6.3 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.

Figure 6.3.1: 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
```

Feedback?

Figure 6.3.2: 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;
}
```

Feedback?

PARTICIPATION ACTIVITY 6.3.1: 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

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

Feedback?

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 6.3.2: Function stub used in incremental program development.

Start 2x speed

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

double ConvKilometersToMiles(double numKm) {
   double milesPerKm = 0.621371;
```

```
   return numKm * milesPerKm;
}
double ConvLitersToGallons(double numLiters) {
   double litersPerGal = 0.264172;
   return numLiters * litersPerGal;
}

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;
}
```

Feedback?

PARTICIPATION ACTIVITY 6.3.3: 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

Feedback?

zyDE 6.3.1: Function stubs.

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

Load default

```
1  #include <iostream>
2  using namespace std;
3  // Program converts a trip's kilometers and liters into miles, gallons, and mpg
4
5  double ConvKilometersToMiles(double numKm) {
6     double milesPerKm = 0.621371;
7     return numKm * milesPerKm;
8  }
9
10 double ConvLitersToGallons(double numLiters) {
11    double litersPerGal = 0.264172;
12    return numLiters * litersPerGal;
13 }
14
15 double CalcMpg(double distMiles, double gasGallons) {
16    cout << "FIXME: Calculate MPG" << endl;
17    return 0.0;
18 }
19
20 int main() {
21    double distKm;
```

410.2 33.5

Run

Feedback?

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 6.3.4: Redundant code can be replaced by multiple calls to one function.

Start 2x speed

```
#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;
```

```
#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() {
   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 "
```

```
    return 0;
}
```

main() with redundant code

```
        << totalPizzaArea
        << " inches squared combined." <<
endl;

    return 0;
}
```

main() calls CalcCircleArea() avoiding redundant code

Feedback?

PARTICIPATION ACTIVITY 6.3.5: 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

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

Feedback?

CHALLENGE ACTIVITY 6.3.1: 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;
}
```

```
3
4  /* Your solution goes here  */
5   double MphAndMinutesToMiles(double milesPerHour, double minutesTraveled){
6      double hoursTraveled;
7     double milesTraveled;
8     hoursTraveled = minutesTraveled / 60.0;
9     milesTraveled = hoursTraveled * milesPerHour;
10    return milesTraveled;
11     }
12
13 int main() {
14    double milesPerHour;
15    double minutesTraveled;
16
17    cin >> milesPerHour;
18    cin >> minutesTraveled;
19
20    cout << "Miles: " << MphAndMinutesToMiles(milesPerHour, minutesTraveled) << endl;
21
22    return 0;
23 }
```

Run ✓ All tests passed

✓ Testing with input: 70 100

Your output `Miles: 116.667`

✓ Testing with input: 30 240

Your output `Miles: 120`

✓ Testing with input: 30 0

Your output `Miles: 0`

Feedback?

CHALLENGE ACTIVITY 6.3.2: 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
```

```
2  using namespace std;
3
4  /* Your solution goes here  */
5  int GetUserNum(){
6     cout << "FIXME: Finish GetUserNum()"<< endl;
7     return -1;
8  }
9
10  int ComputeAvg(int userNum1, int userNum2){
11     cout << "FIXME: Finish ComputeAvg()"<< endl;
12    return -1;
13
14  }
15
16 int main() {
17    int userNum1;
18    int userNum2;
19    int avgResult;
20
21    userNum1 = GetUserNum();
22    userNum2 = GetUserNum();
23
```

Run ✓ All tests passed

✓ Testing with two calls to GetUserNum and one call to ComputeAvg

Your output

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

✓ Testing with one call to GetUserNum and two calls to ComputeAvg

Your output

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

Feedback?