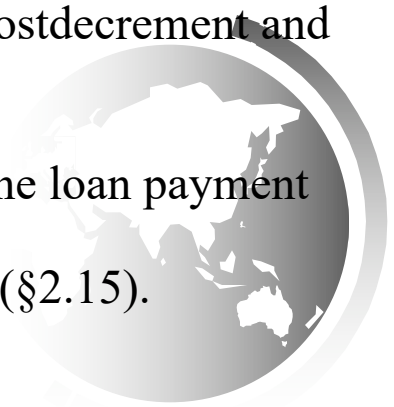
Chapter 2 Elementary Programming

Motivations

In the preceding chapter, you learned how to create, compile, and run a program. Starting from this chapter, you will learn how to solve practical problems programmatically. Through these problems, you will learn primitive data types and related subjects, such as variables, constants, data types, operators, expressions, and input and output.

Objectives

- To write C++ programs that perform simple computations (§2.2).
- To read input from the keyboard (§2.3).
- To use identifiers to name elements such as variables and functions (§2.4).
- To use variables to store data (§2.5).
- To program using assignment statements and assignment expressions (§2.6).
- To name constants using the **const** keyword (§2.7).
- To declare variables using numeric data types (§2.8.1).
- To write integer literals, floating-point literals, and literals in scientific notation (§2.8.2).
- To perform operations using operators +, **-**, *, /, and **%** (§2.8.3).
- To perform exponent operations using the **pow(a, b)** function (§2.8.4).
- To write and evaluate expressions (§2.9).
- To obtain the current system time using **time(0)** (§2.10).
- To use augmented assignment operators (+=, **-**=, *=, /=, **%**=) (§2.11).
- To distinguish between postincrement and preincrement and between postdecrement and predecrement (§2.12).
- To convert numbers to a different type using casting (§2.13).
- To describe the software development process and apply it to develop the loan payment program (§2.14).
- To write a program that converts a large amount of money into smaller (§2.15).
- To avoid common errors in elementary programming (§2.16).



Introducing Programming with an Example

Listing 2.1 Computing the Area of a Circle

This program computes the area of the circle.

ComputeArea

Note: Clicking the green button displays the source code with interactive animation. You can also run the code in a browser. Internet connection is needed for this button.

animation

Trace a Program Execution

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

int main() {
  double radius;
  double area;

  // Step 1: Read in radius
  radius = 20;

  // Step 2: Compute area
  area = radius * radius * 3.14159;

  // Step 3: Display the area
  cout << "The area is ";
  cout << area << endl;
}
```

allocate memory for radius

radius | no value

animation

Trace a Program Execution

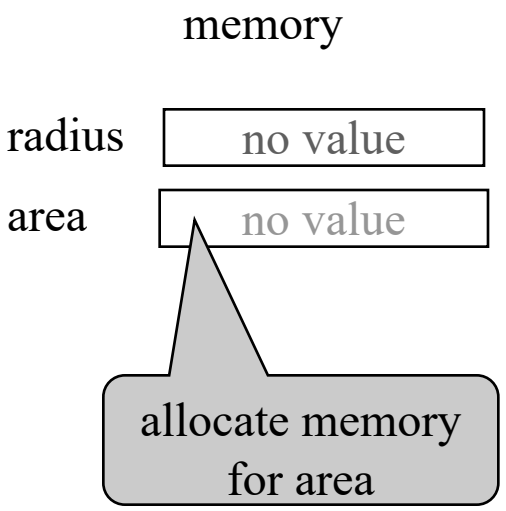
```
#include <iostream>
using namespace std;

int main() {
  double radius;
  double area;

  // Step 1: Read in radius
  radius = 20;

  // Step 2: Compute area
  area = radius * radius * 3.14159;

  // Step 3: Display the area
  cout << "The area is ";
  cout << area << std::endl;
}
```



animation

Trace a Program Execution

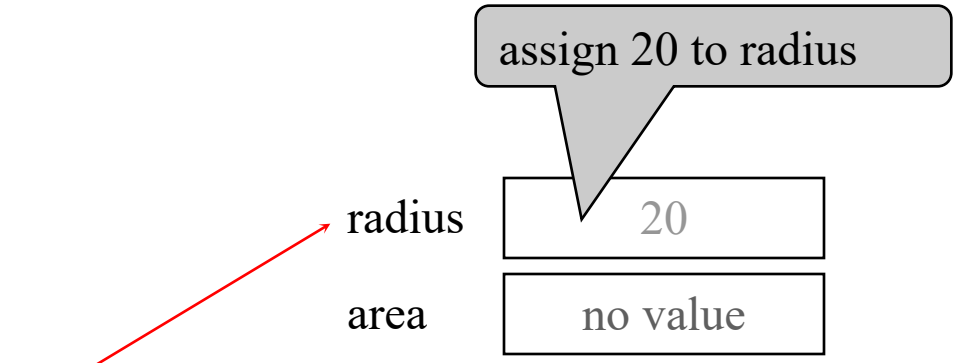
```
#include <iostream>
using namespace std;

int main() {
  double radius;
  double area;

  // Step 1: Read in radius
  radius = 20;

  // Step 2: Compute area
  area = radius * radius * 3.14159;

  // Step 3: Display the area
  cout << "The area is ";
  cout << area << std::endl;
}
```



animation

Trace a Program Execution

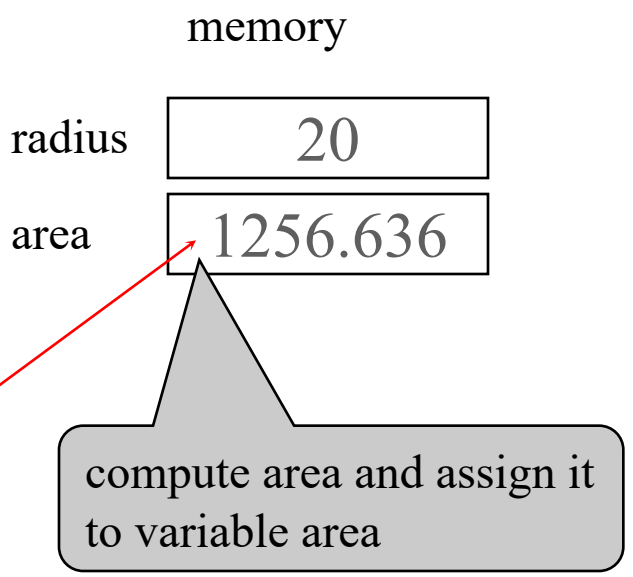
```
#include <iostream>
using namespace std;

int main() {
  double radius;
  double area;

  // Step 1: Read in radius
  radius = 20;

  // Step 2: Compute area
  area = radius * radius * 3.14159;

  // Step 3: Display the area
  cout << "The area is ";
  cout << area << std::endl;
}
```



animation

Trace a Program Execution

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

int main() {
  double radius;
  double area;

  // Step 1: Read in radius
  radius = 20;

  // Step 2: Compute area
  area = radius * radius * 3.14159;

  // Step 3: Display the area
  cout << "The area is ";
  cout << area << std::endl;
}
```

memory

radius	20
area	1256.636

print a message to the console

Reading Input from the Keyboard

You can use the cin object to read input from the keyboard.

ComputeAreaWithConsoleInput

Reading Multiple Input in One Statement

ComputeAverage

Identifiers

- An identifier is a sequence of characters that consists of letters, digits, and underscores (_).
- An identifier must start with a letter or an underscore. It cannot start with a digit.
- An identifier cannot be a reserved word. (See Appendix A, "C++ Keywords," for a list of reserved words.)
- An identifier can be of any length, but your C++ compiler may impose some restriction. Use identifiers of 31 characters or fewer to ensure portability.

Variables

```
// Compute the first area
radius = 1.0;
area = radius * radius * 3.14159;
cout << area;

// Compute the second area
radius = 2.0;
area = radius * radius * 3.14159;
cout << area;
```

Declaring Variables

```
int x;            // Declare x to be an
                  // integer variable;
double radius;    // Declare radius to
                  // be a double variable;
char a;           // Declare a to be a
                  // character variable;
```

Assignment Statements

```
x = 1;              // Assign 1 to x;

radius = 1.0;       // Assign 1.0 to radius;

a = 'A';            // Assign 'A' to a;
```

Declaring and Initializing in One Step

- `int x = 1;`
- `double d = 1.4;`

Named Constants

```
const datatype CONSTANTNAME = VALUE;

const double PI = 3.14159;
const int SIZE = 3;
```

Numerical Data Types

Name	Synonymy	Range	Storage Size
short	short int	-2^{15} to $2^{15}-1$ (-32,768 to 32,767)	16-bit signed
unsigned short	unsigned short int	0 to $2^{16}-1$ (65535)	16-bit unsigned
int	signed	-2^{31} to $2^{31}-1$ (-2147483648 to 2147483647)	32-bit
unsigned	unsigned int	0 to $2^{32}-1$ (4294967295)	32-bit unsigned
long	long int	-2^{31} (-2147483648) to $2^{31}-1$ (2147483647)	32-bit signed
unsigned long	unsigned long int	0 to $2^{32}-1$ (4294967295)	32-bit unsigned
long long		-2^{63} (-9223372036854775808) to 263-1 (9223372036854775807)	64-bit signed
float		Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38	32-bit IEEE 754
double		Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308	64-bit IEEE 754
long double		Negative range: -1.18E+4932 to -3.37E-4932 Positive range: 3.37E-4932 to 1.18E+4932 Significant decimal digits: 19	80-bit

Numerical Data Types

Name	Synonymy	Range	Storage Size
short	short int	-2^{15} to $2^{15}-1$ (-32,768 to 32,767)	16-bit signed
unsigned short	unsigned short int	0 to $2^{16}-1$ (65535)	16-bit unsigned
int	signed	-2^{31} to $2^{31}-1$ (-2147483648 to 2147483647)	32-bit
unsigned	unsigned int	0 to $2^{32}-1$ (4294967295)	32-bit unsigned
long	long int	-2^{31} (-2147483648) to $2^{31}-1$ (2147483647)	32-bit signed
unsigned long	unsigned long int	0 to $2^{32}-1$ (4294967295)	32-bit unsigned
long long		-2^{63} (-9223372036854775808) to 263-1 (9223372036854775807)	64-bit signed
float		Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38	32-bit IEEE 754
double		Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308	64-bit IEEE 754
long double		Negative range: -1.18E+4932 to -3.37E-4932 Positive range: 3.37E-4932 to 1.18E+4932 Significant decimal digits: 19	80-bit

C++11: long long is defined in C++11

sizeof Function

You can use the sizeof function to find the size of a type. For example, the following statement displays the size of int, long, and double on your machine.

```
cout << sizeof(int) << " " << sizeof(long) << " " <<
sizeof(double);
```

Synonymous Types

short int is synonymous to short. unsigned short int is synonymous to unsigned short. unsigned int is synonymous to unsigned. long int is synonymous to long. unsigned long int is synonymous to unsigned long. For example,

```
short int i = 2;
```

is same as

```
short i = 2;
```

Numeric Literals

A *literal* is a constant value that appears directly in a program. For example, 34, 1000000, and 5.0 are literals in the following statements:

```
int i = 34;
long k = 1000000;
double d = 5.0;
```

octal and hex literals

By default, an integer literal is a decimal number. To denote an octal integer literal, use a leading *0* (zero), and to denote a hexadecimal integer literal, use a leading *0x* or *0X* (zero x). For example, the following code displays the decimal value 65535 for hexadecimal number FFFF and decimal value 8 for octal number 10.

```
cout << 0xFFFF << " " << 010;
```

double vs. float

The double type values are more accurate than the float type values. For example,

```
cout << "1.0 / 3.0 is " << 1.0 / 3.0 << endl;
```

displays `1.0 / 3.0 is 0.3333333333333331`

16 digits

```
cout << "1.0F / 3.0F is " << 1.0F / 3.0F << endl;
```

displays `1.0F / 3.0F is 0.3333333432674408`

7 digits

why called floating-point?

The float and double types are used to represent numbers with a decimal point. Why are they called *floating-point numbers*? These numbers are stored into scientific notation. When a number such as 50.534 is converted into scientific notation such as 5.0534e+1, its decimal point is moved (i.e., floated) to a new position.

Numeric Operators

Name	Meaning	Example	Result
+	Addition	34 + 1	35
-	Subtraction	34.0 - 0.1	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Remainder	20 % 3	2

Integer Division

+, -, *, /, and %

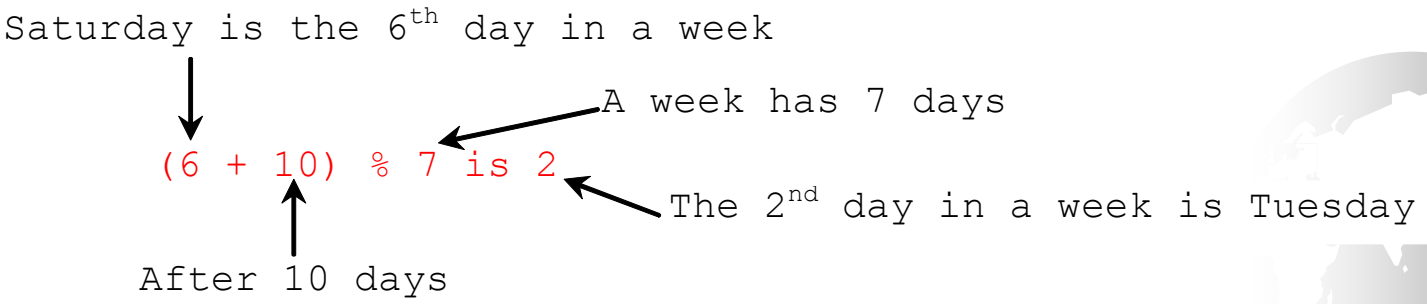
5 / 2 yields an integer 2.

5.0 / 2 yields a double value 2.5

5 % 2 yields 1 (the remainder of the division)

Remainder Operator

Remainder is very useful in programming. For example, an even number % 2 is always 0 and an odd number % 2 is always 1. So you can use this property to determine whether a number is even or odd. Suppose today is Saturday and you and your friends are going to meet in 10 days. What day is in 10 days? You can find that day is Tuesday using the following expression:



Example: Displaying Time

Write a program that obtains hours and minutes from seconds.

DisplayTime

Exponent Operations

```
cout << pow(2.0, 3) << endl; // Display 8.0
cout << pow(4.0, 0.5) << endl; // Display 2.0
cout << pow(2.5, 2) << endl; // Display 6.25
cout << pow(2.5, -2) << endl; // Display 0.16
```

Problem: Monetary Units

This program lets the user enter the amount in decimal representing dollars and cents and output a report listing the monetary equivalent in single dollars, quarters, dimes, nickels, and pennies. Your program should report maximum number of dollars, then the maximum number of quarters, and so on, in this order.

ComputeChange

Trace ComputeChange

Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);
```

remainingAmount 1156

remainingAmount initialized

```
// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```

animation

Trace ComputeChange

Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```

remainingAmount	1156
numberOfOneDollars	11

numberOfOneDollars assigned

animation

Trace ComputeChange

Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```

remainingAmount	56
numberOfOneDollars	11

remainingAmount updated

animation

Trace ComputeChange

Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```

remainingAmount	56
numberOfOneDollars	11
numberOfOneQuarters	2

numberOfOneQuarters assigned

animation

Trace ComputeChange

Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```

remainingAmount	6
numberOfOneDollars	11
numberOfQuarters	2

remainingAmount updated

Overflow

When a variable is assigned a value that is too large to be stored, it causes *overflow*. For example, executing the following statement causes *overflow*, because the largest value that can be stored in a variable of the short type is 32767. 32768 is too large.

short value = 32767 + 1;

Arithmetic Expressions

$$\frac{3+4x}{5} - \frac{10(y-5)(a+b+c)}{x} + 9\left(\frac{4}{x} + \frac{9+x}{y}\right)$$

is translated to

```
(3+4*x)/5 – 10*(y-5)*(a+b+c)/x + 9*(4/x + (9+x)/y)
```

Example: Converting Temperatures

Write a program that converts a Fahrenheit degree to Celsius using the formula:

$$celsius = (\frac{5}{9})(fahrenheit - 32)$$

FahrenheitToCelsius

Case Study: Displaying the Current Time

Write a program that displays current time in GMT in the format hour:minute:second such as 1:45:19.

The time(0) function in the ctime header file returns the current time in seconds elapsed since the time 00:00:00 on January 1, 1970 GMT, as shown in Figure 2.1. This time is known as the Unix epoch because 1970 was the year when the Unix operating system was formally introduced.

ShowCurrentTime

Augmented Assignment Operators

Operator	*Example*	*Equivalent*
`+=`	`i += 8`	`i = i + 8`
`-=`	`f -= 8.0`	`f = f - 8.0`
`*=`	`i *= 8`	`i = i * 8`
`/=`	`i /= 8`	`i = i / 8`
`%=`	`i %= 8`	`i = i % 8`

Increment and Decrement Operators

Operator	Name	Description
++var	preincrement	The expression (++var) increments var by 1 and evaluates to the *new* value in var *after* the increment.
var++	postincrement	The expression (var++) evaluates to the *original* value in var and increments var by 1.
--var	predecrement	The expression (--var) decrements var by 1 and evaluates to the *new* value in var *after* the decrement.
var--	postdecrement	The expression (var--) evaluates to the *original* value in var and decrements var by 1.

Increment and Decrement Operators, cont.

```
int i = 10;
int newNum = 10 * i++;
```

Same effect as

```
int newNum = 10 * i
i = i + 1;
```

```
int i = 10;
int newNum = 10 * (++i);
```

Same effect as

```
i = i + 1;
int newNum = 10 * i;
```

Increment and Decrement Operators, cont.

Using increment and decrement operators makes expressions short, but it also makes them complex and difficult to read. Avoid using these operators in expressions that modify multiple variables, or the same variable for multiple times such as this: int k = ++i + i.

Numeric Type Conversion

Consider the following statements:

```
short i = 100;
long k = i * 3 + 4;
double d = i * 3.1 + k / 2;
```

Type Casting

Implicit casting

double d = 3; (type widening)

Explicit casting

int i = static_cast<int>(3.0); (type narrowing)

int i = (int)3.9; (Fraction part is truncated)

NOTE

Casting does not change the variable being cast. For example, d is not changed after casting in the following code:

```
double d = 4.5;
int i = static_cast<int>(d);  // d is not changed
```

NOTE

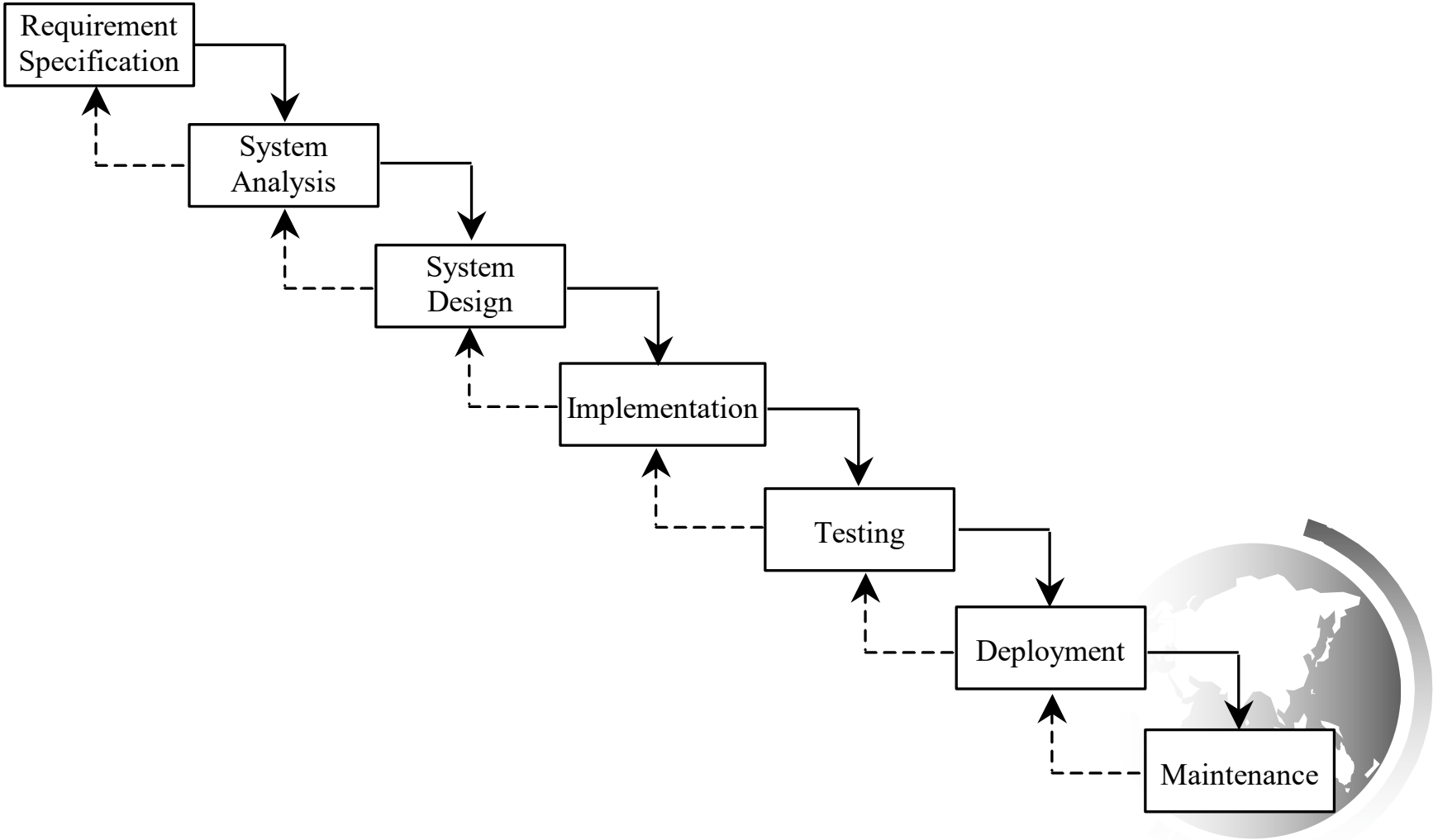
The GNU and Visual C++ compilers will give a warning when you narrow a type unless you use static_cast to make the conversion explicit.

Example: Keeping Two Digits After Decimal Points

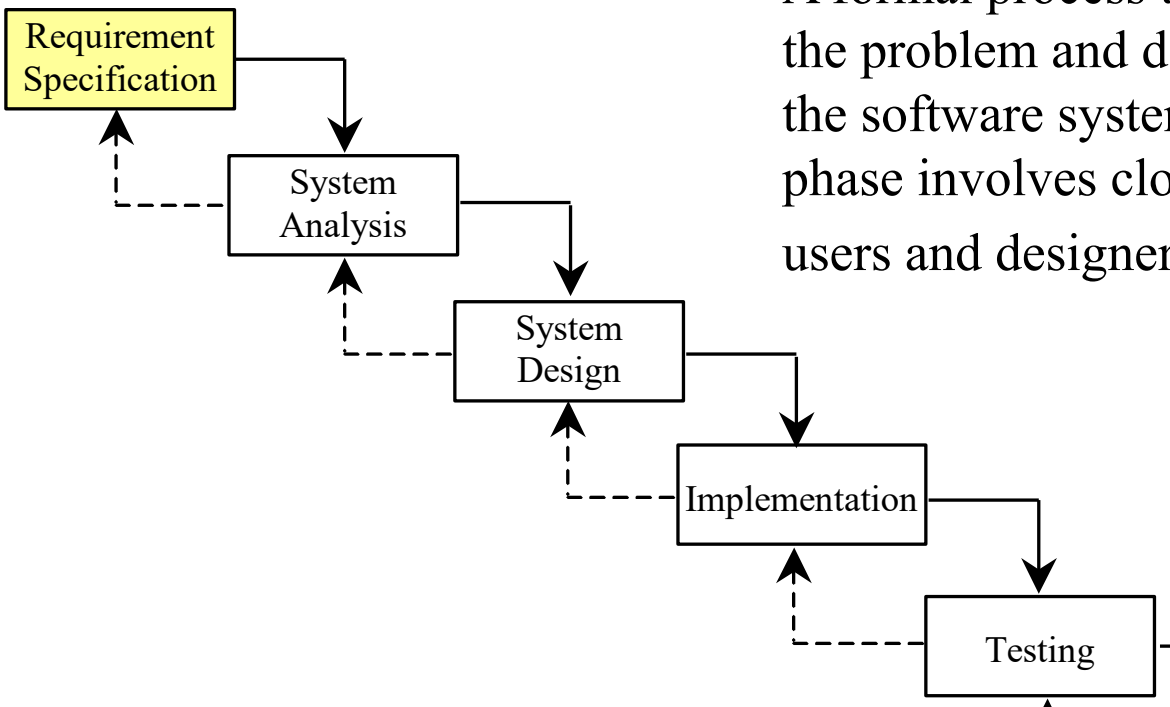
Write a program that displays the sales tax with two digits after the decimal point.

SalesTax

Software Development Process



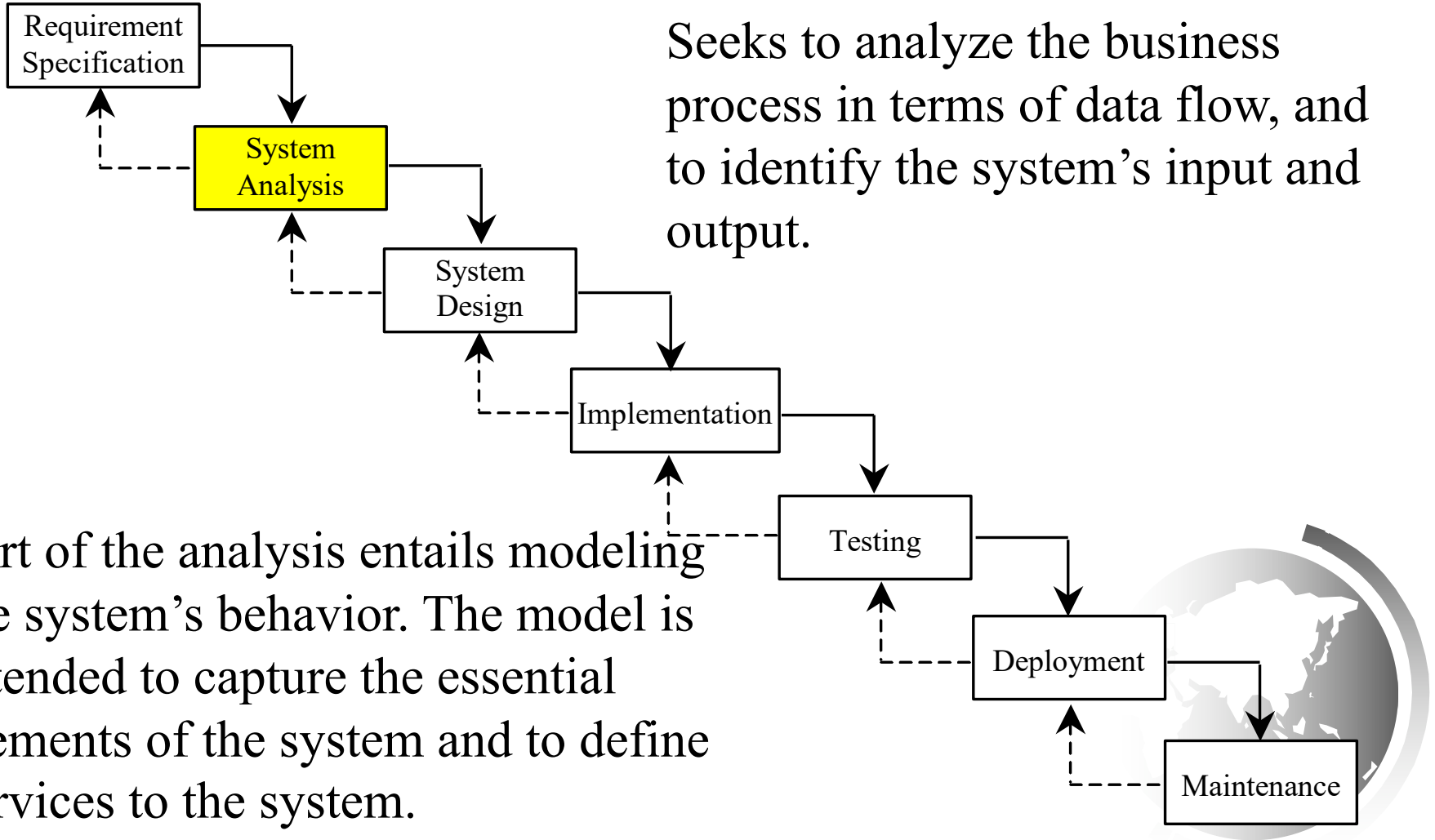
Requirement Specification



A formal process that seeks to understand the problem and document in detail what the software system needs to do. This phase involves close interaction between users and designers.

Most of the examples in this book are simple, and their requirements are clearly stated. In the real world, however, problems are not well defined. You need to study a problem carefully to identify its requirements.

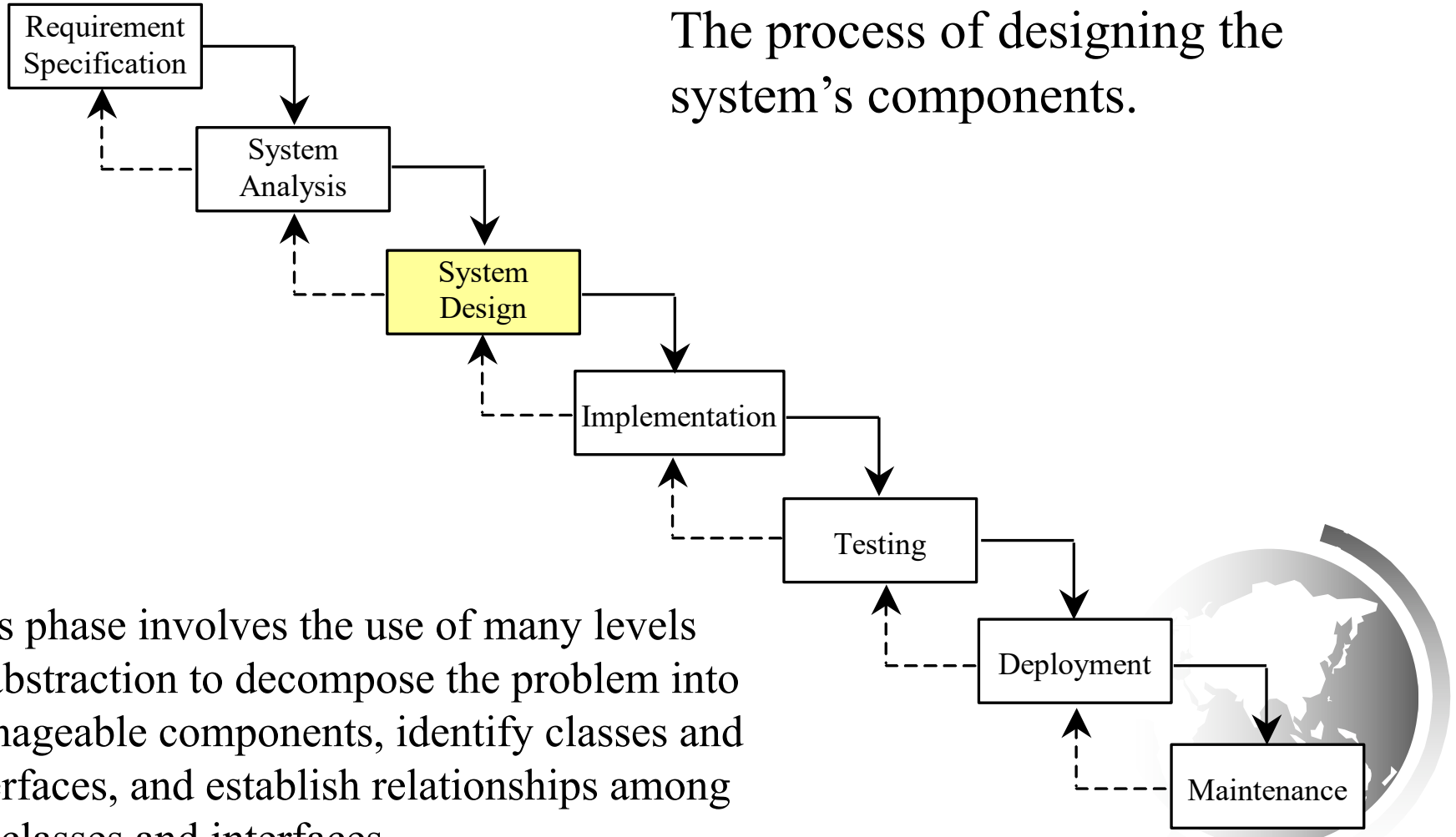
System Analysis



Seeks to analyze the business process in terms of data flow, and to identify the system's input and output.

Part of the analysis entails modeling the system's behavior. The model is intended to capture the essential elements of the system and to define services to the system.

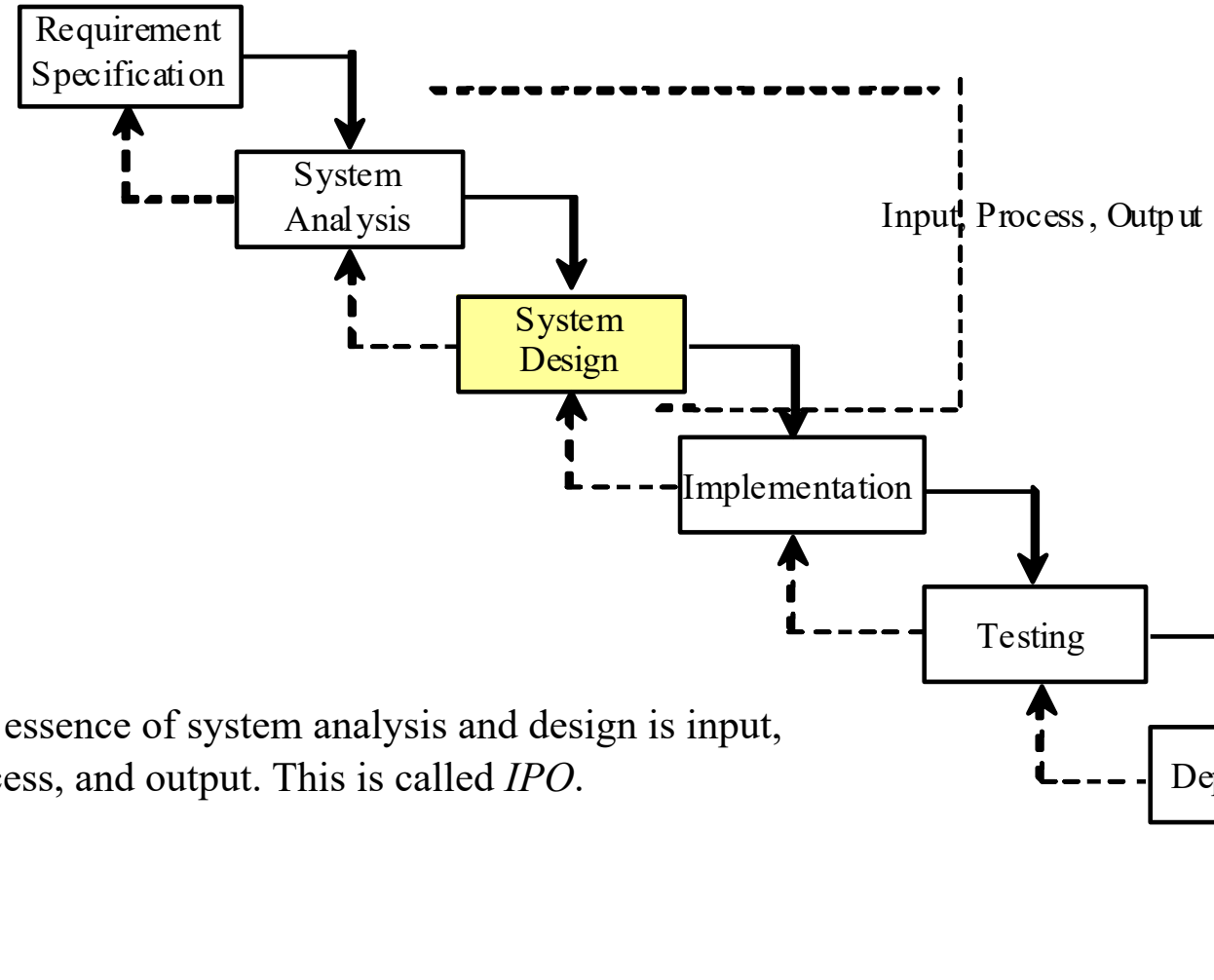
System Design



The process of designing the system's components.

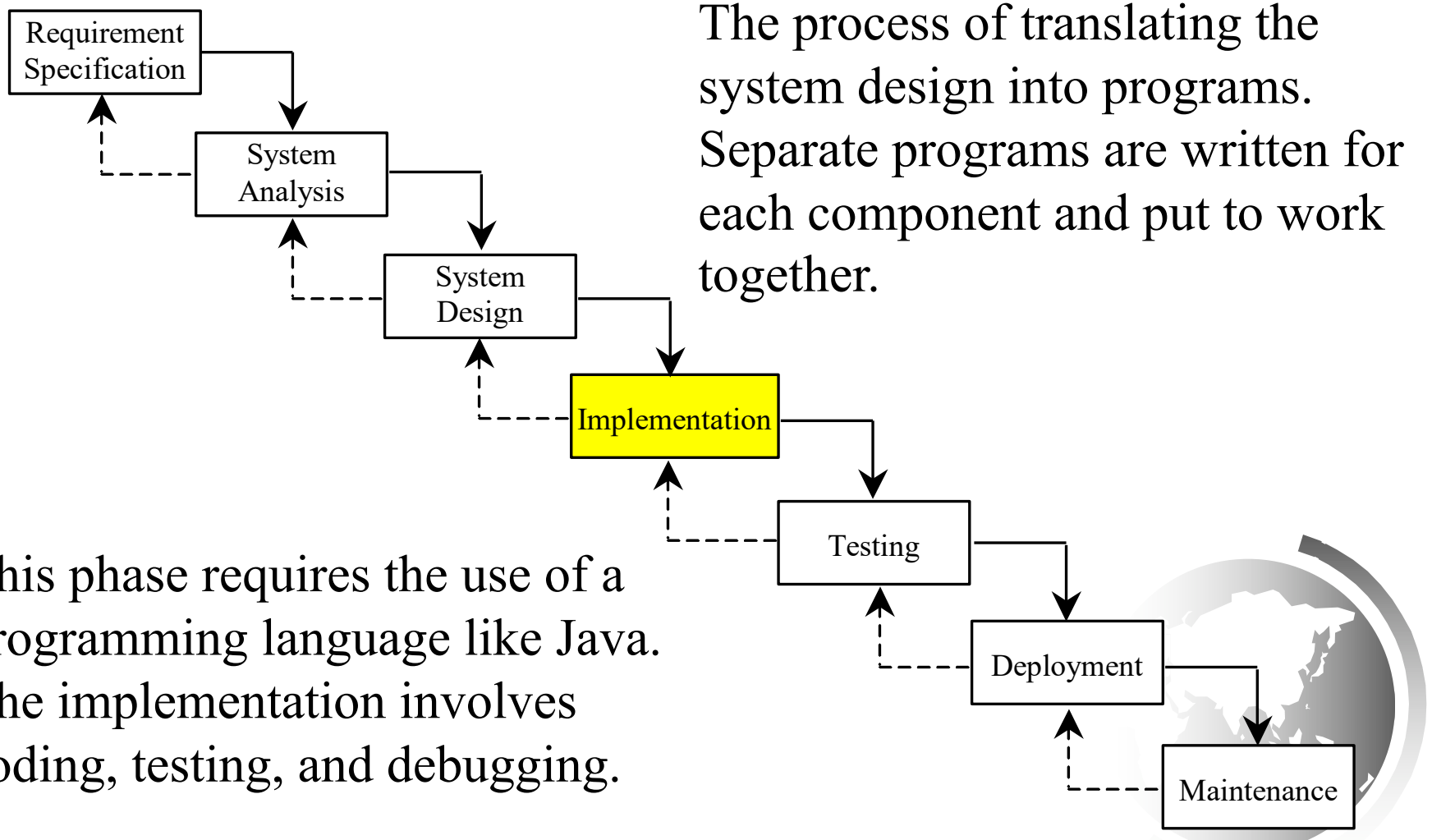
This phase involves the use of many levels of abstraction to decompose the problem into manageable components, identify classes and interfaces, and establish relationships among the classes and interfaces.

IPO



The essence of system analysis and design is input, process, and output. This is called *IPO*.

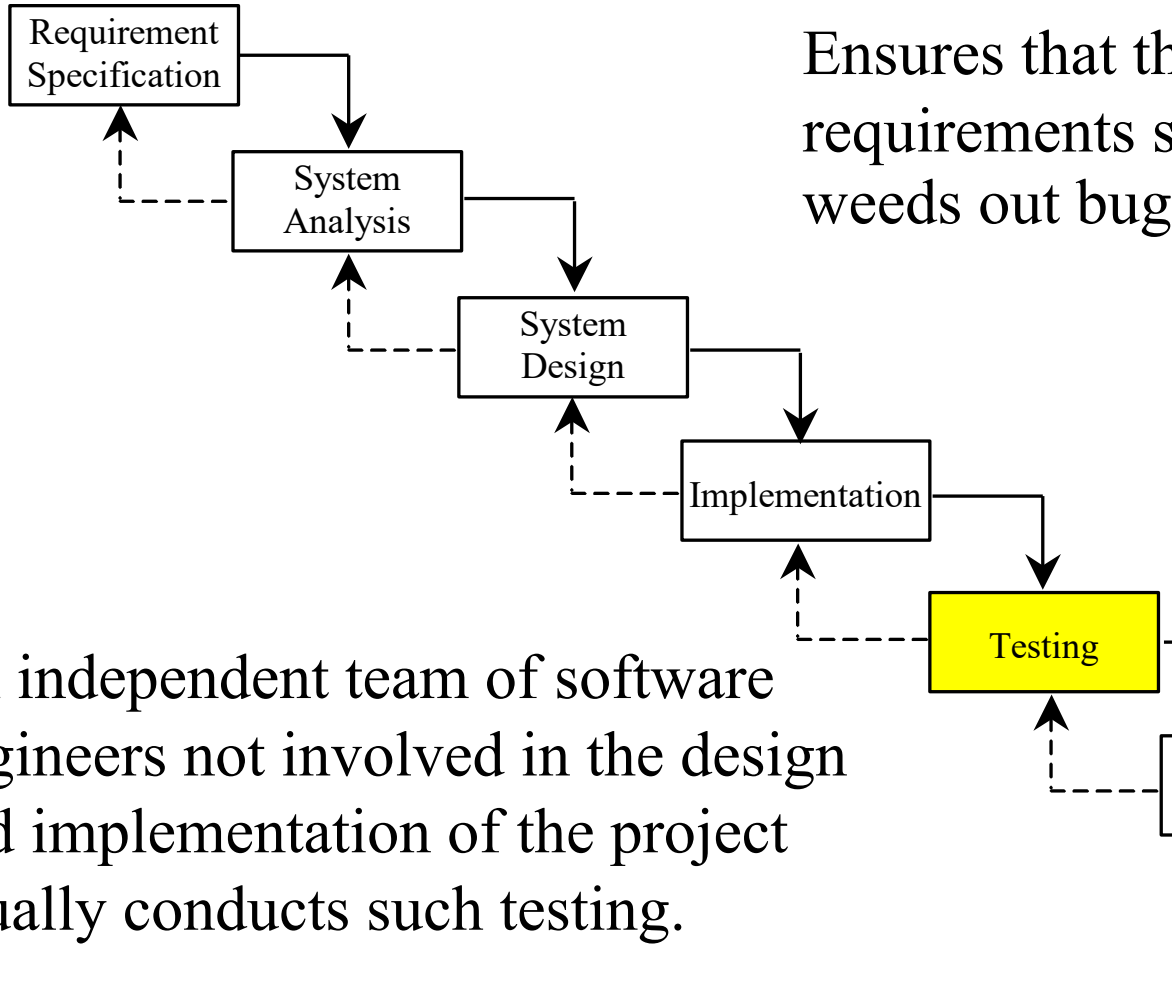
Implementation



The process of translating the system design into programs. Separate programs are written for each component and put to work together.

This phase requires the use of a programming language like Java. The implementation involves coding, testing, and debugging.

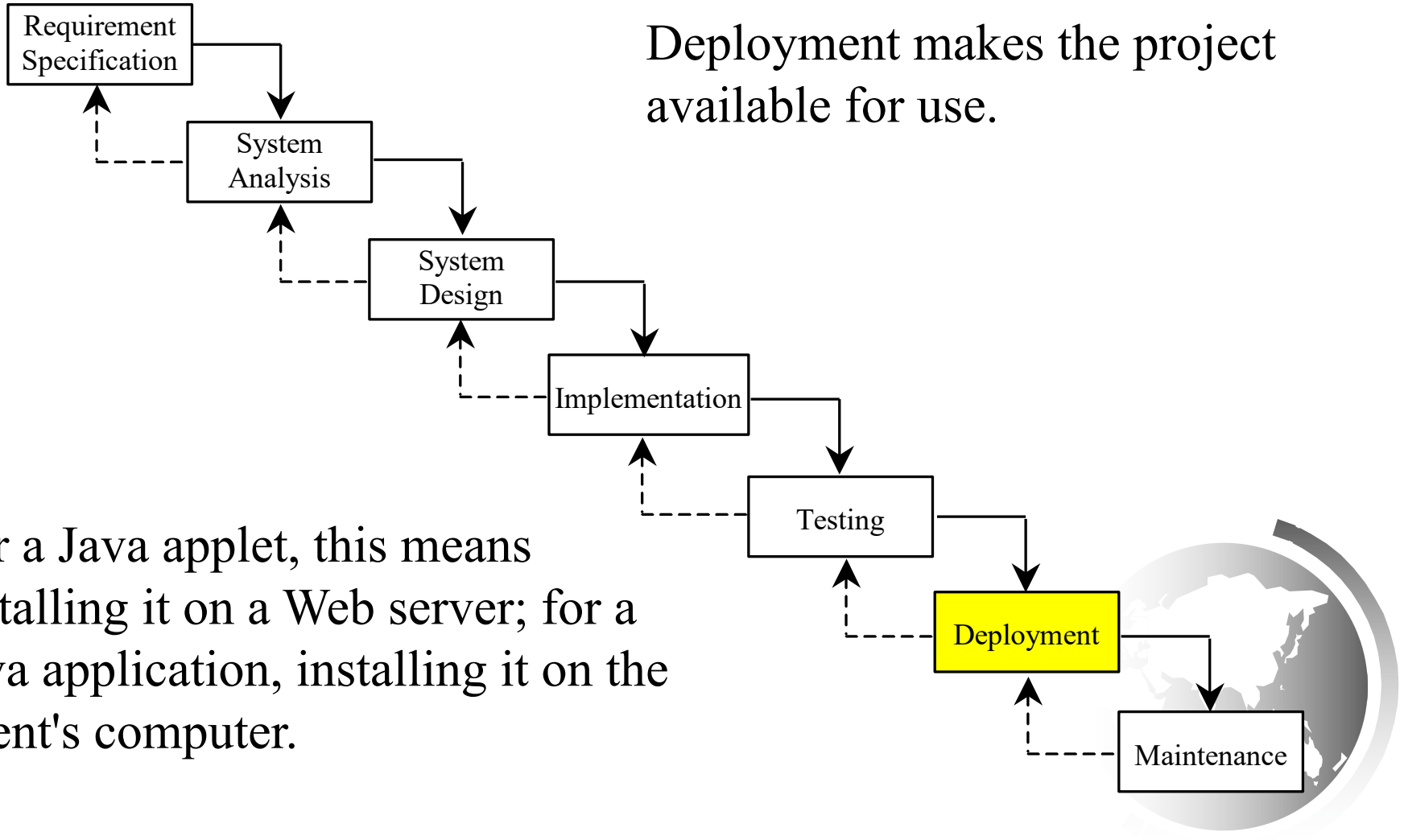
Testing



Ensures that the code meets the requirements specification and weeds out bugs.

An independent team of software engineers not involved in the design and implementation of the project usually conducts such testing.

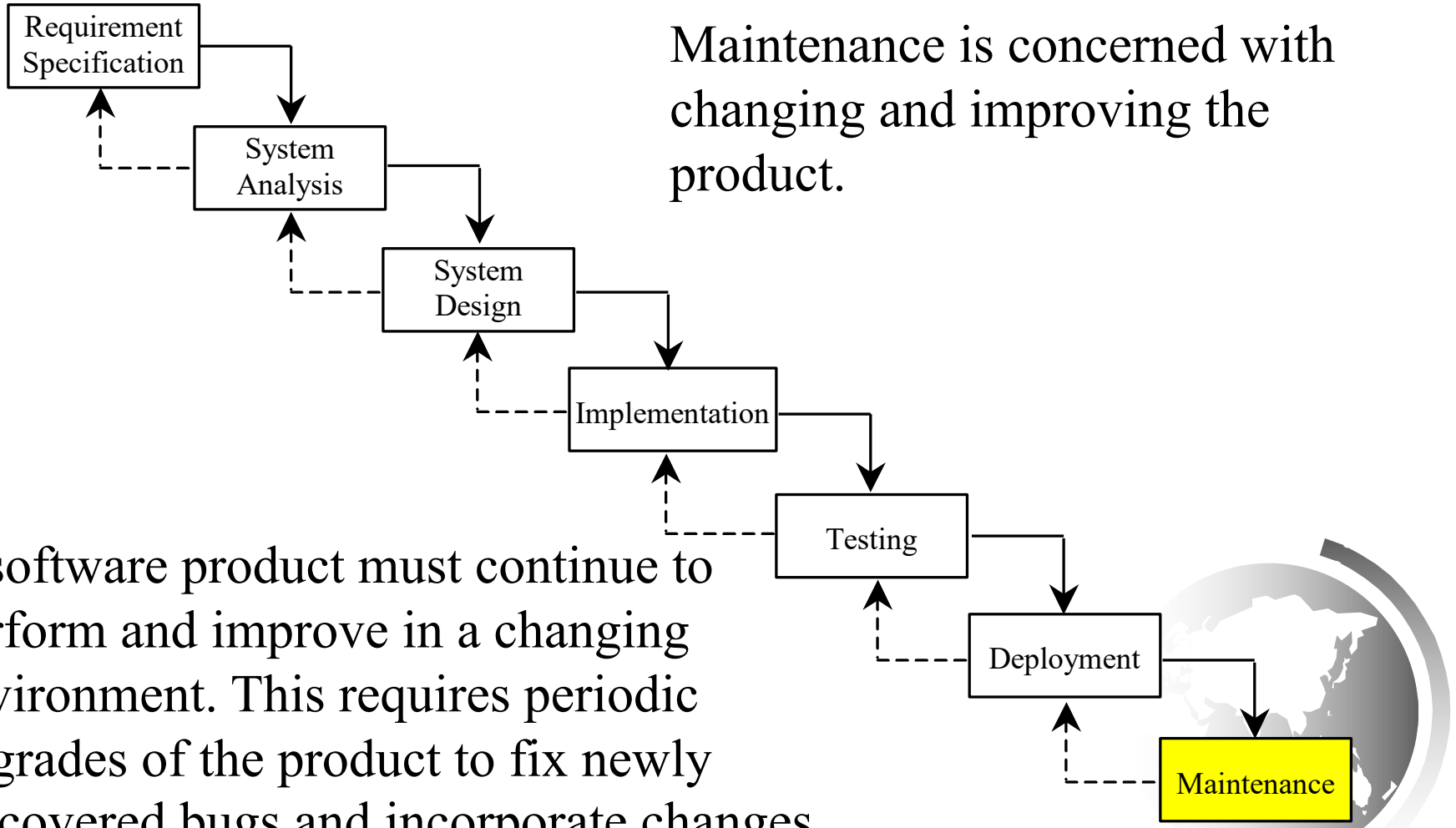
Deployment



Deployment makes the project available for use.

For a Java applet, this means installing it on a Web server; for a Java application, installing it on the client's computer.

Maintenance



Maintenance is concerned with changing and improving the product.

A software product must continue to perform and improve in a changing environment. This requires periodic upgrades of the product to fix newly discovered bugs and incorporate changes.

Problem: Computing Loan Payments

This program lets the user enter the interest rate, number of years, and loan amount and computes monthly payment and total payment.

$$\frac{loanAmount \times monthlyInterestRate}{1 - \frac{1}{(1 + monthlyInterestRate)^{numberOfYears \times 12}}}$$

ComputeLoan

Common Errors

- Common Error 1: Undeclared/Uninitialized Variables and Unused Variables
- Common Error 2: Integer Overflow
- Common Error 3: Round-off Errors
- Common Error 4: Unintended Integer Division
- Common Error 5: Forgetting Header Files