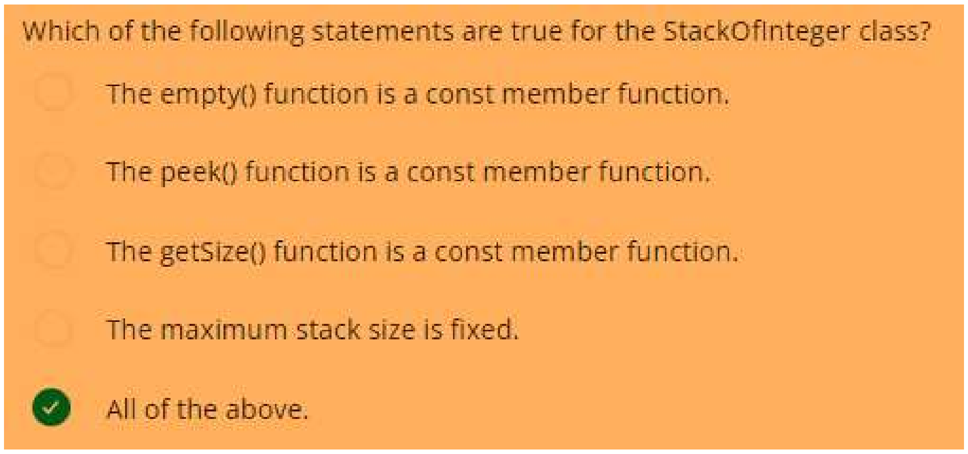
    

    The StackOfIntegers class uses an array to store the elements in a stack.
  - 

```
#include "StackOfIntegers.h"

StackOfIntegers::StackOfIntegers()
{
  size = 0;
}
```

○ 

```cpp
#include "StackOfIntegers.h"

StackOfIntegers::StackOfIntegers()
{
  size = 0;
}

bool StackOfIntegers::empty() const
{
  return size == 0;
}

int StackOfIntegers::peek() const
{
  return elements[size - 1];
}

void StackOfIntegers::push(int value)
{
  elements[size++] = value;
}

int StackOfIntegers::pop()
{
  return elements[--size];
}

int StackOfIntegers::getSize() const
{
  return size;
}
```

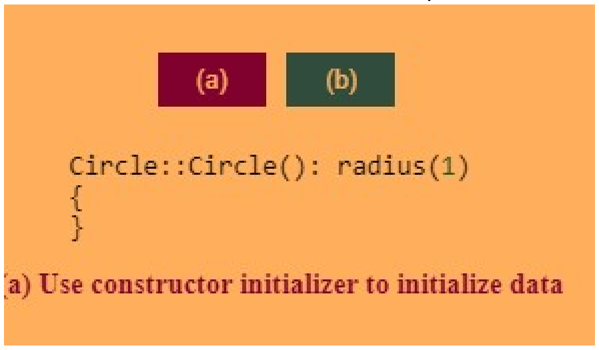
- 

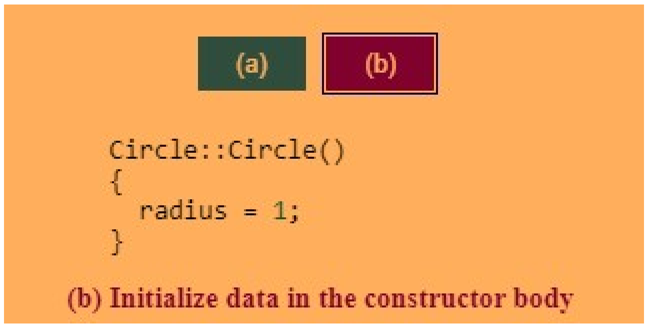
Which of the following statements are true for the StackOfInteger class?

- The empty() function is a const member function.
- The peek() function is a const member function.
- The getSize() function is a const member function.
- The maximum stack size is fixed.
- [✓ All of the above.]

10.10 Constructor initializer Lists

Friday, March 17, 2023 6:22 PM

- Constructor initializer lists can be used to set initial vals for object data fields
- Data fields may be initialized in the constructor using an initializer list in this syntax:
-
```
ClassName(parameterList)
  : datafield1(value1), datafield2(value2) // Initializer list
{
  // Additional statements if needed
}
```
- Initializer list initializes datafield1 w/ val1 and datafield2 w/ val2
- 

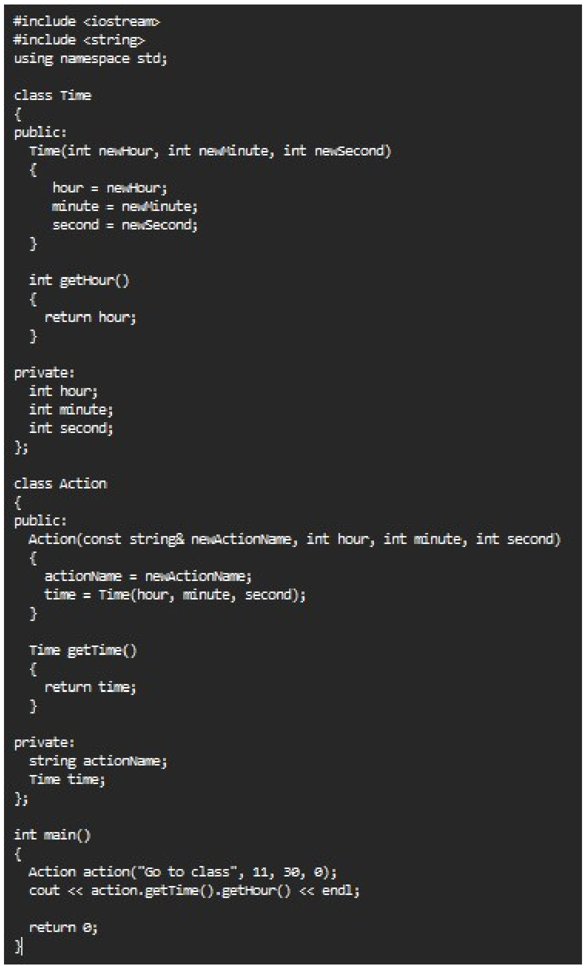
```
Circle::Circle(): radius(1)
{
}
```
(a) Use constructor initializer to initialize data
-
- 

```
Circle::Circle()
{
  radius = 1;
}
```
(b) Initialize data in the constructor body
- B better than a, but using an initializer list is necessary to initialize object data fields that don't have a no-arg constructor
- In C++, can declare an object data field, like name is declared as a string object in this code
-
```
class Student
{
public:
  Student();

private:
  string name;
};
```
- But, declaring an object data field in a class is different from declaring a local object in a fn like this:
-
```
int main()
{
  string name;
```

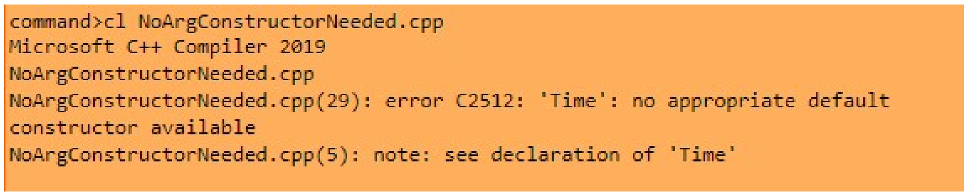
-
  ```
  int main()
  {
    string name;
  };
  ```
- As object data field, object not made when its declared, when object declared in a fn, object made when its declared
- C++11, primitive data fields can be declared w/ an initial val, but a data field of the object type must be declared w/ a no-arg constructor, like this is wrong:

-
  ```
  class Student
  {
  public:
    Student();

  private:
    int age = 5; // OK
    string name("Peter"); // Wrong, must use a no-arg constructor
  };
  ```
-
- The correct declaration is

  ```
  class Student
  {
  public:
    Student();

  private:
    int age = 5;
    string name; // Declare a data field of the string type
  ```
- If data field is of an object type, the no-arg constructor for the object type is auto invoked to make an object for the data field
- If no-arg constructor doesn't exist, compile error will happen, like this:

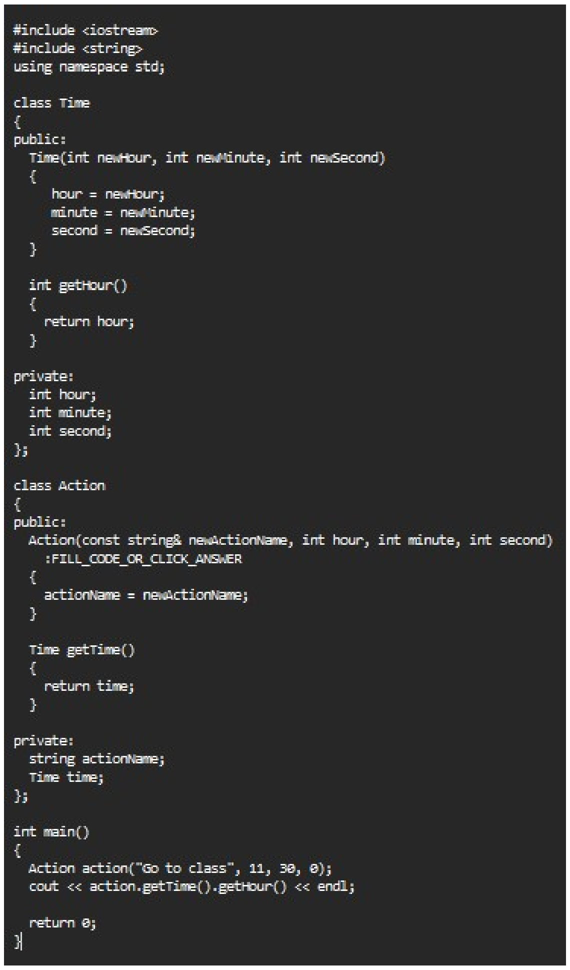
- 

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

class Time
{
public:
  Time(int newHour, int newMinute, int newSecond)
  {
     hour = newHour;
     minute = newMinute;
     second = newSecond;
  }

  int getHour()
  {
    return hour;
  }

private:
  int hour;
  int minute;
  int second;
};

class Action
{
public:
  Action(const string& newActionName, int hour, int minute, int second)
  {
    actionName = newActionName;
    time = Time(hour, minute, second);
  }

  Time getTime()
  {
    return time;
  }

private:
  string actionName;
  Time time;
};

int main()
{
  Action action("Go to class", 11, 30, 0);
  cout << action.getTime().getHour() << endl;

  return 0;
}
```

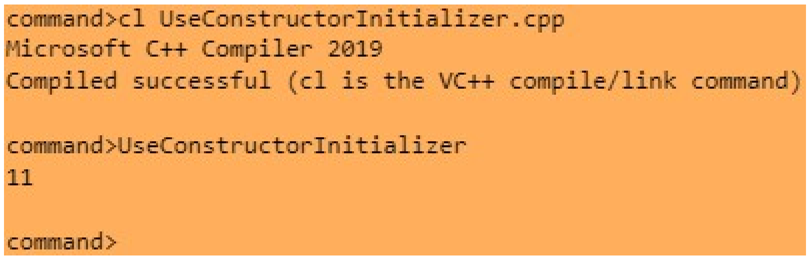
- 

```
command>cl NoArgConstructorNeeded.cpp
Microsoft C++ Compiler 2019
NoArgConstructorNeeded.cpp
NoArgConstructorNeeded.cpp(29): error C2512: 'Time': no appropriate default
constructor available
NoArgConstructorNeeded.cpp(5): note: see declaration of 'Time'
```

- To fix, use the constructor initializer list like this

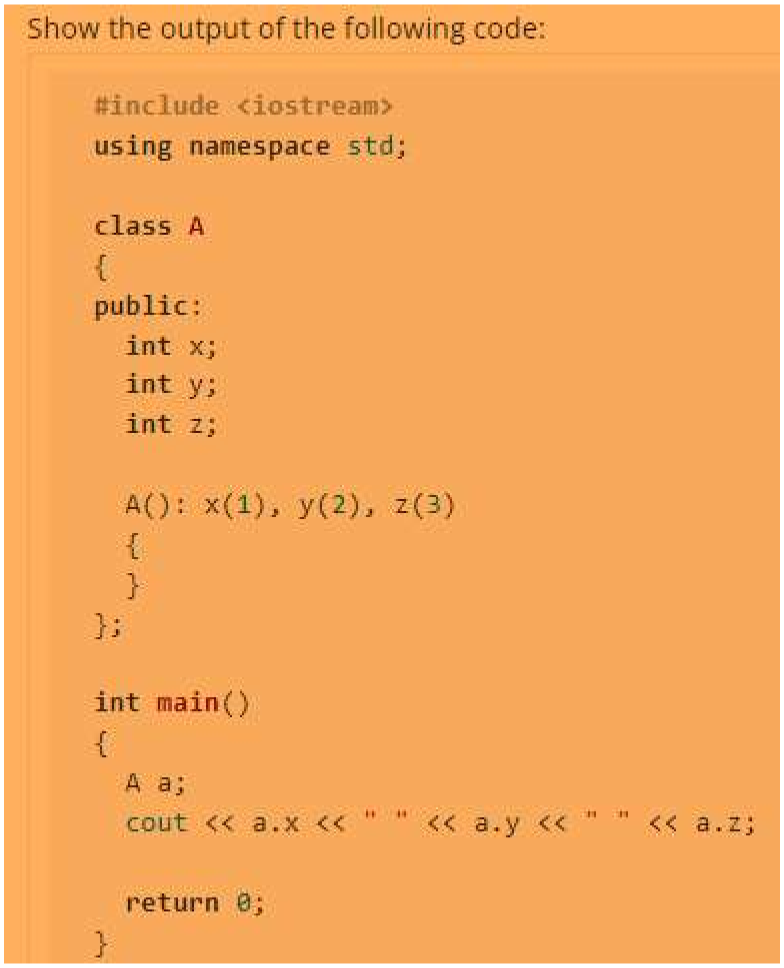
- 

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

class Time
{
public:
  Time(int newHour, int newMinute, int newSecond)
  {
     hour = newHour;
     minute = newMinute;
     second = newSecond;
  }

  int getHour()
  {
    return hour;
  }

private:
  int hour;
  int minute;
  int second;
};

class Action
{
public:
  Action(const string& newActionName, int hour, int minute, int second)
    :FILL_CODE_OR_CLICK_ANSWER
  {
    actionName = newActionName;
  }

  Time getTime()
  {
    return time;
  }

private:
  string actionName;
  Time time;
};

int main()
{
  Action action("Go to class", 11, 30, 0);
  cout << action.getTime().getHour() << endl;

  return 0;
}
```

- 

```
command>cl UseConstructorInitializer.cpp
Microsoft C++ Compiler 2019
Compiled successful (cl is the VC++ compile/link command)

command>UseConstructorInitializer
11

command>
```

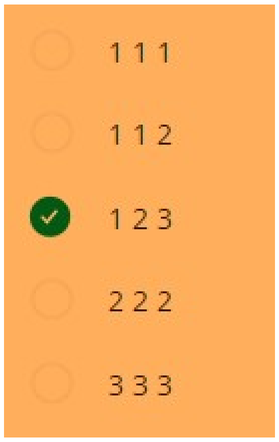
-



- Show the output of the following code:

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

class A
{
public:
  int x;
  int y;
  int z;

  A(): x(1), y(2), z(3)
  {
  }
};

int main()
{
  A a;
  cout << a.x << " " << a.y << " " << a.z;

  return 0;
}
```

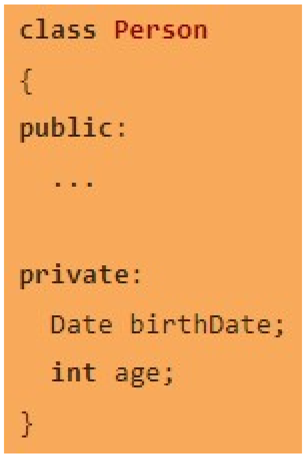
-



- 1 1 1
  1 1 2
  [✓ 1 2 3]
  2 2 2
  3 3 3
-
-

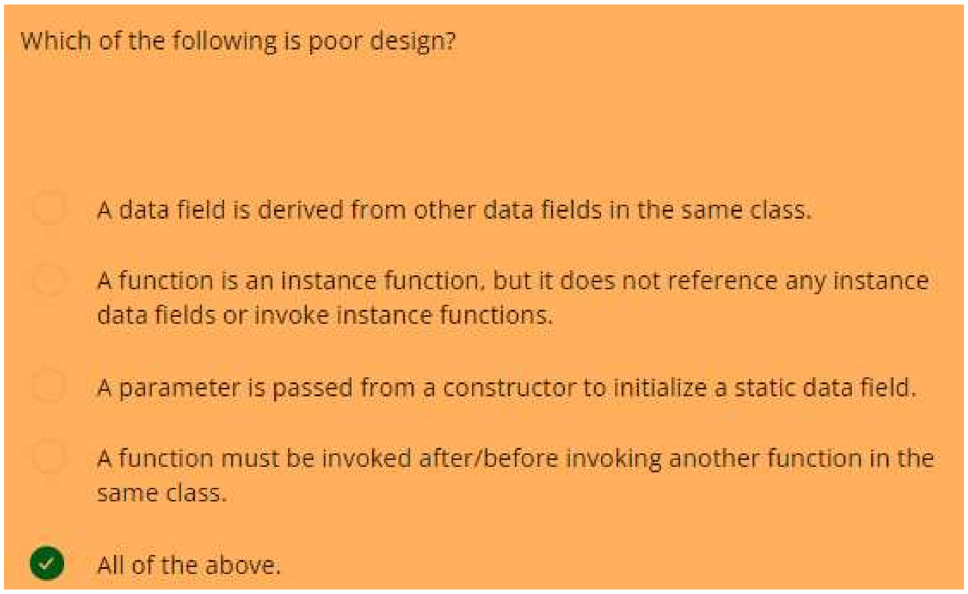
10.11 Class Design Guidelines

Friday, March 17, 2023 6:36 PM

- Class design guidelines are helpful for designing sound classes
- Oo design, UML best notation
- Guidelines:
- 10.11.1 Cohesion
    - Class should describe a single entity, and all class operations should logically fit together to support a coherent purpose
    - EX: Can use a class for students, but shouldn't combo students and staff in same class bc they diff entities
    - Single entity w/ too many responsibilities can be broken into many classes to separate responsibilities
- 10.11.2 Consistency
    - Follow standard pgrming style and naming conventions
    - Choose informative names for classes, data fields, functions
    - Pop style in C++ is place data declaration after the functions, and place constructors b4 fn
    - Choose names consistently, choose same names for similar ops using function overloading (good practice)
    - Generally, consistently give a public no-arg constructor for constructing a default instance, if class don't support a no-arg constructor, doc the reason, if no constructors are defined explicitly, a public default no-arg constructor w/ an empty body is assumed
- 10.11.3 Encapsulation
    - Class should use the private modifier to hide its data from direct access by clients, makes class easy to maintain
    - Give a getter fn only if u want field to be readable, setter fn only if u want field to be updateable
    - Class should also hide fn not intended for client use, define these fn as private
- 10.11.4 Clarity
    - Clear contract, easy to explain and understand
    - Users incorporate classes in many diff combo, orders, and env, so should design class that imposes no restrictions on what user can do w/ it or when, design properties in way that lets user set them in any order and w/ any combo of vals and design fn independently of their order of occurrence
    - Ex:
    - 

```
class Person
{
public:
  ...

private:
  Date birthDate;
  int age;
}
```

    - U shouldn't declare a data field that can be derived from other data fields
- 10.11.5 Completeness

    - Classes are designed for use by many diff customers
    - To be useful for more things, give variety of ways to customize thru properties and fn
    - Ex: the string class has more than 20 fn that are very useful
- 10.11.6 Instance Vs. Static
    - Var/fn that depends on specific instance of the class hsould be an instance var/fn, var that is shared by all the instances of a class should be declared static
    - Fn not dependent on specific instance should be defined as a static fn
    - Always reference static vars & fn from a class name (instead of object) to improve readability and avoid errors
    - Constructor always instance bc used to make a specific instance
    - Static var/fn can be invoked from an instance fn, but an instance var/fn cant be invoked from a static fn
-
- 

  Which of the following is poor design?

  A data field is derived from other data fields in the same class.

  A function is an instance function, but it does not reference any instance data fields or invoke instance functions.

  A parameter is passed from a constructor to initialize a static data field.

  A function must be invoked after/before invoking another function in the same class.

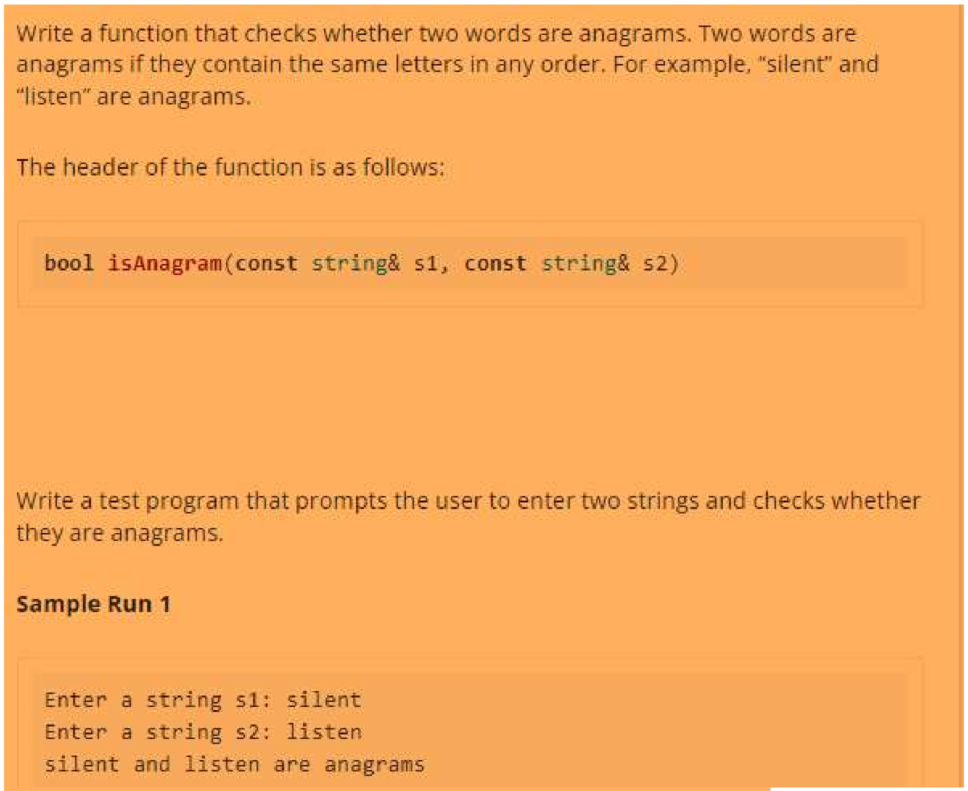
  [✓ All of the above.]
-

Ch 10 Terms and Review

Friday, March 17, 2023 6:52 PM

aggregation
composition
constant function
Constructor initializer list
has-a relationship
instance data field
instance function
instance variable
multiplicity
static function
static variable

1. The C++ string class encapsulates an array of characters and provides many functions for processing strings such as append, assign, at, clear, erase, empty, length, c_str, compare, substr, find, insert, and replace.
2. C++ supports operators ([ ], =, +, +=, <<, >>, ==, !=, <, <=, >, >=) to simplify string operations.
3. You can use cin to read a string ending with a whitespace character and use getline(cin, s, delimiterCharacter) to read a string ending with the specified delimiter character.
4. You can pass an object to a function by value or by reference. For performance, passing by reference is preferred.
5. If the function does not change the object being passed, define the object parameter as a constant reference parameter to prevent the object's data being modified accidentally.
6. An instance variable or function belongs to an instance of a class. Its use is associated with individual instances.
7. A static variable is a variable shared by all instances of the same class.
8. A static function is a function that can be invoked without using instances.
9. Every instance of a class can access the class's static variables and functions. For clarity, however, it is better to invoke static variables and functions using ClassName::staticVariable and ClassName::functionName(arguments).
10. f a function does not change the data fields of an object, define the function constant to prevent errors.
11. A constant function does not change the values of any data fields.
12. You can specify a member function to be constant by placing the const modifier at the end of the function declaration.
13. The object-oriented approach combines the power of the procedural paradigm with an added dimension that integrates data with operations into objects.
14. The procedural paradigm focuses on designing functions. The object-oriented paradigm couples data and functions together into objects.
15. Software design using the object-oriented paradigm focuses on objects and operations on objects.
16. An object can contain another object. The relationship between the two is called composition.
17. Some guidelines for class design are cohesion, consistency, encapsulation, clarity, and completeness.



Write a function that checks whether two words are anagrams. Two words are anagrams if they contain the same letters in any order. For example, "silent" and "listen" are anagrams.

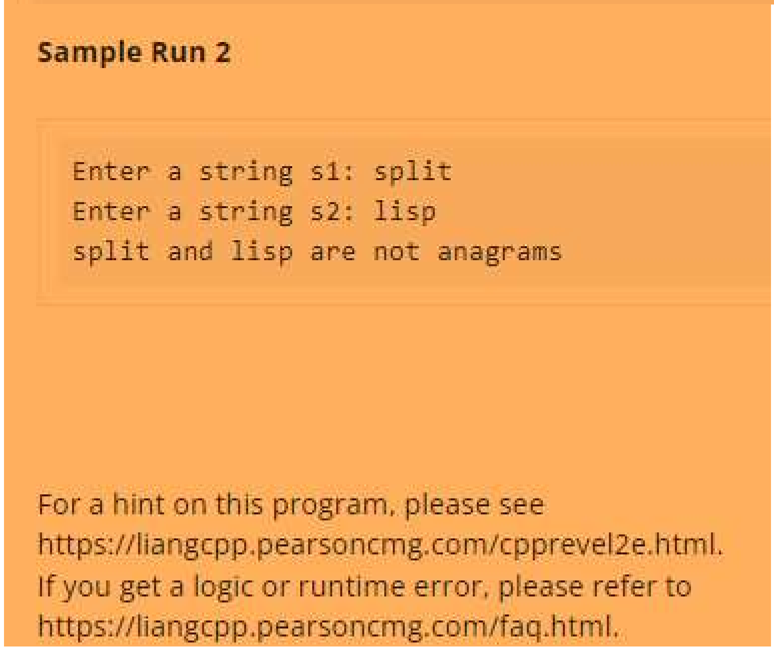
The header of the function is as follows:

```
bool isAnagram(const string& s1, const string& s2)
```

Write a test program that prompts the user to enter two strings and checks whether they are anagrams.

Sample Run 1

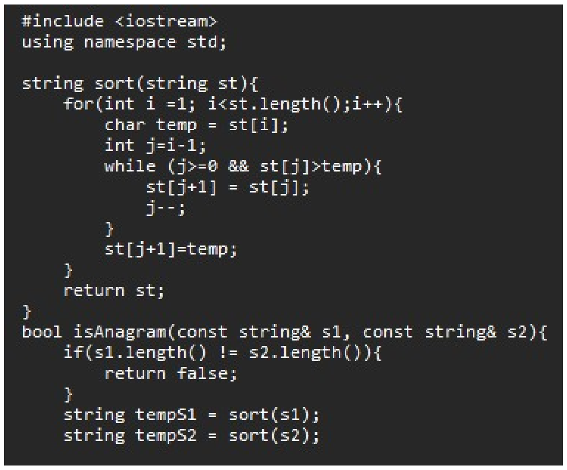
```
Enter a string s1: silent
Enter a string s2: listen
silent and listen are anagrams
```



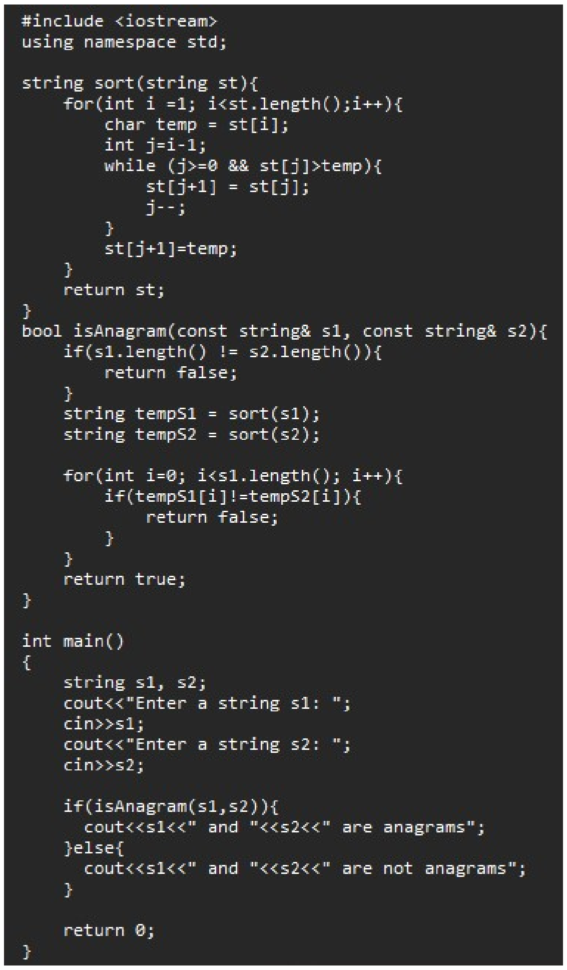
Sample Run 2

```
Enter a string s1: split
Enter a string s2: lisp
split and lisp are not anagrams
```

For a hint on this program, please see https://liangcpp.pearsoncmg.com/cpprevel2e.html. If you get a logic or runtime error, please refer to https://liangcpp.pearsoncmg.com/faq.html.



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

string sort(string st){
    for(int i =1; i<st.length();i++){
        char temp = st[i];
        int j=i-1;
        while (j>=0 && st[j]>temp){
            st[j+1] = st[j];
            j--;
        }
        st[j+1]=temp;
    }
    return st;
}
bool isAnagram(const string& s1, const string& s2){
    if(s1.length() != s2.length()){
        return false;
    }
    string tempS1 = sort(s1);
    string tempS2 = sort(s2);
```


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

string sort(string st){
    for(int i =1; i<st.length();i++){
        char temp = st[i];
        int j=i-1;
        while (j>=0 && st[j]>temp){
            st[j+1] = st[j];
            j--;
        }
        st[j+1]=temp;
    }
    return st;
}
bool isAnagram(const string& s1, const string& s2){
    if(s1.length() != s2.length()){
        return false;
    }
    string tempS1 = sort(s1);
    string tempS2 = sort(s2);

    for(int i=0; i<s1.length(); i++){
        if(tempS1[i]!=tempS2[i]){
            return false;
        }
    }
    return true;
}

int main()
{
    string s1, s2;
    cout<<"Enter a string s1: ";
    cin>>s1;
    cout<<"Enter a string s2: ";
    cin>>s2;

    if(isAnagram(s1,s2)){
      cout<<s1<<" and "<<s2<<" are anagrams";
    }else{
      cout<<s1<<" and "<<s2<<" are not anagrams";
    }

    return 0;
}
```