6.1 Modular Programming

- Modular programming: breaking a program up into smaller, manageable functions or modules
- Function: a collection of statements to perform a task
- Modular programming simplifies the process of writing programs
- Modular programming improves maintenance of programs
- Modular programming is a collection of modules
- Modular programming: breaking a program up into smaller, manageable functions or modules

6.2 Defining and Calling Functions

- Function call: statement causes a function to execute
- Function definition: statements that make up a function
  - return type: data type of the value that the function returns to the part of the program that called it
  - name: name of the function. Function names follow same rules as variables
  - parameter list: variables containing values passed to the function
  - body: statements containing values returned by the function

Define the function that reads the input from stdin.

```python
def read_input():
    return input()
```

### Function Call Example

```python
result = read_input()
```

Think of the problem in the big picture.

- Divide the problem into sections of functionality
- Divide these sections into divisions of functionality
- Continue to break the problem down into smaller sub-problems until each problem is small

```python
def read_list():
    return [read_input() for _ in range(3)]
```
Defining Functions

- The type of each parameter must be specified.
- The number of parameters.
- The return type.
- The name.

Function names must follow the naming conventions of a programming language.

Functions can call other functions.

Main can call any number of functions.

Calling Functions

- To call a function, use the function name followed by parentheses.
  - printMenu();

- When called, the program executes the body of the called function.
  - After the function terminates, execution resumes in the calling function at the point of call.

Flow of Control in Program 6-1

- 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
  - Type of each parameter

Calling Functions

Defining Functions

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

- Return statement is not required.
  - void printHeading()
    {
      cout << "Monthly Sales\n";
    }
6.3 Function Prototypes

- Before the compiler encounters a call to a particular function's return type, the number of parameters it uses, and the type of each parameter. Two ways:
  - Place the function definition before all calls to the function
  - Function prototype: declare a function before it is defined

Function Prototypes

- Variables in a function are arguments.
- Values passed to function are arguments.
- Values passed as arguments at time of call.
- Send values into a function at time of call.
- When a function is called, the program may send values into the function.

6.4 Sending data into a function

- When a function is called, the program may send values into the function.
- Can pass values into a function at time of call:
  - c = pow(2.0, 4.0);

6.3 Function Prototypes

- 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
- Remember to consider all 3:
  - Function prototype
  - Function definition
  - Function call

Function Prototypes

- void printHeading();
- void printHeading();

Program 6.2

Function Prototypes

- void printHeading(
  int a,
  int b,
  int c
)
  // printHeading is called with 3 arguments

Program 6.3

Function Prototypes

- return type functionName(Parameters);
- Function prototype: declare a function before it is defined
- Place the function prototype before all calls to the function
- Parameter's type and the type of each parameter is useful in defining the type of each parameter's function. Return type is the number of parameters.
- Place the function prototype before a call to a function

Function Prototypes

- void printHeading(
  int a,
  int b,
  int c
)
  // printHeading is called with 3 arguments

Program 6.5
A Function with a Parameter

```cpp
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.

**Example**

```
Example
```

### Notes
- Parameters need to match the variable names of the arguments,
  - Same Order
  - Same Type
  - Same Number
- It is very important that parameters and arguments match.

#### Passing Data by Value
- When an argument is passed into a parameter, only a copy of the argument's value is passed.
- Changes to the parameter do not affect the original argument.

#### Example
```
Example
```

### Program 6-5
```
// Program 6-5

#include <iostream>
using namespace std;

int main()
{
    int num = 5;
    displayValue(num);
    cout << num << endl;

    return 0;
}
```

### Program 6-6
```
// Program 6-6

#include <iostream>
using namespace std;

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

int main()
{
    int num = 5;
    displayValue(num);
    return 0;
}
```
Example: Program 6-9

Example:

```cpp
int number = 12;
changeMe(number);
```

changeMe can change variable `myValue`, but it will have no effect on variable `number` argument in calling function.

6.8 Returning a value from a function

Example:

```cpp
double x = pow(4.0, 0.5);
```

For example, `pow` function returns a double. A function can send a value back to the function that called the function.

Example: Program 6-10 in the book

The return statement causes a function to end immediately. If there is no return statement for the void function, the function ends until the last `}`.

Example: calling function

```cpp
int number = 12;
changeMe(number);
```

Parameter `number` can change variable `myValue`?

```
Example: if the return value = 12!
```

See Program 6-10 in the book

Development time of functions and speed of program

Higher-level functions can call general:

- Implement general-purpose tasks:
  - Implement user choices from menu

Functions can be used 6.6 Menu-Driven Program

```
Example: Program 6-9
```

Functions can be used to implement user choices from menu.

6.7 return

- The return statement causes a function to end immediately.
- It can be placed anywhere in the function.
- If there is no return statement for the void function, the function ends until the last `}`.

Example:

```cpp
double x;
x = pow(4.0, 2.0);
```
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:
```c
int sum(int num1, int num2)
{
    int result;
    result = num1 + num2;
    return result;
}
```

Program Output
```c
{ |
    return num1 + num2;
    |
} |
```

Expressions, such as `num1 + num2`, can return the value of functions.

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

int total = sum(value1, value2);
```

The statement in line 17 calls the sum function, passing `value1` and `value2` as arguments.

The return value is assigned to the `total` variable.

Example:
```c
int result;
result = num1 + num2;
```

Value Being Returned:
```c
{ |
    return result;
    |
} |
```

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

Example:
```c
int result;
result = num1 + num2;
```
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
- If you return multiple values, you need to encapsulate them or use reference (will discuss it later).

### Notes:
- Using return you can return one value.
- If you return multiple values, you need to encapsulate them or use reference (will discuss it later).

### Example: Program 6-13
```cpp
double square(double number) { return number * number; }
```
Returning a Boolean Value

Notes:
- When the return statement is reached, the function closes down and returns. No following statements are executed.

```cpp
bool isHeavy(int weight)
{
    if (weight >= 200)
        return true;
    return false;
}
```

6.10 Local and Global Variables

Local Variables

- A function’s local variables exist only while the function is executing. This is known as the lifetime of a local variable.
- When a function is executed, the variables defined within it are visible, but when another function is called, only the variables defined in that function are visible.
- A function’s local variables and its parameter variables are created in memory when the function begins and destroyed when the function ends.
- Any value stored in a local variable is lost between calls to the function in which the variable is declared.

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 a function is executed, its local variables and its parameter variables are created in memory when the function begins and destroyed when the function ends.
- Any value stored in a local variable is lost between calls to the function in which the variable is declared.

Same Name

- Because the variables defined in a function are hidden, other functions may have separate, distinct variables with the same name.
- Local variables defined inside a function are visible, so the name num variable defined in main is visible when another function is called.
- Local variables are defined inside a function and they are not accessible outside the function.
- When the program is executing in main, the num variable defined in main is visible. When anotherFunction is called, however, only the variables defined in anotherFunction are visible.
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.
- You should avoid using global variables because they make programs difficult to debug.
  - Global constants:
    - Any global that you create should be defined function.
    - Global constants can be changed by any function.
    - You should avoid using global variables.

Style Guidelines

- 13. Do NOT use global variables. You may define named constants that are used by more than one function globally.

Example

```
// Example

// Function to initialize local and global variables

void initialize_variables() {
    // Local variables
    int local_int = 0;
    double local_double = 0.0;

    // Global variables
    int global_int = 0;
    double global_double = 0.0;
}
```

Program 6-17

Example

```
// Example

// Function to initialize variables

void initialize_variables() {
    // Local variables
    int local_int = 0;
    double local_double = 0.0;

    // Global variables
    int global_int = 0;
    double global_double = 0.0;
}
```

Example

```
// Example

// Function to initialize variables

void initialize_variables() {
    // Local variables
    int local_int = 0;
    double local_double = 0.0;

    // Global variables
    int global_int = 0;
    double global_double = 0.0;
}
```
6.1 Static Local Variables

Some Notes

• You can have 2 variables with the same name as long as they have different scope.

• Local and Global Variables can have the same name. The name of the local shadows the name of the global.

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

int num;

int main()
{
    cout<<num<<endl;
    int num=1;
    cout<<num<<endl;
}
```

Output:

0
1

6.1.1 Static Local Variables

In this program, each time `showLocal()` is called, the `localNum` variable is re-created and initialized with the value 5.

```cpp
static int statNum = 0;
```

`statNum` is automatically initialized to 0. Notice that it retains its value between function calls.

The local variables are lost when the function terminates.
If you do initialize a local static variable, the initialization only happens once. See Program 6-23.

### 6.12 Default Arguments

A **default argument** 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:
  ```
  bool evenOrOdd(int = 0);
  ```

- Can be declared in header if no prototype is provided:
  ```
  bool evenOrOdd(int num = 0)
  ```

- Multi-parameter functions may have default arguments for some or all of them:
  ```
  int getSum(int, int=0, int=0);
  ```

- Example:
  ```
  Default arguments specified in the prototype (Program Continues)
  ```

- **Notes**
  
  - If not all parameters to a function have default values, the defaultless ones are declared first in the parameter list:
    ```
    int getSum(int, int, int=0);// OK
    int getSum(int, int, 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
    ```

- **in the prototype:**
  ```
  int Volum(int=1, int=1, int=1);
  ```

- **in the definition:**
  ```
  int Volum(int height, int width, int depth)
  ```

- **in the call:**
  ```
  result = Volum(3,2,1);
  result = Volum(3,2);
  result = Volum(3);
  result = Volum();
  ```
6.13 Using Reference Variables as Parameters

• What if we need more than one value returned from a function?
• We can send the address of a variable, then the function can put the value we need at this address.
• Passing by reference: 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
• Works when appropriate — don’t use when it’s not
• Use when the address of the variable you return is meaningful to the user

Passing by reference

• A reference variable is an alias for another variable
• Defined with an ampersand (&)
• A reference variable is passed by reference to implement passing parameters by reference
• Use reference variables to implement parameter passing
• Changes to a reference variable are made to the variable it refers to
• Provides a way for the function to modify the result of the function call
• Works when appropriate — don’t use when it’s not
• Use when the address of the variable you return is meaningful to the user
• Use pass-by-value parameters for functions whenever possible. Pass by reference parameters are to be used only when you need to get more than one value out of a function or when you need to change the value of a variable.

Example

void getDimensions(int&, int&);

Example

The & here in the prototype indicates that the parameter is a reference variable.

Here we are passing value by reference.

Example

The & also appears here in the function header.

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.

Style Guidelines

• 14. Use pass-by-value parameters for functions whenever possible. Pass by reference parameters are to be used only when you need to get more than one value out of a function or when you need to change the value of a variable.

Note

Passing by reference

• A reference variable is an alias for another variable
• Defined with an ampersand (&)
• Changes to a reference variable are made to the variable it refers to
• Use reference variables to implement passing parameters by reference
• Provides a way for the function to modify the result of the function call
• Works when appropriate — don’t use when it’s not
• Use when the address of the variable you return is meaningful to the user

Program 62

Example

The & here in the prototype indicates that the parameter is a reference variable.

Here we are passing value by reference.

Example

The & also appears here in the function header.

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.

Style Guidelines

• 14. Use pass-by-value parameters for functions whenever possible. Pass by reference parameters are to be used only when you need to get more than one value out of a function or when you need to change the value of a variable.

Note

Passing by reference

• A reference variable is an alias for another variable
• Defined with an ampersand (&)
• Changes to a reference variable are made to the variable it refers to
• Use reference variables to implement passing parameters by reference
• Provides a way for the function to modify the result of the function call
• Works when appropriate — don’t use when it’s not
• Use when the address of the variable you return is meaningful to the user

Program 62

Example

The & here in the prototype indicates that the parameter is a reference variable.

Here we are passing value by reference.

Example

The & also appears here in the function header.

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.

Style Guidelines

• 14. Use pass-by-value parameters for functions whenever possible. Pass by reference parameters are to be used only when you need to get more than one value out of a function or when you need to change the value of a variable.

Note

Passing by reference

• A reference variable is an alias for another variable
• Defined with an ampersand (&)
• Changes to a reference variable are made to the variable it refers to
• Use reference variables to implement passing parameters by reference
• Provides a way for the function to modify the result of the function call
• Works when appropriate — don’t use when it’s not
• Use when the address of the variable you return is meaningful to the user

Program 62

Example

The & here in the prototype indicates that the parameter is a reference variable.

Here we are passing value by reference.

Example

The & also appears here in the function header.

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.

Style Guidelines

• 14. Use pass-by-value parameters for functions whenever possible. Pass by reference parameters are to be used only when you need to get more than one value out of a function or when you need to change the value of a variable.

Note

Passing by reference

• A reference variable is an alias for another variable
• Defined with an ampersand (&)
• Changes to a reference variable are made to the variable it refers to
• Use reference variables to implement passing parameters by reference
• Provides a way for the function to modify the result of the function call
• Works when appropriate — don’t use when it’s not
• Use when the address of the variable you return is meaningful to the user

Program 62

Example

The & here in the prototype indicates that the parameter is a reference variable.

Here we are passing value by reference.

Example

The & also appears here in the function header.

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.

Style Guidelines

• 14. Use pass-by-value parameters for functions whenever possible. Pass by reference parameters are to be used only when you need to get more than one value out of a function or when you need to change the value of a variable.

Note

Passing by reference

• A reference variable is an alias for another variable
• Defined with an ampersand (&)
• Changes to a reference variable are made to the variable it refers to
• Use reference variables to implement passing parameters by reference
• Provides a way for the function to modify the result of the function call
• Works when appropriate — don’t use when it’s not
• Use when the address of the variable you return is meaningful to the user

Program 62

Example

The & here in the prototype indicates that the parameter is a reference variable.

Here we are passing value by reference.

Example

The & also appears here in the function header.

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.

Style Guidelines

• 14. Use pass-by-value parameters for functions whenever possible. Pass by reference parameters are to be used only when you need to get more than one value out of a function or when you need to change the value of a variable.

Note

Passing by reference

• A reference variable is an alias for another variable
• Defined with an ampersand (&)
• Changes to a reference variable are made to the variable it refers to
• Use reference variables to implement passing parameters by reference
• Provides a way for the function to modify the result of the function call
• Works when appropriate — don’t use when it’s not
• Use when the address of the variable you return is meaningful to the user

Program 62

Example

The & here in the prototype indicates that the parameter is a reference variable.

Here we are passing value by reference.

Example

The & also appears here in the function header.

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.

Style Guidelines

• 14. Use pass-by-value parameters for functions whenever possible. Pass by reference parameters are to be used only when you need to get more than one value out of a function or when you need to change the value of a variable.

Note

Passing by reference

• A reference variable is an alias for another variable
• Defined with an ampersand (&)
• Changes to a reference variable are made to the variable it refers to
• Use reference variables to implement passing parameters by reference
• Provides a way for the function to modify the result of the function call
• Works when appropriate — don’t use when it’s not
• Use when the address of the variable you return is meaningful to the user

Program 62

Example

The & here in the prototype indicates that the parameter is a reference variable.

Here we are passing value by reference.

Example

The & also appears here in the function header.

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.

Style Guidelines

• 14. Use pass-by-value parameters for functions whenever possible. Pass by reference parameters are to be used only when you need to get more than one value out of a function or when you need to change the value of a variable.

Note

Passing by reference

• A reference variable is an alias for another variable
• Defined with an ampersand (&)
• Changes to a reference variable are made to the variable it refers to
• Use reference variables to implement passing parameters by reference
• Provides a way for the function to modify the result of the function call
• Works when appropriate — don’t use when it’s not
• Use when the address of the variable you return is meaningful to the user

Program 62

Example

The & here in the prototype indicates that the parameter is a reference variable.

Here we are passing value by reference.

Example

The & also appears here in the function header.

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.

Style Guidelines

• 14. Use pass-by-value parameters for functions whenever possible. Pass by reference parameters are to be used only when you need to get more than one value out of a function or when you need to change the value of a variable.

Note

Passing by reference

• A reference variable is an alias for another variable
• Defined with an ampersand (&)
• Changes to a reference variable are made to the variable it refers to
• Use reference variables to implement passing parameters by reference
• Provides a way for the function to modify the result of the function call
• Works when appropriate — don’t use when it’s not
• Use when the address of the variable you return is meaningful to the user

Program 62

Example

The & here in the prototype indicates that the parameter is a reference variable.

Here we are passing value by reference.

Example

The & also appears here in the function header.

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.

Style Guidelines

• 14. Use pass-by-value parameters for functions whenever possible. Pass by reference parameters are to be used only when you need to get more than one value out of a function or when you need to change the value of a variable.

Note

Passing by reference

• A reference variable is an alias for another variable
• Defined with an ampersand (&)
• Changes to a reference variable are made to the variable it refers to
• Use reference variables to implement passing parameters by reference
• Provides a way for the function to modify the result of the function call
• Works when appropriate — don’t use when it’s not
• Use when the address of the variable you return is meaningful to the user

Program 62

Example

The & here in the prototype indicates that the parameter is a reference variable.

Here we are passing value by reference.

Example

The & also appears here in the function header.
6.14 Overloading Functions

- Functions that have the same name but are different by parameter list are called overloaded functions.
- Can be used to create functions that perform the same task but take different parameter types or different number of parameters.
- The standard header defines two constants that indicate success or failure:

  ```
  #include <cstdlib>
  #define EXIT_SUCCESS 0
  #define EXIT_FAILURE -1
  ```

Example:

```c
double divide(double,double);
```

6.15 exit() function

- Terminates the execution of a program.
- Usually used for abnormal termination of a program.
- Can be called from any function.
- Requires `cstdlib` header.
- Example:

  ```c
  exit(0);
  ```

The overloaded functions have different parameter lists.

Example:

- Passing an int
  ```c
  int x = 5;
  ```

- Passing a double
  ```c
  double y = 3.14;
  ```

Example

- evt
  ```c
  struct event {
      int id;
      int data;
  }
  ```

- thing
  ```c
  struct thing {
      int id;
      int data;
  }
  ```

- Overloading Functions

  ```c
  double divide(double,double);
  ```

  ```c
  double divide(double);
  ```

The overloads differ by parameter type.

Example:

- Different number of parameters
  ```c
  double divide(double,double);
  ```

- Same number but different parameter types
  ```c
  double divide(double, double);
  ```

- Example:

  ```c
  double divide(double,double);
  ```

6.14 Overloading Functions
6.16 Stubs and Drivers

- These are development tools. It is a good idea to use them.
- Useful for testing and debugging program logic and design

**Stub**

A stub is a skeleton function for use before the details of the actual function have been written.

```cpp
int calc(double, double)
{
    cout<<"You are in the calc function."
    return 2;
}
```

**Driver**

A driver is used to call a completed function to see if it works. You can use these to test a function before you have main completely written.

```cpp
result = formulate(16, 22.87, 'b');
```