Chapter 5

Functions

Examples of Functions

- $y = 2x^3 - 3x^2 - 3x + 2$
- $f(x) = \sin(x) / \cos(x)$
- $g(x) = \sin(x) / x$

Why Do You Need Functions

- A function is a self-contained block of code with a specific purpose.
- Sometimes you have to repeat the same task several times in a program.
  - With functions, you don't need to replicate the same code at various points.
- Decompose a large program into smaller functions makes it easily manageable for development and testing.
  - A typical program will consist of a large number of small functions, rather than a small number of large functions.

Structure of a Function

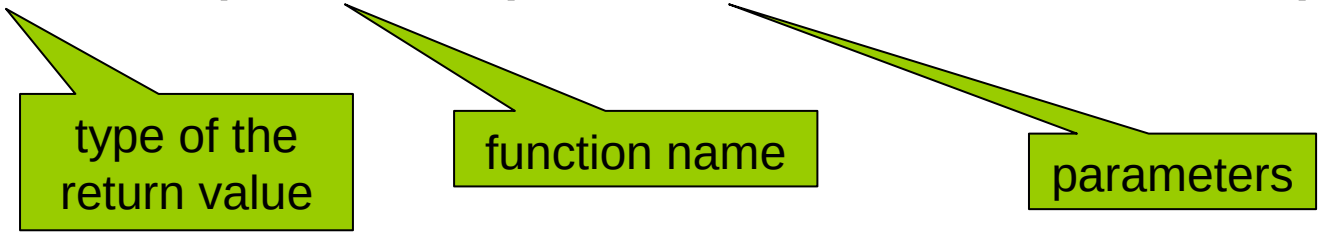
```
// Function to calculate x to the power n
double power(double x, int n)
{
  double result = 1.0;
  for (int i = 1; i<=n; i++)
     result *= x;

  return result;
}
```

Function Header

Function Body

The Function Header

```
double power ( double x, int n )
```



- The name of a function is governed by the same rules as those for a variable.
- A function returns either a single value, or nothing at all (`void`).
  - The single value returned can be a pointer, which contain the address of an array.

Arguments vs. Parameters

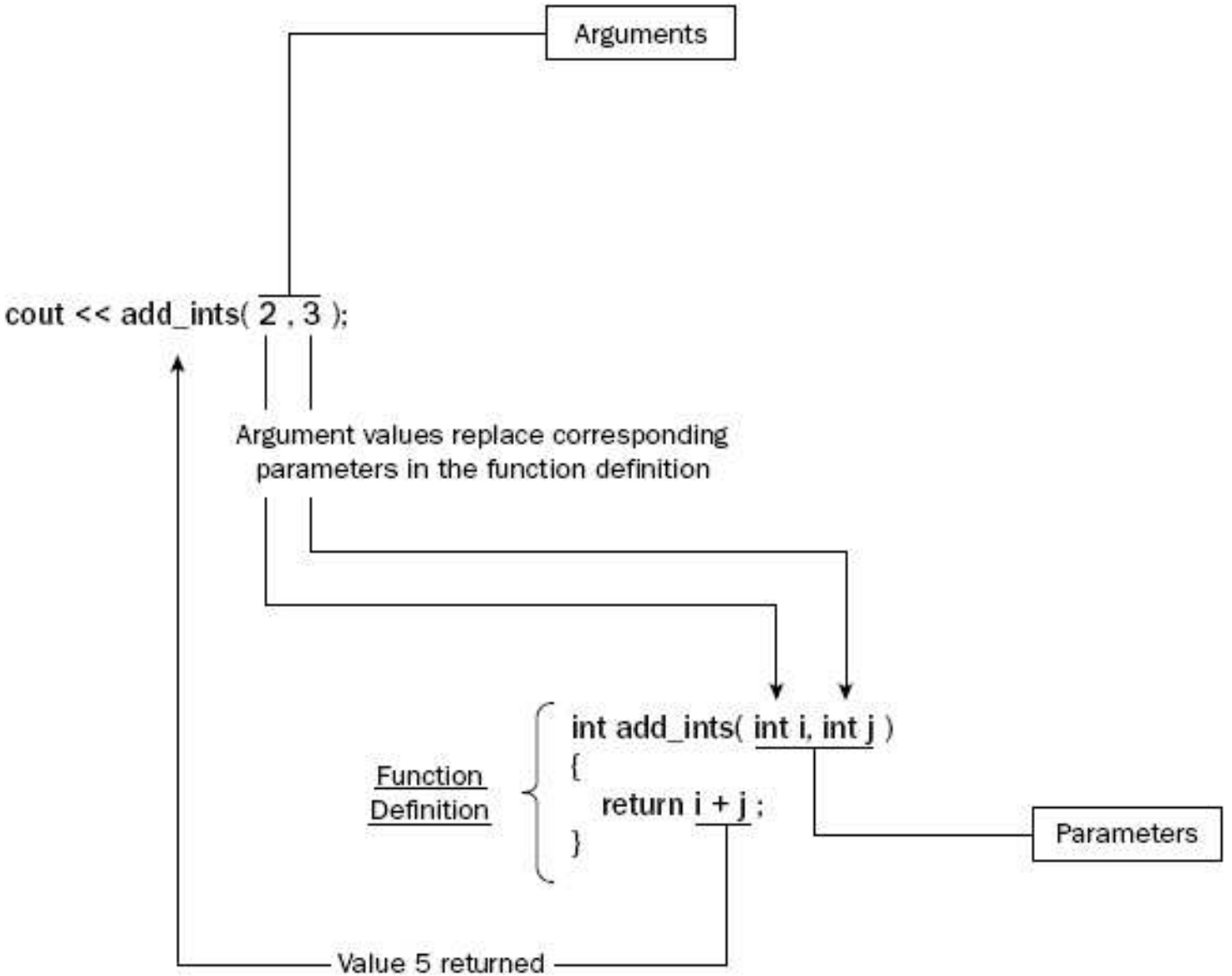


Figure 5-1

The Function Body

```
{
 double result = 1.0;
  for (int i = 1;
  i<=n; i++)
     result *= x;

  return result;
}
```

- The `return` Statement
  - Return a value (evaluated from an expression) to be the functional value.
  - If the type of return value is `void`, there must be no expression.
    - `return;`

Using a Function

- Define the function before it is called.

- However, many programmers prefer to see main() earlier to have a global view.
  - Declare the function using a statement called a function prototype.

```
void print_stars()
{
   cout << "**********"
        << endl;
}

int main()
{
   print_stars();
   cout << "Test" << endl;
   print_stars();
}
```

Function Prototypes

- It contains the same information as appears in the function header, with the addition of a semicolon (;).
  - `double power(double value, int index);`
  - `void print_stars();`
- You can even omit the names of the parameters
  - `double power(double, int);`
  - However, it is better to use meaningful name in a prototype to increase readability.

Using a Function

- Ex5_01.cpp on P.236

- Note the 3 ways to call this function:
  - Passing constants as arguments
  - Outputting the return value of a function
  - Using a function as an argument

Passing Arguments to a Function

- There are two mechanisms in C++ to pass arguments to functions
  - Pass-by-value
  - Pass-by-reference

Pass-by-value

```
int index = 2;
double value = 10.0;
double result = power(value, index);
```

- Copied of arguments are stored in a temporary location in memory.
- This mechanism protect your caller arguments from being accidentally modified by a rogue function.

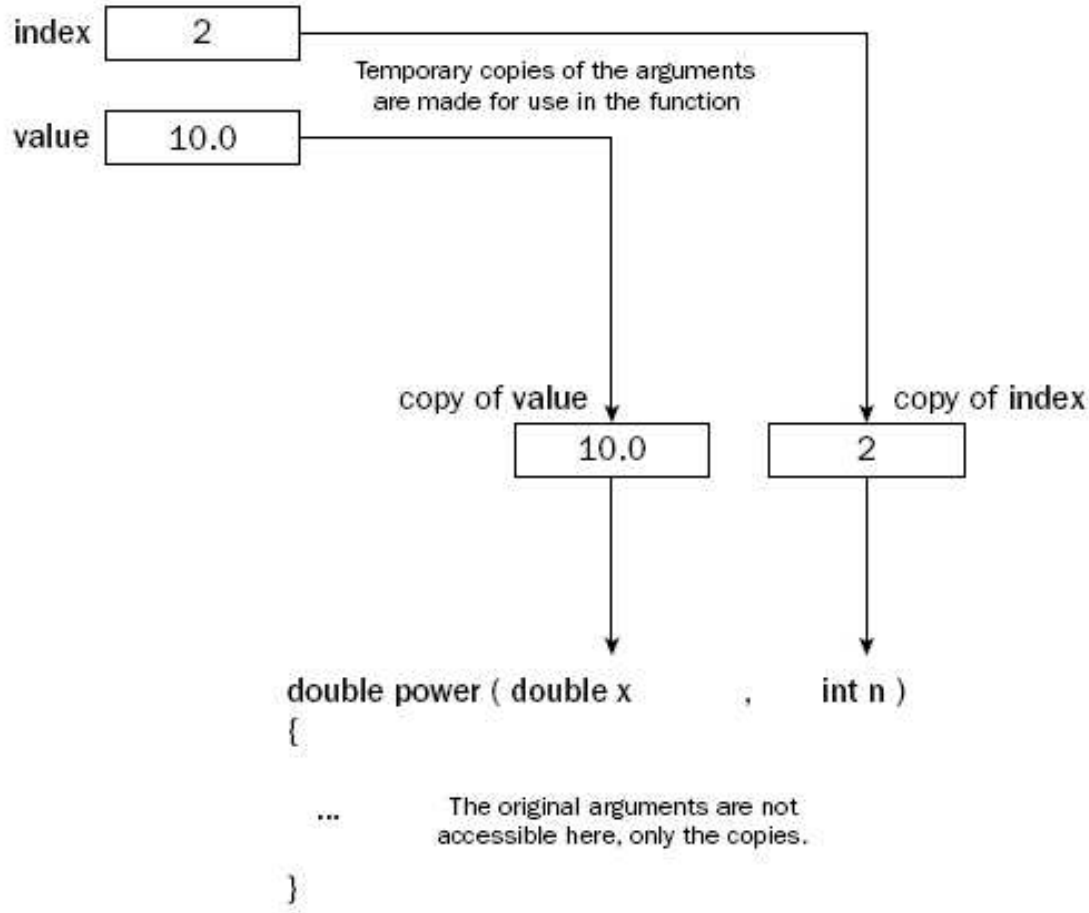


Figure 5-3

Ex5_02.cpp on P.240

```
int main(void)
{  int num = 3;
   cout << endl
        << “incr10(num) = “ << incr10(num)
        << endl << “num = “ << num;
   cout << endl;
   return 0;
}

int incr10(int num)
{ num += 10;
   cout << “In the function, num = “
        << num << endl;
   return num;
}
```

Pointers as Arguments to a Function

- Ex5_03.cpp on P.242
- int * pnum = # // Pointer to num
- int incr10(int *num); // Function Prototype
- return *num;
- // de-reference the pointer to get the return value

Passing Arrays

- The pointer to the beginning of the array is passed by value to the function.
- Ex5_04.cpp on P.243
- `double average(double array[], int count);`
- `average(values, (sizeof values)/(sizeof values[0]));`
  - sizeof values = ?
  - sizeof values[0] =?

What Is a Reference?

- A reference is an alias for another variable (P.197).
  - `long number = 0;`
  - `long& rnumber = number;`
  - `rnumber += 10;`
  - `cout << number;`

- Difference between a pointer and a reference:
  - A pointer needs to be de-referenced
  - A reference is an alias. There is no need for de-referencing.

Pass-by-reference

- Remember that a reference is merely an alias.
- Ex5_07.cpp on P.247
- The output shows that the function `incr10()` is directly modifying the variable passed.

Static Variables in a Function

- With only "automatic" variables within a function, you can't count how many times a function is called.
- Within a function, you can declare a variable as `static` so that its value persists from one call to the next.
  - Initialization of a static variable within a function only occurs the first time that the function is called.
- Ex5_13.cpp on P.261

Recursive Function Calls

- Recursive function - A function calls itself
  - Either directly or indirectly
  - fun1 -> fun2 -> fun1
- Fibonacci sequence:
  - F(n) = F(n-1) + F(n-2)
  - F(0) = 0
  - F(1) = 1
- Factorial
  - N! = N*(N-1)*(N-2)*…*3*2*1
  - F(n) = n * F(n-1)
  - F(0) = 1
- Be sure to specify the boundary condition to stop the recursive call!