Propedéutico de Programación

Coordinación de Ciencias Computacionales

1/13

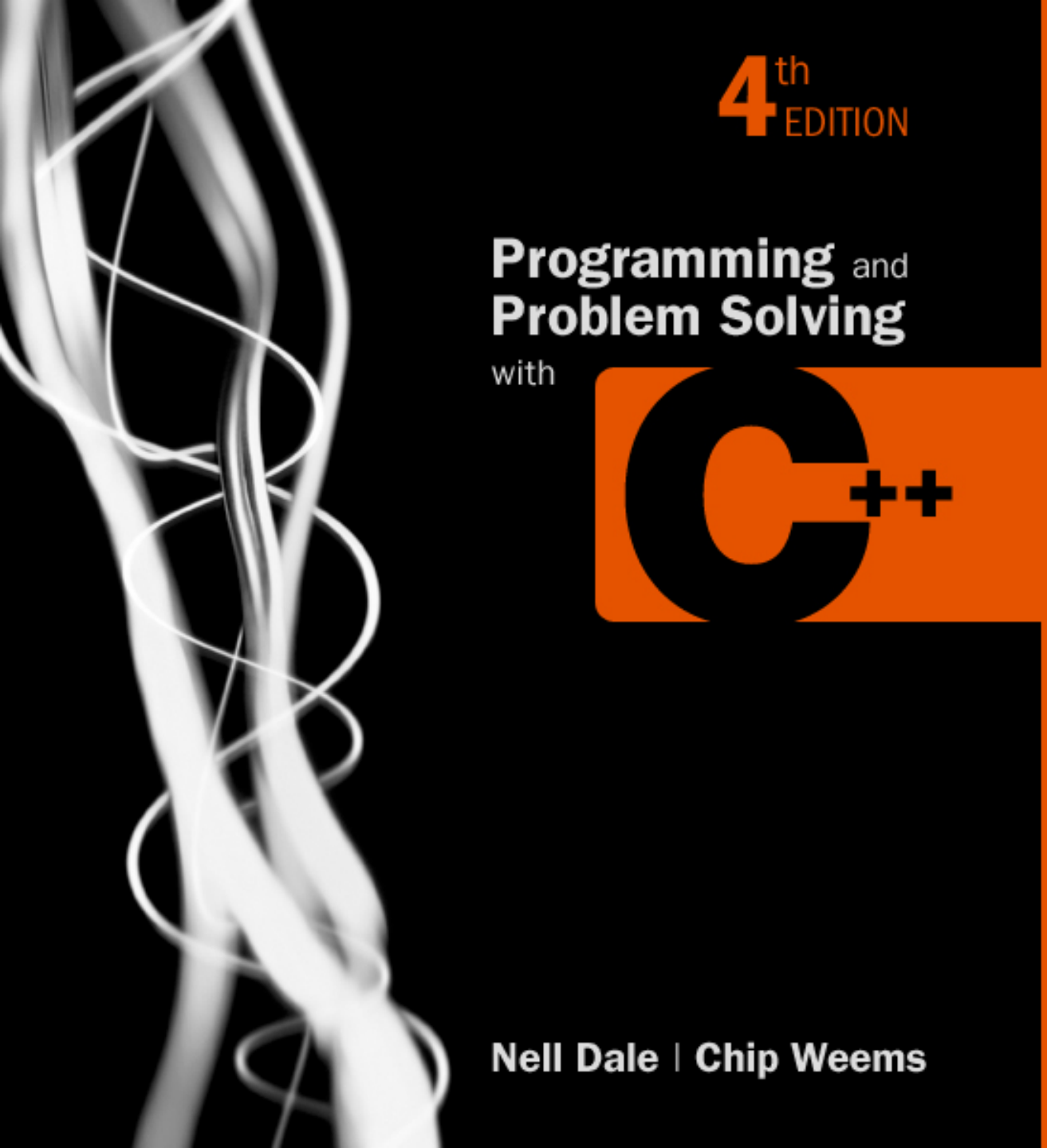
Material preparado por:
Dra. Pilar Gómez Gil

Objetivos del curso

- Revisar conceptos de programación y estructuras de datos con un enfoque orientado a objetos, a través de su implementación en C++.
- Se espera que al final del curso, el/la estudiante muestren un dominio profundo de estos temas.

Temario General

- Revisión de fundamentos de programación
- Desarrollo orientado a objetos
- Apuntadores
- Diseño de datos e implementación
- La estructura abstracta lista
- Pilas y Colas
- Árboles
- Otras estructuras



Dale/Weems

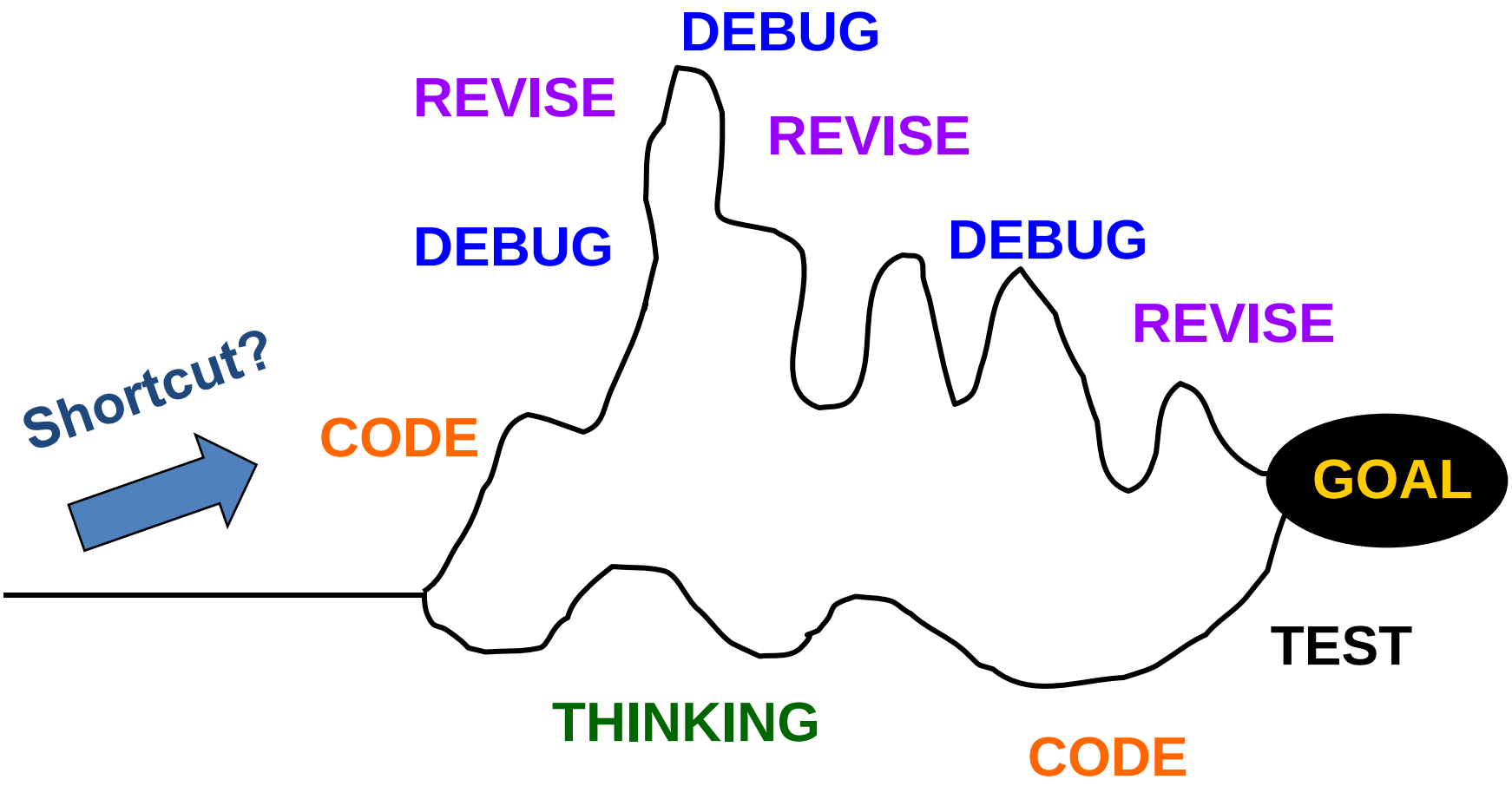
Slides based on work by Sylvia Sorkin, Community College of Baltimore County - Essex Campus

Problem-Solving Phase

- **Analyze the problem and specify what the solution must do**
- **Develop a general solution(algorithm) to solve the problem**
- **Verify that your solution really solves the problem**

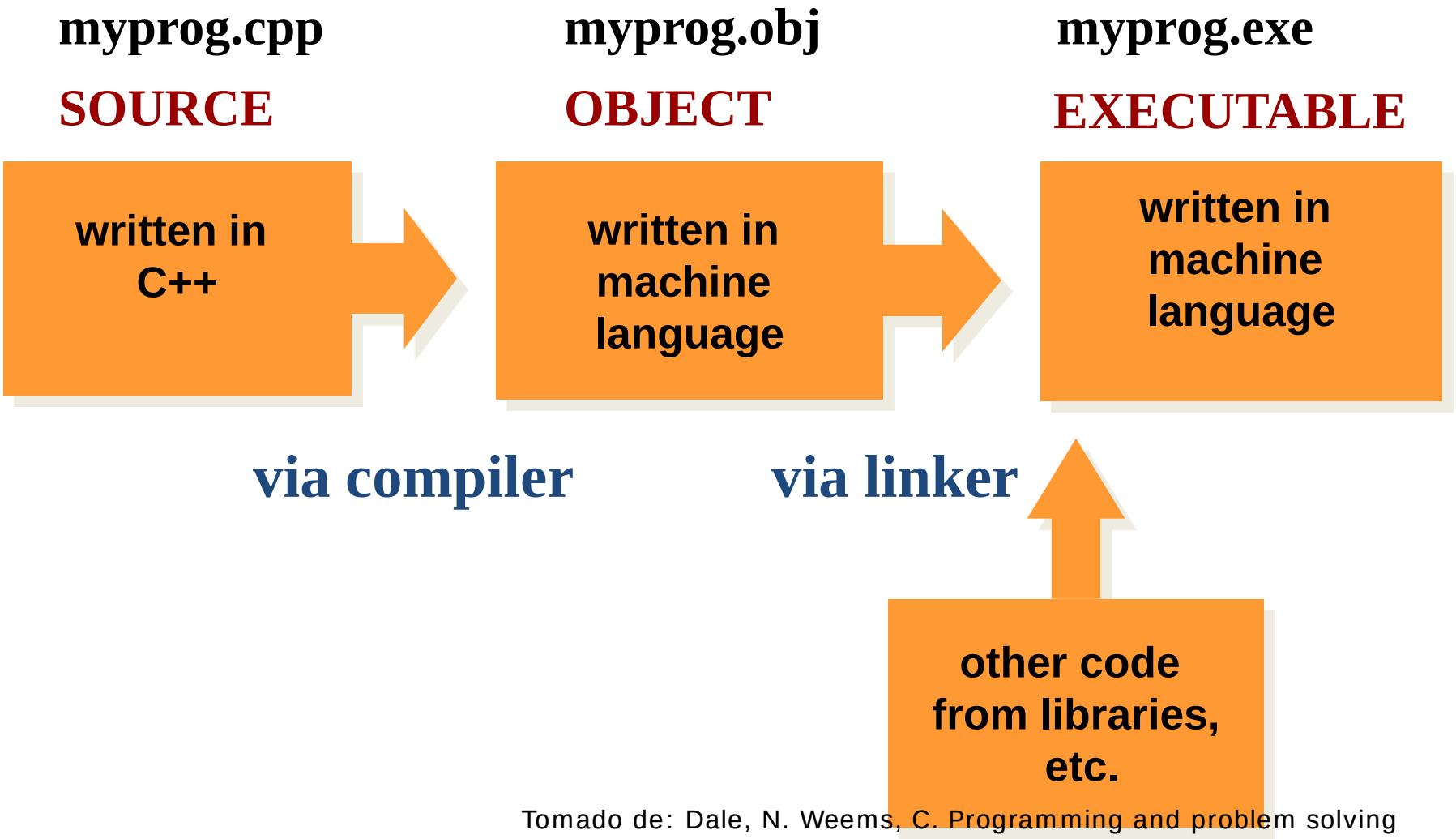
Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

A Tempting Shortcut?



Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Three C++ Program Stages



Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Computing Profession Ethics

- **Copy software only with permission from the copyright holder**
- **Give credit to another programmer by name whenever using his/her code**
- **Use computer resources only with permission**
- **Guard the privacy of confidential data**
- **Use software engineering principles to develop software free from errors**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

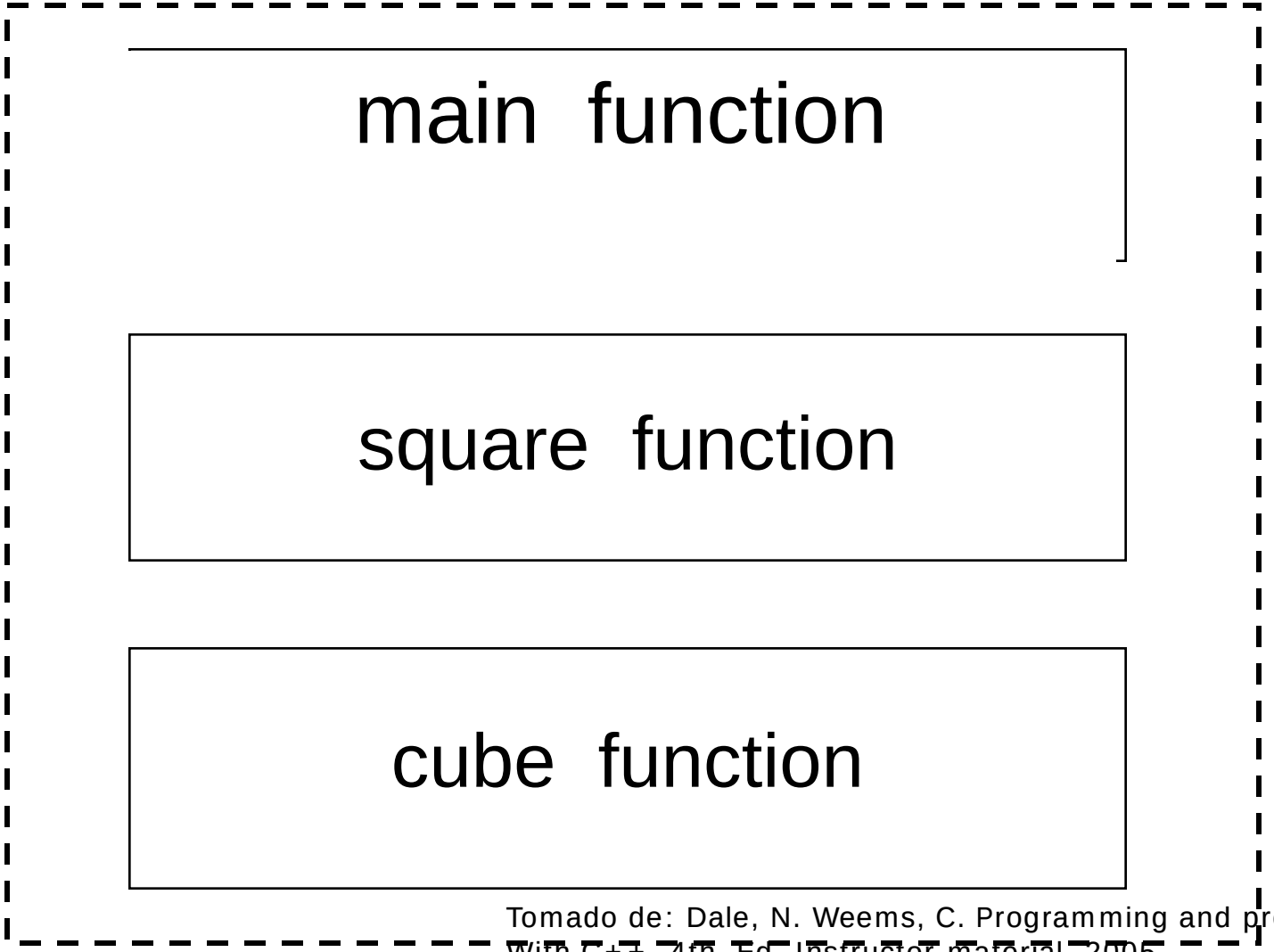
A C++ program is a collection of one or more functions

- **There must be a function called main()**
- **Execution always begins with the first statement in function main()**

- **Any other functions in your program are subprograms and are not executed until they are called**

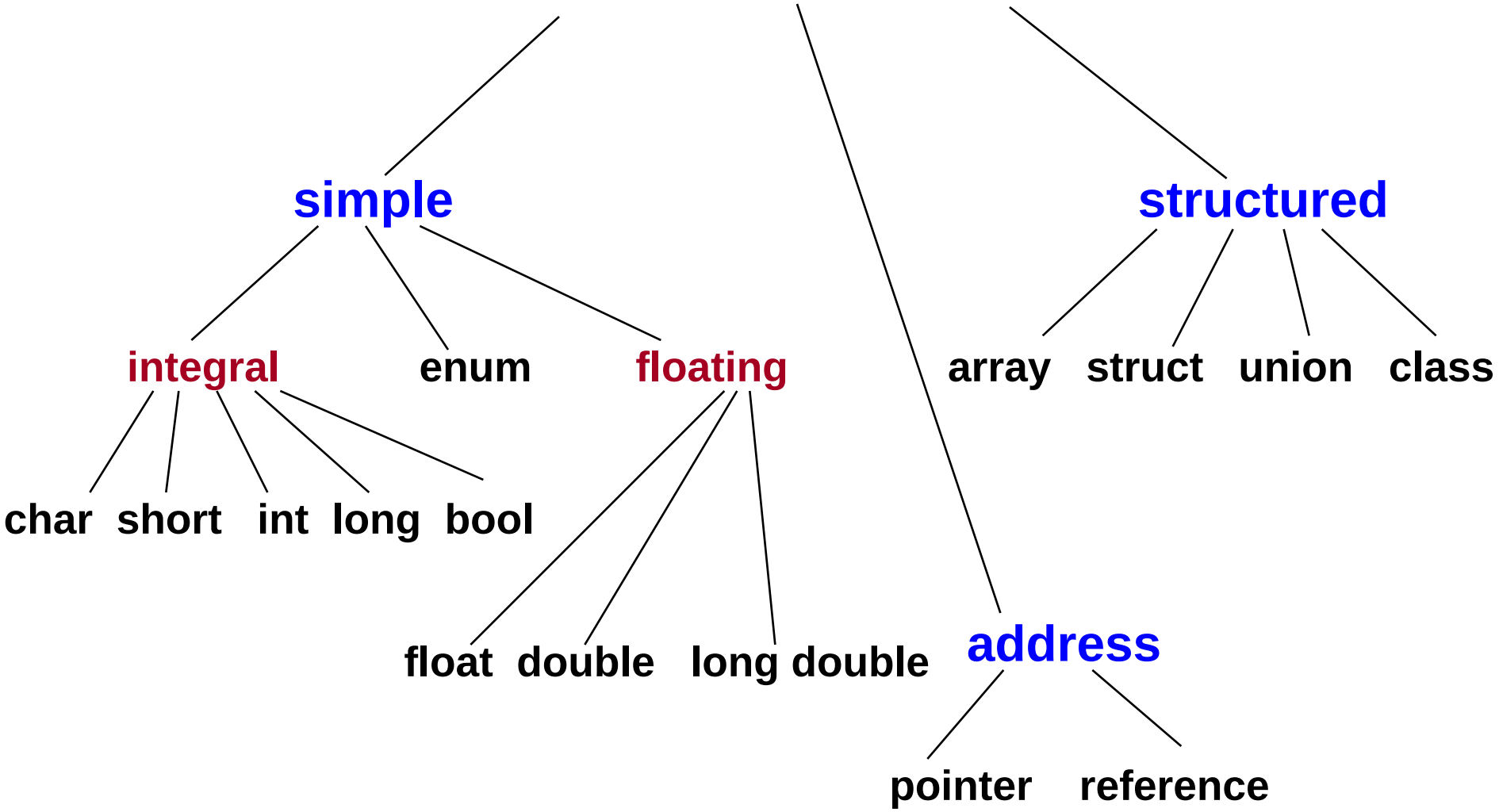
Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Program With Several Functions



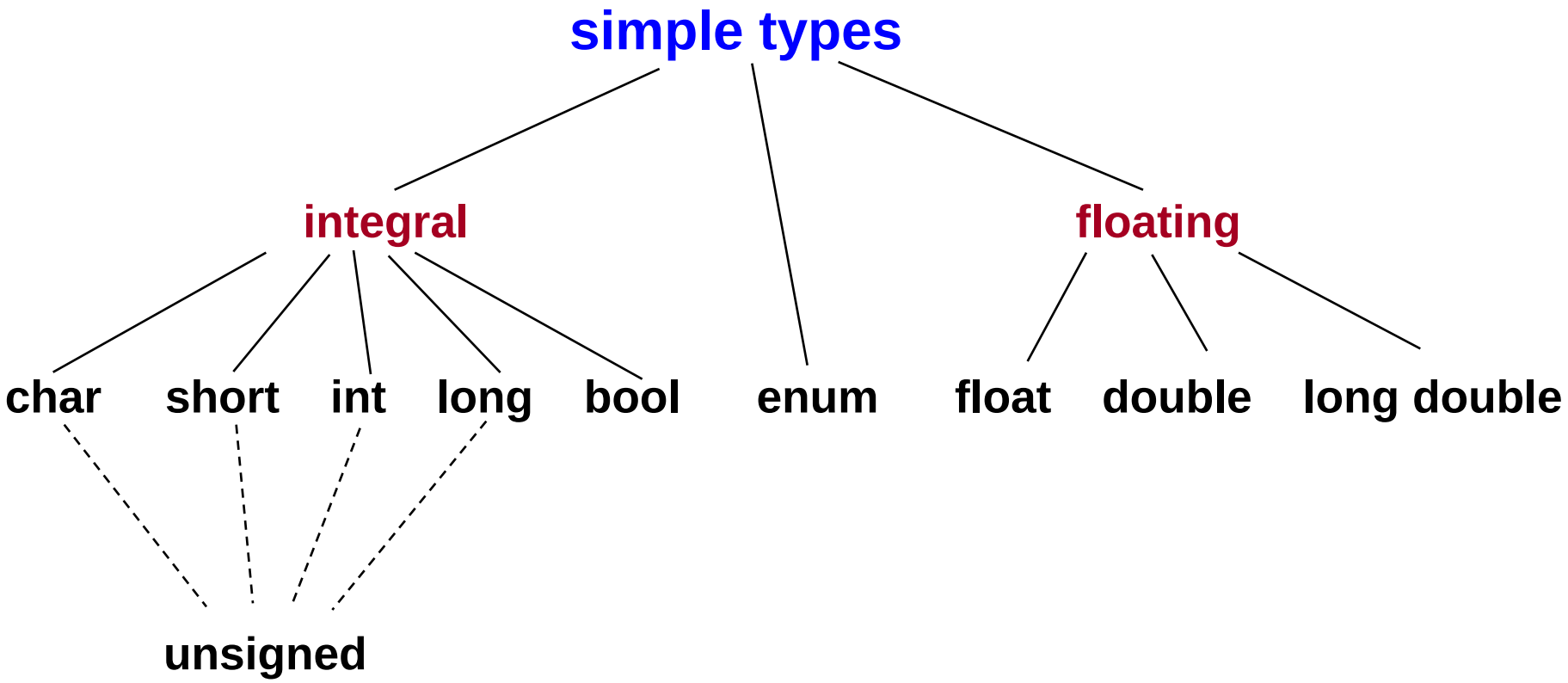
Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

C++ Data Types



Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

C++ Simple Data Types



Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Standard Data Types in C++

- **Integral Types**
  - **represent whole numbers and their negatives**
  - **declared as int, short, or long**
- **Floating Types**
  - **represent real numbers with a decimal point**
  - **declared as float, or double**
- **Character Types**
  - **represent single characters**
  - **declared as char**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Insertion Operator(<<)

- **Variable cout is predefined to denote an output stream that goes to the standard output device(display screen)**
- **The insertion operator << called "put to" takes 2 operands**
- **The left operand is a stream expression, such as cout**
- **The right operand is an expression of a simple type or a string constant**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Output Statements

SYNTAX

```
cout << Expression << Expression . . .;
```

These examples yield the same output:

```
cout << "The answer is ";
cout << 3 * 4;
```

```
cout << "The answer is " << 3 * 4;
```

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Is compilation the first step?

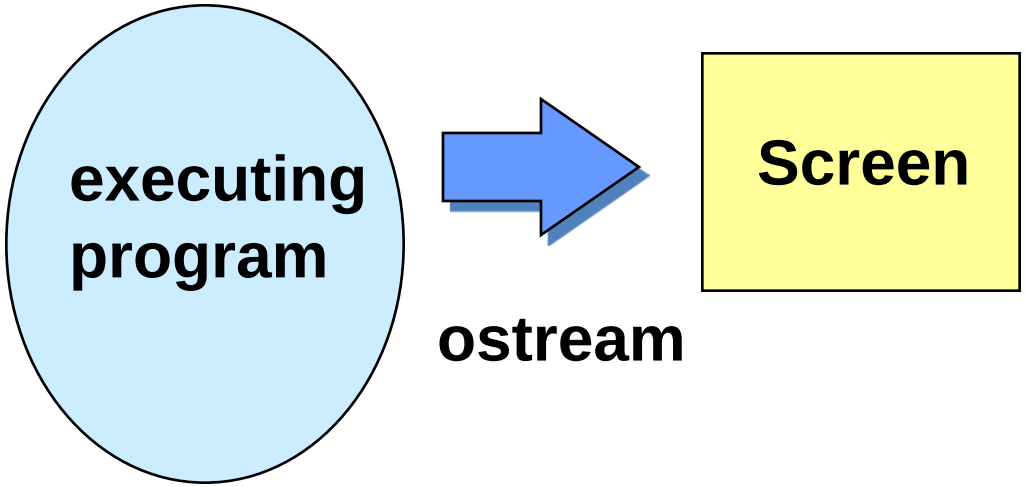
- **No; before your source program is compiled, it is first examined by the preprocessor that**
  - **removes all comments from source code**
  - **handles all preprocessor directives--they begin with the # character such as**

    #include <iostream>
  - **This include tells the preprocessor to look in the standard include directory for the header file called iostream and insert its contents into your source code**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

No I/O is built into C++

- Instead, a library provides an output stream



Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Using Libraries

- **A library has 2 parts**

  Interface (stored in a header file) tells what items are in the library and how to use them

  Implementation (stored in another file) contains the definitions of the items in the library

- **#include <iostream>**

  Refers to the header file for the *iostream* library needed for use of cout and endl.

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Functions

- **Every C++ program must have a function called `main`**
- **Program execution always begins with function main**
- **Any other functions are subprograms and must be called**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

More About Functions

- **It is not considered good practice for the body block of function main to be long**
- **Function calls are used to do subtasks**
- **Every C++ function has a return type**
- **If the return type is not void, the function returns a value to the calling block**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Where are functions?

Functions are subprograms

- **located in libraries, or**
- **written by programmers for their use in a particular program**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

HEADER FILE	FUNCTION	EXAMPLE OF CALL	VALUE
<cstdlib>	abs(i)	abs(-6)	6
<cmath>	pow(x,y)	pow(2.0,3.0)	8.0
	fabs(x)	fabs(-6.4)	6.4
<cmath>	sqrt(x)	sqrt(100.0)	10.0
	sqrt(x)	sqrt(2.0)	1.41421
<cmath>	log(x)	log(2.0)	.693147
<iomanip>	setprecision(n)	setprecision(3)	

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Function Call

- **A function call temporarily transfers control to the called function's code**
- **When the function's code has finished executing, control is transferred back to the calling block**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Function Call Syntax

FunctionName =(Argument List)

The argument list is a way for functions to communicate with each other by passing information

The argument list can contain zero, one, or more arguments, separated by commas, depending on the function

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

A void function call stands alone

```
#include <iostream>

void  DisplayMessage(int  n);
// Declares function

int main()
{
    DisplayMessage(15);
    // Function call
    cout  <<  "Good Bye"  <<  endl;
    return 0;
}
```

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Two Kinds of Functions

Value-Returning	Void
Always returns a single value to its caller and is called from within an expression	Never returns a value to its caller and is called as a separate statement

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

<iostream> is header file

- **For a library that defines 3 objects**

  An istream object named cin (keyboard)

  An ostream object named cout (screen)

  An ostream object named cerr (screen)

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Extraction Operator(>>)

- **Variable cin is predefined to denote an input stream from the standard input device(the keyboard)**
- **The extraction operator >> called "get from" takes 2 operands; the left operand is a stream expression, such as cin--the right operand is a variable of simple type**
- **Operator >> attempts to extract the next item from the input stream and to store its value in the right operand variable**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Input Statements

SYNTAX

```
cin >> Variable >> Variable . . .;
```

These examples yield the same result.

```
cin >> length;
cin >> width;
```

```
cin >> length >> width;
```

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Another Way to Read char Data

The `get()` function can be used to read a single character.

`get()` obtains the very next character from the input stream without skipping any leading whitespace characters

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

getline() Function

- **Because the extraction operator stops reading at the first trailing whitespace, >> cannot be used to input a string with blanks in it**
- **Use the `getline` function with 2 arguments to overcome this obstacle**
- **First argument is an input stream variable, and second argument is a string variable**

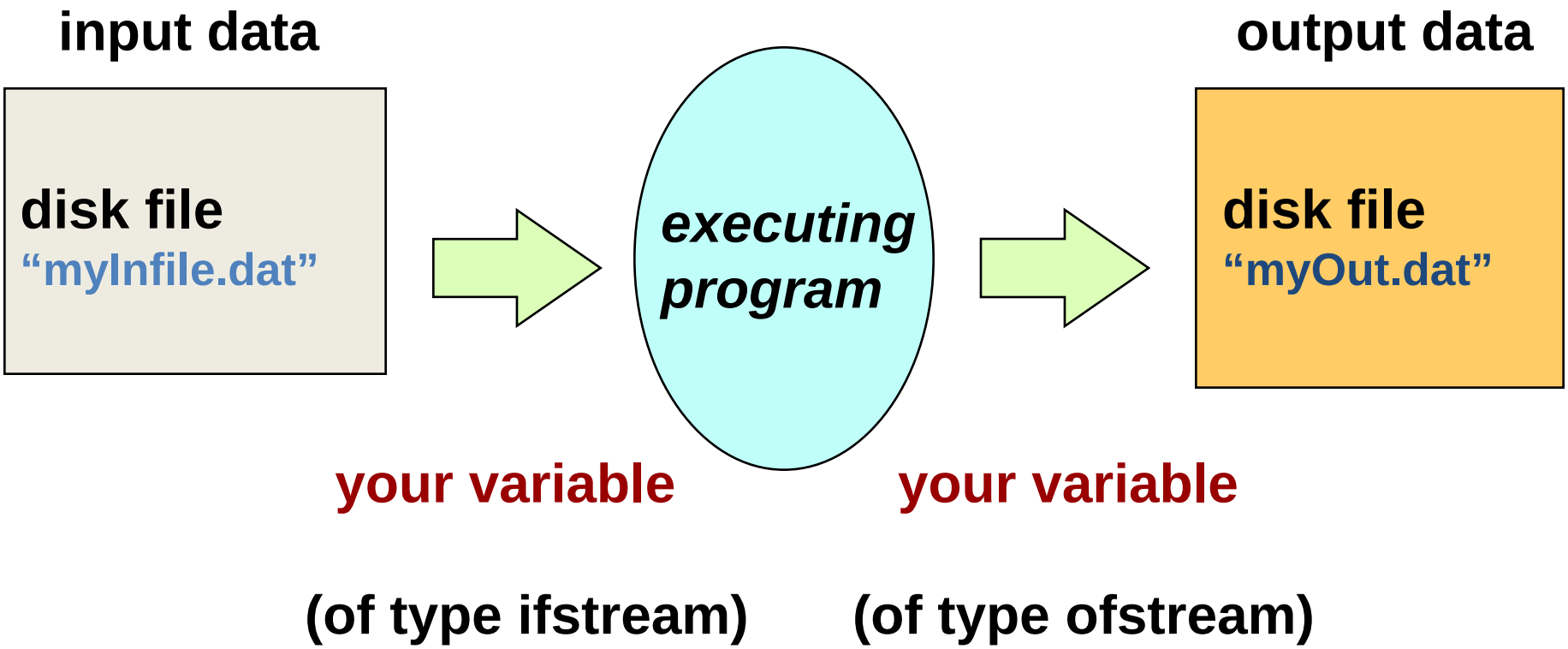
Example

```
string    message;
getline(cin,  message);
```

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Disk Files for I/O

#include <fstream>



Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Disk I/O

To use disk I/O

- **Access #include <fstream>**
- **Choose valid identifiers for your filestreams and declare them**
- **Open the files and associate them with disk names**
- **Use your filestream identifiers in your I/O statements(using >> and << , manipulators, get, ignore)**
- **Close the files**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Disk I/O Statements

```
#include <fstream>

ifstream   myInfile;          // Declarations
ofstream  myOutfile;

myInfile.open("myIn.dat"); // Open files
myOutfile.open("myOut.dat");

myInfile.close();             // Close files
myOutfile.close();
```

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Opening a File

Opening a file

- **Associates the C++ identifier for your file with the physical (disk) name for the file**
  - **If the input file does not exist on disk, open is not successful**
  - **If the output file does not exist on disk, a new file with that name is created**
  - **If the output file already exists, it is erased**
- **Places a file reading marker at the very beginning of the file, pointing to the first character in the file**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Stream Fail State

- **When a stream enters the fail state,**
  - **Further I/O operations using that stream have no effect at all**
  - **The computer does not automatically halt the program or give any error message**
- **Possible reasons for entering fail state include**
  - **Invalid input data (often the wrong type)**
  - **Opening an input file that doesn't exist**
  - **Opening an output file on a disk that is already full or is write-protected**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Run Time File Name Entry

```
#include <string>
// Contains conversion function c_str

ifstream  inFile;
string    fileName;

cout << "Enter input file name: " << endl; // Prompt
cin    >>  fileName;

// Convert string fileName to a C string type
inFile.open(fileName.c_str());
```

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Functional Decomposition

A technique for developing a program in which the problem is divided into more easily handled subproblems, the solutions of which create a solution to the overall problem

In functional decomposition, we work from the abstract (a list of the major steps in our solution) to the particular (algorithmic steps that can be translated directly into code in C++ or another language)

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Functional Decomposition

Focus **is on actions and algorithms**

Begins **by breaking the solution into a series of major steps; process continues until each subproblem cannot be divided further or has an obvious solution**

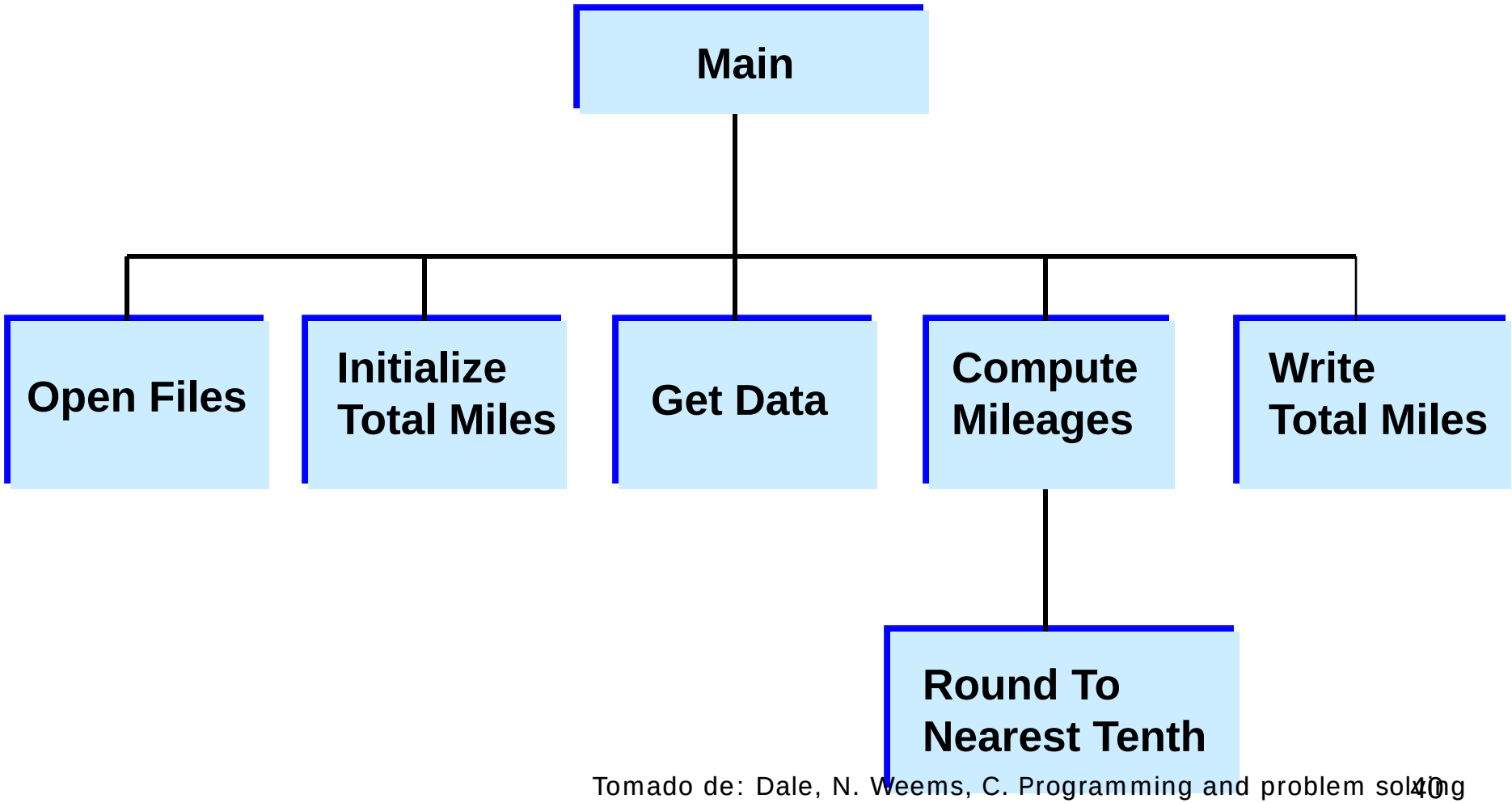
Units **are *modules* representing algorithms**

- **A module is a collection of concrete and abstract steps that solves a subproblem**
- **A module structure chart (hierarchical solution tree) is often created**

Data **plays a secondary role in support of actions to be performed**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Module Structure Chart



Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Chapter 5

Conditions, Logical Expressions, and Selection Control Structures

Dale/Weems

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

`bool` Data Type

- Type **`bool`** is a built-in type consisting of just 2 values, the constants **true** and **false**
- We can declare variables of type **bool**

```
bool hasFever; // true if has high temperature
bool isSenior; // true if age is at least 55
```

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

C++ Control Structures

- **Selection**
  - **if**
  - **if . . . else**
  - **switch**
- **Repetition**
  - **for loop**
  - **while loop**
  - **do . . . while loop**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Logical Expression	Meaning	Description
! p	NOT p	! p is false if p is true ! p is true if p is false
p && q	p AND q	p && q is true if both p and q are true. It is false otherwise.
p || q	p OR q	p || q is true if either p or q or both are true. It is false otherwise.

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

If-Then-Else Syntax

```
if (Expression)
        StatementA
else
        StatementB
```

NOTE: StatementA and StatementB each can be a single statement, a null statement, or a block

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Nested If Statements

```
if (Expression1 )
        Statement1
else if (Expression2 )
        Statement2
                .
                .
                .
else if (ExpressionN )
        StatementN
else
        Statement N+1
```

Exactly 1 of these statements will be executed

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Testing Selection Control Structures

- **To test a program with branches, use enough data sets to ensure that every branch is executed at least once**

- **This strategy is called minimum complete coverage**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Testing Often Combines Two Approaches

WHITE BOX TESTING	BLACK BOX TESTING
Code Coverage	**Data Coverage**
Allows us to see the program code while designing the tests, so that data values at the boundaries, and possibly middle values, can be tested.	Tries to test as many allowable data values as possible without regard to program code.

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Testing

- **Design and implement a test plan**
- **A test plan is a document that specifies the test cases to try, the reason for each, and the expected output**
- **Implement the test plan by verifying that the program outputs the predicted results**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

PHASE	RESULT	TESTING TECHNIQUE
Problem solving	Algorithm	Algorithm walk-through
Implementation	Coded program	Code walk-through, Trace
Compilation	Object program	Compiler messages
Execution	Output	Implement test plan

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Chapter 6

Looping

Dale/Weems

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

While Statement

SYNTAX

```
while (Expression)
{       .
      .                    // loop body
       .
}
```

Loop body can be a single statement, a null statement, or a block

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Complexity

- **Complexity is a measure of the amount of work involved in executing an algorithm relative to the size of the problem**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Chapter 9

Additional Control Structures

Dale/Weems

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Switch Statement

The Switch statement is a selection control structure for multi-way branching

```
switch  (IntegralExpression)
  {
          case  Constant1 :
                  Statement(s);      // optional
          case  Constant2 :
                  Statement(s);      // optional
             .
             .
             .
          default :                  // optional
                  Statement(s);      // optional
  }
```

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Do-While Statement

Do-While is a looping control structure in which the loop condition is tested *after* each iteration of the loop

SYNTAX

```
do
{
        Statement

} while (Expression);
```

Loop body statement can be a single statement or a block

For Loop

SYNTAX

```
for (initialization;  test expression; update)
{
        Zero or more statements to repeat

}
```

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Break Statement

- **The Break statement can be used with Switch or any of the 3 looping structures**
- **It causes an immediate exit from the Switch, While, Do-While, or For statement in which it appears**
- **If the Break statement is inside nested structures, control exits only the innermost structure containing it**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Continue Statement

- **The Continue statement is valid only within loops**
- **It terminates the current loop iteration, but not the entire loop**
- **In a For or While, Continue causes the rest of the body of the statement to be skipped; in a For statement, the update is done**
- **In a Do-While, the exit condition is tested, and if true, the next loop iteration is begun**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Chapter 7

Functions

Dale/Weems

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Function Call Syntax

FunctionName(Argument List)

The argument list is a way for functions to communicate with each other by passing information

The argument list can contain 0, 1, or more arguments, separated by commas, depending on the function

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Prototypes

A prototype looks like a heading but must end with a semicolon, and its parameter list needs only to contain the type of each parameter

```
int  Cube(int );  // Prototype
```

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Function Calls

When a function is called, temporary memory is allocated for its value parameters, any local variables, and for the function's name if the return type is not void

Flow of control then passes to the first statement in the function's body

The called function's statements are executed until a

return statement (with or without a return value) or the closing brace of the function body is encountered

Then control goes back to where the function was called

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Header Files

Header Files contain

- **Named constants like**

  const int INT_MAX = 32767;

- **Function prototypes like**

  float sqrt(float);

- **Classes like**

  string, ostream, istream

- **Objects like**

  cin, cout

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Classified by Location

Arguments (actual parameters)	Parameters (formal parameters)
Always appear in a function call within the calling block	Always appear in the function heading, or function prototype

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Argument in Calling Block

Value Parameter	Reference Parameter
The value of the argument (25) is passed to the function when it is called	The memory address (4000) of the argument is passed to the function when it is called
In this case, the argument can be a variable identifier, constant, or expression	In this case, the argument must be a variable identifier

Default Parameters

- **Simple types, structs, and classes are value parameters by default**
- **Arrays are always reference parameters**
- **Other reference parameters are marked as such by having an ampersand(&) beside their type**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

An Assertion

An assertion is a truth-valued statement--one that is either true or false (not necessarily in C++ code)

Examples

studentCount > 0

sum is assigned && count > 0

response == 'y' or 'n'

0.0 <= deptSales <= 25000.0

beta == beta @ entry * 2

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Preconditions and Postconditions

- **A precondition is an assertion describing everything that the function requires to be true at the moment the function is invoked**
- **A postcondition describes the state at the moment the function finishes executing, providing the precondition is true**
- **The *caller* is responsible for ensuring the precondition, and the *function code* must ensure the postcondition**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Chapter 8

Scope, Lifetime, and More on Functions

Dale/Weems

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Scope of Identifier

The scope of an identifier (or named constant) is the region of program code in which it is legal to use that identifier for any purpose

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Local Scope vs. Global Scope

- **The scope of an identifier that is declared *inside* a block (this includes function parameters) extends from the point of declaration to the end of the block**

- **The scope of an identifier that is declared *outside* of all namespaces, functions, and classes extends from point of declaration to the end of the entire file containing the program code**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Detailed Scope Rules

1. **Function names have global scope**
2. **A function parameter's scope is identical to the scope of a local variable declared in the outermost block of the function body**
3. **A global variable's (or constant's) scope extends from its declaration to the end of the file, except as noted in rule 5**
4. **A local variable's (or constant's) scope extends from its declaration to the end of the block in which it is declared, including any nested blocks, except as noted in rule 5**
5. **An identifier's scope does not include any nested block that contains a locally declared identifier with the same name (local identifiers have name precedence)**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Namespace Scope

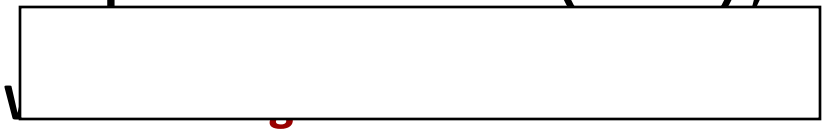
- **The scope of an identifier declared in a namespace definition extends from the point of declaration to the end of the namespace body, and its scope includes the scope of a using directive specifying that namespace**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

3 Ways to Use Namespace Identifiers

- Use a qualified name consisting of the namespace, the scope resolution operator :: and the desired the identifier

```
alpha = std::abs(beta);
```



- W

```
using std::abs;
```

- Write a using directive locally or globally

```
using namespace std;
```

Automatic vs. Static Variable

- **Storage for automatic variable is allocated at block entry and deallocated at block exit**

- **Storage for static variable remains allocated throughout execution of the entire program**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005

Default Allocation

- **Local variables are automatic**

- **To obtain a static local variable, you must use the reserved word `static` in its declaration**

Tomado de: Dale, N. Weems, C. Programming and problem solving With C++. 4th. Ed. Instructor material, 2005