Chapter 6

Wednesday, August 30, 2017

Chapter 6: Functions

- 6.1 Modular Programming
- 6.2 Defining and Calling Functions
- 6.3 Function Prototypes
- 6.4 Sending Data into a Function
- 6.5 Passing Data by Value
- 6.6 Using Functions in Menu-Driven Programs
- 6.7 The return Statement
- 6.8 Returning a Value from a Function
- 6.9 Returning a Boolean Value
- 6.10 Local and Global Variables
- 6.11 Static Local Variables
- 6.12 Default Arguments
- 6.13 Using Reference Variables as Parameters

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

Just 1 Function

```
int main(){
        statement;
        statement;
}
void function2(){
        statement;
        statement;
}
void function3(){
        statement;
        statement;
}
```

3 Functions

- Broken up into smaller, manageable pieces
- The pieces can be re-used

- Functional Decomposition - broken up into smaller, manageable functions
- Code Re-use - the same statements can be used in multiple locations

6.1 Modular Programming

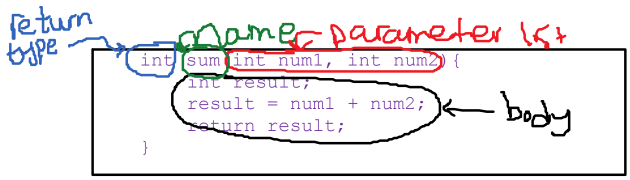
- Functional Decomposition - broken up into smaller, manageable functions
- Function - bundle of statements that perform a task

6.2 Defining and Calling Functions

Function call - statement that causes a function to execute
Function definition - statements that make up a function

Function Definition

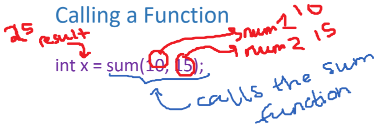
- Return type - data type of value that function sends back
- Name - Function names follow same rules as variables
- Parameter List - Variables passed to the function
- Body - statements that perform the task, enclosed in {}



```
int sum(int num1, int num2) {
    int result;
    result = num1 + num2;
    return result;
}
```

If a function does not return a value, its return type is `void`.

Calling a Function



int x = sum(10, 15);

6.3 Function Prototypes

Call a Function before Defining It

-Place function prototype before main function

Define a Function before Calling It

-No function prototype needed

With Prototype

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

int sum(int, int);   ← prototype

int main(){
      int a = 10;
      int b = 15;
      cout << "Now calling sum function..." << endl;
      int x = sum(a,b);
      cout <<"The sum function returned " << x << endl;
      system("pause");
      return 0;
}
                a 10      b 15
int sum(int num1, int num2){
      int result;
        25
      result = num1 + num2;
      return result;
}
```

Without Prototype

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

int sum(int num1, int num2){
      int result;
      result = num1 + num2;
      return result;
}

int main(){
      int a = 10;
      int b = 15;
      cout << "Now calling sum function..." << endl;
      int x = sum(a, b);
      cout <<"The sum function returned " << x << endl;
      system("pause");
      return 0;
}
```

Output:

```
Now calling sum function...
The sum function returned 25
Press any key to continue...
```

6.4 Sending Data into a Function

- Can pass values into a function at time of call:
  - Example: int x = sum(a, b);
  - c = pow(a,b);
- Values passed to a function are arguments (e.g., a and b)
-

Arguments	Values passed into a function at time of call
Parameters	Variables in header of function definition

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

void displayValue(int);

int main(){                    argument
     displayValue(2017);
     system("pause");
     return 0;                  parameter
}

void displayValue(int year){
     cout << "The year is "<< year << endl;
}
```

Output:

```
The year is 2017
```

Parameters, Prototypes, and Function Headers

Prototype	Must include the data type of each parameter inside parentheses
Header	Must include a declaration i.e. data type and name for each parameter in its ()
Call	Must include the argument

```
void evenOrOdd(int num) ← header
void evenOrOdd(int); ← prototype
evenOrOdd(5);    ← call
```

Passing Multiple Arguments

- When calling a function and passing multiple arguments
  - The number of arguments in the call must match the prototype and definition
    - The 1st argument will be used to initialize the 1st parameter

- The 2nd argument will be used to initialize the 2nd parameter
- And so on…

6.5 Pass By Value

- Pass by value: When an argument is passed into a function, its value is copied into the parameter.
  - Changes to the parameter in the function do not affect the value of the argument.

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

//Prototype
void triple(int);

int main() {
     int val = 5;
     cout << "Val before function call: " << val << endl;
     triple(val);
     cout << "Val after function call: " << val << endl;
     system("pause");
}

void triple(int num) {
     num *= 3;
     cout << num << endl;
}
```

Output:

```
Val before function call: 5
15
Val after function call: 5
Press any key to continue . . .
```

`triple` can change variable `num`, but it will have no effect on variable `val.`

6.6 Using Functions in Menu-Driven Programs

To implement user choices from menu

- To implement general-purpose tasks
- Minimize total number of functions
- Speed up development time
- Example: Program 6.10

6.7 The return Statement

- Used to end the execution of a function
- Can be placed anywhere in a function
- Statements that follow the return statement will not be executed
- Value returning functions must have a return statement
- Void functions do not need a return statement. They end when last closing curly brace } is reached

```
void divide(double arg1, double arg2){
     if (arg2 != 0.0)
          cout << arg1 / arg2 << endl;
     else
          cout << "Is undefined" << endl;
}
```

```
void divide(double arg1, double arg2){
     if (arg2  == 0.0){
          cout << "Is undefined" << endl;
          return;
     }
     cout << arg1 / arg2 << endl;
     }
```

6.8 Returning a Value from a Function

return type

```
int sum (int num1, num2){
     int result;
     result = num1 + num2;
     return result;
```

40 20

```
int sum (int num1, num2){
    int result;
    result = num1 + num2;
    return result;
}
```

40

20

60

value being returned

- A function can return a value back to the statement that called it.

int value1 = 20, int value2 = 40;

total = sum (value1, value2);

6.9 Returning a Boolean Value

Function can return true or false
Declare the return type as bool

```
bool isPrime(int number){
    for (int ctr = 2; ctr < number; ctr++){
    //if number is divisible by something other than 1 or itself
        if (number % ctr == 0){
            return false;
        }
    }
    return true;
}
```

6.10 Local and Global Variables

Local Variables - Variables defined inside a function

- Hidden from statements in other functions
- All parameters are local variables
- Not automatically initialized (they must be initialized by the programmer)

Global Variables - Variables defined outside all the functions in a program

- Scope of a global variable is portion of program from variable declaration to end
- Should avoid in general
- Global variables are usually global constants.
- Are automatically initialized to 0 (numeric or boolean) or NULL (character)

```
//Global constants
const double PAY_RATE = 22.55;
const double BASE_HOURS = 40.0;
const double OT_MULTIPLIER = 1.5;

int main(){
    …
    if (hours > BASE_HOURS)
        overtime = getOvertimePay(hours);
    …
}
double getOvertimePay(int hoursWorked){
    …
    if (hoursWorked > BASE_HOURS){
        overtimePay = (hoursWorked - BASE_HOURS) +
            PAY_RATE * OT_MULTIPLER:
    }
    …
}
```

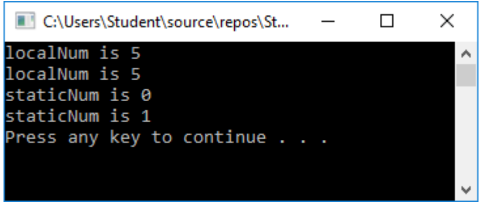
6.11 Static Local Variables

- Contents of local variables are lost when function terminates
- Static local variables retain their contents even after a function terminates
- 0 is the default initialization value

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

//Function prototype
void showLocal();
void showStatic();

int main() {
    showLocal();
    showLocal();
    showStatic();
```

```
int main() {
     showLocal();
     showLocal();
     showStatic();
     showStatic();
     system("pause");
     return 0;
}

void showLocal() {
     int localNum = 5; // Local variable
     cout << "localNum is " << localNum << endl;
     localNum = 99;
}

void showStatic() {
     static int staticNum; // Static variable
     cout << "staticNum is " << staticNum << endl;
     staticNum++;
}
```

Output:



```
localNum is 5
localNum is 5
staticNum is 0
staticNum is 1
Press any key to continue . . .
```

6.12 Default Arguments

Default argument - argument passed automatically into a parameter if the argument is missing in the function call

- Must be a constant declared in a prototype
- If no prototype, declared in header.
- Functions can have default arguments for some or all parameters.
- If not all parameters to a function have default values, the defaultless ones are declared first in the parameter list.

void evenOrOdd(int = 0);	Default argument declared in prototype
int getSum(int, int = 0, int = 0)	Default arguments declared in header

int getSum(int, int = 0, int); // BAD

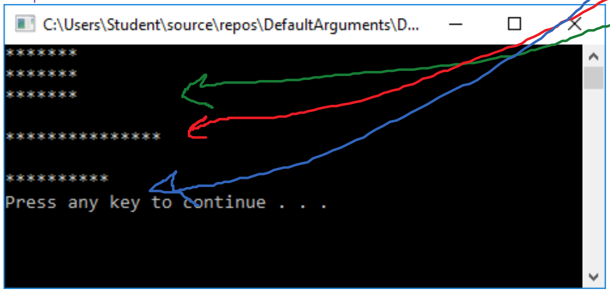
When an argument is omitted from a function call, all arguments after it must be omitted:

sum = getSum(num1, num2);	// OK, the third argument will get set to the default value of 0
sum = getSum(num1, , num3);	// BAD, if you omit any argument, you must omit all following arguments

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

void displayStars(int = 10, int = 1);

int main() {
     displayStars(7, 3);
     cout << endl;
     displayStars(15);
     cout << endl;
     displayStars();
     system("pause");

}

void displayStars(int cols, int rows) {
     int down = 0;
     while (down < rows) {

          int across = 0;
          while (across < cols) {

               cout << "*";
```

```
        int across = 0;
        while (across < cols) {

            cout << "*";
            across++;
            }
        down++;
        cout << endl;
    }
}
```

Output:



```
*******
*******
*******

***************

**********
Press any key to continue . . .
```

6.13 Pass By Reference

Functions can work with

- original argument from the function call (pass by reference)
- or a copy of the argument (pass by value)

- Pass by Reference allows functions to modify the values of arguments.

void getDimensions(int&, int&);	prototype for a function with both arguments passed by reference
void divideByTwo(double& a)	Header for a function with an argument passed by reference

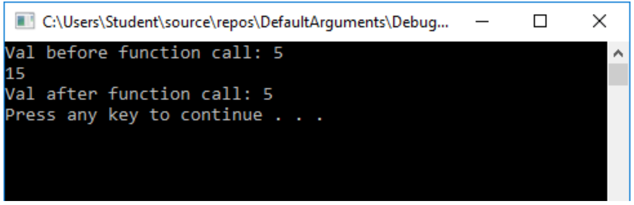
Pass by Value

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

//Prototype
void triple(int);

int main(){
    int val = 5;
    cout << "Val before function call: " << val << endl;
    triple(val);  ← pass by value
    cout << "Val after function call: " << val << endl;
    system("pause");
    return 0;
}

void triple(int num){
    num *=3;
    cout << num << endl;
}
```

Pass by Reference

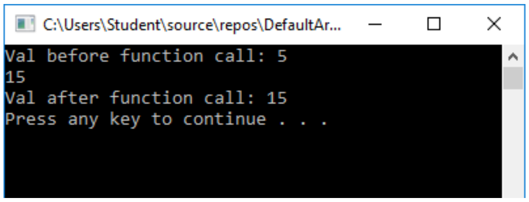
```
#include <iostream>
using namespace std;

//Prototype
void triple(int&);

int main(){
    int val = 5;
    cout << "Val before function call: " << val << endl;
    triple(val);  ← pass by reference
    cout << "Val after function call: " << val << endl;
    system("pause");
    return 0;
}

void triple(int& num){
    num *=3;
    cout << num << endl;
}
```

Output:

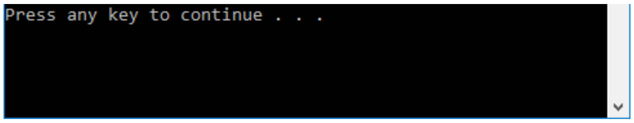


```
Val before function call: 5
15
Val after function call: 5
Press any key to continue . . .
```

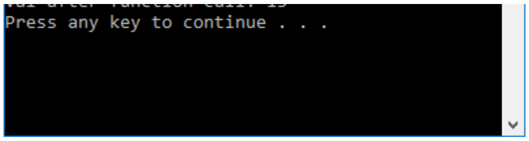
Output:



```
Val before function call: 5
15
Val after function call: 15
Press any key to continue . . .
```



```
Press any key to continue . . .
```



```
val after function call: 15
Press any key to continue . . .
```