COMPREHENSIVE EDITION

PROGRAMMING AND PROBLEM SOLVING WITH C++

SIXTH EDITION

Nell Dale and Chip Weems

Chapter 2

C++ Syntax and Semantics, and the Program Development Process

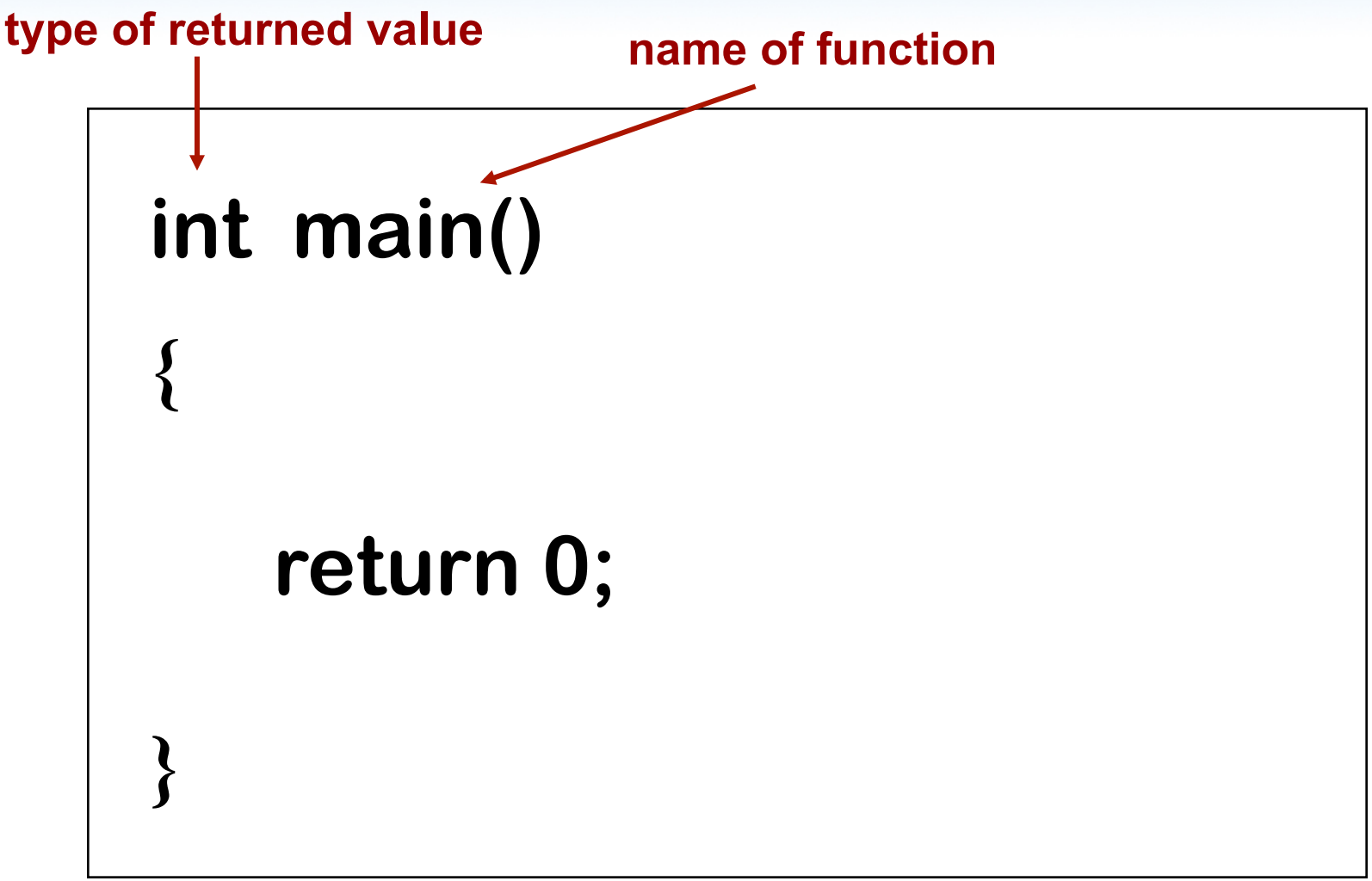
Syntax and Semantics

- The *syntax rules* of a language define how we can put together symbols, reserved words, and identifiers to make a valid program
- The *semantics* of a program statement define what that statement means (its purpose or role in a program)
- A program that is syntactically correct is not necessarily logically (semantically) correct
- A program will always do what we tell it to do, not what we meant to tell it to do

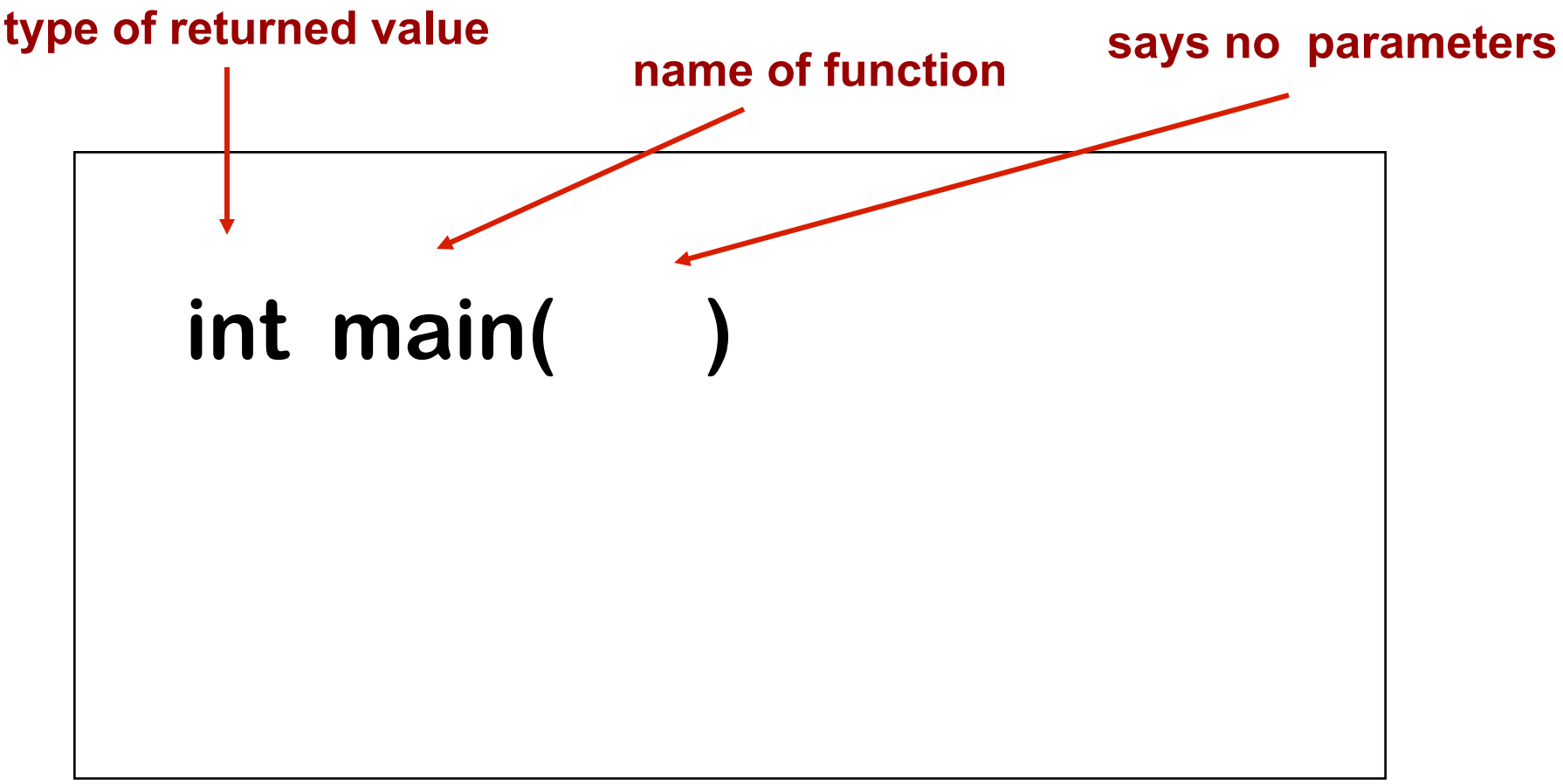
Chapter 2 Topics

- Syntax Templates
- Programs Composed of Several Functions
- Legal C++ Identifiers
- Assigning Values to Variables
- Declaring Named Constants
- String Concatenation
- Output Statements
- C++ Program Comments

Shortest C++ Program



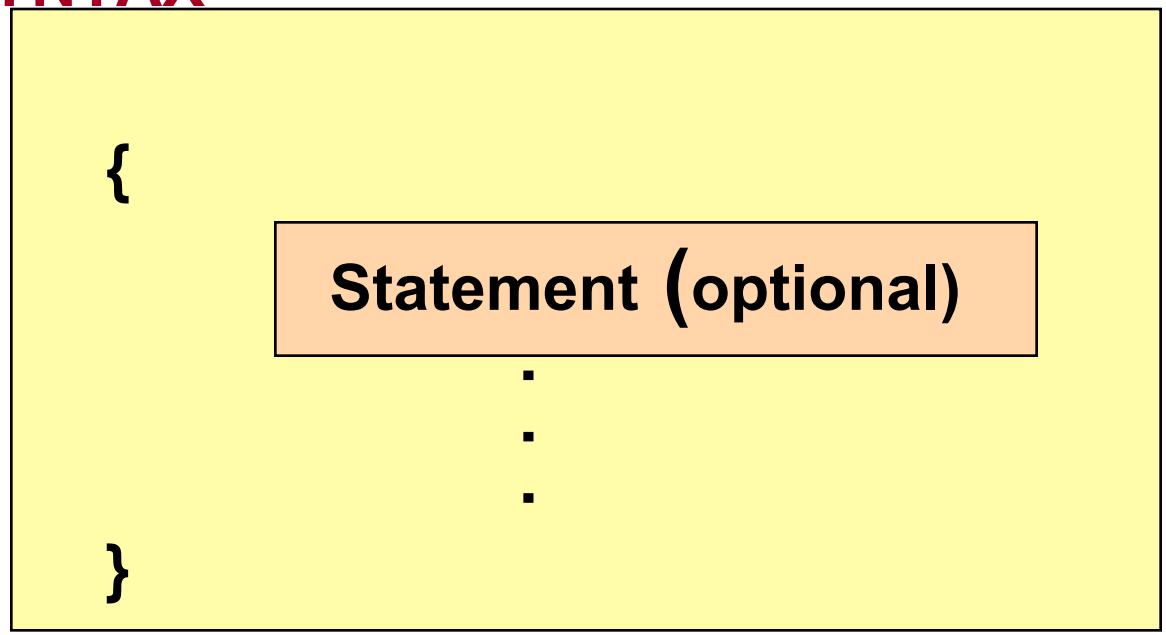
What is in a heading?



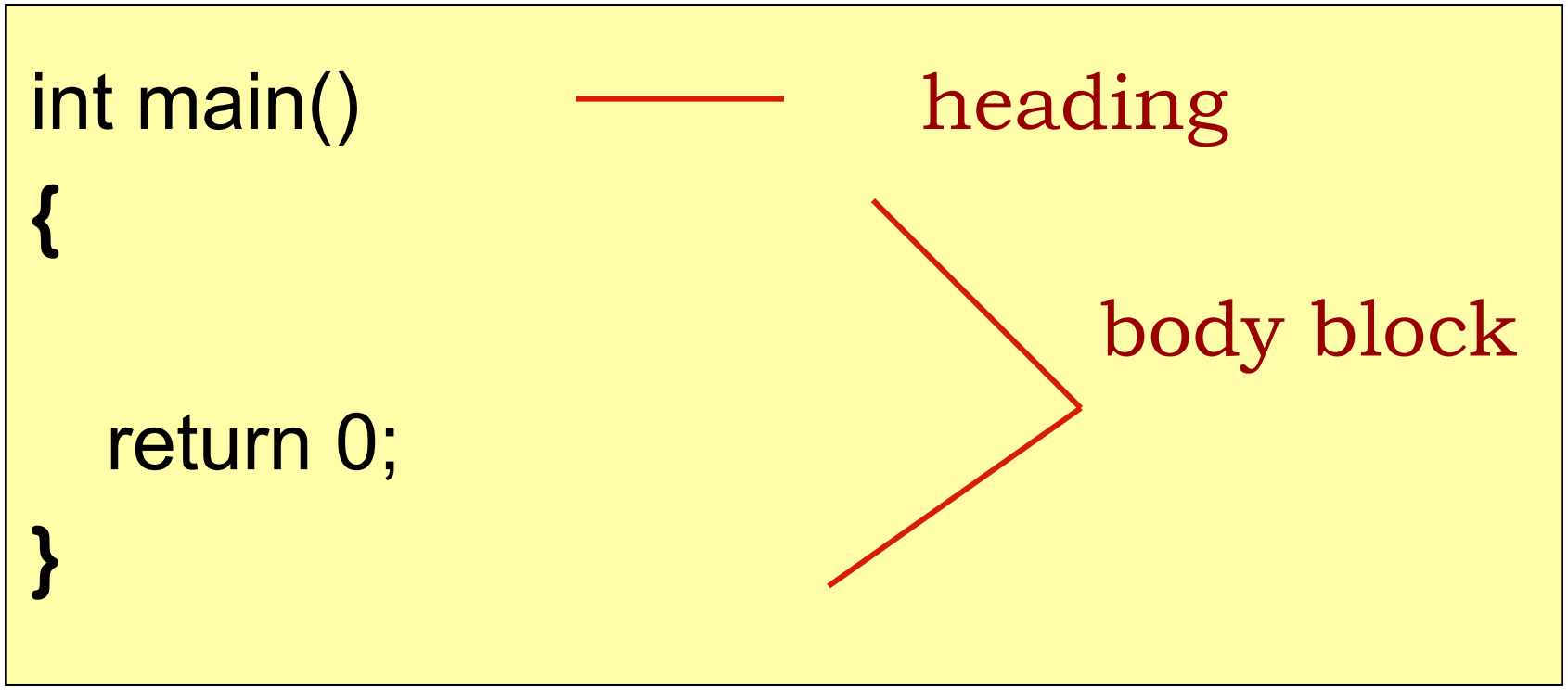
Block(Compound Statement)

- A **block** is a sequence of zero or more statements enclosed by a pair of curly braces { }

SYNTAX



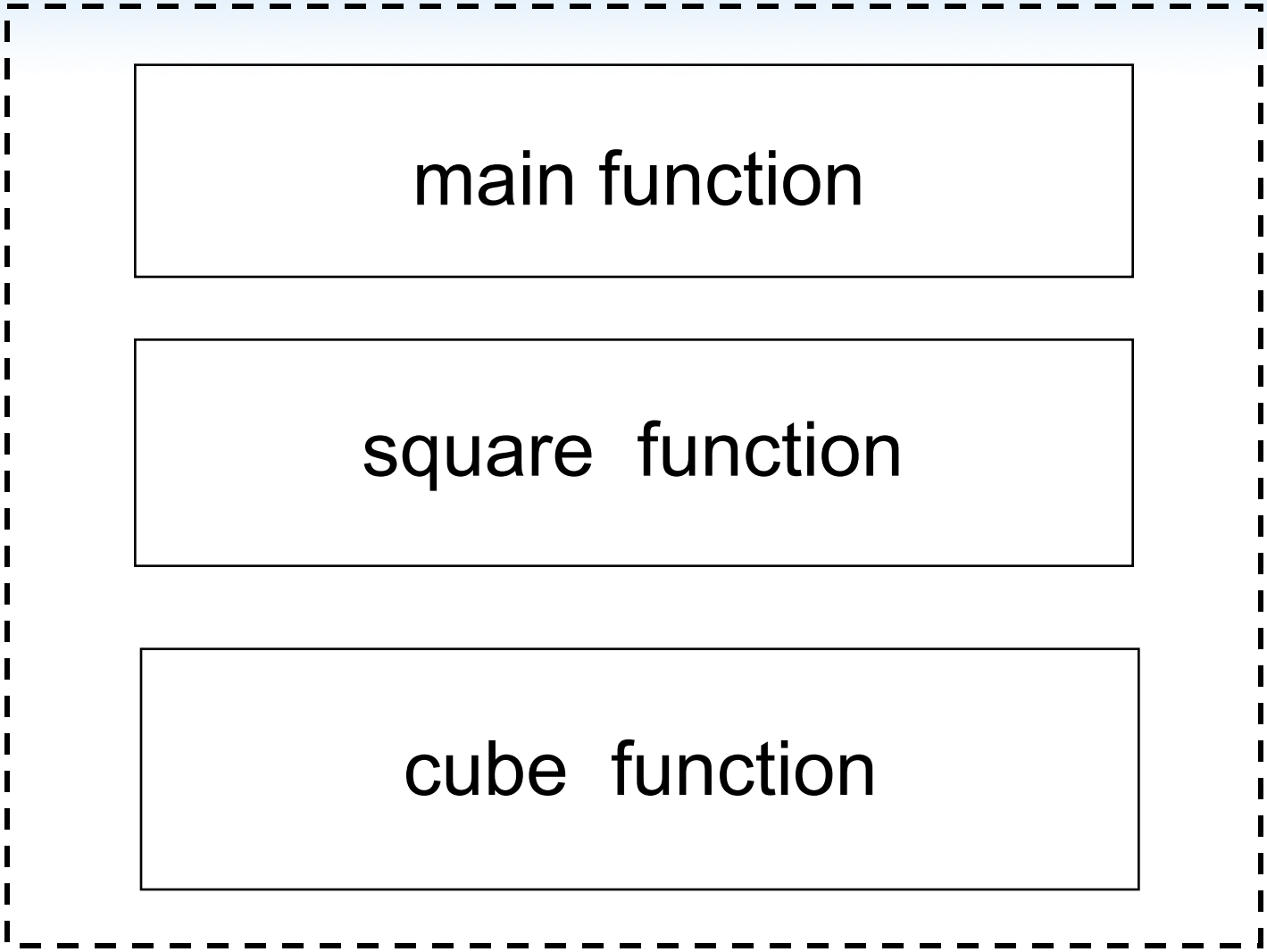
Every C++ function has 2 parts



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

squareCube27.cpp



squareCube27.cpp

```
#include <iostream>

int Square(int);          // Declares these two
int Cube(int);            // value-returning functions

using namespace std;

int main()
{
    cout << "The square of 27 is "
         << Square(27)<< endl;   // Function call

    cout << "The cube of 27 is "
        << Cube(27)<< endl;      // Function call
    return 0;
}
```

squareCube27.cpp (cont.)

```
int Square(int n)
{
    return n * n;
}


int Cube(int n)
{
    return n * n * n;
}
```

Output of squareCube27

```
The square of 27 is 729
The cube of 27 is 19683
```

Variables (1)

- Programs like squareCube27.cpp have little use in practice…
- There is no room for change. They simply print the same output onto the screen every time they are executed.
- A more interesting program might have different behavior under different circumstances.

Variables (2)

- For instance, a program that asks for a number and outputs its square and cube is much more useful than squareCube27.cpp.
  - This program needs placeholders for the incoming values.
  - These placeholders are called **variables**

squareCubeX.cpp

```
#include <iostream>

int Square(int);          // Declares these two
int Cube(int);            // value-returning functions
using namespace std;

int main()
{
    int x = 27;
    cout << "The square of "
         << x  << " is "
         << Square(x)<< endl;     // Function call

    cout << "The cube of " << x << " is "
         << Cube(x)<< endl;      // Function call
    return 0;
}
```

Declaration Statements

- **Before you can use a variable, you must *declare* it. This allows the computer to set aside the memory that will be needed for that variable.**

- Variables consist of a name and its data type. C++ variable declarations are of the form:

```
dataType variableName;
```

- **dataType** can be: int, float, char, double, unsigned int, ...
- **variableName** must be composed of alphanumeric characters or underscore ‘_’.

Declaration Example

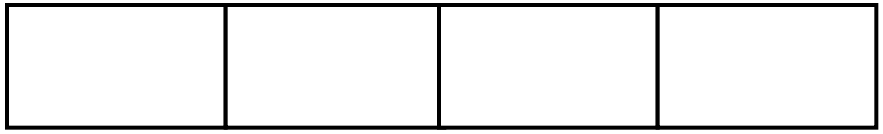
```
#include <iostream>
using namespace std;

int main()
{
  int age;
  float wage;
  char initial;
  double height;

  return 0;     // exit program
}
```

What Does a Declaration Do?

A declaration tells the compiler to allocate enough memory to hold a value of this data type and to associate the name with this location



4 bytes for wage

1 byte for initial

Variables (3)

- Variables are used to hold data within a program (the data is held in your computer's main memory). A program can read-from and write-to variables. That is, their values can *vary*.
- Every variable consists of two parts:
  1. The name (aka *identifier*) of a variable is tied to a location in memory
  2. Its data type (discussed later...)

What is an Identifier?

An identifier is the name used for a data object(a *variable* or a *constant*), or for a function, in a C++ program

Beware: C++ is a case-sensitive language

Using meaningful identifiers is a good programming practice

Identifiers

- An identifier must start with a letter or underscore, and be followed by zero or more letters

  (A-Z, a-z), digits(0-9), or underscores _

- VALID

  age_of_dog taxRateY2K

  PrintHeading ageOfHorse

- NOT VALID *(Why?)*

  age# 2000TaxRate Age-Of-Cat

More About Identifiers

- Some C++ compilers recognize only the first 32 characters of an identifier as significant

- Then these identifiers are considered the same:

```
age_Of_This_Old_Rhinoceros_At_My_Zoo
age_Of_This_Old_Rhinoceros_At_My_Safari
```

- Consider these:

```
Age_Of_This_Old_Rhinoceros_At_My_Zoo
age_Of_This_Old_Rhinoceros_At_My_Zoo
```

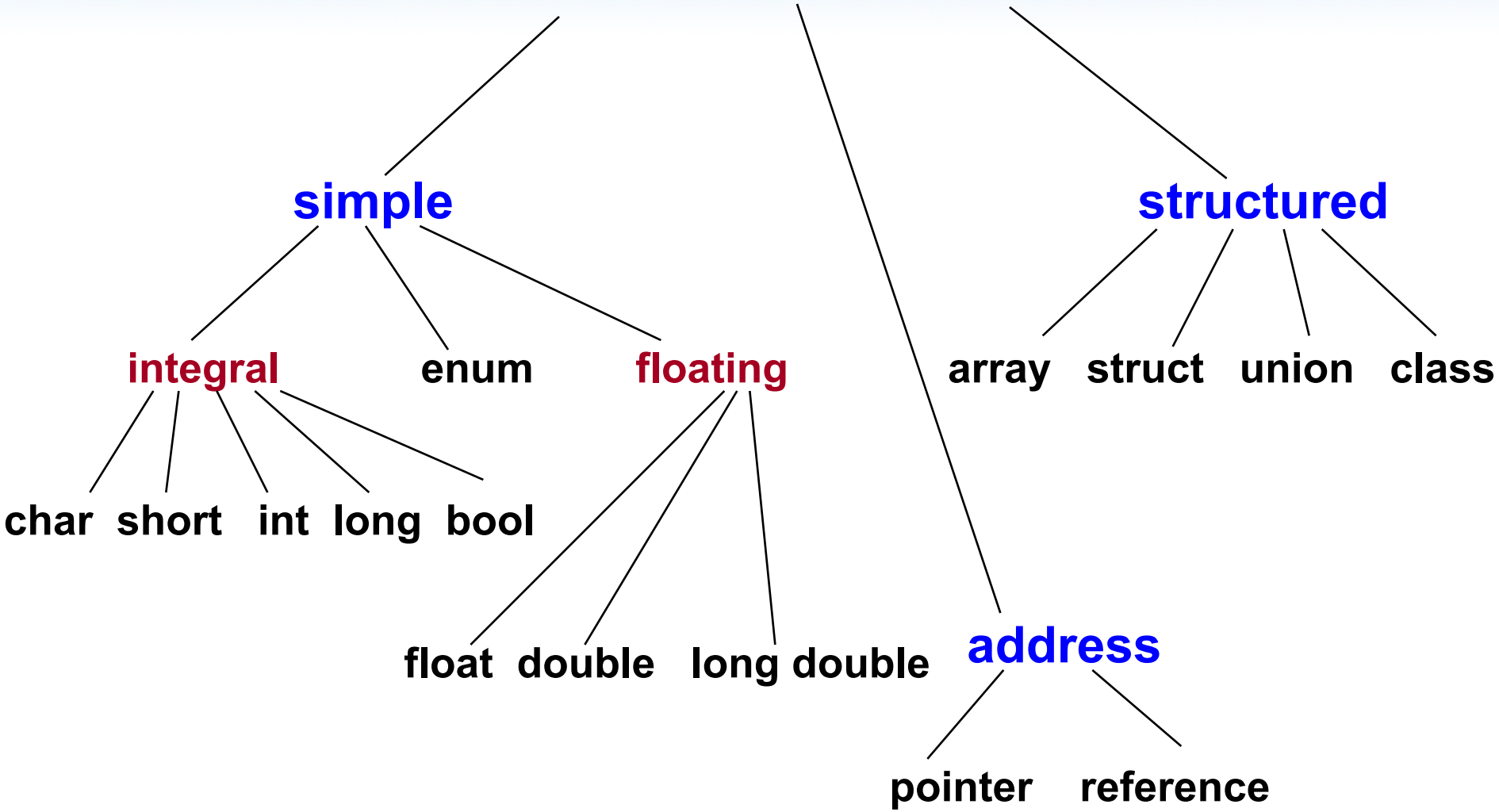
What is a Named Constant?

- A named constant is a location in memory that can be referred to by an identifier and in which a data value that cannot be changed is stored

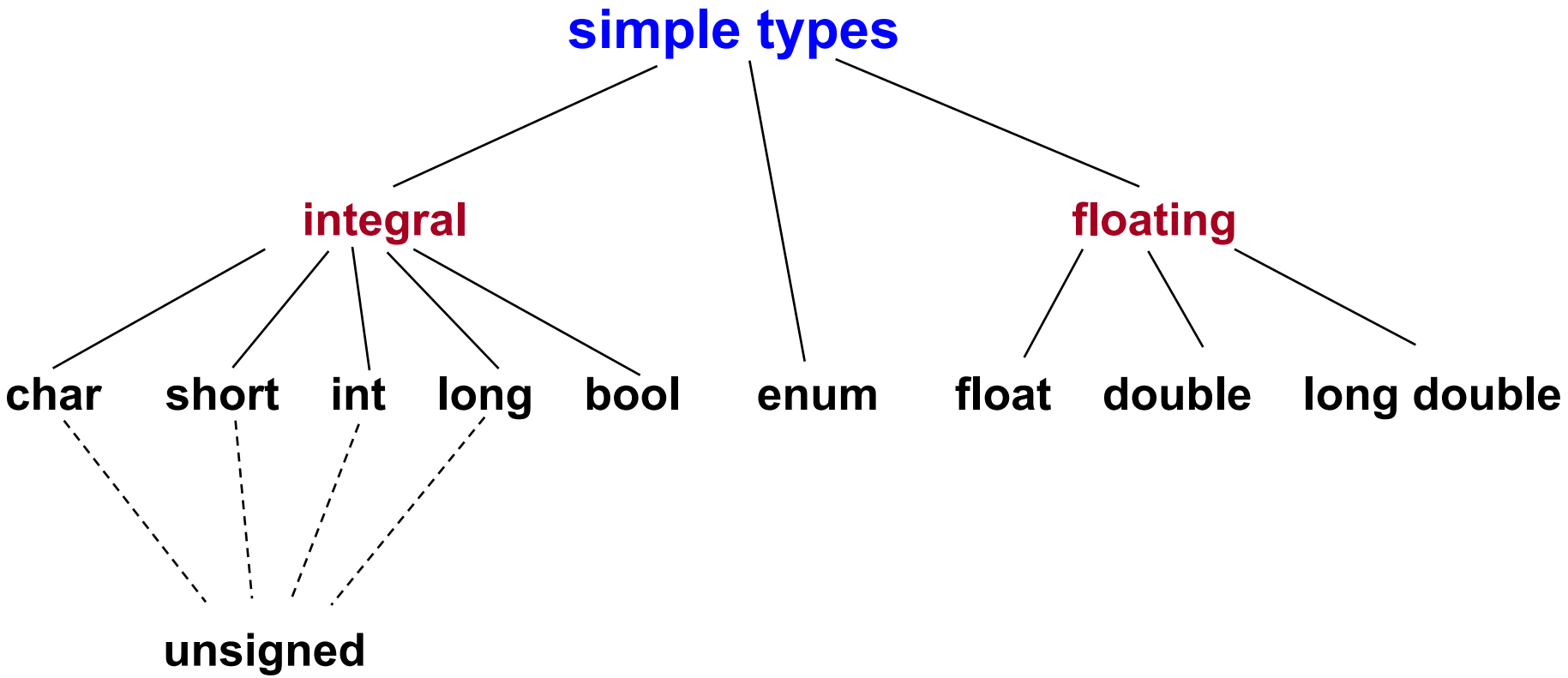
Valid constant declarations

```
const   string   STARS = "****";
const   float    NORMAL_TEMP = 98.6;
const   char     BLANK = ' ';
const   int      VOTING_AGE = 18;
const   float    MAX_HOURS = 40.0;
```

C++ Data Types



C++ Simple Data Types



Data Type: Integers (1)

- An integer value is any number that has no decimal point.

  123 -45000 +1432431 0 are all valid integers

- 1,244 is not a valid integer in C++; no commas are used. Neither is $12 because $ is not a valid part of an integer.

Data Type: Integers (2)

- The largest and smallest integers supported is dependent of the computer on which the program is compiled. Most of today's computers use 32-bits to represent an integer, so 2^{32} values can be represented.

- Integers can be **signed** or **unsigned.**
  - What is the maximum value of an unsigned 32 bit integer?
  - What is the maximum value of a signed 32 bit integer?

Integer Division

- In C++, (15 / 2) is 7. Why?
- Remember that integers have no fractional parts!
- There is no rounding in integer division. If your program contained: `cout << (15 / 16);` The output would be 0.

- The fractional part of an integer division is truncated so the result can be an integer. If you need the remainder, use %. If you need a decimal answer, make your operands floating-point numbers:

```
cout << (15.0 / 16.0);
```

Data Types: Floating Point Numbers (1)

- Floating-point numbers have a decimal point, and they can also be signed or unsigned.
- There are three basic types:

  float

  double

  long double

  - The differences between these are their supported range and precision.

Data Types: Floating Point Numbers (2)

- To represent a floating point number:
  - `float` uses 32 bits (4 bytes)
  - `double` uses 64 bits (8 bytes)
  - `long double` uses 128 bits (16 bytes)
- The tradeoff is storage vs. precision and range
- What exactly is the precision and range, and how are floating point numbers represented in binary format? **IEEE 754 Standard**

Data Types: Floating Point Numbers (3)

- Always use `double` to represent floating point numbers.

Data Types: Floating Point Numbers (4)

- Floating-point numbers can be written in exponential notation:

  134.56 or *1.3456e2*

  -0.00345 or *-3.45e-3*

- Here, `e` is short for "times ten to the power of", just like in scientific notation.

Data Type: Characters

- Characters include:
  - All letters of the alphabet (upper and lower case)
  - The symbolic representation of digits 0 – 9
  - All various symbols such as: + * & ^ % $ | , !
- A **character value** is simply one of these in single quotes, such as 'A' or '8' or ':' or ' ' (blank space).

Data Type: Characters

- A **character value** is a character in **single** quotes, such as 'A' or '8' or ':' or ' ' (blank space).
- NOTE: '8' and 8 are different.
  - '8' is the symbolic representation of the character, ‘8’;
  - 8 is the integer 8.

Data Type: Characters

- Characters usually take up 8 bits (1 byte)
- That means there are 2^8 (or 256) different characters, and every number within the range of [0, 255] is mapped onto some character
- So a character really boils down to a numerical representation known as ASCII encoding.

ASCII Code

Code	Char
32	Space
33	!
34	"
35	#
36	$
37	%
38	&
39	'
40	(
41	)
…	…

Code	Char
48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9
…	…

Code	Char
65	A
66	B
67	C
68	D
69	E
70	F
71	G
72	H
73	I
74	J
…	…

Code	Char
97	a
98	b
99	c
100	d
101	e
102	f
103	g
104	h
105	i
106	j
…	…

ASCII Table

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Characters Strings

- A character string is an array of characters.
- Examples:
  - "Hello"
  - "Hello, World!" (Note: Blank space is part of the string.)
  - "He who hesitates is lost.\nHaste makes waste.\n"
  - "" (The empty string.)

Character Strings

- NOTE: 'A' and "A" are different.
  - 'A' is the symbolic representation of the character, ‘A’;
  - "A" is a string containing a single character.
- NOTE: '8' and "8" and 8 are different.
  - '8' is the symbolic representation of the character, ‘8’;
  - "8" is a string containing a single character;
  - 8 is the integer 8.

Data Type: Boolean

- The Boolean data type is used for just two values: `true` and `false`
- Like characters, they only consume 1 byte of storage.
- It is worth noting that when dealing with Boolean values in C++, any number other than 0 is always interpreted as **true**.

C++ Data Type String

- A **string** is a sequence of characters enclosed in double quotes
- Sample **string** values

  `“Hello” “Year 2000” “1234”`
- The **empty** string (**null** string)contains no characters and is written as `""`

More About Type String

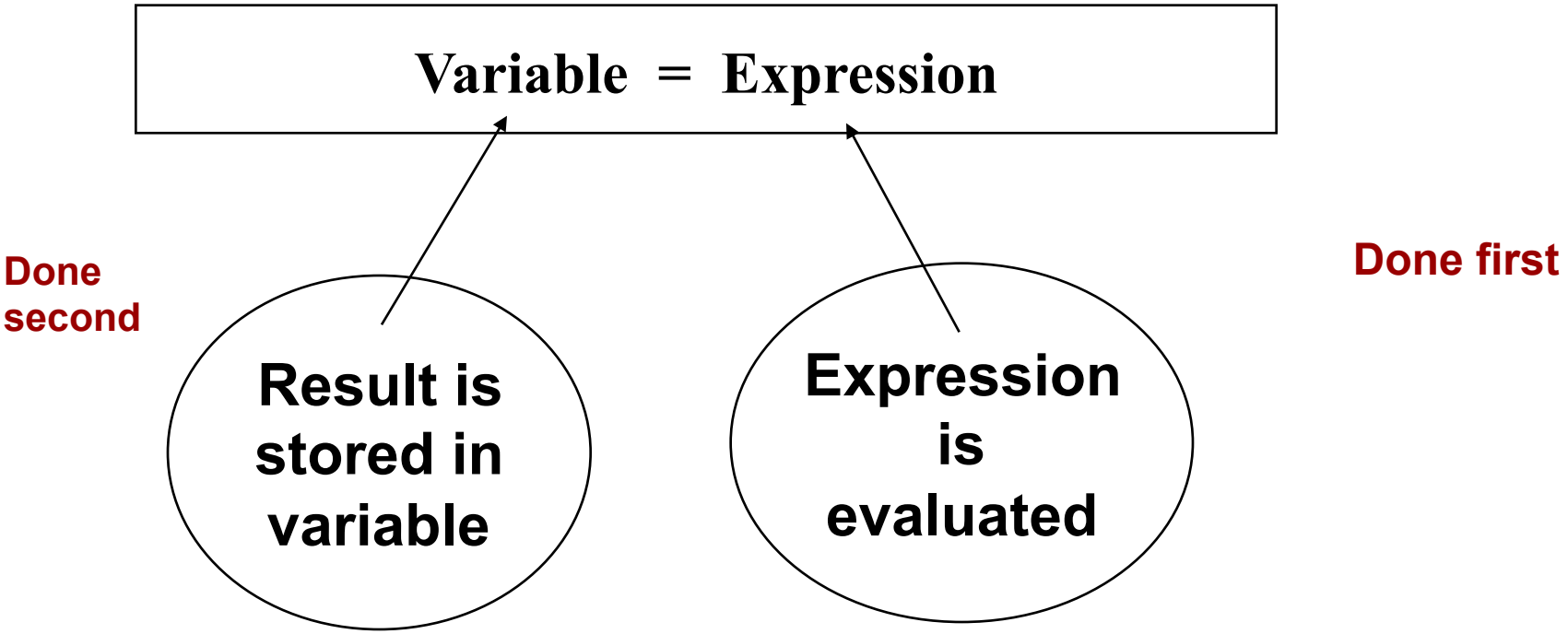
- A string is not a built-in(standard)type
  - It is a programmer-defined data type
  - It is provided in the C++ standard library
- String operations include
  - Comparing 2 string values
  - Searching a string for a particular character
  - Joining one string to another

What is an Expression in C++?

- An **expression** is a valid arrangement of variables, constants, and operators
- In C++ each expression can be evaluated to compute a value of a given type
- The value of the expression

  9 + 5 is **14**

Assignment Operator Syntax



String Concatenation(+)

- **Concatenation** is a binary operation that uses the + operator
- At least **one of the operands of the + operator** must be a **string variable** or **named string constant**--the other operand can be a string literal or a char variable, literal, or constant

Concatenation Example

```
const   string WHEN = "Tomorrow";
 const   char  EXCLAMATION = '!';
 string   message1;
 string  message2;

 message1 = "Yesterday ";
 message2 = "and ";
 message1 = message1 + message2 +
              WHEN + EXCLAMATION;
```

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 two 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

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;
```

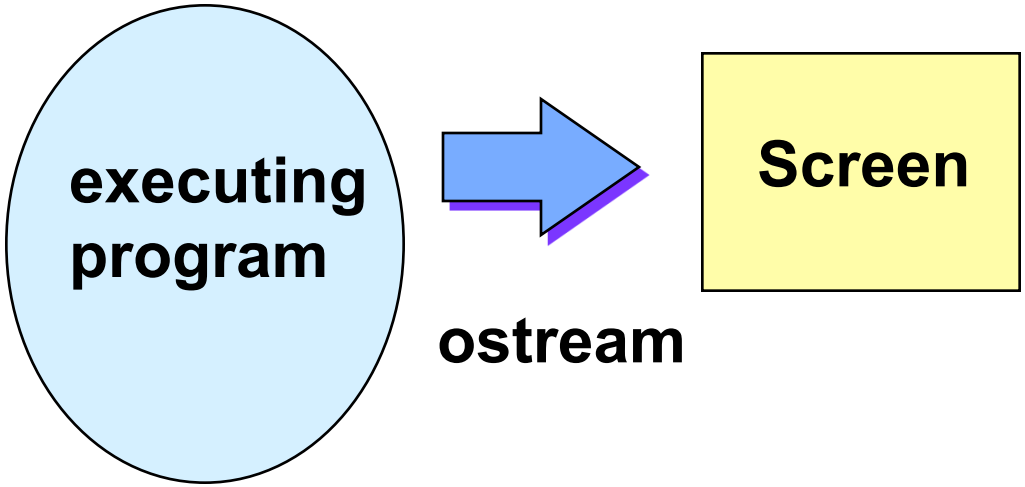
Is compilation the first step?

- No; before your source program is compiled, it is first examined by the C++ 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

No I/O is built into C++

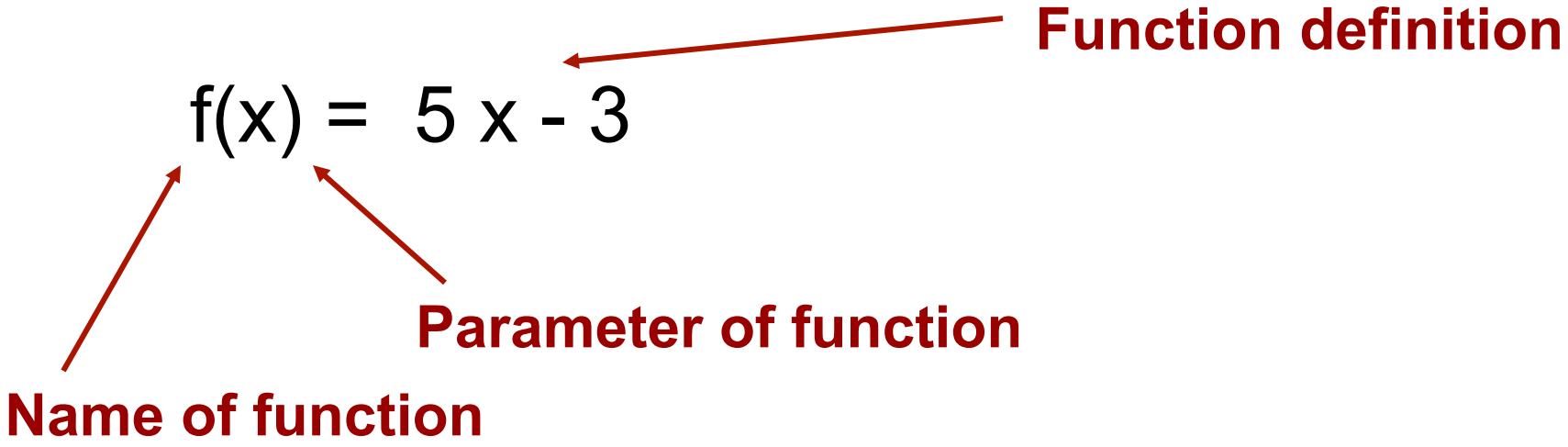
- Instead, a library provides an output stream



Using Libraries

- A library has two 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.

Function Concept in Math



When x = 1, f(x)= 2 is the returned value

When x = 4, f(x)= 17 is the returned value

Returned value is determined by the function definition and by the values of any parameters