Chapter 6 - Functions

The big idea behind this chapter is ....

It relates to the previous chapter how ...

The main purpose of this chapter is ...

The key questions are ...

Why:

When:

How:

Why is this material at this point in the class?

You'll know this material when ...

Main assumptions are ...

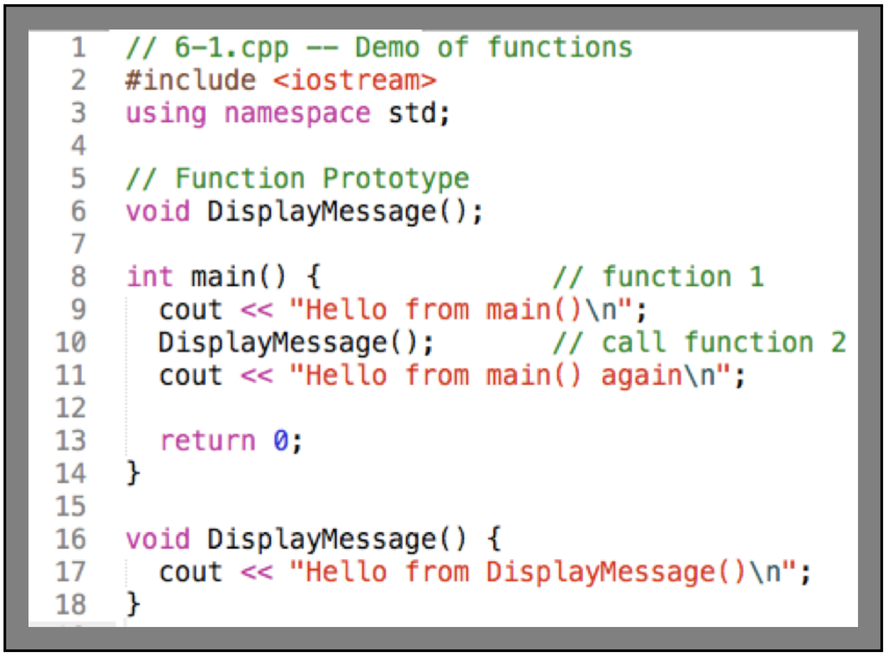
Opening Thoughts. Write any thoughts or questions you have before reading this material. See if you can find the answers while you read.

Key Ideas. Record major points from the chapter.

Idea	Notes

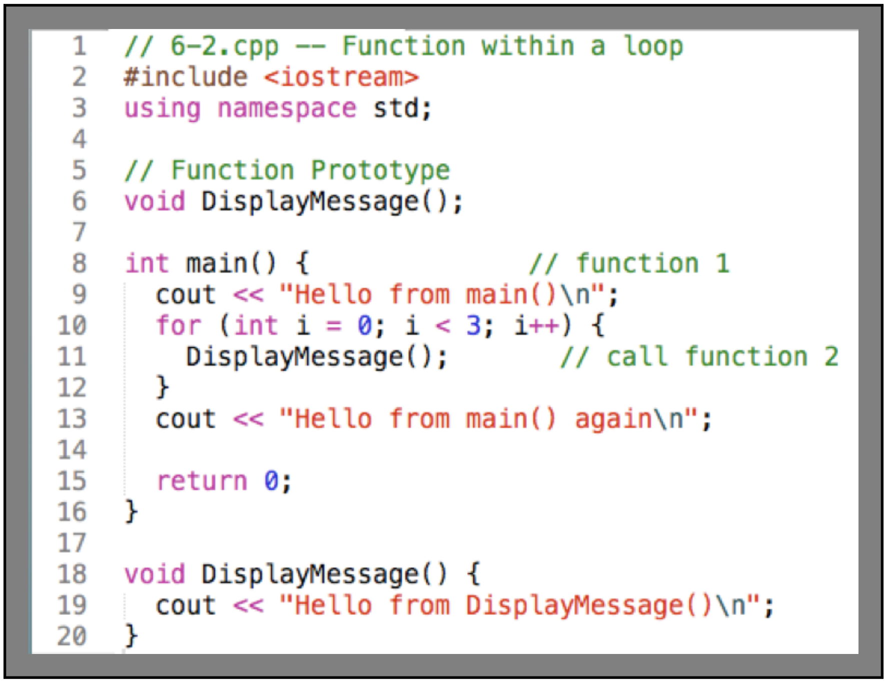
List of Figures

Demonstration Code.



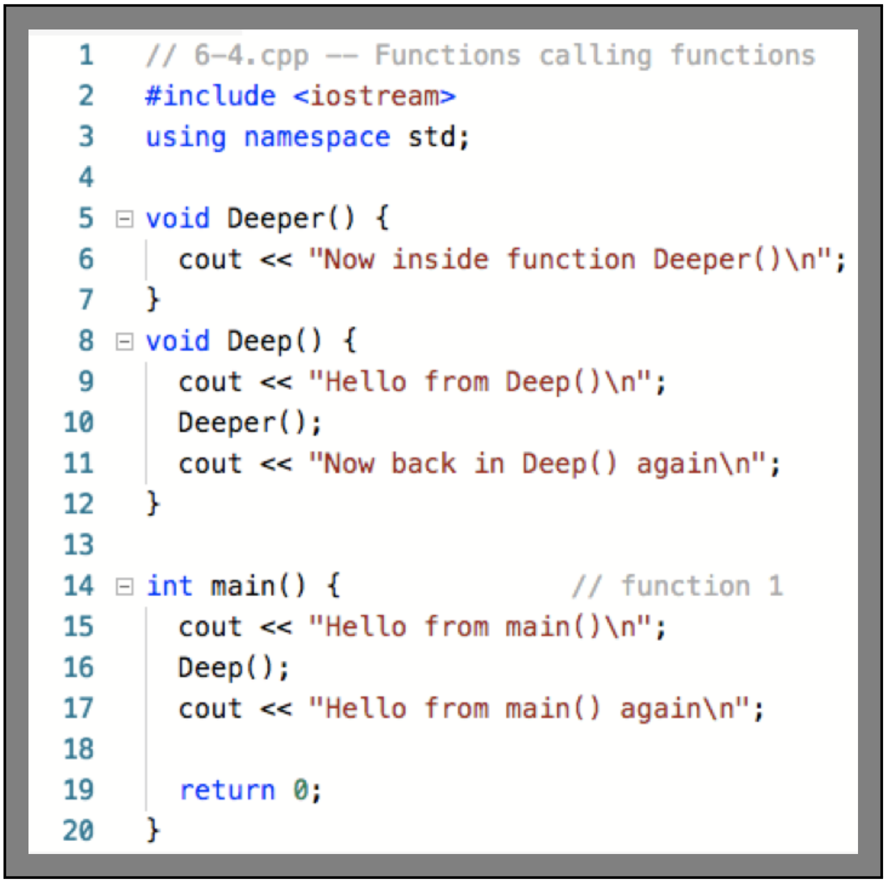
```cpp
// 6-1.cpp -- Demo of functions
#include <iostream>
using namespace std;

// Function Prototype
void DisplayMessage();

int main() {              // function 1
  cout << "Hello from main()\n";
  DisplayMessage();       // call function 2
  cout << "Hello from main() again\n";

  return 0;
}

void DisplayMessage() {
  cout << "Hello from DisplayMessage()\n";
}
```

Figure 1: §6.2 Function demo. Source file: 6-1.cpp



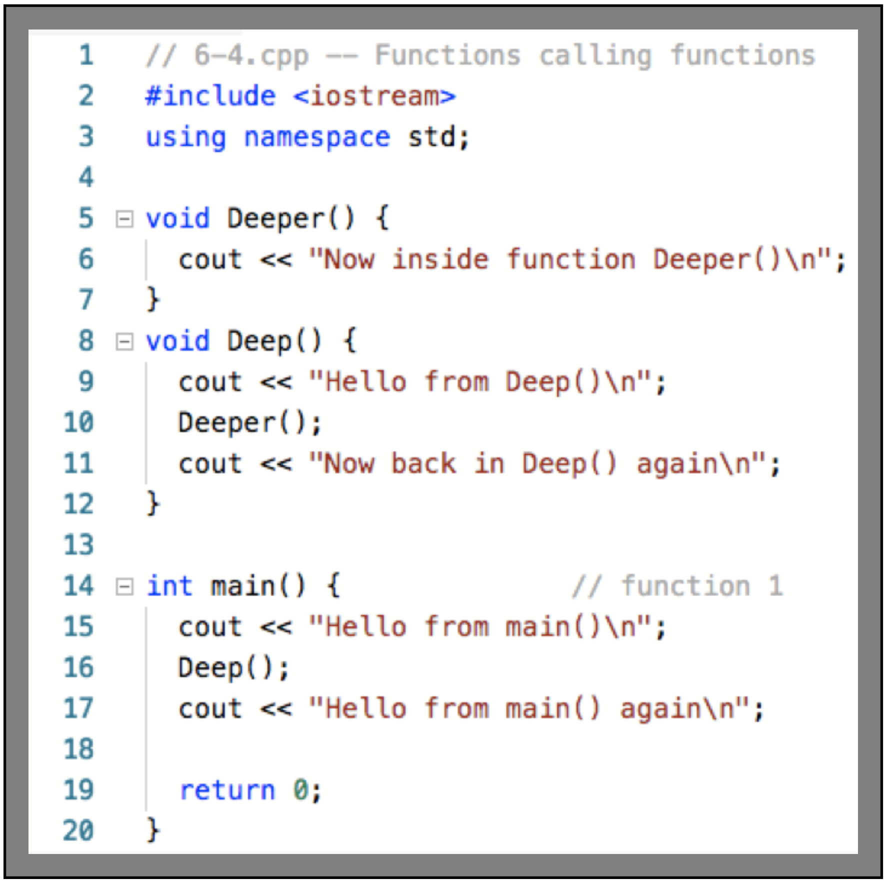
```cpp
// 6-2.cpp -- Function within a loop
#include <iostream>
using namespace std;

// Function Prototype
void DisplayMessage();

int main() {                // function 1
  cout << "Hello from main()\n";
  for (int i = 0; i < 3; i++) {
    DisplayMessage();       // call function 2
  }
  cout << "Hello from main() again\n";

  return 0;
}

void DisplayMessage() {
  cout << "Hello from DisplayMessage()\n";
}
```

Figure 2: §6.2 Demo of function in a loop. Source file: 6-2.cpp



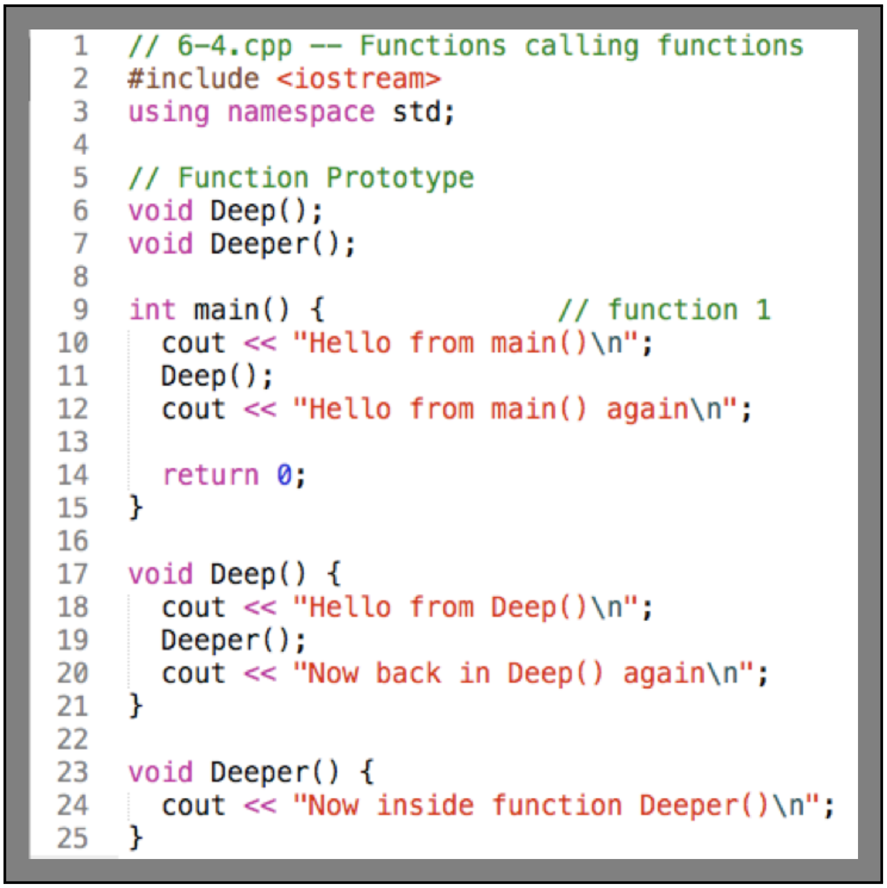
```cpp
// 6-4.cpp -- Functions calling functions
#include <iostream>
using namespace std;

void Deeper() {
  cout << "Now inside function Deeper()\n";
}
void Deep() {
  cout << "Hello from Deep()\n";
  Deeper();
  cout << "Now back in Deep() again\n";
}

int main() {                // function 1
  cout << "Hello from main()\n";
  Deep();
  cout << "Hello from main() again\n";

  return 0;
}
```

Figure 3: §6.2 Functions calling functions. Source file: 6-4.cpp



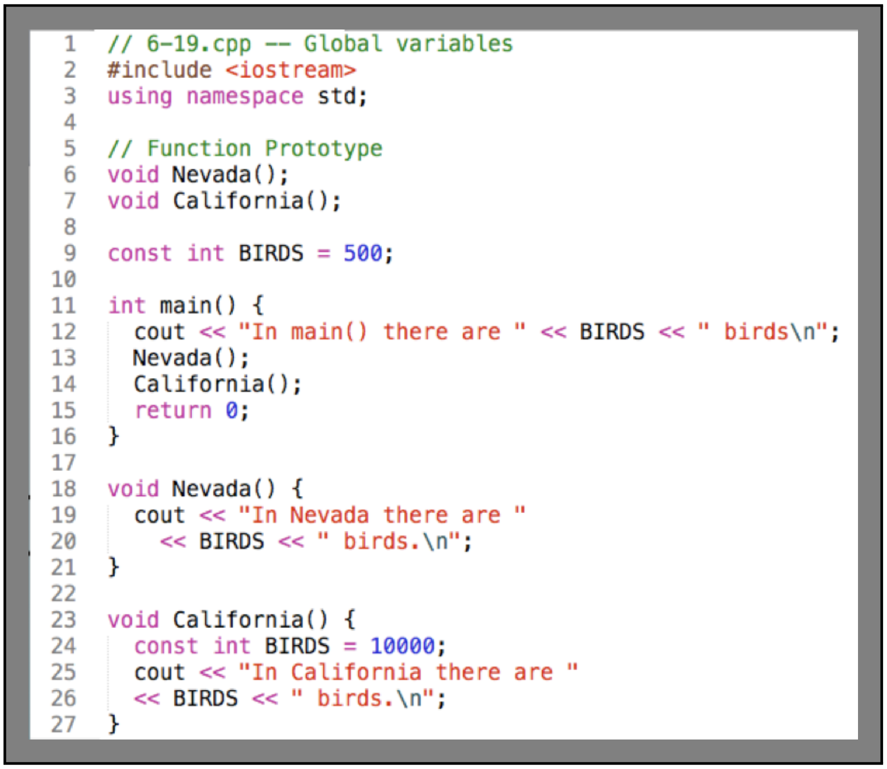
```cpp
// 6-4.cpp -- Functions calling functions
#include <iostream>
using namespace std;

void Deeper() {
    cout << "Now inside function Deeper()\n";
}
void Deep() {
    cout << "Hello from Deep()\n";
    Deeper();
    cout << "Now back in Deep() again\n";
}

int main() {                // function 1
    cout << "Hello from main()\n";
    Deep();
    cout << "Hello from main() again\n";

    return 0;
}
```

Figure 4: §6.3 Functions without function prototypes (BAD). Source file: 6-4.cpp



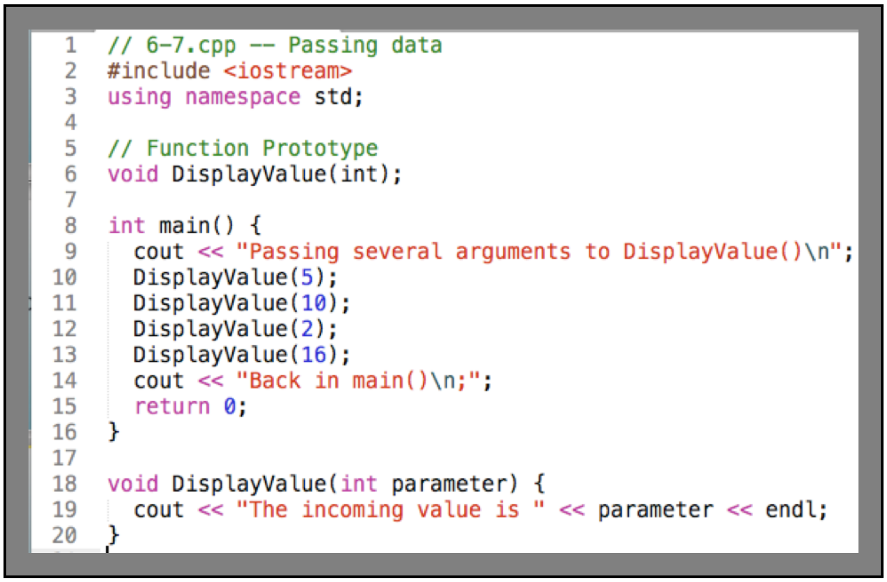
```
// 6-4.cpp -- Functions calling functions
#include <iostream>
using namespace std;

// Function Prototype
void Deep();
void Deeper();

int main() {                 // function 1
  cout << "Hello from main()\n";
  Deep();
  cout << "Hello from main() again\n";

  return 0;
}

void Deep() {
  cout << "Hello from Deep()\n";
  Deeper();
  cout << "Now back in Deep() again\n";
}

void Deeper() {
  cout << "Now inside function Deeper()\n";
}
```

Figure 5: §6.3 Functions with function prototypes (GOOD). Source file: 6-4(fp).cpp



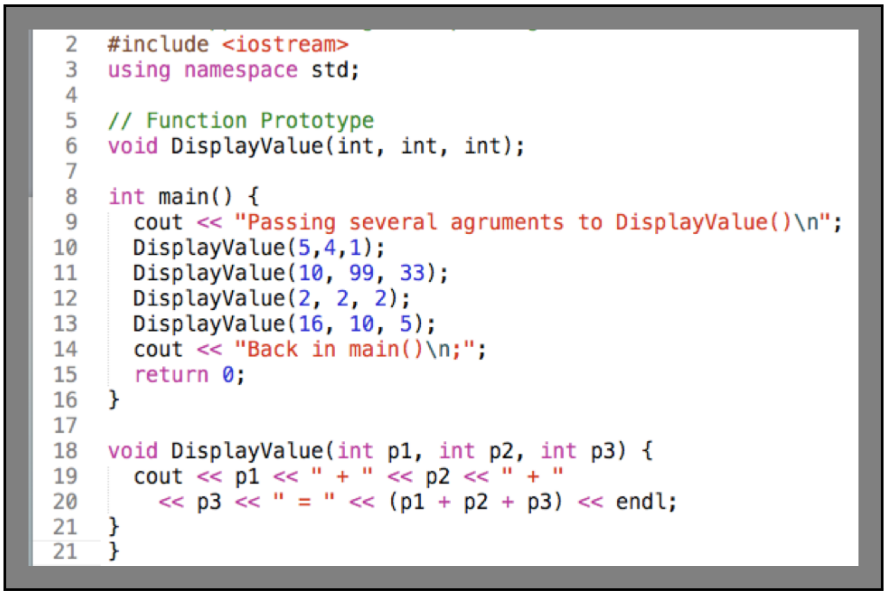
```cpp
// 6-19.cpp -- Global variables
#include <iostream>
using namespace std;

// Function Prototype
void Nevada();
void California();

const int BIRDS = 500;

int main() {
  cout << "In main() there are " << BIRDS << " birds\n";
  Nevada();
  California();
  return 0;
}

void Nevada() {
  cout << "In Nevada there are "
    << BIRDS << " birds.\n";
}

void California() {
  const int BIRDS = 10000;
  cout << "In California there are "
  << BIRDS << " birds.\n";
}
```

Figure 6: §6.10 Demo of global variables. Source file: 6-19.cpp



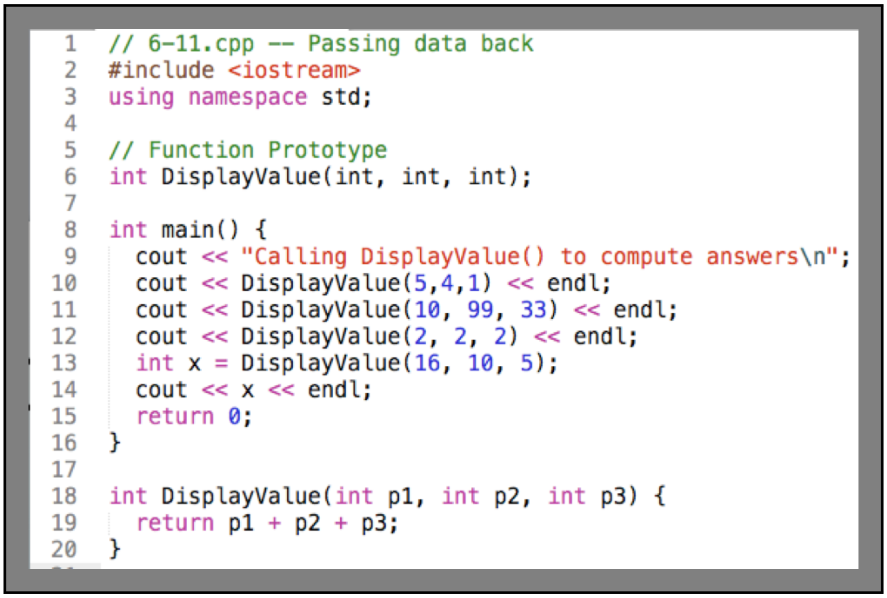
```cpp
// 6-7.cpp -- Passing data
#include <iostream>
using namespace std;

// Function Prototype
void DisplayValue(int);

int main() {
  cout << "Passing several arguments to DisplayValue()\n";
  DisplayValue(5);
  DisplayValue(10);
  DisplayValue(2);
  DisplayValue(16);
  cout << "Back in main()\n;";
  return 0;
}

void DisplayValue(int parameter) {
  cout << "The incoming value is " << parameter << endl;
}
```

Figure 7: §6.4 Passing data example. Source file: 6-7.cpp



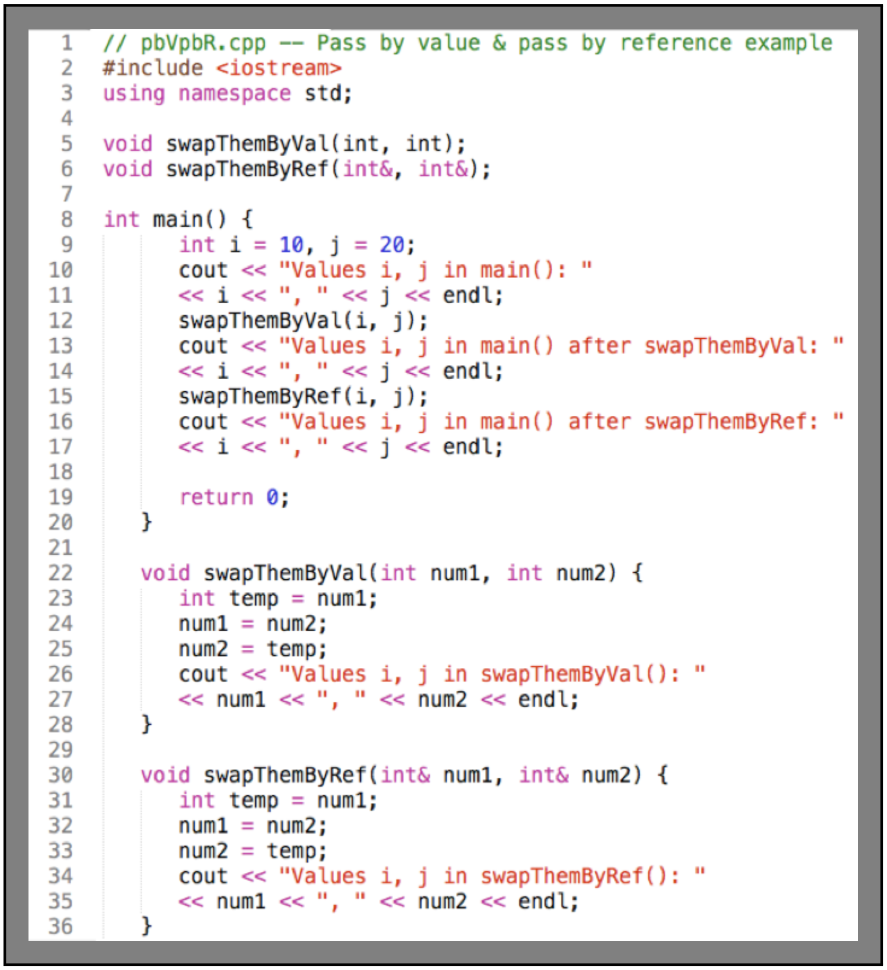
```cpp
#include <iostream>
using namespace std;

// Function Prototype
void DisplayValue(int, int, int);

int main() {
  cout << "Passing several agruments to DisplayValue()\n";
  DisplayValue(5,4,1);
  DisplayValue(10, 99, 33);
  DisplayValue(2, 2, 2);
  DisplayValue(16, 10, 5);
  cout << "Back in main()\n;";
  return 0;
}

void DisplayValue(int p1, int p2, int p3) {
  cout << p1 << " + " << p2 << " + "
    << p3 << " = " << (p1 + p2 + p3) << endl;
}
}
```

Figure 8: §6.4 Passing multiple arguments example. Source file: 6-8.cpp



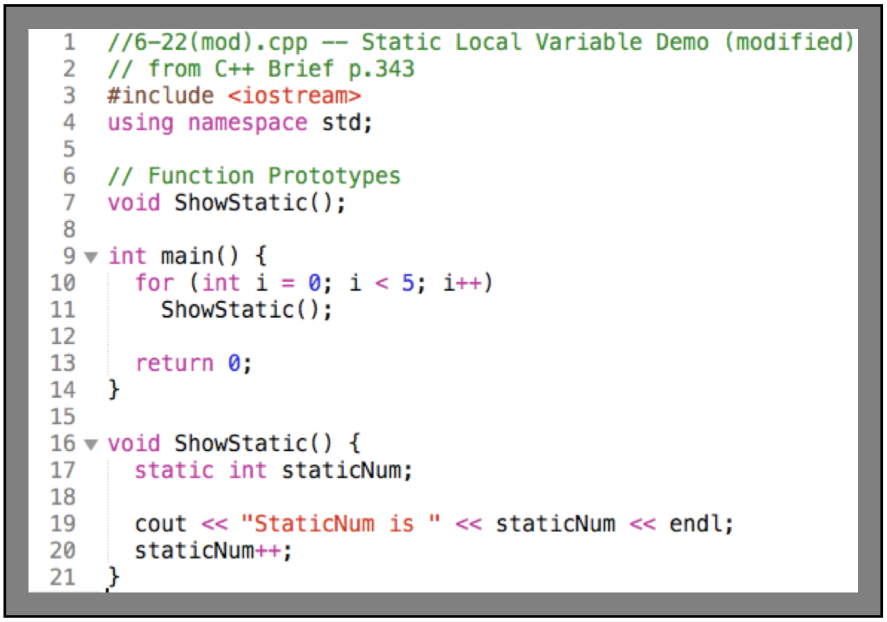
```cpp
// 6-11.cpp -- Passing data back
#include <iostream>
using namespace std;

// Function Prototype
int DisplayValue(int, int, int);

int main() {
  cout << "Calling DisplayValue() to compute answers\n";
  cout << DisplayValue(5,4,1) << endl;
  cout << DisplayValue(10, 99, 33) << endl;
  cout << DisplayValue(2, 2, 2) << endl;
  int x = DisplayValue(16, 10, 5);
  cout << x << endl;
  return 0;
}

int DisplayValue(int p1, int p2, int p3) {
  return p1 + p2 + p3;
}
```

Figure 9: §6.7 Returning data. Source file: 6-11.cpp



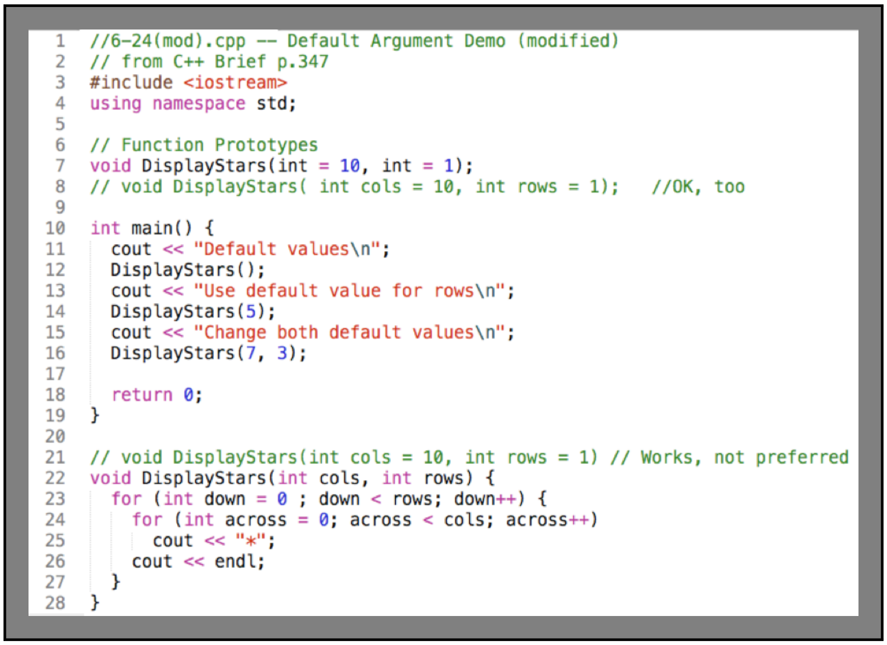
```cpp
// pbVpbR.cpp -- Pass by value & pass by reference example
#include <iostream>
using namespace std;

void swapThemByVal(int, int);
void swapThemByRef(int&, int&);

int main() {
      int i = 10, j = 20;
      cout << "Values i, j in main(): "
      << i << ", " << j << endl;
      swapThemByVal(i, j);
      cout << "Values i, j in main() after swapThemByVal: "
      << i << ", " << j << endl;
      swapThemByRef(i, j);
      cout << "Values i, j in main() after swapThemByRef: "
      << i << ", " << j << endl;

      return 0;
   }

   void swapThemByVal(int num1, int num2) {
      int temp = num1;
      num1 = num2;
      num2 = temp;
      cout << "Values i, j in swapThemByVal(): "
      << num1 << ", " << num2 << endl;
   }

   void swapThemByRef(int& num1, int& num2) {
      int temp = num1;
      num1 = num2;
      num2 = temp;
      cout << "Values i, j in swapThemByRef(): "
      << num1 << ", " << num2 << endl;
   }
```

Figure 10: §6.5 Example of pass by value and pass by reference. Source file: pbVpbR.cpp



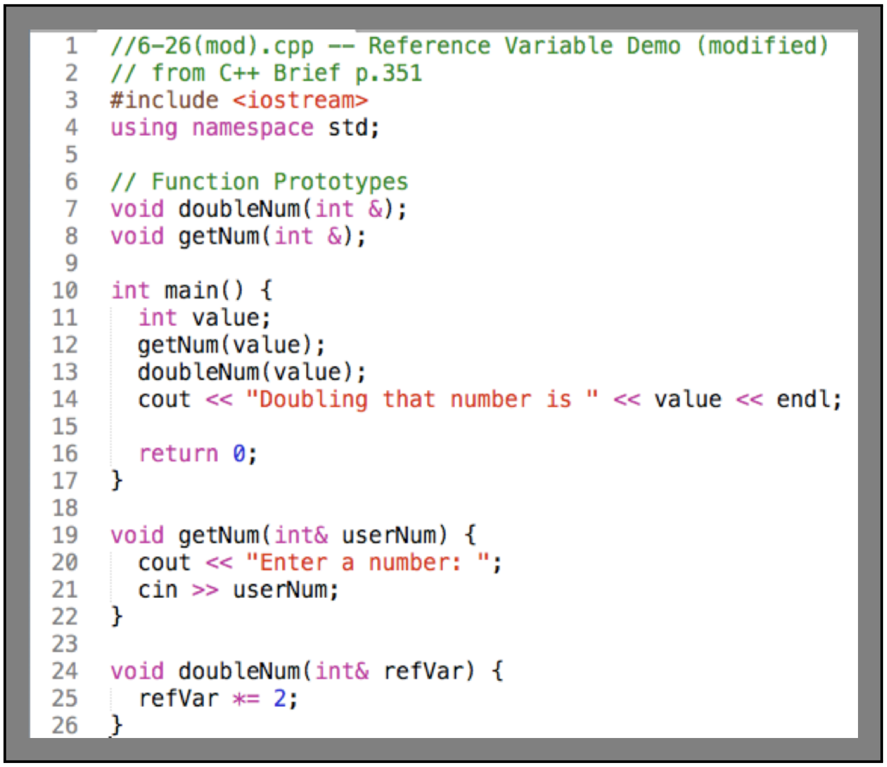
```cpp
//6-22(mod).cpp -- Static Local Variable Demo (modified)
// from C++ Brief p.343
#include <iostream>
using namespace std;

// Function Prototypes
void ShowStatic();

int main() {
  for (int i = 0; i < 5; i++)
    ShowStatic();

  return 0;
}

void ShowStatic() {
  static int staticNum;

  cout << "StaticNum is " << staticNum << endl;
  staticNum++;
}
```

Figure 11: §6.11 Static local variable demo. Source file: 6-22(mod).cpp



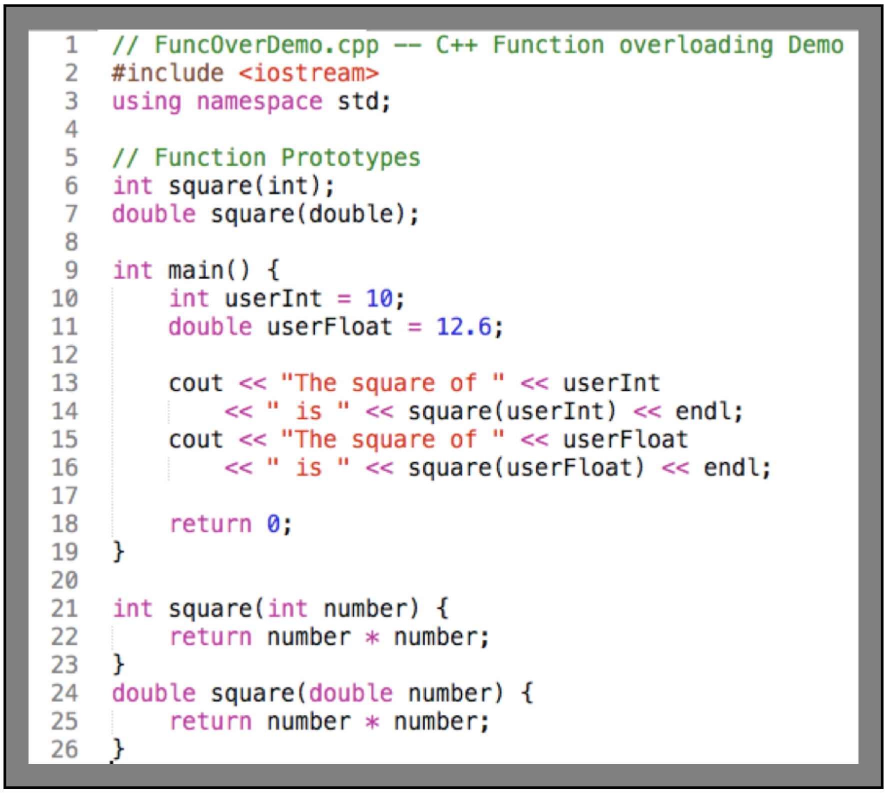
```cpp
//6-24(mod).cpp -- Default Argument Demo (modified)
// from C++ Brief p.347
#include <iostream>
using namespace std;

// Function Prototypes
void DisplayStars(int = 10, int = 1);
// void DisplayStars( int cols = 10, int rows = 1);   //OK, too

int main() {
  cout << "Default values\n";
  DisplayStars();
  cout << "Use default value for rows\n";
  DisplayStars(5);
  cout << "Change both default values\n";
  DisplayStars(7, 3);

  return 0;
}

// void DisplayStars(int cols = 10, int rows = 1) // Works, not preferred
void DisplayStars(int cols, int rows) {
  for (int down = 0 ; down < rows; down++) {
    for (int across = 0; across < cols; across++)
      cout << "*";
    cout << endl;
  }
}
```

Figure 12: §6.12 Default argument demo. Source file: 6-24(mod).cpp



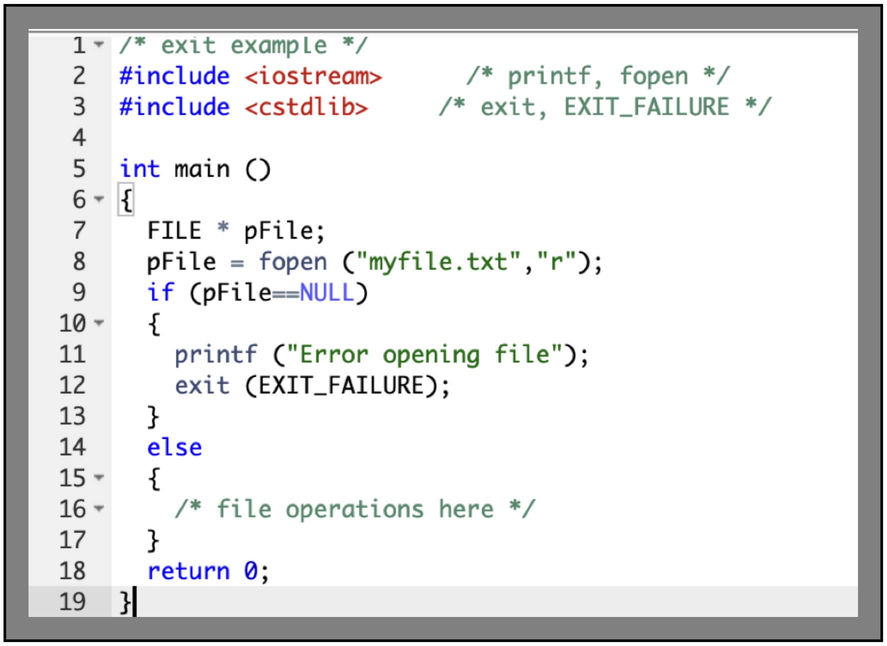
```cpp
//6-26(mod).cpp -- Reference Variable Demo (modified)
// from C++ Brief p.351
#include <iostream>
using namespace std;

// Function Prototypes
void doubleNum(int &);
void getNum(int &);

int main() {
  int value;
  getNum(value);
  doubleNum(value);
  cout << "Doubling that number is " << value << endl;

  return 0;
}

void getNum(int& userNum) {
  cout << "Enter a number: ";
  cin >> userNum;
}

void doubleNum(int& refVar) {
  refVar *= 2;
}
```

Figure 13: §6.13 Reference variable demo. Source file: 6-26(mod).cpp



```cpp
// FuncOverDemo.cpp -- C++ Function overloading Demo
#include <iostream>
using namespace std;

// Function Prototypes
int square(int);
double square(double);

int main() {
    int userInt = 10;
    double userFloat = 12.6;

    cout << "The square of " << userInt
        << " is " << square(userInt) << endl;
    cout << "The square of " << userFloat
        << " is " << square(userFloat) << endl;

    return 0;
}

int square(int number) {
    return number * number;
}
double square(double number) {
    return number * number;
}
```

Figure 14: §6.14 Function overloading demo. Source file: FuncOverDemo.cpp



```cpp
/* exit example */
#include <iostream>      /* printf, fopen */
#include <cstdlib>     /* exit, EXIT_FAILURE */

int main ()
{
  FILE * pFile;
  pFile = fopen ("myfile.txt","r");
  if (pFile==NULL)
  {
    printf ("Error opening file");
    exit (EXIT_FAILURE);
  }
  else
  {
    /* file operations here */
  }
  return 0;
}
```

Figure 15: §6.15 Exit example. Source file: exit.cpp