CS2311 Computer Programming

LT3: Basic Syntax

Part II: Operators, Basic I/O

Outline

- Operators and punctuators
- Basic I/O with **cin** and **cout**

Operators and Punctuators

- An operator specifies an operation to be performed on some values
  - These values/variables are called the operands of the operator
- Some examples: +, -, *, /, %, ++, --,>>,<<
- Some of these have meanings that depend on the context

Operators

Category	Examples
Arithmetic	+, -, /, *, %, =, ++, --
Comparison/relational	==, !=, >, <, >=, <=
Logical	!, &&, ||
Bitwise	~, &, |, ^, <<, >>
Compound assignment	+=, &=, <<=, etc.
Member and pointer	a[b], *, &, ->, etc.
Others	::, sizeof, etc.

Increment & Decrement Operators

- Increment and decrement operators: **++** and **--**
  - **k++** and **++k** are equivalent to **k=k+1**
  - **k--** and **--k** are equivalent to **k=k-1**
- Post-increment and post-decrement: k++ and k--
  - k's value is altered AFTER the expression is evaluated

```
int k=1, j;
j=k++;  /* result: j is 1, k is 2 */
```

- Pre-increment and pre-decrement: **++k** and **--k**
  - k's value is altered BEFORE evaluating the evaluation

```
int k=1, j;
j=++k;  /* result: j is 2, k is 2 */
```


An Example

	Old x	New x	Output
int x=3;	3	3	
cout << x;	3	3	3
cout << ++x;	3	4	4
cout << x;	4	4	4
cout << x++;	4	5	4
cout << x;	5	5	5

What Values Are Printed?

```
int a=0,i=0;
cout << "i= " << i << endl;

a=0;
i=1+(a++);
cout << "i= " << i << endl;
cout << "a= " << a << endl;

a=0;
i=1+(++a);
cout << "i= " << i << endl;
cout << "a= " << a << endl;
```


$i=1+\underbrace{(a++)}_{0};$

Use original value of a before increment

i =1+0
=1

$i=1+\underbrace{(++a)}_{1};$

Use updated value of a after increment

i =1+1
=2

Value of a is 1 in both cases

Answer

```
int a = 0, i = 0;
cout << "i = " << i << endl;

a = 0;
i = 1+(a++);
cout << "i = " << i << endl;
cout << "a = " << a << endl;

a = 0;
i = 1+(++a);
cout << "i = " << i << endl;
cout << "a = " << a << endl;
```

Output

```
i = 0
i = 1
a = 1
i = 2
a = 1
```


Precedence & Associativity of Operators

- An expression may have more than one operator and its precise meaning depends on the precedence and associativity of the involved operators
- What are the values of variables a, b and c after the execution of the following statements

```
int a, b = 2, c = 1;
a = b+++c;
```

- Which of the following interpretation is right?

```
a = (b++) + c; /* right */
```

or

```
a = b + (++c); /* wrong */
```

Precedence & Associativity of Operators

Precedence: order of evaluation for different operators

Associativity: order of evaluation for operators with the same precedence

Operator Precedence (high to low)						Associativity
::						None
.	->	[]				Left to right
()	++(postfix)	--(postfix)				Left to right
+(unary)	- (unary)	++ (prefix)	-- (prefix)			Right to left
*	/	%				Left to right
+	-					Left to right
=	+=	-=	*=	/=	etc.	Right to left

Assignment Operator =

- Generic form

  variable = expression;

- **=** is an assignment operator
  - that has nothing to do with mathematical equality
    - (which is **==** in C++)

- An expression itself has a value, e.g., [demo]

  a=(b=2)+(c=3);

  - An assignment statement has a value equal to the operand
  - In the example, the value of **a** is **5**;
  - Be careful about implicit type conversion

Examples of Assignment Statements

```
/* Invalid: left hand side must be a variable */
a + 10 = b;

/*assignment to constant is not allowed*/
2=c;

/* valid but not easy to understand */
int a, b, c;
a = (b = 2) + (c = 3);

/* avoid complex expressions*/
int a, b, c;
b = 2;
c = 3;
a = b + c;
```


Swapping the Values

- We want to swap the content of two variables, a and b.
- What's wrong with the following program? [demo]

```
int main() {
    int a=3, b=4;
    a=b;
    b=a;
    return 0;
}
```

- We need to make use of a temporary variable

```
c=b; /*save the old value of b*/
b=a; /*put the value of a into b*/
a=c; /*put the old value of b to a*/
```

Efficient/Compound Assignment

- The generic form of efficient assignment operators:

  variable op= expression;

  where op is an operator. The meaning is

  variable = variable op (expression);

- Efficient assignment operators include

  += -= *= /= %= (arithmetic operators)

  >>= <<= &= ^= |= (bitwise operators)

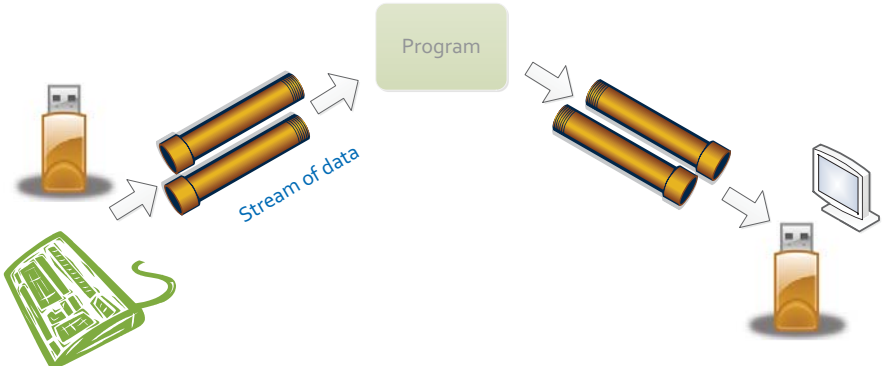
- Examples:

```
a+=5;      // is same as a=a+5;
a-=5;      // is same as a=a-5;
a+=b*c;    // is same as a=a+(b*c);
a*=b+c;    // is same as a=a*(b+c);
```

- Also known as **compound assignment** operators

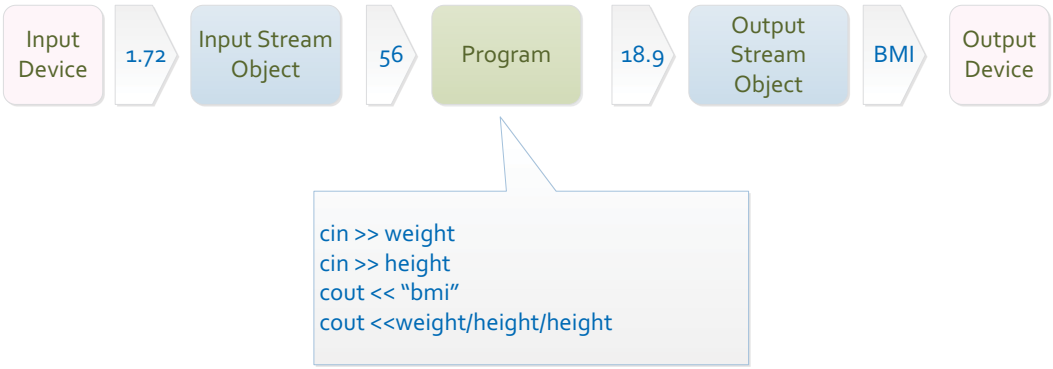
Basic I/O – Keyboard and Screen

- A program can do little if it can't take input and produce output
- Most programs read user input from keyboard and secondary storage
- After processing the input data, result is commonly displayed on screen or write to storage (disk)



Basic I/O – cin and cout

- C++ comes with an **iostream** package (library) for basic I/O.
- **cin** and **cout** are objects defined in **iostream** for keyboard input and screen display respectively
- To read data from **cin** and write data to **cout**, we need to use extraction/input operator (**>>**) and insertion/output operator (**<<**)



cout: Output Operator (<<)

- Preprogrammed for all standard C++ data types
- It sends bytes to an output stream object, e.g. **cout**
- Predefined "manipulators" can be used to change the default format of arguments

cout: Output Operator <<

Type	Expression	Output
Integer	cout << 21	21
Float	cout << 14.5	14.5
Character	cout << 'a'; cout <<'H' << 'i'	a Hi
Bool	cout << true cout << false	1 0
String	cout << "hello"	hello
New line (endl)	cout << 'a' << endl << 'b';	a b
Tab	cout << 'a' << '\t' << 'b';	a b
Special characters	cout << '\"' << "Hello" << '\"' <<endl;	"Hello"
Expression	int x=1; cout << 3+4 +x;	8

cout – Change the width of output

- Change the width of output
  - Calling member function width or using setw manipulator
  - Must **#include <iomanip>** for setw
  - Leading blanks are added to any value fewer than width
  - If formatted output exceeds the width, the entire value is printed
  - Effect last for one field only

Approach	Example	Output (◆ for space)
cout.width(*width*)	cout.width(10); cout << 5.6 << endl; cout.width(10); cout <<57.68 << endl;	◆◆◆◆◆◆◆5.6 ◆◆◆◆◆57.68

setw(*width*)	cout << setw(5) << 1.8;
	cout << setw(5) << 23 <<endl;
	cout << setw(5) << 6.71;
	cout << setw(5) << 1 <<endl;
		◆◆1.8◆◆◆23
		◆6.71◆◆◆◆1

cout – Set the Precision and Format of Floating Point Output

- Must **#include <iomanip>**
- Floating-point precision is six by default, i.e. 6 digits in total
- Use setprecision, fixed and scientific manipulators to change the precision value and printing format
- Effect is permanent

Default behavior

Example	Output
cout << 1.34 << endl; cout << 1.340 << endl; cout << 1.3401234 << endl; cout << 0.0000000134 << endl;	1.34 1.34 1.34012 1.34e-008

fixed and scientific Manipulators

- **fixed**: always uses the fixed point notation
- **scientific**: always uses the scientific notation
- They change the meaning of precision (see the example)

Example	Output
cout << fixed; cout << 1.34 << endl; cout << 1.340 << endl; cout << 0.0000000134 << endl; cout << scientific; cout << 1.34 << endl; cout << 1.340 << endl; cout << 0.0000000134 << endl;	1.34 1.34 0.000000 1.340000e+00 1.340000e+00 1.340000e-08

cout setprecision

- Normally, **setprecision**(n) means output n significant digits in total
- But with "**fixed**" or "**scientific**", **setprecision(n)** means output n significant digits after the decimal points

Example	Output
cout << setprecision(2); cout << 1.34 <<endl; cout << 0.0000000134 << endl; cout << fixed; cout << 0.0000000134 << endl; cout << scientific << 0.0005 << endl;	1.3 1.3e-08 0.00 5.00e-004

cout – Other Manipulators

Manipulators	Example	Output

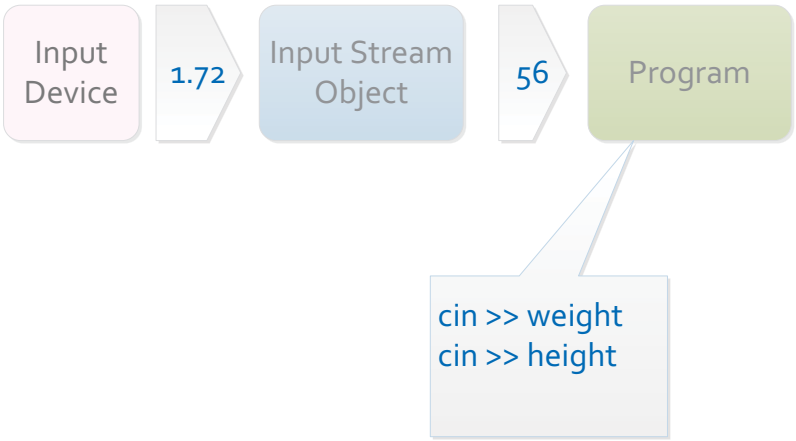
fill	cout << setfill('*');
	cout << setw(10);
	cout << 5.6 << endl;
	cout << setw(10);
	cout <<57.68 << endl;
		*******5.6
		*****57.68

radix	cout << oct << 11 << endl; // octal
	cout << hex << 11 << endl; // hexidecimal
	cout << dec << 11 << endl;
		13
		b
		11

alignment	cout << setiosflags(ios::left);
	cout << setw(10);
	cout << 5.6 <<endl;
		5.6

cin: Extraction Operators (>>)

- Preprogrammed for all standard C++ data types
- Get bytes from an input stream object
- Depend on **white space** to separate incoming data values



Input Operator

Type	Variable	Expression	Input	x	y
Integer	int x,y;	cin >> x;	21	21	
		cin >> x >> y;	5 3	5	3
Float	float x,y;	cin >> x;	14.5	14.5	
Character	char x,y;	cin >> x;	a	a	
		cin >> x >> y;	Hi	H	i
String	char x[20]; char y[20];	cin >> x;	hello	hello	
		cin >> x >> y	Hello World	Hello	World

Programming Styles

- Programmers should write code that is understandable to other people as well
- Meaningful variable names
- Which is more meaningful

```
tax = temp1*temp2; // not meaningful
tax = price*tax_rate; // good
```

- Meaningful Comments
  - Write comments as you write the program
- Indentation

Indentation Styles

```
int main()
{
   int x, y;
   x = y++;
    return 0;
}
```

```
int main() {
   int x, y;
   x = y++;
    return 0;
}
```

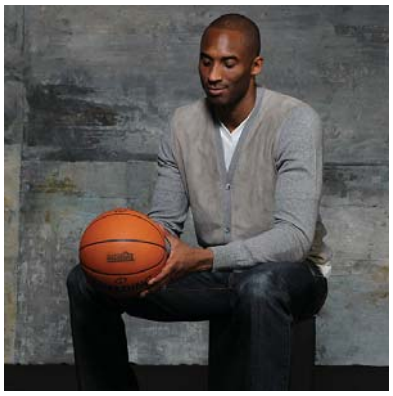
Both are good. Choose one and stick with it.

```
int main()
{
int x, y;
 x= y++;
return 0;}
```

BAD!! Avoid this!!

Style Is Important

Both are good. Choose one and stick with it.



BAD!! Avoid this!!

Use of Comments

- Top of the program
  - Include information such as the name of organization, programmer's name, date and purpose of program
- What is achieved by the function, the meaning of the arguments and the return value of the function
- Short comments should occur to the right of the statements when the effect of the statement is not obvious and you want to illuminate what the program is doing
- Which one of the following is more meaningful?

```
tax = price * rate; /* sales tax formula */
tax = price * rate; /* multiply price by rate */
```