CSE 1062 **Fundamentals of Programming**

Lecture #3

Spring 2015

Computer Science & Engineering Program
The School of EE & Computing
Adama Science & Technology University

Basics of C++ Programming

- Modular Program
- The Main Function
- Identifiers
- Program Output using cout
- Data Types
- Arithmetic Operations
- Variables
- Assignment Operations

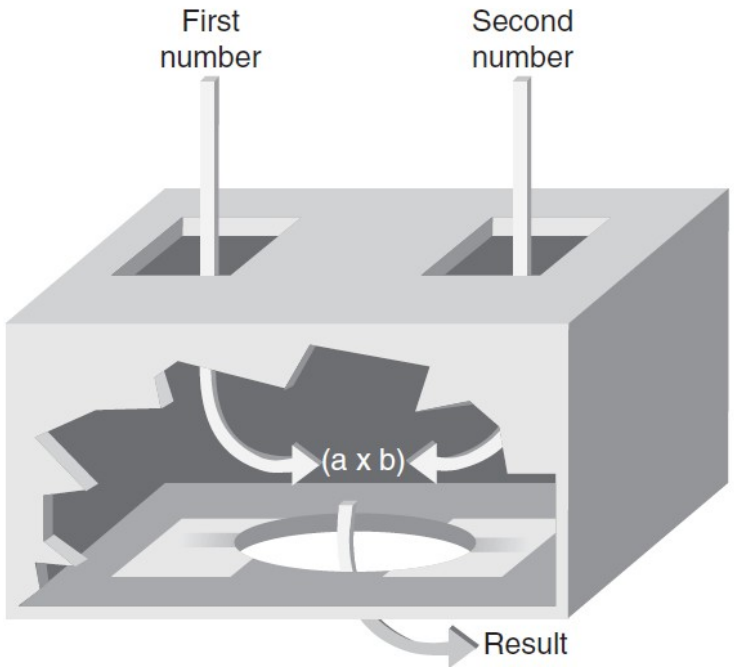
Case study: Radar Speed Trap, Heat Transfer

Reading assignment

- Chapter 2 of the textbook
- Section 3.1 from Chapter 3 of the textbook

- **Modular program:** A program consisting of interrelated segments (or **modules)** a rranged in a logical and understandable form
  - Easy to develop, correct, and modify
- Modules in C++ can be classes or fu nctions

- **Function:** Accepts an input, processes t he input, and produces an output
  - A function's processing is encapsulated and hidden within the function



- **Class:** Contains both data and functions used to manipulate the data
- **Identifier:** A name given to an element of the language, such as a class or function
  - Rules for forming identifier names:
    - First character must be a letter or underscore
    - Only letters, digits, or underscores may follow the initial letter (no blanks allowed)
    - Keywords cannot be used as identifiers
    - Maximum length of an identifier = 1024 characters

- **Keyword**: A reserved name that represen ts a built-in object or function of the lan guage

auto	delete	goto	public	this
break	do	if	register	template
case	double	inline	return	typedef
catch	else	int	short	union
char	enum	long	signed	unsigned
class	extern	new	sizeof	virtual
const	float	overload	static	void
continue	for	private	struct	volatile
default	friend	protected	switch	while

- Examples of valid C++ identifiers:

```
degToRad  intersect   addNums
slope     bessell     multTwo
findMax     density
```

  - Examples of invalid C++ identifiers:

1AB3 (begins with a number)

E*6 (contains a special character)

while (this is a keyword)

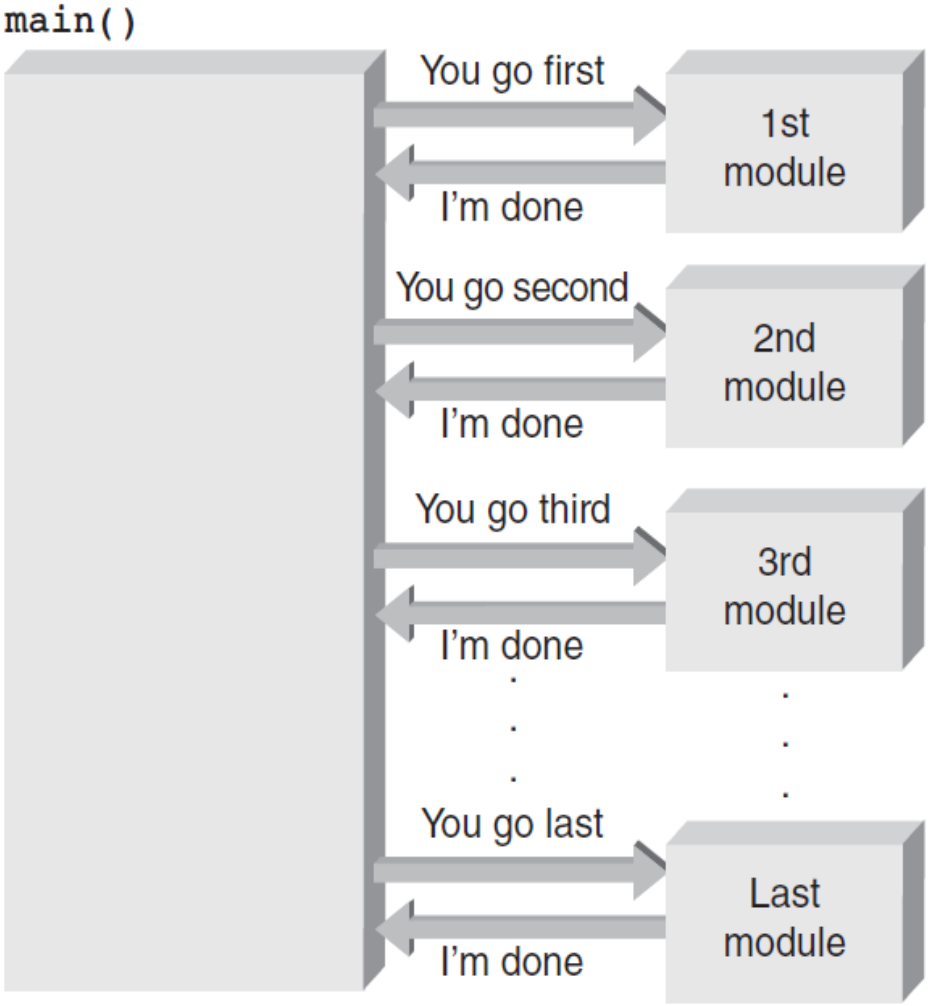
- Function names
  - Require a set of parentheses at the end
  - Can use mixed upper and lower case
  - Should be meaningful, or be a **mnemonic**
- Examples of function names:

  `easy()  c3po()  r2d2()  theForce()`
  - Note that C++ is a case-sensitive language!

The `main()` Function

ASTU

- Overall structure of a C++ program contains one function named `main()`, called the **driver function**
- All other functions are invoked from `main()`



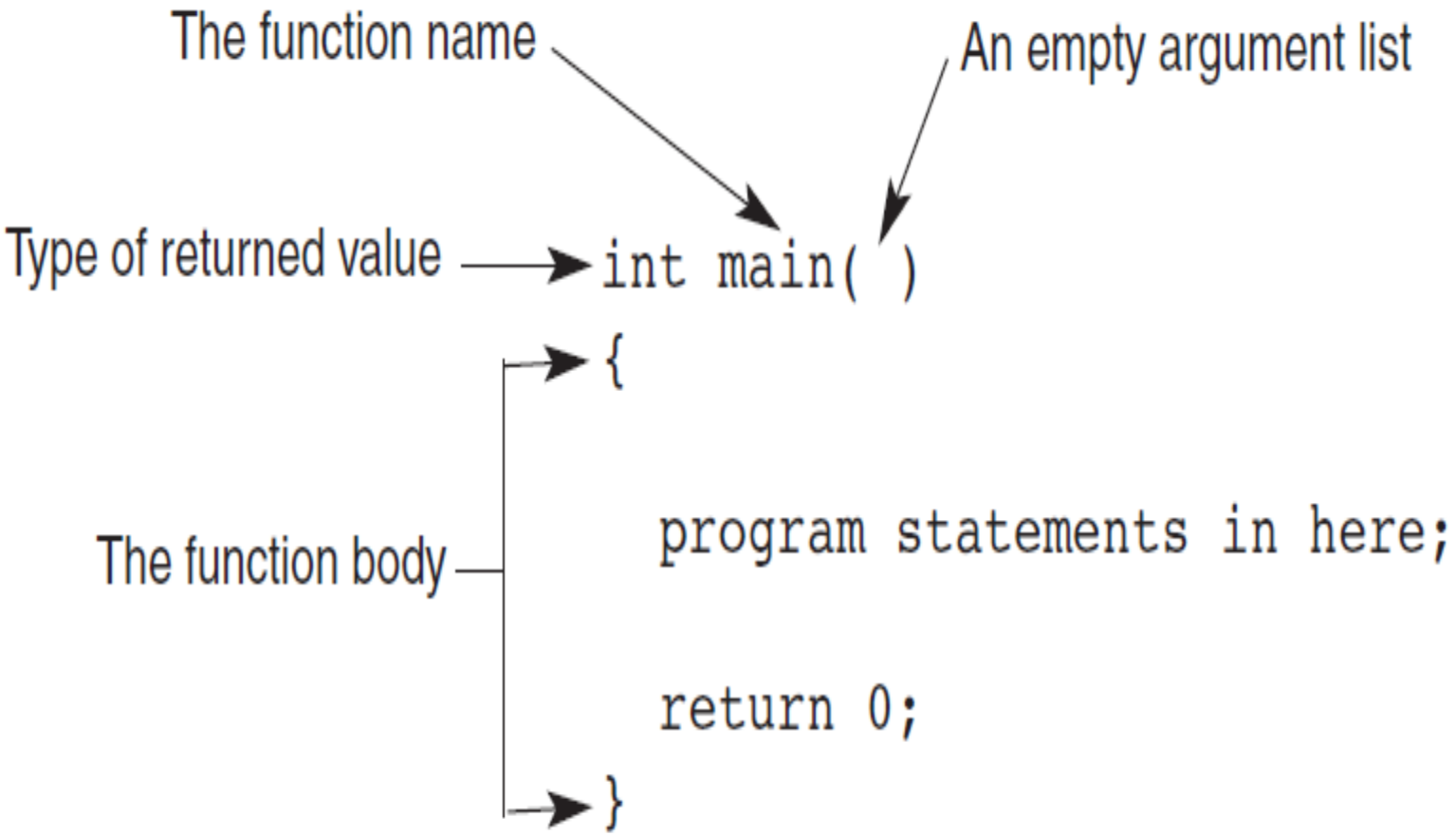
Defining a Function

- **Function header line:** First line of a function, which contains:
  - The type of data returned by the function (if any)
  - The name of the function
  - The parameter declarations to receive data that must be passed into the function when it is invoked (if any)
- **Arguments:** The data passed into a function
- **Function body:** The statements inside a function
  - enclosed in braces

Defining a Function

- Each statement inside the function must be terminated with a semicolon
- **`return`**: A keyword causing the appropriate value to be returned from the function
- The statement `return 0` in the `main()` function causes the program to end

The `main()` Function Definition



The First program

ASTU

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

int main()
{
   cout << "Hello there world!";

   return 0;
}
```

- **cout** object: An output object that send s data to a standard output display devic e

- **Preprocessor command**: Starts with a #
  - Causes an action before the source code is compiled into machine code
- **#include <file name>**
  - Causes the named file to be inserted into t he source code
  - Example: **#include<iostream>**
  - C++ provides a standard library with many p re-written classes that can be included
  - They are also called **Header files**

- **`using namespace <namespace name>`** I ndicates where header file is located
  - Namespaces qualify a name
    - A function name in your class can be the same as on e used in a standard library class
- **String:** Any combination of letters, numbers, and special characters enclosed in double qu otes
- **Delimiter:** A symbol that marks the beginning and ending of a string; not part of the string

Another Example

ASTU

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

int main()
{
   cout << "Computers, computers everywhere";
   cout << "\n as far as I can C";

   return 0;
}
```

Escape sequence

ASTU

- **Escape sequence:** One or more characters preceded by a backslash, ****
  - **Examples: \n , \t, **

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

int main()
{
   cout << "Computers everywheren\n as far as\n\nI can see";

   return 0;
}
```

Comments

ASTU

- **Comments:** Explanatory remarks in the source code ad ded by the programmer.
  - They are Code documentations
- **Line comment:** Begins with **//** and continues to the e nd of the line
  - Example:

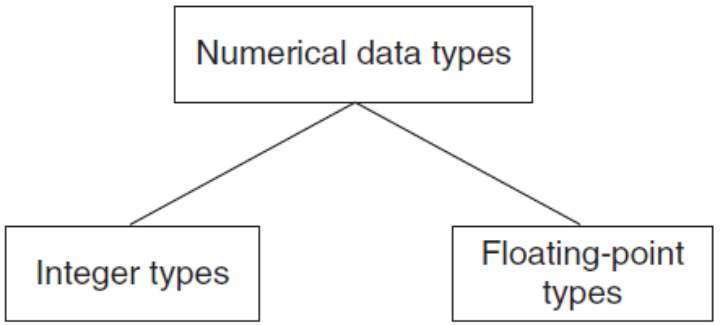
```
// this program displays a message
    #include <iostream>
    using namespace std;

     int main ()
     {
       cout << “Hello there world!”; //displays text
         return 0;
     }
```

- **Block comments:** comments that span a cross two or more lines
  - Begin with **/*** and end with ***/**
  - Example:

```
/* This is a block comment that
spans
across three lines */
```

- **Data type:** A set of values and the operat ions that can be applied to these values
- Two fundamental C++ data groupings:
  - **Class data type** (a class)
    - Created by the programmer
  - **Built-in data type** (primitive type)
    - Part of the C++ compiler



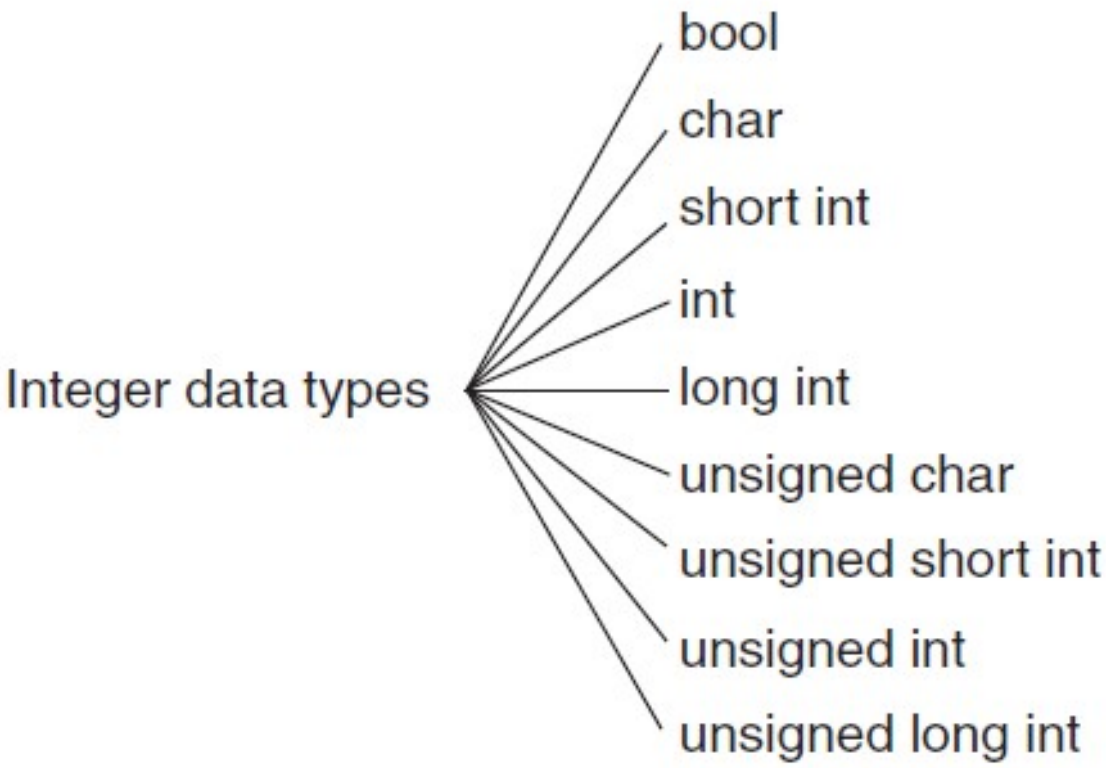
- **Literal (constant):** An actual value
  - Examples:

```
3.6      //numeric literal
“Hello” //string literal
```

  - **Integer:** A whole number
  - C++ has nine built-in integer data types
    - Each provides different amounts of storage (compiler dependent)

Integer Data type

ASTU



- **`int`** data type: Whole numbers (integers), optionally with plus (**+**) or minus (**–**) sign
  - Example: **`2, -5`**
- **`char`** data type: Individual character; any letter, digit, or special character enclosed in single quotes
  - Example: **`'A'`**
  - Character values are usually stored in **ASCII code**

ASCII TABLE

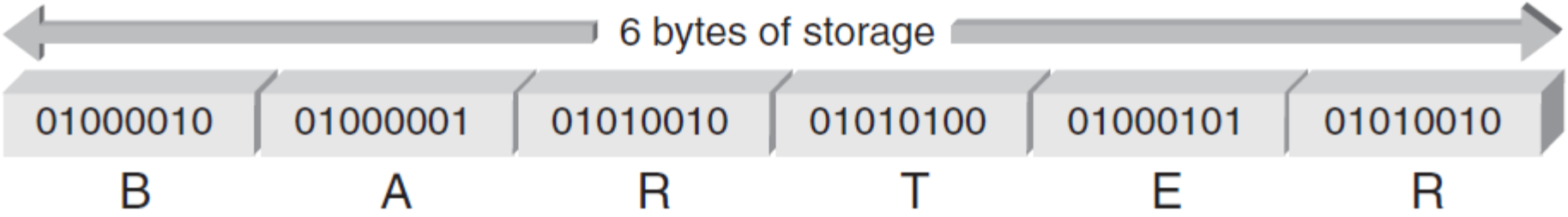
ASTU

Letter	ASCII Code	Letter	ASCII Code
A	01000001	N	01001111
B	01000010	O	01001110
C	01000011	P	01010000
D	01000100	Q	01010001
E	01000101	R	01010010
F	01000110	S	01010011
G	01000111	T	01010100
H	01001000	U	01010101
I	01001001	V	01010110
J	01001010	W	01010111
K	01001011	X	01011000
L	01001100	Y	01011001
M	01001101	Z	01011010

ASCII Code

ASTU

- When storing the ASCII codes to represe nt text, each letter takes one byte of me mory and is represented by the associate d number from the chart



- **`bool`** data type: Represents Boolean (logical) data
  - Restricted to two values: true or false
  - Useful when a program must examine a condition and take a prescribed course of action, based on whether the condition is true or false

Signed and Unsigned Data Types

ASTU

- **Signed data type:** One that permits negative, positive, and zero values
- **Unsigned data type:** Permits only positive and zero values
  - An unsigned data type provides essentially double the range of its signed counterpart

Name of Data Type	Storage Size	Range of Values
`char`	1	256 characters
`bool`	1	true (considered as any positive value) and false (which is a 0)
`short int`	2	-32,768 to +32,767
`unsigned short int`	2	0 to 65,535
`int`	4	-2,147,483,648 to +2,147,483,647
`unsigned int`	4	0 to 4,294,967,295
`long int`	4	-2,147,483,648 to +2,147,483,647
`unsigned long int`	4	0 to 4,294,967,295

Floating-Point Types

- **Floating-point number** (real number): Zero or any positive or negative number containing a decimal point
  - Examples: `+10.625 5. -6.2`
  - No special characters are allowed
  - Three floating-point data types in C++:
    - `float` (single precision)
    - `double` (double precision)
    - `long double`

Type	Storage	Absolute Range of Values (+ and -)
`float`	4 bytes	$1.4012984643248 1707 \times 10^{-45}$ to $3.4028234663852886 \times 10^{38}$
`double` and `long double`	8 bytes	$4.9406564584124654 \times 10^{-324}$ to $1.7976931348623157 \times 10^{308}$

Floating-Point literal

ASTU

- `float literal`: Append an `f` or `F` to t he number
- `long double literal`: Append an `l` o r `L` to the number
  - Examples:

```
9.234   // a double literal
9.234F  // a float literal
9.234L  // a long double literal
```

Arithmetic Operations

ASTU

- C++ supports addition, subtraction, multiplic ation, division, and modulus division operati ons.
- Different data types can be used in the same arithmetic expression
- Arithmetic operators are binary operators
  - **Binary operators:** Require two operands
  - **Unary operator:** Requires only one operand
  - **Negation operator (-):** Reverses the sign of the number

Arithmetic Operations

ASTU

Operation	Operator
Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulus division	%

Arithmetic Operations

ASTU

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

int main()
{
  cout << "15.0 plus 2.0 equals "          << (15.0 + 2.0) << endl
       << "15.0 minus 2.0 equals "         << (15.0 - 2.0) << endl
       << "15.0 times 2.0 equals "         << (15.0 * 2.0) << endl
       << "15.0 divided by 2.0 equals "    << (15.0 / 2.0) << endl;

  return 0;
}
```

- **Expression:** Any combination of operators and operands that can be evaluated to yield a value
- If all operands are the same data type, the expression is named by the data type used (integer expression, floating-point expression, etc.)
- **Mixed-mode expression:** Contains integer and floating-point operands
  - Yields a double-precision value

Integer Division

- Integer division: Yields an integer result
  - Any fractional remainders are dropped (truncated)
  - Example: `15/2` yields `7`
- Modulus (remainder) operator: Returns only the remainder
  - Example: `9 % 4` yields `1`

Operator Precedence

ASTU

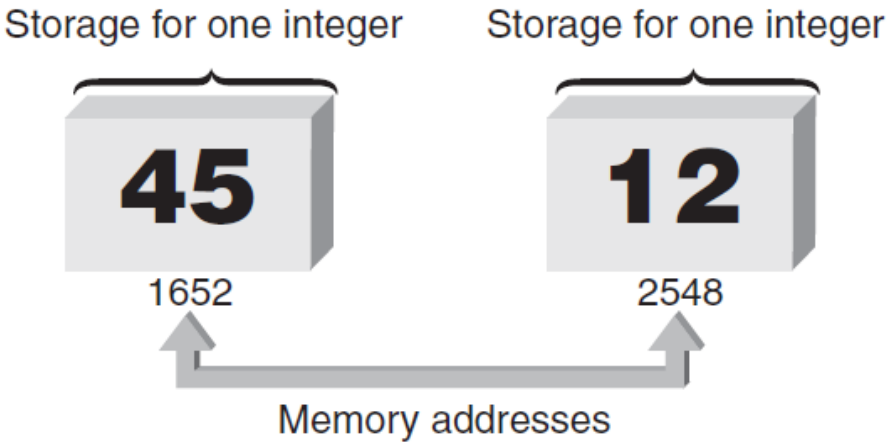
- Expressions with multiple operators are evaluated by precedence of operators:
  - All negations occur first
  - Multiplication, division, and modulus are ne xt, from left to right
  - Addition and subtraction are last, from left to right

Operator	Associativity
Unary –	Right to left
* / %	Left to right
+ –	Left to right

Variables and Declaration Statements

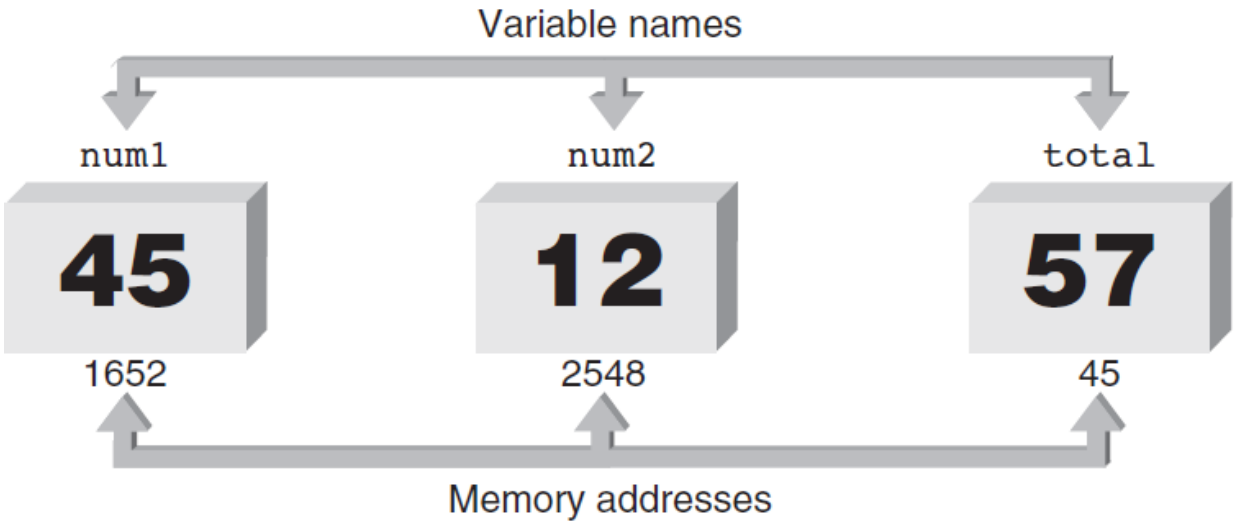
ASTU

- **Variable:** All integer, float-point, and oth er values used in a program are stored a nd retrieved from the computer's memo ry
- Each memory location has a unique addr ess



ASTU

- **Variable:** Symbolic identifier for a memor y address where data can be held
- Use identifier naming rules for variable n ames



- **Assignment statement:** Used to store a v alue into a variable
- Value of the expression on the right is as signed to the memory location of the var iable on the left side
  - Examples:

```
num1 = 45;
num2 = 12;
total = num1 + num2;
```

- **Declaration statement:** Specifies the dat a type and identifier of a variable; sets u p the memory location
  - Syntax: *`dataType variableName;`*
- Data type is any valid C++ data type
  - Example: `int sum;`
- Declarations may be used anywhere in a function
  - Usually grouped at the opening brace

Variables and Declaration Statements

- Multiple variables of the same data type can be declared in a single declaration statemen t
  - Example:

```
double grade1, grade2, total, average;
```

- Variables can be initialized in a declaration
  - Example:

```
double grade1 = 87.0
```

- **A variable must be declared before it is used**

Variables and Declaration Statements

ASTU

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

int main()
{
   double grade1 = 85.5;
   double grade2 = 97.0;
   double total, average;

   total = grade1 + grade2;
   average = total/2.0; // divide the total by 2.0
   cout << "The average grade is " << average << endl;

   return 0;
}
```

- **Assignment Statement:** Assigns the value of the expression on the right side of the = to the variable on the left side of the =
- Another assignment statement using the same variable will overwrite the previous value with the new value

  Examples:

```
slope = 3.7;
slope = 6.28;
```

Assignment Operations

- Right side of an assignment statement may contain any expression that can be evaluated to a value

  Examples:

```
newtotal = 18.3 + total;
taxes = .06*amount;
average = sum / items;
```

  - Only one variable can be on the left side of an assignment statement

Assignment Operations

ASTU

```
// This program calculates the volume of a cylinder,
// given its radius and height
#include <iostream>
using namespace std;

int main()
{
   double radius, height, volume;
   radius = 2.5;
   height = 16.0;
   volume = 3.1416 * radius * radius * height;
   cout << "The volume of the cylinder is " << volume << endl;

   return 0;
}
```

Assignment Operations

ASTU

- **Assignment operator:** The = sign
- **C++ statement:** Any expression terminated by a semicolon
- **Accumulation statement:** Has the effect of accumulating, or totaling
- Syntax:

```
variable = variable + newValue;
```

Assignment Operations

- Additional assignment operators provide short cut s: **+=**, **-=**, ***=**, **/=**, **%=**

  Example:

```
sum = sum + 10;
```

  is equivalent to: **sum += 10;**

```
price *= rate +1;
```

  is equivalent to:

```
price = price * (rate + 1);
```

Assignment Operations

ASTU

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

int main()
{
   int sum;
   sum = 0;
   cout << "The value of sum is initially set to " << sum << endl;
   sum = sum + 96;
   cout << "  sum is now " << sum << endl;
   sum = sum + 70;
   cout << "  sum is now " << sum << endl;
   sum = sum + 85;
   cout << "  sum is now " << sum << endl;
   sum = sum + 60;
   cout << "  The final sum is " << sum << endl;

   return 0;
}
```

Assignment Operations

- **Increment operator ++:** Unary operator for the special case when a variable is increased by 1
- **Prefix increment operator** appears before the variable
  - Example: **`++i`**
- **Postfix increment operator** appears after the variable
  - Example: **`i++`**

Assignment Operations

- Example: **`k = ++n; //prefix increment`**
  is equivalent to:

```
n = n + 1;  //increment n first
k = n;    //assign n's value to k
```

- Example: **`k = n++; //postfix increment`**
  is equivalent to

```
k = n;  //assign n's value to k
 n = n + 1; //and then increment n
```

Assignment Operations

- **Decrement operator `--`:** Unary operator for the special case when a variable is decreased by 1
- **Prefix decrement operator** appears before the variable
  - Example: **`--i;`**
- **Postfix decrement operator** appears after the variable
  - Example: **`i--;`**

Case Study: Radar Speed Trap

ASTU

Case Study: Radar Speed Trap

ASTU

- A highway-patrol speed-detection radar emits a beam of microwaves at a frequency designated as *fe*
- The beam is reflected off an approaching car, and the radar unit picks up and analyzes the reflected beam's frequency, *fr*
- The reflected beam's frequency is shifted slightly from *fe* to *fr* because of the car's motion.

- The relationship between the speed of the car, **v**, in miles per hour (mph), and the two microwave frequencies is

$$v = (6.685\text{x}10^{8})\left(\frac{f_r - f_e}{f_r + f_e}\right) mph$$

where the emitted waves have a frequency of

$f_e = 2 \times 10^{10}\ \text{sec}^{-1}$

- we will write a C ++ program
  - using the software development procedure
  - calculate and display the speed correspondi ng to a received frequency of 2.000004 × 1 010 sec-1

- Step 1: Analyze the Problem
  - Understand the desired outputs
  - Determine the required inputs
- Step 2: Develop a Solution
  - Determine the algorithms to be used
  - Use **top-down approach** to design
- Step 3: Code the Solution
- Step 4: Test and Correct the Program

- Analyze the Problem
  - Output: Speed of the car
  - Inputs: Emitted frequency and received freq uency
- Develop a Solution
  - Algorithm:
    - Assign values to fe and fr
    - Calculate and display speed

ASTU

- Code the Solution

// 2e10 is programming notation for 2 times 10 to 10th power

// = 20,000,000,000 or 20 billion

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main()
6   {
7       double speed, fe, fr;
8       fe=2e10;
9       fr=2.0000004e10;
10      speed=6.685e8*(fr-fe)/(fr+fe);
11      cout << "The speed is " <<speed<<"  miles/hour"<< endl;
12      return 0;
13  }
14
```

- Test and Correct the Program
  - Verify that the calculation and displayed val ue agree with the previous hand calculation
  - Use the program with different values of re ceived frequencies
- Ethiopia uses metric system therefore W hat if we want the answer to be in Km/h r.