12.1 Objects: Introduction

Grouping things into objects

The physical world is made up of material items like wood, metal, plastic, fabric, etc. To keep the world understandable, people deal with higher-level objects, like chairs, tables, and TV's. Those objects are groupings of the lower-level items.

Likewise, a program is made up of items like variables and functions. To keep programs understandable, programmers often deal with higher-level groupings of those items known as objects. In programming, an ***object*** is a grouping of data (variables) and operations that can be performed on that data (functions).

PARTICIPATION ACTIVITY | 12.1.1: The world is viewed not as materials, but rather as objects.

Green fabric
Wood
Red fabric
Wood
Wood
Metal bar

Chair
Sit
Green fabric
Wood

Couch
Sit
Lie down
Red fabric
Wood

Drawer
Put stuff in
Take stuff out
Wood
Metal bar

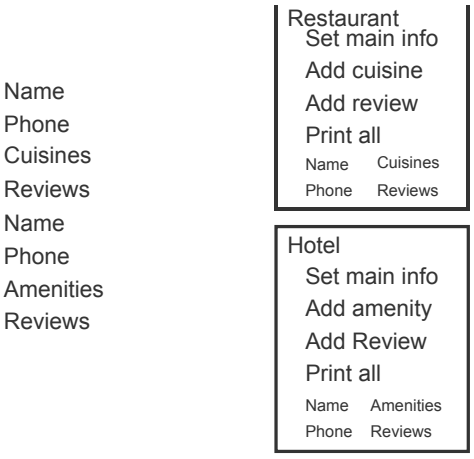
Animation content:

Step 1: A list of materials from an image of a seating area reads green fabric, wood, red fabric, wood, wood, metal bar. Step 2: The material list is separated into object categories. The chair category includes the materials green fabric and wood. The couch category includes red fabric and wood. The drawer category includes wood and metal bar. Step 3: Operations are added to each of the categories. Sit is added to the chair category. Sit and lie down are added to the couch category. Put stuff in and take stuff out are added to the drawer category.

Animation captions:

1. The world consists of items like, wood, metal, fabric, etc.
2. But people think in terms of higher-level objects, like chairs, couches, and drawers.
3. In fact, people think mostly of the operations that can be done with the object. For a drawer, operations are put stuff in, or take stuff out.

PARTICIPATION ACTIVITY | 12.1.2: Programs commonly are not viewed as variables and functions/methods, but rather as objects.



Animation content:

To the left is a list of variables: Name, Phone, Cuisines, Reviews, Name, Phone, Amenities, Reviews.

To the right, these variables are split into two categories, Restaurant and Hotel.
In the restaurant category, we have the following set of operations and objects:

Set main info
Add cuisine
Add review
Print all
Name
Cuisines
Phone
Reviews

In the hotel category, we have the following set of operations and objects:

Set main info
Add amenity
Add review
Print all
Name
Amenities
Phone
Reviews

Animation captions:

1. A program consists of variables and functions/methods. But programmers may prefer to think of higher-level objects like Restaurants and Hotels.
2. In fact, programmers think mostly of the operations that can be done with the object, like setting main info, or adding a review.

PARTICIPATION ACTIVITY 12.1.3: Objects.

Some of the variables and functions for a used-car inventory program are to be grouped into an object type named CarOnLot. Select True if the item should become part of the CarOnLot object type, and False otherwise.

1) int carStickerPrice;
   - True
   - False

2) double todaysTemperature;
   - True
   - False

3) int daysOnLot;
   - True
   - False

4) int origPurchasePrice;
   - True
   - False

5) int numSalespeople;
   - True
   - False

6) GetDaysOnLot()
   - True
   - False

7) DecreaseStickerPrice()
   - True
   - False

8) DetermineTopSalesperson()
   - True
   - False

Abstraction / Information hiding

Abstraction means to have a user interact with an item at a high-level, with lower-level internal details hidden from the user (aka ***information hiding*** or ***encapsulation***). Ex: An oven supports an abstraction of a food compartment and a knob to control heat. An oven's user need not interact with internal parts of an oven.

Objects strongly support abstraction, hiding entire groups of functions and variables, exposing only certain functions to a user.

An ***abstract data type*** (***ADT***) is a data type whose creation and update are constrained to specific well-defined operations. A class can be used to implement an ADT.

PARTICIPATION ACTIVITY 12.1.4: Objects strongly support abstraction / information hiding.


Don't do this

Animation content:

On the left is a picture of an oven with its door closed. To the right is the oven with its door open and a woman reaching in as if to adjust the heat. Underneath this picture is a disclaimer saying "Don't do this."

Animation captions:

1. Abstraction simplifies our world. An oven is viewed as having a compartment for food, and a knob that can be turned to heat the food.
2. People need not be concerned with an oven's internal workings. Ex: People don't reach inside trying to adjust the flame.
3. Similarly, an object has operations that a user can apply. The object's internal data, and possibly other operations, are hidden from the user.

PARTICIPATION ACTIVITY 12.1.5: Abstraction / information hiding.

1) A car presents an abstraction to a user, including a steering wheel, gas pedal, and brake.
   - ○ True
   - ○ False

2) A refrigerator presents an abstraction to a user, including refrigerant gas, a compressor, and a fan.
   - ○ True
   - ○ False

3) A software object is created for a soccer team. A reasonable abstraction allows setting the team's name, adding or deleting players, and printing a roster.

- True
- False

4) A software object is created for a university class. A reasonable abstraction allows viewing and modifying variables for the teacher's name, and viewing variables for the list of students' names.

- True
- False

12.2 Using a class

Survey

The following questions are part of a zyBooks survey to help us improve our content so we can offer the best experience for students. The survey can be taken anonymously and takes just 3-5 minutes. Please take a short moment to answer by clicking the following link.

Link: Student survey

Classes intro: Public member functions

The ***class*** construct defines a new type that can group data and functions to form an object. A class' ***public member functions*** indicate all operations a class user can perform on the object. The power of classes is that a class user need not know how the class' data and functions are implemented, but need only understand how each public member function behaves. The animation below shows a class' public member function declarations only; the remainder of the class definition is discussed later.

PARTICIPATION ACTIVITY 12.2.1: A class example: Restaurant class.


```
class Restaurant {                          // Info about a restaurant
   public:
      void SetName(string restaurantName);  // Sets the restaurant's name
      void SetRating(int userRating);       // Sets the rating (1-5, with 5 best)
      void Print();                         // Prints name and rating on one line
```

```
    ...
};

Restaurant favPlace;
favPlace.SetName("Central Deli");
favPlace.SetRating(4);
favPlace.Print();
```

Animation content:

Static Figure:
Begin Java code:

```
public class Restaurant {                    // Info about a restaurant
  // Internal fields

  public void setName(String restaurantName) {   // Sets the restaurant's name
    ...
  }

  public void setRating(int userRating) {       // Sets the rating (1-5, with 5 best)
    ...
  }

  public void print() {                     // Prints name and rating on one line
    ...
  }
}
```

End Java code.
Java code lines, Restaurant favPlace = new Restaurant();, favPlace.setName("Central Deli");
, favPlace.setRating(4);, and favPlace.print(); are displayed. A box is shown representing an object favPlace. The box is labeled favPlace object and displays text, Name: Central Deli and Rating: 4. An output monitor displays text, "Central Deli -- 4".

Step 1: A class definition creates a new type that can be used to create objects. The class declares all methods a programmer can call to operate on such an object.

The Java Code appears showing the Restaurant Class.

Step 2: A class user can create an object by declaring and initializing a variable of the class type with new instance of the class type.

The Java code line, Restaurant favPlace = new Restaurant(); appears. A new empty Restraurant object, favPlace is created.

Step 3: Then, the class user can call the methods to operate on the object. A class user need not know how the class' fields or methods are implemented.

The Java code lines, favPlace.setName("Central Deli"); and favPlace.setRating(4); appear. "Central Deli" is assigned to the variable name and 4 is assigned to the rating of the favPlace object. The Java code line, favPlace.print(); appears. "Central Deli – 4" appears in the output monitor.

Animation captions:

1. A class definition creates a new type that can be used to create objects. The class declares all functions a programmer can call to operate on such an object.
2. A class user can declare a variable of the class type to create a new object.
3. Then, the class user can call the functions to operate on the object. A class user need not know how the class' data or functions are implemented.

PARTICIPATION ACTIVITY 12.2.2: Using a class.

Consider the example above.

1) Which operation can a class user perform on an object of type Restaurant?
   - Get the name
   - Set the name
   - Get the rating

2) Calling Print() on an object of type Restaurant might yield which output?
   - Marias -- 5
   - 5
Marias
   - Marias
5

3) Although not visible in the part of the class definition shown above, how many internal data variables does the class contain?
   - 1
   - 2
   - Unknown

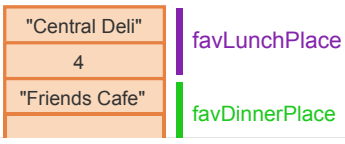
Using a class

A programmer can create one or more objects of the same class. Declaring a variable of a class type creates an ***object*** of that type. Ex: `Restaurant favLunchPlace;` declares a Restaurant object named favLunchPlace.

The "." operator, known as the ***member access operator***, is used to invoke a function on an object. Ex: `favLunchPlace.SetRating(4)` calls the SetRating() function on the favLunchPlace object, which sets the object's rating to 4.

PARTICIPATION ACTIVITY 12.2.3: Using the Restaurant class.

```
int main() {
   Restaurant favLunchPlace;
   Restaurant favDinnerPlace;

   favLunchPlace.SetName("Central Deli");
```



```
   favLunchPlace.SetRating(4);

   favDinnerPlace.SetName("Friends Cafe");
   favDinnerPlace.SetRating(5);

   cout << "My favorite restaurants: " << endl;
   favLunchPlace.Print();
   favDinnerPlace.Print();

   return 0;
}
```

5

My favorite restaurants:
Central Deli -- 4
Friends Cafe -- 5

Animation content:

Static Figure:
int main() {
  Restaurant favLunchPlace;
  Restaurant favDinnerPlace;

  favLunchPlace.SetName("Central Deli");
  favLunchPlace.SetRating(4);

  favDinnerPlace.SetName("Friends Cafe");
  favDinnerPlace.SetRating(5);

  cout << "My favorite restaurants: " <<endl;
  favLunchPlace.Print();
  favDinnerPlace.Print();

  return 0;
}
A graphical representation of the memory used by the program are displayed. There are four memory locations.

The first two memory locations are marked with purple and the text " favLunchPlace". The first memory location reads "Central Deli". The second memory location reads 4.

The next two memory locations are marked with green and the text "Friends Cafe". The third memory location reads "Friends Cafe". The fourth memory location reads 5.

Below the memory is an output monitor, which reads:
My favorite restaurants:
Central Deli -- 4
Friends Cafe -- 5

Step 1: A variable declared of the class type Restaurant can refer to a Restaurant object. The new operator creates a Restaurant object, allocating memory for the object. Each object may require numerous memory locations.

A highlight box highlights the C++ code line, Restaurant favLunchPlace = new Restaurant();. The favLunchPlace object is created, allocating 2 memory locations. The highlighted box highlights the C++ code line, Restaurant favDinnerPlace = new Restaurant();. The favDinnerPlace object is created, allocating 2 additional memory locations.

Step 2: The setName() and setRating() methods are invoked on the object favLunchPlace, setting that object's name to "Central Deli" and rating to 4. The object stores these values internally.

The highlighted box highlights the C++ code lines, favLunchPlace.setName("Central Deli"); and favLunchPlace.setRating(4);. The value "Centeral Deli" is set as the name variable and is added to the first memory address of the favLunchPlace object. The value 4 is set as the rating variable and is added to the second memory address of the favLunchPlace object.

Step 3: Invoking the setName() and setRating() methods on the favDinnerPlace object sets that object's name to "Friends Cafe" and rating to 5.

The highlighted box highlights the C++ code lines, favDinnerPlace.setName("Friends Cafe"); and favDinnerPlace.setRating(5);.The value "Friends Cafe" is set as the name variable and is added to the first memory address of the favDinnerPlace object. The value 5 is set as the rating variable and is added to the second memory address of the favDinnerPlace object.

Step 4: Invoking the print() operation on a Restaurant object, prints the restaurant's name and rating.

The highlighted box highlights the C++ code lines, System.out.println("My favorite restaurants: ");, favLunchPlace.print();, and favDinnerPlace.print();. The name and rating of Restraunt object favLunchPlace and favDinnerPlace are displayed on the output monitor. The text displayed on the monitor:
My favorite restaurants:
Central Deli -- 4
Friends Cafe – 5.

Animation captions:

1. Declaring a variable of the class type Restaurant creates an object of that type. The compiler allocates memory for the objects, each of which may require numerous memory locations.
2. The SetName() and SetRating() functions are invoked on the object favLunchPlace, setting that object's name to "Central Deli" and rating to 4. The object stores these values internally.
3. Invoking the SetName() and SetRating() method on the favDinnerPlace object sets that object's name to "Friends Cafe" and rating to 5.
4. Invoking the Print() operation on a Restaurant object, prints the restaurant's name and rating.

PARTICIPATION ACTIVITY 12.2.4: Using the Restaurant class.

The following questions consider *using* the Restaurant class.

1) Type a variable declaration that creates an object named favBreakfastPlace.

Check Show answer

2) Using separate variable declarations, create an object bestDessertPlace, followed by an object bestIndianFood.

Check Show answer

3) Given the code below, how many objects are created?

```
Restaurant bestIndianFood;
Restaurant bestSushi;
Restaurant bestCoffeeShop;
```

Check Show answer

4) Object bestSushi is of type Restaurant. Type a statement that sets the name of bestSushi to "Sushi Station".

Check Show answer

5) Type a statement to print bestCoffeeShop's name and rating.

Check Show answer

Class example: string

C++'s string type is a class. The string class stores a string's characters in memory, along with variables indicating the length and other things, but a string's user need not know such details. Instead, the string's user just needs to know what public member functions can be used, such as those shown below. (Note: size_t is an unsigned integer type).

Figure 12.2.1: Some string public member functions (many more exist).

```
char& at(size_t pos); // Returns a reference to the character at position
pos in the string.

size_t length() const; // Returns the number of characters in the string

void push_back(char c); // Appends character c to the string's end
(increasing length by 1).
```

PARTICIPATION ACTIVITY 12.2.5: Using the string class.

Consider the public member functions shown above for the string class.

1) Given string s = "Hi". How many bytes does object s utilize in memory?

- 2
- 3
- Unknown

2) Given string s = "Hi", how can a user append "!" to have s become "Hi!".

- s.push_back('!')
- s.at('!')
- Unknown

3) What enables a user to utilize the string class?

- Nothing; strings are built into C++
- #include <string>

12.3 Defining a class

Private data members

In addition to public member functions, a class definition has ***private data members***: variables that member functions can access but class users cannot. Private data members appear after the word "private:" in a class definition.

PARTICIPATION ACTIVITY 12.3.1: Private data members.

```
class Restaurant {                          // Keeps a user's review of a restaurant
   public:
      void SetName(string restaurantName);  // Sets the restaurant's name
      void SetRating(int userRating);       // Sets the rating (1-5, with 5 best)
      void Print();                         // Prints name and rating on one line
   private:
      string name;
      int rating;
};

int main() {
   Restaurant favLunchPlace;

   favLunchPlace.rating = 5;
   ...
}
```

Animation content:

Static figure: A code block is displayed.
Begin C++ code:

```
class Restaurant {                          // Keeps a user's review of a restaurant
  public:
    void SetName(string restaurantName);  // Sets the restaurant's name
    void SetRating(int userRating);       // Sets the rating (1-5, with 5 best)
    void Print();                         // Prints name and rating on one line
  private:
    string name;
    int rating;
};
```

```
int main() {
   Restaurant favLunchPlace;

   favLunchPlace.rating = 5;
   ...
}
```

End C++ code.

Step 1: A class definition has private data members for storing local data. The lines of code, private:, string name;, int rating;, are highlighted.

Step 2: A class user cannot access a class' private data members; only the class' member functions can. The line of code, favLunchPlace.rating = 5;, is struck-through.

Animation captions:

1. A class definition has private data members for storing local data.
2. A class user cannot access a class' private data members; only the class' member functions can.

PARTICIPATION ACTIVITY 12.3.2: Private data members.

Consider the example above.

1) After declaring Restaurant x, a class *user* can use a statement like x.name = "Sue's Diner".
   - True
   - False

2) After declaring a Restaurant object x, a class user can use a statement like myString = x.name.
   - True
   - False

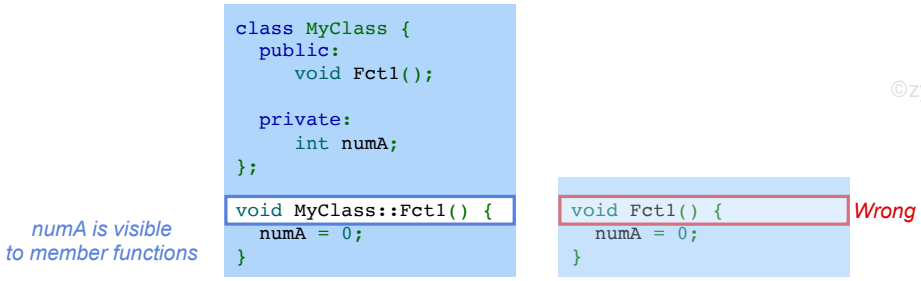
3) A class definition should provide comments along with each private data member so that a class user knows how those data members are used.
   - True
   - False

Defining a class' public member functions

A programmer defining a class first *declares* member functions after the word "public:" in the class definition. A ***function declaration*** provides the function's name, return type, and parameter types, but not the function's statements.

The programmer must also *define* each member function. A ***function definition*** provides a class name, return type, parameter names and types, and the function's statements. A member function definition has the class name and two colons (::), known as the ***scope resolution operator***, preceding the function's name. A member function definition can access private data members.

PARTICIPATION ACTIVITY | 12.3.3: Defining a member function of a class using the scope resolution operator.



Animation content:

Static figure: A code block is displayed.
Begin C++ code:

```
class MyClass {
 public:
   void Fct1();

 private:
   int numA;
};

void Fct1() {
 numA = 0;
}

void Fct1() {
 numA = 0;
}
```

End C++ code.
Step 1: Without the scope resolution operator, Fct1() will yield compiler errors: numA is undefined, MyClass' Fct1()'s definition is missing. The line of code, void Fct1() {, is highlighted and the text, Wrong, appears beside it.
Step 2: Using the scope resolution operator as in MyClass::Fct1() indicates Fct1() is a member function of MyClass. Private class data becomes visible. The lines of code, void Fct1() {, numA = 0;, }, moves next to the code block and is replaced with the following lines of code, void MyClass::Fct1() {, numA = 0;, }. The line of code, void MyClass::Fct1() {, is highlighted and the text, numA is visible to member functions, appears beside it.

Animation captions:

1. Without the scope resolution operator, Fct1() will yield compiler errors: numA is undefined, MyClass' Fct1()'s definition is missing.
2. Using the scope resolution operator as in MyClass::Fct1() indicates Fct1() is a member function of MyClass. Private class data becomes visible.

Figure 12.3.1: A complete class definition, and use of that class.

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

class Restaurant {                          // Info about
a restaurant
   public:
      void SetName(string restaurantName);  // Sets the
restaurant's name
      void SetRating(int userRating);       // Sets the
rating (1-5, with 5 best)
      void Print();                         // Prints name
and rating on one line

   private:
      string name;
      int rating;
};

// Sets the restaurant's name
void Restaurant::SetName(string restaurantName) {
   name = restaurantName;
}

// Sets the rating (1-5, with 5 best)
void Restaurant::SetRating(int userRating) {
   rating = userRating;
}

// Prints name and rating on one line
void Restaurant::Print() {
   cout << name << " -- " << rating << endl;
}

int main() {
   Restaurant favLunchPlace;
   Restaurant favDinnerPlace;

   favLunchPlace.SetName("Central Deli");
   favLunchPlace.SetRating(4);

   favDinnerPlace.SetName("Friends Cafe");
   favDinnerPlace.SetRating(5);

   cout << "My favorite restaurants: " << endl;
   favLunchPlace.Print();
   favDinnerPlace.Print();

   return 0;
}
```

```
My favorite
restaurants:
Central Deli -- 4
Friends Cafe -- 5
```


PARTICIPATION ACTIVITY 12.3.4: Class definition.

Consider the example above.

1) How is the Print() member function *declared*?

- `Print();`
- `void Print();`
- `void Restaurant::Print();`

2) How does the Print() member function's *definition* begin?

- `void Print();`
- `void Restaurant::Print();`
- `void Restaurant::Print()`

3) Could the Print() function's definition begin as follows?

```
void Restaurant::Print(int x)
```

- Yes
- No

4) Which private data members of class Restaurant do the Print() function definition's statements access?

- SetName
- favLunchPlace and favDinnerPlace
- name and rating

Example: RunnerInfo class

The RunnerInfo class below maintains information about a person who runs, allowing a class user to set the time run and the distance run, and to get the runner's speed. The subsequent question set asks for the missing parts to be completed.

Figure 12.3.2: Simple class example: RunnerInfo.

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

class RunnerInfo {
   public:
      void SetTime(int timeRunSecs);       // Time run in seconds
      void SetDist(double distRunMiles);   // Distance run in miles
      double GetSpeedMph();                // Speed in miles/hour
   __(A)__
      int timeRun;
      double distRun;
};

void __(B)__::SetTime(int timeRunSecs) {
   timeRun = timeRunSecs;  // timeRun refers to data member
}

void __(C)__SetDist(double distRunMiles) {
   distRun = distRunMiles;
}

double RunnerInfo::GetSpeedMph(){
   return distRun / (timeRun / 3600.0); // miles / (secs / (hrs / 3600
secs))
}

int main() {
   RunnerInfo runner1; // User-created object of class type RunnerInfo
   RunnerInfo runner2; // A second object

   runner1.SetTime(360);
   runner1.SetDist(1.2);

   runner2.SetTime(200);
   runner2.SetDist(0.5);

   cout << "Runner1's speed in MPH: " << runner1.__(D)__ << endl;
   cout << "Runner2's speed in MPH: " << __(E)__ << endl;

   return 0;
}
```

```
Runner1's speed in MPH: 12
Runner2's speed in MPH: 9
```


PARTICIPATION ACTIVITY 12.3.5: Class example: RunnerInfo.

Complete the missing parts of the figure above.

1) (A)

Check Show answer

2) (B)

Check Show answer

3) (C)

Check Show answer

4) (D)

Check Show answer

5) (E)

Check Show answer

Exploring further:

- Classes from cplusplus.com
- Classes from msdn.microsoft.com

CHALLENGE ACTIVITY 12.3.1: Classes.

539740.3879454.qx3zqy7

Start

Type the program's output

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

class Person {
   public:
      void SetName(string nameToSet);
      string GetName();
   private:
      string name;
};

void Person::SetName(string nameToSet) {
   name = nameToSet;
}

string Person::GetName() {
   return name;
}

int main() {
   string userName;
   Person person1;

   userName = "Joe";

   person1.SetName(userName);
   cout << "I am " << person1.GetName();

   return 0;
}
```

```
I am Joe
```

1 2 3 4

Check Next

CHALLENGE ACTIVITY 12.3.2: Basic class use.

Print person1's kids, apply the IncNumKids() function, and print again, outputting text as below. End each line with a newline.
Sample output for below program with input 3:

```
Kids: 3
New baby, kids now: 4
```

Learn how our autograder works

539740.3879454.qx3zqy7

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

class PersonInfo {
   public:
      void   SetNumKids(int personsKidsToSet);
      void   IncNumKids();
      int    GetNumKids();
   private:
      int    numKids;
};

void PersonInfo::SetNumKids(int personsKidsToSet) {
   numKids = personsKidsToSet;
}
```

Run

CHALLENGE ACTIVITY 12.3.3: Defining a class.

539740.3879454.qx3zqy7

Start

In the Shop class, complete the function definition for SetNumSold() with the integer parameter customNumSold.

Ex: If the input is `34 194 136`, then the output is:


```
Years open: 34
Items in stock: 194
Number of items sold: 136
```

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

class Shop {
   public:
```

```
6       void SetYearsOpen(int customYearsOpen);
7       void SetNumStock(int customNumStock);
8       void SetNumSold(int customNumSold);
9       int GetYearsOpen();
10      int GetNumStock();
11      int GetNumSold();
12    private:
13      int yearsOpen;
14      int numStock;
15      int numSold;
```


1 2

Check Next level

12.4 Inline member functions

Inline member functions

A member function's definition may appear within the class definition, known as an ***inline member function***. Programmers may use inline short function definitions to yield more compact code, keeping longer function definitions outside the class definition to avoid clutter.

PARTICIPATION ACTIVITY 12.4.1: Inline member functions.

```
class MyClass {
   public:
      void Fct1();
   private:
      int numA;
};

void MyClass::Fct1() {
   numA = 0;
}
```

```
class MyClass {
   public:
      void Fct1() {
         numA = 0;
      }
   private:
      int numA;
};
```

Inline member function

Animation content:

Static figure: A code block is displayed.
Begin C++ code:
class MyClass {
  public:
    void Fct1();
  private:
    int numA;
}
End C++ code.
Step 1: A member function's definition normally appears separate from the class definition,

associated with the class using the :: operator. The lines of code, void MyClass::Fct1() {, numA = 0;, }, appear at the bottom of the code block.
Step 2: Some programmers put a short member function's definition in the class definition, for compacted code. Care must be taken not to clutter the class definition. A copy of the code block is displayed and the lines of code, void MyClass::Fct1() {, numA = 0;, }, move to on top of the lines of code, public:, void Fct1();, replacing all of them with, public:, void Fct1() {, numA = 0;, }. The text, Inline member function, appears next to the new code.

Animation captions:

1. A member function's definition normally appears separate from the class definition, associated with the class using the :: operator.
2. Some programmers put a short member function's definition in the class definition, for compacted code. Care must be taken not to clutter the class definition.

Figure 12.4.1: A class with two inline member functions.

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

class Restaurant {                          // Info about
a restaurant
   public:
      void SetName(string restaurantName) { // Sets the
restaurant's name
         name = restaurantName;
      }
      void SetRating(int userRating) {      // Sets the
rating (1-5, with 5 best)
         rating = userRating;
      }
      void Print();                         // Prints name
and rating on one line

   private:
      string name;
      int rating;
};

// Prints name and rating on one line
void Restaurant::Print() {
   cout << name << " -- " << rating << endl;
}

int main() {
   Restaurant favLunchPlace;
   Restaurant favDinnerPlace;

   favLunchPlace.SetName("Central Deli");
   favLunchPlace.SetRating(4);

   favDinnerPlace.SetName("Friends Cafe");
   favDinnerPlace.SetRating(5);

   cout << "My favorite restaurants: " << endl;
   favLunchPlace.Print();
   favDinnerPlace.Print();

   return 0;
}
```

```
My favorite
restaurants:
Central Deli -- 4
Friends Cafe -- 5
```


PARTICIPATION ACTIVITY 12.4.2: Inline member functions.

Consider the example above.

1) Member function SetName() was defined ____.
   - inlined
   - not inlined

2) Inline member function SetRating() ____ a semicolon after the function name and parentheses, just like a function definition.
   - has
   - does not have

3) Member function Print() was _____.

- inlined
- not inlined

4) A function with a long definition likely ____ be inlined.

- should
- should not

5) A function defined as an inline member function _____ also have a definition outside the class as well.

- may
- may not

Exception to variables being declared before used

Normally, items like variables must be declared before being used, but this rule does not apply within a class definition. Ex: Above, SetRating() accesses rating, even though rating is declared a few lines after. This rule exception allows a class to have the desired form of a public region at the top and a private region at the bottom: A public inline member function can thus access a private data member even though that private data member is declared after the function.

PARTICIPATION ACTIVITY 12.4.3: Inline member functions.

Consider the following class definition.

```
class PickupTruck {
   public:
      void SetLength(double fullLength);
      void SetWidth (double fullWidth) {
         widthInches = fullWidth;
      }
   private:
      double lengthInches;
      double widthInches;
};

void PickupTruck::SetLength(double fullLength) {
   lengthInches = fullLength;
}
```


1) Inside the class definition, SetLength() is declared but not defined.

- True
- False

2) Inside the class definition, SetWidth() is declared but not defined.

- True
- False

3) SetWidth() is an inline member function.

- True
- False

4) SetWidth()'s use of widthInches is an error because widthInches is declared after that use.

- True
- False

5) If the programmer defines SetWidth() inline as above, then the programmer should probably define SetLength() as inline too.

- True
- False

Inline member functions on one line

Normally, good style dictates putting a function's statements below the function's name and indenting. But, many programmers make an exception by putting very-short inline member function statements on the same line, for improved readability. This material may use that style at times. Example:

```
...
      void SetName(string restaurantName) { name = restaurantName; }
      void SetRating(int userRating) { rating = userRating; }
...
```

CHALLENGE ACTIVITY 12.4.1: Inline member functions.

539740.3879454.qx3zqy7

Start

Type the program's output

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

class Course {
   public:
      void SetName(string courseName) {
         name = courseName;
      }
      void SetClassSize(int courseClassSize) {
         classSize = courseClassSize;
      }
      void Print() const;

   private:
      string name;
      int classSize;
};

void Course::Print() const {
   cout << name << ". Class size: " << classSize << endl;
}

int main() {
   Course myCourse;

   myCourse.SetName("Formal Logic");
   myCourse.SetClassSize(197);

   myCourse.Print();

   return 0;
}
```



```
Formal Logic. Class size: 19
```

1 2

Check Next

12.5 Mutators, accessors, and private helpers

Mutators and accessors

A class' public functions are commonly classified as either mutators or accessors.

- A ***mutator*** function may modify ("mutate") a class' data members.
- An ***accessor*** function accesses data members but does not modify a class' data members.

Commonly, a data member has two associated functions: a mutator for setting the value, and an accessor for getting the value, known as a ***setter*** and ***getter*** function, respectively, and typically with names starting with set or get. Other mutators and accessors may exist that aren't associated with just one data member, such as the Print() function below.

Accessor functions usually are defined as const to make clear that data members won't be changed. The keyword ***const*** after a member function's name and parameters causes a compiler error if the function modifies a data member. If a const member function calls another member function, that function must also be const.

Figure 12.5.1: Mutator and accessor public member functions.

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

class Restaurant {
   public:
      void   SetName(string restaurantName); // Mutator
      void   SetRating(int userRating);      // Mutator
      string GetName() const;                // Accessor
      int    GetRating() const;              // Accessor
      void   Print() const;                  // Accessor

   private:
      string name;
      int rating;
};

void Restaurant::SetName(string restaurantName) {
   name = restaurantName;
}

void Restaurant::SetRating(int userRating) {
   rating = userRating;
}

string Restaurant::GetName() const {
   return name;
}

int Restaurant::GetRating() const {
   return rating;
}

void Restaurant::Print() const {
   cout << name << " -- " << rating << endl;
}

int main() {
   Restaurant myPlace;

   myPlace.SetName("Maria's Diner");
   myPlace.SetRating(5);

   cout << myPlace.GetName() << " is rated ";
   cout << myPlace.GetRating() << endl;

   return 0;
}
```

```
Maria's Diner is rated 5
```


PARTICIPATION ACTIVITY 12.5.1: Mutators and accessors.

1) A mutator should not change a class' private data members.

   - ○ True
   - ○ False

2) An accessor should not change a class' private data members.

- True
- False

3) A private data member sometimes has a pair of associated set and get functions.

- True
- False

4) Accessor functions are required to be defined as const.

- True
- False

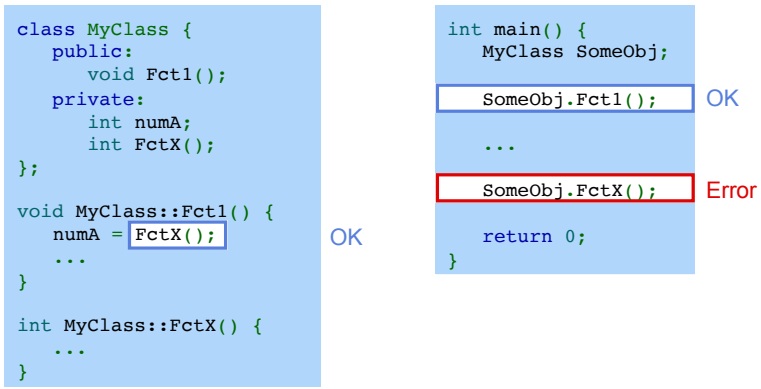
5) A const accessor function may call a non-const member function.

- True
- False

Private helper functions

A programmer commonly creates private functions, known as ***private helper functions***, to help public functions carry out tasks.

PARTICIPATION ACTIVITY 12.5.2: Private helper member functions.



Animation content:

Static figure: A code block is displayed.
Begin C++ code:
class MyClass {
public:
void Fct1();
private:
int numA;

```
    int FctX();
};
void MyClass::Fct1() {
  …
}

int MyClass::FctX() {
  ...
}
```


End C++ code.

Step 1: In addition to public member functions, a class may define private member functions. The lines of code, int FctX();, int MyClass::FctX() {, …, }, are highlighted.

Step 2: Any member function (public or private) may call a private member function. The lines of code, void MyClass::Fct1() {, …, }, are replaced with the code, void MyClass::Fct1() {, numA = FctX();, …, }. The code, FctX();, is highlighted and the text, OK, appears next to the highlighted code.

Step 3: A user of the class can call public member functions, but a user can not call private member functions (which would yield a compiler error). A new code block is displayed.

Begin C++ code:

```
int main() {
  MyClass SomeObj;

  SomeObj.Fct1();

  ...

  SomeObj.FctX();

  return 0;
}
```

End C++ code.

The line of code, SomeObj.Fct1();, is highlighted and the text, OK, appears next to the highlighted code. The line of code, SomeObj.FctX();, is highlighted and the text, Error, appears next to the highlighted code.

Animation captions:

1. In addition to public member functions, a class may define private member functions.
2. Any member function (public or private) may call a private member function.
3. A user of the class can call public member functions, but a user can not call private member functions (which would yield a compiler error).

PARTICIPATION ACTIVITY 12.5.3: Private helper functions.

1) A class' private helper function can be called from main().
   - ○ True
   - ○ False

2) A private helper function typically helps public functions carry out their tasks.

- True
- False

3) A private helper function cannot call another private helper function.

- True
- False

4) A public member function may not call another public member function.

- True
- False

CHALLENGE ACTIVITY 12.5.1: Mutators, accessors, and private helpers.

539740.3879454.qx3zqy7

Start

Type the program's output

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

class Dog {
   public:
      void SetAge(int monthsToSet);
      string GetStage() const;
   private:
      int months;
};

void Dog::SetAge(int monthsToSet) {
   months = monthsToSet;
}

string Dog::GetStage() const {
   string stage;
   if (months < 10) {
      stage = "Puppy";
   }
   else if (months < 13) {
      stage = "Adolescence";
   }
   else if (months < 80) {
      stage = "Adulthood";
   }
   else {
      stage = "Senior";
   }

   return stage;
}

int main() {
   Dog buddy;

   buddy.SetAge(18);

   cout << buddy.GetStage();
   return 0;
}
```

Adulthood

1 2 3 4

Check Next

CHALLENGE ACTIVITY 12.5.2: Mutators, accessors, and private helpers.

539740.3879454.qx3zqy7

Start

Define the Monday class's SetWeather() mutator that sets data member weather to mondayWeather and the SetHumidity() mutator that sets data member humidity to mondayHumidity.

Ex: If the input is `overcast 76.0`, then the output is:

```
Monday: overcast
Humidity: 76.0%
```

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

class Monday {
   public:
      void SetWeather(string mondayWeather);
      void SetHumidity(double mondayHumidity);
      void Print() const;
   private:
      string weather;
      double humidity;
};

```

1 2 3 4

Check Next level

12.6 Initialization and constructors

A good practice is to initialize all variables when declared. This section deals with initializing the data members of a class when a variable of the class type is declared.

Data member initialization (C++11)

Since C++11, a programmer can initialize data members in the class definition. Any variable declared of that class type will initially have those values.

Figure 12.6.1: A class definition with initialized data members.

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

class Restaurant {
   public:
      void SetName(string restaurantName);
      void SetRating(int userRating);
      void Print();

   private:
      string name = "NoName";  // NoName indicates name was not
set
      int rating = -1;         // -1 indicates rating was not
set
};

void Restaurant::SetName(string restaurantName) {
   name = restaurantName;
}

void Restaurant::SetRating(int userRating) {
   rating = userRating;
}

void Restaurant::Print() {
   cout << name << " -- " << rating << endl;
}

int main() {
   Restaurant favLunchPlace;  // Initializes members with
values in class definition

   favLunchPlace.Print();

   favLunchPlace.SetName("Central Deli");
   favLunchPlace.SetRating(4);

   favLunchPlace.Print();

   return 0;
}
```

```
NoName -- -1
Central Deli
-- 4
```


PARTICIPATION ACTIVITY 12.6.1: Initialization.

Consider the example above.

1) When favLunchPlace is initially declared, what is the value of favLunchPlace's rating?

Check Show answer

2) After the call to SetRating(), what is the value of favLunchPlace's rating?

Check Show answer

Constructors

C++ has a special class member function, a ***constructor***, called *automatically* when a variable of that class type is declared, and which can initialize data members. A constructor callable without arguments is a ***default constructor***, like the Restaurant constructor below.

A constructor has the same name as the class. A constructor function has no return type, not even void. Ex: `Restaurant::Restaurant() {...}` defines a constructor for the Restaurant class.

If a class has no programmer-defined constructor, then the compiler *implicitly* defines a default constructor having no statements.

Figure 12.6.2: Adding a constructor member function to the Restaurant class.

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

class Restaurant {
   public:
      Restaurant();
      void SetName(string restaurantName);
      void SetRating(int userRating);
      void Print();
   private:
      string name;
      int rating;
};

Restaurant::Restaurant() {  // Default constructor
   name = "NoName";         // Default name: NoName indicates name was not
set
   rating = -1;             // Default rating: -1 indicates rating was not
set
}

void Restaurant::SetName(string restaurantName) {
   name = restaurantName;
}

void Restaurant::SetRating(int userRating) {
   rating = userRating;
}

// Prints name and rating on one line
void Restaurant::Print() {
   cout << name << " -- " << rating << endl;
}

int main() {
   Restaurant favLunchPlace;  // Automatically calls the default constructor

   favLunchPlace.Print();

   favLunchPlace.SetName("Central Deli");
   favLunchPlace.SetRating(4);
   favLunchPlace.Print();

   return 0;
}
```

```
NoName -- -1
Central Deli -- 4
```


PARTICIPATION ACTIVITY 12.6.2: Default constructors.

Assume a class named Seat.

1) A default constructor declaration in class Seat { ... } is:

```
class Seat {
   ...
   void Seat();
}
```

- ○ True
- ○ False

2) A default constructor definition has this form:

```
Seat::Seat() {
   ...
}
```

- True
- False

3) Not defining any constructor is essentially the same as defining a constructor with no statements.

- True
- False

4) The following calls the default constructor once:

```
Seat mySeat;
```

- True
- False

5) The following calls the default constructor once:

```
Seat seat1;
Seat seat2;
```

- True
- False

6) The following calls the default constructor 5 times:

```
vector<Seat> seats(5);
```

- True
- False

Note: Since C++11, data members can be initialized in the class definition as in `int price = -1;`, which is usually preferred over using a constructor. However, sometimes initializations are more complicated, in which case a constructor is needed.

Exploring further:

- Constructors from msdn.microsoft.com

CHALLENGE ACTIVITY 12.6.1: Enter the output of classes.

539740.3879454.qx3zqy7

Start

Type the program's output

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

class Bicycle {
   public:
      void SetType(string bicycleType);
      void SetYear(int bicycleYear);
      void Print();
   private:
      string type = "NoType";  // NoType indicates brand was not set
      int year = -1;           // -1 indicates year was not set
};

void Bicycle::SetType(string bicycleType) {
   type = bicycleType;
}

void Bicycle::SetYear(int bicycleYear) {
   year = bicycleYear;
}

void Bicycle::Print() {
   cout << type << " " << year << endl;
}

int main() {
   Bicycle commuterBike;

   commuterBike.Print();

   commuterBike.SetType("tandem");

   commuterBike.Print();

   return 0;
}
```



```
NoType -1
tandem -1
```

1 2

Check Next

CHALLENGE ACTIVITY 12.6.2: Initialization and constructors.

539740.3879454.qx3zqy7

Start

In the class definition, initialize the data members, integer gasTank, char weekendPrice, and integer mileage, with the default values 0, 'N', and 0, respectively.

Ex: If the input is `75 Y 126000`, then the output is:

```
Gas tank: 0, Weekend price: N, Mileage: 0
Gas tank: 75, Weekend price: Y, Mileage: 126000
```

Note: The class's print function is called first after the default constructor, then again after the inputs are passed to the setter

```
1  #include <iostream>
2  using namespace std;
3
4  class RentalCar {
5     public:
6        void SetGasTank(int rentalCarGasTank);
7        void SetWeekendPrice(char rentalCarWeekendPrice);
8        void SetMileage(int rentalCarMileage);
9        void Print();
10
11    private:
12
```

```
13      /* Your code goes here */
14
15  };
```

1 2

Check Next level

12.7 Classes and vectors/classes

Vector of objects: A reviews program

Combining classes and vectors is powerful. The program below creates a Review class (reviews might be for a restaurant, movie, etc.), then manages a vector of Review objects.

Figure 12.7.1: Classes and vectors: A reviews program.

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

class Review {
   public:
      void SetRatingAndComment(int revRating, string
revComment) {
         rating = revRating;
         comment = revComment;
      }
      int GetRating() const { return rating; }
      string GetComment() const { return comment; }

   private:
      int rating = -1;
      string comment = "NoComment";
};

int main() {
   vector<Review> reviewList;
   Review currReview;
   int currRating;
   string currComment;
   unsigned int i;

   cout << "Type rating + comments. To end: -1" <<
endl;
   cin >> currRating;
   while (currRating >= 0) {
      getline(cin, currComment); // Gets rest of line
      currReview.SetRatingAndComment(currRating,
currComment);
      reviewList.push_back(currReview);
      cin >> currRating;
   }

   // Output all comments for given rating
   cout << endl << "Type rating. To end: -1" << endl;
   cin >> currRating;
   while (currRating != -1) {
      for (i = 0; i < reviewList.size(); ++i) {
         currReview = reviewList.at(i);
         if (currRating == currReview.GetRating()) {
            cout << currReview.GetComment() << endl;
         }
      }
      cin >> currRating;
   }

   return 0;
}
```

```
Type rating +
comments. To end: -1
5 Great place!
5 Loved the food.
2 Pretty bad service.
4 New owners are nice.
2 Yuk!!!
4 What a gem.
-1

Type rating. To end:
-1
5
 Great place!
 Loved the food.
1
4
 New owners are nice.
 What a gem.
-1
```


PARTICIPATION ACTIVITY 12.7.1: Reviews program.

Consider the reviews program above.

1) How many member functions does the Review class have?

Check Show answer

2) When currReview is declared, what is the initial rating?

Check Show answer

3) As rating and comment pairs are read from input, what function adds them to vector reviewList? Type the name only, like: append.

Check Show answer

4) How many comments were output for reviews having a rating of 5?

Check Show answer

A class with a vector: The Reviews class

A class' private data often involves vectors. The program below redoes the example above, creating a Reviews class for managing a vector of Review objects.

The Reviews class has functions for reading reviews and printing comments. The resulting main() is clearer than above.

The Reviews class has a "getter" function returning the average rating. The function computes the average rather than reading a private data member. The class user need not know how the function is implemented.

Figure 12.7.2: Improved reviews program with a Reviews class.

```
Type ratings +
comments. To end: -1
5 Great place!
5 Loved the food.
2 Pretty bad service.
4 New owners are nice.
2 Yuk!!!
4 What a gem.
-1

Average rating: 3

Type rating. To end:
-1
5
 Great place!
 Loved the food.
1
4
 New owners are nice.
 What a gem.
-1
```

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

class Review {
   public:
      void SetRatingAndComment(int revRating, string
revComment) {
         rating = revRating;
         comment = revComment;
      }
      int GetRating() const { return rating; }
      string GetComment() const { return comment; }

   private:
      int rating = -1;
      string comment = "NoComment";
};
// END Review class


class Reviews {
   public:
      void InputReviews();
      void PrintCommentsForRating(int currRating) const;
      int GetAverageRating() const;

   private:
      vector<Review> reviewList;
};

// Get rating comment pairs, add each to list. -1 rating
ends.
void Reviews::InputReviews() {
   Review currReview;
   int currRating;
   string currComment;

   cin >> currRating;
   while (currRating >= 0) {
      getline(cin, currComment); // Gets rest of line
      currReview.SetRatingAndComment(currRating,
currComment);
      reviewList.push_back(currReview);
      cin >> currRating;
   }
}

// Print all comments for reviews having the given
rating
void Reviews::PrintCommentsForRating(int currRating)
const {
   Review currReview;
   unsigned int i;

   for (i = 0; i < reviewList.size(); ++i) {
      currReview = reviewList.at(i);
      if (currRating == currReview.GetRating()) {
         cout << currReview.GetComment() << endl;
      }
   }
}

int Reviews::GetAverageRating() const {
   int ratingsSum;
   unsigned int i;

   ratingsSum = 0;
   for (i = 0; i < reviewList.size(); ++i) {
      ratingsSum += reviewList.at(i).GetRating();
   }
   return (ratingsSum / reviewList.size());
```

```
}
// END Reviews class

int main() {
   Reviews allReviews;
   string currName;
   int currRating;

   cout << "Type ratings + comments. To end: -1" <<
endl;
   allReviews.InputReviews();

   cout << endl << "Average rating: ";
   cout << allReviews.GetAverageRating() << endl;

   // Output all comments for given rating
   cout << endl << "Type rating. To end: -1" << endl;
   cin >> currRating;
   while (currRating != -1) {
      allReviews.PrintCommentsForRating(currRating);
      cin >> currRating;
   }

   return 0;
}
```


PARTICIPATION ACTIVITY 12.7.2: Reviews program.

Consider the reviews program above.

1) The first class is named Review. What is the second class named?
   - Reviews
   - reviewList
   - allReviews

2) How many private data members does the Reviews class have?
   - 0
   - 1
   - 2

3) Which function reads all reviews?
   - GetReviews()
   - InputReviews()

4) What does PrintCommentsForRating() do?
   - Prints reviews sorted by rating level.
   - Print all reviews above a rating level.
   - Print all reviews having a particular rating level.

5) Does main() declare a vector?

- Yes
- No

Using Reviews in the Restaurant class

Programmers commonly use classes within classes. The program below improves the Restaurant class by having a Reviews object rather than a single rating.

Figure 12.7.3: Improved reviews program with a Restaurant class.

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

// Review and Reviews classes omitted from figure
// ...


class Restaurant {
   public:
      void SetName(string restaurantName) {
         name = restaurantName;
      }
      void ReadAllReviews();
      void PrintCommentsByRating() const;

   private:
      string name;
      Reviews reviews;
};

void Restaurant::ReadAllReviews() {
   cout << "Type ratings + comments. To end: -1"
<< endl;
   reviews.InputReviews();
}

void Restaurant::PrintCommentsByRating() const {
   int i;

   cout << "Comments for each rating level: " <<
endl;
   for (i = 1; i <= 5; ++i) {
     cout << i << ":" << endl;
     reviews.PrintCommentsForRating(i);
   }
}

int main() {
   Restaurant ourPlace;
   string currName;

   cout << "Type restaurant name: " << endl;
   getline(cin, currName);
   ourPlace.SetName(currName);
   cout << endl;

   ourPlace.ReadAllReviews();
   cout << endl;

   ourPlace.PrintCommentsByRating();

   return 0;
}
```

```
Type restaurant name:
Maria's Healthy Food

Type ratings + comments. To
end: -1
5 Great place!
5 Loved the food.
2 Pretty bad service.
4 New owners are nice.
2 Yuk!!!
4 What a gem.
-1

Comments for each rating
level:
1:
2:
 Pretty bad service.
 Yuk!!!
3:
4:
 New owners are nice.
 What a gem.
5:
 Great place!
 Loved the food.
```


PARTICIPATION ACTIVITY 12.7.3: Restaurant program with reviews.

Consider the Restaurant program above.

1) How many private data members does the Restaurant class have?

- 0
- 1
- 2

2) Which Restaurant member function reads all reviews?

- GetReviews()
- InputReviews()
- ReadAllReviews()

3) What does PrintCommentsByRating() do?

- Prints comments sorted by rating level.
- Print all reviews having a particular rating level.

4) Does main() declare a Reviews object?

- Yes
- No

CHALLENGE ACTIVITY 12.7.1: Enter the output of classes and vectors.

539740.3879454.qx3zqy7

Start

Type the program's output

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

class Product {
   public:
      void SetPriceAndName(int productPrice, string productName) {
         price = productPrice;
         name = productName;
      };
      int GetPrice() const { return price; };
      string GetName() const { return name; };
   private:
      int price; // in dollars
      string name;
};

int main() {
   vector<Product> productList;
   Product currProduct;
   int currPrice;
   string currName;
   unsigned int i;
   Product resultProduct;

   cin >> currPrice;
   while (currPrice > 0) {
      cin >> currName;
      currProduct.SetPriceAndName(currPrice, currName);
      productList.push_back(currProduct);
      cin >> currPrice;
   }

   resultProduct = productList.at(0);
   for (i = 0; i < productList.size(); ++i) {
      if (productList.at(i).GetPrice() < resultProduct.GetPrice()) {
         resultProduct = productList.at(i);
      }
   }

   cout << resultProduct.GetName() << ": " << resultProduct.GetPrice() << endl;

   return 0;
}
```


Input

```
10 Cheese
8 Paper
7 Shirt
-1
```

Output

```
Shirt: 7
```

1 2 3

Check Next

CHALLENGE ACTIVITY 12.7.2: Writing vectors with classes.

539740.3879454.qx3zqy7

Start

Write code to assign x and y coordinates to currCoordinate, and store currCoordinate in locations. Input first receives an x value, then a y value. Input example: `12 32 88 2 -1 -1`

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  class Coordinate {
6     public:
7        void SetXAndY(int coordinateX, int coordinateY) {
8           x = coordinateX;
9           y = coordinateY;
10       }
11       void PrintCoordinate() const {
12          cout << x << " - " << y << endl;
13       }
```

```
14      int GetX() const { return x; }
15      int GetY() const { return y; }
16
17   private:
18      int x;
19      int y;
20 };
21
22 int main() {
23    vector<Coordinate> locations;
24    Coordinate currCoordinate;
25    int currX;
26    int currY;
27    unsigned int i;
28
29    cin >> currX;
30    cin >> currY;
31    while ((currX >= 0) && (currY >= 0)) {
32
33       /* Your code goes here */
34
35       cin >> currX;
```


1 2 3

Check Next

CHALLENGE ACTIVITY 12.7.3: Classes and vectors/classes.

539740.3879454.qx3zqy7

Start

The program first reads integer cityCount from input, representing the number of pairs of inputs to be read. Each pair has a s and a character, representing the city's name and rating, respectively. One City object is created for each pair and added to ve cityList. If a City object's rating is equal to 'A', call the City object's Print() function.

Ex: If the input is:

```
3
Sundance A Hanna B Worland A
```

then the output is:

```
City: Sundance, Rating: A
City: Worland, Rating: A
```

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  class City {
6     public:
7        void SetDetails(string newName, char newRating);
8        char GetRating() const;
9        void Print() const;
10    private:
11       string name;
12       char rating;
```

```
13 };
14
15 void City::SetDetails(string newName, char newRating) {
```

1 2 3 4

Check Next level

12.8 Separate files for classes

Two files per class

Programmers typically put all code for a class into two files, separate from other code.

- ***ClassName.h*** contains the class definition, including data members and member function declarations.
- ***ClassName.cpp*** contains member function definitions.

A file that uses the class, such as a main file or ClassName.cpp, must include ClassName.h. The .h file's contents are sufficient to allow compilation, as long as the corresponding .cpp file is eventually compiled into the program too.

The figure below shows how all the .cpp files might be listed when compiled into one program. Note that the .h file is not listed in the compilation command, due to being included by the appropriate .cpp files.

Figure 12.8.1: Using two separate files for a class.

StoreItem.h

```
#ifndef STOREITEM_H
#define STOREITEM_H

class StoreItem {
   public:
      void
SetWeightOunces(int ounces);
      void Print() const;
   private:
      int weightOunces;
};

#endif
```

StoreItem.cpp

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

#include "StoreItem.h"

void StoreItem::SetWeightOunces(int
ounces) {
   weightOunces = ounces;
}

void StoreItem::Print() const {
   cout << "Weight (ounces): " <<
weightOunces << endl;
}
```

main.cpp

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

#include "StoreItem.h"

int main() {
   StoreItem item1;

   item1.SetWeightOunces(16);
   item1.Print();

   return 0;
}
```

Compilation example

```
% g++ -Wall StoreItem.cpp main.cpp
% a.out
Weight (ounces): 16
```


Good practice for .cpp and .h files

Sometimes multiple small related classes are grouped into a single file to avoid a proliferation of files. But for typical classes, good practice is to create a unique .cpp and .h file for each class.

PARTICIPATION ACTIVITY 12.8.1: Separate files.

1) Commonly a class definition and associated function definitions are placed in a .h file.
   - True
   - False

2) The .cpp file for a class should #include the associated .h file.
   - True
   - False

3) A drawback of the separate file approach is longer compilation times.
   - True
   - False

Ex: Restaurant review classes

The restaurant review program, introduced in an earlier section, declared the Review, Reviews, and Restaurant classes in main.cpp. Each of the 3 classes should instead be implemented in .h/.cpp files, thus making for cleaner code in main.cpp.

Figure 12.8.2: .h and .cpp files for Review, Reviews, and Restaurant classes.

Review.h

```
#ifndef REVIEW_H
#define REVIEW_H

#include <string>

class Review {
   public:
      void
SetRatingAndComment(
         int revRating,
         std::string
revComment);
      int GetRating()
const;
      std::string
GetComment() const;

   private:
      int rating = -1;
      std::string comment =
"NoComment";
};

#endif
```

Review.cpp

```
#include "Review.h"
using namespace std;

void Review::SetRatingAndComment(int
revRating, string revComment) {
   rating = revRating;
   comment = revComment;
}

int Review::GetRating() const {
   return rating;
}

string Review::GetComment() const {
   return comment;
}
```

Reviews.h

```
#ifndef REVIEWS_H
#define REVIEWS_H

#include <vector>
#include "Review.h"

class Reviews {
   public:
      void InputReviews();
      void
PrintCommentsForRating(int
currRating) const;
      int
GetAverageRating() const;

   private:
      std::vector<Review>
reviewList;
};

#endif
```

Reviews.cpp

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

// Get rating comment pairs, add each to
list. -1 rating ends.
void Reviews::InputReviews() {
   Review currReview;
   int currRating;
   string currComment;

   cin >> currRating;
   while (currRating >= 0) {
      getline(cin, currComment); // Gets
rest of line

currReview.SetRatingAndComment(currRating,
currComment);
      reviewList.push_back(currReview);
      cin >> currRating;
   }
}

// Print all comments for reviews having
the given rating
void Reviews::PrintCommentsForRating(int
currRating) const {
   Review currReview;
   unsigned int i;

   for (i = 0; i < reviewList.size(); ++i)
{
      currReview = reviewList.at(i);
      if (currRating ==
currReview.GetRating()) {
         cout << currReview.GetComment() <<
endl;
      }
   }
}

int Reviews::GetAverageRating() const {
   int ratingsSum;
   unsigned int i;

   ratingsSum = 0;
   for (i = 0; i < reviewList.size(); ++i)
{
      ratingsSum +=
reviewList.at(i).GetRating();
   }
   return (ratingsSum / reviewList.size());
}
```

Restaurant.h

```
#ifndef RESTAURANT_H
#define RESTAURANT_H

#include <string>
#include "Reviews.h"

class Restaurant {
   public:
      void
SetName(std::string
restaurantName);
      void
ReadAllReviews();
      void
PrintCommentsByRating()
const;

   private:
      std::string name;
      Reviews reviews;
};

#endif
```

Restaurant.cpp

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

void Restaurant::SetName(string
restaurantName) {
   name = restaurantName;
}

void Restaurant::ReadAllReviews() {
   cout << "Type ratings + comments. To
end: -1" << endl;
   reviews.InputReviews();
}

void Restaurant::PrintCommentsByRating()
const {
   int i;

   cout << "Comments for each rating level:
" << endl;
   for (i = 1; i <= 5; ++i) {
      cout << i << ":" << endl;
      reviews.PrintCommentsForRating(i);
   }
}
```


PARTICIPATION ACTIVITY 12.8.2: Restaurant reviews program's main.cpp.

main.cpp

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

int main() {
   Restaurant ourPlace;
   string currName;

   cout << "Type restaurant name: " << endl;
   getline(cin, currName);
   ourPlace.SetName(currName);
   cout << endl;

   ourPlace.ReadAllReviews();
   cout << endl;

   ourPlace.PrintCommentsByRating();

   return 0;
}
```

Console:

```
Type restaurant name:
Maria's Healthy Food

Type ratings + comments. To end: -1
5 Great place!
5 Loved the food.
2 Pretty bad service.
4 New owners are nice.
2 Yuk!!!
4 What a gem.
-1

Comments for each rating level:
1:
2:
 Pretty bad service.
 Yuk!!!
3:
4:
 New owners are nice.
 What a gem.
5:
 Great place!
 Loved the food.
```


Animation content:

Static figure: A code block labeled main.cpp and an output console are displayed.
Begin C++ code:
#include <iostream>

```
#include "Restaurant.h"
using namespace std;

int main() {
   Restaurant ourPlace;
   string currName;

   cout << "Type restaurant name: " << endl;
   getline(cin, currName);
   ourPlace.SetName(currName);
   cout << endl;

   ourPlace.ReadAllReviews();
   cout << endl;

   ourPlace.PrintCommentsByRating();

   return 0;
}
```

End C++ code.

Step 1: The Review, Reviews, and Restaurant classes are included in main.cpp by including Restaurant.h. The line of code, #include "Restaurant.h", is highlighted.

Step 2: main()'s code is reasonably short, since reusable code resides in external files. The lines of code, Restaurant ourPlace;, string currName;, cout << "Type restaurant name: " << endl;, getline(cin, currName);, ourPlace.SetName(currName);, cout << endl;, ourPlace.ReadAllReviews();, cout << endl;, ourPlace.PrintCommentsByRating();, return 0;, are highlighted and the output console now contains the text:

```
Type restaurant name:
Maria's Healthy Food

Type ratings + comments. To end: -1
5 Great place!
5 Loved the food.
2 Pretty bad service.
4 New owners are nice.
2 Yuk!!!
4 What a gem.
-1

Comments for each rating level:
1:
2:
 Pretty bad service.
 Yuk!!!
3:
4:
 New owners are nice.
 What a gem.
5:
 Great place!
 Loved the food.
```


©zyBooks 01/31/24 17:52 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation captions:

1. The Review, Reviews, and Restaurant classes are included in main.cpp by including Restaurant.h.
2. main()'s code is reasonably short, since reusable code resides in external files.

PARTICIPATION ACTIVITY | 12.8.3: Restaurant review program .h and .cpp files.

If unable to drag and drop, refresh the page.

Restaurant.h **Reviews.cpp** **Review.h** **Restaurant.cpp** **Reviews.h** **Review.cpp**

	#includes the "Restaurant.h" header file.
	Uses cin and getline() statements to get ratings and comments from the user.
	Makes the Restaurant, Reviews, and Review classes available when being #included by another code file.
	Does not #include any of the 3 header files.
	#includes the <vector> header file.
	Implements class member functions, none of which use cin or cout.

Reset

CHALLENGE ACTIVITY | 12.8.1: Enter the output of separate files.

539740.3879454.qx3zqy7

Start

Type the program's output

main.cpp | Product.h | Product.cpp

Input

```
5 Berries
8 Paper
7 Belt
-1
```

Output

```
Berries: 5
```

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

int main() {
   vector<Product> productList;
   Product currProduct;
   int currPrice;
   string currName;
   unsigned int i;
   Product resultProduct;

   cin >> currPrice;
   while (currPrice > 0) {
      cin >> currName;
      currProduct.SetPriceAndName(currPrice, currName);
      productList.push_back(currProduct);
      cin >> currPrice;
   }

   resultProduct = productList.at(0);
   for (i = 0; i < productList.size(); ++i) {
      if (productList.at(i).GetPrice() < resultProduct.GetPrice()) {
         resultProduct = productList.at(i);
      }
   }

   cout << resultProduct.GetName() << ": " << resultProduct.GetPrice() << endl;

   return 0;
}
```


1 2 3

Check Next

12.9 Choosing classes to create

Decomposing into classes

Creating a program may start by a programmer deciding what "things" exist, and what each thing contains and does.

Below, the programmer wants to maintain a soccer team. The programmer realizes the team will have people, so decides to sketch a Person class. Each Person class will have private (shown by "-") data like name and age, and public (shown by "+") functions like get/set name, get/set age, and print. The programmer then sketches a Team class, which uses Person objects.

PARTICIPATION ACTIVITY 12.9.1: Creating a program by first sketching classes.

My program

Will have a soccer team

The team will have a head coach, assistant coach, list of players, name, etc.

Each coach and player will have a name, age, phone, etc.

I need a class for a "person" (coaches, players)

Person
-name : string -age : int
+get/set name

And for a "team"

Team
-head coach : Person -asst coach : Person
+get/set head coach

Animation content:

Programmer decides what "things" exist:
My program
Will have a soccer team
The team will have a head coach, assistant coach, list of players, name, etc.
Each coach and player will have a name, age, phone, etc.

Programmer sketches a Person class:
I need a class for a "person" (coaches, players)
Person
-name : string
-age : int
+get/set name
+get/set age
+print

Programmer sketches a Team class:
And for a "team"
Team
-head coach : Person
-asst coach : Person
+get/set head coach
+get/set asst coach
+print
More to come (list of players, name, etc.)

Animation captions:

1. A programmer thinks of what "things" a program may involve. The programmer decides one thing is a Team, and another thing is a Person.
2. The programmer sketches a Person class. Private items (shown by "-") are name and age. Public items (shown by "+") are getters/setters and print.
3. The programmer then sketches a Team class. Private items are head coach and asst coach, both of Person type. Public items are getters/setters and print.

PARTICIPATION ACTIVITY 12.9.2: Decomposing a program into classes.

Consider the example above.

1) Only one way exists to decompose a program into classes.
   - ○ True
   - ○ False

2) The - indicates a class' private item.

- True
- False

3) The + indicates additional private items.

- True
- False

4) The Team class uses the Person class.

- True
- False

5) The Person class uses the Team class.

- True
- False

Coding the classes

A programmer can convert the class sketches above into code. The programmer likely would first create and test the Person class, followed by the Team class.

Figure 12.9.1: SoccerTeam and TeamPerson classes.

TeamPerson.h

```
#ifndef TEAMPERSON_H
#define TEAMPERSON_H

#include <string>
using namespace std;

class TeamPerson {
   public:
      void   SetFullName(string
firstAndLastName);
      void   SetAgeYears(int ageInYears);
      string GetFullName() const;
      int    GetAgeYears() const;
      void   Print() const;

   private:
      string fullName;
      int    ageYears;
};

#endif
```

TeamPerson.cpp

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

#include "TeamPerson.h"

void TeamPerson::SetFullName(stri
firstAndLastName) {
   fullName = firstAndLastName;
}

void TeamPerson::SetAgeYears(int
ageInYears) {
   ageYears = ageInYears;
}

string TeamPerson::GetFullName() c
{
   return fullName;
}

int TeamPerson::GetAgeYears() cons
   return ageYears;
}

void TeamPerson::Print() const {
   cout << "Full name: "    << full
<< endl;
   cout << "Age (years): " << age
<< endl;
}
```

SoccerTeam.h

```
#ifndef SOCCERTEAM_H
#define SOCCERTEAM_H

#include "TeamPerson.h"

class SoccerTeam {
   public:
      void SetHeadCoach(TeamPerson
teamPerson);
      void SetAssistantCoach (TeamPerson
teamPerson);

      TeamPerson GetHeadCoach() const;
      TeamPerson GetAssistantCoach() const;

      void Print() const;

   private:
      TeamPerson headCoach;
      TeamPerson assistantCoach;
      // Players omitted for brevity
};

#endif
```

SoccerTeam.cpp

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

#include "SoccerTeam.h"

void SoccerTeam::SetHeadCoach(Tea
teamPerson) {
   headCoach = teamPerson;
}

void
SoccerTeam::SetAssistantCoach(Tea
teamPerson) {
   assistantCoach = teamPerson;
}

TeamPerson SoccerTeam::GetHeadCoa
const {
   return headCoach;
}

TeamPerson
SoccerTeam::GetAssistantCoach() c
   return assistantCoach;
}

void SoccerTeam::Print() const {
   cout << "HEAD COACH: " << endl
   headCoach.Print();
   cout << endl;

   cout << "ASSISTANT COACH: " <<
   assistantCoach.Print();
   cout << endl;
}
```


main.cpp

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

#include "SoccerTeam.h"
#include "TeamPerson.h"

int main() {
   SoccerTeam teamCalifornia;
   TeamPerson headCoach;
   TeamPerson asstCoach;

   headCoach.SetFullName("Mark Miwerds");
   headCoach.SetAgeYears(42);
   teamCalifornia.SetHeadCoach(headCoach);

   asstCoach.SetFullName("Stanley Lee");
   asstCoach.SetAgeYears(30);

teamCalifornia.SetAssistantCoach(asstCoach);

   teamCalifornia.Print();

   return 0;
}
```

```
HEAD COACH:
Full name: Mark Miwerds
Age (years): 42

ASSISTANT COACH:
Full name: Stanley Lee
Age (years): 30
```

PARTICIPATION ACTIVITY 12.9.3: Coding classes.

Consider the example above.

1) The programmer first sketched the desired classes, before writing the code seen above.
   - True
   - False

2) The programmer wrote one large file containing all the classes.
   - True
   - False

3) Good practice would be to first write the TeamPerson class and then test that class, followed by writing the SoccerTeam class and testing that class.
   - True
   - False

Included files

Above, note that each file only includes needed header files. SoccerTeam.h has a TeamPerson member so includes TeamPerson.h. SoccerTeam.cpp includes SoccerTeam.h. main.cpp declares objects of both types so also includes both .h files. *A common error is to include unnecessary .h files, which misleads the reader.*

Note that only .h files are included, never .cpp files.

PARTICIPATION ACTIVITY 12.9.4: Classes and includes.

Consider the earlier SoccerTeam and TeamPerson classes. Indicate which .h files should be included in each file.

1) TeamPerson.h
   - TeamPerson.h
   - SoccerTeam.h
   - No .h file needed

2) TeamPerson.cpp
   - TeamPerson.h
   - SoccerTeam.h
   - No .h file needed

3) SoccerTeam.h

- TeamPerson.h
- SoccerTeam.h
- No .h file needed

4) SoccerTeam.cpp

- TeamPerson.h
- SoccerTeam.h
- TeamPerson.cpp
- TeamPerson.h and SoccerTeam.h

5) main.cpp

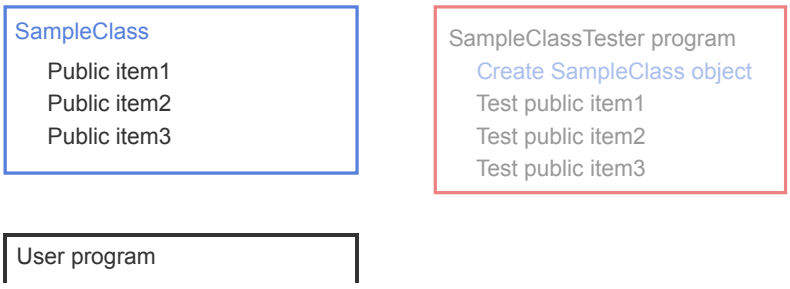
- main.h
- TeamPerson.h
- TeamPerson.h and SoccerTeam.h
- TeamPerson.cpp
- SoccerTeam.cpp

12.10 Unit testing (classes)

Testbenches

Like a chef who tastes food before serving, a class creator should test a class before allowing use. A ***testbench*** is a program whose job is to thoroughly test another program (or portion) via a series of input/output checks known as ***test cases***. ***Unit testing*** means to create and run a testbench for a specific item (or "unit") like a function or a class.

PARTICIPATION ACTIVITY 12.10.1: Unit testing of a class.



Create SampleClass object
Use public item 2

Animation content:

Three different programs are shown, each outlined with a box.

The first box contains the following:

SampleClass
Public item1
Public item2
Public item3

The second box contains the following:

User program
Create SampleClass object
Use public item 2

The third box contains the following:

SampleClassTester program
Create SampleClass object
Test public item1
Test public item2
Test public item3

Animation captions:

1. A typical program may not thoroughly use all class items.
2. A testbench's job is to thoroughly test all public class items.
3. After testing, class is ready for use. The tester program is kept for later tests.

The testbench below creates an object, then checks public functions for correctness. Some tests failed.

Features of a good testbench include:

- Automatic checks. Ex: Values are compared, as in `testData.GetNum1() != 100`. For conciseness, only fails are printed.
- Independent test cases. Ex: The test case for GetAverage() assigns new values, vs. relying on earlier values.
- ***100% code coverage***: Every line of code is executed. A good testbench would have more test cases than below.
- Includes not just typical values but also ***border cases***: Unusual or extreme test case values like 0, negative numbers, or large numbers.

Figure 12.10.1: Unit testing of a class.

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

// Note: This class intentionally has errors

class StatsInfo {
public:
   void SetNum1(int numVal) { num1 = numVal; }
   void SetNum2(int numVal) { num2 = numVal; }
   int GetNum1() const { return num1; }
   int GetNum2() const { return num1; }
   int GetAverage() const;

private:
   int num1;
   int num2;
};

int StatsInfo::GetAverage() const {
   return num1 + num2 / 2;
}
// END StatsInfo class


// TESTBENCH main() for StatsInfo class
int main() {
   StatsInfo testData;

   // Typical testbench tests more thoroughly

   cout << "Beginning tests." << endl;

   // Check set/get num1
   testData.SetNum1(100);
   if (testData.GetNum1() != 100) {
      cout << "   FAILED set/get num1" << endl;
   }

   // Check set/get num2
   testData.SetNum2(50);
   if (testData.GetNum2() != 50) {
      cout << "   FAILED set/get num2" << endl;
   }

   // Check GetAverage()
   testData.SetNum1(10);
   testData.SetNum2(20);
   if (testData.GetAverage() != 15) {
      cout << "   FAILED GetAverage for 10, 20" <<
endl;
   }

   testData.SetNum1(-10);
   testData.SetNum2(0);
   if (testData.GetAverage() != -5) {
      cout << "   FAILED GetAverage for -10, 0" <<
endl;
   }

   cout << "Tests complete." << endl;

   return 0;
}
```

```
Beginning tests.
   FAILED set/get num2
   FAILED GetAverage for
10, 20
   FAILED GetAverage for
-10, 0
Tests complete.
```


©zyBooks 01/31/24 17:52 1939727
Rob Daglio
MDCCOP2335Spring2024

Defining a testbench as a friend class (discussed elsewhere) enables direct testing of private member functions

PARTICIPATION ACTIVITY 12.10.2: Unit testing of a class.

1) A class should be tested individually (as a "unit") before use in another program.
   - True
   - False

2) Calling every function at least once is a prerequisite for 100% code coverage.
   - True
   - False

3) If a testbench achieves 100% code coverage and all tests passed, the class must be bug free.
   - True
   - False

4) A testbench should test all possible values, to ensure correctness.
   - True
   - False

5) A testbench should print a message for each test case that passes and for each that fails.
   - True
   - False

Regression testing

Regression testing means to retest an item like a class anytime that item is changed; if previously-passed test cases fail, the item has "regressed".

A testbench should be maintained along with the item, to always be usable for regression testing. A testbench may be in a class' file, or in a separate file as in MyClassTest.cpp for a class in MyClass.cpp.

Testbenches may be complex, with thousands of test cases. Various tools support testing, and companies employ *test engineers* who only test other programmers' items. A large percent, like 50% or more, of commercial software development time may go into testing.

PARTICIPATION ACTIVITY 12.10.3: Regression testing.

1) Testbenches are typically disposed of after use.
   - True
   - False

2) Regression testing means to check if a change to an item caused previously-passed test cases to fail.

- True
- False

3) For commercial software, testing consumes a large percentage of time.

- True
- False

Erroneous unit tests

An erroneous unit test may fail even if the code being tested is correct. *A common error is for a programmer to assume that a failing unit test means that the code being tested has a bug.* Such an assumption may lead the programmer to spend time trying to "fix" code that is already correct. *Good practice is to inspect the code of a failing unit test before making changes to the code being tested.*

Figure 12.10.2: Correct implementation of StatsInfo class.

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

class StatsInfo {
public:
    void SetNum1(int numVal) { num1 =
numVal; }
    void SetNum2(int numVal) { num2 =
numVal; }
    int GetNum1() const { return num1; }
    int GetNum2() const { return num2; }
    int GetAverage() const;

private:
    int num1;
    int num2;
};

int StatsInfo::GetAverage() const {
    return (num1 + num2) / 2;
}
```

PARTICIPATION ACTIVITY 12.10.4: Erroneous unit test code causes failures even when StatsInfo is correctly implemented.


```
StatsInfo testData;
testData.SetNum1(20);
testData.SetNum2(30);
if (testData.GetAverage() != 35) {
   cout << "   FAILED GetAverage for 20, 30" << endl;
}
```

Wrong expected value

```
StatsInfo testData;
testData.SetNum1(20);
```

```
testData.SetNum1(30);
if (testData.GetAverage() != 25) {
  cout << "   FAILED GetAverage for 20, 30" << endl;
}
```

Test object's data not properly set

Animation content:

Static figure: Two code blocks are displayed.
First code block:
Begin C++ code:
StatsInfo testData;
testData.SetNum1(20);
testData.SetNum2(30);
if (testData.GetAverage() != 35) {
  cout << "   FAILED GetAverage for 20, 30" << endl;
}
End C++ code.
Second code block:
Begin C++ code:
StatsInfo testData;
testData.SetNum1(20);
testData.SetNum1(30);
if (testData.GetAverage() != 25) {
  cout << "   FAILED GetAverage for 20, 30" << endl;
}
End C++ code.
Step 1: testData is instantiated and num1 and num2 are properly set to 20 and 30. In the first code block, the lines of code, StatsInfo testData;, testData.SetNum1(20);, testData.SetNum2(30);, are highlighted.
Step 2: Whether a typo or miscalculation, the unit test expects 35 instead of 25, and fails. A wrong expected value is one reason a unit test may fail. In the first code block, the line of code, if (testData.GetAverage() != 35) {, is highlighted and the text, 25 != 30, appears briefly above the code, testData.GetAverage() != 35. The line of code, cout << "   FAILED GetAverage for 20, 30" << endl;, is briefly highlighted. The line of code, if (testData.GetAverage() != 35) {, is highlighted with an arrow pointing to the text, Wrong expected value.
Step 3: Calling SetNum1 twice and not calling SetNum2 is also an error, even if the expected value is now correct. In the second code block, the line of code, testData.SetNum1(30);, is highlighted.
Step 4: Not properly initializing the test object's data is another common error. An arrow appears pointing the highlighted code in the second code block to the text, Test object's data not properly set.

Animation captions:

1. testData is instantiated and num1 and num2 are properly set to 20 and 30.
2. Whether a typo or miscalculation, the unit test expects 35 instead of 25, and fails. A wrong expected value is one reason a unit test may fail.
3. Calling SetNum1 twice and not calling SetNum2 is also an error, even if the expected value is now correct.
4. Not properly initializing the test object's data is another common error.

PARTICIPATION ACTIVITY 12.10.5: Identifying erroneous test cases.

Assume that StatsInfo is correctly implemented and identify each test case as valid or erroneous.

1) num1 = 1.5, num2 = 3.5, and the expected average = 2.5

- Valid
- Erroneous

2) num1 = 33, num2 = 11, and the expected average = 22

- Valid
- Erroneous

3) num1 = 101, num2 = 202, and the expected average = 152

- Valid
- Erroneous

Exploring further:

- C++ Unit testing frameworks from accu.org.

CHALLENGE ACTIVITY 12.10.1: Enter the output of the unit tests.

Note: There's always an error.

539740.3879454.qx3zqy7

Start

Type the program's output

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

class Rectangle {
public:
   void SetSize(int heightVal, int widthVal) {
      height = heightVal;
      width = widthVal;
   }
   int GetArea() const;
   int GetPerimeter() const;

private:
   int height;
   int width;
};

int Rectangle::GetArea() const {
   return height + width;
}

int Rectangle::GetPerimeter() const {
   return (height * 2) + (width * 2);
}

int main() {
   Rectangle myRectangle;

   myRectangle.SetSize(1, 1);
   if (myRectangle.GetArea() != 1) {
      cout << "FAILED GetArea() for 1, 1" << endl;
   }
   if (myRectangle.GetPerimeter() != 4) {
      cout << "FAILED GetPerimeter() for 1, 1" << endl;
   }

   myRectangle.SetSize(2, 3);
   if (myRectangle.GetArea() != 6) {
      cout << "FAILED GetArea() for 2, 3" << endl;
   }
   if (myRectangle.GetPerimeter() != 10) {
      cout << "FAILED GetPerimeter() for 2, 3" << endl;
   }

   return 0;
}
```



```
FAILED GetArea() for 1, 1
FAILED GetArea() for 2, 3
```

1 2 3

Check Next

CHALLENGE ACTIVITY 12.10.2: Unit testing of a class.

Write a unit test for addInventory(), which has an error. Call redSweater.addInventory() with argument sweaterShipment. Print the shown error if the subsequent quantity is incorrect. Sample output for failed unit test given initial quantity is 10 and sweaterShipment is 5:

```
Beginning tests.
   UNIT TEST FAILED: addInventory()
Tests complete.
```


Note: UNIT TEST FAILED is preceded by 3 spaces.

Learn how our autograder works

539740.3879454.qx3zqy7

```
1 #include <iostream>
2 using namespace std;
3
```

```
4  class InventoryTag {
5  public:
6     InventoryTag();
7     int getQuantityRemaining() const;
8     void addInventory(int numItems);
9
10 private:
11    int quantityRemaining;
12 };
13
14 InventoryTag::InventoryTag() {
15    quantityRemaining = 0;
```

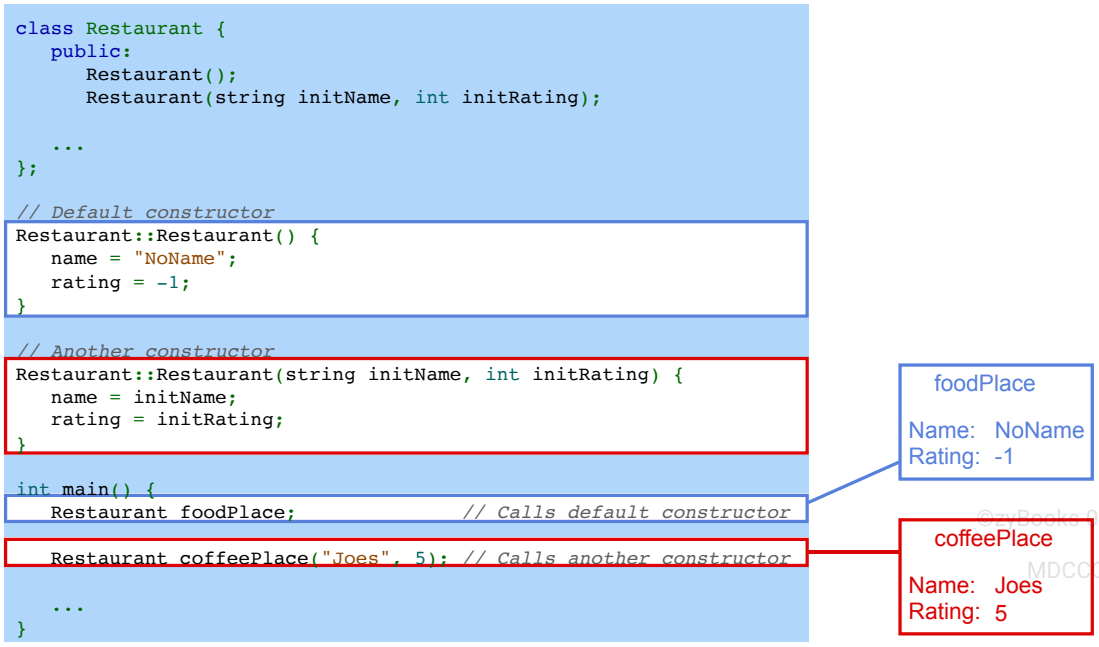

Run

12.11 Constructor overloading

Basics

Programmers often want to provide different initialization values when creating a new object. A class creator can ***overload*** a constructor by defining multiple constructors differing in parameter types. A constructor declaration can have arguments. The constructor with matching parameters will be called.

PARTICIPATION ACTIVITY | 12.11.1: Overloaded constructors.



Animation content:

Shows code having two constructors, one with no parameters, and one with two parameters. Then in main(), one declaration has no arguments, and another has two arguments.

Animation captions:

1. A declaration with no arguments calls the default constructor. In this case, the object gets initialized with NoName and -1.
2. This declaration's string and int arguments match another constructor, which is called instead. The object gets initialized with those argument values.

zyDE 12.11.1: Overloading a constructor.

Load default template... Run

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  class Restaurant {
6     public:
7        Restaurant();
8        Restaurant(string
9        void Print();
10
11    private:
12       string name;
13       int rating;
14 };
15
```

PARTICIPATION ACTIVITY | 12.11.2: Overloaded constructors.

Given the three constructors below, indicate which will be called for each declaration.

```
class SomeClass {
   SomeClass();                    // A
   SomeClass(string name);         // B
   SomeClass(string name, int num); // C
}
```

1) `SomeClass myObj("Lee");`

- A
- B
- C
- Error

2) `SomeClass myObj();`

- A
- B
- Error

3) `SomeClass myObj;`

- A
- B
- Error

4) `SomeClass myObj("Lee", 5, 0);`

- C
- Error

5) `vector<SomeClass> myVect(5);`

- A
- Error

If any constructor defined, should define default

If a programmer defines any constructor, the compiler does not implicitly define a default constructor, so good practice is for the programmer to also explicitly define a default constructor so that a declaration like `MyClass x;` remains supported.

Figure 12.11.1: Error - The programmer defined a constructor, so the compiler does not automatically define a default constructor.

```
class Restaurant {
   public:
      Restaurant(string initName,
int initRating);

      // No other constructors
      ...
};

int main() {
   Restaurant foodPlace;
   ...
}
```

```
tmp1.cpp:37:15: error: no matching
constructor for initialization of
      'Restaurant'
   Restaurant foodPlace;
```

PARTICIPATION ACTIVITY 12.11.3: Constructor definitions.

Which of the following is OK as the entire set of constructors for class MyClass? Assume a declaration like `MyClass x;` should be supported.

1) `MyClass();`

- OK
- Error

2) `// None`

- OK
- Error

3) `MyClass();`
`MyClass(string name);`

- OK
- Error

4) `MyClass(string name);`

- OK
- Error

Constructors with default parameter values

Like any function, a constructor's parameters may be assigned default values.

If those default values allow the constructor to be called without arguments, then that constructor can serve as the default constructor.

The default values could be in the function definition, but are clearer to class users in the declaration.

Figure 12.11.2: A constructor with default parameter values can serve as the default constructor.

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

class Restaurant {
   public:
      Restaurant(string initName = "NoName", int initRating =
-1);
      void Print();

    private:
      string name;
      int rating;
};

Restaurant::Restaurant(string initName, int initRating) {
   name = initName;
   rating = initRating;
}

// Prints name and rating on one line
void Restaurant::Print() {
   cout << name << " -- " << rating << endl;
}

int main() {
   Restaurant foodPlace;
   Restaurant coffeePlace("Joes", 5);

   foodPlace.Print();
   coffeePlace.Print();

   return 0;
}
```

```
NoName --
-1
Joes -- 5
```

PARTICIPATION ACTIVITY | 12.11.4: Constructor with default parameter values may serve as default constructor.

Which of the following is OK as the entire set of constructors for class YourClass? Assume a declaration like `YourClass obj;` should be supported.

1)
```
YourClass();
```
- OK
- Error

2)
```
YourClass(string name, int num);
```
- OK
- Error

3)
```
YourClass(string name = "", int
num = 0);
```
- OK
- Error

4)
```
YourClass();
YourClass(string name = "", int
num = 0);
```
- OK
- Error

CHALLENGE ACTIVITY | 12.11.1: Enter the output of the constructor overloading.

539740.3879454.qx3zqy7

Start

Type the program's output

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

class Pet {
   public:
      Pet();
      Pet(string petName, int yearsOld);
      void Print();

   private:
      string name;
      int age;
};

Pet::Pet() {
   name = "Unnamed";
   age = -9999;
}

Pet::Pet(string petName, int yearsOld) {
   name = petName;
   age = yearsOld;
}

void Pet::Print() {
   cout << name << ", " << age << endl;
}

int main() {
   Pet dog;
   Pet cat("Leo", 8);

   dog.Print();
   cat.Print();

   return 0;
}
```



```
Unnamed, -9999
Leo, 8
```

1 2 3 4

Check Next

CHALLENGE ACTIVITY 12.11.2: Constructor overloading.

539740.3879454.qx3zqy7

Start

The Play class has a default constructor, a constructor with two parameters, and a constructor with three parameters. Decla following objects:

- play1 with no arguments
- play2 with playTitle and playAuthor as arguments
- play3 with playTitle, playAuthor, and playYear as arguments

Ex: If the input is `Parrot Cole 1904`, then the output is:

```
Play: Unspecified, Unknown, 0
Play: Parrot, Cole, 0
Play: Parrot, Cole, 1904
```



```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 class Play {
6    public:
7       Play();
```

```
8       Play(string playTitle, string playAuthor);
9       Play(string playTitle, string playAuthor, int playYear);
10      void Print();
11
12   private:
13      string title;
14      string author;
15      int year;
```

1 2 3

Check Next level

12.12 Constructor initializer lists

A ***constructor initializer list*** is an alternative approach for initializing data members in a constructor, coming after a colon and consisting of a comma-separated list of variableName(initValue) items.

Figure 12.12.1: Member initialization: (left) Using statements in the constructor, (right) Using a constructor initializer list.

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

class SampleClass {
   public:
      SampleClass();
      void Print() const;

   private:
      int field1;
      int field2;
};

SampleClass::SampleClass() {
   field1 = 100;
   field2 = 200;
}

void SampleClass::Print() const
{
   cout << "Field1: " << field1
<< endl;
   cout << "Field2: " << field2
<< endl;
}

int main() {
   SampleClass myClass;
   myClass.Print();
   return 0;
}
```

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

class SampleClass {
   public:
      SampleClass();
      void Print() const;

   private:
      int field1;
      int field2;
};

SampleClass::SampleClass() :
field1(100), field2(200) {
}



void SampleClass::Print() const {
   cout << "Field1: " << field1 <<
endl;
   cout << "Field2: " << field2 <<
endl;
}

int main() {
   SampleClass myClass;
   myClass.Print();
   return 0;
}
```

`Field1: 100` `Field2: 200`	`Field1: 100` `Field2: 200`

PARTICIPATION ACTIVITY 12.12.1: Member initialization.

1) Convert this constructor to use a constructor initializer list.

```
MyClass::MyClass() {
   x = -1;
   y = 0;
}
```

```
MyClass::MyClass()
{
}
```

Check Show answer

The approach is important when a data member is a class type that must be explicitly constructed. Otherwise, that data member is by default constructed. Ex: If you have studied vectors, consider a data member consisting of a vector of size 2.

Figure 12.12.2: Member initialization in a constructor.

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

class SampleClass {
   public:
      SampleClass();
      void Print() const;

   private:
      vector<int> itemList;
};

SampleClass::SampleClass() {
   // itemList gets default
constructed, size 0
   itemList.resize(2);
}

void SampleClass::Print() const {
   cout << "Size: " <<
itemList.size() << endl;
   cout << "Item1: " <<
itemList.at(0) << endl;
   cout << "Item2: " <<
itemList.at(1) << endl;
}

int main() {
   SampleClass myClass;
   myClass.Print();
   return 0;
}
```

```
Size: 2
Item1: 0
Item2: 0
```

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

class SampleClass {
   public:
      SampleClass();
      void Print() const;

   private:
      vector<int> itemList;
};

SampleClass::SampleClass() :
itemList(2) {
   // itemList gets constructed
with size 2
}

void SampleClass::Print() const {
   cout << "Size: " <<
itemList.size() << endl;
   cout << "Item1: " <<
itemList.at(0) << endl;
   cout << "Item2: " <<
itemList.at(1) << endl;
}

int main() {
   SampleClass myClass;
   myClass.Print();
   return 0;
}
```

```
Size: 2
Item1: 0
Item2: 0
```


On the left, the constructor initially creates a vector of size 0, then resizes to size 2, where each element has the value 0. On the right, itemList(2) is provided in the SampleClass constructor initialization list, causing the vector constructor to be called with size 2 and each vector element to be initialized with the value 0. Using the initialization list avoids the inefficiency of constructing and then modifying an item.

Note: Since C++11, the data member could have been initialized in the class definition: `vector<int> itemList(2);`. However, initialization lists are still useful for other cases.

PARTICIPATION ACTIVITY 12.12.2: Constructor initializer list.

Consider the example above.

1) On the left, itemList is first constructed with size 0, then resized to size 2.
   - ○ True
   - ○ False

2) On the right, itemList is first constructed with size 0, then resized to size 2.

- True
- False

CHALLENGE ACTIVITY 12.12.1: Enter the output of constructor initializer lists.

539740.3879454.qx3zqy7

Start

Type the program's output

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

class Tutor {
   public:
      Tutor();
      void Print() const;

   private:
      string name;
      string topic;
};

Tutor::Tutor() : name("Unnamed"), topic("NoTopic") {}

void Tutor::Print() const {
   cout << name << " teaches " << topic << endl;
}

int main() {
   Tutor myTutor;

   myTutor.Print();

   return 0;
}
```

```
Unnamed teaches NoTopic
```

1 2 3 4 5

Check Next

CHALLENGE ACTIVITY 12.12.2: Constructor initializer lists.

539740.3879454.qx3zqy7

Start

Add a constructor initializer list to the default Applicant constructor to initialize shortName with "Unknown", experience with -1, and partTime with 'X'.

Ex: If the input is `Aya 10 Y`, then the output is:

```
Applicant: Unknown, Experience: -999, Part Time: X
Applicant: Aya, Experience: 10, Part Time: Y
```

```
1 #include <iostream>
2 using namespace std;
3
```

```
4  class Applicant {
5     public:
6        Applicant();
7        void SetData(string newShortName, int newExperience, char newPartTime);
8        void Print() const;
9     private:
10       string shortName;
11       int experience;
12       char partTime;
13 };
14
15 Applicant::Applicant() /* Your code goes here */ {
```


1 2 3

Check Next level

Exploring further:

- Classes from cplusplus.com, see "Member initialization in constructors" section.
- Constructors from msdn.microsoft.com, see "Member lists".

12.13 The 'this' implicit parameter

Implicit parameter

An object's member function is called using the syntax `object.Function()`. The object variable before the function name is known as an ***implicit parameter*** of the member function because the compiler converts the call syntax `object.Function(...)` into a function call with a pointer to the object implicitly passed as a parameter. Ex: `Function(object, ...)`.

Within a member function, the implicitly-passed object pointer is accessible via the name ***this***. In particular, a member can be accessed as `this->member`. The -> is the member access operator for a pointer, similar to the "." operator for non-pointers.

Using `this->` makes clear that a class member is being accessed and is essential if a data member and parameter have the same identifier. In the example below, `this->` is necessary to differentiate between the data member sideLength and the parameter sideLength.

Figure 12.13.1: Using 'this' to refer to an object's member.

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

class ShapeSquare {
   public:
      void   SetSideLength(double sideLength);
      double GetArea() const;
   private:
      double sideLength;
};

void ShapeSquare::SetSideLength(double sideLength) {
   this->sideLength = sideLength;
   // Data member      Parameter
}

double ShapeSquare::GetArea() const{
   return sideLength * sideLength; // Both refer to data
member
}

int main() {
   ShapeSquare square1;

   square1.SetSideLength(1.2);
   cout << "Square's area: " << square1.GetArea() << endl;

   return 0;
}
```

```
Square's area:
1.44
```


PARTICIPATION ACTIVITY 12.13.1: The 'this' implicit parameter.

Given a class Spaceship with private data member numYears and public member function:
`void Spaceship::AddNumYears(int numYears)`

1) In AddNumYears(), which line assigns the data member numYears with 0?

   - `numYears = 0;`
   - `this.numYears = 0;`
   - `this->numYears = 0;`

2) In AddNumYears(), which line assigns the data member numYears with the parameter numYears?

   - `numYears = this->numYears;`
   - `this->numYears = numYears;`

3) In AddNumYears(), which line adds the parameter numYears to the existing value of data member numYears?

- `this->numYears = this->numYears + numYears;`
- `this->numYears = numYears + numYears;`
- `numYears = this->numYears + numYears;`

4) Given variable `Spaceship ss1` is declared in main(), which line assigns ss1's numYears with 5?

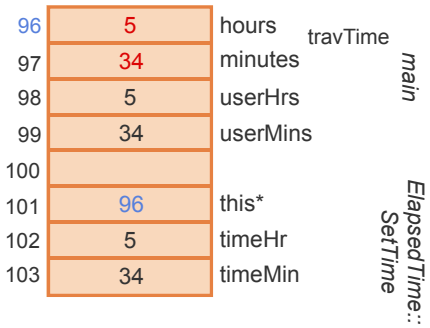
- `ss1.numYears = 5;`
- `ss1->numYears = 5;`
- `this->numYears = 5;`
- None of the above.

Using 'this' in class member functions and constructors

The animation below illustrates how member functions work. When an object's member function is called, the object's memory address is passed to the function via the implicit "this" parameter. An access in SetTime() to `this->hours` first goes to the object's address, then to the hours data member. If SetTime() instead had the assignment hours = timeHr, the compiler would use `this->hours` for hours because no other variable in SetTime() is named hours.

PARTICIPATION ACTIVITY 12.13.2: How a member function works.

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

class ElapsedTime {
   private:
      int hours;
      int minutes;

   public:
      void SetTime(int timeHr, int timeMin);
};

void ElapsedTime::SetTime(int timeHr, int timeMin) {
   this->hours = timeHr;
   this->minutes = timeMin;
}

int main() {
   ElapsedTime travTime;
   int userHrs;
   int userMins;

   userHrs = 5;
   userMins = 34;

   travTime.SetTime(userHrs, userMins);

   return 0;
}
```



Animation content:

Static Figure:
Begin C++ code:

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

class ElapsedTime {
  private:
    int hours;
    int minutes;

  public:
    void SetTime(int timeHr, int timeMin);
};

void ElapsedTime::SetTime(int timeHr, int timeMin) {
  this->hours = timeHr;
  this->minutes = timeMin;
}

int main() {
  ElapsedTime travTime;
  int userHrs;
  int userMins;

  userHrs = 5;
  userMins = 34;

  travTime.SetTime(userHrs, userMins);

  return 0;
}
```

End C++ code.
A box containing 8 memory addresses from 96 to 103, consecutively. The memory address 96 is labeled hours and contains “5”. The memory address 97 is labeled minutes and contains “35”. The memory address 98 is labeled userHrs and contains “5”. The memory address 99 is labeled userMins and contains “35”. Memory addresses 96 to 99 are labeled main and represents the object travTime.

The memory address 100 is not labeled and is empty. The memory address 101 is labeled this* and contains “96”. The memory address 101 is labeled this* and contains “96”. The memory address 102 is labeled timeHr and contains “5”. The memory address 103 is labeled timeMin and contains “34”. Memory addresses 96 to 99 are labeled ElapsedTime:: SetTime.

Step 1: travTime is an object of class type ElapsedTime.

A highlight box highlights the C++ code lines, ElapsedTime travTime;, int userHrs;, and int userMins;. The traveTime object is created, allocating four memory addresses 96 to 103 for variables hours, minutes, userHrs, and userMins.
The highlight box highlights C++ code lines userHrs = 5; and userMins = 34;. The variable userHrs is initialized with the value 5 in memory address 98. The variable userMins is initialized to 34 in memory address 99.

Step 2: When travTime's SetTime() member function is called, travTime's memory address is passed to the function via the implicit "this" parameter.

The highlight box highlights the C++ line travTime.SetTime(userHrs, userMins);. The SetTime function is called and another highlight box highlights the C++ code line void ElapsedTime::SetTime(int timeHr, int timeMin) {. 3 memory addresses 101 to 103 are allocated for the SetTime function variables. The first memory address 101 contains 96 which points to the beginning of the tracTime memory address. The memory address 102 contains the passed userHrs value, 5. The memory address 103 contains the passed userMins value, 34.

Step 3: The implicitly-passed object pointer is accessible within the member function via the name "this". Ex: this->hours first goes to travTime's address, then to the hours data member.

The highlight box highlights the C++ code lines this->hours = timeHr; and this->minutes = timeMin;. A copy of the value 5 from memory address 102 is added the memory address 96. A copy of the value 34 from memory address 103 is added the memory address 97.

Animation captions:

1. travTime is an object of class type ElapsedTime.
2. When travTime's SetTime() member function is called, travTime's memory address is passed to the function via the implicit "this" parameter.
3. The implicitly-passed object pointer is accessible within the member function via the name "this". Ex: this->hours first goes to travTime's address, then to the hours data member.

PARTICIPATION ACTIVITY | 12.13.3: Using the 'this' pointer in member functions and constructors.

1) Complete the code to assign the value of mins to data member minutes using `this->` notation.

```
void
ElapsedTime::SetMinutes(int
mins) {
    [ ] =
mins;
}
```

Check Show answer

2) Complete the code to assign the value of parameter hours to data member hours using `this->` notation.


```
void ElapsedTime::SetHours(int
hours) {
    [ ];
}
```

Check Show answer

Exploring further:

- The 'this' pointer from msdn.microsoft.com.

CHALLENGE ACTIVITY 12.13.1: Enter the output of the function.

539740.3879454.qx3zqy7

Start

Type the program's output

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

class Airplane {
   public:
      Airplane();
      void Print() const;
      void SetSpeed(int speed);
   private:
      int speed;
};

Airplane::Airplane() {
   speed = 0;
}

void Airplane::SetSpeed(int speed) {
   this->speed = speed;
}

void Airplane::Print() const {
   cout << speed << " MPH" << endl;
}

int main() {
   Airplane boeing747;

   boeing747.SetSpeed(600);
   boeing747.Print();

   return 0;
}
```

```
600 MPH
```

1 2 3

Check Next

CHALLENGE ACTIVITY 12.13.2: The this implicit parameter.

Define the missing member function. Use "this" to distinguish the local member from the parameter name.

Learn how our autograder works

539740.3879454.qx3zqy7

```
1 #include <iostream>
2 using namespace std;
3
4 class CablePlan{
```

```
   public:
      void  SetNumDays(int numDays);
      int   GetNumDays() const;
   private:
      int   numDays;
};

// FIXME: Define SetNumDays() member function, using "this" implicit parame
void CablePlan::SetNumDays(int numDays) {

   /* Your solution goes here  */
```


Run

12.14 Operator overloading

Overview

C++ allows a programmer to redefine the functionality of built-in operators like +, -, and *, to operate on programmer-defined objects, a process known as ***operator overloading***. Suppose a class TimeHrMn has data members hours and minutes. Overloading + would allow two TimeHrMn objects to be added with the + operator.

PARTICIPATION ACTIVITY 12.14.1: Operator overloading allows use of operators like + on classes.

Without operator overloading

```
TimeHrMn time1(3, 22);
TimeHrMn time2(2, 50);
TimeHrMn timeTot;
timeTot.hours = time1.hours + time2.hours;
timeTot.minutes = time1.minutes + time2.minutes;

timeTot.Print();
```

Console:

```
H:5, M:72
```

With operator overloading

```
TimeHrMn time1(3, 22);
TimeHrMn time2(2, 50);
TimeHrMn timeTot;
timeTot = time1 + time2;

timeTot.Print();
```

Console:

```
H:5, M:72
```

Animation content:

Static figure: A code block labeled, Without operator overloading, and a console are displayed.
Begin C++ code:
TimeHrMn time1(3, 22);
TimeHrMn time2(2, 50);
TimeHrMn timeTot;
timeTot.hours = time1.hours + time2.hours;
timeTot.minutes = time1.minutes + time2.minutes;

timeTot.Print();
End C++ code.
Step 1: timeTot is initialized to the sum of time1 and time2 by adding the hours and minutes fields separately. The lines of code, timeTot.hours = time1.hours + time2.hours;, timeTot.minutes =

time1.minutes + time2.minutes;, are highlighted. The line of code, timeTot.Print();, is highlighted and the text, H:5, M:72, is output to the console.
Step 2: Overloading the + operator in the TimeHrMn class allows the time1 and time2 objects to be added directly. A code block labeled, With operator overloading, and a console are displayed.
Begin C++ code:

```
TimeHrMn time1(3, 22);
TimeHrMn time2(2, 50);
TimeHrMn timeTot;
timeTot = time1 + time2;

timeTot.Print();
```

End C++ code.
The line of code, timeTot = time1 + time2;, is highlighted.
Step 3: The same result is achieved with simpler, more readable code. The line of code, timeTot.Print();, is highlighted and the text, H:5, M:72, is output to the console.

Animation captions:

1. timeTot is initialized to the sum of time1 and time2 by adding the hours and minutes fields separately.
2. Overloading the + operator in the TimeHrMn class allows the time1 and time2 objects to be added directly.
3. The same result is achieved with simpler, more readable code.

PARTICIPATION ACTIVITY 12.14.2: Operator overloading.

Refer to the example above.

1) The expression `time1 + time2` results in a compiler error if the + operator is not overloaded inside the TimeHrMn class.
   - True
   - False

2) The expressions `time1 + time2` and `time2 + time1` are expected to produce the same result.
   - True
   - False

Overloading TimeHrMn's + operator

To overload +, the programmer creates a member function named operator+. Although + requires left and right operands as in `time1 + time2`, the member function only requires the right operand (rhs: right-hand-side) as the parameter, because the left operand is the calling object. In other words, `time1 + time2` is equivalent to the function call `time1.operator+(time2)`, which is valid syntax but almost never used.

Figure 12.14.1: TimeHrMn class implementation with overloaded + operator.

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

class TimeHrMn {
public:
   TimeHrMn(int timeHours = 0, int timeMinutes = 0);
   void Print() const;
   TimeHrMn operator+(TimeHrMn rhs) ;
private:
   int hours;
   int minutes;
};

// Overload + operator for TimeHrMn
TimeHrMn TimeHrMn::operator+(TimeHrMn rhs) {
   TimeHrMn timeTotal;

   timeTotal.hours   = hours   + rhs.hours;
   timeTotal.minutes = minutes + rhs.minutes;

   return timeTotal;
}

TimeHrMn::TimeHrMn(int timeHours, int timeMinutes) {
   hours  = timeHours;
   minutes = timeMinutes;
}

void TimeHrMn::Print() const {
   cout << "H:" << hours << ", " << "M:" << minutes <<
endl;
}
```


PARTICIPATION ACTIVITY | 12.14.3: TimeHrMn::operator+ is called when two TimeHrMn objects are added with the + operator.

```
#include "TimeHrMn.h"

int main() {
  TimeHrMn time1(3, 22);
  TimeHrMn time2(2, 50);
  TimeHrMn sumTime;

  sumTime = time1 + time2;
  sumTime.Print();

  return 0;
}
```

Console:

```
H:5, M:72
```

```
TimeHrMn TimeHrMn::operator+(TimeHrMn rhs) {
  TimeHrMn timeTotal;

  timeTotal.hours   = hours   + rhs.hours;
  timeTotal.minutes = minutes + rhs.minutes;

  return timeTotal;
}
```

time1:

hours	3
minutes	22

time2:

hours	2
minutes	50

sumTime:

hours	5
minutes	72

Animation content:

Static figure: Two code blocks and a console are displayed.
First code block:
Begin C++ code:
```
#include "TimeHrMn.h"

int main() {
  TimeHrMn time1(3, 22);
  TimeHrMn time2(2, 50);
  TimeHrMn sumTime;

  sumTime = time1 + time2;
  sumTime.Print();

  return 0;
}
```
End C++ code.

Second code block:
Begin C++ code:
```
TimeHrMn TimeHrMn::operator+(TimeHrMn rhs) {
  TimeHrMn timeTotal;

  timeTotal.hours   = hours   + rhs.hours;
  timeTotal.minutes = minutes + rhs.minutes;

  return timeTotal;
}
```
End C++ code.
Step 1: time1, time2, and sumTime are initialized. sumTime initially has 0 hours and 0 minutes. The lines of code, TimeHrMn time1(3, 22);, TimeHrMn time2(2, 50);, TimeHrMn sumTime;, are highlighted and three tables labeled time1, time2, and sumTime are displayed.
time1:
hours 3
minutes 22
time2:
hours 2
minutes 50
sumTime:
hours 0
minutes 0
Step 2: The expression time1 + time2 results in TimeHrMn's operator+ member function being called. The line of code, sumTime = time1 + time2;, is highlighted. The lines of code, TimeHrMn TimeHrMn::operator+(TimeHrMn rhs) {, TimeHrMn timeTotal;, are highlighted and a table labeled timeTotal is displayed.
timeTotal:
hours
minutes
Step 3: In the expression hours + rhs.hours, hours is time1's hours, and rhs.hours is time2's hours. The line of code, timeTotal.hours   = hours   + rhs.hours;, is highlighted and 5 is added next to hours in the timeTotal table.
Step 4: The line of code, timeTotal.minutes = minutes + rhs.minutes;, is highlighted and 72 is added next to minutes in the timeTotal table. The line of code, return timeTotal;, is highlighted and the timeTotal table disappears and the sumTime table is updated with the values from timeTotal. The

line of code, sumTime.Print();, is highlighted and the text, H:5, M:72, is output to the console. The line of code, return 0;, is highlight and the program ends.

Animation captions:

1. time1, time2, and sumTime are initialized. sumTime initially has 0 hours and 0 minutes.
2. The expression time1 + time2 results in TimeHrMn's operator+ member function being called.
3. In the expression hours + rhs.hours, hours is time1's hours, and rhs.hours is time2's hours.
4. The minutes are computed similarly. The combined time is returned and displayed.

PARTICIPATION ACTIVITY 12.14.4: Operator overloading basics.

1) Given `TimeHrMn time1(10, 0)` and `TimeHrMn time2(3, 5)`, and the above overloading of +, what is sumTime.hours after `sumTime = time1 + time2`?

Check Show answer

2) Write the start of a TimeHrMn member function definition that overloads the subtraction (-) operator, naming the parameter rhs.

```
// Overloaded '-' function
definition

{
   /* Implementation */
}
```

Check Show answer

3) Which parameter should be removed from this line, that strives to overload the * operator? Type the parameter name only; don't list the type.

```
TimeHrMn
TimeHrMn::operator*
(TimeHrMn lhs, TimeHrMn
rhs) {
```

Check Show answer

Overloading the + operator multiple times

When an operator like + has been overloaded, the compiler determines which + operation to invoke based on the operand types. In `4 + 9`, the compiler sees two integer operands and thus applies the built-in + operation. In `time1 + time2`, where time1 and time2 are TimeHrMn objects, the compiler sees two TimeHrMn operands and thus invokes the programmer-defined function.

A programmer can define several functions that overload the same operator, as long as each involves different types so that the compiler can determine which to invoke. The code below overloads the + operator twice in the TimeHrMn class.

main() uses the + operator in 4 statements. The first + involves two TimeHrMn operands, so the compiler invokes the first operator+ function ("A"). The second + involves TimeHrMn and int operands, so the compiler invokes the second operator+ function ("B"). The third + involves two int operands, so the compiler invokes the built-in + operation. The fourth +, commented out, involves an int and TimeHrMn operands. Because no function has those operands ("B" has TimeHrMn and int, not int and TimeHrMn; order matters), that statement would generate a compiler error.

Figure 12.14.2: Overloading the + operator multiple times.

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

class TimeHrMn {
public:
   TimeHrMn(int timeHours = 0, int timeMinutes = 0);
   void Print() const;
   TimeHrMn operator+(TimeHrMn rhs);
   TimeHrMn operator+(int rhsHours);
private:
   int hours;
   int minutes;
};

// Operands: TimeHrMn, TimeHrMn. Call this "A"
TimeHrMn TimeHrMn::operator+(TimeHrMn rhs) {
   TimeHrMn timeTotal;

   timeTotal.hours   = hours   + rhs.hours;
   timeTotal.minutes = minutes + rhs.minutes;

   return timeTotal;
}

// Operands: TimeHrMn, int. Call this "B"
TimeHrMn TimeHrMn::operator+(int rhsHours) {
   TimeHrMn timeTotal;

   timeTotal.hours = hours + rhsHours;
   timeTotal.minutes = minutes; // Stays same

   return timeTotal;
}

TimeHrMn::TimeHrMn(int timeHours, int timeMinutes) {
   hours  = timeHours;
   minutes = timeMinutes;

   return;
}

void TimeHrMn::Print() const {
   cout << "H:" << hours << ", " << "M:" << minutes <<
endl;
}

int main() {
   TimeHrMn time1(3, 22);
   TimeHrMn time2(2, 50);
   TimeHrMn sumTime;
   int num;

   num = 91;

   sumTime = time1 + time2; // Invokes "A"
   sumTime.Print();

   sumTime = time1 + 10;    // Invokes "B"
   sumTime.Print();

   cout << num + 8 << endl; // Invokes built-in add

   // sumTime = 10 + time1; // ERROR: No (int, TimeHrMn)

   return 0;
}
```

```
H:5, M:72
H:13,
M:22
99
```

PARTICIPATION ACTIVITY 12.14.5: Determining which function is invoked.

Given:

```
Course course1;
Course course2;
int num1;
int num2;
```

If unable to drag and drop, refresh the page.

course1 + course2; **num2 + course2;** **num1 + num2;** **course1 + num1;**

	Error
	Course Course::operator+(int val) {
	Course Course::operator+(Course rhs) {
	Built-in + operation

Reset

CHALLENGE ACTIVITY 12.14.1: Enter the output of operator overloading.

539740.3879454.qx3zqy7

Start

Type the program's output

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

class InchSize {
   public:
      InchSize(int wholeInches = 0, int sixteenths = 0);
      void Print() const;
      InchSize operator+(InchSize rhs);
   private:
      int inches;
      int sixteenths;
};

InchSize InchSize::operator+(InchSize rhs) {
   InchSize totalSize;

   totalSize.inches = inches + rhs.inches;
   totalSize.sixteenths = sixteenths + rhs.sixteenths;

   // If sixteenths is greater than an inch, carry 1 to inches.
   if (totalSize.sixteenths >= 16) {
      totalSize.inches += 1;
      totalSize.sixteenths -= 16;
   }

   return totalSize;
}

InchSize::InchSize(int wholeInches, int sixteenthsOfInch) {
   inches = wholeInches;
   sixteenths = sixteenthsOfInch;
}

void InchSize::Print() const {
   cout << inches << " " << sixteenths << "/16 inches" << endl;
}

int main() {
   InchSize size1(4, 10);
   InchSize size2(3, 12);
   InchSize sumSize;

   sumSize = size1 + size2;

   sumSize.Print();

   return 0;
}
```



```
8 6/16 inches
```

1 2 3

Check Next

CHALLENGE ACTIVITY 12.14.2: Operator overloading.

539740.3879454.qx3zqy7

Start

Four doubles are read from input, where the first two doubles are the kilometers and meters of distance1 and the second two doubles are the kilometers and meters of distance2. Complete the function to overload the + operator.

Ex: If the input is 13.0 313.0 7.0 6.0, then the output is:

```
13 kilometers, 313 meters
7 kilometers, 6 meters
Sum: 20 kilometers, 319 meters
```

Note: The sum of two distances is:

- the sum of the number of kilometers
- the sum of the number of meters

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

class Distance {
   public:
      Distance(double km = 0.0, double m = 0.0);
      void Print() const;
      Distance operator+(Distance rhs);
   private:
      double kilometers;
      double meters;
};

Distance::Distance(double km, double m) {
   kilometers = km;
```


1 2 3

Check Next level

Exploring further:

- Overloadable operators from cplusplus.com. Provides a list of operators that can be overloaded, including a description of how to declare overloaded operator functions for operators with different operands like += and ++.

12.15 Overloading comparison operators

Overloading the equality (==) operator

A programmer can overload the equality operator (==) to allow comparing objects of a programmer-defined class for equality. To overload ==, the programmer creates a function named `operator==` that returns bool and takes two const reference arguments of the class type for the left-hand-side and right-hand-side operands. Ex: To overload the == operator for a Review class, the programmer defines a function `bool operator==(const Review& lhs, const Review& rhs)`.

The programmer must also determine when two objects are considered equal. In the Review class below, two Review objects are equal if the objects have the same rating and comment.

PARTICIPATION ACTIVITY 12.15.1: Overloading the == operator.

```
// ...            true



if (myReview == bestReview) {
   cout << "Must be my favorite" << endl;
}
```

90 "Great" comment

Animation content:

Static figure: A code block and a memory box containing six memory addresses, labeled from 85 to 90, consecutively are displayed. Memory location 85, labeled rating, contains a five and memory location 86, labeled comment, contains the text “Great”. Memory location 85 and 86 have the text, myReview, next to them. Memory location 89, labeled rating, contains a five and memory location 90, labeled comment, contains the text “Great”. Memory location 89 and 90 have the text, bestReview, next to them.
Begin C++ code:

```
bool operator==(const Review& lhs, const Review& rhs) {
  return (lhs.GetRating() == rhs.GetRating()) &&
         (lhs.GetComment() == rhs.GetComment());
}



// ...



if (myReview == bestReview) {
  cout << "Must be my favorite" << endl;
}
```

End C++ code.
Step 1: myReview is the left operand of the == operator and is passed as the first argument. bestReview is the right operand and is passed as the second argument. The line of code, if (myReview == bestReview) {, is highlighted. The line of code, bool operator==(const Review& lhs, const Review& rhs) {, is highlighted and the text myReview appears above the code, const Review& lhs and the text, bestReview, appears above the code, const Review& rhs.
Step 2: The operator== function returns true if both operands have the same rating and comment. myReview and bestReview both have a rating of 5 and a comment of "Great", so the operator returns true. The line of code, return (lhs.GetRating() == rhs.GetRating()) &&, (lhs.GetComment() == rhs.GetComment());, is highlighted and the contents of the memory box appear in the text:
(5 == 5) &&
("Great" == "Great")
true && true
true,
below the highlighted code. The line of code, if (myReview == bestReview) {, is highlighted and the text, true, is briefly displayed above the code, myReview == bestReview. The line of code, cout << "Must be my favorite" << endl;, is highlighted.

Animation captions:

1. myReview is the left operand of the == operator and is passed as the first argument. bestReview is the right operand and is passed as the second argument.

2. The operator== function returns true if both operands have the same rating and comment. myReview and bestReview both have a rating of 5 and a comment of "Great", so the operator returns true.

PARTICIPATION ACTIVITY 12.15.2: Overloading == operator for Restaurant class.

Given a Restaurant class, which of the following are valid function signatures for overloading the equality (==) operator?

1)
```
operator==(const Restaurant&
lhs, const Restaurant& rhs)
```

- Valid
- Invalid

2)
```
bool operator==(const Restaurant
lhs, const Restaurant rhs)
```

- Valid
- Invalid

3)
```
bool operator==(const
Restaurant& lhs, const string&
rhs)
```

- Valid
- Invalid

Overloading the < operator

A programmer can also overload relational operators like the less than operator (<). The < operator should return true if the object on the left side of the < operator is less than the object on the right side of the operator. In the Review class below, the `operator<` function returns true if the left Review operand has a lower rating than the right Review operand.

Figure 12.15.1: Overloading the Reviews class' < operator.

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

class Review {
public:
   void SetRatingAndComment(int revRating, string revComment) {
      rating = revRating;
      comment = revComment;
   }
   int GetRating() const { return rating; }
   string GetComment() const { return comment; }

private:
   int rating = -1;
   string comment = "NoComment";
};

// Equality (==) operator for two Review objects
bool operator==(const Review& lhs, const Review& rhs) {
   return (lhs.GetRating() == rhs.GetRating()) &&
          (lhs.GetComment() == rhs.GetComment());
}

// Less-than (<) operator for two Review objects
bool operator<(const Review& lhs, const Review& rhs) {
   return lhs.GetRating() < rhs.GetRating();
}

int main() {
   vector<Review> reviewList;
   Review currentReview;
   Review lowestReview;
   int currentRating;
   string currentComment;
   int i;

   cout << "Type rating + comments. To end: -1" << endl;
   cin >> currentRating;
   while (currentRating >= 0) {
      getline(cin, currentComment); // Gets rest of line
      currentReview.SetRatingAndComment(currentRating,
currentComment);
      reviewList.push_back(currentReview);
      cin >> currentRating;
   }

   // Find and output lowest review
   lowestReview = reviewList.at(0);
   for (i = 1; i < reviewList.size(); ++i) {
      if (reviewList.at(i) < lowestReview ) {
         lowestReview = reviewList.at(i);
      }
   }

   cout << endl;
   cout << lowestReview.GetRating() << " "
        << lowestReview.GetComment() << endl;

   return 0;
}
```


©zyBooks 01/31/24 17:52 1939727
Rob Daglio
MDCCOP2335Spring2024

```
Type rating + comments. To end: -1
5 Great place!
5 Loved the food.
2 Pretty bad service.
4 New owners are nice.
2 Yuk!!!
4 What a gem.
-1

2  Pretty bad service.
```


PARTICIPATION ACTIVITY 12.15.3: Overloading the < operator.

Given the Review class above, complete the definition for the overloaded < operator for the given comparison types. Use const reference parameters, and name the operands lhs and rhs.

1) Review < int

```
bool operator<(const Review&
lhs,                        ) {
   return lhs.GetRating() <
rhs;
}
```

Check Show answer

2) int < Review

```
bool operator<(const int& lhs,
const Review& rhs) {
   return lhs <
                            ;
}
```

Check Show answer

3) Review < double

```

{
   return lhs.GetRating() < rhs;
}
```

Check Show answer

Overloading all equality and relational operators

A common approach is to first overload the == and < operators and then overload other comparison operators using == and <.

- Overloading != using ==:

```
bool operator!=(const Review& lhs, const Review& rhs) { return !(lhs == rhs); }
```

- Overloading >, <=, and >= using <:

```
bool operator>(const Review& lhs, const Review& rhs)  { return rhs < lhs;    }
```

```
bool operator<=(const Review& lhs, const Review& rhs) { return !(lhs > rhs); }
bool operator>=(const Review& lhs, const Review& rhs) { return !(lhs < rhs); }
```

PARTICIPATION ACTIVITY 12.15.4: Overloading comparison operators.

Given two Review objects named userReview and bestReview, which overloaded operators are called for the following comparisons?

1) `userReview != bestReview`
   - operator==
   - operator!=
   - operator!= and operator==

2) `userReview > bestReview`
   - operator<
   - operator>
   - operator> and operator<

3) `userReview <= bestReview`
   - operator<= and operator==
   - operator<=, operator>, and operator==;
   - operator<=, operator>, and operator<

Sorting a vector

The ***sort()*** function, defined in the C++ Standard Template Library's (STL) algorithms library, can sort vectors containing objects of programmer-defined classes. To use sort(), a programmer must:

1. Add `#include <algorithm>` to enable the use of sort().
2. Overload the < operator for the programmer-defined class.
3. Call the sort() function as `sort(myVector.begin(), myVector.end())`

Figure 12.15.2: Sorting a vector of Review objects.

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

class Review {
public:
   void SetRatingAndComment(int revRating, string revComment) {
      rating = revRating;
      comment = revComment;
   }
   int GetRating() const { return rating; }
   string GetComment() const { return comment; }

private:
   int rating = -1;
   string comment = "NoComment";
};

// Less-than (<) operator for two Review objects
bool operator<(const Review& lhs, const Review& rhs) {
   return lhs.GetRating() < rhs.GetRating();
}

int main() {
   vector<Review> reviewList;
   Review currentReview;
   int currentRating;
   string currentComment;
   int i;

   cout << "Type rating + comments. To end: -1" << endl;
   cin >> currentRating;
   while (currentRating >= 0) {
      getline(cin, currentComment); // Gets rest of line
      currentReview.SetRatingAndComment(currentRating,
currentComment);
      reviewList.push_back(currentReview);
      cin >> currentRating;
   }

   // Sort reviews from lowest to highest
   sort(reviewList.begin(), reviewList.end());

   cout << endl;
   for (i = 0; i < reviewList.size(); ++i) {
      cout << reviewList.at(i).GetRating() << ": "
           << reviewList.at(i).GetComment() << endl;
   }

   return 0;
}
```

```
Type rating + comments. To end: -1
5 Great place!
5 Loved the food.
2 Pretty bad service.
4 New owners are nice.
2 Yuk!!!
4 What a gem.
-1

2:  Pretty bad service.
2:  Yuk!!!
4:  New owners are nice.
4:  What a gem.
5:  Great place!
5:  Loved the food.
```

PARTICIPATION ACTIVITY 12.15.5: Sorting vectors.

1) Sorting a vector of integers requires overloading the less than operator for two int operands.
   - True
   - False

2) If a vector contains duplicate elements, a programmer must overload both the less than operator and the equality operator.
   - True
   - False

3) The sort() function can be used to sort a range of elements within a vector instead of the entire vector.
   - True
   - False

CHALLENGE ACTIVITY 12.15.1: Enter the output of the program using overloading operators.

539740.3879454.qx3zqy7

Start

Type the program's output

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

class Movie {
public:
   Movie(string movieTitle);
   void SetUpVotesAndDownVotes(int numUpVotes, int numDownVotes) {
      upVotes = numUpVotes;
      downVotes = numDownVotes;
   }
   string GetTitle() const { return title; }
   int GetUpVotes() const { return upVotes; }
   int GetDownVotes() const { return downVotes; }

private:
   string title;
   int upVotes;
   int downVotes;
};

Movie::Movie(string movieTitle) {
   title = movieTitle;
   upVotes = 0;
   downVotes = 0;
}

bool operator==(const Movie& movie1, const Movie& movie2) {
   return movie1.GetUpVotes() == movie2.GetUpVotes();
}

int main() {
   Movie movie1("Up");
   Movie movie2("Thor");

   movie1.SetUpVotesAndDownVotes(9, 3);
   movie2.SetUpVotesAndDownVotes(7, 2);

   if (movie1 == movie2) {
      cout << "Equal" << endl;
   }
   else {
       cout << "Not equal" << endl;
   }

   return 0;
}
```



```
Not equal
```

1 2 3 4

Check Next

12.16 Vector ADT

vector ADT

The ***standard template library*** (***STL***) defines classes for common Abstract Data Types (ADTs). A ***vector*** is an ADT of an ordered, indexable list of items. The vector ADT is implemented as a class (actually a class template that supports different types such as `vector<int>` or `vector<string>`, although templates are discussed elsewhere).

For the commonly-used vector member functions below, assume a vector is declared as:

```
vector<T> vectorName();
```

where T represents the vector's element type, such as:

```
vector<int> teamNums(5);
```

Table 12.16.1: Vector ADT functions.

Notes: size_type is an unsigned integer type. T represents the vector's element type.

at()	`at(size_type n)` Accesses element n.	`teamNums.at(3)` *Assigns 99 to* `x = teamNums.a` *Assigns elemen* `x`
size()	`size_type size() const;` Returns vector's size.	`if (teamNums.s` *Size is 5 so c* `...` `}`
empty()	`bool empty() const;` Returns true if size is 0.	`if (teamNums.e` *is 5 so condit* `...` `}`
clear()	Removes all elements. Vector size becomes 0.	`teamNums.clear` *Vector now has* `cout << teamNu` *Prints 0* `teamNums.at(3)` *Error; element*
push_back()	`void push_back(const T& x);` Copies x to new element at vector's end, increasing size by 1. Parameter is pass by reference to avoid making local copy, but const to make clear not changed.	*// Assume vect* `teamNums.push_` *Vector is: 77* `teamNums.push_` *Vector is: 77,* `cout << teamNu` *Prints 2*
erase()	`iterator erase (iteratorPosition);` Removes element from position. Elements from higher positions are shifted back to fill gap. Vector size decrements.	*// Assume vect* `teamNums.erase` `+ 1); // Now 7` *// (Strange pc* *explained belc*
insert()	`iterator insert(iteratorPosition, const T& x);` Copies x to element at position. Items at that position and higher are shifted over to make room. Vector size increments.	*// Assume vect* `teamNums.inser` `+ 1, 33); // N`

Basic vector functions

Use of at(), size(), empty(), and clear() should be straightforward.

PARTICIPATION ACTIVITY 12.16.1: Vector functions at(), size(), empty(), and clear().

Given `vector<int> itemList(10);` Assume all elements have been assigned 0.

1) itemList().size returns 10.

- True
- False

2) itemList.size(10) returns 10.

- True
- False

3) itemList.size() returns 10.

- True
- False

4) itemList.at(10) returns 0.

- True
- False

5) itemList.empty() removes all elements.

- True
- False

6) After itemList.clear(), itemList.at(0) is an invalid access.

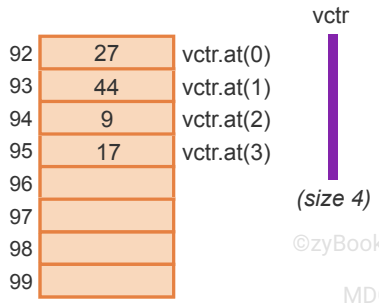
- True
- False

vector's push_back() function

push_back() appends an item to the vector's end, automatically resizing the vector.

PARTICIPATION ACTIVITY 12.16.2: Vector push_back() member function.

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

int main() {
   unsigned int i;
   vector<int> vctr;

   cout << "Size before: " << vctr.size();

   vctr.push_back(27);
   vctr.push_back(44);
   vctr.push_back(9);
   vctr.push_back(17);

   cout << ", after: " << vctr.size() << endl;
   cout << "Contents:" << endl;
   for (i = 0; i < vctr.size(); ++i) {
      cout << vctr.at(i) << endl;
   }

   return 0;
}
```




```
Size before: 0, after: 4
Contents:
27
44
9
17
```

Animation content:

Static Figure:
Begin C++ code:

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

int main() {
   unsigned int i;
   vector<int> vctr;

   cout << "Size before: " << vctr.size();

   vctr.push_back(27);
   vctr.push_back(44);
   vctr.push_back(9);
   vctr.push_back(17);

   cout << ", after: " << vctr.size() << endl;
   cout << "Contents:" << endl;
   for (i = 0; i < vctr.size(); ++i) {
      cout << vctr.at(i) << endl;
   }

   return 0;
}
```

End C++ code.

A box containing 8 memory addresses from 92 to 99, consecutively. The memory address 92 is labeled vctr.at(0) and contains “27”. The memory address 93 is labeled vctr.at(1) and contains “44”. The memory address 94 is labeled vctr.at(2) and contains “9”. The memory address 95 is labeled vctr.at(3) and contains “17”. Memory addresses 96 to 99 are empty. A purple bar representing the size of the vector is labeled vctr and spans the length of memory addresses 92 to 95. The text “(size 4)” is displayed at the end of the size bar. An output monitor displays text.

Monitor text begin:
Size before: 0, after: 4
Contents:
27
44
9
17
Monitor text end.

©zyBooks 01/31/24 17:52 1939727
Rob Daglio
MDCCOP2335Spring2024

Step 1: When initially declared, the vector vctr has a size of 0.

A highlight box highlights the C++ code line, vector<int> vctr;. Creating an empty vector named vctr. The highlight box moves to the C++ code line, cout << "Size before: " << vctr.size(); outputting Size before: 0 to the monitor.

Step 2: The push_back() function appends a new element to the vector's end and automatically

resizes the vector.

The highlighted box highlights each C++ code line vctr.push_back(27);, vctr.push_back(44);, vctr.push_back(9);, vctr.push_back(17); individually. As the line is highlighted the number value within the push_back method is added to the memory address location along with vector location vector.at(). The size bar is increased each time the push_back method is highlighted. The C++ code , cout << ", after: " << vctr.size() << endl;
cout << "Contents:" << endl;
for (i = 0; i < vctr.size(); ++i) {
  cout << vctr.at(i) << endl;
}
Is highlighted. The monitor outputs the contents of the vector as follows:
, after: 4
Contents:
27
44
9
17

Animation captions:

1. When initially declared, the vector vctr has a size of 0.
2. The push_back() function appends a new element to the vector's end and automatically resizes the vector.

One can deduce that the vector class has a private data member that stores the current size. In fact, the vector class has several private data members. However, to use a vector, a programmer only needs to know the public abstraction of the vector ADT.

Example: List of players' jersey numbers

The program below assists a soccer coach scouting players, allowing the coach to enter the jersey number of players, and printing a list of those numbers when requested.

The line highlighted in the PlayersAdd() function illustrates use of the push_back() member method. Note from the sample input/output that the items are stored in the vector in the order the items were added. Note that the programmer did not specify an initial vector size in main(), meaning the initial size is 0.

Figure 12.16.1: Using vector member functions: A player jersey numbers program.

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

// Adds playerNum to end of vector
void PlayersAdd(vector<int>& players, int
playerNum) {
   players.push_back(playerNum);
}

void PlayersPrint(const vector<int>& players) {
   unsigned int i;

   for (i = 0; i < players.size(); ++i) {
      cout << " " << players.at(i) << endl;
   }
}

// Maintains vector of player numbers
int main() {
   vector<int> players;
   int playerNum;
   char userKey;

   userKey = '?';

   cout << "Commands: 'a' add, 'p' print" << endl;
   cout << "    'q' quits" << endl;
   while (userKey != 'q') {
      cout << "Command: ";
      cin  >> userKey;
      if (userKey == 'a') {
         cout << " Player number: ";
         cin  >> playerNum;
         PlayersAdd(players, playerNum);
      }
      else if (userKey == 'p') {
         PlayersPrint(players);
      }
   }

   return 0;
}
```

```
Commands: 'a' add, 'p'
print
   'q' quits
Command: p
Command: a
 Player number: 23
Command: a
 Player number: 47
Command: p
 23
 47
Command: a
 Player number: 19
Command: p
 23
 47
 19
Command: q
```


PARTICIPATION ACTIVITY 12.16.3: push_back() function.

Given: `vector<int> itemList;`
If appropriate, type: Error
Answer the questions in order; each may modify the vector.

1) What is the initial vector's size?

Check Show answer

2) After itemList.at(0) = 99, what is the vector's size?

Check Show answer

3) After itemList.push_back(99), what is the vector's size?

Check Show answer

4) After itemList.push_back(77), what are the vector's contents? Type element values in order separated by one space as in: 44 66

Check Show answer

5) After itemList.push_back(44), what is the vector's size?

Check Show answer

6) What does itemList.at(itemList.size()) return?

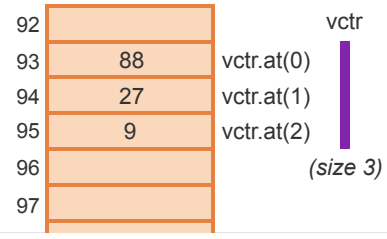
Check Show answer

vector's insert() and erase() member functions

The insert() function takes a position argument indicating where the new element should be inserted. However, position is not just a number like 1, but is rather: myVector.begin() + 1. The reason is beyond our scope here, but has to do with *iterators* that can be useful when iterating through a vector in a loop. The erase() function is similar.

PARTICIPATION ACTIVITY 12.16.4: insert() and erase() vector member functions.


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

int main() {
   unsigned int i;
   vector<int> vctr;

   vctr.push_back(27);
```



```
    vctr.push_back(44);
    vctr.push_back(9);
    vctr.push_back(17);
    vctr.erase(vctr.begin() + 1);
    vctr.insert(vctr.begin() + 0, 88);
    vctr.erase(vctr.begin() + 3);

    cout << "Contents:" << endl;
    for (i = 0; i < vctr.size(); ++i) {
       cout << " " << vctr.at(i) << endl;
    }
    return 0;
}
```

Contents:
88
27
9

Animation content:

Static Figure:
Begin C++ code:
#include <iostream>
#include <vector>
using namespace std;

int main() {
  unsigned int i;
  vector<int> vctr;

  vctr.push_back(27);
  vctr.push_back(44);
  vctr.push_back(9);
  vctr.push_back(17);
  vctr.erase(vctr.begin() + 1);
  vctr.insert(vctr.begin() + 0, 88);
  vctr.erase(vctr.begin() + 3);

  cout << "Contents:" << endl;
  for (i = 0; i < vctr.size(); ++i) {
    cout << " " << vctr.at(i) << endl;
  }
  return 0;
}
End C++ code.

A box containing 8 memory addresses from 92 to 99, consecutively. The memory address 93 is labeled vctr.at(0) and contains the value “88”. The memory address 94 is labeled vctr.at(1) and contains the value “27”. The memory address 95 is labeled vctr.at(2) and contains the value “9”. Memory addresses 92 and 96 through 99 are empty. A purple bar representing the size of the vector is labeled vctr and spans the length of memory addresses 93 to 95. The text “(size 3)” is displayed at the end of the size bar. An output monitor displays text.
Begin monitor text:
Contents:
88
27
9
End monitor text.

Step 1: The erase() function takes a position argument indicating where the element should be removed. Elements from higher positions are shifted back. The vector size is automatically

decremented.

The Java program runs, creating the vector. Values 27, 44, 8, and 17 are pushed into memory addresses 93, 94, 95, and 96 consecutivly. The size of the vector is 4. A highlighted box highlighted the C++ code line vctr.erase(vctr.begin() + 1);. vector.beingin + 1 locates the value 44 in the memory address 94. The value 44 is erased. Value 9 moves to memory address 94 and value 17 moves to memory address 95. The size of the vector changes to 3.

Step 2: The insert() function takes a position argument indicating where the element should be added and the element's value. Elements in that position and higher positions are shifted forward. The vector size is automatically incremented.

The highlighted box highlights the C++ code line vctr.insert(vctr.begin() + 0, 88);. vector.beingin + 0 locates the value 27 in the memory address 93. The value 88 is added to the vector. Value 88 is added to memory address 93. Value 27 moves to memory address 94, value 9 moves to memory address 95 and value 17 moves to memory address 96. The size of the vector changes to 4.

Step 3: Note that the position argument is not an integer, but an iterator. The begin() function returns an iterator pointing to the first element of the vector. Then, an integer value is added to indicate the desired element's position.
The highlighted box highlights the C++ code line vctr.erase(vctr.begin() + 3););. vector.beingin + 3 locates the value 17 in the memory address 96. The value 17 is erased. The size of the vector changes to 3.

The C++ code
```
cout << "Contents:" << endl;
  for (i = 0; i < vctr.size(); ++i) {
    cout << " " << vctr.at(i) << endl;
  }
  return 0;
```
is highlighted. The monitor displays the contents of the vector,
“Contents:
88
27
9”.

Animation captions:

1. The erase() function takes a position argument indicating where the element should be removed. Elements from higher positions are shifted back. The vector size is automatically decremented.
2. The insert() function takes a position argument indicating where the element should be added and the element's value. Elements in that position and higher positions are shifted forward. The vector size is automatically incremented.
3. Note that the position argument is not an integer, but an iterator. The begin() function returns an iterator pointing to the first element of the vector. Then, an integer value is added to indicate the desired element's position.

Example: Players' jersey numbers program with delete option

The erase() function can be used to extend the player jersey numbers program with a player delete option, as shown below.

The program's PlayersDelete() function uses a common while loop form for finding an item in a vector. The loop body checks if the current item is a match. If so, the item is deleted using the erase() function, and the variable found is set to true. The loop expression exits the loop if found is true, since no further search is necessary. A while loop is used rather than a for loop because the number of iterations is not known beforehand.

Figure 12.16.2: Using the vector erase() function.

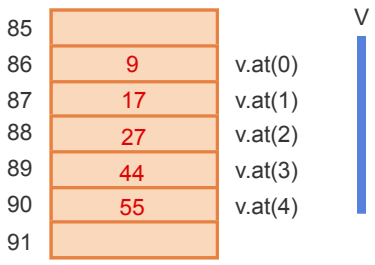
```
#include <iostream>
#include <vector>
using namespace std;

// Adds playerNum to end of vector
void PlayersAdd(vector<int>& players, int
playerNum) {
   players.push_back(playerNum);
}

void PlayersPrint(const vector<int>& players) {
   unsigned int i;

   for (i = 0; i < players.size(); ++i) {
      cout << " " << players.at(i) << endl;
   }
}

// Deletes playerNum from vector
void PlayersDelete(vector<int>& players, int
playerNum) {
   unsigned int i = 0;
   bool found = false;

   // Search for playerNum in vector
   while (!found && (i < players.size())) {
      if (players.at(i) == playerNum) {
         players.erase(players.begin() + i); //
Delete
         found = true;
      }
      ++i;
   }
}

// Maintains vector of player numbers
int main() {
   vector<int> players;
   int playerNum;
   char userKey;

   userKey = '?';

   cout << "Commands: 'a' add, 'p' print, 'd' del"
<< endl;
   cout << "   'q' quits" << endl;
   while (userKey != 'q') {
      cout << "Command: ";
      cin  >> userKey;
      if (userKey == 'a') {
         cout << " Player number: ";
         cin  >> playerNum;
         PlayersAdd(players, playerNum);
      }
      else if (userKey == 'p') {
         PlayersPrint(players);
      }
      else if (userKey == 'd') {
         cout << " Player number: ";
         cin  >> playerNum;
         PlayersDelete(players, playerNum);
      }
   }

   return 0;
}
```

```
Commands: 'a' add, 'p'
print, 'd' del
   'q' quits
Command: a
 Player number: 23
Command: a
 Player number: 47
Command: a
 Player number: 19
Command: p
 23
 47
 19
Command: d
 Player number: 23
Command: p
 47
 19
Command: d
 Player number: 19
Command: p
 47
Command: q
```

Inserting elements in sorted order

A common use of insert() is to insert a new item in sorted order.

PARTICIPATION ACTIVITY | 12.16.5: Intuitive depiction of how to add items to a vector while maintaining items in ascending sorted order.



Animation content:

Static Figure:
A box containing 7 memory addresses from 85 to 91, consecutively. The memory address 86 is labeled v.at(0) and contains the value “9”. The memory address 87 is labeled v.at(1) and contains the value “17”. The memory address 88 is labeled v.at(2) and contains the value “27”. The memory address 89 is labeled v.at(3) and contains the value “44”. The memory address 90 is labeled v.at(4) and contains the value “55”. Memory addresses 85 and 91 are empty. A blue bar representing the size of the vector is labeled V and spans the length of memory addresses 86 to 90.

Step 1: The first number is added to the vector.

A list of numbers, 55, 17, 9, 14, and 27 appear. The number 27 is added to the memory address 86 and is labeled v.at(0). The vector size is 1.

Step 2: 44 is greater than 27 so it is added to the end of the vector.

The number 44 is added to the memory address 86 and is labeled v.at(1). The vector size increases by one.

Step 3: 9 is less than 27. 44 and 27 are moved down so 9 can be added to front of the vector.

The number 9 is added to the memory address 86. The value 27 is moved to memory address 87. The value of 44 moves to memory address 88 and is labeled v.at(2). The vector size increases by one.

Step 4: The rest of the numbers are added in the appropriate spots.

The number 17 is added to the memory address 87 since 17 is greater than 9 and less than 27. The value 27 is moved to memory address 88. The value of 44 moves to memory address 89 and is labeled v.at(3). The vector size increases by one.

The number 55 is added to the memory address 90 since 55 is greater than 9, 17, 27, and 44. Memory address is labeled v.at(4) and the vector size increases by one. The final size of the vector is 5.

Animation captions:

1. The first number is added to the vector.
2. 44 is greater than 27 so it is added to the end of the vector.
3. 9 is less than 27. 44 and 27 are moved down so 9 can be added to front of the vector.
4. The rest of the numbers are added in the appropriate spots.

zyDE 12.16.1: Insert in sorted order.

Run the program and observe the output to be: 55 4 250 19. Modify the numsInsert function to insert each item in sorted order. The new program should output: 4 19 55 250

Load default template...

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

void numsInsert(vector<int>& numsList, int newNum) {
   unsigned int i;

   for (i = 0; i < numsList.size(); ++i) {
      if (newNum < numsList.at(i)) {
         // FIXME: insert newNum at element i
         break; // Exits the for loop
      }
   }
```

Run

PARTICIPATION ACTIVITY 12.16.6: The insert() and erase() functions.

Given: `vector<int> itemList;`
Assume itemList currently contains: 33 77 44.
Answer questions in order, as each may modify the vector.

1) itemList.at(1) returns 77.

- True
- False

2) itemList.insert(itemList.begin() + 1, 55) changes itemList to: 33 55 77 44.

- True
- False

3) itemList.insert(itemList.begin() + 0, 99) inserts 99 at the front of the list.

- True
- False

4) Assuming itemList is 99 33 55 77 44, then itemList.erase(itemList.begin() + 55) results in:
99 33 77 44

- True
- False

5) To maintain a list in ascending sorted order, a given new item should be inserted at the position of the first element that is greater than the item.

- True
- False

6) To maintain a list in descending sorted order, a given new item should be inserted at the position of the first element that is equal to the item.

- True
- False

Exploring further:

- Vectors at cplusplus.com
- Vectors at msdn.microsoft.com

CHALLENGE ACTIVITY 12.16.1: Enter the output of the vector ADT functions.

539740.3879454.qx3zqy7

Start

Type the program's output

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

void PrintSize(vector<int> numsList) {
   cout << "size " << numsList.size() << endl;
}

int main() {
   int currVal;
   vector<int> intList(3);

   PrintSize(intList);

   cin >> currVal;
   while (currVal >= 0) {
      intList.push_back(currVal);
      cin >> currVal;
   }

   PrintSize(intList);

   intList.clear();

   PrintSize(intList);

   return 0;
}
```

Input

1 2 3 4 -1

Output

```
size 3
size 7
size 0
```


1 2 3

Check Next

CHALLENGE ACTIVITY 12.16.2: Modifying vectors.

Modify the existing vector's contents, by erasing the element at index 1 (initially 200), then inserting 100 and 102 in the shown locations. Use Vector ADT's erase() and insert() only, and remember that the first argument of those functions is special, involving an iterator and not just an integer. Sample output of below program with input 33 200 10:

```
100 33 102 10
```

Learn how our autograder works

539740.3879454.qx3zqy7

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  void PrintVectors(vector<int> numsList) {
6     unsigned int i;
7
8     for (i = 0; i < numsList.size(); ++i) {
9        cout << numsList.at(i) << " ";
10    }
11    cout << endl;
12 }
13
14 int main() {
15    vector<int> numsList;
```


Run

12.17 Namespaces

Defining a namespace

A ***name conflict*** occurs when two or more items like variables, classes, or functions, have the same name. Ex: One programmer creates a Seat class for auditoriums, and a second programmer creates a Seat class for airplanes. A third programmer creating a reservation system for airline and concert tickets wants to use both Seat classes, but a compiler error occurs due to the name conflict.

A ***namespace*** defines a region (or scope) used to prevent name conflicts. Above, the auditorium seat class code can be put in an `auditorium` namespace, and airplane seat class code in an `airplane` namespace. The ***scope resolution operator ::*** allows specifying in which namespace to find a name, as in: `auditorium::Seat concertSeat;` and `airplane::Seat flightSeat;`.

PARTICIPATION ACTIVITY 12.17.1: Namespaces can resolve name conflicts.

main.cpp

```
#include "auditorium.h"
#include "airplane.h"

int main() {
   auditorium::Seat concertSeat;
   airplane::Seat flightSeat;

   // ...

   return 0;
};
```

auditorium.h

```
namespace auditorium {
   class Seat {
      ...
   };
}
```

airplane.h

```
namespace airplane {
   class Seat {
      ...
   };
}
```

Animation content:

Static figure: Three code blocks labeled main.cpp, auditorium.h, and airplane.h are displayed.
main.cpp code block:
Begin C++ code:
#include "auditorium.h"
#include "airplane.h"

int main() {
  Seat concertSeat;
  Seat flightSeat;

  // ...

  return 0;
};
End C++ code.

auditorium.h code block:
Begin C++ code:
```
  class Seat {
    ...
  };
```
End C++ code.

airplane.h code block:
Begin C++ code:
```
  class Seat {
    ...
  };
```
End C++ code.
Step 1: A Seat class is declared in auditorium.h, and another Seat class in airplane.h. The compiler generates an error due to a name conflict. The line of code in main.cpp, #include "auditorium.h", is highlighted and the line of code in auditorium.h, class Seat {, is highlighted. The line of code in main.cpp, #include "airplane.h", is highlighted and the line of code in airplane.h, class Seat {, is highlighted. The lines of code in main.cpp, Seat concertSeat;, Seat flightSeat;, are highlighted.
Step 2: The auditorium Seat class may be put in namespace "auditorium", and airplane seat code in a namespace "airplane". The line of code, namespace auditorium {, is added to the top of auditorium.h and is highlighted.
auditorium.h code block:
Begin C++ code:
```
namespace auditorium {
  class Seat {
    ...
  };
}
```
End C++ code.
The line of code, namespace airplane {, is added to the top of airplane.h and is highlighted.
airplane.h code block:
Begin C++ code:
```
namespace airplane {
  class Seat {
    ...
  };
}
```
End C++ code.
Step 3: The two kinds of seats can then be declared as auditorium::Seat concertSeat and airplane::Seat flightSeat. The compiler now knows which Seat is which. The highlighted lines of code in main.cpp, Seat concertSeat;, Seat flightSeat;, are replaced with the lines of code, auditorium::Seat concertSeat;, airplane::Seat flightSeat;.

Animation captions:

1. A Seat class is declared in auditorium.h, and another Seat class in airplane.h. The compiler generates an error due to a name conflict.
2. The auditorium Seat class may be put in namespace "auditorium", and airplane seat code in a namespace "airplane".
3. The two kinds of seats can then be declared as auditorium::Seat concertSeat and airplane::Seat flightSeat. The compiler now knows which Seat is which.

PARTICIPATION ACTIVITY 12.17.2: Namespaces.

1) Two same-named classes can cause a name conflict, but two same-named functions cannot.
   - True
   - False

2) A namespace helps avoid name conflicts among classes, functions, and other items in a program.
   - True
   - False

3) With namespaces, name conflicts cannot occur.
   - True
   - False

std namespace

All items in the C++ standard library are part of the ***std*** namespace (short for standard). To use classes like string or predefined objects like cout, a programmer can use one of two approaches:

1. ***Scope resolution operator (::):*** A programmer can use the scope resolution operator to specify the std namespace before C++ standard library items. Ex: `std::cout << "Hello";` or `std::string userName;`
2. ***Namespace directive***: A programmer can add the statement ***using*** `namespace std;` to direct the compiler to check the std namespace for any names later in the file that aren't otherwise declared. Ex: For `string userName;`, the compiler will check namespace std for string.

For code clarity, most programming guidelines discourage `using namespace` directives except perhaps for std.

PARTICIPATION ACTIVITY 12.17.3: std namespace.

1) Standard library items like classes and functions are part of a namespace named std.
   - True
   - False

2) The namespace directive `using namespace std;` is required in any program.
   - True
   - False

3) Without `using namespace std;`, a programmer can access cout using std::cout.

- True
- False

4) Without any namespace directive, `cout << num1` causes the compiler to check the std namespace for cout.

- True
- False

CHALLENGE ACTIVITY 12.17.1: Enter the output from the proper namespace.

539740.3879454.qx3zqy7

Start

Type the program's output

main.cpp | imperial.h | metric.h

```
#include "imperial.h"
#include "metric.h"
#include <iostream>

int main() {
   metric::BiggerUnit();

   imperial::SmallerUnit();

   return 0;
}
```

```
1m = 0.001km
1ft = 12in
```

1 2

Check Next

12.18 Static data members and functions

Static data members

The keyword ***static*** indicates a variable is allocated in memory only once during a program's execution. Static variables are allocated memory once and reside in the program's static memory region for the entire program. Thus, a static variable retains a value throughout the program.

In a class, a ***static data member*** is a data member of the class instead of a data member of each class object. Thus, static data members are independent of any class object, and can be accessed without creating a class object.

A static data member is declared inside the class definition, but must also be defined outside the class declaration. Within a class function, a static data member can be accessed just by variable name. A public static data member can be accessed

outside the class using the scope resolution operator: ClassName::variableName.

PARTICIPATION ACTIVITY 12.18.1: Static data member used to create object ID numbers.

```
class Store {
   public:
      Store(string storeName, string storeType);
      int getId();
      static int nextId;   // Declare static member variable
   private:
      string name = "None";
      string type = "None";
      int id = 0;
};

Store::Store(string storeName, string storeType) {
   name = storeName;
   type = storeType;
   id = nextId;   // Assign object id with nextId

   ++nextId;      // Increment nextId for next object to be created
}
...
int Store::nextId = 101; // Define and initialize static data member

int main() {

   Store store1("Macy's", "Department");
   Store store2("Albertsons", "Grocery");
   Store store3("Ace", "Hardware");

   cout << "Store 1's ID: "<< store1.getId() << endl;
   cout << "Store 2's ID: "<< store2.getId() << endl;
   cout << "Store 3's ID: "<< store3.getId() << endl;
   cout << "Next ID: " << Store::nextId << endl;

   return 0;
}
```

Animation content:

Static Figure:
Begin C++ code:
class Store {
  public:
    Store(string storeName, string storeType);
    int getId();
    static int nextId; // Declare static member variable
  private:
    string name = "None";
    string type = "None";
    int id = 0;
};

Store::Store(string storeName, string storeType) {
  name = storeName;
  type = storeType;
  id = nextId; // Assign object id with nextId

  ++nextId;   // Increment nextId for next object to be created
}
...

```
int Store::nextId = 101; // Define and initialize static data member

int main() {

   Store store1("Macy's", "Department");
   Store store2("Albertsons", "Grocery");
   Store store3("Ace", "Hardware");

   cout << "Store 1's ID: " << store1.getId() << endl;
   cout << "Store 2's ID: " << store2.getId() << endl;
   cout << "Store 3's ID: " << store3.getId() << endl;
   cout << "Next ID: " << Store::nextId << endl;

   return 0;
}
```

End C++ code.

The program in the static figure is executed and the steps and substeps of the program are described in the following three tables and five steps. The program uses four different regions of memory: Static memory, store1, store2, and store3. All ten memory locations, as well as the output console, are initially empty.

Step 1: The Store class' static data member nextId is declared in the Store class declaration.

Code executed	Memory label	Value stored	Region of memory
static int nextId;	Store::nextId		Static memory

Step 2: Store::nextId must be defined and initialized outside the class declaration. Only one instance of that variable will exist in memory.

Code executed	Memory label	Value stored	Region of memory
int Store::nextId = 101; // Define and initialize static data member	Store::nextId	101	Static memory

Step 3: When a Store object is created, memory is allocated for the object's name, type, and id data members, but not the static member nextId.

Code executed	Memory label	Value stored	Region of memory
Store store1("Macy's", "Department");			store1

Step 4: The constructor assigns an object's id with nextId, and then increments nextId. Each time an object is created, nextId is incremented, and each object has a unique id.

Code executed	Memory label	Value stored	Region of memory

name = storeName;	store1.name	Macy's	store1
type = storeType;	store1.type	Department	store1
id = nextId;	store1.id	101	store1
++nextId;	Store::nextId	102	Static memory
Store store2("Albertsons", "Grocery");	Three memory locations are allocated: store2.name, store2.type, and, store2.id		store2
name = storeName;	store2.name	Albertsons	store2
type = storeType;	store2.type	Grocery	store2
id = nextId;	store2.id	102	store2
++nextId;	Store::nextId	103	Static memory
Store store3("Ace", "Hardware");	Three memory locations are allocated: store3.name, store3.type, and, store3.id		store3
name = storeName;	store3.name	Ace	store3
type = storeType;	store3.type	Hardware	store3
id = nextId;	store3.id	103	store3
++nextId;	Store::nextId	104	Static memory

Step 5: Any class member function can access or mutate a static data member. nextId can also be accessed outside the class using the scope resolution operator (::).

The following code is executed:
cout << "Store 1's ID: " << store1.getId() << endl;
cout << "Store 2's ID: " << store2.getId() << endl;
cout << "Store 3's ID: " << store3.getId() << endl;
cout << "Next ID: " << Store::nextId << endl;

The output console now contains four lines of output:
Store 1's ID: 101
Store 2's ID: 102
Store 3's ID: 103
Next ID: 104

Animation captions:

1. The Store class' static data member nextId is declared in the Store class declaration.
2. Store::nextId must be defined and initialized outside the class declaration. Only one instance of that variable will exist in memory.
3. When a Store object is created, memory is allocated for the object's name, type, and id data members, but not the static member nextId.

4. The constructor assigns an object's id with nextId, and then increments nextId. Each time an object is created, nextId is incremented, and each object has a unique id.
5. Any class member function can access or mutate a static data member. nextId can also be accessed outside the class using the scope resolution operator (::).

PARTICIPATION ACTIVITY 12.18.2: Static data members.

1) Each constructed class object creates a new instance of a static data member.
   - True
   - False

2) All static data members can be accessed anywhere in a program.
   - True
   - False

3) Outside of the class where declared, private static data members can be accessed using dot notation.
   - True
   - False

Static member functions

A ***static member function*** is a class function that is independent of class objects. Static member functions are typically used to access and mutate private static data members from outside the class. Since static methods are independent of class objects, the `this` parameter is not passed to a static member function. So, a static member function can only access a class' static data members.

Figure 12.18.1: Static member function used to access a private static data member.

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

class Store {
public:
   Store(string storeName, string storeType);
   int getId();
   static int getNextId();

private:
   string name = "None";
   string type = "None";
   int id = 0;
   static int nextId;   // Declare static member variable
};

Store::Store(string storeName, string storeType) {
   name = storeName;
   type = storeType;
   id = nextId;        // Assign object id with nextId

   ++nextId;          // Increment nextId for next object to be created
}

int Store::getId() {
   return id;
}

int Store::getNextId() {
   return nextId;
}

int Store::nextId = 101;   // Define and initialize static data member

int main() {
   Store store1("Macy's", "Department");
   Store store2("Albertsons", "Grocery");
   Store store3("Ace", "Hardware");

   cout << "Store 1's ID: "<< store1.getId() << endl;
   cout << "Store 2's ID: "<< store2.getId() << endl;
   cout << "Store 3's ID: "<< store3.getId() << endl;
   cout << "Next ID: " << Store::getNextId() << endl;

   return 0;
}
```

```
Store 1's ID: 101
Store 2's ID: 102
Store 3's ID: 103
Next ID: 104
```


PARTICIPATION ACTIVITY 12.18.3: Static member functions.

1) A static member function is needed to access or mutate a _____ static data member from outside of the class.

- ○ public
- ○ private

2) The `this` parameter can be used in a static member function to access an object's non-static data members.

- True
- False

CHALLENGE ACTIVITY 12.18.1: Enter the output with static members.

539740.3879454.qx3zqy7

Start

Type the program's output

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

class FoodType {
   public:
      FoodType(string foodType);
      static int nextId;
      void Print();
   private:
      string type = "None";
      int id = 0;
};

FoodType::FoodType(string foodType) {
   type = foodType;
   id = nextId;

   nextId += 3;
}

void FoodType::Print() {
   cout << type << ": " << id << endl;
}

int FoodType::nextId = 10;

int main() {
   FoodType order1("Soup");
   FoodType order2("Burrito");
   FoodType order3("Lobster");

   order3.Print();

   return 0;
}
```

```
Lobster: 16
```

1 2

Check Next

12.19 C++ example: Salary calculation with classes

zyDE 12.19.1: Calculate salary: Using classes.

The program below uses a class, TaxTableTools, which has a tax table built in. The main function prompts for a salary, then uses a TaxTableTools function to get the tax rate. The program then calculates the tax to pay and displays the results to the user. Run the program with annual salaries of 10000, 50000, 50001, 100001 and -1 (to end the program) and note the output tax rate and tax to pay.

1. Modify the TaxTableTools class to use a setter function that accepts a new salary and tax rate table.
2. Modify the program to call the new function, and run the program again, noting the same output.

The program's two classes are in separate tabs at the top.

Note: The calculation is inaccurate to how taxes are formally assessed and is a simplification for educational purposes only.

Current file: IncomeTaxMain.cpp Load default template...

```
#include <iostream>
#include <limits>
#include <vector>
#include <string>
#include "TaxTableTools.h"

int GetInteger(const string userPrompt) {
   int inputValue;

   cout << userPrompt << ": " << endl;
   cin >> inputValue;

   return inputValue;
}
```

10000 50000 50001 100001 -1

Run

zyDE 12.19.2: Calculate salary: Using classes (solution).

A solution to the above problem follows.

Note that the program's two classes are in separate tabs at the top.

Current file: IncomeTaxMain.cpp Load default template...

```
#include <iostream>
#include <limits>
#include <vector>
#include <string>
#include "TaxTableTools.h"

int GetInteger(const string userPrompt) {
   int inputValue;

   cout << userPrompt << ": " << endl;
   cin >> inputValue;

   return inputValue;
}
```


10000 50000 50001 100001 -1

Run

zyDE 12.19.3: Salary calculation: Overloading a constructor.

The program below calculates a tax rate and tax to pay given an annual salary. The program uses a class, TaxTableTools, which has the tax table built in. Run the program with annual salaries of 10000, 50000, 50001, 100001 and -1 (to end the program) and note the output tax rate and tax to pay.

1. Overload the constructor.
   a. Add to the TaxTableTools class an overloaded constructor that accepts the base salary table and corresponding tax rate table as parameters.
   b. Modify the main function to call the overloaded constructor with the two tables (vectors) provided in the main function. Be sure to set the nEntries value, too.
   c. Note that the tables in the main function are the same as the tables in the TaxTableTools class. This sameness facilitates testing the program with the same annual salary values listed above.
   d. Test the program with the annual salary values listed above.
2. Modify the salary and tax tables
   a. Modify the salary and tax tables in the main function to use different salary ranges and tax rates.
   b. Use the just-created overloaded constructor to initialize the salary and tax tables.
   c. Test the program with the annual salary values listed above.

Note: The calculation is inaccurate to how taxes are formally assessed and is a simplification for educational purposes only.

Current file: IncomeTaxMain.cpp Load default template...

```
1  #include <iostream>
2  #include <limits>
3  #include <vector>
4  #include <string>
5  #include "TaxTableTools.h"
6  using namespace std;
7
8  int main() {
9     const string PROMPT_SALARY = "\nEnter annual salar
10    int annualSalary;
11    double taxRate;
12    int taxToPay;
13    vector<int> salaryBase(5);
14    vector<double> taxBase(5);
15
```



```
10000
50000
50001
```

Run

zyDE 12.19.4: Salary calculation: Overloading a constructor (solution).

A solution to the above problem follows.

Note that the program's two classes are in separate tabs at the top.

Current file: IncomeTaxMain.cpp Load default template...

```
#include <iostream>
#include <limits>
#include <vector>
#include <string>
#include "TaxTableTools.h"
using namespace std;

int main() {
   const string PROMPT_SALARY = "\nEnter annual salar
   int annualSalary;
   double taxRate;
   int taxToPay;
   vector<int> salaryBase(5);
   vector<double> taxBase(5);
```



```
10000
50000
50001
```

Run

12.20 C++ example: Domain name availability with classes

zyDE 12.20.1: Domain name availability: Using classes.

The program below uses a class, DomainAvailabilityTools, which includes a table of registered domain names. The main function prompts for domain names until the user presses Enter at the prompt. The domain name is checked against a list of the registered domains in the DomainAvailabilityTools class. If the domain name is not available, the program displays similar domain names.

1. Run the program and observe the output for the given input.
2. Modify the DomainAvailabilityClass's function named GetSimilarDomainNames so that some unavailable domain names do not get a list of similar domain names. Run the program again and observe that unavailable domain names with TLDs of .org or .biz do not have similar names.

Current file: DomainAvailabilityMain.cpp Load default template...

```
#include <iostream>
#include <string>
#include <cctype>
#include "DomainAvailabilityTools.h"
using namespace std;

// ************************************************

// prompts user string. Returns string.
string GetString(string prompt) {
   string userInput;

   cout << prompt << endl;
   cin >> userInput;
```



```
programming.com
apple.com
oracle.com
```

Run

zyDE 12.20.2: Domain validation: Using classes (solution).

A solution to the above problem follows.

Current file: DomainAvailabilityMain.cpp Load default template...

```
#include <iostream>
#include <string>
#include <cctype>
#include "DomainAvailabilityTools.h"
using namespace std;

// ***********************************************

// Prompts user for input string and returns the stri
string GetString(string prompt) {
   string userInput;

   cout << prompt << endl;
   cin >> userInput;

```



```
programming.com
apple.com
oracle.com
```

Run

12.21 LAB: Triangle area comparison (classes)

Given class `Triangle` (in files Triangle.h and Triangle.cpp), complete main() to read and set the base and height of triangle1 and of triangle2, determine which triangle's area is smaller, and output that triangle's info, making use of `Triangle`'s relevant member functions.

Ex: If the input is:

```
3.0 4.0
4.0 5.0
```

where 3.0 is triangle1's base, 4.0 is triangle1's height, 4.0 is triangle2's base, and 5.0 is triangle2's height, the output is:

```
Triangle with smaller area:
Base: 3.00
Height: 4.00
Area: 6.00
```


539740.3879454.qx3zqy7

LAB ACTIVITY | 12.21.1: LAB: Triangle area comparison (classes) | 0 / 10

File is marked as read only Current file: Triangle.cpp

```
#include "Triangle.h"
#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;

void Triangle::SetBase(double userBase) {
   base = userBase;
}

void Triangle::SetHeight(double userHeight) {
   height = userHeight;
}
```


Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → **Triangle.cpp** (Your program) → Output (shown below)

Program output displayed here

Coding trail of your work What is this?

History of your effort will appear here once you begin working on this zyLab.

12.22 LAB: Car value (classes)

Given main(), complete the `Car` class (in files Car.h and Car.cpp) with member functions to set and get the purchase price of a car (SetPurchasePrice(), GetPurchasePrice()), and to output the car's information (PrintInfo()).

Ex: If the input is:

```
2011
18000
2018
```

where 2011 is the car's model year, 18000 is the purchase price, and 2018 is the current year, the output is:

```
Car's information:
  Model year: 2011
  Purchase price: $18000
  Current value: $5770
```

Notes:

- PrintInfo() should use two spaces for indentation.
- Add `cout << fixed << setprecision(0);` right before printing currentValue to ignore cents in the output.

LAB ACTIVITY | 12.22.1: LAB: Car value (classes) | 0 / 10

File is marked as read only Current file: main.cpp

```
1  #include <iostream>
2  #include "Car.h"
3  using namespace std;
4
5  int main() {
6     int userYear;
7     int userPrice;
8     int userCurrentYear;
9     Car myCar;
10
11    cin >> userYear;
12    cin >> userPrice;
13    cin >> userCurrentYear;
14
15    myCar.SetModelYear(userYear);
```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program Input (from above) → main.cpp (Your program) → Output (shown below)

Program output displayed here

Coding trail of your work What is this?

History of your effort will appear here once you begin working on this zyLab.

12.23 LAB: Winning team (classes)

Given main(), define the Team class (in files Team.h and Team.cpp). For class member function GetWinPercentage(), the formula is:
`wins / (wins + losses)`. *Note: Use casting to prevent integer division.*

For class member function PrintStanding(), output the win percentage of the team with two digits after the decimal point and whether the team has a winning or losing average. A team has a winning average if the win percentage is 0.5 or greater.

Ex: If the input is *Ravens 13 3*, where Ravens is the team's name, 13 is the number of team wins, and 3 is the number of team losses, the output is:

```
Win percentage: 0.81
Congratulations, Team Ravens has a winning average!
```

Ex: If the input is *Angels 80 82*, the output is:

```
Win percentage: 0.49
Team Angels has a losing average.
```


LAB ACTIVITY | 12.23.1: LAB: Winning team (classes) 0 / 10

File is marked as read only Current file: main.cpp

```
1  #include <iostream>
2  #include <string>
3  #include "Team.h"
4  using namespace std;
5
6  int main() {
7     string name;
8     int wins;
9     int losses;
10    Team team;
11
12    cin >> name;
13    cin >> wins;
14    cin >> losses;
15
```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program Input (from above) → main.cpp (Your program) → Output (shown below)

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin working
on this zyLab.
```

12.24 LAB: Nutritional information (classes/constructors)

Given main(), complete the `FoodItem` class (in files FoodItem.h and FoodItem.cpp) with constructors to initialize each food item. The default constructor should initialize the name to "Water" and all other class data members to 0.0. The second constructor should have four parameters (food name, grams of fat, grams of carbohydrates, and grams of protein) and should assign each class data member with the appropriate parameter value.

Ex: If the input is:

```
Water
```

the output is:

```
Nutritional information per serving of Water:
  Fat: 0.00 g
  Carbohydrates: 0.00 g
  Protein: 0.00 g
Number of calories for 1.00 serving(s): 0.00
```

Ex: If the input is:

```
M&M's
10.0
34.0
2.0
3.0
```

where M&M's is the food name, 10.0 is the grams of fat, 34.0 is the grams of carbohydrates, 2.0 is the grams of protein, and 3.0 is the number of servings, the output is:


```
Nutritional information per serving of M&M's:
  Fat: 10.00 g
  Carbohydrates: 34.00 g
  Protein: 2.00 g
Number of calories for 1.00 serving(s): 234.00
Number of calories for 3.00 serving(s): 702.00
```

Note: The program outputs the number of calories for one serving of a food and for the input number of servings as well. The program only outputs the calories for one serving of water.

539740.3879454.qx3zqy7

LAB ACTIVITY | 12.24.1: LAB: Nutritional information (classes/constructors) 0 / 10

Current file: FoodItem.cpp Load default template...


```
#include "FoodItem.h"
#include <iostream>
#include <iomanip>

using namespace std;

// Define default constructor

// Define second constructor with parameters
// to initialize private data members

string FoodItem::GetName() {
   return name;
}
```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program Input (from above) → FoodItem.cpp (Your program) → Output (shown below)

Program output displayed here

Coding trail of your work What is this?

History of your effort will appear here once you begin working on this zyLab.

12.25 LAB: Artwork label (classes/constructors)

Given main(), complete the `Artist` class (in files Artist.h and Artist.cpp) with constructors to initialize an artist's information, get member functions, and a PrintInfo() member function. The default constructor should initialize the artist's name to

"unknown" and the years of birth and death to -1. PrintInfo() displays "Artist:", then a space, then the artist's name, then another space, then the birth and death dates in one of three formats:

- `(XXXX to YYYY)` if both the birth and death years are nonnegative
- `(XXXX to present)` if the birth year is nonnegative and the death year is negative
- `(unknown)` otherwise

Complete the `Artwork` class (in files Artwork.h and Artwork.cpp) with constructors to initialize an artwork's information, get member functions, and a PrintInfo() member function. The default constructor should initialize the title to "unknown", the year created to -1. PrintInfo() displays an artist's information by calling the PrintInfo() function in the `Artist` class, followed by the artwork's title and the year created. Declare a private field of type `Artist` in the `Artwork` class.

Ex: If the input is:

```
Pablo Picasso
Three Musicians
1881
1973
1921
```

1881 and 1973 being the birth and death years respectively, with 1921 being the year the work was created, the output is:

```
Artist: Pablo Picasso (1881 to 1973)
Title: Three Musicians, 1921
```

Ex: If the input is:

```
Brice Marden
Distant Muses
1938
-1
2000
```

the output is:

```
Artist: Brice Marden (1938 to present)
Title: Distant Muses, 2000
```

Ex: If the input is:

```
Banksy
Balloon Girl
-1
-1
2002
```


the output is:

```
Artist: Banksy (unknown)
Title: Balloon Girl, 2002
```


LAB ACTIVITY | 12.25.1: LAB: Artwork label (classes/constructors) 0 / 10

File is marked as read only Current file: main.cpp

```
#include "Artist.h"
#include "Artwork.h"
#include <iostream>
#include <string>
using namespace std;

int main() {
   string userTitle, userArtistName;
   int yearCreated, userBirthYear, userDeathYear;

   getline(cin, userArtistName);
   getline(cin, userTitle);
   cin >> userBirthYear;
   cin >> userDeathYear;
   cin >> yearCreated;
```


Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program Input (from above) → main.cpp (Your program) → Output (shown below)

Program output displayed here

Coding trail of your work What is this?

History of your effort will appear here once you begin working on this zyLab.

12.26 LAB: Vending machine

Given two integers as user inputs that represent the number of drinks to buy and the number of bottles to restock, create a VendingMachine object that performs the following operations:

- Purchases input number of drinks
- Restocks input number of bottles
- Reports inventory

Review the definition of "VendingMachine.cpp" by clicking on the orange arrow. A VendingMachine's initial inventory is 20 drinks.

Ex: If the input is:

```
5 2
```

the output is:

```
Inventory: 17 bottles
```

539740.3879454.qx3zqy7

LAB ACTIVITY | 12.26.1: LAB: Vending machine 0 / 10

Current file: main.cpp Load default template...

```
1 #include <iostream>
2
3 #include "VendingMachine.h"
4 using namespace std;
5
6 int main() {
7    /* Type your code here */
8 }
```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → main.cpp (Your program) → Output (shown below)

Program output displayed here

Coding trail of your work What is this?

History of your effort will appear here once you begin working on this zyLab.

12.27 Module 1 Project: Grocery shopping list (linked list: inserting at the end of a list)

Given main(), define an InsertAtEnd() member function in the `ItemNode` class that adds an element to the end of a linked list. **DO NOT print the dummy head node.**

Ex. if the input is:


```
4
Kale
Lettuce
Carrots
Peanuts
```

where 4 is the number of items to be inserted; Kale, Lettuce, Carrots, Peanuts are the names of the items to be added at the end of the list.

The output is:

```
Kale
Lettuce
Carrots
Peanuts
```


LAB ACTIVITY | 12.27.1: Module 1 Project: Grocery shopping list (linked list: inserting at the end of a list) 0 / 10

Current file: ItemNode.h Load default template...

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  class ItemNode {
6  private:
7     string item;
8     ItemNode* nextNodeRef;
9
10 public:
11    // Constructor
12    ItemNode() {
13       item = "";
14       nextNodeRef = NULL;
15    }
```

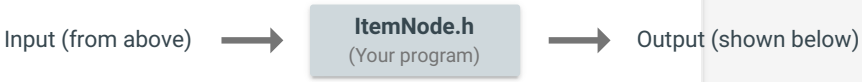

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program



Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin working
on this zyLab.
```

©zyBooks 01/31/24 17:52 1939727
Rob Daglio
MDCCOP2335Spring2024