Chapter 5:
Function Usage Basics & Libraries

Computational assistants

Functions

- Previous examples
  - Programmer-defined functions
    - **main()**
    - **ApiMain()**
  - Library-defined functions
    - **cin.get()**
    - **string** member functions **size()**
    - **RectangleShape** member function **Draw()**
    - **SimpleWindow** member function **Open()**
- Advice
  - Don't reinvent the wheel! There are lots of libraries out there

Terminology

- A function is invoked by a *function call / function invocation*

```
y = f(a);
```

Terminology

- A function call specifies
  - The *function name*
    - The name indicates what function is to be called

```
y = f(a);
```

  - The *actual parameters* to be used in the invocation
    - The values are the information that the called function requires from the invoking function to do its task

```
y = f(a);
```

Terminology

- A function call produces a *return value*
  - The return value is the value of the function call

```
y = f(a);
```

Invocation Process

- *Flow of control* is temporarily transferred to the invoked function
  - Correspondence established between *actual* parameters of the invocation with the *formal* parameters of the definition

```
cout << "Enter number: ";
double a;
cin >> a;
y = f(a);
cout << y;
```

  - Value of **a** is given to **x**

```
double f(double x) {
   double result =
        x*x + 2*x + 5;
   return result;
}
```

Invocation Process

- *Flow of control* is temporarily transferred to the invoked function
  - Local objects are also maintained in the invocation's *activation record*. Even main() has a record

```
cout << "Enter number: ";
double a;
cin >> a;
y = f(a);
cout << y;
```

  - Activation record is large enough to store values associated with each object that is defined by the function

```
double f(double x) {
   double result =
        x*x + 2*x + 5;
   return result;
}
```

Invocation Process

- *Flow of control* is temporarily transferred to the invoked function
  - Other information may also be maintained in the invocation's *activation record*

```
cout << "Enter number: ";
double a;
cin >> a;
y = f(a);
cout << y;
```

  - Possibly a pointer to the current statement being executed and a pointer to the invoking statement

```
double f(double x) {
   double result =
        x*x + 2*x + 5;
   return result;
}
```

Invocation Process

- *Flow of control* is temporarily transferred to the invoked function
  - Next statement executed is the first one in the invoked function

```
cout << "Enter number: ";
double a;
cin >> a;
y = f(a);
cout << y;
```

```
double f(double x) {
   double result =
        x*x + 2*x + 5;
   return result;
}
```

Invocation Process

- *Flow of control* is temporarily transferred to the invoked function
  - After function completes its action, flow of control is returned to the invoking function and the return value is used as value of invocation

```
cout << "Enter number: ";
double a;
cin >> a;
y = f(a);
cout << y;
```

```
double f(double x) {
   double result =
        x*x + 2*x + 5;
   return result;
}
```

Execution Process

- Function body of invoked function is executed
- Flow of control then returns to the invocation statement
- The return value of the invoked function is used as the value of the invocation expression

Function Prototypes

- Before a function can appear in an invocation its interface must be specified
  - *Prototype* or complete definition

Type of value that the function returns

A description of the form the parameters (if any) are to take

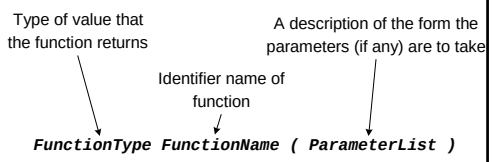
Identifier name of function

```
FunctionType FunctionName ( ParameterList )
```

```
int Max(int a, int b)
```

Function Prototypes

- Before a function can appear in an invocation its interface must be specified
  - Prototypes are normally kept in library header files

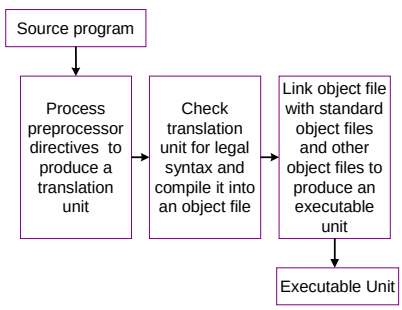


int Max(int a, int b)

Libraries

- Library
  - Collection of functions, classes, and objects grouped by commonality of purpose
  - Include statement provides access to the names and descriptions of the library components
  - Linker connects program to actual library definitions
- Previous examples
  - String: STL's string class
  - Graphics: EzWindows

Basic Translation Process



Some Standard Libraries

- fstream
  - File stream processing
- Assert
  - C-based library for assertion processing
- Iomanip
  - Formatted input/output (I/O) requests
- Ctype
  - C-based library for character manipulations
- Math
  - C-based library for trigonometric and logarithmic functions
- Note
  - C++ has many other libraries

Library Header Files

- Describes library components
- Typically contains
  - Function prototypes
    - Interface description
  - Class definitions
- Sometimes contains
  - Object definitions
    - Example: **cout** and **cin** in **iostream**

Library Header Files

- Typically do not contain function definitions
  - Definitions are in source files
  - Access to compiled versions of source files provided by a linker

```
#include <iostream>
#include <cmath>        <-- Library header files
using namespace std;
int main() {
    cout << "Enter Quadratic coefficients: ";
    double a, b, c;
    cin >> a >> b >> c;
    if ( (a != 0) && (b*b - 4*a*c > 0) ) {        Invocation
        double radical = sqrt(b*b - 4*a*c);
        double root1 = (-b + radical) / (2*a);
        double root2 = (-b - radical) / (2*a);
        cout << "Roots: " << root1 << " " << root2;
    }
    else {
        cout << "Does not have two real roots";
    }
    return 0;
}
```

```
#include <iostream>
#include <fstream>    // file stream library
using namespace std;
int main() {
    ifstream fin("mydata.txt");
    int ValuesProcessed = 0;
    float ValueSum = 0;
    float Value;
    while (fin >> Value) {
        ValueSum += Value;
        ++ValuesProcessed;
    }
    if (ValuesProcessed > 0) {
        ofstream fout("average.txt");
        float Average = ValueSum / ValuesProcessed;
        fout << "Average: " << Average << endl;
        return 0;
    }
    else {
        cerr << "No list to average" << endl;
        return 1;
    }
}
```

```
ifstream sin("in1.txt");   // extract from in1.txt
ofstream sout("out1.txt"); // insert to out1.txt
string s;
while (sin >> s) {
    sout << s << endl;
}
sin.close();         // done with in1.txt
sout.close();        // done with out1.txt
sin.open("in2.txt");   // now extract from in2.txt
sout.open("out.txt",   // now append to out2.txt
(ios_base::out | ios_base::app));
while (sin >> s) {
    sout << s << endl;
}
sin.close();         // done with in2.txt
sout.close();        // done with out2.txt
```