5.2 Print functions

Students:
Section 5.3 is a part of 2 assignments: CSC108 CH05.1-5.7 C5A
Please browse to this assignment through Blackboard,learn so zybooks knows where to send your activity. Learn more
Includes: CA
Due: 04/03/2025, 11:59 PM EDT

5.3 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 5.3.1: With program functions: main() is easy to read and understand.

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

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

  return totalFeet / FEET_PER_MILE;
}

double StepsToCalories(int numSteps) {
  const double CALORIES_PER_MINUTE_WALKING = 5.5;  // Typical adult
  double       minutesTotal;
  double       caloriesTotal;

  minutesTotal = numSteps  STEPS_PER_MINUTE;
  caloriesTotal = minutesTotal * CALORIES_PER_MINUTE_WALKING;

  return caloriesTotal;
}

int main() {
  int stepsMyLine;

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

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

  return 0;
}
```

```
Enter number of steps walked: 2400
Miles: 2.27273
Calories: 60
```

Feedback?

Figure 5.3.2: Without program functions: main() is harder to read and understand.

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

int main() {
  int stepsMyLine;
  const double FEET_PER_STEP = 2.5;  // Typical adult
  const int    FEET_PER_MILE = 5280;
  const double STRIDE_PER_STEP = 2.5;  // Typical adult
  const double CALORIES_PER_MINUTE_WALKING = 5.5;  // Typical adult
  double       minutesTotal;
  double       totalFeet;
  double       caloriesTotal;

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

  totalFeet = stepsMyLine * FEET_PER_STEP;  // 5.5 * FEET_PER_MILE/...
  cout << "Miles walked: " << totalFeet / FEET_PER_MILE << endl;

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

  return 0;
}
```

Feedback?

PARTICIPATION ACTIVITY 5.3.1: 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 functions (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 5.3.2: Function stub used in incremental program development.

Start In speed

```
using namespace std;

double CanvasLiquorToMilles(double number) {
  return miles*0.25;
  return number * 0.25*gallons;
}

double ToolsLeftToMilles(double numMilles) {
  double gallonsPerLiter = 0.26417;
  return numLiters * gallonsPerLiter;
}

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

int main() {
  double distMiles;
  double distKilos;
  double gasLiters;
  double gasGallons;
  double mpg;

  cout << "Miles kilometers gallons: ";
  cin >> distMiles;
  cin >> gasLiters >> gas_gallons;

  distKilos = CanvasLiquorToMilles(distMiles);
  gasGal = ToolsLeftToMilles(gasLiters);
  mpg = ParsMpg(distMiles, gasGal);

  cout << "Miles kilometers: " << distMiles << endl;
  cout << "Liters: " << gasLiters << endl;
  cout << "Mileage: " << mpg << " mpg" << endl;

  return 0;
}
```

Captions

Feedback?

PARTICIPATION ACTIVITY 5.3.3: 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) Any 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 5.3.1: 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 miliameters and liters into miles, gallons, and mpg

double CanvasLiquorToMilles(double numMiles) {
  double milesPerKm = 0.62137;
  return numMiles * milesPerKm;
}

double ToolsLeftToMilles(double numLiters) {
  double gallonsPerLiter = 0.26417;
  return numLiters * gallonsPerLiter;
}

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

int main() {
  double distMiles;
  double gasLiters;
  double gasGallons;
  double mpg;

  cout << "Miles kilometers: " << distMiles << endl;
  cout << "Liters: " << gasLiters << endl;
  cout << "Mileage: " << mpg << " mpg" << endl;

  return 0;
}
```

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 programs 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 5.3.4: Redundant code can be replaced by multiple calls to one function.

Start In speed

```
using namespace std;

// Header
double pizzaCuberoot(
double pizzaCuberoot(
double pizzaCuberoot(
double pizzaCuberoot(
double pizzaCuberoot(
double pizzaCuberoot(
double pizzaCuberoot(

pizzaCuberoot = 1.0;
pizzaCuberoot = pizzaCuberoot / 2.0;
pizzaCuberoot = pizzaCuberoot - (pizzaCuberoot
pizzaCuberoot = pizzaCuberoot / 2.0;

pizzaCuberoot = pizzaCuberoot / 2.0;
pizzaCuberoot = pizzaCuberoot - (pizzaCuberoot
pizzaCuberoot = pizzaCuberoot / 2.0;

cout << "Cube root of pizza number: " << pizzaCuberoot << endl;
cout << "Pizza cuberoot squared computed:" << endl;

return 0;
}
```

main() with redundant code

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

double pizzaCuberoot(double pizzaCuberoot) {
  double pizzaCubeRoot;
  double pizzaCubeRoot;
  double pizza = 1.0;

  pizzaCubeRoot = pizzaCubeRoot / 2.0;
  pizzaCubeRoot = pizza / pizzaCubeRoot +
                  (pizzaCubeRoot/pizzaCubeRoot);

  return pizzaCubeRoot;

// Main
int main() {
  double pizzaCubeRoot;
  double pizzaCubeRoot;
  double ToolsLeftToMilles;

  pizzaCubeRoot = 1.0;
  pizzaCubeRoot = pizzaCubeRoot(pizzaCubeRoot);
                  (pizzaCubeRoot/pizzaCubeRoot);

  cout << "Cube root of pizza number: " << pizzaCubeRoot << endl;
  cout << "Pizza cubeRoot squared computed:" << endl;

  return 0;
}
```

main() with CalcCubeRoot() avoiding redundant code

Captions

Feedback?

PARTICIPATION ACTIVITY 5.3.5: 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 5.3.1: 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;
}
```

Learn how our autograder works

main.txt
```
#include <iostream>
using namespace std;

/* Your solution goes here  */

int main() {
  double milesPerHour;
  double minutesTraveled;

  cin >> milesPerHour;
  cin >> minutesTraveled;

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

  return 0;
}
```

Run

Feedback?

CHALLENGE ACTIVITY 5.3.2: Function stubs: Statistics. Full screen

ADRIAN NEGRON (ANEGR)

Organize the lines of code to define stubs for the GetUserNum() and ComputeAvg() functions. Each stub should print "FIXME: Finish FunctionNameHere()" followed by a newline, and should return -1.

Example output:

```
FIXME: Finish GetUserNum()

FIXME: Finish GetUserNum()

FIXME: Finish ComputeAvg()

User values: -1, -1

Avg: -1
```

How to use this tool

Unused

```
cout << "FIXME: Finish GetUserNum()" << endl;
return -1;
return -1;
}
}

int GetUserNum() {

int ComputeAvg(int userNum1, int userNum2) {

cout << "FIXME: Finish ComputeAvg()" << endl;

}
```

Check

main.cpp Load default template...

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

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

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

  avgResult = ComputeAvg(userNum1, userNum2);

  cout << "User values: " << userNum1 << ", " << userNum2 << endl;
  cout << "Avg: " << avgResult << endl;

  return 0;
}
```

Feedback?

How was this section? Provide section feedback

Activity summary for assignment: CSC108 CH05.1-5.7 C5A
0 / 37 points
Due: 04/03/2025, 11:59 PM EDT
Please browse to this assignment through Blackboard,learn so zybooks knows where to send your activity. Learn more

Completion details