# Introduction to Computers, the Internet, and the Web

Deitel 8th Edition, Chapter 1

# OBJECTIVES

Basic computer concepts

Types of programming languages

The C programming language

The C Standard Library

C programming development environment

Software development method

# INTRODUCTION

C

- Structured programming

C++

- Object-oriented programming

Software

- Instructions to perform actions and make decisions

Hardware

- Executes the actions

# BASIC COMPUTER CONCEPTS

# BASIC COMPUTER CONCEPTS – A REVIEW

Hardware and software

Computer organization

Data hierarchy

# HARDWARE AND SOFTWARE

Computer programs (software)

- Ordered actions

Programmers

- Create software programs

Computer consists of devices

Moore's Law

- Capacity of computers approximately doubles every two years since the 60's
- Gordon Moore of Intel
- Especially true of memory, disk space, and processor speeds

# COMPUTER ORGANIZATION

Computers divided into logical units

- "Logical" means by function, not necessarily separate physical units

**Input unit**

- Receives input from external to the computer
- Devices such as keyboard, mouse, microphone, etc.

**Output unit**

- Sends output from computer to the outside
- Devices such as screen, printer, speakers, etc.

# COMPUTER ORGANIZATION, PART 2

## Memory unit

- Temporarily holds information created or received while a computer is executing
- Volatile – lost when power is turned off or program stops executing
- Called primary memory or RAM (Random Access Memory)

## Arithmetic and logic unit (ALU)

- Performs calculations and decisions
- Part of the CPU

# COMPUTER ORGANIZATION, PART 3

## Central processing unit (CPU)

- Coordinates and supervises operations of other units
- Multi-core processor – multiple processors on a single chip

## Secondary storage unit

- Hard drive (hdd) – long-term storage
- Persistent – not lost when power is lost

# DATA HIERARCHY – BIT

## Bit

- Binary digit
- Smallest item of data
- Zero or one (0 or 1)

Three actions on a bit:

- Examine its value (read it)
- Set its value (initialize it)
- Reverse its value (change it)

# Data Hierarchy – Bytes and Characters

## Byte

- Grouping of 8 bits

## Characters

- Groupings of 1, 2, or 4 bytes
- Decimal digits, letters, special symbols

## Character sets

- The standard used to interpret a series of bytes
- Unicode
- ASCII – subset of Unicode

# Data Hierarchy – Fields and Files

## Fields

- Composed of characters
- Group of characters that have meaning, e.g. name, age, etc.

## Records

- Composed of fields
- Group of related fields
- Example: all the pieces of info about a person

## Files

- Composed of records
- Data files might have a line in a file for each record
- Still just a sequence of bits

# Data Hierarchy – Databases

## Database

- Collection of data organized for easy access
- Relational – data is stored in tables

## Big data

- Massive amounts of data
- 2.5 quintillion bytes of data are created daily
- 90% of data created in last 2 years

# TYPES OF PROGRAMMING LANGUAGES

# There are a Variety of Types of Programming Languages

Machine languages

Assembly languages

High-level languages w/compilers

Interpreted languages

# MACHINE LANGUAGES

Understandable directly by a computer

Defined by its hardware design

Machine dependent

Strings of numbers ultimately reduced to 0's and 1's

# ASSEMBLY LANGUAGES

Machine language is difficult to program in

Uses English-like abbreviations to make it easier for programmers

Still must be translated to machine language

# High-Level Languages & Compilers

Look almost like English

Single statements can accomplish several basic tasks

Compilers convert to machine languages

# Interpreted Languages

Can execute high-level language directly, without having to compile first

Slower than compiled programs

Must execute within some other container, like a browser

# THE C PROGRAMMING LANGUAGE

# C Programming Language

Developed at Bell Labs by Dennis Ritchie*

Used to create the UNIX operating system

Most OS's today are written in C and/or C++

Mostly hardware independent

*Dennis Ritchie died around the same time as Steve Jobs, but his death was not as widely publicized. But without Ritchie, we wouldn't have C. Without C, we wouldn't have Unix. Without Unix, we wouldn't have web servers. And without web servers…

# BUILT FOR PERFORMANCE

Operating systems

- UNIX, Linux, parts of Windows & Android
- Mac OS X built in Objective-C which is derived from C

Embedded systems

- Majority of microprocessors today are used in devices other than computers
- Navigation systems, home security systems, phones, tablets, robots, etc.
- Need to run fast without using much memory

Real-time systems

- “Mission-critical” systems where response time is critical
- Air-traffic control, etc.

Communication systems

- Need to route massive amounts of data quickly

# STANDARDIZATION

Rapid expansion led to many versions for different hardware platforms

Eventually settled on a standard, called Standard C or ANSI C

C11 Standard

- Major version of the standard, from 2011
- Many new and improved features

C18 Standard

- Current version of the standard, from 2018
- Mostly technical improvements not visible to programmers

# C STANDARD LIBRARY

# C Standard Library – Built Into the Language

C programs consist of sections of code called **functions**

Can create your own, or use the ones in the **Standard Library**

Two steps to learning C

- Learning the language itself
- Learning how to use the functions that already exist in the Standard Library

# C Standard Library Provides Building Blocks

Building block approach to writing programs

Software reuse

- Using existing pieces of code to assemble a program

In a program, you'll typically use some combination of:

- C Standard library functions
- Functions you write yourself
- Functions written by others that aren't in the Standard Library

# C++ and Other C-Based Languages

# C++

C++ developed by Bjarne Stroustrup at Bell Labs

Has its roots in C

Object-oriented programming

- Uses reusable software components called **objects** that model items in the real world

# Other C-Based Languages

Objective-C

- Used for OS X/macOS and iOS devices

Java

- Often used for large-scale enterprise systems
- Web servers
- Apps for consumer devices (phones, TVs, etc.)
- Android

C#

- Microsoft
- Developed for integrating Internet and web into applications
- Non-Microsoft versions also

# Other C-Based Languages, part 2

PHP

- Object-oriented scripting language used by web sites
- Supports databases, including MySQL

Python

- Object-oriented scripting language

JavaScript

- Most widely used scripting language
- Used to add dynamic behavior to web sites

Swift

- For macOS and iOS apps
- Will replace Objective-C eventually

# Typical C Program Development Environment – How Programs are Processed

# HUMANS & COMPUTERS DON'T SPEAK THE SAME LANGUAGE

Humans understand a variety of languages

Computers understand two things: 0 and 1

So how do we get from words on a screen to a running program?

# Six Steps of Development/Execution

Editing

Preprocessing

Compilation

Linking

Loading

Execution

# STEP 1 – EDITING

Programmer creates source code file using a text editor

Save it with appropriate file extension for the language

- C source code uses **.c**

Input to this step: your brain

Output from this step: C source code file

# Step 1 – Editing Step

## STEP 2 – PREPROCESSING

Part of compilation, actually

Preprocessor program reads commands called **preprocessor directives** to manipulate program in various ways before actual compilation begins

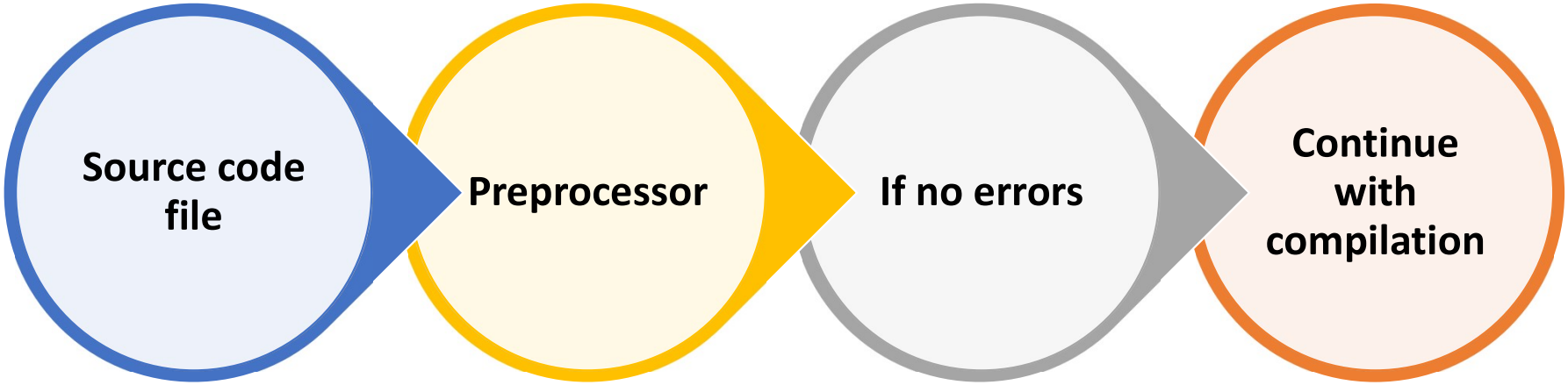
- Including other files, replacing text with other text, etc.

Input to this step: C source code file

Output from this step: intermediate step within compilation

Errors in this step: files/libraries that can't be found

# Step 2 – Preprocessing Step



## STEP 3 – COMPILATION

Compiler reads source code file

Attempts to translate it into machine language

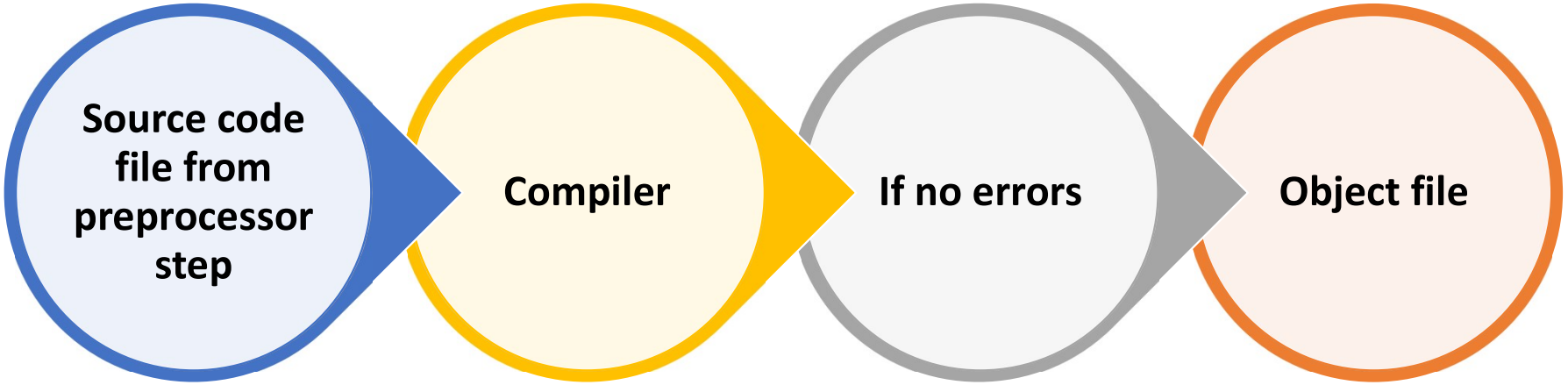
If successful, creates a binary **object file** with appropriate file extension for the language (C compilers create .o files)

Input to this step: C source code file

Output from this step: machine language in a .o file

Errors in this step: syntax errors (a.k.a. compile errors)

# Step 3 – Compilation Step



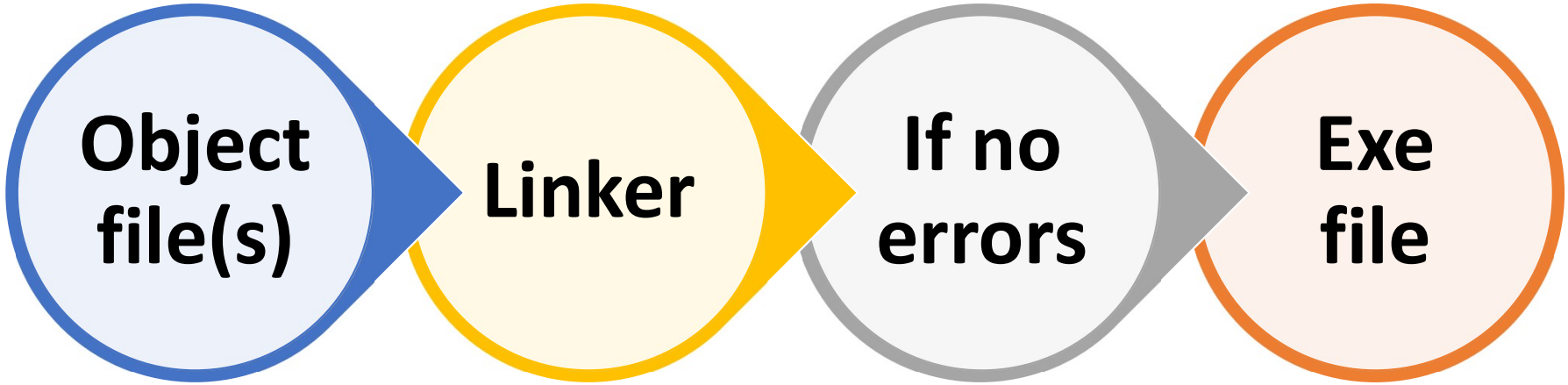
## STEP 4 – LINKING

**Linker** links object file with any other needed object files

Creates a single executable file (C linkers create .exe files)

| | |
|---|---|
| Input to this step: | .o machine language files |
| Output from this step: | executable program with a .exe file extension (Windows) or .out extension (Linux) |
| Errors in this step: | files/libraries that can't be found |

# Step 4 – Linking Step



## STEP 5 – LOADING

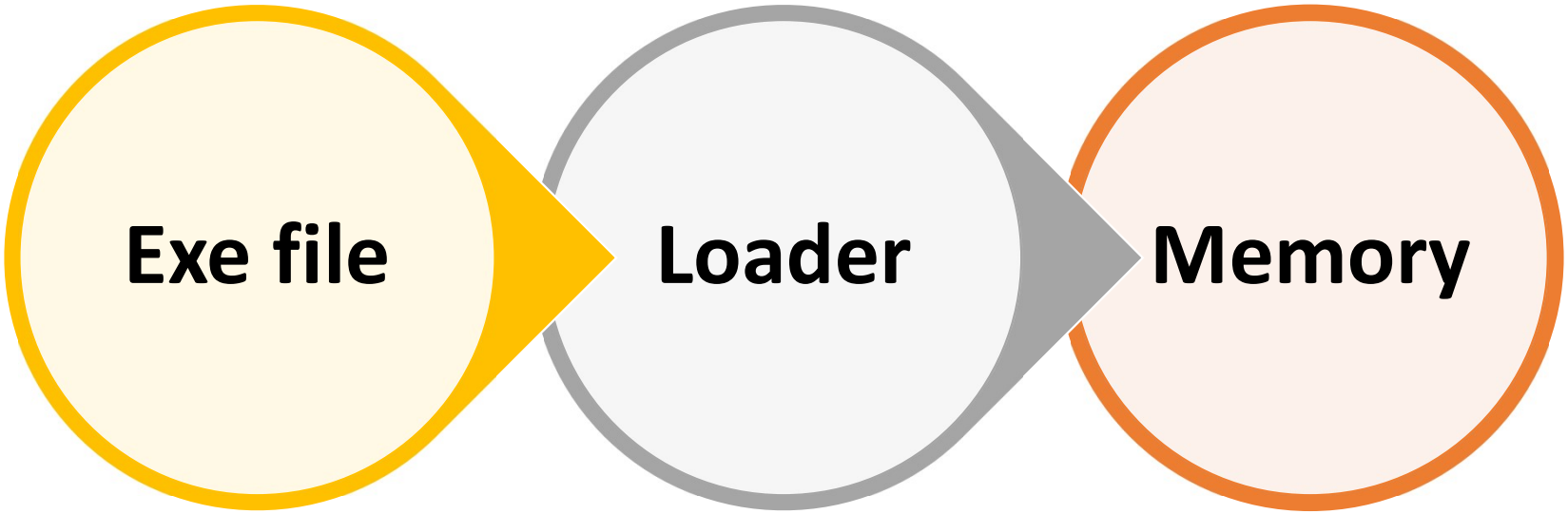
**Loader** copies the executable file into memory

And initiates execution

Input to this step: executable program (.exe file)

Output from this step: program execution

# Step 5 – Loading Step

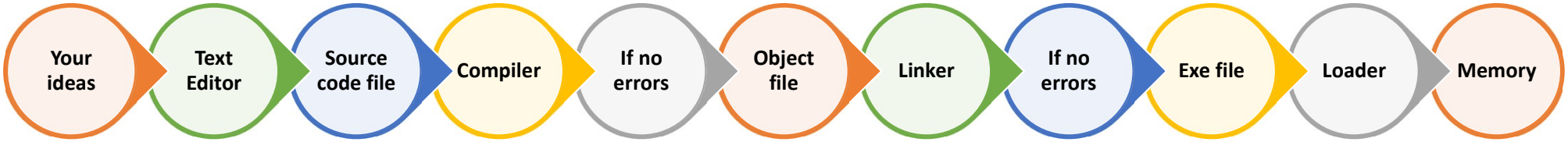


## STEP 6 – EXECUTION

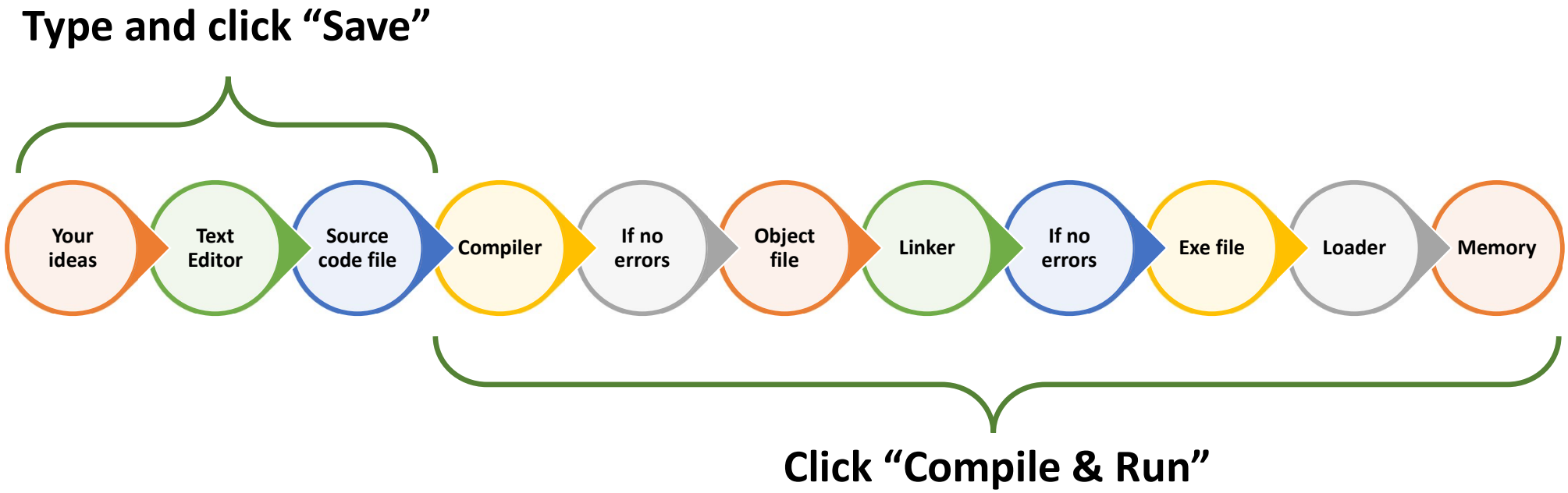
Program is executed by CPU, under control of OS and user

Errors in this step: runtime errors

# The Whole Process



# The Whole Process in Two Physical Steps



## Depending on IDE, The Individual Steps Can Be Isolated

Although all these steps seem like a complex process, as a programmer, you are shielded from the individual parts by your IDE.

But, depending on the IDE and the needs of your system, you can work with the individual pieces

This can be one advantage of using C over other languages that do not allow the complete process (source code to compilation) to be interrupted.

# Standard Input, Standard Output, and Standard Error Streams

# Standard Input, Output, and Error Streams

Standard input

- Usually keyboard

Standard output

- Usually the screen

Standard error

- Also usually output to screen

# Creating Programs: The Software Development Method

# The Software Development Method

Programming is problem-solving!

Steps:

- Specify
- Analyze
- Design
- Implement
- Test and Verify
- Maintain and Update

## STEP 1 – SPECIFY THE PROBLEM REQUIREMENTS

State the problem clearly.

Zero in on root problem.

# STEP 2 – ANALYZE THE PROBLEM

Identify:

- Inputs (what data you have to work with)
- Outputs (desired results)
- Any additional constraints

Abstraction

- Process of modeling the problem by extracting essential variables and their relationships

# STEP 3 – DESIGN ALGORITHM TO SOLVE THE PROBLEM

Algorithm

- List of steps to solve the problem

Top-Down design (divide & conquer)

- List major steps, then break down major steps into smaller steps

Typical algorithm for a programming problem:

- Get the data
- Process the data
- Display results

# STEP 3 – DESIGN ALGORITHM TO SOLVE THE PROBLEM, CONTINUED

## Stepwise refinement

- Development of a detailed list of steps to solve a particular step in the original algorithm

## Desk check

- Step-by-step simulation of the computer execution of an algorithm

# STEP 4 – IMPLEMENT THE ALGORITHM

Write the steps as a program, using a programming language

## STEP 5 – TEST AND VERIFY COMPLETED PROGRAM

Run program many times with different sets of test data

Test that program works correctly for every situation provided for in algorithm

## STEP 6 – MAINTAIN AND UPDATE THE PROGRAM

Fix previously undetected errors, modifying it as necessary

**"Failure" is Part of the Process!**