Activity #13: Parameter Passing

Recorder's Report

Manager: Reader:

Recorder: Driver:

Date: Score: Satisfactory / Not Satisfactory

Record your team's answers to the key questions (marked with ) below.

a) Model 1, Question #6

b) Model 2, Question #15

c) Model 3, Question #22

Activity #13: Parameter Passing

In this activity, you will work in teams of 3–4 students to learn new concepts. This activity will introduce you to parameter-passing in C++ functions.

Content Learning Objectives

After completing this activity, students should be able to:

- Explain the use of default arguments in C++ functions
- Explain the difference between pass-by-reference and pass-by-value parameters
- Identify a function's signature

Process Skill Goals

During the activity, students should make progress toward:

- Write functions that use default arguments values
- Write functions including pass-by-value and pass-by-reference parameters
- Write functions with the same name but different signatures

Model 1 Two C++ Functions and a Main Program

```
1  int addOneA(int x) {
2    x = x + 1;
3    return x;
4  }
5
6  int addOneB(int &x) {
7    x = x + 1;
8    return x;
9  }
10
11
```

```
1  int main() {
2    int a = 1;
3    cout << "Function: " << addOneA(a) << ", ";
4    cout << "Argument: " << a << endl;
5
6    int b = 1;
7    cout << "Function: " << addOneB(b) << ", ";
8    cout << "Argument: " << b << endl;
9  }
10
```

Refer to Model 2 above as your team develops consensus answers to the questions below.

Questions (20 min) Start time:

1. Without running them, determine what the functions `addOneA` and `addOneB` do.

2. How do the function headers differ between these two functions?

3. The code for this model can be found in `activity13a.cpp`. Run it and record the output.

4. In this activity we will see two ways to pass arguments to a parameter in C++.

- When an argument is *passed by value*, a copy of the argument value is made in the function parameter variable and any changes made to that parameter variable inside the function are thrown away when the function is done.
- When an argument is *passed by reference*, the parameter variable is a reference to the actual argument variable outside the function. Any changes made to the parameter variable inside the function are actually made to the argument variable and persist when the function is done.

5. Based on these two definitions, which function in this model uses pass by reference?

6. By looking at the function header, how can you determine if an argument will be passed by value or passed by reference?

7. Consider the following C++ function.

```
1 // Normally sine and cosine expect angles given in radians
2 // This function returns the sine and cosine of an angle in degrees
3 void getSinCos(double degrees, double &sinOut, double &cosOut) {
4   const double PI = 3.14159265358979323846;
5   double radians = degrees * PI / 180.0;
6   sinOut = sin(radians);
7   cosOut = cos(radians);
8 }
9
```

a) Which parameter(s) take pass by value arguments?

b) Which parameter(s) take pass by reference arguments?

c) What is the return type of this function?

d) What does the comment mean when it says "This function returns the sine and cosine. . . ?"

e) Why couldn't we use a `return` statement at the end of the function to return the sine and cosine?

8. The function below is meant to swap the values of the integer argument variables passed in to it.

```
1  /* function header */ {
2    int temp = x; // save x in temporary var
3    x = y;        // replace x with y
4    y = temp;     // put original x value in y
5  }
6
7  int main() {
8    int a = 2;
9    int b = 3;
10   swap(a,b);
11   cout << a << ", " << b << endl;
12 }
13
```

a) Write a function header for line 1 that makes the program output `2,3`.

b) Write a function header for line 1 that makes the program output `3,2`.

9. Using the function header from (b), which function calls below are valid? Check all that apply.

a) `swap(b,a)` b) `swap(a,2)` c) `swap(3,2)`

Model 2 A C++ Function Prototype

```
1 int sum(int a, int b=0, int c=0, int d=0);
2
3 int main() {
4   cout << "Test One: " << sum(1,2,3,4) << endl;
5   cout << "Test Two: " << sum(1,2,3) << endl;
6   cout << "Test Three: " << sum(1,2) << endl;
7 }
8
```

Output:

```
Test One:  10
Test Two:  6
Test Three:  3
```

Refer to Model 1 above as your team develops consensus answers to the questions below.

Questions (15 min) Start time:

10. A prototype for the function `sum` is defined on line 1 of the model above. Assume that this function is defined elsewhere so as to produce the output shown given the `main` program.

a) How many parameters does the function `sum` have?

b) How many arguments does the call to this function on line 4 have?

c) How many arguments does the call to this function on line 5 have?

d) How many arguments does the call to this function on line 6 have?

11. What are the values of the parameters `a`, `b`, `c`, and `d` in the body of the function `sum` for each call?

a) Line 4: `sum(1,2,3,4)` `a =` `b =` `c =` and `d =`

b) Line 5: `sum(1,2,3)` `a =` `b =` `c =` and `d =`

c) Line 6: `sum(1,2)` `a =` `b =` `c =` and `d =`

12. A *default argument* is a value provided in a function declaration that is automatically assigned by the compiler when function calls don't provide a value for the argument. Which parameters in the `sum` function have default arguments? How can you tell?

13. Rewrite the prototype of the `sum` function from line one so that the function calls below return the indicated value. We will use the notation `sum(1,2,3,4)` $\rightarrow$ 10 to indicate the function call returns 10.

a) `sum(0,0,0)` $\rightarrow$ 2

b) `sum(3,2)` $\rightarrow$ 8

c) `sum(1)` $\rightarrow$ 10

14. The file `activity13b.cpp` contains the code from the model as well as the function definition. Replace the function prototype on line 1 of that file with each of the following. Place a check next to the prototypes for which the program compiles without errors.

a) `int sum(int a, int b=0, int c, int d=0)`

b) `int sum(int a, int b, int c=0, int d=0)`

c) `int sum(int a=0, int b=0, int c, int d)`

d) `int sum(int a=0, int b=0, int c=0, int d=0)`

15. Based on these results, what rule does C++ enforces in regard to default arguments?

16. Follow the steps below to write a function that returns the product of between two and four arguments, as shown in the examples below. Use the file `activity13b.cpp` to test your code.

- `product(2,2,2,3)` $\rightarrow$ 24
- `product(3,3,5)` $\rightarrow$ 45
- `product(5,5)` $\rightarrow$ 25

a) First, write the function definition based on the function header shown below. Enter your definition below the `main` program in the `activity13a.cpp` file.

```
int product(int a, int b, int c, int d)
```

b) Next, add a function prototype above the `main` program. Include appropriate default arguments.

c) Finally, add the three function calls above to the `main` program. Do they work as expected?

Model 3 Similar C++ Function Prototypes

```
1 int Maximum(int, int);
2 int Maximum(int, int, int);
3 double Maximum(double, double);
4 double Maximum(double, double, double);
5
```

Refer to Model 3 above as your team develops consensus answers to the questions below.

Questions (15 min) Start time:

17. What is the name of the function whose prototype is given on each of the following lines?

a) Line 1

b) Line 2

c) Line 3

d) Line 4

18. On which line number is the function prototype to which each function call below matches? Try running the code in `activity13c.cpp` and reading any compile error messages to help you decide.

a) `cout << Maximum(5,1,3)`

b) `cout << Maximum(5,1.5,3)`

c) `cout << Maximum(5,3)`

d) `cout << Maximum(2.3,4.7)`

19. How could you tell which function call go with which prototype?

20. A function's *signature* is its name and the number, type, and order of its parameters. Give prototypes for at least two additional functions with the name `Maximum` that have different function signatures.

21. Give at least one new (different) prototype for a function named `Maximum` that has the same signature as one of those in the model.

22. Add your prototypes from the last two problems to the file `activity13c.cpp` to see which the compiler accepts and which produce errors. Based on that, make a conjecture about when you can have functions with the same name in C++.