10.1 Enumerations

Some variables only need to store a small set of named values. For example, a variable representing a traffic light need only store values named GREEN, YELLOW, or RED. An ***enumeration type*** (enum) declares a name for a new type and possible values for that type.

Construct 10.1.1: Enumeration type.

```
enum identifier {enumerator1, enumerator2,
...};
```

The items within the braces ("enumerators") are integer constants automatically assigned an integer value, with the first item being 0, the second 1, and so on. An enumeration declares a new data type that can be used like the built-in types int, char, etc.

Figure 10.1.1: Enumeration example.

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

/* Manual controller for traffic light */
int main() {
   enum LightState {LS_RED, LS_GREEN, LS_YELLOW,
LS_DONE};
   LightState lightVal;
   char userCmd;

   lightVal = LS_RED;
   userCmd = '-';

   cout << "User commands: n (next), r (red), q
(quit)." << endl << endl;

   lightVal = LS_RED;
   while (lightVal != LS_DONE) {

      if (lightVal == LS_GREEN) {
         cout << "Green light  ";
         cin >> userCmd;
         if (userCmd == 'n') { // Next
            lightVal = LS_YELLOW;
         }
      }
      else if (lightVal == LS_YELLOW) {
         cout << "Yellow light  ";
         cin >> userCmd;
         if (userCmd == 'n') { // Next
            lightVal = LS_RED;
         }
      }
      else if (lightVal == LS_RED) {
         cout << "Red light  ";
         cin >> userCmd;
         if (userCmd == 'n') { // Next
            lightVal = LS_GREEN;
         }
      }

      if (userCmd == 'r') { // Force immediate
red
         lightVal = LS_RED;
      }
      else if (userCmd == 'q') { // Quit
         lightVal = LS_DONE;
      }
   }

   cout << "Quit program." << endl;

   return 0;
}
```

```
User commands: n (next), r
(red), q (quit).

Red light  n
Green light  n
Yellow light  n
Red light  n
Green light  r
Red light  n
Green light  n
Yellow light  n
Red light  q
Quit program.
```


The program declares a new enumeration type named LightState. The program then declares a new variable lightVal of that type. The loop updates lightVal based on the user's input.

The example illustrates the idea of a ***state machine*** that is sometimes used in programs, especially programs that interact with physical objects, wherein the program moves among particular situations ("states") depending on input; see What is: State machine.

Because different enumerated types might use some of the same names, e.g., `enum Colors {RED, PURPLE, BLUE, GREEN};` might also appear in the same program, the program above follows the practice of prepending a distinguishing prefix, in this case "LS" (for Light State).

One might ask why the light variable wasn't simply declared as a string, and then compared with strings "GREEN", "RED", and "YELLOW". Enumerations are safer. If using a string, an assignment like `light = "ORANGE"` would not yield a compiler error, even though ORANGE is not a valid light color. Likewise, `light == "YELOW"` would not yield a compiler error, even though YELLOW is misspelled.

One could instead declare constant strings like `const string LS_GREEN = "GREEN";` or even integer values like `const int LS_GREEN = 0;` and then use those constants in the code, but an enumeration is clearer, requires less code, and is less prone to error.

Note: Each enumerator by default is assigned an integer value of 0, 1, 2, etc. However, a programmer can assign a specific value to any enumerator. Ex: `enum TvChannels {TC_CBS = 2, TC_NBC = 5, TC_ABC = 7};`

PARTICIPATION ACTIVITY 10.1.1: Enumeration syntax.

1) Which of the following declares a new enumeration type named CarGear, with PARK, REVERSE, and DRIVE?

- `enum CarGear (PARK, REVERSE, DRIVE);`
- `enum CarGear {PARK, REVERSE, DRIVE}`
- `enum CarGear {PARK, REVERSE, DRIVE};`
- `CarGear {PARK, REVERSE, DRIVE};`

PARTICIPATION ACTIVITY 10.1.2: Enumerations.

1) Declare a new enumeration type named HvacStatus with three named values HVAC_OFF, AC_ON, FURNACE_ON, in that order.

Check Show answer

2) Declare a variable of the enumeration type HvacStatus named systemStatus.

Check Show answer

3) Assign AC_ON to the variable systemStatus.

Check Show answer

4) What is the integer value of systemStatus after the following?

```
systemStatus = FURNACE_ON;
```

Check Show answer

5) Given `enum TvChannels {TC_CBS = 2, TC_NBC = 5, TC_ABC = 7};`, what does `cout << TC_ABC;` output?

Check Show answer

CHALLENGE ACTIVITY 10.1.1: Enumerations: Grocery items.

Output "Fruit" if the value of userItem is a type of fruit. Otherwise, if the value of userItem is a type of drink, output "Drink". Otherwise, output "Unknown". End each output with a newline.

Ex: If userItem is `GR_APPLES`, then the output is:

```
Fruit
```

Learn how our autograder works

519134.1787752.qx3zqy7

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

int main() {
   enum GroceryItem {GR_APPLES, GR_BANANAS, GR_JUICE, GR_WATER};
   GroceryItem userItem;

   userItem = GR_APPLES; /* Your code will be tested with GR_APPLES and other values

   /* Your solution goes here  */

   return 0;
}
```

Run

View your last submission

CHALLENGE ACTIVITY 10.1.2: Soda machine with enums.

The following program reads a number from input to indicate the coin inserted into a soda machine. Complete the code provided to add the appropriate amount to totalDeposit.

- Enumerator ADD_QUARTER has a value of 0 (add 25).
- Enumerator ADD_DIME has a value of 1 (add 10).
- Enumerator ADD_NICKEL has a value of 2 (add 5).

Learn how our autograder works

519134.1787752.qx3zqy7

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

int main() {
   enum AcceptedCoins {ADD_QUARTER, ADD_DIME, ADD_NICKEL};
   int totalDeposit;
   int userInput;

   totalDeposit = 0;
   cin >> userInput;

   if (userInput == ADD_QUARTER) {
      totalDeposit = totalDeposit + 25;
   }

   /* Your solution goes here  */

   else {
```

Run

View your last submission

10.2 Grouping data: struct

Sometimes two data items are really aspects of the same data. For example, time might be recorded in hours and minutes, as in 4 hours and 23 minutes. Or a point on a plot might be recorded as x = 5, y = 7. Storing such data in separate variables, such as runTimeHours and runTimeMinutes, is not as clear as grouping that data into a single variable, like runTime, which might have subitems runTime.hourValue and runTime.minuteValue.

PARTICIPATION ACTIVITY 10.2.1: Naturally grouped data.

1) Select the pair forming part of a person's height (in U.S. units)

- Feet and inches
- Inches and salary
- Pounds and ounces

2) Select the group of items indicating the change provided to a person who pays for a meal.

- Ounce, gill, pint, quart, and gallon
- Mile, furlong, yard, feet, and inches
- Dollars, quarters, dimes, nickels, and pennies

The ***struct*** construct defines a new type, which can be used to declare a variable with subitems. The following animation illustrates.

PARTICIPATION ACTIVITY 10.2.2: A struct enables creating a variable with data members.

```
struct TimeHrMin {
   int hourValue;
   int minuteValue;
};

...

   TimeHrMin runTime1;
   TimeHrMin runTime2;
   TimeHrMin runTime3;

   runTime1.hourValue = 5;
   runTime1.minuteValue = 46;
   runTime3.hourValue = runTime1.hourValue;
```

96	5	hourValue	runTime1
97	46	minuteValue	
98	?	hourValue	runTime2
99	?	minuteValue	
100	5	hourValue	runTime3
101	?	minuteValue	
102			

Animation content:

Code snippet is as follows:

```
struct TimeHrMin {
  int hourValue;
  int minuteValue;
};

...

  TimeHrMin runTime1;
  TimeHrMin runTime2;
  TimeHrMin runTime3;

  runTime1.hourValue = 5;
  runTime1.minuteValue = 46;
  runTime3.hourValue = runTime1.hourValue;
```

Final memory contents is as follows:
96 (runTime1's hourValue): 5
97 (runTime1's hourValue): 46
98 (runTime2's hourValue): ?
99 (runTime2's hourValue): ?
100 (runTime3's hourValue): 5
101 (runTime3's hourValue): ?
102: empty

Animation captions:

1. The struct construct just declares new type; no memory is allocated.
2. Variable definitions allocate memory for each object's member.
3. Accesses refer to an object member's memory location.

The programmer uses a struct to define and use a new type as follows.

Construct 10.2.1: Defining and using a new struct type.

```
struct StructTypeName
{
   type item1;
   type item2;
   ...
   type itemN;
};

...
StructTypeName myVar;

myVar.item1 = ...
```

Each `type` may be any type like int or char. Each struct subitem is called a ***data member***. For a declared variable, each struct data member can be accessed using ".", known as a ***member access*** operator, sometimes called ***dot notation***.

Assigning a variable of a struct type to another such variable automatically assigns each corresponding data member, as shown below.

PARTICIPATION ACTIVITY 10.2.3: Assigning a struct type.


```
struct TimeHrMin {
   int hourValue;
   int minuteValue;
};

...

   TimeHrMin runTime1;
   TimeHrMin runTime2;
   TimeHrMin runTime3;

   runTime1.hourValue = 5;
   runTime1.minuteValue = 46;
   runTime2 = runTime1;
```

96	5	hourValue	runTime1
97	46	minuteValue	
98	5	hourValue	runTime2
99	46	minuteValue	
100	?	hourValue	runTime3
101	?	minuteValue	
102			

Animation content:

Code snippet is as follows:

```
struct TimeHrMin {
  int hourValue;
  int minuteValue;
};

...

  TimeHrMin runTime1;
  TimeHrMin runTime2;
  TimeHrMin runTime3;

  runTime1.hourValue = 5;
  runTime1.minuteValue = 46;
  runTime2 = runTime1;
```

Final memory contents is as follows:

```
96 (runTime1's hourValue): 5
97 (runTime1's hourValue): 46
98 (runTime2's hourValue): ?
99 (runTime2's hourValue): ?
100 (runTime3's hourValue): 5
101 (runTime3's hourValue): ?
102: empty
```

Animation captions:

1. Assigning a variable of a struct type to another such variable automatically assigns each corresponding data member.

PARTICIPATION ACTIVITY | 10.2.4: The struct construct.

1) A struct definition for CartesianPoint has subitems int x and int y. How many int locations in memory does the struct definition allocate?

Check | Show answer

2) If struct definition CartesianPoint has subitems int x and int y, how many total int locations in memory are allocated for these variable declarations?

```
int myNum;
CartesianPoint myPoint1;
CartesianPoint myPoint2;
```

Check Show answer

3) Given time1 is of type TimeHrMin defined earlier. What is the value of variable min after the following statements?

```
time1.hourValue   = 5;
time1.minuteValue = 4;
min = (60 * time1.hourValue)
+ time1.minuteValue;
```

Check Show answer

4) Write a statement to assign 12 to the hourValue data member of TimeHrMin variable time1.

Check Show answer

5) Write a statement that assigns the value of the hourValue data member of time1 into the hourValue data member of time2.

Check Show answer

6) Write a single statement that assigns the values of all data members of time1 to the corresponding data members of time2.

Check Show answer

7) Declare a variable person1 of type Person, where Person is already defined as a struct type.

Check Show answer

CHALLENGE ACTIVITY 10.2.1: Enter the output using struct.

519134.1787752.qx3zqy7

Start

Type the program's output

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

struct Height {
   int feet;
   int inches;
};

int main() {
   Height annHeight;

   annHeight.feet = 3;
   annHeight.inches = 2;

   cout << "Ann: " << annHeight.feet << "ft " << annHeight.inches << endl;

   return 0;
}
```

```
Ann: 3ft 2
```

1 2 3 4 5

Check Next

CHALLENGE ACTIVITY 10.2.2: Declaring a struct.

Define a struct named PatientData that contains two integer data members named heightInches and weightPounds. Sample output for the given program with inputs 63 115:

```
Patient data: 63 in, 115 lbs
```

Learn how our autograder works

519134.1787752.qx3zqy7

```
1  #include <iostream>
2  using namespace std;
3
4  /* Your solution goes here  */
5
6  int main() {
7     PatientData lunaLovegood;
8
9     cin >> lunaLovegood.heightInches;
10    cin >> lunaLovegood.weightPounds;
```

```
   cin >> lunaLovegood.weightPounds;

   cout << "Patient data: "
        << lunaLovegood.heightInches << " in, "
        << lunaLovegood.weightPounds << " lbs" << endl;

   return 0;
}
```

Run

View your last submission

CHALLENGE ACTIVITY 10.2.3: Accessing a struct's data members.

Write a statement to print the data members of InventoryTag. End with newline. Ex: if itemID is 314 and quantityRemaining is 500, print:

```
Inventory ID: 314, Qty: 500
```

Learn how our autograder works

519134.1787752.qx3zqy7

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

struct InventoryTag {
   int itemID;
   int quantityRemaining;
};

int main() {
   InventoryTag redSweater;

   cin >> redSweater.itemID;
   cin >> redSweater.quantityRemaining;

   /* Your solution goes here  */

   return 0;
}
```

Run

View your last submission

10.3 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 | 10.3.1: 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 | 10.3.2: Programs commonly are not viewed as variables and functions/methods, but rather as objects.

Name
Phone
Cuisines
Reviews
Name
Phone
Amenities
Reviews

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

Hotel
Set main info
Add amenity

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 | 10.3.3: 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 10.3.4: Objects strongly support abstraction / information hiding.

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 | 10.3.5: 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

10.4 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 10.4.1: 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();
```

favPlace object

Name: Central Deli
Rating: 4

Central Deli -- 4

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:

©zyBooks 04/03/24 12:00 893876
Anthony Hamlin
DMACCCIS161Spring2024

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 10.4.2: 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.

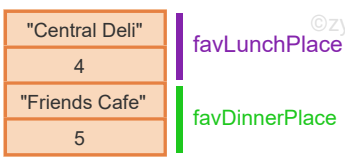
PARTICIPATION ACTIVITY 10.4.3: Using the Restaurant class.

```
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();
```




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

```
    return 0;
}
```

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 | 10.4.4: 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 10.4.1: 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 10.4.5: 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>

10.5 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 10.5.1: 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 10.5.2: 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

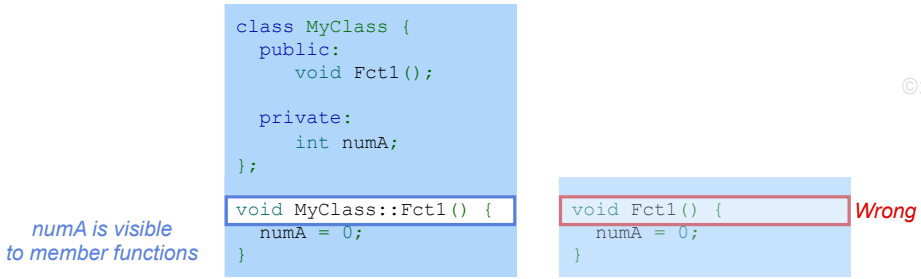
3) A class definition should provide comments along with each private data member so that a class user knows how those data members are used.
   - True
   - False

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 | 10.5.3: 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 10.5.1: 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 10.5.4: 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 10.5.2: 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 10.5.5: 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 10.5.1: Classes.

519134.1787752.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 = "Sam";

   person1.SetName(userName);
   cout << "You are " << person1.GetName();

   return 0;
}
```

```
You are Sam
```

1 2 3 4

Check Next

CHALLENGE ACTIVITY 10.5.2: 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

519134.1787752.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;
}

void PersonInfo::IncNumKids() {
   numKids = numKids + 1;
```

Run

View your last submission

CHALLENGE ACTIVITY 10.5.3: Defining a class.

519134.1787752.qx3zqy7

Start

In the Customer class, complete the function definition for SetNumPoints() with the integer parameter newNumPoints.

Ex: If the input is `4.5 Milton 76`, then the output is:

```
Height: 4.5
Name: Milton
Number of points: 76
```

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

```
4  class Customer {
5     public:
6        void SetHeight(double newHeight);
7        void SetName(string newName);
8        void SetNumPoints(int newNumPoints);
9        double GetHeight();
10       string GetName();
11       int GetNumPoints();
12    private:
13       double height;
14       string name;
15       int numPoints;
16 };
17
18 void Customer::SetHeight(double newHeight) {
```


1 2

Check Next level

10.6 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 10.6.1: 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 10.6.1: 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 10.6.2: 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 10.6.3: 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 10.6.1: Inline member functions.

519134.1787752.qx3zqy7

Start

Type the program's output

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

class State {
   public:
      void SetName(string stateName) {
         name = stateName;
      }
      void SetCode(string stateCode) {
         code = stateCode;
      }
      void Print() const;

   private:
      string name;
      string code;
};

void State::Print() const {
   cout << name << ", " << code << endl;
}

int main() {
   State stateToVisit;

   stateToVisit.SetName("Wisconsin");
   stateToVisit.SetCode("WI");

   stateToVisit.Print();

   return 0;
}
```



```
Wisconsin, WI
```

1 2

Check Next

10.7 Preprocessor and include

The ***preprocessor*** is a tool that scans the file from top to bottom looking for any lines that begin with #, known as a ***hash symbol***. Each such line is not a program statement, but rather directs the preprocessor to modify the file in some way before compilation continues, each such line being known as a ***preprocessor directive***. The directive ends at the end of the line, no semicolon is used at the end of the line.

Perhaps the most commonly-used preprocessor directive is ***#include***, known as an ***include directive***. #include directs the compiler to replace that line by the contents of the given filename.

Construct 10.7.1: Include directives.

```
#include
"filename"
#include
<filename>
```


The following animation illustrates.

PARTICIPATION ACTIVITY 10.7.1: Preprocessor's handling of an include directive.

```
#include "myfile.h"
// myfile.h
void myFct1( );
int Fct2(int parm1);

int main() {
   myFct1();
   if (x < Fct2(9)) {
      ...
   }
   return 0;
}
```

```
// myfile.h
void myFct1( );
int Fct2(int parm1);
```


Animation content:

Static figure:
Begin code:
#include "myfile"

// myfile.h
void myFct1( );
int Fct2(int parm1);

int main() {
   myFct1();
   if (x < Fct2(9)) {
      ...
   }
   return 0;
}
End code.
Begin code:
// myfile.h
void myFct1( );
int Fct2(int parm1);
End code.
Two files are displayed. A main file that includes an include directive for the myfile.h file and a file named myfile.h that contains two function declarations.

Step 1: The preprocessor replaces include by contents of myfile.h during compilation. The preprocessor replaces the include directive with the two function declarations from file myfile.h.

Animation captions:

1. The preprocessor replaces include by contents of myfile.h during compilation.

Good practice is to use a .h suffix for any file that will be included in another file. The h is short for header, to indicate that the file is intended to be included at the top (or header) of other files. Although any file can be included in any other file, convention is to only include .h files.

The characters surrounding the filename determine where the preprocessor looks for the file.

- `#include "myfile.h"` -- A filename in quotes causes the preprocessor to look for the file in the same folder/directory as the including file.
- `#include <stdfile>` -- A filename in angle brackets causes the preprocessor to look in the system's standard library folder/directory. Programmers typically use angle brackets only for standard library files, using quotes for all other include files. Note that nearly every previous example has included at least one standard library file, using angle brackets.

- Header files that are part of the standard C++ library do not have a .h extension.
- Items that were originally part of the C standard library have a "c" prepended, as in cmath.

PARTICIPATION ACTIVITY 10.7.2: Include directives.

1) The preprocessor processes any line beginning with what symbol?
   - #
   - <filename>
   - "filename"

2) After a source file is processed by the preprocessor, is it correct to say that all hash symbols will be removed from the code remaining to be compiled?
   - yes
   - no

3) Do header files have to end in .h?
   - yes
   - no

4) Where does the preprocessor look for myfile.h in the line:
   `#include "myfile.h"`
   - Current folder
   - System folder
   - Unknown

5) What one symbol is incorrect in the following:
   `#include <stdlib.h>;`
   - #
   - <>
   - ;

Exploring further:

- Preprocessor tutorial on cplusplus.com
- Preprocessor directives on MSDN

10.8 Separate files

Separating part of a program's code into a separate file can yield several benefits. One benefit is preventing a main file from becoming unmanageably large. Another benefit is that the separated part could be useful in other programs.

Suppose a program has several related functions that operate on triples of numbers, such as computing the maximum of three numbers or computing the average of three numbers. Those related functions' definitions can be placed in their own file as shown below in the file `threeintsfcts.cpp`.

Figure 10.8.1: Putting related functions in their own file.

main.cpp

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

// Normally lots of other code here

int main() {

   cout << ThreeIntsSum(5, 10, 20) << endl;
   cout << ThreeIntsAvg(5, 10, 20) << endl;

   return 0;
}

// Normally lots of other code here
```

threeintsfcts.cpp

```
int ThreeIntsSum(int num1, int num2, int num3) {
   return (num1 + num2 + num3);
}

int ThreeIntsAvg(int num1, int num2, int num3) {
   int sum;
   sum = num1 + num2 + num3;
   return (sum / 3);
}
```

threeintsfcts.h

```
int ThreeIntsSum(int num1, int num2, int num3);
int ThreeIntsAvg(int num1, int num2, int num3);
```

```
>
a.out
35
11
>
```

One could then compile the main.cpp and threeintsfcts.cpp files together as shown below.

Figure 10.8.2: Compiling multiple files together.

Without `#include "threeintsfcts.h"` in main.cpp	With `#include "threeintsfcts.h"` in main.cpp

```
> g++ -Wall main.cpp threeintsfcts.cpp
main.cpp: In function  int main():
main.cpp:8: error: ThreeIntsSum was not
declared in this scope
main.cpp:9: error: ThreeIntsAvg was not
declared in this scope
```

```
> g++ -Wall main.cpp threeintsfcts.cpp
>
```

Just compiling those two files (without the `#include "threeintsfcts.h"` line in the main file) would yield an error, as shown above on the left. The problem is that the compiler does not see the function definitions while processing the main file because those definitions are in another file, which is similar to what occurs when defining functions after main(). The solution for both situations is to provide function declarations before main() so the compiler knows enough about the functions to compile calls to those functions. Instead of typing the declarations directly above main(), a programmer can provide the function declarations in a header file, such as the threeintsfcts.h file provided in the figure above. The programmer then includes the contents of that file into a source file via the line: `#include "threeintsfcts.h"`.

The reader may note that the .h file could have contained function definitions rather than just function declarations, eliminating the need for two files (one for declarations, one for definitions). However, the two file approach has two key advantages. One advantage is that with the two file approach, the .h file serves as a brief summary of all functions available. A second advantage is that the main file's copy does not become exceedingly large during compilation, which can lead to slow compilation.

One last consideration that must be dealt with is that a header file could get included multiple times, causing the compiler to generate errors indicating an item defined in that header file is defined multiple times (the above header files only declared functions and didn't define them, but other header files may define functions, types, constants, and other items). Multiple inclusion commonly can occur when one header file includes another header file, e.g., the main file includes file1.h and file2.h, and file1.h also includes file2.h -- thus, file2.h would get included twice into the main file.

The solution is to add some additional preprocessor directives, known as header file guards, to the .h file as follows.

Construct 10.8.1: Header file guards.

```
#ifndef FILENAME_H
#define FILENAME_H

// Header file
contents

#endif
```

Header file guards are preprocessor directives, which cause the compiler to only include the contents of the header file once. `#define FILENAME_H` defines the symbol FILENAME_H to the preprocessor. The `#ifndef FILENAME_H` and `#endif` form a pair that instructs the preprocessor to process the code between the pair only if FILENAME_H is not defined ("ifndef" is short for "if not defined"). Thus, if the preprocessor includes encounter the header more than once, the code in the file during the second and any subsequent encounters will be skipped because FILENAME_H was already defined.

Good practice is to guard every header file. The following shows the threeintsfcts.h file with the guarding code added.

Figure 10.8.3: All header files should be guarded.

```
#ifndef THREEINTSFCTS_H
#define THREEINTSFCTS_H

int ThreeIntsSum(int num1, int num2, int
num3);
int ThreeIntsAvg(int num1, int num2, int
num3);

#endif
```


PARTICIPATION ACTIVITY 10.8.1: Header files.

1) Header files must end with .h.
   - True
   - False

2) Header files should contain function definitions for functions declared in another file.
   - True
   - False

3) Guarding a header file prevents multiple inclusion of that file by the preprocessor.
   - True
   - False

4) Is the following the correct two-line sequence to guard a file named myfile.h?

```
#ifdef MYFILE_H
#define MYFILE_H
```

   - True
   - False

Exploring further:

- Preprocessor tutorial on cplusplus.com
- Preprocessor directives on MSDN

10.9 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 10.9.1: 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 10.9.1: 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 10.9.2: .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 10.9.2: 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 04/03/24 12:00 893876
Anthony Hamlin
DMACCCIS161Spring2024

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 | 10.9.3: Restaurant review program .h and .cpp files.

If unable to drag and drop, refresh the page.

Reviews.h **Review.h** **Restaurant.cpp** **Reviews.cpp** **Restaurant.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 | 10.9.1: Enter the output of separate files.

519134.1787752.qx3zqy7

Start

Type the program's output

main.cpp | Product.h | Product.cpp

Input

```
10 Cheese
6 Flowers
7 Belt
-1
```

Output

```
$10 Cheese
```

```
#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.GetPrice() << " " << resultProduct.GetName() << endl;

   return 0;
}
```


1 2 3

Check Next