Modular Programming

- Modular programming: breaking a program up into smaller, manageable functions or modules
- Function: a collection of statements to perform a task
- Motivation for modular programming:
  - Improves maintainability of programs
  - Simplifies the process of writing programs

This program has one long, complex function containing all of the statements necessary to solve a problem.

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

In this program, the problem has been divided into smaller problems, each of which is handled by a separate function.


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

main function

```
void function2()
{
   statement;
   statement;
   statement;
}
```

function 2

```
void function3()
{
   statement;
   statement;
   statement;
}
```

function 3

```
void function4()
{
   statement;
   statement;
   statement;
}
```

function 4

Defining and Calling Functions

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

Function Definition

- Definition includes:
  - <u>return type:</u> data type of the value that function returns to the part of the program that called it
  - <u>name:</u> name of the function. Function names follow same rules as variables
  - <u>parameter list:</u> variables containing values passed to the function
  - <u>body:</u> statements that perform the function's task, enclosed in { }

Function Definition

Return type

Function name

Parameter list (this one is empty)

Function body

```
int main ()
{
    cout << "Hello World\n";
    return 0;
}
```

Function Return Type

- If a function returns a value, the type of the value must be indicated:

```
int main()
```

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

```
void printHeading()
{
        cout << "Monthly Sales\n";
}
```

Calling a Function

- To call a function, use the function name followed by `()` and `;`

```
printHeading();
```

- When called, program executes the body of the called function

- After the function terminates, execution resumes in the calling function at point of call.

Functions in Program 6-1

Pr6-1.cpp - Code::Blocks 20.03

```cpp
// This program has two functions: main and displayMessage
#include <iostream>
using namespace std;

//****************************************
// Definition of function displayMessage  *
// This function displays a greeting.     *
//****************************************

void displayMessage()
{
    cout << "Hello from the function displayMessage.\n";
}

//****************************************
// Function main                          *
//****************************************

int main()
{
    cout << "Hello from main.\n";
    displayMessage();
    cout << "Back in function main again.\n";
    return 0;
}
```

"F:\BlackUSB\evc\2022\Fall\CS75\labexamples\Week6Chapter 06\Pr6-1.exe"

```
Hello from main.
Hello from the function displayMessage.
Back in function main again.

Process returned 0 (0x0)   execution time : 0.855 s
Press any key to continue.
```

Logs & others

```
Checking for existence: F:\BlackUSB\evc\2022\Fall\CS75\labexamples\Week6Chapter 06\Pr6-1.exe
Executing: '"C:\Program Files\CodeBlocks/cb_console_runner.exe" "F:\BlackUSB\evc\2022\Fall\CS75\labexamples\Week6Chapter 06\Pr6-1.exe"' (in 'F:\BlackUSB\evc\2022\Fall\CS75\labexamples\Week6Chapter 06')
Set variable: PATH=C:\Program Files\CodeBlocks\MinGW\bin;C:\Program Files\CodeBlocks\MinGW;C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\ProgramData\Oracle\Java\javapath;c:\Program Files (x86)\Intel\iCLS Client;c:\Program Files\Intel\iCLS
Client;C:\WINDOWS\System32;C:\WINDOWS;C:\WINDOWS\System32\wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0;C:\Program Files\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files
(x86)\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files (x86)\Common Files\Acronis\SnapAPI;C:\Program Files (x86)\Acronis\TrueImageHome;C:\Program Files\Microsoft SQL
Server\110\Tools\Binn;C:\Program Files\Common Files\NeatReceipts\Drivers\M12;C:\Program Files (x86)\nodejs;C:\usr\bin;C:\Program Files\dotnet;C:\Program Files\Microsoft SQL Server\130\Tools\Binn;C:\Program Files\Common Files\Autodesk Shared;C:\Program
Files\PuTTY;C:\Program Files\Microsoft SQL Server\120\Tools\Binn;C:\Program Files\Java\jdk1.8.0_171\bin;C:\Users\garry\AppData\Roaming\npm
```

F:\BlackUSB\evc\2022\Fall\CS75\labexamples\Week6Chapter 06\Pr6-1.cpp | C/C++ | Windows (CR+LF) | WINDOWS-1252 | Line 1, Col 1, Pos 0 | Insert | Read/Write | default

3:10 PM 10/7/2022

Flow of Control in Program 6-1

```
void displayMessage()
{
   cout << "Hello from the function displayMessage.\n";
}
```

```
int main()
{
   cout << "Hello from main.\n"
   displayMessage();
   cout << "Back in function main again.\n";
   return 0;
}
```

Calling Functions

- `main` can call any number of functions
- Functions can call other functions
- Compiler must know the following about a function before it is called:
  - name
  - return type
  - number of parameters
  - data type of each parameter

Function Prototypes

- Ways to notify the compiler about a function before a call to the function:
  - Place function definition before calling function's definition
  - Use a <u>function prototype</u> (<u>function declaration</u>) – like the function definition without the body
    - Header: `void printHeading()`
    - Prototype: `void printHeading();`

Prototype Notes

- Place prototypes near top of program
- Program must include either prototype or full function definition before any call to the function – compiler error otherwise
- When using prototypes, can place function definitions in any order in source file

Sending Data into a Function

- Can pass values into a function at time of call:

```
c = pow(a, b);
```

- Values passed to function are <u>arguments</u>

- Variables in a function that hold the values passed as arguments are <u>parameters</u>

A Function with a Parameter Variable

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

The integer variable `num` is a parameter.
It accepts any integer value passed to the function.

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

<global> | displayValue(int num) : void

Management | Projects | Workspace

Pr6-1.cpp | Pr6-6.cpp

```
1   // This program demonstrates a function with a parameter.
2   #include <iostream>
3   using namespace std;
4
5   // Function Prototype
6   void displayValue(int);
7
8   int main()
9   {
10      cout << "I am passing 5 to displayValue.\n";
11      displayValue(5);  // Call displayValue with argument 5
12      cout << "Now I am back in main.\n";
13      return 0;
14  }
15
16  //*********************************************************
17  // Definition of function displayValue.                   *
18  // It uses an integer parameter whose value is displayed. *
19  //*********************************************************
20
21  void displayValue(int num)
22  {
23      cout << "The value is " << num << endl;
24  }
```

"F:\BlackUSB\evc\2022\Fall\CS75\labexamples\Week6Chapter 06\Pr...

```
I am passing 5 to displayValue.
The value is 5
Now I am back in main.

Process returned 0 (0x0)   execution time : 0.472 s
Press any key to continue.
```

Logs & others

Code::Blocks | Search results | Cccc | Build log | CppCheck/Vera++ | CppCheck/Vera++ messages | Cscope | Debugger | DoxyBlocks | Fortran info | Closed files list | Thread search

```
Checking for existence: F:\BlackUSB\evc\2022\Fall\CS75\labexamples\Week6Chapter 06\Pr6-6.exe
Executing: '"C:\Program Files\CodeBlocks/cb_console_runner.exe" "F:\BlackUSB\evc\2022\Fall\CS75\labexamples\Week6Chapter 06\Pr6-6.exe"' (in 'F:\BlackUSB\evc\2022\Fall\CS75\labexamples\Week6Chapter 06')
Set variable: PATH=C:\Program Files\CodeBlocks\MinGW\bin;C:\Program Files\CodeBlocks\MinGW;C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\ProgramData\Oracle\Java\javapath;c:\Program Files (x86)\Intel\iCLS Client;c:\Program Files\Intel\iCLS
Client;C:\WINDOWS\System32;C:\WINDOWS;C:\WINDOWS\System32\wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0;C:\Program Files\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files
(x86)\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files (x86)\Common Files\Acronis\SnapAPI;C:\Program Files (x86)\Acronis\TrueImageHome;C:\Program Files\Microsoft SQL
```

Function with a Parameter in Program 6-6

```
displayValue(5);
```

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

The function call in line 11 passes the value 5 as an argument to the function.

Other Parameter Terminology

- A parameter can also be called a <u>formal parameter</u> or a <u>formal argument</u>
- An argument can also be called an <u>actual parameter</u> or an <u>actual argument</u>

Parameters, Prototypes, and Function Headers

- For each function argument,
  - the prototype must include the data type of each parameter inside its parentheses
  - the header must include a declaration for each parameter in its `()`

```
void evenOrOdd(int);  //prototype
void evenOrOdd(int num) //header
evenOrOdd(val);       //call
```

Function Call Notes

- Value of argument is copied into parameter when the function is called
- A parameter's scope is the function which uses it
- Function can have multiple parameters
- There must be a data type listed in the prototype（）and an argument declaration in the function header（）for each parameter
- Arguments will be promoted/demoted as necessary to match parameters

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 first argument will be used to initialize the first parameter, the second argument to initialize the second parameter, etc.

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

Management

Projects

Workspace

Pr6-1.cpp | Pr6-6.cpp | Pr6-8.cpp

```cpp
// This program demonstrates a function with three parameters.
#include <iostream>
using namespace std;

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

int main()
{
   int value1, value2, value3;

   // Get three integers.
   cout << "Enter three integers and I will display ";
   cout << "their sum: ";
   cin >> value1 >> value2 >> value3;

   // Call showSum passing three arguments.
   showSum(value1, value2, value3);
   return 0;
}

//*************************************************************
// Definition of function showSum.                            *
// It uses three integer parameters. Their sum is displayed. *
//*************************************************************

void showSum(int num1, int num2, int num3)
{
   cout << (num1 + num2 + num3) << endl;
}
```

"F:\BlackUSB\evc\2022\Fall\CS75\labexamples\Week6Chapter 06\Pr6-8.exe"

```
Enter three integers and I will display their sum: 3 6 8
17

Process returned 0 (0x0)   execution time : 16.995 s
Press any key to continue.
```

Logs & others

Code::Blocks | Search results | Cccc | Build log | CppCheck/Vera++ | CppCheck/Vera++ messages | Cscope | Debugger | DoxyBlocks | Fortran info | Closed files list | Thread search

```
Checking for existence: F:\BlackUSB\evc\2022\Fall\CS75\labexamples\Week6Chapter 06\Pr6-8.exe
Executing: '"C:\Program Files\CodeBlocks/cb_console_runner.exe" "F:\BlackUSB\evc\2022\Fall\CS75\labexamples\Week6Chapter 06\Pr6-8.exe"' (in 'F:\BlackUSB\evc\2022\Fall\CS75\labexamples\Week6Chapter 06')
Set variable: PATH=C:\Program Files\CodeBlocks\MinGW\bin;C:\Program Files\CodeBlocks\MinGW;C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\ProgramData\Oracle\Java\javapath;c:\Program Files (x86)\Intel\iCLS Client;c:\Program Files\Intel\iCLS
Client;C:\WINDOWS\System32;C:\WINDOWS;C:\WINDOWS\System32\wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0;C:\Program Files\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files
(x86)\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files (x86)\Common Files\Acronis\SnapAPI;C:\Program Files (x86)\Acronis\TrueImageHome;C:\Program Files\Microsoft SQL
Server\110\Tools\Binn;C:\Program Files\Common Files\NeatReceipts\Drivers\M12;C:\Program Files (x86)\nodejs;C:\usr\bin;C:\Program Files\dotnet;C:\Program Files\Microsoft SQL Server\130\Tools\Binn;C:\Program Files\Common Files\Autodesk Shared;C:\Program
Files\PuTTY;C:\Program Files\Microsoft SQL Server\120\Tools\Binn;C:\Program Files\Java\jdk1.8.0_171\bin;C:\Users\garry\AppData\Roaming\npm
```

Passing Multiple Arguments in Program 6-8

Function Call → `showSum(value1, value2, value3)`

```
void showSum(int num1, int num2, int num3)
{
      cout << (num1 + num2 + num3) << endl;
}
```

The function call in line 18 passes value1, value2, and value3 as a arguments to the function.

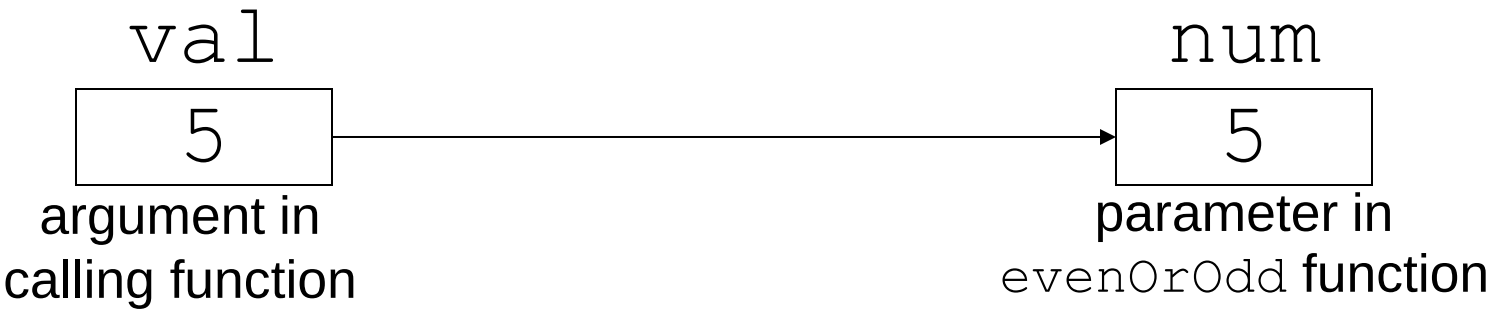
Passing Data by Value

- Pass by value: when an argument is passed to a function, its value is copied into the parameter.

- Changes to the parameter in the function do not affect the value of the argument

Passing Information to Parameters by Value

- Example: `int val=5;`
  `evenOrOdd(val);`



- `evenOrOdd` can change variable `num`, but it will have no effect on variable `val`

Using Functions in Menu-Driven Programs

- Functions can be used
  - to implement user choices from menu
  - to implement general-purpose tasks:
    - Higher-level functions can call general-purpose functions, minimizing the total number of functions and speeding program development time

The `return` Statement

- Used to end execution of a function
- Can be placed anywhere in a function
  - Statements that follow the `return` statement will not be executed
- Can be used to prevent abnormal termination of program
- In a `void` function without a `return` statement, the function ends at its last `}`

Returning a Value From a Function

- A function can return a value back to the statement that called the function.
- You've already seen the `pow` function, which returns a value:

```
double x;
x = pow(2.0, 10.0);
```

Returning a Value From a Function

- In a value-returning function, the `return` statement can be used to return a value from function to the point of call. Example:

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

A Value-Returning Function

Return Type

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

Value Being Returned

A Value-Returning Function

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

Functions can return the values of expressions, such as `num1 + num2`

p - Code::Blocks 20.03

View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

Pr6-1.cpp | Pr6-6.cpp | Pr6-8.cpp | Pr6-12.cpp

```cpp
// This program uses a function that returns a value.
#include <iostream>
using namespace std;

// Function prototype
int sum(int, int);

int main()
{
   int value1 = 20,   // The first value
       value2 = 40,   // The second value
       total;         // To hold the total

   // Call the sum function, passing the contents of
   // value1 and value2 as arguments. Assign the return
   // value to the total variable.
   total = sum(value1, value2);

   // Display the sum of the values.
   cout << "The sum of " << value1 << " and "
        << value2 << " is " << total << endl;
   return 0;
}

//*****************************************************
// Definition of function sum. This function returns  *
// the sum of its two parameters.                     *
//*****************************************************

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

"F:\BlackUSB\evc\2022\Fall\CS75\labexamples\Week6Chapter 06\Pr6-12.exe"

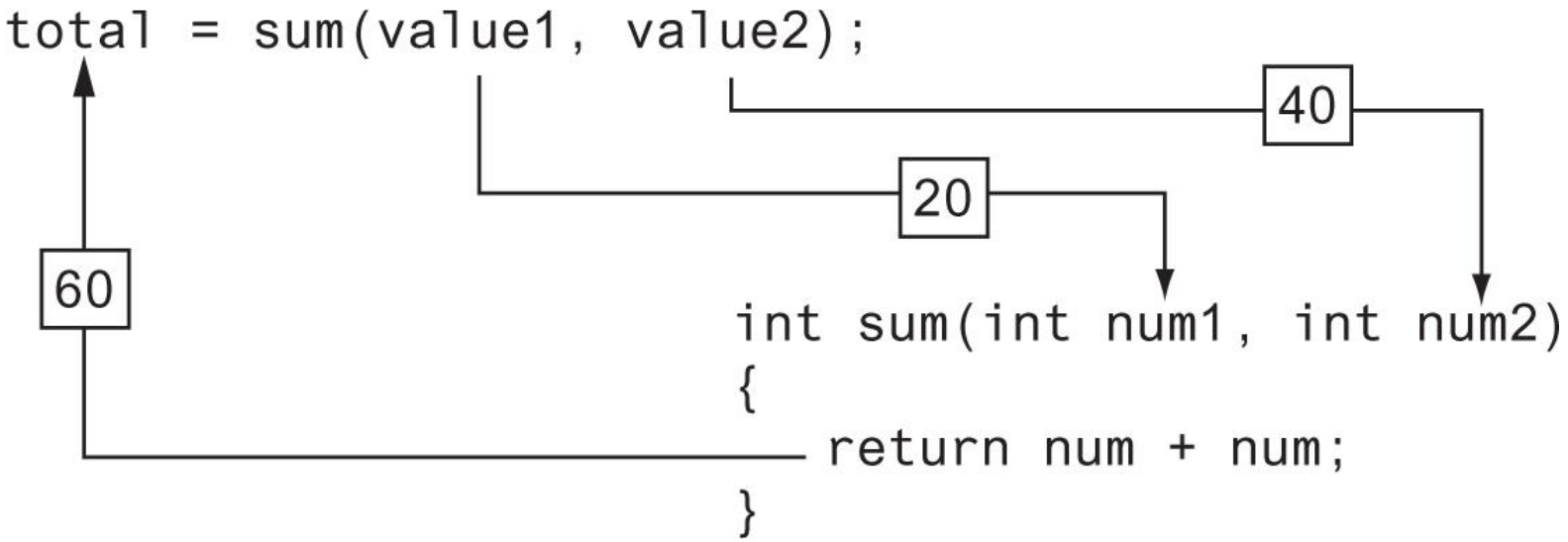
```
The sum of 20 and 40 is 60

Process returned 0 (0x0)   execution time : 0.312 s
Press any key to continue.
```

Logs & others

Code::Blocks | Search results | Cccc | Build log | CppCheck/Vera++ | CppCheck/Vera++ messages | Cscope | Debugger | DoxyBlocks | Fortran info | Closed files list | Thread search

Checking for existence: F:\BlackUSB\evc\2022\Fall\CS75\labexamples\Week6Chapter 06\Pr6-12.exe

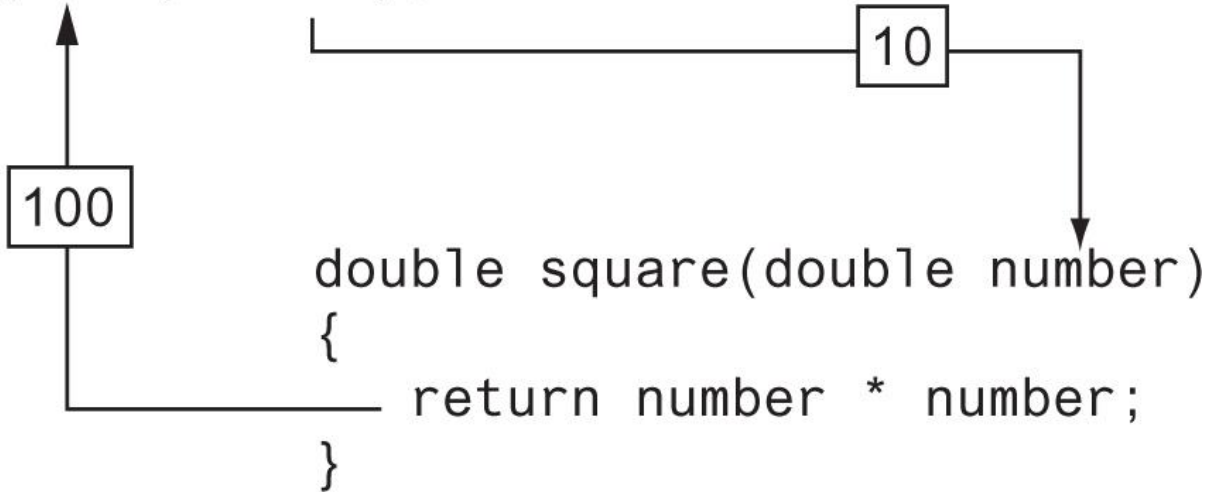
Function Returning a Value in Program 6-12



The statement in line 17 calls the sum function, passing `value1` and `value2` as arguments. The return value is assigned to the `total` variable.

Another Example from Program 6-13

```
area = PI * square(radius);
```



Returning a Value From a Function

- The prototype and the definition must indicate the data type of return value (not `void`)

- Calling function should use return value:
  - assign it to a variable
  - send it to `cout`
  - use it in an expression

Returning a Boolean Value

- Function can return `true` or `false`
- Declare return type in function prototype and heading as `bool`
- Function body must contain `return` statement(s) that return `true` or `false`
- Calling function can use return value in a relational expression

Local and Global Variables

- Variables defined inside a function are *local* to that function. They are hidden from the statements in other functions, which normally cannot access them.
- Because the variables defined in a function are hidden, other functions may have separate, distinct variables with the same name.

Pr6-16.cpp - Code::Blocks 20.03

File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

<global> | anotherFunction() : void

Management | Projects | Workspace

Pr6-12.cpp | Pr6-16.cpp

```cpp
// This program shows that variables defined in a function
// are hidden from other functions.
#include <iostream>
using namespace std;

void anotherFunction(); // Function prototype

int main()
{
   int num = 1;   // Local variable

   cout << "In main, num is " << num << endl;
   anotherFunction();
   cout << "Back in main, num is " << num << endl;
   return 0;
}

//*****************************************************
// Definition of anotherFunction                      *
// It has a local variable, num, whose initial value  *
// is displayed.                                      *
//*****************************************************

void anotherFunction()
{
   int num = 20;  // Local variable

   cout << "In anotherFunction, num is " << num << endl;
}
```

"F:\BlackUSB\evc\2022\Fall\CS75\labexamples\Week6Chapter 06\Pr6-16.exe"

```
In main, num is 1
In anotherFunction, num is 20
Back in main, num is 1

Process returned 0 (0x0)   execution time : 0.342 s
Press any key to continue.
```

Local Variable Lifetime

- A function's local variables exist only while the function is executing. This is known as the *lifetime* of a local variable.
- When the function begins, its local variables and its parameter variables are created in memory, and when the function ends, the local variables and parameter variables are destroyed.
- This means that any value stored in a local variable is lost between calls to the function in which the variable is declared.

Global Variables and Global Constants

- A global variable is any variable defined outside all the functions in a program.
- The scope of a global variable is the portion of the program from the variable definition to the end.
- This means that a global variable can be accessed by *all* functions that are defined after the global variable is defined.

Global Variables and Global Constants

- You should avoid using global variables because they make programs difficult to debug.
- Any global that you create should be *global constants*.

Initializing Local and Global Variables

- Local variables are not automatically initialized. They must be initialized by programmer.

- Global variables (not constants) are automatically initialized to `0` (numeric) or `NULL` (character) when the variable is defined.

Static Local Variables

- Local variables only exist while the function is executing. When the function terminates, the contents of local variables are lost.

- `static` local variables retain their contents between function calls.

- `static` local variables are defined and initialized only the first time the function is executed. `0` is the default initialization value.

r6-21.cpp

```cpp
// This program shows that local variables do not retain
// their values between function calls.
#include <iostream>
using namespace std;

// Function prototype
void showLocal();

int main()
{
   showLocal();
   showLocal();
   return 0;
}

//*********************************************************
// Definition of function showLocal.                      *
// The initial value of localNum, which is 5, is displayed. *
// The value of localNum is then changed to 99 before the  *
// function returns.                                      *
//*********************************************************

void showLocal()
{
   int localNum = 5; // Local variable

   cout << "localNum is " << localNum << endl;
   localNum = 99;
}
```

"F:\BlackUSB\evc\2022\Fall\CS75\labexamples\Week6Chapter 06\Pr6-21.exe"

```
localNum is 5
localNum is 5

Process returned 0 (0x0)   execution time : 0.286 s
Press any key to continue.
```

gs & others

Default Arguments

A <u>Default argument</u> is an argument that is passed automatically to a parameter if the argument is missing on the function call.

- Must be a constant declared in prototype:

```
void evenOrOdd(int = 0);
```

- Can be declared in header if no prototype
- Multi-parameter functions may have default arguments for some or all of them:

```
int getSum(int, int=0, int=0);
```

Default Arguments

- If not all parameters to a function have default values, the defaultless ones are declared first in the parameter list:

```
int getSum(int, int=0, int=0);// OK
int getSum(int, int=0, int);  // NO
```

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

```
sum = getSum(num1, num2);     // OK
sum = getSum(num1, , num3);   // NO
```

Using Reference Variables as Parameters

- A mechanism that allows a function to work with the original argument from the function call, not a copy of the argument
- Allows the function to modify values stored in the calling environment
- Provides a way for the function to ‘return’ more than one value

Passing by Reference

- A <u>reference variable</u> is an alias for another variable
- Defined with an ampersand (`&`)

```
void getDimensions(int&, int&);
```

- Changes to a reference variable are made to the variable it refers to
- Use reference variables to implement passing parameters *by reference*

Reference Variable Notes

- Each reference parameter must contain &
- Space between type and & is unimportant
- Must use & in both prototype and header
- Argument passed to reference parameter must be a variable – cannot be an expression or constant
- Use when appropriate – don’t use when argument should not be changed by function, or if function needs to return only 1 value

Overloading Functions

- Overloaded functions have the same name but different parameter lists
- Can be used to create functions that perform the same task but take different parameter types or different number of parameters
- Compiler will determine which version of function to call by argument and parameter lists

Function Overloading Examples

Using these overloaded functions,

```
void getDimensions(int);                // 1
void getDimensions(int, int);           // 2
void getDimensions(int, double);        // 3
void getDimensions(double, double);// 4
```

the compiler will use them as follows:

```
int length, width;
double base, height;
getDimensions(length);                // 1
getDimensions(length, width);         // 2
getDimensions(length, height);        // 3
getDimensions(height, base);          // 4
```

The `exit()` Function

- Terminates the execution of a program
- Can be called from any function
- Can pass an `int` value to operating system to indicate status of program termination
- Usually used for abnormal termination of program
- Requires `cstdlib` header file

The `exit()` Function

- Example:

```
exit(0);
```

- The `cstdlib` header defines two constants that are commonly passed, to indicate success or failure:

```
exit(EXIT_SUCCESS);
exit(EXIT_FAILURE);
```