7.1 Reasons for defining functions

Improving program readability

Programs can become hard for humans to read and understand. Decomposing a program into functions can greatly aid program readability, helping yield an initially correct program, and easing future maintenance. Below, the program with functions has a main() that is easier to read and understand. For larger programs, the effect is even greater.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Figure 7.1.1: With program functions: main() is easy to read and understand.

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

double StepsToMiles(int numSteps) {
   const double FEET_PER_STEP = 2.5;               // Typical
adult
   const int    FEET_PER_MILE = 5280;

   return numSteps * FEET_PER_STEP * (1.0 / FEET_PER_MILE);
}

double StepsToCalories(int numSteps) {
   const double STEPS_PER_MINUTE = 70.0;           // Typical
adult
   const double CALORIES_PER_MINUTE_WALKING = 3.5; // Typical
adult
   double minutesTotal;
   double caloriesTotal;

   minutesTotal = numSteps / STEPS_PER_MINUTE;
   caloriesTotal = minutesTotal * CALORIES_PER_MINUTE_WALKING;

   return caloriesTotal;
}

int main() {
   int stepsWalked;

   cout << "Enter number of steps walked: ";
   cin  >> stepsWalked;

   cout << "Miles walked: " << StepsToMiles(stepsWalked)    <<
endl;
   cout << "Calories: "     << StepsToCalories(stepsWalked) <<
endl;

   return 0;
}
```

```
Enter number of steps walked: 1600
Miles walked: 0.757576
Calories: 80
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Figure 7.1.2: Without program functions: main() is harder to read and understand.

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

int main() {
   int    stepsWalked;
   const double FEET_PER_STEP = 2.5;                 // Typical
adult
   const int    FEET PER MILE = 5280;
   const double STEPS_PER_MINUTE = 70.0;             // Typical
adult
   const double CALORIES_PER_MINUTE_WALKING = 3.5; // Typical
adult
   double minutesTotal;
   double caloriesTotal;
   double milesWalked;

   cout << "Enter number of steps walked: ";
   cin >> stepsWalked;

   milesWalked = stepsWalked * FEET_PER_STEP * (1.0 /
FEET_PER_MILE);
   cout << "Miles walked: " << milesWalked << endl;

   minutesTotal = stepsWalked / STEPS_PER_MINUTE;
   caloriesTotal = minutesTotal * CALORIES_PER_MINUTE_WALKING;
   cout << "Calories: " << caloriesTotal << endl;

   return 0;
}
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

PARTICIPATION ACTIVITY 7.1.1: Improved readability.

Consider the above examples.

1) In the example *without* functions, how many statements are in main()?
   - 6
   - 16

2) In the example *with* functions, how many statements are in main()?
   - 6
   - 16

3) Which has fewer *total* lines of code (including blank lines), the program with or without functions?
   - With
   - Without
   - Same

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

4) The program with functions called the functions directly in output statements. Did the program without functions directly put calculations in output statements?

- No
- Yes

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Modular and incremental program development

Programmers commonly use functions to write programs modularly and incrementally.

- ***Modular development*** is the process of dividing a program into separate modules that can be developed and tested separately and then integrated into a single program.
- ***Incremental development*** is a process in which a programmer writes, compiles, and tests a small amount of code, then writes, compiles, and tests a small amount more (an incremental amount), and so on.
- A ***function stub*** is a function definition whose statements have not yet been written.

A programmer can use function stubs to capture the high-level behavior of main() and the required function (or modules) before diving into details of each function, like planning a route for a road trip before starting to drive. A programmer can then incrementally develop and test each function independently.

PARTICIPATION ACTIVITY 7.1.2: Function stub used in incremental program development.

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

double ConvKilometersToMiles(double numKm) {
   double milesPerKm = 0.621371;
   return numKm * milesPerKm;
}

double ConvLitersToGallons(double numLiters) {
   double gallonsPerLiter = 0.264172;
   return numLiters * gallonsPerLiter;
}

double CalcMpg(double distMiles, double gasGallons) {
   cout << "FIXME: Calculate MPG" << endl;
   return 0.0;
}

int main() {
   double distKm;
   double distMiles;
   double gasLiters;
   double gasGal;
   double userMpg;

   cout << "Enter kilometers driven: ";
   cin >> distKm;
   cout << "Enter liters of gas consumed: ";
   cin >> gasLiters;

   distMiles = ConvKilometersToMiles(distKm);
   gasGal = ConvLitersToGallons(gasLiters);
   userMpg = CalcMpg(distMiles, gasGal);

   cout << "Miles driven: " << distMiles << endl;
   cout << "Gallons of gas: " << gasGal << endl;
   cout << "Mileage: " << userMpg << " mpg" << endl;

   return 0;
}
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation content:

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

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

double ConvKilometersToMiles(double numKm) {
   cout << "FIXME: Convert km to m" << endl;
   return 0.0;
}

double ConvLitersToGallons(double numLiters) {
   cout << "FIXME: Convert l to gal" << endl;
   return 0.0;
}

double CalcMpg(double distMiles, double gasGallons) {
   cout << "FIXME: Calculate MPG" << endl;
   return 0.0;
}

int main() {
   double distKm;
   double distMiles;
   double gasLiters;
   double gasGal;
   double userMpg;

   cout << "Enter kilometers driven: ";
   cin >> distKm;
   cout << "Enter liters of gas consumed: ";
   cin >> gasLiters;

   distMiles = ConvKilometersToMiles(distKm);
   gasGal = ConvLitersToGallons(gasLiters);
   userMpg = CalcMpg(distMiles, gasGal);

   cout << "Miles driven: " << distMiles << endl;
   cout << "Gallons of gas: " << gasGal << endl;
   cout << "Mileage: " << userMpg << " mpg" << endl;

   return 0;
}
```

End C++ code.
Step 1: main() captures the program's high-level behavior and makes calls to three function stubs. The program can be compiled and tested before writing and testing those functions. The lines of code, distMiles = ConvKilometersToMiles(distKm);, gasGal = ConvLitersToGallons(gasLiters);, userMpg = CalcMpg(distMiles, gasGal);, are highlighted and the three function stubs, double ConvKilometersToMiles(double numKm) {, double ConvLitersToGallons(double numLiters) {, double CalcMpg(double distMiles, double gasGallons) {, are highlighted.
Step 2: The programmer then writes and tests the ConvKilometersToMiles() and

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

ConvLitersToGallons() functions. The body of the ConvKilometersToMiles() function is replaced with the lines of code, double milesPerKm = 0.621371;, return numKm * milesPerKm;. The body of the ConvLitersToGallons() function is replaced with the lines of code, double gallonsPerLiter = 0.264172;, return numLiters * gallonsPerLiter.
Step 3: After testing the ConvKilometersToMiles() and ConvLitersToGallons() functions, the programmer can then complete the program by writing and testing the CalcMpg() function. The lines of code, double CalcMpg(double distMiles, double gasGallons) {, cout << "FIXME: Calculate MPG" << endl;, return 0.0;, }, are highlighted.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation captions:

1. main() captures the program's high-level behavior and makes calls to three functions stubs. The program can be compiled and tested before writing and testing those functions.
2. The programmer then writes and tests the ConvKilometersToMiles() and ConvLitersToGallons() functions.
3. After testing the ConvKilometersToMiles() and ConvLitersToGallons() functions, the programmer can then complete the program by writing and testing the CalcMpg() function.

PARTICIPATION ACTIVITY 7.1.3: Incremental development.

1) Incremental development may involve more frequent compilation, but ultimately lead to faster development of a program.
   - ○ True
   - ○ False

2) The program above does not compile because CalcMpg() is a function stub.
   - ○ True
   - ○ False

3) A key benefit of function stubs is faster running programs.
   - ○ True
   - ○ False

4) Modular development means to divide a program into separate modules that can be developed and tested independently and then integrated into a single program.
   - ○ True
   - ○ False

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

zyDE 7.1.1: Function stubs.

Complete the program by writing and testing the CalcMpg() function.

Load default template...

```
#include <iostream>
using namespace std;
// Program converts a trip's kilometers and liters ir

double ConvKilometersToMiles(double numKm) {
   double milesPerKm = 0.621371;
   return numKm * milesPerKm;
}

double ConvLitersToGallons(double numLiters) {
   double gallonsPerLiter = 0.264172;
   return numLiters * gallonsPerLiter;
}

double CalcMpg(double distMiles, double gasGallons) {
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

410.2 33.5

Run

Avoid writing redundant code

A function can be defined once, then called from multiple places in a program, thus avoiding redundant code. Examples of such functions are math functions like abs() that relieve a programmer from having to write several lines of code each time an absolute value needs to be computed.

The skill of decomposing a program's behavior into a good set of functions is a fundamental part of programming that helps characterize a good programmer. Each function should have easily-recognizable behavior, and the behavior of main() (and any function that calls other functions) should be easily understandable via the sequence of function calls.

A general guideline (especially for beginner programmers) is that a function's definition usually shouldn't have more than about 30 lines of code, although this guideline is not a strict rule.

PARTICIPATION ACTIVITY 7.1.4: Redundant code can be replaced by multiple calls to one function.

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

int main() {
   double pizzaDiameter1;
   double pizzaDiameter2;
   double totalPizzaArea;
   double circleRadius1;
   double circleRadius2;
   double circleArea1;
   double circleArea2;
   double piVal = 3.14159265;

   pizzaDiameter1 = 12.0;
   circleRadius1 = pizzaDiameter1 / 2.0;
   circleArea1 = piVal * circleRadius1 *
```

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

double CalcCircleArea(double circleDiameter) {
   double circleRadius;
   double circleArea;
   double piVal = 3.14159265;

   circleRadius = circleDiameter / 2.0;
   circleArea = piVal * circleRadius *
                circleRadius;

   return circleArea;
}

int main() {
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

```
                circleRadius1;

   pizzaDiameter2 = 14.0;
   circleRadius2 = pizzaDiameter2 / 2.0;
   circleArea2 = piVal * circleRadius2 *
                circleRadius2;

   totalPizzaArea = circleArea1 + circleArea2;

   cout << "A 12 and 14 inch pizza has "
        << totalPizzaArea
        << " inches squared combined." << endl;

   return 0;
}
```

main() with redundant code

```
   double pizzaDiameter1;
   double pizzaDiameter2;
   double totalPizzaArea;

   pizzaDiameter1 = 12.0;
   pizzaDiameter2 = 14.0;

   totalPizzaArea = CalcCircleArea(pizzaDiameter1)
                    CalcCircleArea(pizzaDiameter2);

   cout << "A 12 and 14 inch pizza has "
        << totalPizzaArea
        << " inches squared combined." << endl;

   return 0;
}
```

main() calls CalcCircleArea() avoiding redundant code

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation content:

Static figure: A code block is displayed.
Begin C++ code:
#include <iostream>
using namespace std;

int main() {
  double pizzaDiameter1;
  double pizzaDiameter2;
  double totalPizzaArea;
  double circleRadius1;
  double circleRadius2;
  double circleArea1;
  double circleArea2;
  double piVal = 3.14159265;

  pizzaDiameter1 = 12.0;
  circleRadius1 = pizzaDiameter1 / 2.0;
  circleArea1 = piVal * circleRadius1 *
          circleRadius1;

  pizzaDiameter2 = 14.0;
  circleRadius2 = pizzaDiameter2 / 2.0;
  circleArea2 = piVal * circleRadius2 *
          circleRadius2;

  totalPizzaArea = circleArea1 + circleArea2;

  cout << "A 12 and 14 inch pizza has "
     << totalPizzaArea
     << " inches squared combined." << endl;

  return 0;
}
End C++ code.
Step 1: Circle area is calculated twice, leading to redundant code. The lines of code, circleRadius1 = pizzaDiameter1 / 2.0;, circleArea1 = piVal * circleRadius1 * circleRadius1;, circleRadius2 = pizzaDiameter2 / 2.0;, circleArea2 = piVal * circleRadius2 *circleRadius2;, are highlighted.
Step 2: The redundant code can be replaced by defining a CalcCircleArea function. A new code block

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

is displayed:
Begin C++ code:

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

double CalcCircleArea(double circleDiameter) {
  double circleRadius;
  double circleArea;
  double piVal = 3.14159265;

  circleRadius = circleDiameter / 2.0;
  circleArea = piVal * circleRadius *
               circleRadius;

return circleArea;
}
```

End C++ code.
Step 3: Then main is simplified by calling the CalcCircleArea() function from multiple places in the program. The main is added to the bottom of the new code block:
Begin C++ code:

```
int main() {
  double pizzaDiameter1;
  double pizzaDiameter2;
  double totalPizzaArea;

  pizzaDiameter1 = 12.0;
  pizzaDiameter2 = 14.0;

  totalPizzaArea = CalcCircleArea(pizzaDiameter1) +
           CalcCircleArea(pizzaDiameter2);

  cout << "A 12 and 14 inch pizza has "
     << totalPizzaArea
     << " inches squared combined." << endl;

  return 0;
}
```

End C++ code.
The text, main() with redundant code, appears below the original code block while the text, main() calls CalcCircleArea() avoiding redundant code, appears below the new code block.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation captions:

1. Circle area is calculated twice, leading to redundant code.
2. The redundant code can be replaced by defining a CalcCircleArea function.
3. Then main is simplified by calling the CalcCircleArea() function from multiple places in the program.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

PARTICIPATION ACTIVITY 7.1.5: Reasons for defining functions.

1) A key reason for creating functions is to help main() run faster.

- True
- False

2) Avoiding redundancy means to avoid calling a function from multiple places in a program.

- True
- False

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

3) If a function's internal statements are revised, all function calls will have to be modified too.

- True
- False

4) A benefit of functions is to increase redundant code.

- True
- False

CHALLENGE ACTIVITY 7.1.1: Functions: Factoring out a unit-conversion calculation.

Write a function so that the main() code below can be replaced by the simpler code that calls function MphAndMinutesToMiles(). Original main():

```
int main() {
   double milesPerHour;
   double minutesTraveled;
   double hoursTraveled;
   double milesTraveled;

   cin >> milesPerHour;
   cin >> minutesTraveled;

   hoursTraveled = minutesTraveled / 60.0;
   milesTraveled = hoursTraveled * milesPerHour;

   cout << "Miles: " << milesTraveled << endl;

   return 0;
}
```

Learn how our autograder works

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

539740.3879454.qx3zqy7

```
1  #include <iostream>
2  using namespace std;
3
4  /* Your solution goes here  */
5
6  int main() {
7     double milesPerHour;
8     double minutesTraveled;
9
```

```
    cin >> milesPerHour;
    cin >> minutesTraveled;

    cout << "Miles: " << MphAndMinutesToMiles(milesPerHour, minutesTraveled)

    return 0;
```

Run

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

CHALLENGE ACTIVITY 7.1.2: Function stubs: Statistics.

Define stubs for the functions called by the below main(). Each stub should print "FIXME: Finish FunctionName()" followed by a newline, and should return -1. Example output:

```
FIXME: Finish GetUserNum()
FIXME: Finish GetUserNum()
FIXME: Finish ComputeAvg()
Avg: -1
```

Learn how our autograder works

539740.3879454.qx3zqy7

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

/* Your solution goes here  */

int main() {
   int userNum1;
   int userNum2;
   int avgResult;

   userNum1 = GetUserNum();
   userNum2 = GetUserNum();

   avgResult = ComputeAvg(userNum1, userNum2);
```

Run

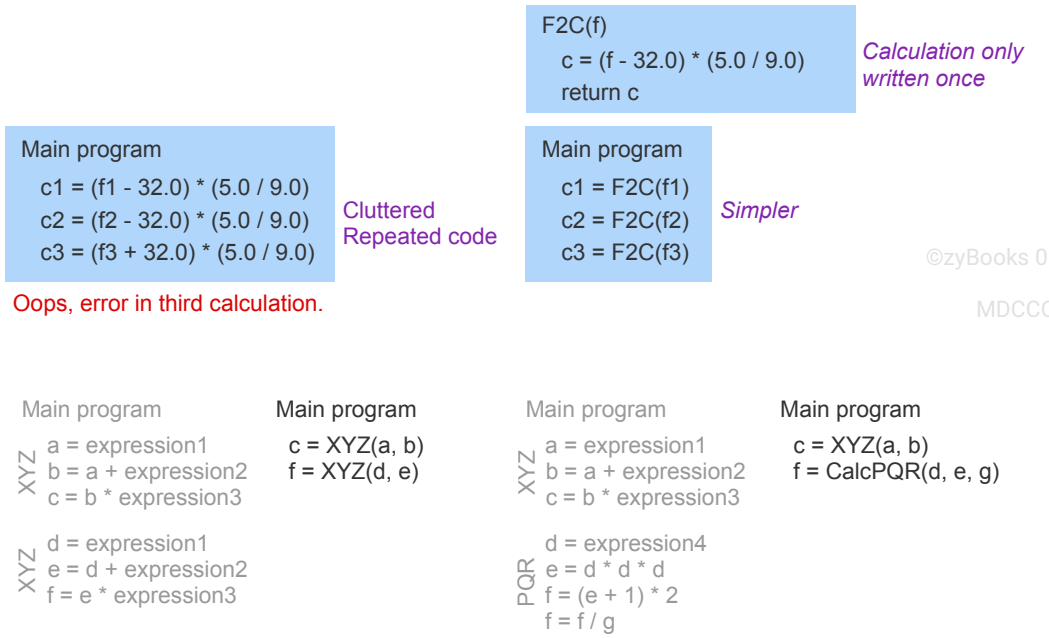
7.2 User-defined function basics

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Functions (general)

A program may perform the same operation repeatedly, causing a large and confusing program due to redundancy. Program redundancy can be reduced by creating a grouping of predefined statements for repeatedly used operations, known as a ***function***. Even without redundancy, functions can prevent a main program from becoming large and confusing.

PARTICIPATION ACTIVITY 7.2.1: Functions can reduce redundancy and keep the main program simple.



©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation content:

Step 1: Commonly, a program performs the same operation, such as a calculation, in multiple places. Here, the Fahrenheit to Celsius calculation is done in three places. A code block labeled Main program is displayed:

```
Main program
   c1 = (f1 - 32.0) * (5.0 / 9.0)
   c2 = (f2 - 32.0) * (5.0 / 9.0)
   c3 = (f3 + 32.0) * (5.0 / 9.0)
```

Step 2: Repeated operations clutter the main program. And such repeated operations are more prone to errors. The text, Cluttered Repeated code, appears next to Main program. The text, Oops, error in third calculation, appears under the last line of code in Main program.

Step 3: A better approach defines the Fahrenheit to Celsius calculation once, named F2C here. Then, F2C can be "called" three times, yielding a simpler main program. Two new code blocks are displayed.

The first code block contains the following code:

```
F2C(f)
   c = (f - 32.0) * (5.0 / 9.0)
   return c
```

The first code block has the text, Calculation only written once, next to it.

The second code block contains the following code:

```
Main program
   c1 = F2C(f1)
   c2 = F2C(f2)
   c3 = F2C(f3)
```

The second code block has the text, Simpler, next to it.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Step 4: The impact is even greater when the operation has multiple statements -- here 3 statements, but commonly tens of statements. The main program is much simpler. Two versions of the same code are displayed.

The first version of the code contains the following:

```
Main program
   a = expression1
   b = a + expression2
   c = b * expression3
```

d = expression1
e = d + expression2
f = e * expression3

The lines that define a, b, and c and the lines that define d, e, and f are labeled "XYZ".
The second version of the code contains the following:

```
Main program
  c = XYZ(a, b)
  f = XYZ(d, e)
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Step 5: Even without repeated operations, calling predefined operations keeps the main program simple and intuitive. Two versions of the same code are displayed.
The first version of the code contains the following:

```
Main program
  a = expression1
  b = a + expression2
  c = b * expression3

  d = expression4
  e = d * d * d
  f = (e + 1) * 2
  f = CalcPQR(d, e, g)
```

The lines that define a, b, and c are labeled "XYZ". The lines that define d, e, and f are labeled "PQR".
The second version of the code contains the following:

```
Main program
  c = XYZ(a, b)
  f = CalcPQR(d, e, g)
```

Animation captions:

1. Commonly, a program performs the same operation, such as a calculation, in multiple places. Here, the Fahrenheit to Celsius calculation is done in three places.
2. Repeated operations clutter the main program. And such repeated operations are more prone to errors.
3. A better approach defines the Fahrenheit to Celsius calculation once, named F2C here. Then, F2C can be "called" three times, yielding a simpler main program.
4. The impact is even greater when the operation has multiple statements -- here 3 statements, but commonly tens of statements. The main program is much simpler.
5. Even without repeated operations, calling predefined operations keeps the main program simple and intuitive.

PARTICIPATION ACTIVITY 7.2.2: Reasons for functions.

Consider the animation above.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

1) In the original main program, the Fahrenheit to Celsius calculation appeared how many times?
   - ○ 1
   - ○ 3

2) Along with yielding a simpler main program, using the predefined Fahrenheit to Celsius calculation prevented what error in the original program?

- Adding rather than subtracting 32.0
- Multiplying by 9.0 / 5.0 rather than by 5.0 / 9.0

3) In the last example above, the main program was simplified by _____.

- eliminating redundant code for operation XYZ
- predefining operations for XYZ and CalcPQR

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Basics of functions

A ***function*** is a named list of statements.

- A ***function definition*** consists of the new function's name and a block of statements. Ex: `double CalcPizzaArea() { /* block of statements */ }`
- A ***function call*** is an invocation of a function's name, causing the function's statements to execute.

The function's name can be any valid identifier. A ***block*** is a list of statements surrounded by braces.

Below, the function call CalcPizzaArea() causes execution to jump to the function's statements. Execution returns to the original location after executing the function's last statement.

PARTICIPATION ACTIVITY 7.2.3: Function example: Calculating pizza area.

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

double CalcPizzaArea() {
   double pizzaDiameter;
   double pizzaRadius;
   double pizzaArea;
   double piVal = 3.14159265;

   pizzaDiameter = 12.0;
   pizzaRadius = pizzaDiameter / 2.0;
   pizzaArea = piVal * pizzaRadius * pizzaRadius;
   return pizzaArea;
}

int main() {
   cout << "12 inch pizza is " << CalcPizzaArea();
   cout << " inches squared." << endl;
   return 0;
}
```

```
12 inch pizza is 113.097 inches squared.
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation content:

Static figure: A code block and an output console are displayed.
Begin C++ code:

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

double CalcPizzaArea() {
  double pizzaDiameter;
  double pizzaRadius;
  double pizzaArea;
  double piVal = 3.14159265;

  pizzaDiameter = 12.0;
  pizzaRadius = pizzaDiameter / 2.0;
  pizzaArea = piVal * pizzaRadius * pizzaRadius;
  return pizzaArea;
}

int main() {
  cout << "12 inch pizza is " << CalcPizzaArea();
  cout << " inches squared." << endl;
  return 0;
}
```

End C++ code.
Step 1: The function call CalcPizzaArea() jumps execution to the function's statements. The lines of code, cout << "12 inch pizza is " << CalcPizzaArea();, double CalcPizzaArea() {, are highlighted.
Step 2: After the last statement of the CalcPizzaArea() function, execution returns to the original location and the area of the pizza is returned and printed in main(). The lines of code, cout << "12 inch pizza is " << CalcPizzaArea();, cout << " inches squared." << endl;, are highlighted and the text, 12 inch pizza is 113.097 inches squared., is displayed in the output console.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation captions:

1. The function call CalcPizzaArea() jumps execution to the function's statements.
2. After the last statement of the CalcPizzaArea() function, execution returns to the original location and the area of the pizza is returned and printed in main().

PARTICIPATION ACTIVITY 7.2.4: Function basics.

Given the CalcPizzaArea() function defined above and the following main() function:

```
int main() {
   cout << "12 inch pizza is " << CalcPizzaArea() << " square inches" <<
endl;
   cout << "12 inch pizza is " << CalcPizzaArea() << " square inches" <<
endl;
   return 0;
}
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

1) How many function calls to CalcPizzaArea() exist in main()?

Check Show answer

2) How many function definitions of CalcPizzaArea() exist *within* main()?

Check Show answer

3) How many output statements exist in main()?

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Check Show answer

4) How many output statements exist in CalcPizzaArea()?

Check Show answer

5) Is main() itself a function definition? Answer yes or no.

Check Show answer

Returning a value from a function

A function may return one value using a ***return statement***. Below, the ComputeSquare() function is defined to have a return type of int; thus, the function's return statement must have an expression that evaluates to an int.

PARTICIPATION ACTIVITY 7.2.5: Function returns computed square.

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

int ComputeSquare(int numToSquare) {
   return numToSquare * numToSquare;
}

int main() {
   int numSquared;

   numSquared = ComputeSquare(7);
   cout << "7 squared is " << numSquared << endl;

   return 0;
}
```

```
7 squared is 49
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation content:

Static figure: A code block and an output console are displayed.
Begin C++ code:

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

int ComputeSquare(int numToSquare) {
  return numToSquare * numToSquare;
}

int main() {
  int numSquared;

  numSquared = ComputeSquare(7);
  cout << "7 squared is " << numSquared << endl;

  return 0;
}
```

End C++ code.

Step 1: ComputeSquare() is called, passing 7 to the function's numToSquare parameter. The lines of code, numSquared = ComputeSquare(7);, int ComputeSquare(int numToSquare) {, are highlighted.

Step 2: The function computes the square of the parameter numToSquare. The line of code, return numToSquare * numToSquare;, is highlighted and a 49 is displayed below the line of code.

Step 3: ComputeSquare(7) evaluates to 49, which is then assigned to numSquared. The line of code, cout << "7 squared is " << numSquared << endl;, is highlighted and the text, 7 squared is 49, is displayed in the output console.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation captions:

1. ComputeSquare() is called, passing 7 to the function's numToSquare parameter.
2. The function computes the square of the parameter numToSquare.
3. ComputeSquare(7) evaluates to 49, which is then assigned to numSquared.

Other return types are allowed, such as char, double, etc. A function can only return one item, not two or more. A return type of ***void*** indicates that a function does not return any value.

PARTICIPATION ACTIVITY 7.2.6: Return.

Given the definition below, indicate which are valid return statements:

```
int CalculateSomeValue(int num1, int num2) { ... }
```

1) `return 9;`

- Yes
- No

2) `return num1;`

- Yes
- No

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

3) `return (num1 + num2) + 1;`

- Yes
- No

4) `return;`

- Yes
- No

5) `return num1 num2;`

- Yes
- No

6) `return (0);`

- Yes
- No

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

PARTICIPATION ACTIVITY 7.2.7: More on return.

1) The following is a valid function definition:
`char GetItem() { ... }`

- True
- False

2) The following is a valid function definition for a function that returns two items:
`int, int GetItems() { ... }`

- True
- False

3) The following is a valid function definition:
`void PrintItem() { ... }`

- True
- False

Parameters

A programmer can influence a function's behavior via an input.

- A ***parameter*** is a function input specified in a function definition. Ex: A pizza area function might have diameter as an input.
- An ***argument*** is a value provided to a function's parameter during a function call. Ex: A pizza area function might be called as CalcPizzaArea(12.0) or as CalcPizzaArea(16.0).

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

A parameter is like a variable declaration. Upon a call, the parameter's memory location is allocated, and the parameter is assigned with the argument's value. Upon returning to the original call location, the parameter is deleted from memory.

An argument may be an expression, like 12.0, x, or x * 1.5.

PARTICIPATION ACTIVITY | 7.2.8: Function with parameters example: Calculating pizza area for different diameters.

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

double CalcPizzaArea(double pizzaDiameter) {
   double pizzaRadius;
   double pizzaArea;
   double piVal = 3.14159265;

   pizzaRadius = pizzaDiameter / 2.0;
   pizzaArea = piVal * pizzaRadius * pizzaRadius;
   return pizzaArea;
}

int main() {
   cout << "12.0 inch pizza is "<< CalcPizzaArea(12.0)
        << " square inches." << endl;
   cout << "16.0 inch pizza is "<< CalcPizzaArea(16.0)
        << " square inches." << endl;
   return 0;
}
```

```
12.0 inch pizza is 113.097 square inches.
16.0 inch pizza is 201.062 square inches.
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation content:

Static figure: A code block and an output console are displayed.
Begin C++ code:
#include <iostream>
using namespace std;

double CalcPizzaArea(double pizzaDiameter) {
  double pizzaRadius;
  double pizzaArea;
  double piVal = 3.14159265;

  pizzaRadius = pizzaDiameter / 2.0;
  pizzaArea = piVal * pizzaRadius * pizzaRadius;
  return pizzaArea;
}

int main() {
  cout << "12.0 inch pizza is "<< CalcPizzaArea(12.0)
     << " square inches." << endl;
  cout << "16.0 inch pizza is "<< CalcPizzaArea(16.0)
     << " square inches." << endl;
  return 0;
}
End C++ code.
Step 1: The function call CalcPizzaArea() jumps execution to the function's statements, passing 12.0 to the function's pizzaDiameter parameter, and returning the area of the pizza. The line of code, cout << "12.0 inch pizza is "<< CalcPizzaArea(12.0) << " square inches." << endl;, is highlighted and each line of code in the CalcPizzaArea() function is highlighted one by one. The text, 12.0 inch pizza is 113.097

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

square inches., is displayed in the output console.
Step 2: The next function call passes 16.0 to the function's pizzaDiameter parameter, which results in a different pizza area. The line of code, cout << "16.0 inch pizza is "<< CalcPizzaArea(16.0) << " square inches." << endl;, is highlighted and each line of code in the CalcPizzaArea() function is highlighted one by one. The text, 16.0 inch pizza is 201.062 square inches., is displayed in the output console.

Animation captions:

1. The function call CalcPizzaArea() jumps execution to the function's statements, passing 12.0 to the function's pizzaDiameter parameter, and returning the area of the pizza.
2. The next function call passes 16.0 to the function's pizzaDiameter parameter, which results in a different pizza area.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

PARTICIPATION ACTIVITY 7.2.9: Parameters.

1) Complete the function beginning to have a parameter named numServings of type int.

```
int CalcCalories(
) {
```

Check Show answer

2) Write a statement that calls a function named CalcCalories, passing the value 3 as an argument.

Check Show answer

3) Is the following a valid function definition beginning? Type yes or no.

```
int MyFct(int userNum + 5)
{ ... }
```

Check Show answer

4) Assume a function `int GetBirthdayAge(int userAge)` simply returns the value of userAge + 1. What will the following output?

```
cout << GetBirthdayAge(42);
cout << GetBirthdayAge(20);
```

Check Show answer

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Multiple or no parameters

A function definition may have multiple parameters, separated by commas. Parameters are assigned with argument values by position: First parameter with first argument, second with second, etc.

A function definition with no parameters must still have the parentheses, as in: `int DoSomething() { ... }`. The call must include parentheses, with no argument, as in: `DoSomething()`.

Figure 7.2.1: Function with multiple parameters.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

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

double CalcPizzaVolume(double pizzaDiameter, double pizzaHeight) {
   double pizzaRadius;
   double pizzaArea;
   double pizzaVolume;
   double piVal = 3.14159265;

   pizzaRadius = pizzaDiameter / 2.0;
   pizzaArea = piVal * pizzaRadius * pizzaRadius;
   pizzaVolume = pizzaArea * pizzaHeight;
   return pizzaVolume;
}

int main() {
   cout << "12.0 x 0.3 inch pizza is " << CalcPizzaVolume(12.0, 0.3);
   cout << " inches cubed." << endl;
   cout << "12.0 x 0.8 inch pizza is " << CalcPizzaVolume(12.0, 0.8);
   cout << " inches cubed." << endl;
   cout << "16.0 x 0.8 inch pizza is " << CalcPizzaVolume(16.0, 0.8);
   cout << " inches cubed." << endl;

   return 0;
}
```

```
12.0 x 0.3 inch pizza is 33.9292 inches cubed.
12.0 x 0.8 inch pizza is 90.4779 inches cubed.
16.0 x 0.8 inch pizza is 160.85 inches cubed.
```

PARTICIPATION ACTIVITY 7.2.10: Multiple parameters.

1) Which correctly defines two integer parameters x and y for a function definition:
`int CalcVal(...)`?

- (int x; int y)
- (int x, y)
- (int x, int y)

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

2) Which correctly passes two integer arguments for the function call: `CalcVal(...)`?

- (99, 44 + 5)
- (int 99, 44)
- (int 99, int 44)

3) Given a function definition:
`int CalcVal(int a, int b, int c)`
b is assigned with what value during this function call:
`CalcVal(42, 55, 77);`

- Unknown
- 42
- 55

4) Given a function definition:
`int CalcVal(int a, int b, int c)`
and given int variables i, j, and k, which are valid arguments in the call `CalcVal(...)`?

- (i, j)
- (k, i + j, 99)
- (i + j + k)

PARTICIPATION ACTIVITY 7.2.11: Calls with multiple parameters.

Given:

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

1) What will be returned for the following function call?
`GetSum(1, 2);`

Check Show answer

2) Write a statement that calls GetSum() to return the sum of x and 400 (providing the arguments in that order). End with ;

Check Show answer

©zyBooks 01/31/24 17:49 1939727 Rob Daglio MDCCOP2335Spring2024

Calling functions from functions

A function's statements may call other functions. In the example below, the PizzaCalories() function calls the CalcCircleArea() function. (Note that main() itself is the first function called when a program executes, and calls other functions.)

Figure 7.2.2: Functions calling functions.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

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

double CalcCircleArea(double circleDiameter) {
   double circleRadius;
   double circleArea;
   double piVal = 3.14159265;

   circleRadius = circleDiameter / 2.0;
   circleArea = piVal * circleRadius * circleRadius;

   return circleArea;
}

double PizzaCalories(double pizzaDiameter) {
   double totalCalories;
   double caloriesPerSquareInch = 16.7;    // Regular crust pepperoni
pizza

   totalCalories = CalcCircleArea(pizzaDiameter) * caloriesPerSquareInch;

   return totalCalories;
}


int main() {
   cout << "12 inch pizza has " << PizzaCalories(12.0) << " calories." <<
endl;
   cout << "14 inch pizza has " << PizzaCalories(14.0) << " calories." <<
endl;

   return 0;
}
```

```
12 inch pizza has 1888.73 calories.
14 inch pizza has 2570.77 calories.
```

PARTICIPATION ACTIVITY 7.2.12: Functions calling functions.

Complete the PizzaCaloriesPerSlice() function to compute the calories for a single slice of pizza. A PizzaCalories() function returns a pizza's total calories given the pizza diameter passed as an argument. A PizzaSlices() function returns the number of slices in a pizza given the pizza diameter passed as an argument.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

```
double PizzaCaloriesPerSlice(double pizzaDiameter) {
   double totalCalories;
   double caloriesPerSlice;

   totalCalories = Placeholder_A;
   caloriesPerSlice = Placeholder_B;

   return caloriesPerSlice;
}
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

1) Type the expression for Placeholder_A to compute the total calories for a pizza with diameter pizzaDiameter.

```
totalCalories =
                ;
```

Check Show answer

2) Type the expression for Placeholder_B to compute the calories per slice.

```
caloriesPerSlice =
                ;
```

Check Show answer

Exploring further:

- Functions tutorial from cplusplus.com
- Passing arguments by value and by reference from msdn.microsoft.com
- Function definition from msdn.microsoft.com
- Function call from msdn.microsoft.com

CHALLENGE ACTIVITY 7.2.1: Basic function call.

Complete the function definition to return the hours given minutes. Output for sample program when the user inputs 210.0:

```
3.5
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Learn how our autograder works

539740.3879454.qx3zqy7

```
1 #include <iostream>
2 using namespace std;
3
4 double GetMinutesAsHours(double origMinutes) {
5
```

```
6    /* Your solution goes here  */
7
8  }
9
10 int main() {
11    double minutes;
12
13    cin >> minutes;
14
15    // Will be run with 210.0, 3600.0, and 0.0.
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Run

CHALLENGE ACTIVITY 7.2.2: Enter the output of the returned value.

539740.3879454.qx3zqy7

Start

Type the program's output

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

int ChangeValue(int x) {
   return x + 4;
}

int main() {
   cout << ChangeValue(2) << endl;

   return 0;
}
```

6

1 2 3

Check Next

CHALLENGE ACTIVITY 7.2.3: Function definition: Volume of a pyramid.

Define a function CalcPyramidVolume with double data type parameters baseLength, baseWidth, and pyramidHeight, that returns as a double the volume of a pyramid with a rectangular base. Relevant geometry equations:
Volume = base area x height x 1/3
Base area = base length x base width.
(Watch out for integer division).

Learn how our autograder works

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

539740.3879454.qx3zqy7

```
1  #include <iostream>
2  using namespace std;
3
4  /* Your solution goes here  */
5
6  int main() {
7     double userLength;
8     double userWidth;
```

```
    double userWidth;
    double userHeight;

    cin >> userLength;
    cin >> userWidth;
    cin >> userHeight;

    cout << "Volume: " << CalcPyramidVolume(userLength, userWidth, userHeigh
```

Run

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

CHALLENGE ACTIVITY | 7.2.4: Functions with parameters and return values.

539740.3879454.qx3zqy7

Start

Define a function CalcVal() that has one double parameter and returns the parameter minus 4.9 if the parameter is less than Otherwise, CalcVal() returns the parameter plus 4.9.

▶ **Click here for examples**

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

/* Your code goes here */

int main() {
   double input;
   double result;

   cin >> input;

   result = CalcVal(input);

   cout << fixed << setprecision(2) << result << endl;
```

1 2

Check Next level

7.3 Print functions

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Printing from a function

A common operation for a function is to print text. Large text outputs can clutter the main() function of a program, especially if the text needs to be output multiple times. A function that only prints typically does not return a value. The ***void*** keyword indicates a function does not return a value. A function with a void return type is often called a ***void function***. Once a void function finishes execution, control returns back to the caller and no value is returned.

PARTICIPATION ACTIVITY 7.3.1: Printing with a void function.

```
void PrintSummary(int id, int items, double price) {
   cout << "Order " << id << ":" << endl;
   cout << "   Items: " << items << endl;
   cout << fixed << setprecision(2);
   cout << "   Total: $" << price << endl;
}

int main() {
   ...
   // Assume id = 42, items = 4, price = 13.99
   PrintSummary(id, items, price);

   // Continues execution
   ...
}
```

```
Order 42:
  Items: 4
  Total: $ 13.99
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation content:

Static figure: A code block and an output console are displayed.
Begin C++ code:
int main() {
  ...
  cout << "Order " << id << ":" << endl;
  cout << "   Items: " << items << endl;
  cout << fixed << setprecision(2);
  cout << "   Total: $" << price << endl;
  ...
}
End C++ code.
Step 1: Printing instructions can clutter a program. The print statements in main() are highlighted.
Step 2: A print function can handle output and reduce clutter in main(). The print statements in main() move to a new function, void PrintSummary(int id, int items, double price). The line of code, PrintSummary(id, items, price);, appears inside of main() and is highlighted.
Step 3: main() calls function PrintSummary(), which prints the parameters as formatted output. A comment appears in main() with the text, // Assume id = 42, items = 4, price = 13.99, and the line of code, PrintSummary(id, items, price);, is highlighted again. The print statements in PrintSummary() are highlighted one by one and the output console now contains three lines of text:
Order 42:
  Items: 4
  Total: $ 13.99
Step 4: PrintSummary() completes execution and returns back to the caller, main(). The comment in main(), // Continues execution, is briefly highlighted.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation captions:

1. Printing instructions can clutter a program.
2. A print function can handle output and reduce clutter in main().
3. main() calls function PrintSummary(), which prints the parameters as formatted output.
4. PrintSummary() completes execution and returns back to the caller, main().

PARTICIPATION ACTIVITY 7.3.2: Print functions.

1) Print operations must be performed in main().
   - True
   - False

2) A void function can have any number of parameters.
   - True
   - False

3) A print function must return the value that was output.
   - True
   - False

4) A void function can be defined with either "void" or no type specifier.
   - True
   - False

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

A function that produces output can also return a value, but this material separates these operations for clarity. A function that both outputs and returns a value must have a return type that is not void.

Calling a print function multiple times

One benefit of a print function is that complex output statements can be written in code once. Then the print function can be called multiple times to produce the output instead of rewriting complex statements for every necessary instance. Changes to output and formatting are made easier and are less prone to error.

PARTICIPATION ACTIVITY 7.3.3: Calling a print function repeatedly.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

```
int main() {
   ...
   // Read from input: word = "Show"
   cout << "--The Greatest *";
   cout << word;
   cout << "* on Earth!--" << endl;
   ...
   // Read from input: word = "Song"
   cout << "--The Greatest *";
   cout << word;
   cout << "* on Earth!--" << endl;
}
```

```
void PrintGreatest(string word) {
   cout << "--The Greatest *";
   cout << word;
   cout << "* on Earth!--" << endl;
}

int main() {
   ...
   // Read from input: word = "Show"
   PrintGreatest(word);
   ...
   // Read from input: word = "Song"
   PrintGreatest(word);
}
```

Program A | Program B

Animation content:

Static figure: Two code blocks are displayed.
Program A:
Begin C++ code:

```
int main() {
   ...
   // Read from input: word = "Show"
   cout << "--The Greatest *";
   cout << word;
   cout << "* on Earth!--" << endl;
   ...
}
```

End C++ code.

Program B:
Begin C++ code:

```
void PrintGreatest(string word) {
   cout << "--The Greatest *";
   cout << word;
   cout << "* on Earth!--" << endl;
}

int main() {
   ...
   // Read from input: word = "Show"
   PrintGreatest(word);
   ...
}
```

End C++ code.
Step 1: A print function can improve the organization of a program.
Step 2: The benefit of a print function increases with repeated calls. Both Program A and B output the formatted text twice, but the output code for Program B is only written once in the PrintGreatest() function. The lines of code, // Read from input: word = "Song", cout << "--The Greatest *";, cout << word;, }, are added to Program A. The line of code, // Read from input: word = "Song", PrintGreatest(word);, are added to program B.
Step 3: In Program B, any changes to the output, e.g. adding '!', only have to be made in the PrintGreatest() function The line of code, cout << "* on Earth!--" << endl;, is highlighted in Program B.
Step 4: Without a function, like in Program A, output must be changed in multiple instances, which can be time-consuming and lead to errors. The lines of code, cout << "* on Earth!--" << endl;, cout << "* on Earth!--" << endl;, are highlighted in Program A.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation captions:

1. A print function can improve the organization of a program.
2. The benefit of a print function increases with repeated calls. Both Program A and B output the formatted text twice, but the output code for Program B is only written once in the PrintGreatest() function.

3. In Program B, any changes to the output, e.g. adding '!', only have to be made in the PrintGreatest() function.
4. Without a function, like in Program A, output must be changed in multiple instances, which can be time-consuming and lead to errors.

PARTICIPATION ACTIVITY 7.3.4: Calling a print function multiple times.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Refer to the example programs above.

1) To output the phrase, "The Greatest [word] on Earth", with ten different words, what is the minimum number of times main() would need to call PrintGreatest() in Program B?
   - 1
   - 10
   - 30

2) To output "in the Galaxy" instead of "on Earth" in the phrase (Ex: "--The Greatest *Cafe* in the Galaxy!-- "), how many statements need to be changed in Program A and Program B?
   - Program A: 1
Program B: 1
   - Program A: 2
Program B: 1
   - Program A: 2
Program B: 2

Example: Menu system

Figure 7.3.1: Example: Menu System.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

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

void PrintMenu() {
   cout << "Today's Menu:" << endl;
   cout << "   1) Gumbo" << endl;
   cout << "   2) Jambalaya" << endl;
   cout << "   3) Quit" << endl <<
endl;
}

int main() {
   bool quit = false;
   int choice;

   while (!quit) {
      PrintMenu();
      cout << "Enter choice: ";
      cin >> choice;
      if (choice == 3) {
         cout << "Goodbye" << endl;
         quit = true;
      }
      else {
         cout << "Order: ";
         if (choice == 1) {
            cout << "Gumbo" << endl;
         }
         else if (choice == 2) {
            cout << "Jambalaya" <<
endl;
         }
         cout << endl;
      }
   }
   return 0;
}
```

```
Today's Menu:
   1) Gumbo
   2) Jambalaya
   3) Quit

Enter choice: 2
Order:
Jambalaya

Today's Menu:
   1) Gumbo
   2) Jambalaya
   3) Quit

Enter choice: 1
Order: Gumbo

Today's Menu:
   1) Gumbo
   2) Jambalaya
   3) Quit

Enter choice: 3
Goodbye
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

PARTICIPATION ACTIVITY 7.3.5: Example: Menu System.

Consider the example above.

1) How many times is PrintMenu() called?

- 1
- 2
- 3

2) Which of the following code statements if added to PrintMenu() would produce a compiler error?

- int numOptions;
- return 0;
- return;

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Exploring further:

- Function definition from msdn.microsoft.com
- Function call from msdn.microsoft.com

CHALLENGE ACTIVITY 7.3.1: Function parameters.

539740.3879454.qx3zqy7

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Start

Type the program's output

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

void PrintAge(int userAge) {
   cout << "I am " << userAge << endl;
}

int main() {
   int ageToPrint;

   ageToPrint = 23;
   PrintAge(ageToPrint);

   return 0;
}
```

```
I am 23
```

1 2 3 4 5

Check Next

CHALLENGE ACTIVITY 7.3.2: Function call with parameter: Printing formatted measurement.

Define a function PrintFeetInchShort(), with int parameters numFeet and numInches, that prints using ' and " shorthand. End with a newline. Remember that outputting 'endl' outputs a newline. Ex: PrintFeetInchShort(5, 8) prints:

```
5' 8"
```

Hint: Use \" to print a double quote.

Learn how our autograder works

539740.3879454.qx3zqy7

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

```
1  #include <iostream>
2  using namespace std;
3
4  /* Your solution goes here  */
5
6  int main() {
7     int userFeet;
8     int userInches;
9
10    cin >> userFeet;
11    cin >> userInches;
12
13    PrintFeetInchShort(userFeet, userInches);  // Will be run with (5, 8), t
```

```
14
15    return 0;
```

Run

CHALLENGE ACTIVITY | 7.3.3: Print functions.

539740.3879454.qx3zqy7

Start

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Define a function PrintEmployeeSalary() that has one string parameter and one integer parameter and outputs as follows. Er with a newline. PrintEmployeeSalary() should not return any value.

Ex: If the input is `Eli 85000`, then the output is:

```
Eli earns 85000 dollars per year.
```

```
1  #include <iostream>
2  using namespace std;
3
4  /* Your code goes here */
5
6  int main() {
7     string employeeName;
8     int employeeSalary;
9
10    cin >> employeeName;
11    cin >> employeeSalary;
12
13    PrintEmployeeSalary(employeeName, employeeSalary);
14
15    return 0;
```

1 2 3 4

Check Next level

7.4 Writing mathematical functions

Mathematical functions

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

A function is commonly defined to compute a mathematical calculation involving several numerical parameters and returning a numerical result. The program below uses a function to convert a person's height in U.S. units (feet and inches) into total centimeters.

Figure 7.4.1: Program with a function to convert height in feet/inches to centimeters.

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

/* Converts a height in feet/inches to centimeters */
double HeightFtInToCm(int heightFt, int heightIn) {
   const double CM_PER_IN = 2.54;
   const int IN_PER_FT = 12;
   int totIn;
   double cmVal;

   totIn = (heightFt * IN_PER_FT) + heightIn; // Total
inches
   cmVal = totIn * CM_PER_IN;                 // Conv inch
to cm
   return cmVal;
}

int main() {
   int userFt;  // User defined feet
   int userIn;  // User defined inches

   // Prompt user for feet/inches
   cout << "Enter feet: ";
   cin >> userFt;

   cout << "Enter inches: ";
   cin >> userIn;

   // Output the conversion result
   cout << "Centimeters: ";
   cout << HeightFtInToCm(userFt, userIn) << endl;

   return 0;
}
```

```
Enter feet: 5
Enter inches: 8
Centimeters:
172.72
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Human average height is increasing, attributed to better nutrition. Source: Our World in Data: Human height.

PARTICIPATION ACTIVITY 7.4.1: Mathematical functions.

Indicate which is a valid use of the HeightFtInToCm() function above. x is type double.

1) x = HeightFtInToCm(5, 0);

- ○ Valid
- ○ Not valid

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

2) x = 2 * (HeightFtInToCm(5, 0) + 1.0);

- ○ Valid
- ○ Not valid

3) x = (HeightFtInToCm(5, 0) + HeightFtInToCm(6, 1)) / 2.0;

- Valid
- Not valid

4) Suppose `int pow(int y, int z)` returns y to the power of z. Is the following valid?
x = pow(2, pow(3, 2));

- Valid
- Not valid

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

zyDE 7.4.1: Temperature conversion.

Complete the program by writing and calling a function that converts a temperature from Celsius into Fahrenheit using the formula:

$$Fahrenheit = \frac{9}{5} * Celsius + 32$$

Load default template...

```
1  #include <iostream>
2  using namespace std;
3
4
5  // FINISH: Define Celsiu
6
7
8  int main() {
9     double tempF;
10    double tempC;
11
12    cout << "Enter temper
13    cin >> tempC;
14
15    tempF = 0.0; // FIXM
```

100

Run

Calling functions in expressions

A function call evaluates to the returned value. Thus, a function call often appears within an expression. Ex: `5 + computeSquare(4)` evaluates to 5 + 16, or 21.

A function with a void return type cannot be used within an expression, instead being used in a statement like: `outputData(x, y);`

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

PARTICIPATION ACTIVITY 7.4.2: Function called twice in an expression.

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

int ComputeSquare(int numToSquare) {
```

7 squared plus 9 squared is 130

```
   return numToSquare * numToSquare;
}

int main() {
   int c2;

   c2 = ComputeSquare(7) + ComputeSquare(9);
   cout << "7 squared plus 9 squared is "
        << c2 << endl;
   return 0;
}
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation content:

Static figure: A code block and an output console are displayed.
Begin C++ code:
#include <iostream>
using namespace std;

int ComputeSquare(int numToSquare) {
  return numToSquare * numToSquare;
}

int main() {
  int c2;

  c2 = ComputeSquare(7) + ComputeSquare(9);
  cout << "7 squared plus 9 squared is "
     << c2 << endl;
  return 0;
}
End C++ code.
Step 1: ComputeSquare() is called within an expression two times, first with the argument 7. 7 is passed to ComputeSquare() the value 49 is returned. The line of code, c2 = ComputeSquare(7) + ComputeSquare(9);, is highlighted and the ComputeSquare function is called with the argument 7. The lines of code, int ComputeSquare(int numToSquare) {, return numToSquare * numToSquare;, }, are briefly highlighted and 49 is returned.
Step 2: ComputeSquare() is called a second time with the argument 9 and the value 81 is returned to the expression. The lines of code, int ComputeSquare(int numToSquare) {, return numToSquare * numToSquare;, }, are briefly highlighted and 81 is returned.
Step 3: The expression then evaluates to c2 = 49 + 81, which assigns variable c2 with 130. Lastly, the print statement executes. The line of code, cout << "7 squared plus 9 squared is " << c2 << endl;, is highlighted and the text, 7 squared plus 9 squared is 130, is displayed in the output console.

Animation captions:

1. ComputeSquare() is called within an expression two times, first with the argument 7. 7 is passed to ComputeSquare() the value 49 is returned.
2. ComputeSquare() is called a second time with the argument 9 and the value 81 is returned to the expression.
3. The expression then evaluates to c2 = 49 + 81, which assigns variable c2 with 130. Lastly, the print statement executes.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

PARTICIPATION ACTIVITY 7.4.3: Calls in an expression.

Given the definitions below, which are valid statements?

```
double SquareRoot(double x) { ... }
void PrintVal(double x) { ... }
double y;
```

1) `y = SquareRoot(49.0);`
   - Valid
   - Invalid

2) `SquareRoot(49.0) = z;`
   - Valid
   - Invalid

3) `y = 1.0 + SquareRoot(144.0);`
   - Valid
   - Invalid

4) `y = SquareRoot(SquareRoot(16.0));`
   - Valid
   - Invalid

5) `y = SquareRoot();`
   - Valid
   - Invalid

6) `SquareRoot(9.0);`
   - Valid
   - Invalid

7) `y = PrintVal(9.0);`
   - Valid
   - Invalid

8) `PrintVal(9.0);`
   - Valid
   - Invalid

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Modular functions for mathematical expressions

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Modularity allows more complex functions to incorporate simpler functions. Complex mathematical functions often call other mathematical functions. Ex: A function that calculates the volume or surface area of a cylinder calls a function that returns the area of the cylinder's base, which is needed for both calculations.

Figure 7.4.2: Program that calculates cylinder volume and surface area by calling a modular function for the cylinder's base.

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

double CalcCircularBaseArea(double radius) {
   return M_PI * radius * radius;
}

double CalcCylinderVolume(double baseRadius, double
height) {
   return CalcCircularBaseArea(baseRadius) * height;
}

double CalcCylinderSurfaceArea(double baseRadius, double
height) {
   return (2 * M_PI * baseRadius * height) + (2 *
CalcCircularBaseArea(baseRadius));
}

int main() {
   double radius;  // User defined radius
   double height;  // User defined height

   // Prompt user for radius
   cout << "Enter base radius: ";
   cin >> radius;

   // Prompt user for height
   cout << "Enter height: ";
   cin >> height;

   // Output the cylinder volume result
   cout << "Cylinder volume: ";
   cout << CalcCylinderVolume(radius, height) << endl;

   // Output the cylinder surface area result
   cout << "Cylinder surface area: ";
   cout << CalcCylinderSurfaceArea(radius, height) <<
endl;

   return 0;
}
```

```
Enter base radius:
10
Enter height: 5
Cylinder volume:
1570.8
Cylinder surface
area: 942.478
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

CHALLENGE ACTIVITY 7.4.1: Writing mathematical functions.

539740.3879454.qx3zqy7

Start

Complete the function MillilitersToTablespoons() that takes one integer parameter as a volume in milliliters. The function ret a double as the volume converted to tablespoons, given that 1 milliliter = 0.067628 tablespoons.

Ex: If the input is `148`, then the output is:

```
10.0089 tablespoons
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

```
1 #include <iostream>
2 using namespace std;
3
4 double MillilitersToTablespoons(int milliliters) {
5
6    /* Your code goes here */
7
8 }
```

```
 9
10 int main() {
11    int userMilliliters;
12
13    cin >> userMilliliters;
14
15    cout << MillilitersToTablespoons(userMilliliters);
```

1 2 3

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Check Next level

7.5 Functions with branches

Example: Shipping cost calculator

A function's statements may include branches and other statements. The following example uses a function to calculate a package's shipping cost based on weight.

Figure 7.5.1: Shipping cost calculator.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

```
#include <iostream>
#include <iomanip>

using namespace std;

double CalcTax(double cost) {
   return cost * 0.15;
}

// Determine shipping cost based on weight
double CalcShippingCost(double weight) {
   double cost;

   if (weight < 1) {
      cost = 7.88;
   }
   else if (weight < 6) {
      cost = 14.32;
   }
   else if (weight < 10) {
      cost = 21.11;
   }
   else {
      cost = 25.5;
   }
   cost = cost + CalcTax(cost);
   return cost;
}

int main() {
   double weightOfPackage;        // User defined package
weight

   cout << "Enter package weight: ";
   cin >> weightOfPackage;
   cout << "Shipping cost: $";
   cout << fixed << setprecision(2) <<
CalcShippingCost(weightOfPackage) << endl;
   return 0;
}
```

```
Enter package
weight: 6
Shipping cost:
$24.28

...

Enter package
weight: 10.5
Shipping cost:
$29.33


...

Enter package
weight: 3.0
Shipping cost:
$16.47
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

PARTICIPATION ACTIVITY 7.5.1: Analyzing the shipping cost calculator.

1) For a package weight of 7.5 lbs, what is the cost returned by CalcShippingCost? Type as #.##

Check Show answer

2) When main() is executed, which user defined function is called first?

Check Show answer

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Example: Auction website fee calculator

The following example uses a function to compute the fee charged by eBay when a customer sells an item online.

Figure 7.5.2: Function example: Determining fees given an item selling price for an auction website.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

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

/* Returns fee charged by ebay.com given the selling
   price of fixed-price books, movies, music, or
video-games.
   Fee is $0.50 to list plus a % of the selling
price:
   13% for $50.00 or less
    plus 5% for $50.01 to $1000.00
    plus 2% for $1000.01 or more
   Source: http://pages.ebay.com/help/sell/fees.html,
2012.

   Note: double variables often are not used for
dollars/cents,
   but here the dollar fraction may extend past two
decimal places.
*/

// Function determines eBay price given item selling
price
double CalcEbayFee(double sellPrice) {
   const double BASE_LIST_FEE     = 0.50; // Listing
Fee
   const double PERC_50_OR_LESS   = 0.13; // % $50 or
less
   const double PERC_50_TO_1000   = 0.05; // %
$50.01..$1000.00
   const double PERC_1000_OR_MORE = 0.02; // %
$1000.01 or more
   double feeTotal;                       //
Resulting eBay fee

   feeTotal = BASE_LIST_FEE;

   // Determine additional fee based on selling price
   if (sellPrice <= 50.00) { // $50.00 or lower
      feeTotal = feeTotal + (sellPrice *
PERC_50_OR_LESS);
   }
   else if (sellPrice <= 1000.00) { //
$50.01..$1000.00
      feeTotal = feeTotal + (50 * PERC_50_OR_LESS )
      + ((sellPrice - 50) * PERC_50_TO_1000);
   }
   else { // $1000.01 and higher
      feeTotal = feeTotal + (50 * PERC_50_OR_LESS)
      + ((1000 - 50) * PERC_50_TO_1000)
      + ((sellPrice - 1000) * PERC_1000_OR_MORE);
   }

   return feeTotal;
}

int main() {
   double sellingPrice;  // User defined selling
price

   cout << "Enter item selling price (Ex: 65.00): ";
   cin >> sellingPrice;

   cout << "eBay fee: $" << CalcEbayFee(sellingPrice)
<< endl;

   return 0;
}
```

```
Enter item selling price
(Ex: 65.00): 9.95
eBay fee: $1.7935

...

Enter item selling price
(Ex: 65.00): 40
eBay fee: $5.7

...

Enter item selling price
(Ex: 65.00): 100
eBay fee: $9.5

...

Enter item selling price
(Ex: 65.00): 500.15
eBay fee: $29.5075

...

Enter item selling price
(Ex: 65.00): 2000
eBay fee: $74.5
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

PARTICIPATION ACTIVITY 7.5.2: Analyzing the eBay fee calculator.

1) For any call to CalcEbayFee() , how many assignment statements for the variable `feeTotal` will execute?

Check Show answer

2) What does CalcEbayFee() return if the sellPrice argument is 0.0 (show your answer in the form #.##)?

Check Show answer

3) What does CalcEbayFee() return if the sellPrice argument is 100.00 (show your answer in the form #.##)?

Check Show answer

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

CHALLENGE ACTIVITY 7.5.1: Output of functions with branches.

539740.3879454.qx3zqy7

Start

Type the program's output

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

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

int NormalizeGrade(int grade) {
   int upperBound = 72;
   int lowerBound = 42;

   if (grade > upperBound) {
      return upperBound;
   }
   else if (grade < lowerBound){
      return lowerBound;
   }
   else {
      return grade;
   }
}

int main() {
   int grade1 = NormalizeGrade(22);
   int grade2 = NormalizeGrade(65);
   int grade3 = NormalizeGrade(94);

   cout << grade1 << endl;
   cout << grade2 << endl;
   cout << grade3 << endl;

   return 0;
}
```

```
42
65
72
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

1 2

Check Next

CHALLENGE ACTIVITY 7.5.2: Function with branch: Popcorn.

Complete function PrintPopcornTime(), with int parameter bagOunces, and void return type. If bagOunces is less than 2, print "Too small". If greater than 10, print "Too large". Otherwise, compute and print 6 * bagOunces followed by " seconds". End with a newline. Example output for ounces = 7:

```
42 seconds
```

Learn how our autograder works

539740.3879454.qx3zqy7

```
1  #include <iostream>
2  using namespace std;
3
4  void PrintPopcornTime(int bagOunces) {
5
6     /* Your solution goes here  */
7
8  }
9
10 int main() {
11    int userOunces;
12
13    cin >> userOunces;
14    PrintPopcornTime(userOunces);
15
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Run

CHALLENGE ACTIVITY 7.5.3: Functions with branches.

539740.3879454.qx3zqy7

Start

Define a function FindPrize() that takes one integer parameter as the lottery number, and returns the prize as an integer. The is returned as follows:

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

- If the lottery number is 193 or 387, then the prize is $17000.
- If the lottery number is 620 or 914, then the prize is $13500.
- Otherwise, the prize is $0.

Ex: If the input is `193`, then the output is:

```
17000
```

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

/* Your code goes here */

int main() {
   int number;

   cin >> number;

   cout << FindPrize(number) << endl;

   return 0;
}
```

1 2

Check Next level

7.6 Functions with loops

Example: Computing the average of a list of numbers

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

The following example uses a function with a for loop to calculate the average of a list of numbers.

Figure 7.6.1: Computing the average of a list of numbers.

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

double ComputeAverage(int numCount) {
   double valuesSum = 0;
   int currValue = 0;

   for (int i = 0; i < numCount; ++i) {
      cout << "Enter number: ";
      cin >> currValue;
      valuesSum += currValue;
   }
   return valuesSum / numCount;
}

int main() {
   int numValues;
   double averageVal;

   cout << "Enter number of values: ";
   cin >> numValues;
   averageVal = ComputeAverage(numValues);

   cout << "Average: ";
   cout << fixed << setprecision(3) << averageVal <<
endl;
   return 0;
}
```

```
Enter number of values:
3
Enter number: 10
Enter number: 5
Enter number: 5
Average: 6.667
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

PARTICIPATION ACTIVITY 7.6.1: Analyzing the compute average program.

1) How many calls to user-defined functions exist in main()?

Check Show answer

2) What does ComputeAverage() return if the input is 4 2 3 4 5? Type as #.#. Note: The first number is the number of values in the list.

Check Show answer

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Example: Least-common multiple calculator

The following is another example with user-defined functions. The functions keep main()'s behavior readable and understandable.

Figure 7.6.2: User-defined functions make main() easy to understand.

```
Enter value for first input
Enter a positive number (>0):
13

Enter value for second input
Enter a positive number (>0):
7

Least common multiple of 13
and 7 is 91
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

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

// Function prompts user to enter positive non-
zero number
int PromptForPositiveNumber() {
   int userNum;

   userNum = 0;

   while (userNum <= 0) {
      cout << "Enter a positive number (>0): " <<
endl;
      cin >> userNum;

      if (userNum <= 0) {
         cout << "Invalid number." << endl;
      }
   }

   return userNum;
}


// Function returns greatest common divisor of
two inputs
int FindGCD(int aVal, int bVal) {
   int numA;
   int numB;

   numA = aVal;
   numB = bVal;

   while (numA != numB) { // Euclid's algorithm
      if (numB > numA) {
         numB = numB - numA;
      }
      else {
         numA = numA - numB;
      }
   }

   return numA;
}

// Function returns least common multiple of two
inputs
int FindLCM(int aVal, int bVal) {
   int lcmVal;

   lcmVal = abs(aVal * bVal) / FindGCD(aVal,
bVal);

   return lcmVal;
}

int main() {
   int usrNumA;
   int usrNumB;
   int lcmResult;

   cout << "Enter value for first input" << endl;
   usrNumA = PromptForPositiveNumber();

   cout << endl << "Enter value for second input"
<< endl;
   usrNumB = PromptForPositiveNumber();

   lcmResult = FindLCM(usrNumA, usrNumB);

   cout << endl << "Least common multiple of " <<
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

```
usrNumA
        << " and " << usrNumB << " is " <<
lcmResult << endl;

   return 0;
}
```

PARTICIPATION ACTIVITY 7.6.2: Analyzing the least common multiple program.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

1) Other than main(), which user-defined function calls another user-defined function? Just write the function name.

Check Show answer

2) How many user-defined function calls exist in the program code? Note: abs() is not a user-defined function.

Check Show answer

CHALLENGE ACTIVITY 7.6.1: Output of functions with loops.

539740.3879454.qx3zqy7

Start

Type the program's output

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

int Compute(int val1, int val2) {
   int result = 1;
   int i;

   for (i = 0; i < val1; ++i) {
      result *= val2 + 3;
   }

   return result;
}

int main() {
   int value1 = 4;
   int value2 = 1;
   int computedValue;

   computedValue = Compute(value1, value2);
   cout << computedValue << endl;

   return 0;
}
```

256

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

1 2

Check Next

CHALLENGE ACTIVITY 7.6.2: Functions with loops.

539740.3879454.qx3zqy7

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Start

Define a function PrintVal() that takes two integer parameters and outputs the sum of all integers starting with the first and ending with the second parameter, followed by a newline. The function does not return any value.

Ex: If the input is `1 5`, then the output is:

```
15
```

Note: Assume the first integer parameter is less than the second.

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

/* Your code goes here */

int main() {
   int num1;
   int num2;

   cin >> num1;
   cin >> num2;
   PrintVal(num1, num2);

   return 0;
}
```

1 2 3 4

Check Next level

7.7 Scope of variable/function definitions

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

The name of a defined variable or function item is only visible to part of a program, known as the item's ***scope***. A variable declared in a function has scope limited to inside that function. In fact, because a compiler scans a program line-by-line from top-to-bottom, the scope starts *after* the declaration until the function's end. The following highlights the scope of local variable cmVal.

Figure 7.7.1: Local variable scope.

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

const double CM_PER_IN = 2.54;
const int    IN_PER_FT = 12;

/* Converts a height in feet/inches to centimeters */
double HeightFtInToCm(int heightFt, int heightIn) {
   int totIn;
   double cmVal;

   totIn = (heightFt * IN_PER_FT) + heightIn; // Total
inches
   cmVal = totIn * CM_PER_IN;                 // Conv inch
to cm
   return cmVal;
}

int main() {
   int userFt;     // User defined feet
   int userIn;     // User defined inches

   // Prompt user for feet/inches
   cout << "Enter feet: ";
   cin >> userFt;

   cout << "Enter inches: ";
   cin >> userIn;

   // Output the conversion result
   cout << "Centimeters: ";
   cout << HeightFtInToCm(userFt, userIn) << endl;

   return 0;
}
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Note that variable cmVal is invisible to the function main(). A statement in main() like `newLen = cmVal;` would yield a compiler error, e.g., the "error: cmVal was not declared in this scope". Likewise, variables userFt and userIn are invisible to the function HeightFtInToCm(). Thus, a programmer is free to define items with names userFt or userIn in function HeightFtInToCm().

A variable declared outside any function is called a ***global variable***, in contrast to a *local variable* declared inside a function. A global variable's scope extends after the declaration to the file's end, and reaches into functions. For example, HeightFtInToCm() above accesses global variables CM_PER_IN and IN_PER_FT.

Global variables should be used sparingly. If a function's local variable (including a parameter) has the same name as a global variable, then in that function the name refers to the local item and the global is inaccessible. Such naming can confuse a reader. Furthermore, if a function updates a global variable, the function has effects that go beyond its parameters and return value, known as ***side effects***, which make program maintenance hard. Global variables are typically limited to const variables like the number of centimeters per inch above. Beginning programmers sometimes use globals to avoid having to use parameters, which is bad practice. *Good practice is to minimize the use of non-const global variables.*

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

PARTICIPATION ACTIVITY 7.7.1: Variable/function scope.

1) A local variable is declared inside a function, while a global is declared outside any function.
   - ○ True
   - ○ False

2) A local variable's scope extends from a function's opening brace to the function's closing brace.

- True
- False

3) If a programmer declares a function's local variable to have the same name as a function parameter, the name will refer to the local variable.

- True
- False

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

4) If a programmer declares a function's local variable to have the same name as a global variable, the name will refer to the local variable.

- True
- False

5) A function that changes the value of a global variable is sometimes said to have "side effects".

- True
- False

A function also has scope, which extends from its definition to the end of the file. Commonly, a programmer wishes to have the main() definition appear near the top of a file, with other functions definitions appearing further below, so that the main function is the first thing a reader sees. However, given function scope, main() would not be able to call any of those other functions. A solution involves function declarations. A ***function declaration*** specifies the function's return type, name, and parameters, ending with a semicolon where the opening brace would have gone. A function declaration is also known as a ***function prototype***. The function declaration gives the compiler enough information to recognize valid calls to the function. So by placing function declarations at the top of a file, the main function can then appear next, with actual function definitions appearing later in the file.

Figure 7.7.2: A function declaration allows a function definition to appear later in a file.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

```
#include <iostream>
#include <cmath> // To use "pow" function
using namespace std;

/* Program to convert given-year U.S. dollars to
   current dollars, using simplistic method of 4% annual inflation.
   Source: http://inflationdata.com (See: Historical) */

// (Function DECLARATION)
double ToCurrDollars (double pastDol, int pastYr, int currYr);

int main() {
   double pastDol;        // Starting dollar amount
   double currDol;        // Ending dollar amount (converted value)
   int pastYr;            // Starting year
   int currYr;            // Ending year (converted to year)

   // Prompt user for previous year/dollar and current year
   cout << "Enter current year: ";
   cin >> currYr;
   cout << "Enter past year: ";
   cin >> pastYr;
   cout << "Enter past dollars (Ex: 1000): ";
   cin >> pastDol;

   // Function call to convert past to current dollars
   currDol = ToCurrDollars(pastDol, pastYr, currYr);

   cout << "$" << pastDol << " in " << pastYr;
   cout << " is about $" << currDol << " in ";
   cout << currYr << endl;

   return 0;
}

// (Function DEFINITION)
// Function returns equivalent value of pastDol in pastYr to currYr
double ToCurrDollars (double pastDol, int pastYr, int currYr) {
   double currDol;      // Equivalent dollar amount given inflation

   currDol = pastDol * pow(1.04, currYr - pastYr );

   return currDol;
}
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

```
Enter current year: 2015
Enter past year: 1970
Enter past dollars (Ex: 1000): 10000
$10000 in 1970 is about $58411.8 in 2015
(Note: Average annual U.S. income in 1970)

...

Enter current year: 2015
Enter past year: 1970
Enter past dollars (Ex: 1000): 23000
$23000 in 1970 is about $134347 in 2015
(Note: Average U.S. house price in 1970)

...

Enter current year: 2015
Enter past year: 1933
Enter past dollars (Ex: 1000): 37
$37 in 1933 is about $922.435 in 2015
(Note: Cost of Golden Gate Bridge, in millions)

...

Enter current year: 2015
Enter past year: 1969
Enter past dollars (Ex: 1000): 25
$25 in 1969 is about $151.871 in 2015
(Note: Cost of Apollo space program, in billions)
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

A common error is for the function definition to not match the function declaration, such as a parameter defined as double in the declaration but as int in the definition, or with a slightly different identifier. The compiler detects such errors.

PARTICIPATION ACTIVITY 7.7.2: Function declaration and definition.

1) A function declaration lists the contents of a function, while a function definition just specifies the function's interface.
   - True
   - False

2) A function declaration enables calls to the function before the function definition.
   - True
   - False

Exploring further:

- More on Scope from msdn.microsoft.com

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

7.8 Unit testing (functions)

Testing is the process of checking whether a program behaves correctly. Testing a large program can be hard because bugs may appear anywhere in the program, and multiple bugs may interact. *Good practice is to test small parts of the program individually, before testing the entire program, which can more readily support finding and fixing bugs.* ***Unit testing*** is the

process of individually testing a small part or unit of a program, typically a function. A unit test is typically conducted by creating a ***testbench***, a.k.a. test harness, which is a separate program whose sole purpose is to check that a function returns correct output values for a variety of input values. Each unique set of input values is known as a ***test vector***.

Consider a function HrMinToMin() that converts time specified in hours and minutes to total minutes. The figure below shows a test harness that tests that function. The harness supplies various input vectors like (0,0), (0,1), (0,99), (1,0), etc.

Figure 7.8.1: Test harness for the function HrMinToMin().

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

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

// Function converts hrs/min to min
int HrMinToMin(int origHours, int origMinutes) {
   int totMinutes; // Resulting minutes

   totMinutes = (origHours * 60) + origMinutes;

   return origMinutes;
}

int main() {

   cout << "Testing started" << endl;

   cout << "0:0, expecting 0, got: "    <<
HrMinToMin(0, 0)  << endl;
   cout << "0:1, expecting 1, got: "    <<
HrMinToMin(0, 1)  << endl;
   cout << "0:99, expecting 99, got: "  <<
HrMinToMin(0, 99) << endl;
   cout << "1:0, expecting 60, got: "   <<
HrMinToMin(1, 0)  << endl;
   cout << "5:0, expecting 300, got: "  <<
HrMinToMin(5, 0)  << endl;
   cout << "2:30, expecting 150, got: " <<
HrMinToMin(2, 30) << endl;
   // Many more test vectors would be typical...

   cout << "Testing completed" << endl;

   return 0;
}
```

```
Testing started
0:0, expecting 0, got:
0
0:1, expecting 1, got:
1
0:99, expecting 99,
got: 99
1:0, expecting 60, got:
0
5:0, expecting 300,
got: 0
2:30, expecting 150,
got: 30
Testing completed
```

Manually examining the program's printed output reveals that the function works for the first several vectors, but fails on the next several vectors, highlighted with colored background. Examining the output, one may note that the output minutes is the same as the input minutes; examining the code indeed leads to noticing that parameter origMinutes is being returned rather than variable totMinutes. Returning totMinutes and rerunning the test harness yields correct results.

Each bug a programmer encounters can improve a programmer by teaching him/her to program differently, just like getting hit a few times by an opening door teaches a person not to stand near a closed door.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

PARTICIPATION ACTIVITY 7.8.1: Unit testing.

1) A test harness involves temporarily modifying an existing program to test a particular function within that program.

- True
- False

2) Unit testing means to modify function inputs in small steps known as units.

- True
- False

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Manually examining a program's printed output is cumbersome and error prone. A better test harness would only print a message for incorrect output. The language provides a compact way to print an error message when an expression evaluates to false. assert() is a macro (similar to a function) that prints an error message and exits the program if assert()'s input expression is false. The error message includes the current line number and the expression (a nifty trick enabled by using a macro rather than an actual function; details are beyond our scope). Using assert requires first including the cassert library, part of the standard library, as shown below.

Figure 7.8.2: Test harness with assert for the function HrMinToMin().

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

double HrMinToMin(int origHours, int origMinutes) {
   int totMinutes;  // Resulting minutes

   totMinutes = (origHours * 60) + origMinutes;

   return origMinutes;
}

int main() {

   cout << "Testing started" << endl;

   assert(HrMinToMin(0, 0)  == 0);
   assert(HrMinToMin(0, 1)  == 1);
   assert(HrMinToMin(0, 99) == 99);
   assert(HrMinToMin(1, 0)  == 60);
   assert(HrMinToMin(5, 0)  == 300);
   assert(HrMinToMin(2, 30) == 150);
   // Many more test vectors would be typical...

   cout << "Testing completed" << endl;

   return 0;
}
```

```
Testing started
Assertion failed: (HrMinToMin(1, 0) == 60), function main, file main.cpp, line 20.
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Using branches for unit tests

If you have studied branches, you may recognize that each assert statement in main() could be replaced by an if statement like:

```
if ( HrMinToMin(0, 0) != 0 ) {
   cout << "0:0, expecting 0, got: " << HrMinToMin(0, 0) << endl;
}
```

But the assert is more compact.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

assert() enables compact readable test harnesses, and also eases the task of examining the program's output for correctness; a program without detected errors would simply output "Testing started" followed by "Testing completed".

A programmer should choose test vectors that thoroughly exercise a function. Ideally the programmer would test all possible input values for a function, but such testing is simply not practical due to the large number of possibilities -- a function with one integer input has over 4 billion possible input values, for example. Good test vectors include a number of normal cases that represent a rich variety of typical input values. For a function with two integer inputs as above, variety might include mixing small and large numbers, having the first number large and the second small (and vice-versa), including some 0 values, etc. Good test vectors also include ***border cases*** that represent fringe scenarios. For example, border cases for the above function might include inputs 0 and 0, inputs 0 and a huge number like 9999999 (and vice-versa), two huge numbers, a negative number, two negative numbers, etc. The programmer tries to think of any extreme (or "weird") inputs that might cause the function to fail. For a simple function with a few integer inputs, a typical test harness might have dozens of test vectors. For brevity, the above examples had far fewer test vectors than typical.

PARTICIPATION ACTIVITY 7.8.2: Assertions and test cases.

1) Using assert() is a preferred way to test a function.
   - True
   - False

2) For a function, border cases might include 0, a very large negative number, and a very large positive number.
   - True
   - False

3) For a function with three integer inputs, about 3-5 test vectors is likely sufficient for testing purposes.
   - True
   - False

4) A good programmer takes the time to test all possible input values for a function.
   - True
   - False

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Exploring further:

- assert reference page from cplusplus.com

CHALLENGE ACTIVITY 7.8.1: Unit testing.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Add two more statements to main() to test inputs 3 and -1. Use print statements similar to the existing one (don't use assert).

Learn how our autograder works

539740.3879454.qx3zqy7

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

// Function returns origNum cubed
int CubeNum(int origNum) {
   return origNum * origNum * origNum;
}

int main() {

   cout << "Testing started" << endl;

   cout << "2, expecting 8, got: " << CubeNum(2) << endl;

   /* Your solution goes here  */
```

Run

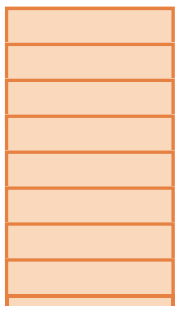
7.9 How functions work

Each function call creates a new set of local variables, forming part of what is known as a ***stack frame***. A return causes those local variables to be discarded.

PARTICIPATION ACTIVITY 7.9.1: Function calls and returns.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

```
int FtInToIn(int inFeet, int inInches) {
   int totInches;
   ...
   return totInches;
}

double FtInToCm(int inFeet, int inInches) {
   int totIn;
   double totCm;
   ...
   totIn = FtInToIn(inFeet, inInches);
   ...
   return totCm;
}
```



```
int main() {
   int userFt;
   int userIn;
   int userCm;
   ...
   userCm = FtInToCm(userFt, userIn);
   ...
   return 0;
}
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation content:

Static Figure:
Being C++ code:

```
int FtInToIn(int inFeet, int inInches) {
  int totInches;
  ...
  return totInches;
}

double FtInToCm(int inFeet, int inInches) {
  int totIn;
  double totCm;
  ...
  totIn = FtInToIn(inFeet, inInches);
  ...
  return totCm;
}

int main() {
  int userFt;
  int userIn;
  int userCm;
  ...
  userCm = FtInToCm(userFt, userIn);
  ...
  return 0;
}
```

End C++ code.
A memory space containing 12 empty cells is shown.

Step 1: Each function call creates a new set of local variables.
The C++ code is highlighted into three separate section. The memory space is also separated into three separate sections, indicating the spaces needed for the local variable within the C++ code.
The first section highlighted is the main function and is highlighted in purple:

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

```
int main() {
  int userFt;
  int userIn;
  int userCm;
  ...
  userCm = FtInToCm(userFt, userIn);
  ...
  return 0;
}
```

The first three memory cells are highlighted in purple showing that local variables, “userFt”, “userIn”, and “userCm” are set to these three cells.
The second section highlighted is the FtInToCm() function and is highlighted in blue:

```
double FtInToCm(int inFeet, int inInches) {
  int totIn;
  double totCm;
  ...
  totIn = FtInToIn(inFeet, inInches);
  ...
  return totCm;
}
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

The fifth through the eighth memory cells are highlighted in blue showing that local variables, “inFeet”, “inInches”, “totIn”, and “totCm” are set to these four cells.
The third section highlighted is the FtInToIn() function and is highlighted in green:

```
int FtInToIn(int inFeet, int inInches) {
  int totInches;
  ...
  return totInches;
}
```

The eleventh, twelfth, and thirteenth memory cells are highlighted in green showing that local variables, “inFeet”, “inInches”, and “totInches” are set to these three cells.

Step 2: Each return causes those local variables to be discarded.
Once the functions execute the return line of code, the memory space is freed, removing the local variables.

Animation captions:

1. Each function call creates a new set of local variables.
2. Each return causes those local variables to be discarded.

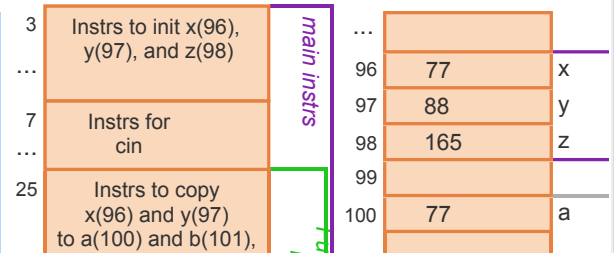
Some knowledge of how a function call and return works at the assembly level can not only satisfy curiosity, but can also lead to fewer mistakes when parameter and return items become more complex. The following animation illustrates by showing, for a function named FindMax(), some sample high-level code, compiler-generated assembly instructions in memory, and data in memory during runtime. This animation presents advanced material intended to provide insight and appreciation for how a function call and return works.

The compiler generates instructions to copy arguments to parameter local variables, and to store a return address. A jump instruction jumps from main to the function's instructions. The function executes and stores results in a designated return value location. When the function completes, an instruction jumps back to the caller's location using the previously-stored return address. Then, an instruction copies the function's return value to the appropriate variable.

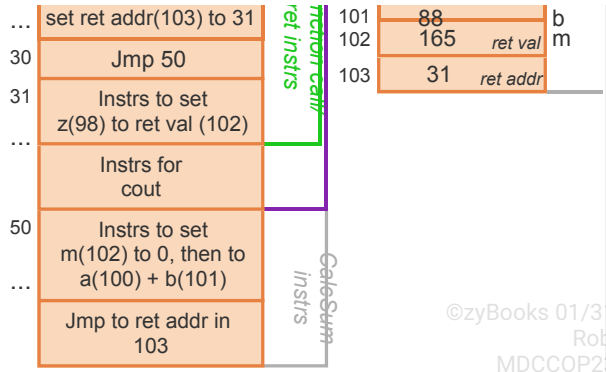
PARTICIPATION ACTIVITY 7.9.2: How function call/return works.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

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

int CalcSum(int a, int b) {
  int m;

  m = 0;
  m = a + b;
  return m;
}
```



```
int main() {
  int x;
  int y;
  int z;

  x = 0;
  y = 0;
  z = 0;
  cin >> x;
  cin >> y;
  z = CalcSum(x,y);

  cout << z << "\n";
  return 0;
}
```



©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation content:

Static Figure:
Being C++ code:
#include <iostream>
using namespace std;

int CalcSum(int a, int b) {
  int m;

  m = 0;
  m = a + b;
  return m;
}

int main() {
  int x;
  int y;
  int z;

  x = 0;
  y = 0;
  z = 0;
  cin >> x;
  cin >> y;
  z = CalcSum(x,y);

  cout << z << "\n";
  return 0;
}
End C++ code.
Two memory spaces are shown. The first memory space contains 8 memory cells/addresses. The first through sixth memory cells are designated for main assembly instructions and is highlighted in purple. The third through fifth cells are designated for Function call/ ret assembly instructions and is highlighted in green. The seventh and eight cells are designated for CalcSum assembly instructions and are highlighted in grey. The first cell contains “Instrs to init x(96), y(97), and z(98)”. The second cell contains “Instrs for cin”. The third cell contains, “Instrs to copy x(96) and y(97) to a(100) and b(101), set ret addr(103) to 31”. The fourth memory space contains “Jmp 50”. The fifth cell contains “Instrs to set z(98) to ret val (102)”. The sixth cell contains “Instrs for cout”. The seventh cell contains “Instrs to set m(102) to 0, then to a(100) + b(101)”. The eighth cell contains “Jmp to ret addr in 103”.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

The second memory space contains 9 memory cells/addresses. The second through fourth cells are designated for main data and is highlighted in purple. The sixth through ninth cells are designated for CalcSum data and is highlighted in grey. The first and fifth cells are empty. The second cell contains the value of x, 77. The third cell contains the value of y, 88. The fourth cell contains the value of z, 165. The sixth cell contains the value of a, 77. The seventh cell contains the value of b, 88. The eighth cell contains the value of m, 165, which is the return value. The nineth cell contins the return address value, 31.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Step 1: The compiler converts high-level code into assembly instructions (instrs) in memory.
The C++ code is processed through a compiler and is converted into assembly code in memory. The memory space contains 8 memory cells/addresses. The first through sixth memory cells are designated for main assembly instructions and is highlighted in purple. The third through fifth cells are designated for Function call/ ret assembly instructions and is highlighted in green. The seventh and eight cells are designated for CalcSum assembly instructions and is highlighted in grey. The first cell contains “Instrs to init x(96), y(97), and z(98)”. The second cell contains “Instrs for cin”. The third cell contains, “Instrs to copy x(96) and y(97) to a(100) and b(101), set ret addr(103) to 31”. The fourth memory space contains “Jmp 50”. The fifth cell contains “Instrs to set z(98) to ret val (102)”. The sixth cell contains “Instrs for cout”. The seventh cell contains “Instrs to set m(102) to 0, then to a(100) + b(101)”. The eighth cell contains “Jmp to ret addr in 103”.

Step 2: Before executing the function, arguments are copied to parameter local variables and a return address is stored.
The main function executes adding the values of the local variables to memory. The C++ main function code lines, “int x; “, “int y; “, and “int z;“ execute, designating 3 memory cells/addresses in memory for local variables x, y, and z. The code line, “x = 0; “, “y = 0; “, and “z = 0;“ execute, adding 0 to each local memory address x, y, and z. The code lines “cin >> x;” and “cin >> y;”, assigning value 77 to memory address for variable “x” and value 88 to memory address for variable “y”. Next, code line, “z = CalcSum(x,y);” designating 3 memory cells/addresses in memory for local variables “a”, “b”, and the return address for function CalcSum. The memory address for “a” contains the value 77, the memory address for “b” contains the value 88, and the memory address for the return address contains the value 31.

Step 3: The function executes and stores the result in a designated return value location.
The CalcSum function executes adding the values of the local variables to memory. The lines of code “int m;”, “m = 0;”, “m = a + b;”, and “return m;” executes, adding the value 165 to the memory address assigned to local variable m.

Step 4: When the function completes, an instruction jumps back to the caller's location using the previously-stored return address. Then, an instruction copies the function's return value to the appropriate variable.
The execution flow exits the CalcSum function and returns to the line “z = CalcSum(x,y);” in the main functions. The memory address for “z” now contains the value 165.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation captions:

1. The compiler converts high-level code into assembly instructions (instrs) in memory.
2. Before executing the function, arguments are copied to parameter local variables and a return address is stored.
3. The function executes and stores the result in a designated return value location.
4. When the function completes, an instruction jumps back to the caller's location using the previously-stored return address. Then, an instruction copies the function's return value to the appropriate variable.

PARTICIPATION ACTIVITY 7.9.3: How functions work.

1) After a function returns, its local variables keep their values, which serve as their initial values the next time the function is called.
   - True
   - False

2) A return address indicates the value returned by the function.
   - True
   - False

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

7.10 Functions: Common errors

A common error is to copy-and-paste code among functions but then not complete all necessary modifications to the pasted code. For example, a programmer might have developed and tested a function to convert a temperature value in Celsius to Fahrenheit, and then copied and modified the original function into a new function to convert Fahrenheit to Celsius as shown:

Figure 7.10.1: Copy-paste common error: Pasted code not properly modified. Find error on the right.

```
double Cel2Fah(double celVal) {
   double convTmp;
   double fahVal;

   convTmp = (9.0 / 5.0) * celVal;
   fahVal = convTmp + 32;

   return fahVal;
}
```

```
double Fah2Cel(double fahVal) {
   double convTmp;
   double celVal;

   convTmp = fahVal - 32;
   celVal = convTmp * (5.0 / 9.0);

   return fahVal;
}
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

The programmer forgot to change the return statement to return celVal rather than fahVal. Copying-and-pasting code is a common and useful time-saver, and can reduce errors by starting with known-correct code. Our advice is that when you copy-paste code, be extremely vigilant in making all necessary modifications. Just as the awareness that dark alleys or wet roads may be dangerous can cause you to vigilantly observe your surroundings or drive carefully, the awareness that copying-and-pasting is a common source of errors, may cause you to more vigilantly ensure you modify a pasted function correctly.

PARTICIPATION ACTIVITY 7.10.1: Copy-pasted sum-of-squares code.

Original parameters were num1, num2, num3. Original code was:

```
int sum;

sum = (num1 * num1) + (num2 * num2) + (num3 * num3);

return sum;
```

New parameters are num1, num2, num3, num4. Find the error in the copy-pasted new code below.

1)

```
int sum;

sum =
(num1 * num1) + (num2 * num2) + (num3 * num3)
+ (num3 * num4)
;

return sum;
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Another common error is to return the wrong variable, such as typing ***`return convTmp;`*** *instead of fahVal or celVal.* The function will work and sometimes even return the correct value.

Failing to return a value for a function is another common error. If execution reaches the end of a function's statements, the function automatically returns. For a function with a void return type, such an automatic return poses no problem, although some programmers recommend including a return statement for clarity. But for a function defined to return a value, the returned value is undefined; the value could be anything. For example, the user-defined function below lacks a return statement:

Figure 7.10.2: Missing return statement common error: Program may sometimes work, leading to hard-to-find bug.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

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

int StepsToFeet(int baseSteps) {
   const int FEET_PER_STEP = 3;  // Unit conversion
   int feetTot;                  // Corresponding feet to steps

   feetTot = baseSteps * FEET_PER_STEP;
}

int main() {
   int stepsInput;            // User defined steps
   int feetTot;               // Corresponding feet to steps

   // Prompt user for input
   cout << "Enter number of steps walked: ";
   cin >> stepsInput;

   // Call functions to convert steps to feet
   feetTot = StepsToFeet(stepsInput);
   cout << "Feet: " << feetTot << endl;

   return 0;
}
```

```
Enter number of steps walked: 1000
Feet: 3000
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Sometimes a function with a missing return statement (or just `return;`) still returns the correct value. The reason is that the compiler uses a memory location to return a value to the calling expression. That location may have also been used by the compiler to store a local variable of that function. If that local variable happens to be the item that was supposed to be returned, the value in that location is the correct return value. But a later seemingly unrelated change to a function, like defining a new variable, may cause the compiler to use different memory locations, and the function suddenly no longer returns the correct value, leading to a bewildered programmer.

PARTICIPATION ACTIVITY 7.10.2: Common function errors.

Find the error in the function's code.

```
1)  int
    ComputeSumOfSquares(int num1, int num2) {
      int sum;

      sum =
    (num1 * num1) + (num2 * num2)
    ;

      return;

    }
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

2)
```
int ComputeEquation1(int num, int val, int k
) {
   int sum;

   sum = (num * val) + (k * val)
;

   return num;

}
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

PARTICIPATION ACTIVITY 7.10.3: Common function errors.

1) Forgetting to return a value from a function is a common error.
   - True
   - False

2) Copying-and-pasting code can lead to common errors if all necessary changes are not made to the pasted code.
   - True
   - False

3) Returning the incorrect variable from a function is a common error.
   - True
   - False

4) Is this function correct for squaring an integer?

```
int sqr(int a) {
   int t;
   t = a * a;
}
```

   - Yes
   - No

5) Is this function correct for squaring an integer?

```
int sqr(int a) {
   int t;
   t = a * a;
   return a;
}
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

   - Yes
   - No

CHALLENGE ACTIVITY 7.10.1: Function errors: Copying one function to create another.

Using the CelsiusToKelvin function as a guide, create a new function, changing the name to KelvinToCelsius, and modifying the function accordingly.

Learn how our autograder works

539740.3879454.qx3zqy7

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

double CelsiusToKelvin(double valueCelsius) {
   double valueKelvin;

   valueKelvin = valueCelsius + 273.15;

   return valueKelvin;
}

/* Your solution goes here  */

int main() {
   double valueC;
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Run

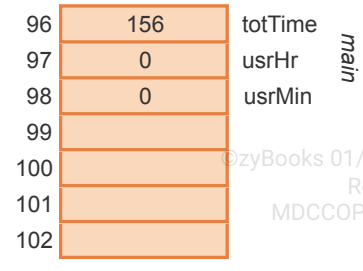
7.11 Pass by reference

Pass by reference

New programmers sometimes assign a value to a parameter, believing the assignment updates the corresponding argument variable. An example situation is when a function should return two values, whereas a function's *return* construct can only return one value. Assigning a normal parameter fails to update the argument's variable, because normal parameters are ***pass by value***, meaning the argument's value is copied into a local variable for the parameter.

PARTICIPATION ACTIVITY 7.11.1: Assigning a normal pass by value parameter has no impact on the corresponding argument.

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

void ConvHrMin(int timeVal, int hrVal, int minVal) {
   hrVal  = timeVal / 60;
   minVal = timeVal % 60;
}

int main() {
   int totTime;
   int usrHr;
   int usrMin;

   totTime = 0;
   usrHr = 0;
   usrMin = 0;

   cout << "Enter total minutes: ";
   cin >> totTime;

   ConvHrMin(totTime, usrHr, usrMin);
```



©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Enter total minutes: 156
Equals: 0 hrs 0 mins

```
   cout << "Equals: ";
   cout << usrHr << " hrs ";
   cout << usrMin << " mins" << endl;

   return 0;
}
```

Animation content:

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Static figure:
Begin C++ code:

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

void ConvHrMin(int timeVal, int hrVal, int minVal) {
   hrVal  = timeVal / 60;
   minVal = timeVal % 60;
}

int main() {
   int totTime;
   int usrHr;
   int usrMin;

   totTime = 0;
   usrHr = 0;
   usrMin = 0;

   cout << "Enter total minutes: ";
   cin >> totTime;

   ConvHrMin(totTime, usrHr, usrMin);

   cout << "Equals: ";
   cout << usrHr << " hrs ";
   cout << usrMin << " mins" << endl;

   return 0;
}
```

End C++ code.
Integer variables totTime, usrHr, and usrMin are displayed along with their memory locations 96, 97, and 98, respectively.

Step 1: The user is prompted to specify the total time in minutes. Function ConvHrMin is then called with arguments totTime, usrHr, and usrMin. All arguments are initialized with 0 and totTime is assigned with 156.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Step 2: ConvHrMin's parameters are passed by value, so the arguments' values are copied into local variables. Parameter timeVal is passed by value with argument totTime's value of 156 and is stored at memory location 100. Parameter hrVal is passed by value with argument usrHr's value of 0 and is stored at memory location 101. Parameter minVal is passed by value with argument usrMin's value of 0 and is stored at memory location 102. Statement hrVal = timeVal / 60; assigns hrVal with 2 at memory location 101 and statement minVal = timeVal % 60; assigns minVal with 36 at memory location 102, since timeVal's value is 156.

Step 3: Upon return, ConvHrMin's local variables are discarded. hrVal and minVal are local copies that do not impact usrHr and usrMin. hrVal and minVal's values 2 and 36, stored at memory locations 101 and 102, respectively, are discarded. Therefore, when statement cout << usrHr << " hrs "; outputs usrHr's value, "0 hrs" is displayed on a monitor, since 0 is stored at usrHr's memory location 97. When cout << usrMin << " mins" << endl; outputs userMin's value, "0 mins" is displayed on a monitor, since 0 is stored at userMin's memory location 98.

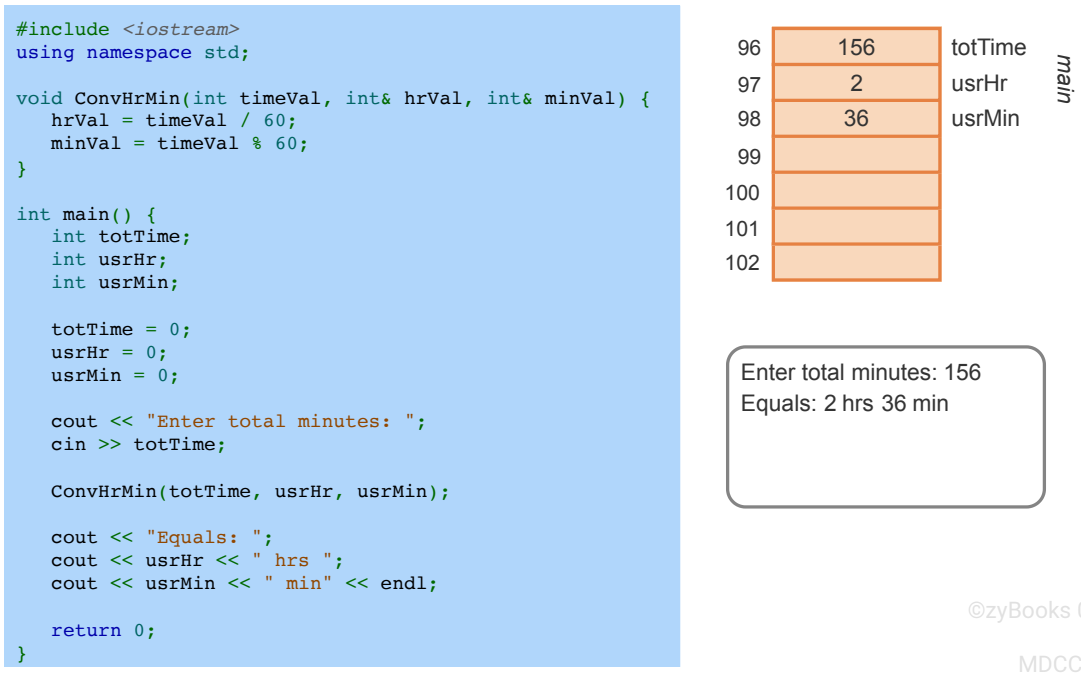
Animation captions:

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

1. The user is prompted to specify the total time in minutes. Function ConvHrMin is then called with arguments totTime, usrHr, and usrMin.
2. ConvHrMin's parameters are passed by value, so the arguments' values are copied into local variables.
3. Upon return, ConvHrMin's local variables are discarded. hrVal and minVal are local copies that do not impact usrHr and usrMin.

C++ supports another kind of parameter that enables updating of an argument variable. A ***pass by reference*** parameter does *not* create a local copy of the argument, but rather the parameter refers directly to the argument variable's memory location. Appending & to a parameter's data type makes the parameter pass by reference type.

PARTICIPATION ACTIVITY | 7.11.2: A pass by reference parameter allows a function to update an argument variable.



©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Animation content:

Static figure:
Begin C++ code:
#include <iostream>
using namespace std;

```
void ConvHrMin(int timeVal, int& hrVal, int& minVal) {
   hrVal  = timeVal / 60;
   minVal = timeVal % 60;
}

int main() {
   int totTime;
   int usrHr;
   int usrMin;

   totTime = 0;
   usrHr = 0;
   usrMin = 0;

   cout << "Enter total minutes: ";
   cin >> totTime;

   ConvHrMin(totTime, usrHr, usrMin);

   cout << "Equals: ";
   cout << usrHr << " hrs ";
   cout << usrMin << " mins" << endl;

   return 0;
}
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

End C++ code.
Integer variables totTime, usrHr, and usrMin are displayed along with their memory locations 96, 97, and 98, respectively.

Step 1: The user is prompted to specify the total time in minutes. Function ConvHrMin is then called with arguments totTime, usrHr, and usrMin. All arguments are initialized with 0 and totTime is assigned with 156.

Step 2: The & indicates that the hrVal and minVal parameters are passed by reference. Parameter timeVal is passed by value with argument totTime's value of 156 and is stored at memory location 100. Parameters hrVal and minVal are passed by reference with arguments usrHr and usrMin's memory locations and are stored at memory locations 101 and 102, respectively.

Step 3: Parameters passed by reference refer to that variable's memory location, so updates to hrVal and minVal update usrHr and usrMin. Statement hrVal = timeVal / 60; assigns hrVal with 2, therefore updates usrHr with 2 at memory location 97, since hrVal refers to usrHr's memory location. Statement minVal = timeVal % 60; assigns minVal with 36, therefore updates usrMin with 36 at memory location 98, since minVal refers to usrMin's memory location.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Step 4: Upon return from ConvHrMin, usrHr and usrMin retain the updated values. usrHr and usrMin's values 2 and 36, stored at memory locations 97 and 98, respectively, are retained. Therefore, when statement cout << usrHr << " hrs "; outputs usrHr's value, "2 hrs" is displayed on a monitor, since 2 is stored at usrHr's memory location 97. When cout << usrMin << " mins" << endl; outputs userMin's value, "36 mins" is displayed on a monitor, since 36 is stored at userMin's memory location 98.

Animation captions:

1. The user is prompted to specify the total time in minutes. Function ConvHrMin is then called with arguments totTime, usrHr, and usrMin.
2. The & indicates that the hrVal and minVal parameters are passed by reference.
3. Parameters passed by reference refer to that variable's memory location, so updates to hrVal and minVal update usrHr and usrMin.
4. Upon return from ConvHrMin, usrHr and usrMin retain the updated values.

Pass by reference parameters should be used sparingly. For the case of two return values, commonly a programmer should instead create two functions. For example, defining two separate functions `int StepsToFeet(int baseSteps)` and `int StepsToCalories(int baseSteps)` is better than a single function `void StepsToFeetAndCalories(int baseSteps, int& baseFeet, int& totCalories)` . The separate functions support modular development, and enables use of the functions in an expression as in `if (StepsToFeet(mySteps) < 100)`.

Using multiple pass by reference parameters makes sense when the output values are intertwined, such as computing monetary change, whose function might be `void ComputeChange(int totCents, int& numQuarters, int& numDimes, int& numNickels, int& num` or converting from polar to Cartesian coordinates, whose function might be `void PolarToCartesian(int radialPol, int anglePol, int& xCar, int& yCar)`.

zyDE 7.11.1: Calculating monetary change.

Complete the monetary change program. Use the fewest coins (i.e., using maximum larger coins first).

Load default template...

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

// FIXME: Add parameters for dimes, nickels, and penn
void ComputeChange(int totCents, int& numQuarters ) {

   cout << "FIXME: Finish writing ComputeChange" << e

   numQuarters = totCents / 25;
}

int main() {
   int userCents;
   int numQuarters;
```

83

Run

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

PARTICIPATION ACTIVITY 7.11.3: Function definition returns and arguments.

Choose the most appropriate function definition.

1) Convert inches into centimeters.

- `void InchToCM(double inches, double centimeters) ...`
- `double InchToCM(double inches) ...`
- More than one function should be written.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

2) Get a user's full name by prompting "Enter full name" and then automatically separating into first and last names.

- `void GetUserFullName(string& firstName, string& lastName) ...`
- `string GetUserFullName() ...`
- `string, string GetUserFullName() ...`
- More than one function should be written.

3) Compute the area and diameter of a circle given the radius.

- `void GetCircleAreaDiam(double radius, double& area, double& diameter) ...`
- `double GetCircleAreaDiam (double radius, double& area) ...`
- `double, double GetCircleAreaDiam(double radius) ...`
- More than one function should be written.

PARTICIPATION ACTIVITY 7.11.4: Function definitions with pass by value and pass by reference.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Complete the function definition, creating pass by value or pass by reference parameters as appropriate.

1) Convert gallons to liters. Parameter is userGallons, type is double.

```
double GallonsToLiters(
) {
```

Check Show answer

2) Convert userMeters into userFeet and userInches (three parameters, in that order), types are doubles.

```
void MetersToFeetInches(
) {
```

Check Show answer

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Avoid assigning pass by value parameters

Although a pass by value parameter creates a local copy, good practice is to avoid assigning such a parameter. The following code is correct but bad practice.

Figure 7.11.1: Programs should not assign pass by value parameters.

```
int IntMax(int numVal1, int numVal2) {
   if (numVal1 > numVal2) {
      numVal2 = numVal1; // numVal2 holds
max
   }

   return numVal2;
}
```

Assigning a parameter can reduce code slightly, but is widely considered a lazy programming style. Assigning a parameter can mislead a reader into believing the argument variable is supposed to be updated. Assigning a parameter also increases likelihood of a bug caused by a statement reading the parameter later in the code but assuming the parameter's value is the original passed value.

PARTICIPATION ACTIVITY 7.11.5: Assigning a pass by value parameter.

1) Assigning a pass by value parameter in a function is discouraged due to potentially confusing a program reader into believing the argument is being updated.

- ○ True
- ○ False

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

2) Assigning a pass by value parameter in a function is discouraged due to potentially leading to a bug where a later line of code reads the parameter assuming the parameter still contains the original value.

- True
- False

3) Assigning a pass by value parameter can avoid having to declare an additional local variable.

- True
- False

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Reference variables

A programmer can also declare a reference variable. A ***reference*** is a variable type that refers to another variable. Ex: `int& maxValRef` declares a reference to a variable of type int. The programmer must initialize each reference with an existing variable, which can be done by initializing the reference variable when the reference is declared. Ex: `int& maxValRef = usrInput3;`

In the example below, usrValRef is a reference that refers to usrValInt. The user-entered number is assigned to the variable usrValInt. Because usrValRef refers to usrValInt, printing usrValInt or usrValRef will print the number.

Figure 7.11.2: Reference variable example.

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

int main() {
   int usrValInt;
   int& usrValRef = usrValInt;  // Refers to usrValInt

   cout << "Enter an integer: ";
   cin  >> usrValInt;

   cout << "We wrote your integer to usrValInt." << endl;
   cout << "usrValInt is: " << usrValInt << "." << endl;
   cout << "usrValRef refers to usrValInt, and is: " << usrValRef << "." <<
endl;

   usrValInt = 99;
   cout << endl << "We assigned usrValInt with 99." << endl;
   cout << "usrValInt is now: " << usrValInt << "." << endl;
   cout << "usrValRef is now: " << usrValRef << "." << endl;
   cout << "Note that usrValRef refers to usrValInt, so it changed too." <<
endl;
   return 0;
}
```

```
Enter an integer: 42
We wrote your integer to usrValInt.
usrValInt is: 42.
usrValRef refers to usrValInt, and is: 42.

We assigned usrValInt with 99.
usrValInt is now: 99.
usrValRef is now: 99.
Note that usrValRef refers to usrValInt, so it changed too.
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

PARTICIPATION ACTIVITY 7.11.6: Reference variables.

1) What does the following output?

```
int numAStudents = 12;
int numBStudents = 5;
int& studentsRef =
numAStudents;

cout << studentsRef;
```

Check Show answer

2) What does the following output?

```
int examGrade = 95;
int& gradeRef = examGrade;

examGrade = examGrade + 1;
cout << gradeRef;
```

Check Show answer

3) What does the following output?

```
double treeHeightFt = 7.1;
double& heightRef =
treeHeightFt;

heightRef = 12.2;
cout << treeHeightFt;
```

Check Show answer

4) Declare a reference named myScore and initialize the reference to the previously declared variable int teamScore.

Check Show answer

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Exploring further:

- Passing arguments by value and by reference from msdn.microsoft.com

CHALLENGE ACTIVITY 7.11.1: Function pass by reference: Transforming coordinates.

Define a function CoordTransform() that transforms the function's first two input parameters xVal and yVal into two output parameters xValNew and yValNew. The function returns void. The transformation is new = (old + 1) * 2. Ex: If xVal = 3 and yVal = 4, then xValNew is 8 and yValNew is 10.

Learn how our autograder works

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

539740.3879454.qx3zqy7

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

/* Your solution goes here  */

int main() {
   int xValNew;
   int yValNew;
   int xValUser;
   int yValUser;

   cin >> xValUser;
   cin >> yValUser;

   CoordTransform(xValUser, yValUser, xValNew, yValNew);
```

Run

CHALLENGE ACTIVITY 7.11.2: Pass by reference.

539740.3879454.qx3zqy7

Start

Declare char reference variable letterRef and initialize letterRef to letterValue.

Ex: If input is X, then output is:

```
Referenced letter is X.
```

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

int main() {
   char letterValue;

   /* Your code goes here */

   cin >> letterValue;

   cout << "Referenced letter is " << letterRef << "." << endl;

   return 0;
}
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

1 2 3

Check Next level

7.12 Using pass by reference to modify string/vector parameters

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Functions commonly modify a string or vector. The following function modifies a string by replacing spaces with hyphens.

Figure 7.12.1: Modifying a string parameter, which should be pass by reference.

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

// Function replaces spaces with hyphens
void StrSpaceToHyphen(string& modStr) {
   unsigned int i;   // Loop index

   for (i = 0; i < modStr.size(); ++i) {
      if (modStr.at(i) == ' ') {
         modStr.at(i) = '-';
      }
   }
}

int main() {
   string userStr;  // Input string from user

   // Prompt user for input
   cout << "Enter string with spaces: " << endl;
   getline(cin, userStr);

   // Call function to modify user defined string
   StrSpaceToHyphen(userStr);

   // Output modified string
   cout << "String with hyphens: ";
   cout << userStr << endl;

   return 0;
}
```

```
Enter string with spaces:
Hello there everyone.
String with hyphens: Hello-there-everyone.

...

Enter string with spaces:
Good bye  now   !!!
String with hyphens: Good-bye--now---!!!
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

The string serves as function input and output. The string parameter must be pass by reference, achieved using &, so that the function modifies the original string argument (userStr) and not a copy.

zyDE 7.12.1: Modifying a string parameter: Spaces to hyphens.

1. Run the program, noting correct output.

2. Remove the & and run again, noting the string is not modified, because the string is pass by value and thus the function modifies a copy. When done replace the &
3. Modify the function to also replace each '!' by a '?'.

Load default template...

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

// Function replaces spa
void StrSpaceToHyphen(st
   unsigned int i;  // I

   for (i = 0; i < modSt
      if (modStr.at(i) =
         modStr.at(i) =
      }
   }
}
```

Hello there everyone!!!

Run

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Sometimes a programmer defines a vector or string parameter as pass by reference even though the function does not modify the parameter, to prevent the performance and memory overhead of copying the argument that would otherwise occur.

The keyword ***const*** can be prepended to a function's vector or string parameter to prevent the function from modifying the parameter. Programmers commonly make a large vector or string input parameter pass by reference, to gain efficiency, while also making the parameter const, to prevent assignment.

The following illustrates. The first function modifies the vector so it defines a normal pass by reference. The second function does *not* modify the vector but for efficiency uses constant pass by reference.

Figure 7.12.2: Normal and constant pass by reference vector parameters in a vector reversal program.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

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

void ReverseVals(vector<int>& vctrVals) {
   unsigned int i;  // Loop index
   int tmpVal;      // Temp variable for swapping

   for (i = 0; i < (vctrVals.size() / 2); ++i) {
      tmpVal = vctrVals.at(i); // These statements
swap
      vctrVals.at(i) = vctrVals.at(vctrVals.size()
- 1 - i);
      vctrVals.at(vctrVals.size() - 1 - i) =
tmpVal;
   }
}

void PrintVals(const vector<int>& vctrVals) {
   unsigned int i;  // Loop index

   // Print updated vector
   cout << endl << "New values: ";
   for (i = 0; i < vctrVals.size(); ++i) {
      cout << " " << vctrVals.at(i);
   }
   cout << endl;
}

int main() {
   const int NUM_VALUES = 8;            // Vector
size
   vector<int> userValues(NUM_VALUES);  // User
values
   int i;                               // Loop
index

   // Prompt user to populate vector
   cout << "Enter " << NUM_VALUES << " values..."
<< endl;
   for (i = 0; i < NUM_VALUES; ++i) {
      cout << "Value: ";
      cin >> userValues.at(i);
   }

   // Call function to reverse vector values
   ReverseVals(userValues);

   // Print reversed values
   PrintVals(userValues);

   return 0;
}
```

```
Enter 8 values...
Value: 10
Value: 20
Value: 30
Value: 40
Value: 50
Value: 60
Value: 70
Value: 80

New values:  80 70 60 50 40
30 20 10
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

A reader might wonder why all input parameters are not defined as constant pass by reference parameters: Why make local copies at all? The reason is efficiency. For parameters involving just a few memory locations, making a local copy enables the compiler to generate more efficient code, in part because the compiler can place those copies inside a tiny-but-fast memory inside the processor called a register file—further details are beyond our scope.

In summary:

- Define a function's output or input/output parameters as pass by reference.
  - But create output parameters sparingly, striving to use return values instead.
- Define input parameters as pass by value.

- Except for large items (perhaps 10 or more elements); use constant pass by reference for those.

PARTICIPATION ACTIVITY 7.12.1: Constants and pass by reference.

How should a function's vector parameter `ages` be defined for the following situations?

1) ages will always be small (fewer than 10 elements) and the function will not modify the vector.
   - Constant and pass by reference.
   - Constant but not pass by reference.
   - Pass by reference but not constant.
   - Neither constant nor pass by reference.

2) ages will always be small, and the function will modify the vector.
   - Constant and pass by reference.
   - Constant but not pass by reference.
   - Pass by reference but not constant.
   - Neither constant nor pass by reference.

3) ages may be very large, and the function will modify the vector.
   - Constant and pass by reference.
   - Constant but not pass by reference.
   - Pass by reference but not constant.
   - Neither constant nor pass by reference.

4) ages may be very large, and the function will not modify the vector.
   - Constant and pass by reference.
   - Constant but not pass by reference.
   - Pass by reference but not constant.
   - Neither constant nor pass by reference.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

PARTICIPATION ACTIVITY 7.12.2: Vector parameters.

Define a function's vector parameter `ages` for the following situations. Assume ages is a vector of integers. Example: ages will always be small (fewer than 10 elements) and the function will not modify the vector: `const vector<int> ages`.

1) ages will always be small, and the function will modify the vector.

```
void MyFct (
                                ) {
```

Check Show answer

2) ages may be very large, and the function will modify the vector

```
void MyFct (
                                ) {
```

Check Show answer

3) ages may be very large, and the function will not modify the vector.

```
void MyFct (
                                ) {
```

Check Show answer

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

CHALLENGE ACTIVITY 7.12.1: Use an existing function.

Use function GetUserInfo to get a user's information. If user enters 20 and Holly, sample program output is:

```
Holly is 20 years old.
```

Learn how our autograder works

539740.3879454.qx3zqy7

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  void GetUserInfo(int& userAge, string& userName) {
6     cout << "Enter your age: " << endl;
7     cin >> userAge;
8     cout << "Enter your name: " << endl;
9     cin >> userName;
10 }
11
12 int main() {
13    int userAge;
14    string userName;
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

```
14    string userName;
15
```

Run

CHALLENGE ACTIVITY 7.12.2: Modify a string parameter.

Complete the function to replace any period by an exclamation point. Ex: "Hello. I'm Miley. Nice to meet you." becomes:

```
"Hello! I'm Miley! Nice to meet you!"
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Learn how our autograder works

539740.3879454.qx3zqy7

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

void MakeSentenceExcited(string& sentenceText) {

   /* Your solution goes here  */

}

int main() {
   string testStr;

   getline(cin, testStr);
   MakeSentenceExcited(testStr);
```

Run

CHALLENGE ACTIVITY 7.12.3: Modify a vector parameter.

Write a function SwapVectorEnds() that swaps the first and last elements of its vector parameter. Ex: sortVector = {10, 20, 30, 40} becomes {40, 20, 30, 10}.

Learn how our autograder works

539740.3879454.qx3zqy7

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

/* Your solution goes here  */

int main() {
   vector<int> sortVector(4);
   unsigned int i;
   int userNum;

   for (i = 0; i < sortVector.size(); ++i) {
      cin >> userNum;
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

```
14      sortVector.at(i) = userNum;
15    }
```

Run

CHALLENGE ACTIVITY | 7.12.4: Functions with string parameters.

539740.3879454.qx3zqy7

Start

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Complete the function ContainsSome() that has one string parameter and one character parameter. The function returns tru least one character in the string is not equal to the character parameter. Otherwise, the function returns false.

Ex: If the input is `txzd x`, then the output is:

```
True, at least one character is not equal to x.
```

```
1  #include <iostream>
2  using namespace std;
3
4  bool ContainsSome(string inputString, char x) {
5
6     /* Your code goes here */
7
8  }
9
10 int main() {
11    string inputString;
12    char x;
13    bool result;
14
15    cin >> inputString;
```

1 2 3

Check Next level

CHALLENGE ACTIVITY | 7.12.5: Functions with vector parameters.

539740.3879454.qx3zqy7

Start

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Define a function TestSelectiveSum() that takes one integer vector parameter and one integer parameter. The function comp the sum of the vector's elements that are greater than the integer parameter. Then, the function returns true if the sum is pos and returns false otherwise.

Ex: If the input is:

```
4
7 -8 -2 4
-2
```

then the vector has 4 elements {7, -8, -2, 4}, and the integer parameter is -2. The output is:

```
True, the sum of the elements that are greater than -2 is positive.
```

Note: The sum is zero if no element is greater than the integer parameter.

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

/* Your code goes here */

int main() {
   vector<int> inputVector;
   int size;
   int input;
   int i;
   int x;
   bool result;

   // Read the vector's size, and then the vector's elements
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

1 2

Check Next level

7.13 Functions with C string parameters

Functions commonly modify C strings. The following function modifies a string by replacing spaces with hyphens.

Figure 7.13.1: Modifying a C string parameter.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

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

// Function replaces spaces with hyphens
void StrSpaceToHyphen(char modString[]) {
   int i;      // Loop index

   for (i = 0; i < strlen(modString); ++i) {
      if (modString[i] == ' ') {
         modString[i] = '-';
      }
   }
}

int main() {
   const int INPUT_STR_SIZE = 50;  // Input C
string size
   char userStr[INPUT_STR_SIZE];   // Input C
string from user

   // Prompt user for input
   cout << "Enter string with spaces: " << endl;
   cin.getline(userStr, INPUT_STR_SIZE);

   // Call function to modify user defined C
string
   StrSpaceToHyphen(userStr);

   cout << "String with hyphens: " << userStr <<
endl;

   return 0;
}
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

```
Enter string with spaces:
Hello there everyone.
String with hyphens: Hello-
there-everyone.

...

Enter string with spaces:
Good bye  now   !!!
String with hyphens: Good-bye-
-now---!!!
```

The parameter definition uses [] to indicate an array parameter. The function call's argument does not use []. The compiler *automatically passes the C string as a pointer*. Hence, the above function modifies the original string argument (userStr) and not a copy.

The strlen() function can be used to determine the length of the string argument passed to the function. So, unlike functions with array parameters of other types, a function with a C string parameter does not require a second parameter to specify the string size.

zyDE 7.13.1: Modifying a C string parameter: Spaces to hyphens.

1. Run the program, noting correct output.
2. Modify the function to also replace each '!' by a '?'.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Load default template...

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

// Function replaces spa
void StrSpaceToHyphen(ch
   int i;      // Loop
   for (i = 0; i < strle
      if (modString[i] =
         modString[i] =
      }
   }
}
```

Hello there everyone!!!

Run

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

PARTICIPATION ACTIVITY 7.13.1: Functions with string parameters.

1) A parameter declared as `char movieTitle[]` is a string.
   - True
   - False

2) For a function with a string parameter, the function must include a second parameter for the string size.
   - True
   - False

3) To pass a string to a function, the argument must include [], as in `GetMovieRating(favMovie[])`.
   - True
   - False

A programmer can explicitly define an array parameter as a pointer. The following uses `char* modString` instead of the earlier `char modString[]`. Such pointer parameters are common for C string parameters, such as in the C string library functions.

Figure 7.13.2: Modifying a C string using a pointer parameter.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

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

// Function replaces spaces with hyphens
void StrSpaceToHyphen(char* modString) {
   int i;      // Loop index

   for (i = 0; i < strlen(modString); ++i) {
      if (modString[i] == ' ') {
         modString[i] = '-';
      }
   }
}

int main() {
   const int INPUT_STR_SIZE = 50;  // Input string size
   char userStr[INPUT_STR_SIZE];   // Input C string from user

   // Prompt user for input
   cout << "Enter string with spaces: " << endl;
   cin.getline(userStr, INPUT_STR_SIZE);

   // Call function to modify user defined C string
   StrSpaceToHyphen(userStr);

   cout << "String with hyphens: " << userStr << endl;

   return 0;
}
```

```
Enter string with spaces:
Hello there everyone!
String with hyphens: Hello-there-everyone!

...

Enter string with spaces:
Good bye  now   !!!
String with hyphens: Good-bye--now---!!!
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

PARTICIPATION ACTIVITY 7.13.2: Functions with C string parameters.

1) Passing a C string to a function creates a copy of that string within the function.
   - True
   - False

2) A C string is automatically passed by pointer.
   - True
   - False

CHALLENGE ACTIVITY 7.13.1: Modify a C string parameter.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Complete the function to replace any period by an exclamation point. Ex: "Hello. I'm Miley. Nice to meet you." becomes:

```
"Hello! I'm Miley! Nice to meet you!"
```

Learn how our autograder works

539740.3879454.qx3zqy7

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

void MakeSentenceExcited(char* sentenceText) {

   /* Your solution goes here  */

}

int main() {
   const int TEST_STR_SIZE = 50;
   char testStr[TEST_STR_SIZE];

   cin.getline(testStr, TEST_STR_SIZE);
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Run

7.14 Default parameter values

Sometimes a function's last parameter (or last few) should be optional. A function call could then omit the last argument, and instead the program would use a default value for that parameter. A function can have a ***default parameter value*** for the last parameter(s), meaning a call can optionally omit a corresponding argument.

Figure 7.14.1: Parameter with a default value.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

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

// Function prints date in two styles (0: American (default), 1:
European)
void PrintDate(int currDay, int currMonth, int currYear, int
printStyle = 0) {

   if (printStyle == 0) {       // American
      cout << currMonth << "/" << currDay << "/" << currYear;
   }
   else if (printStyle == 1) { // European
      cout << currDay << "/" << currMonth << "/" << currYear;
   }
   else {
      cout << "(invalid style)";
   }
}

int main() {

   // Print dates given various style settings
   PrintDate(30, 7, 2012, 0);
   cout << endl;

   PrintDate(30, 7, 2012, 1);
   cout << endl;

   PrintDate(30, 7, 2012); // Uses default value for printStyle
   cout << endl;

   return 0;
}
```

```
7/30/2012
30/7/2012
7/30/2012
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

The fourth (and last) parameter has a default value: `int printStyle = 0`. If a function call does not provide a fourth argument, then the style parameter is 0.

The same can be done for other parameters, as in:
`void PrintDate(int currDay = 1, int currMonth = 1, int currYear = 2000, int printStyle = 0)`
Because arguments are matched with parameters based on their ordering in the function call, only the last arguments can be omitted. The following are valid calls to this PrintDate() function having default values for all parameters:

Figure 7.14.2: Valid function calls with default parameter values.

```
PrintDate(30, 7, 2012, 0); // No defaults
PrintDate(30, 7, 2012);    // Defaults:                          style=0
PrintDate(30, 7);          // Defaults:                 year=2000, style=0
PrintDate(30);             // Defaults:        month=1, year=2000, style=0
(strange, but valid)
PrintDate();               // Defaults: day=1, month=1, year=2000, style=0
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

If a parameter does not have a default value, then failing to provide an argument generates a compiler error. Ex: Given: void PrintDate(int currDay, int currMonth, int currYear, int printStyle = 0). Then the call PrintDate(30, 7) generates the following error message from g++.

Figure 7.14.3: Compiler error if parameters corresponding to omitted arguments don't have default values.

```
fct_defparm.cpp: In function int main():
fct_defparm.cpp:5: error: too few arguments to function void PrintDate(int, int, int,
int)
fct_defparm.cpp:22: error: at this point in file
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

PARTICIPATION ACTIVITY 7.14.1: Function parameter defaults.

Given:

```
void CalcStat(int num1, int num2, int num3 = 0, char usrMethod = 'a') { ...
}
```

1) A compiler error will occur because only an int parameter can have a default value.
   - True
   - False

2) The call CalcStat(44, 47, 42, 'b') uses usrMethod = 'a' because the parameter default value of 'a' overrides the argument 'b'.
   - True
   - False

3) The call CalcStat(44, 47, 42) uses usrMethod = 'a'.
   - True
   - False

4) The call CalcStat(44, 47, 'b') uses num3 = 0.
   - True
   - False

5) The following is a valid start of a function definition: `void myFct(int num1 = 0, int num2 = 0, char usrMethod) {`
   - True
   - False

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Exploring further:

- Default arguments from msdn.microsoft.com

CHALLENGE ACTIVITY 7.14.1: Functions with default parameters.

539740.3879454.qx3zqy7

Start

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Type the program's output

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

void PrintNums(int a, int b, int c = 17) {
   cout << a << ", " << b << ", " << c << endl;
}

int main() {
   PrintNums(2, 4, 7);
   PrintNums(5, 6);

   return 0;
}
```

```
2, 4, 7
5, 6, 17
```

1 2

Check Next

CHALLENGE ACTIVITY 7.14.2: Return number of pennies in total.

Write a function NumberOfPennies() that returns the total number of pennies given a number of dollars and (optionally) a number of pennies. Ex: 5 dollars and 6 pennies returns 506.

Learn how our autograder works

539740.3879454.qx3zqy7

```
1  #include <iostream>
2  using namespace std;
3
4  /* Your solution goes here  */
5
6  int main() {
7     cout << NumberOfPennies(5, 6) << endl; // Should print 506
8     cout << NumberOfPennies(4) << endl;    // Should print 400
9     return 0;
10 }
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Run

7.15 Function name overloading

Sometimes a program has two functions with the same name but differing in the number or types of parameters, known as ***function name overloading*** or just ***function overloading***. The following two functions print a date given the day, month, and year. The first function has parameters of type int, int, and int, while the second has parameters of type int, string, and int.

Figure 7.15.1: Overloaded function name.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

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

void PrintDate(int currDay, int currMonth, int currYear)
{
   cout << currMonth << "/" << currDay << "/" <<
currYear;
}

void PrintDate(int currDay, string currMonth, int
currYear) {
   cout << currMonth << " " << currDay << ", " <<
currYear;
}

int main() {

   PrintDate(30, 7, 2012);
   cout << endl;

   PrintDate(30, "July", 2012);
   cout << endl;

   return 0;
}
```

```
7/30/2012
July 30,
2012
```

The compiler determines which function to call based on the argument types. PrintDate(30, 7, 2012) has argument types int, int, int, so calls the first function. PrintDate(30, "July", 2012) has argument types int, string, int, so calls the second function.

More than two same-named functions is allowed as long as each has distinct parameter types. Thus, in the above program:

- PrintDate(int month, int day, int year, int style) can be added because the types int, int, int, int differ from int, int, int, and from int, string, int.
- PrintDate(int month, int day, int year) yields a compiler error, because two functions have types int, int, int (the parameter names are irrelevant).

A function's return type does not influence overloading. Thus, having two same-named function definitions with the same parameter types but different return types still yield a compiler error.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

The use of overloading and of default parameter values may be combined as long as no ambiguity is introduced. Adding the function `void PrintDate(int month, int day, int year, int style = 0)` above would generate a compiler error because the compiler cannot determine if the function call `PrintDate(7, 30, 2012)` should go to the "int, int, int" function or to that new "int, int, int, int" function with a default value for the last parameter.

PARTICIPATION ACTIVITY 7.15.1: Function name overloading.

Given the following function definitions, select the number that each function call would print. If the function call would not compile, choose Error.

```
void PrintDate(int day, int month, int year) {
   cout << "1" << endl;
}

void PrintDate(int day, string month, int year) {
   cout << "2" << endl;
}

void PrintDate(int month, int day) {
   cout << "3" << endl;
}
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

1) PrintDate(30, 7, 2012);

- 1
- 2
- 3
- Error

2) PrintDate(30, "July", 2012);

- 1
- 2
- 3
- Error

3) PrintDate(7, 2012);

- 1
- 2
- 3
- Error

4) PrintDate(30, 7);

- 1
- 2
- 3
- Error

5) PrintDate("July", 2012);

- 1
- 2
- 3
- Error

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Exploring further:

- Overloaded functions from cplusplus.com.

CHALLENGE ACTIVITY 7.15.1: Overload salutation printing.

Complete the second PrintSalutation function to print the following given personName "Holly" and customSalutation "Welcome":

```
Welcome, Holly
```

End with a newline.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Learn how our autograder works

539740.3879454.qx3zqy7

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

void PrintSalutation(string personName) {
   cout << "Hello, " << personName << endl;
}

// Define void  PrintSalutation(string personName, string customSalutation)

/* Your solution goes here  */

int main() {
   PrintSalutation("Holly", "Welcome");
   PrintSalutation("Sanjiv");
```

Run

CHALLENGE ACTIVITY 7.15.2: Convert a height into inches.

Write a second ConvertToInches() with two double parameters, numFeet and numInches, that returns the total number of inches. Ex: ConvertToInches(4.0, 6.0) returns 54.0 (from 4.0 * 12 + 6.0).

Learn how our autograder works

539740.3879454.qx3zqy7

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

double ConvertToInches(double numFeet) {
   return numFeet * 12.0;
}

/* Your solution goes here  */

int main() {
   double totInches;

   totInches = ConvertToInches(4.0, 6.0);
   cout << "4.0, 6.0 yields " << totInches << endl;
```

Run

CHALLENGE ACTIVITY 7.15.3: Function name overloading.

539740.3879454.qx3zqy7

Start

The function PrintPumpkinWeight() has a double parameter. Define a second PrintPumpkinWeight() function that has an inte parameter. The second function outputs the following in order, all on one line:

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

- "Pumpkin weight as a whole number: "
- the value of the integer parameter
- " kilograms"

End with a newline.

Ex: If the input is `2.5 2`, then the output is:

```
Pumpkin weight to one decimal place: 2.5 kilograms
Pumpkin weight as a whole number: 2 kilograms
```

```
1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4
5  void PrintPumpkinWeight(double pumpkinWeight) {
6     cout << fixed << setprecision(1) << "Pumpkin weight to one decimal place: ";
7     cout << pumpkinWeight << " kilograms" << endl;
8  }
9
10 /* Your code goes here */
11
12 int main() {
13    double pumpkinWeight1;
14    int pumpkinWeight2;
15
```

1 2 3

Check Next level

7.16 Parameter error checking

zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Verifying parameter values

Commonly, a function expects parameter values to be within some range. *A good practice is to check that a parameter's value is within an expected range.* If not in the range, the function might take one or more of various actions, like outputting an error message, assigning a valid value, returning a value indicating failure, exiting the program, etc.

Figure 7.16.1: Function with parameter error checking.

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

void PrintDate(int currDay, int currMonth, int currYear) {

   // Parameter error checking
   if ((currDay < 1) || (currDay > 31)) {
      cout << "Invalid day (" << currDay << "). Using 1." << endl;
      currDay = 1;
   }

   if ((currMonth < 1) || (currMonth > 12)) {
      cout << "Invalid month (" << currMonth << "). Using 1." << endl;
      currMonth = 1;
   }

   // Begin function's normal behavior
   cout << currMonth << "/" << currDay << "/" << currYear;
}


int main() {

   PrintDate(30, 7, 2012);
   cout << endl << endl;

   PrintDate(40, 7, 2012);
   cout << endl << endl;

   PrintDate(30, 13, 2012);
   cout << endl << endl;

   return 0;
}
```

```
7/30/2012

Invalid day (40). Using 1.
7/1/2012

Invalid month (13). Using 1.
1/30/2012
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

PARTICIPATION ACTIVITY 7.16.1: Checking parameter values.

Consider the example above.

1) The month must be in the range from 1 to ____ .

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Check Show answer

2) If the month parameter is not in the valid range, the code outputs an error message, and assigns currMonth with _____ .

Check Show answer

3) In addition to currMonth, which other parameter is checked for being in a valid range?

Check Show answer

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

7.17 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 7.17.1: Include directives.

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

The following animation illustrates.

PARTICIPATION ACTIVITY 7.17.1: Preprocessor's handling of an include directive.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

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

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

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

```
    return 0;
}
```

Animation content:

Static figure:
Begin code:
#include "myfile"

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

// 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.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

- 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 7.17.2: 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>;`
   - #
   - <>
   - ;

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Exploring further:

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

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

7.18 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 7.18.1: 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
>
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

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

Figure 7.18.2: Compiling multiple files together.

Without `#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
```

With `#include "threeintsfcts.h"` in main.cpp

```
> 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.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Construct 7.18.1: 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 7.18.3: 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 7.18.1: Header files.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

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

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

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

7.19 C++ example: Salary calculation with functions

zyDE 7.19.1: Calculate salary: Using functions.

Separating calculations into functions simplifies modifying and expanding programs.

The following program calculates the tax rate and tax to pay, using functions. One function returns a tax rate based on an annual salary.

1. Run the program below with annual salaries of 40000, 60000, and 0.
2. Change the program to use a function to input the annual salary.
3. Run the program again with the same annual salaries as above. Are results the same?

Note: The calculation is inaccurate to how taxes are formally assessed and is a simplification for educational purposes only.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Load default template...

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

double GetCorrespondingTableValue(int search, vector<
   int baseTableLength;
   double value;
   int i;
   bool keepLooking;

   baseTableLength = baseTable.size();
   i = 0;
   keepLooking = true;

   while ((i < baseTableLength) && keepLooking) {
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

40000 60000 0

Run

A solution to the above problem follows. The program was altered slightly to allow a zero annual salary and to end when a user enters a negative number for an annual salary.

zyDE 7.19.2: Calculate salary: Using function (solution).

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Load default template...

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

// Function to prompt for and input an integer
int PromptForInteger(const string userPrompt) {
   int inputValue;

   cout << userPrompt << ": " << endl;
   cin >> inputValue;

   return inputValue;
}
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

```
60000 40000 1000000
-1
```

Run

7.20 C++ example: Domain name validation with functions

zyDE 7.20.1: Validate domain names with functions.

Functions facilitate breaking down a large problem into a collection of smaller ones.

A ***top-level domain*** (TLD) name is the last part of an Internet domain name like .com in example.com. A ***core generic top-level domain*** (core gTLD) is a TLD that is either .com, .net, .org, or .info. A ***restricted top-level domain*** is a TLD that is either .biz, .name, or .pro. A ***second-level domain*** is a single name that precedes a TLD as in apple in apple.com

The following program repeatedly prompts for a domain name and indicates whether that domain name is valid and has a core gTLD. For this program, a valid domain name has a second-level domain followed by a TLD, and the second-level domain has these three characteristics:

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

1. Is 1-63 characters in length.
2. Contains only uppercase and lowercase letters or a dash.
3. Does not begin or end with a dash.

For this program, a valid domain name must contain only one period, such as apple.com, but not support.apple.com. The program ends when the user presses just the Enter key in response to a prompt.

1. Run the program. Note that a restricted gTLD is not recognized as such.
2. Change the program by writing an input function and adding the validation for a restricted gTLD. Run the program again.

Load default template...

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

const int MAX_NUMS = 4;  // Global variable used for

// ************************************************
// GetPeriodPosition - Pass a string and return the p
// ************************************************
int GetPeriodPosition(string stringToSearch) {
   int periodCounter;
   int periodPosition;
   unsigned int i;
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

```
apple.com
APPLE.com
apple.comm
```

Run

zyDE 7.20.2: Validate domain names with functions.

A solution to the above problem follows.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Load default template...

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

const int MAX_NUMS = 4;  // Global variable used for

// ***********************************************
// GetPeriodPosition - Pass a string and return the p
// ***********************************************
int GetPeriodPosition(string stringToSearch) {
   int periodCounter;
   int periodPosition;
   unsigned int i;
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

```
apple.com
APPLE.com
apple.comm
```

Run

7.21 Lab training: Unit tests to evaluate your program

Auto-graded programming assignments may use a *Unit test* to test small parts of a program. Unlike a *Compare output test*, which evaluates your program's output for specific input values, a *Unit test* evaluates individual functions to determine if each function:

- is named correctly and has the correct parameters and return type
- calculates and returns the correct value (or prints the correct output)

This example lab uses multiple unit tests to test the KiloToPounds() function.

Complete a program that takes a weight in kilograms as input, converts the weight to pounds, and then outputs the weight in pounds. 1 kilogram = 2.204 pounds (lbs).

Ex: If the input is:

```
10
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

the output is:

```
22.04 lbs
```

Note: Your program must define the function
`double KiloToPounds(double kilos)`

The program below has an error in the KiloToPounds() function.

1. Try submitting the program for grading (click "Submit mode", then "Submit for grading"). Notice that the first two test cases fail, but the third test case passes. The first test case fails because the program outputs the result from the KiloToPounds() function, which has an error. The second test case uses a *Unit test* to test the KiloToPounds() function, which fails.
2. Change the KiloToPounds() function to multiply the variable kilos by 2.204, instead of dividing. The return statement should be: `return (kilos * 2.204);` Submit again. Now the test cases should all pass.

Note: A common error is to mistype a function name with the incorrect capitalization. Function names are case sensitive, so if a lab program asks for a KiloToPounds() function, a kiloToPounds() function that works for you in develop mode will result in a failed unit test (the unit test will not be able to find KiloToPounds()).

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

539740.3879454.qx3zqy7

LAB ACTIVITY | 7.21.1: Lab training: Unit tests to evaluate your program | 0 / 3

main.cpp

```
1 Loading latest submission...
```

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

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Coding trail of your work What is this?

Retrieving signature

7.22 LAB: Swapping variables

Define a function named SwapValues that takes four integers as parameters and swaps the first with the second, and the third with the fourth values. Then write a main program that reads four integers from input and calls function SwapValues() to swap the input values. The main program then prints the swapped values on a single line separated with spaces and ending with a newline.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

The program must define and call the following function:

```
void SwapValues(int& userVal1, int& userVal2, int& userVal3, int& userVal4)
```

Ex: If the input is:

```
3 8 2 4
```

function SwapValues() stores 8, 3, 4, and 2 in userVal1, userVal2, userVal3, and userVal4, respectively. The main program then outputs:

```
8 3 4 2
```

Function SwapValues() swaps the values referenced by the parameters and does not print any output.

539740.3879454.qx3zqy7

LAB ACTIVITY | 7.22.1: LAB: Swapping variables | 0 / 10

main.cpp — Load default template...

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

/* Define your function here */

int main() {
   /* Type your code here. Your code must call the function.  */

   return 0;
}
```

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.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

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

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

Coding trail of your work What is this?

```
History of your effort will appear here once you begin working
on this zyLab.
```

7.23 LAB: Flip a coin

Define a function named CoinFlip that returns "Heads" or "Tails" according to a random value 1 or 0. Assume the value 1 represents "Heads" and 0 represents "Tails". Then, write a main program that reads the desired number of coin flips as an input, calls function CoinFlip() repeatedly according to the number of coin flips, and outputs the results. Assume the input is a value greater than 0.

Hint: Use the modulo operator (%) to limit the random integers to 0 and 1.

Ex: If the random seed value is 2 and the input is:

```
3
```

the output is:

```
Tails
Heads
Tails
```

Note: For testing purposes, a pseudo-random number generator with a fixed seed value is used in the program. The program uses a seed value of 2 during development, but when submitted, a different seed value may be used for each test case.

The program must define and call the following function:
`string CoinFlip()`

539740.3879454.qx3zqy7

LAB ACTIVITY | 7.23.1: LAB: Flip a coin 0 / 10

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

main.cpp Load default template...

```
1 #include <iostream>
2 #include <cstdlib>
3 using namespace std;
4
5 /* Define your function here */
6
7 int main() {
8    // Add more variables as needed
9
```

```
10    srand(2);  // Unique seed
11
12    /* Type your code here */
13
14    return 0;
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.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

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.

7.24 LAB: Sort a vector

Define a function named SortVector that takes a vector of integers as a parameter. SortVector() modifies the vector parameter by sorting the elements in descending order (highest to lowest). Then write a main program that reads a list of integers from input, stores the integers (starting from the second integer) in a vector, calls SortVector(), and outputs the sorted vector. The first input integer indicates how many numbers are in the list.

Ex: If the input is:

```
5 10 4 39 12 2
```

the output is:

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

```
39,12,10,4,2,
```

For coding simplicity, follow every output value by a comma, including the last one.

Your program must define and call the following function:

```
void SortVector(vector<int>& myVec)
```

Hint: Sorting a vector can be done in many ways. You are welcome to look up and use any existing algorithm. Some believe the simplest to code is bubble sort: https://en.wikipedia.org/wiki/Bubble_sort. But you are welcome to try others: https://en.wikipedia.org/wiki/Sorting_algorithm.

539740.3879454.qx3zqy7

LAB ACTIVITY | 7.24.1: LAB: Sort a vector | 0 / 10

main.cpp Load default template...

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

/* Define your function here */

int main() {
   /* Type your code here */

   return 0;
}
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

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.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

7.25 LAB: Word frequencies - functions

Define a function named GetWordFrequency that takes a vector of strings and a search word as parameters. Function GetWordFrequency() then returns the number of occurrences of the search word in the vector parameter (***case insensitive***).

Then, write a main program that reads a list of words into a vector, calls function GetWordFrequency() repeatedly, and outputs the words in the vector with their frequencies. The input begins with an integer indicating the number of words that follow.

Ex: If the input is:

```
5 hey Hi Mark hi mark
```

the output is:

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

```
hey 1
Hi 2
Mark 2
hi 2
mark 2
```

Hint: Use tolower() to set the first letter of each word to lowercase before comparing.

The program must define and use the following function:

```
int GetWordFrequency(vector<string> wordsList, string currWord)
```

539740.3879454.qx3zqy7

LAB ACTIVITY | 7.25.1: LAB: Word frequencies - functions | 0 / 10

main.cpp — Load default template...

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

/* Define your function here */

int main() {
   /* Type your code here */

   return 0;
}
```

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.

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

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.
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

7.26 LAB: Fibonacci sequence

The Fibonacci sequence begins with 0 and then 1 follows. All subsequent values are the sum of the previous two, for example: 0, 1, 1, 2, 3, 5, 8, 13. Complete the Fibonacci() function, which has an index, n (starting at 0), as a parameter and returns the nth value in the sequence. Any negative index values should return -1.

Ex: If the input is:

```
7
```

the output is:

```
Fibonacci(7) is 13
```

539740.3879454.qx3zqy7

LAB ACTIVITY | 7.26.1: LAB: Fibonacci sequence 0 / 10

main.cpp Load default template...

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

int Fibonacci(int n) {

   /* Type your code here. */

}

int main() {
   int startNum;

   cin >> startNum;
   cout << "Fibonacci(" << startNum << ") is " << Fibonacci(startNum) << endl
```

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

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

©zyBooks 01/31/24 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024

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 17:49 1939727
Rob Daglio
MDCCOP2335Spring2024