

02: Elementary Programming

Programming Technique I
(SECJ1013)

What a Is a Program Made Of?

- Common elements in programming languages:
 - Key Words
 - Programmer-Defined Identifiers
 - Operators
 - Punctuation
 - Syntax

Key Words

- Also known as **reserved words**
- Have a special meaning in C++
- Can not be used as identifier
- Written using lowercase letters
- Examples in program (shown in green):

```
using namespace std;  
int main()
```

Example Program

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

int main()
{
    double num1 = 5,
           num2, sum;
    num2 = 12;

    sum = num1 + num2;
    cout << "The sum is " << sum;
    return 0;
}
```

Operators

- Used to perform operations on data
- Many types of operators
 - Arithmetic: $+$, $-$, $*$, $/$
 - Assignment: $=$
- Examples in program (shown in green):
`num2 = 12;`
`sum = num1 + num2;`

Example Program

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

int main()
{
    double num1 = 5, num2, sum;
    num2 = 12;

    sum = num1 + num2;
    cout << "The sum is " << sum;
    return 0;
}
```

Punctuation

- Characters that mark the end of a statement, or that separate items in a list
- Example in program (shown in green):

```
double num1 = 5,  
      num2, sum;  
num2 = 12;
```

Example Program

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

int main()
{
    double num1 = 5,
           num2, sum;
    num2 = 12;

    sum = num1 + num2;
    cout << "The sum is " << sum;
    return 0;
}
```


The `#include` Directive

- Inserts the contents of another file into the program
- Is a preprocessor directive
 - Not part of the C++ language
 - Not seen by compiler

- Example:

```
#include <iostream>
```



No ; goes here

Comments

- Are used to document parts of a program
- Are written for persons reading the source code of the program
 - Indicate the purpose of the program
 - Describe the use of variables
 - Explain complex sections of code
- Are ignored by the compiler

Single-Line Comments

- Begin with `//` through to the end of line

```
int length = 12; // length in inches
int width = 15;  // width in inches
int area;        // calculated area

// Calculate rectangle area
area = length * width;
```

Multi-Line Comments

- Begin with `/*` and end with `*/`
- Can span multiple lines

```
/*-----  
    Here's a multi-line comment  
-----*/
```

- Can also be used as single-line comments

```
int area;    /* Calculated area */
```

The Parts of a C++ Program

Statement	Purpose
<code>// sample C++ program</code>	comment
<code>#include <iostream></code>	preprocessor directive
<code>using namespace std;</code>	which namespace to use
<code>int main()</code>	beginning of function named <code>main</code>
<code>{</code>	beginning of block for <code>main</code>
<code> cout << "Hello, there!";</code>	output statement
<code> return 0;</code>	send 0 back to the operating system
<code>}</code>	end of block for <code>main</code>

Special Characters

Character	Name	Description
//	Double Slash	Begins a comment
#	Pound Sign	Begins preprocessor directive
< >	Open, Close Brackets	Encloses filename used in <code>#include</code> directive
()	Open, Close Parentheses	Used when naming function
{ }	Open, Close Braces	Encloses a group of statements
" "	Open, Close Quote Marks	Encloses string of characters
;	Semicolon	Ends a programming statement

Important Details

- C++ is case-sensitive. Uppercase and lowercase characters are different characters. 'Main' is not the same as 'main'.
- Every { must have a corresponding }, and vice-versa.

Variables

Variables

- A variable is a named location in computer memory (in RAM)
- It holds a piece of data
- It must be *defined* before it can be used
- Example variable definition:

```
double num1;
```

Example Program

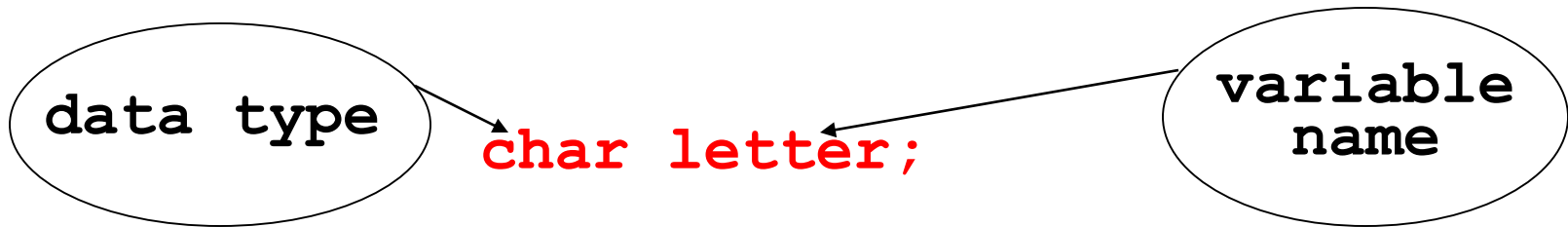
```
#include <iostream>
using namespace std;

int main()
{
    double num1 = 5,
           num2, sum;
    num2 = 12;

    sum = num1 + num2;
    cout << "The sum is " << sum;
    return 0;
}
```

Variables, Constants, and the Assignment Statement

- Variable
 - Has a name and a type of data it can hold



- Is used to reference a location in memory where a value can be stored
- Must be defined before it can be used
- The value that is stored can be changed, *i.e.*, it can “vary”

Variables

- If a new value is stored in the variable, it replaces the previous value
- The previous value is overwritten and can no longer be retrieved


```
int age;  
age = 17;           // age is 17  
cout << age;        // Displays 17  
age = 18;           // Now age is 18  
cout << age;        // Displays 18
```

Variables: Example

Program 2-7

```
1  // This program has a variable.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      int number;
8
9      number = 5;
10     cout << "The value in number is " << number << endl;
11     return 0;
12 }
```

Variable Definition



Program Output

The value in number is 5

Identifiers

Identifiers

- Programmer-chosen names to represent parts of the program, such as variables
- Name should indicate the use of the identifier
- Cannot use C++ key words as identifiers
- Must begin with alphabetic character or `_`, followed by alphabetic, numeric, or `_`. Alpha may be uppercase or lowercase
- Example in program (shown in green):
`double num1;`

Example Program

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

int main()
{
    double num1 = 5,
           num2, sum;
    num2 = 12;

    sum = num1 + num2;
    cout << "The sum is " << sum;
    return 0;
}
```


Valid and Invalid Identifiers

IDENTIFIER	VALID?	REASON IF INVALID
<code>totalSales</code>	●	
<code>total_Sales</code>	●	
<code>total.Sales</code>	●	
<code>4thQtrSales</code>	●	
<code>totalSale\$</code>	●	

Lines vs. Statements

In a source file,

A **line** is all of the characters entered before a carriage return.

Blank lines improve the readability of a program.

Here are four sample lines. Line 3 is blank:

```
double num1 = 5, num2, sum;  
num2 = 12;  
  
sum = num1 + num2;
```

Lines vs. Statements

In a source file,

A **statement** is an instruction to the computer to perform an action.

A statement may contain keywords, operators, programmer-defined identifiers, and punctuation.

A statement may fit on one line, or it may occupy multiple lines.

Here is a single statement that uses two lines:

```
double num1 = 5,  
       num2, sum;
```

Literals


- Literal: a value that is written into a program's code.
 - "hello, there" (string literal)
 - 12 (integer literal)

Literals: Example

Program 2-9

```
1 // This program has literals and a variable.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     int apples;
8
9     apples = 20;
10    cout << "Today we sold " << apples << " bushels of apples.\n";
11    return 0;
12 }
```

20 is an integer literal



Program Output

Today we sold 20 bushels of apples.

Literals: Example

Program 2-9

```
1 // This program has literals and a variable.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     int apples;
8
9     apples = 20;
10    cout << "Today we sold " << apples << " bushels of apples.\n";
11    return 0;
12 }
```

This is a string literal

Program Output

Today we sold 20 bushels of apples.

In-Class Exercise

Examine the following program. List all the **variables** and **literals** that appear in the program.

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

```
int main()
{   int little;
    int big;

    little = 2;
    big = 2000;
    cout<<"The little number is " <<little<<endl;
    cout<<"The big number is "<<big<<endl;
    return 0;
}
```

In-Class Exercise

What will the following program display on the screen?

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

The value is 712

```
int main()
{
    int num;
    num = 712;
    cout<< "The value is " << num << endl;
    return 0;
}
```


Input and Output

Input using cin

The **cin** Object

- Standard input object
- Like **cout**, requires **iostream** file
- Used to read input from keyboard
- Information retrieved from **cin** with **>>**
- Input is stored in one or more variables

Program 3-1

```
1  // This program asks the user to enter the length and width of
2  // a rectangle. It calculates the rectangle's area and displays
3  // the value on the screen.
4  #include <iostream>
5  using namespace std;
6
7  int main()
8  {
9      int length, width, area;
10
11      cout << "This program calculates the area of a ";
12      cout << "rectangle.\n";
13      cout << "What is the length of the rectangle? ";
14      cin >> length;
15      cout << "What is the width of the rectangle? ";
16      cin >> width;
17      area = length * width;
18      cout << "The area of the rectangle is " << area << ".\n";
19      return 0;
20 }
```

Program Output with Example Input Shown in Bold

This program calculates the area of a rectangle.
What is the length of the rectangle? **10 [Enter]**
What is the width of the rectangle? **20 [Enter]**
The area of the rectangle is 200.

The `cin` Object

- `cin` converts data to the type that matches the variable:

```
int height;  
cout << "How tall is the room? ";  
cin >> height;
```

The `cin` Object

- Can be used to input more than one value:

```
cin >> height >> width;
```

- Multiple values from keyboard must be separated by spaces
- Order is important: first value entered goes to first variable, etc.

Displaying a Prompt

- A prompt is a message that instructs the user to enter data.
- You should always use **cout** to display a prompt before each cin statement.

```
cout << "How high is the room? ";  
cin >> height;
```

Program 3-2

```
1  // This program asks the user to enter the length and width of
2  // a rectangle. It calculates the rectangle's area and displays
3  // the value on the screen.
4  #include <iostream>
5  using namespace std;
6
7  int main()
8  {
9      int length, width, area;
10
11      cout << "This program calculates the area of a ";
12      cout << "rectangle.\n";
13      cout << "Enter the length and width of the rectangle ";
14      cout << "separated by a space.\n";
15      cin >> length >> width;
16      area = length * width;
17      cout << "The area of the rectangle is " << area << endl;
18      return 0;
19 }
```

Program Output with Example Input Shown in Bold

This program calculates the area of a rectangle.

Enter the length and width of the rectangle separated by a space.

10 20 [Enter]

The area of the rectangle is 200

Reading Strings with `cin`

- Can be used to read in a string
- Must first declare an array to hold characters in string:

```
char myName[21];
```

- `myName` is a name of an array, 21 is the number of characters that can be stored (the size of the array), including the NULL character at the end
- Can be used with `cin` to assign a value:

```
cin >> myName;
```

Program 3-4

```
1  // This program demonstrates how cin can read a string into
2  // a character array.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8      char name[21];
9
10     cout << "What is your name? ";
11     cin >> name;
12     cout << "Good morning " << name << endl;
13     return 0;
14 }
```

Program Output with Example Input Shown in Bold

What is your name? **Charlie** [Enter]
Good morning Charlie

In-Class Exercise

- Solve the problem. Add array of characters to the output.

Sample of output:

Enter an integer: 7

Enter a decimal number : 2.25

Enter a single character : R

Enter an array of characters: Programming

Output using cout

The `cout` Object

- Displays information on computer screen
- Use `<<` to send information to `cout`

```
cout << "Hello, there!";
```

- Can use `<<` to send multiple items to `cout`

```
cout << "Hello, " << "there!";
```

Or

```
cout << "Hello, ";
```

```
cout << "there!";
```

Starting a New Line


- To get multiple lines of output on screen

- Use **endl**

```
cout << "Hello, there!" << endl;
```

- Use **\n** in an output string

```
cout << "Hello, there!\n";
```



Notice that the **\n** is **INSIDE**
the string.

In-Class Exercise

- Rearrange the following program statements in the correct order.

```
int main()  
{  
    return 0;  
#include <iostream>  
    cout<<"In 1492 Columbus sailed the ocean  
        blue.";  
}  
using namespace std;
```

- What is the output of the program when it is properly arranged?

Data type and constant

Number Systems

- Numbers can be represented in a variety of ways.
- The representation depends on what is called the BASE.
- You write these numbers as:
 - **Number**_{base}

Number Systems

- The following are the four most common representations.
- Decimal (base 10)
 - Commonly used
 - Valid digits are from 0 to 9
 - Example: 126₁₀ (normally written as just 126)
- Binary (base 2)
 - Valid digits are 0 and 1
 - Example: 1111110₂

- The following are the four most common representations.
- Octal (base 8)
 - Valid digits are from 0 to 7
 - Example: 1768
- Hexadecimal (base 16)
 - Valid digits are from 0 to 9 and A to F (or from a to f)
 - Example: 7E16

Integer Data Types

- Designed to hold whole numbers
- Can be **signed** or **unsigned**

12 -6 +3

- Available in different sizes (*i.e.*, number of bytes): **short**, **int**, and **long**
- Size of **short** \leq size of **int** \leq size of **long**

Integral Constants

- To store an integer constant in a long memory location, put 'L' at the end of the number:

1234L

- Constants that begin with '0' (zero) are octal, or base 8: 075
- Constants that begin with '0x' are hexadecimal, or base 16: 0x75A

Defining Variables

- Variables of the same type can be defined
 - In separate statements

```
int length;  
int width;
```

- In the same statement

```
int length,  
    width;
```

- Variables of different types must be defined in separate statements

Floating-Point Data Types

- Designed to hold real numbers

12.45 -3.8

- Stored in a form similar to scientific notation
- Numbers are all signed
- 3 data types to represent floating-point numbers: **float**, **double**, and **long double**
- Size of **float** \leq size of **double**
 \leq size of **long double**

Floating-point Constants

- Can be represented in
 - Fixed point (decimal) notation:
31.4159 **0.0000625** double
 - E notation:
3.14159E1 **6.25e-5**
- Are **double** by default
- Can be forced to be float **3.14159F** or long double **0.0000625L**

Assigning Floating-point Values to Integer Variables

If a floating-point value is assigned to an integer variable

- The fractional part will be truncated (*i.e.*, “chopped off” and discarded)
- The value is not rounded

```
int rainfall = 3.88;
```

only store 3 because use int

```
cout << rainfall; // Displays 3
```

The `bool` Data Type

- Represents values that are **true** or **false**
- `bool` values are stored as small integers
- **false** is represented by 0, **true** by 1

```
bool allDone = true;    allDone    finished
bool finished = false;  

|   |
|---|
| 1 |
|---|



|   |
|---|
| 0 |
|---|


```

The `char` Data Type

- Used to hold single characters or very small integer values
- Usually occupies 1 byte of memory
- A numeric code representing the character is stored in memory

SOURCE CODE

MEMORY

```
char letter = 'C'; letter
```

67

ASCII

The **char** Data Type

- Used to hold single characters or very small integer values
- Usually occupies 1 byte of memory
- A numeric code representing the character is stored in memory

SOURCE CODE

MEMORY

```
char letter = 'C';    letter
```

67

In-Class Exercise

- What is wrong with the following program?

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

int main()
{
    char letter;

    letter = "z";
    cout<<letter<<endl;
    return 0;
}
```

Summary of data types

to minimize file size

Name	Description	Size	Range
char	Character or small integer.	1byte	signed: -128 to 127 unsigned: 0 to 255
short int (short)	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
int	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int (long)	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	Boolean value. It can take one of two values: true or false.	1byte	true or false
float	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	Long double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)

Naming Constant

Named Constants

- Named constant (constant variable): variable whose content cannot be changed during program execution
- Used for representing constant values with descriptive names:

```
const double TAX_RATE = 0.0675;  
const int NUM_STATES = 50;
```

- Often named in uppercase letters

Defining constants

- You can define your own names for constants that you use very often without having to resort to memory-consuming variables, simply by using the `#define` preprocessor directive.

- Its format:

```
#define identifier value
```

- Example:

```
#include <iostream>
using namespace std;
#define PI 3.14159 syntax
#define NEWLINE '\n'
int main ()
{ double r=5.0;
  double circle;
  circle = 2 * PI * r;
  cout << circle;
  cout << NEWLINE;  return 0;}
```

Declared constants (const)

- With the const prefix you can declare constants with a specific type in the same way as you would do with a variable
- Example:

```
#include <iostream>
using namespace std;
int main ()
{ double r=5.0,circle;
  const double PI = 3.14159;
  const char NEWLINE = '\\n';
  circle = 2 * PI * r;
  cout << circle;
  cout << NEWLINE; return 0; }
```

String Constant

- Can be stored a series of characters in consecutive memory locations

"Hello"

- Stored with the null terminator, \0, at end

H	e	l	l	o	\0
---	---	---	---	---	----

- Is comprised of characters between the " "

A character or a string constant?

- A character constant is a single character, enclosed in single quotes:

`'C'`

- A string constant is a sequence of characters enclosed in double quotes:

`"Hello, there!"`

- A single character in double quotes is a string constant, not a character constant:

`"C"`

The C++ `string` Class

- Must **`#include <string>`** to create and use string objects
- Can define `string` variables in programs
`string name;`
- Can assign values to string variables with the assignment operator
`name = "George";`
- Can display them with `cout`
`cout << name;`

Determining the Size of a Data Type

The **sizeof** operator gives the size of any data type or variable

```
double amount;  
  
cout << "A float is stored in "  
      << sizeof(float) << " bytes\n";  
  
cout << "Variable amount is stored in "  
      << sizeof(amount) << " bytes\n";
```

More on Variable Assignments and Initialization

- Assigning a value to a variable
 - Assigns a value to a previously created variable
 - A single variable name must appear on left side of the = symbol

```
int size;  
size = 5;      // legal  
5 = size;      // not legal
```

Variable Assignment vs. Initialization

- Initializing a variable
 - Gives an initial value to a variable at the time it is created
 - Can initialize some or all variables of definition

```
int length = 12;  
int width = 7, height = 5, area;
```


Scope

- The **scope** of a variable is that part of the program where the variable may be used
- A variable cannot be used before it is defined

```
int a;  
cin >> a;    // legal  
cin >> b;    // illegal  
int b;
```

In-Class Exercise

- Trace the following program. Can it be compiled?

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

```
int main()
```

```
{
```

```
    cout<<"value";
```

NO

```
    int value;
```



```
    return 0;
```

```
}
```

Arithmetic Expression

Arithmetic Operators and Expression

Arithmetic Operators

- Used for performing numeric calculations
- C++ has unary, binary, and ternary operators
 - unary (1 operand) -5 
 - binary (2 operands) $13 - 7$ 
 - ternary (3 operands) $\text{exp1} ? \text{exp2} : \text{exp3}$

Binary Arithmetic Operators

SYMBOL	OPERATION	EXAMPLE	ans
+	addition	<code>ans = 7 + 3;</code>	10
-	subtraction	<code>ans = 7 - 3;</code>	4
*	multiplication	<code>ans = 7 * 3;</code>	21
/	division	<code>ans = 7 / 3;</code>	2
%	modulus	<code>ans = 7 % 3;</code>	1

integer value only
to get the remainder

/ Operator

- C++ division operator (/) performs integer division if both operands are integers

```
cout << 13 / 5;      // displays 2  
cout <<  2 / 4;      // displays 0
```

- If either operand is floating-point, the result is floating-point

```
cout << 13 / 5.0;    // displays 2.6  
cout << 2.0 / 4;     // displays 0.5
```

% Operator

- C++ modulus operator (%) computes the remainder resulting from integer division

```
cout << 9 % 2;    // displays 1
```

- % requires integers for both operands

```
cout << 9 % 2.0; // error
```


In-Class Exercise

- Identify as many syntax errors as you can in the following program

```
*/ what is wrong with this program?/*  
#include iostream  
using namespace std;  
  
int main();  
{  
    int a, b, c  
    a=3  
    b=4  
    c=a+b  
    Cout<"The value of c is "<C;  
    return 0;  
}
```

Order of Operations

In an expression with more than one operator, evaluation is in this order:

()

– (unary negation), in order, right to left

* / %, in order, left to right

+ –, in order, left to right

In the expression $2 + 2 * 2 - 2$



evaluate
second

evaluate
first

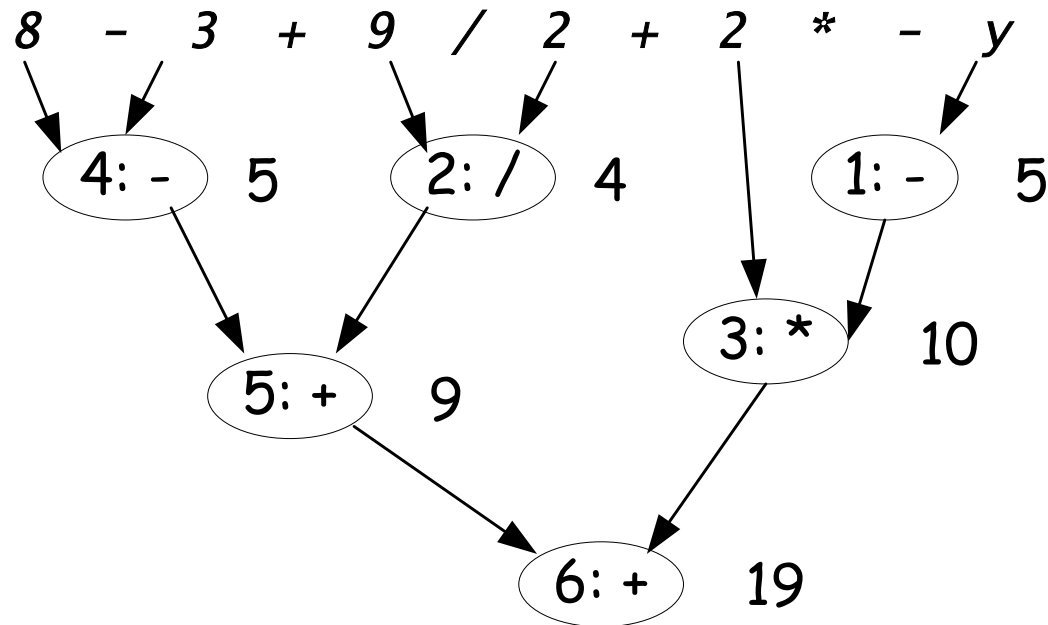
evaluate
third

Example

```
int z, y=-5;
```

```
z= 8 - 3 + 9 / 2 + 2 * - y;
```

```
z= 8 - (3 + 9 / 2) + 2 * - y; // try this
```



Order of Operations

Show prove for the following expression

Table 3-2 Some Expressions

Expression	Value
$5 + 2 * 4$	13
$10 / 2 - 3$	2
$8 + 12 * 2 - 4$	28
$4 + 17 \% 2 - 1$	4
$6 - 3 * 2 + 7 - 1$	6

Associativity of Operators

- $-$ (unary negation) associates right to left
- $*$, $/$, $\%$, $+$, $-$ associate left to right
- parentheses () can be used to override the order of operations:

$$2 + 2 * 2 - 2 = 4$$

$$(2 + 2) * 2 - 2 = 6$$

$$2 + 2 * (2 - 2) = 2$$

$$(2 + 2) * (2 - 2) = 0$$

Grouping with Parentheses

Table 3-4 More Expressions

Expression	Value
$(5 + 2) * 4$	28
$10 / (5 - 3)$	5
$8 + 12 * (6 - 2)$	56
$(4 + 17) \% 2 - 1$	0
$(6 - 3) * (2 + 7) / 3$	9

Type Conversion

When You Mix Apples and Oranges: *Type Conversion*


- Operations are performed between operands of the same type.
- If not of the same type, C++ will convert one to be the type of the other
- This can impact the results of calculations.

Type Conversion

- Type Conversion: automatic conversion of an operand to another data type
- Promotion: convert to a higher type
- Demotion: convert to a lower type

Hierarchy of Types

Highest: long double
double
float
unsigned long
long
unsigned int
Lowest: int



Ranked by largest number they can hold

Conversion Rules

1) `char, short, unsigned short` automatically promoted to `int`

– For arithmetic operation

```
char c= 'A' ; cout<<6+c; // int
```

2) When operating on values of different data types, the lower one is promoted to the type of the higher one.

```
int i=25; cout<<double6.1+i; // float
```

3) When using the `=` operator, the type of expression on right will be converted to type of variable on left

```
int x, y =25; float z=2.5;  
x=y+z; //int
```

Algebraic Expressions

- Multiplication requires an operator:

$Area = lw$ is written as `Area = l * w;`

- There is no exponentiation operator:

$Area = s^2$ is written as `Area = pow(s, 2);`

- Parentheses may be needed to maintain order of operations:

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

is written as

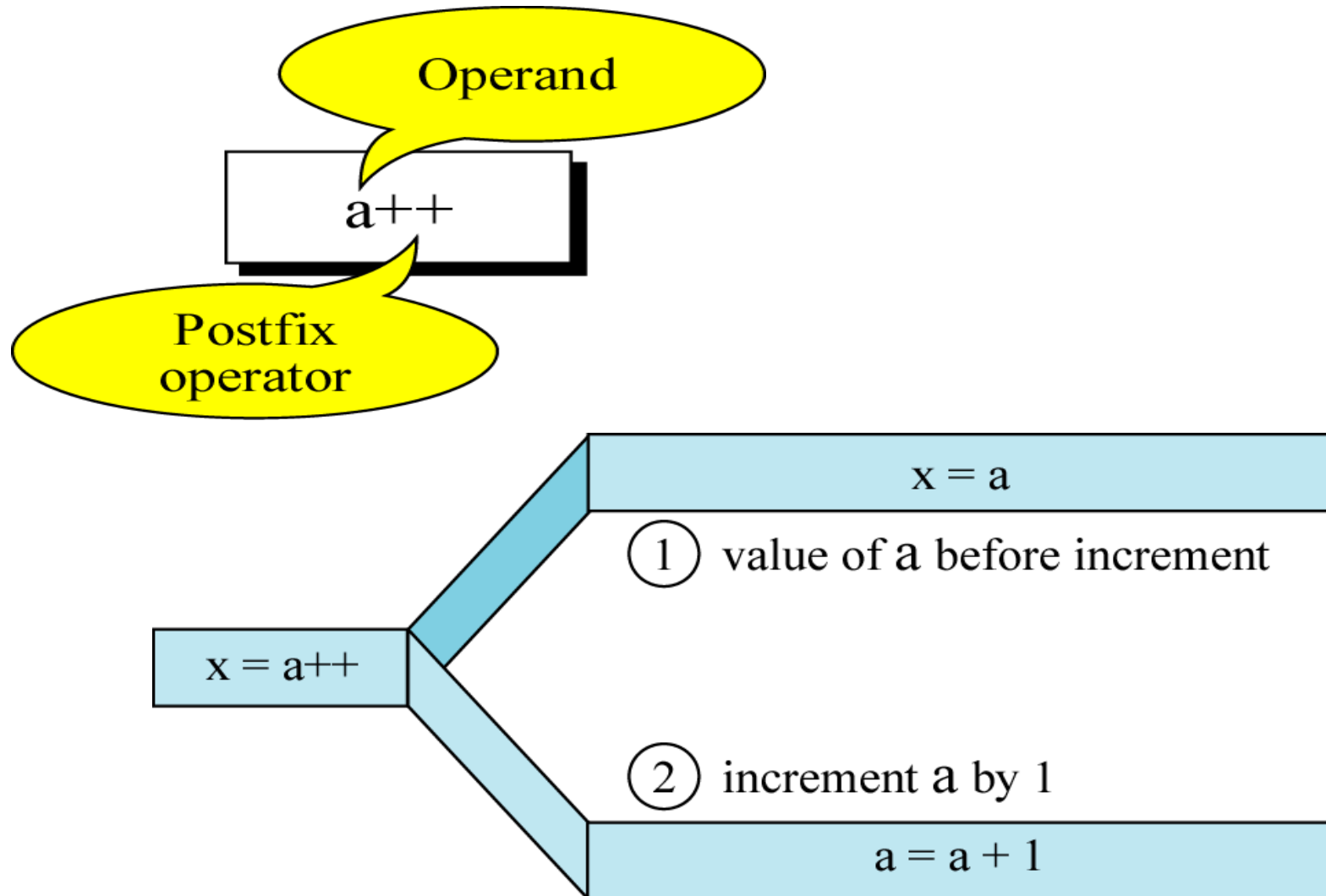
`m = (y2-y1) / (x2-x1);`

Algebraic Expressions

Table 3-5 Algebraic and C++ Multiplication Expressions

Algebraic Expression	Operation	C++ Equivalent
$6B$	6 times B	<code>6 * B</code>
$(3)(12)$	3 times 12	<code>3 * 12</code>
$4xy$	4 times x times y	<code>4 * x * y</code>

Postfix expression

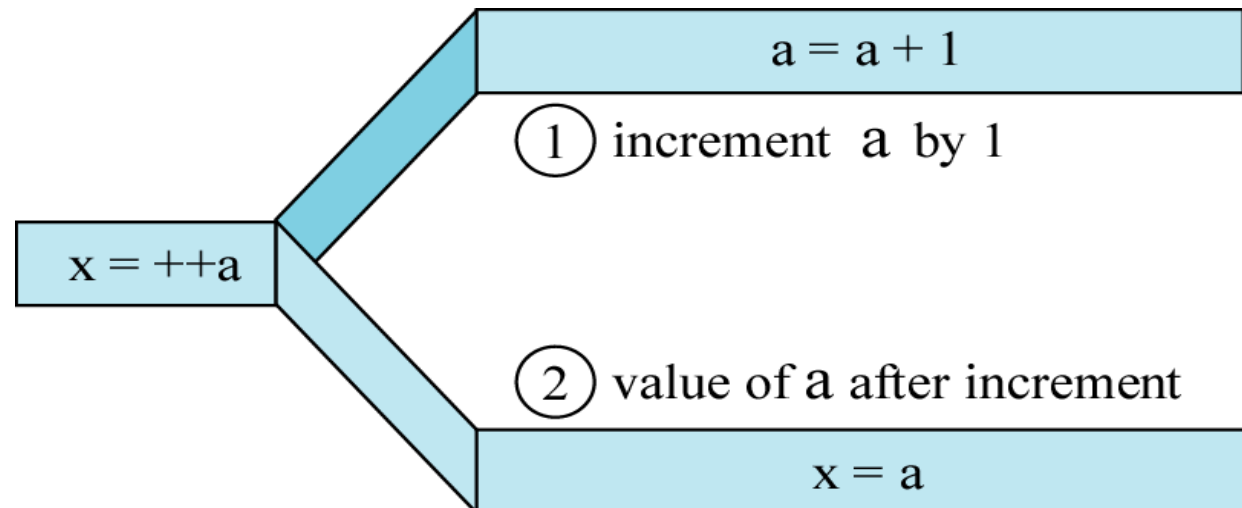


Prefix expression

Unary
Operator

$++a$

Operand





In-Class Exercise

- What would be the value of nilai_kedua:

```
int kira = 5;
```

```
int nilai_pertama = 10, nilai_kedua;
```

```
nilai_kedua = 5* kira-- + nilai_pertama;
```



```
nilai_kedua = 5* --kira + nilai_pertama;
```



Overflow and Underflow

Overflow and Underflow

- Occurs when assigning a value that is too large (overflow) or too small (underflow) to be held in a variable
- Variable contains value that is ‘wrapped around’ set of possible values
- Different systems may display a warning/error message, stop the program, or continue execution using the incorrect value

Type Casting

Type Casting


- Used for manual data type conversion
- Useful for floating point division using int:

```
double m;  
m = static_cast<double>(y2-y1)  
    to convert          / (x2-x1) ;
```



- Useful to see int value of a char variable:

```
char ch = 'C';  
cout << ch << " is "  
     << static_cast<int>(ch) ;
```



Example

Program 3-10

```
1  // This program uses a type cast to avoid integer division.
2  #include <iostream>
3  using namespace std;
4
5  int main()
6  {
7      int books;           // Number of books to read
8      int months;          // Number of months spent reading
9      double perMonth;     // Average number of books per month
10
11     cout << "How many books do you plan to read? ";
12     cin >> books;
13     cout << "How many months will it take you to read them? ";
14     cin >> months;
15     perMonth = static_cast<double>(books) / months;
16     cout << "That is " << perMonth << " books per month.\n";
17     return 0;
18 }
```

Program Output with Example Input Shown in Bold

How many books do you plan to read? **30 [Enter]**


How many months will it take you to read them? **7 [Enter]**

That is 4.28571 books per month.

C-Style and Prestandard Type Cast Expressions

- C-Style cast: data type name in ()

```
cout << ch << " is " << (int)ch;
```



- Prestandard C++ cast: value in ()

```
cout << ch << " is " << int(ch);
```

- Both are still supported in C++, although `static_cast` is preferred

Multiple Assignment and Combined Assignment

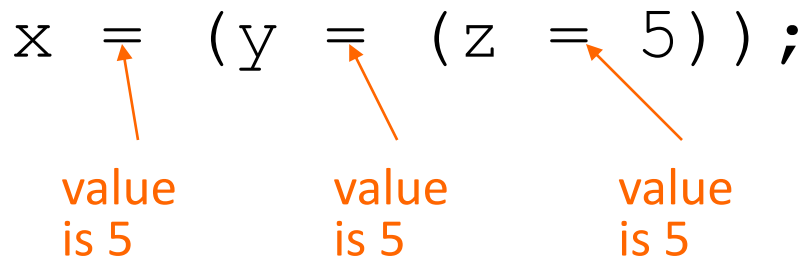
Multiple Assignment and Combined Assignment

- The = can be used to assign a value to multiple variables:

`x = y = z = 5;`

- Value of = is the value that is assigned
- Associates right to left:

`x = (y = (z = 5)) ;`



value is 5 value is 5 value is 5

Combined Assignment

- Look at the following statement:

```
sum = sum + 1;
```

```
/ sum += 1  
sum++  
, ++sum
```

This adds 1 to the variable **sum**.

Combined Assignment

- The combined assignment operators provide a shorthand for these types of statements.
- The statement

`sum = sum + 1;`

is equivalent to

`sum += 1;`

Combined Assignment Operators

Operator	Example	Equivalent to
+=	$i += 3$ $i += j + 3$	$i = i + 3$ $i = i + (j + 3)$
-=	$i -= 3$ $i -= j + 3$	$i = i - 3$ $i = i - (j + 3)$
*=	$i *= 3$ $i *= j + 3$	$i = i * 3$ $i = i * (j + 3)$
/=	$i /= 3$ $i /= j + 3$	$i = i / 3$ $i = i / (j + 3)$
%=	$i \% = 3$ $i \% = j + 3$	$i = i \% 3$ $i = i \% (j + 3)$

In-Class Exercise

Assume that `int a = 1` and `double d = 1.0`, and that each expression is independent. What are the results of the following expressions?

- i) `a = 46/9;`
- ii) `a = 46 % 9 + 4 * 4 - 2;`
- iii) `a = 45 + 43 % 5 * (23 * 3 % 2);`
- iv) `a %= 3 / a + 3;`
- v) `d += 1.5 * 3 + (++a);`
- vi) `d -= 1.5 * 3 + a++;`