CSC 211: Compute Programming

Scope, Parameter passing, Call stack

Michael Conti

Department of Computer Science and Statistics
University of Rhode Island

Fall 2022

Administrative Notes

- Exam#01 next Tuesday
  - Calculator without internet Ok (no phone)
  - 8x11 hand written cheat sheet
  - last day of exam content
- A02 due 10/16

Scope of Variables, Passing Parameters

Scope (where is a variable visible?)

- **Local** variables
  - local to a function, cannot be used outside the function
- **Global** variables
  - available to all functions in the same program
  - declared outside any function
  - not recommended, make programs difficult to maintain
- **Global** constants
  - same as global variables, but require the `const` type qualifier

A Global Named Constant (*part 1 of 2*)

```
//Computes the area of a circle and the volume of a sphere.
//Uses the same radius for both calculations.
#include <iostream>
#include <cmath>
using namespace std;

const double PI = 3.14159;

double area(double radius);
//Returns the area of a circle with the specified radius.

double volume(double radius);
//Returns the volume of a sphere with the specified radius.

int main()
{
    double radius_of_both, area_of_circle, volume_of_sphere;

    cout << "Enter a radius to use for both a circle\n"
         << "and a sphere (in inches): ";
    cin >> radius_of_both;

    area_of_circle = area(radius_of_both);
    volume_of_sphere = volume(radius_of_both);

    cout << "Radius = " << radius_of_both << " inches\n"
         << "Area of circle = " << area_of_circle
         << " square inches\n"
         << "Volume of sphere = " << volume_of_sphere
         << " cubic inches\n";

    return 0;
}
```

A Global Named Constant (*part 2 of 2*)

```
double area(double radius)
{
    return (PI * pow(radius, 2));
}

double volume(double radius)
{
    return ((4.0/3.0) * PI * pow(radius, 3));
}
```

Sample Dialogue

```
Enter a radius to use for both a circle
and a sphere (in inches): 2
Radius = 2 inches
Area of circle = 12.5664 square inches
Volume of sphere = 33.5103 cubic inches
```


Block Scope Revisited

Local and Global scope are examples of Block scope. A variable can be directly accessed only within its scope.

```
1   #include <iostream>
2   using namespace std;
3
4   const double GLOBAL_CONST = 1.0;
5
6   int function1 (int param);
7
8   int main()
9   {
10      int x;
11      double d = GLOBAL_CONST;
12
13      for (int i = 0; i < 10; i++)
14      {
15          x = function1(i);
16      }
17      return 0;
18  }
19
20  int function1 (int param)
21  {
22      double y = GLOBAL_CONST;
23      ...
24      return 0;
25  }
```

*Block scope: Variable **i** has scope from lines 13-16*

*Local scope to **main:** Variable **x** has scope from lines 10-18 and variable **d** has scope from lines 11-18*

*Global scope: The constant **GLOBAL_CONST** has scope from lines 4-25 and the function **function1** has scope from lines 6-25*

*Local scope to **function1:** Variable **param** has scope from lines 20-25 and variable **y** has scope from lines 22-25*

Passing parameters (pass by value)

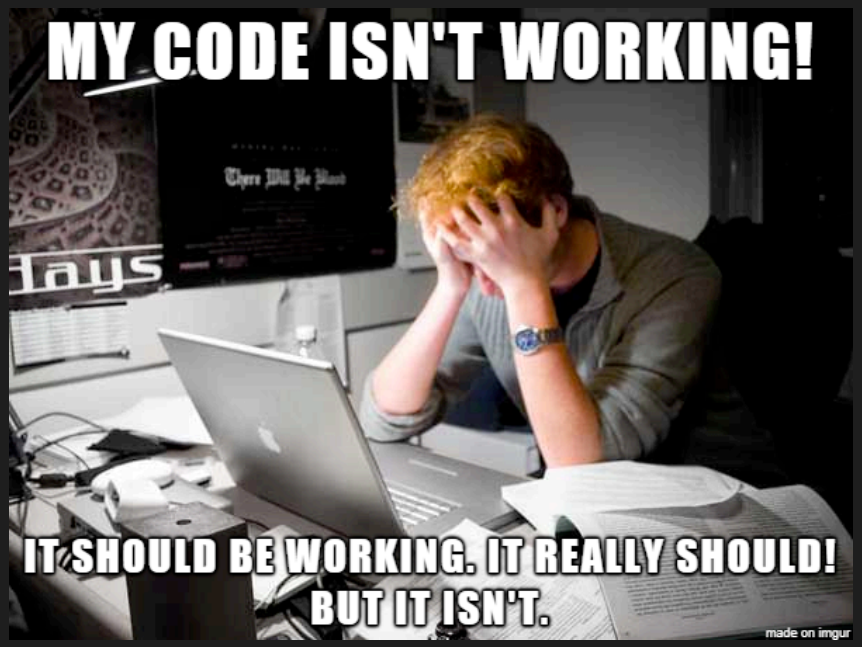
- Parameters are actually **local variables** to the function
- The **pass by value** mechanism (default method)
  - parameters are initialized to the values of the arguments in the function call
  - when invoking a function call, **arguments are copied into the parameters** of a function

Lets try a `swap` function …

```
void swap (int x, int y) {
    int temp;

    temp = x;
    x = y;
    y = temp;

    return;
}
```

What is the output?

```
#include <iostream>

void swap (int x, int y);

int main () {
    int x = 100;
    int y = 200;

    std::cout << "Value of x :" << x << '\n';
    std::cout << "Value of y :" << y << '\n';

    swap(x, y);

    std::cout << "Value of x :" << x << '\n';
    std::cout << "Value of y :" << y << '\n';

    return 0;
}
```

An **Integrated Development Environment** (IDE) usually provides a built-in **debugger**

References

- A **reference** is an alias for another variable
  - just another name for the same memory location

```
int main() {
    int val1 = 1, val2 = 5;
    int &ref = val1;

    val1 += 1;
    ref += 1;
    ref = val2;
    ref *= 2;

    return 0;
}
```

&

Pass by reference

- You can pass arguments to functions **by reference**
- Modifying the reference parameter modifies the actual argument!

```
void swap (int& x, int& y) {
    int temp;

    temp = x;
    x = y;
    y = temp;

    return;
}
```


What is the output

```
#include <iostream>

void mystery(int& b, int c, int& a) {
    a ++;
    b --;
    c += a;
}

int main() {
    int a = 5;
    int b = 10;
    int c = 15;

    mystery(c, a, b);
    std::cout << a << ' ' << b << ' ' << c << '\n';

    return 0;
}
```

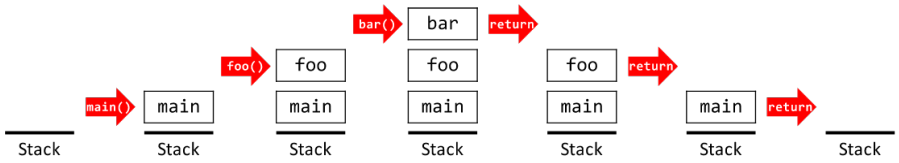

The call stack

Function calls and the call stack

- Variables are stored at different locations in memory
- In practice, it is well more structured …
  - **stack-based memory management** is used by many language implementations
- Program execution needs a **call stack** to deal with functions
  - a **stack frame** stores data for a function call, essentially local variables

Stack frames

```
void bar() {
}

void foo() {
    bar();
}

int main() {
    foo();
}
```



Stack frames (detailed view)

```
#include <iostream>

int plus_one(int x) {
    return x + 1;
}

int plus_two(int x) {
    return plus_one(x + 1);
}

int main() {
    int result = 0;
    result = plus_one(0);
    result = plus_two(result);
    std::cout << result;
}
```


Trace the stack

```
int bar(int b) {
  int c = 0;
  while (b > 0){
    c += 2;
     b -= 2;

  }
  return c;
}

int foo(int a) {
  int temp = 7;
  a = a + bar(temp);
  return a;
}

int main() {
  int a = 5;
  int b = 5;
  int c = foo(a + b);

  return 0;
}
```

Line 7

Report the status of the call stack if we pause the execution of the program exactly at line number 7. Assume the stack grows from top to bottom.

Frame	Variable Name	Current Value

Additional remarks on functions

Preconditions and Postconditions

DISPLAY 5.9 Supermarket Pricing

```
1   //Determines the retail price of an item according to
2   //the pricing policies of the Quick-Shop supermarket chain.
3   #include <iostream>
4   const double LOW_MARKUP = 0.05; //5%
5   const double HIGH_MARKUP = 0.10; //10%
6   const int THRESHOLD = 7;//Use HIGH_MARKUP if ... expected
7                           //to sell in 7 ... s or less
8   void introduction();
9   //Postcondition: Description of program ... written
10  void getInput(double& cost, int& turnover
11  //Precondition: User is ready to enter values correctly.
12  //Postcondition: The value of cost has been set to the
13  //wholesale cost of one item. The value of turnover has been
14  //set to the expected number of days ... the ... s sold
15  double price(double cost, int turn...
16  //Precondition: cost is the wholes...
17  //turnover is the expected number ... day...
18  //Returns the retail price of th... item.
19  void giveOutput(double cost, int turnover, double price);
20  //Precondition: cost is the wholesale cost of one item; turnover is the
21  //expected time until sale of the item; price is the retail price of the item.
22  //Postcondition: The values of cost, turnover, and price ...
23  //written to the screen.
24  int main( )
25  {
26      double wholesaleCost, retailPrice;
27      int shelfTime;
28      introduction( );
29      getInput(wholesaleCost, shelfTime);
30      retailPrice = price(wholesaleCost, shelfTime);
31      giveOutput(wholesaleCost, shelfTime, retailPrice);
32      return 0;
33  }
34    //Uses iostream:
35    void introduction( )
36  {
37      using namespace std;
38      cout<< "This program determines the retail price for\n"
39          << "an item at a Quick-Shop supermarket store.\n";
40  }
41  //Uses iostream:
42  void getIn...(double ... int& turnover)
43  {
44      using ... std;
45      cout ... cost of item: $";
46      cin ... cost
47      cout << "Enter the expected number of days until sold: ";
48      cin >> turnover;
49  }
50  //S... e...:
51  d... (... ver, double price)
52  {
53      using ...
54      cout.setf(ios::fixed);
55      cout.setf(ios::showpoint);
56      cout.precision(2);
57      ...
58          << "Expected ...
59          << turnover <...
60          << "Retail p...
61  }
62  //Uses defined consta... THRESHOLD:
63  double price(double ...
64  {
65      if (turnover <= ...
66          return ( cos...
67      else
68          return ( cost ...
69
70  }
```

from: Problem Solving with C++, 10th Edition, Walter Savitch

Testing and Debugging

- Each function must be tested as a separate an independent unit
- Once properly tested, the function then can be used in the program

Functions must be tested in environments where every other function has already been fully tested and debugged