Programming Fundamentals with C++

Lecture 4 – Fundamentals - II

Dr. Muhammad Sajjad

RA: Asad Ullah

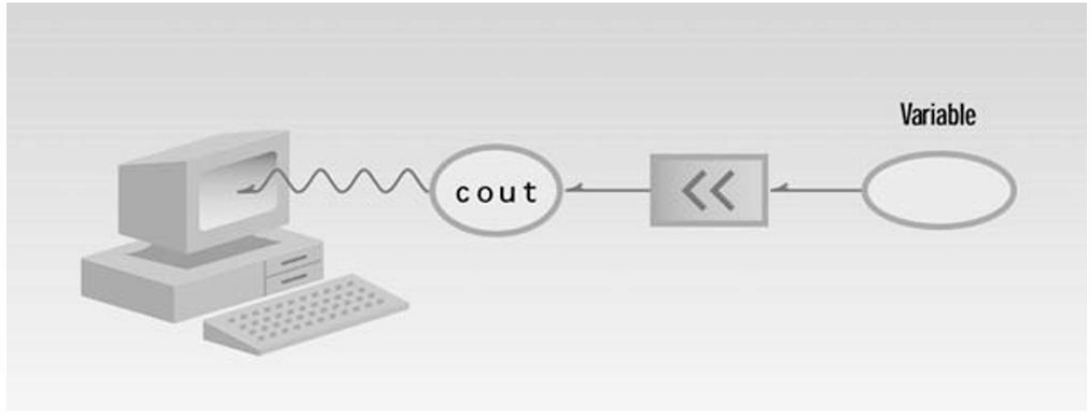
Overview

- **Basic Input/Output (I/O) in C++**
  - What is I/O in Programming?
  - Standard I/O Streams in C++
- **The cout Object**
- **Escape Sequence**
- **The endl & setw Manipulators**
- **The cin Object**
- **Assignment Statements**
- **Increment / Decrement Operators**
- **The Comment Statement**

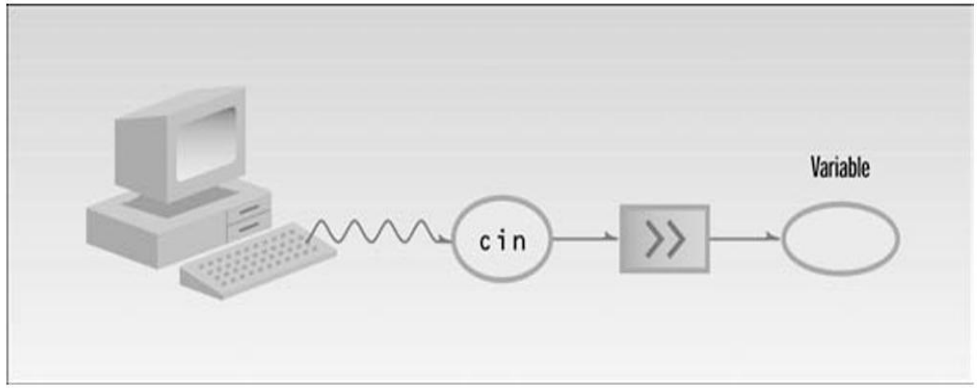
Basic Input / Output (I/O) in C++

- **What is I/O in Programming?**
  - **Input:** Information provided by the user to the program.
  - **Output:** Information displayed by the program to the user.

- **Standard I/O Streams in C++**
  - **cin (Console Input):** Receives input from the user.
  - **cout (Console Output):** Displays output on the screen.



cout



cin

The cout Object

- **What is cout?**
  - **cout** stands for "console output" and is used to print text or data to the screen.
  - Defined in the **<iostream>** library and requires the **std** namespace.
- **Using the ' << ' Operator**
  - The << (insertion operator) sends data to cout.

```
int x = 10;
cout<< "The value of x is: "<< x << endl;
```

1_cout_object.cpp

- **Chaining << for Multiple Outputs**
  - You can chain << to print text and variables in one line.

```
int a = 10, b = 20;
cout<< "The value of a is "<< a << "and b is "<< b << endl;
```

Escape Sequence

- **Escape Sequence**
  - Escape Sequence are special non-printing characters used to control printing on the output device.
  - These can be written inside string constant or independently in single or double quotes.
  - An Escape Sequence is the combination of backslash '\' (control character) and a code character.
  - For example to transfer the printing control to the next line the escape sequence is written as;
    - cout<<"I Love Pakistan\n";
  - It can be used anywhere in the output stream, in the beginning, middle or end of the string constant.
  - For printing above string in three lines: cout<<"I \n Love \n Pakistan";

Escape Sequence	Meaning
\a	Alarm or Beep
\b	Backspace
\n	New Line
\t	Tab (Horizontal)
\v	Vertical Tab
\\	Backslash
\'	Single Quote
\"	Double Quote

2_escape_sequence.cpp

The endl & setw Manipulators

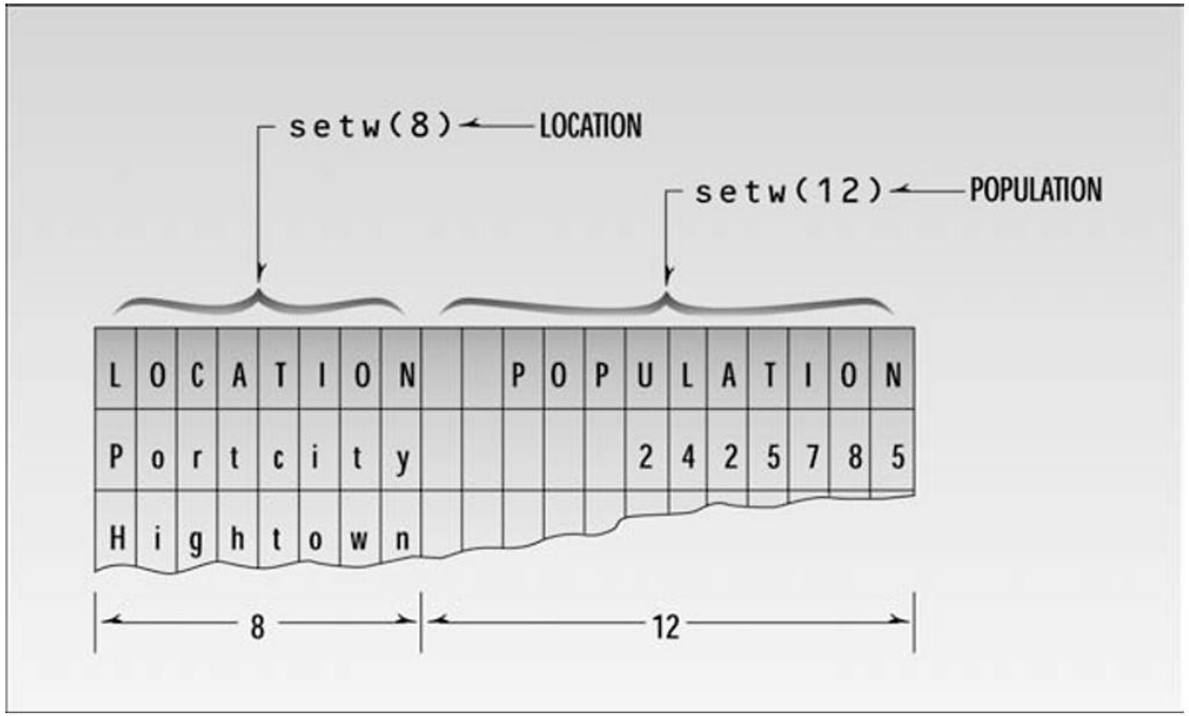
- Manipulators are operators that are used with insertion(<<) operator.
- **The endl Manipulator**
  - endl inserts a newline character and flushes the output buffer, ensuring all output is displayed immediately.
  - Place endl after text to move to a new line.
  - **Difference Between \n and endl**
    - **\n**: Adds a new line but doesn't flush the buffer.
    - **endl**: Adds a new line and flushes the buffer, which can be useful in real-time applications.

```
cout << "Hello, World!" << endl;
cout << "Welcome to C++ programming!" << endl;
```

```
Hello, World!
Welcome to C++ programming!
```

The endl & setw Manipulators

- Manipulators are operators that are used with insertion(<<) operator.
- **The setw Manipulator**
  - The setw (set width) manipulator specifies a fixed width for the next output field, useful for aligning columns.
  - Include <iomanip> library to use setw.
  - **Why Use setw?**
    - setw aligns output into neat columns, useful in data tables, reports, or any output where readability is important.



3_setw.cpp

The cin Object

- **What is cin?**
  - **cin** stands for "console input" and is used to get input from the user.
  - Defined in the **<iostream>** library, it captures keyboard input and stores it in variables.
- **Using the ‘ >> ’ Operator**
  - The >> (extraction operator) reads input and stores it in variables.
- **Chaining >> for Multiple Inputs**
  - You can use >> multiple times to read several variables at once.
- **Input Prompting and Validation**
  - Always prompt users with a **cout** statement before using **cin**, so they know what to enter.

4_cin_object.cpp

```
int age;
cout << "Enter your age: ";
cin >> age;  // User enters a value, stored in age
cout << "Your age is: " << age << endl;
```

```
int num1, num2;
cout << "Enter two numbers separated by a space: ";
cin >> num1 >> num2;
cout << "You entered " << num1 << " and " << num2 << endl;
```

```
double price;
cout << "Enter the item price: $";
cin >> price;
cout << "The entered price is: $" << price << endl;
```

Complete Example Program: Combining cin, cout, endl, and setw

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

int main() {
    int age;
    double score;

    // Input from user
    cout << "Enter your age: ";
    cin >> age;
    cout << "Enter your test score: ";
    cin >> score;

    // Display in formatted table
    cout << endl; // Add a blank line
    cout << setw(10) <<"Age" << setw(10) << "Score" << endl;
    cout << setw(10)) << age << setw(10) << score << endl;

    return 0;
}
```

Assignment Statements

- **Assignment Operator (=)**
  - The assignment operator in C++ is used to assign a value to a variable.
  - The operator = assigns the value on its right side to the variable on its left side.
- **Compound Assignment Operators**
  - **Compound assignment operators** are a shorthand for performing an operation on a variable and then assigning the result to that same variable.
  - These operators combine an arithmetic operation with the assignment operation. Some common compound assignment operators include:
    - += (Add and assign)
    - -= (Subtract and assign)
    - *= (Multiply and assign)
    - /= (Divide and assign)
    - %= (Modulus and assign)

```
int a = 5; // Assigns the value 5 to variable 'a'
```

```
int a = 10;
int b = 2;
```

operator	example	equivalent to
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b

5_assignment_operators.cpp

Assignment Statements

- **Compound Assignment Expressions**
  - A **compound assignment expression** uses a compound assignment operator in an expression, often as part of a larger computation.
  - These expressions may appear within larger calculations, conditional statements, or loops, where the result of a compound assignment operator is used immediately in subsequent code.

```
int a = 5;
int b = 10;
int result = (a += 3) + (b *= 2); // result is 26

// Explanation:
// 1. (a += 3) modifies 'a' to be 8 (5 + 3) and returns 8.
// 2. (b *= 2) modifies 'b' to be 20 (10 * 2) and returns 20.
// 3. result = 8 + 20, so result is assigned the value 26.
```

Increment / Decrement Operators

- **Increment Operator (++)**

  The increment operator (++) increases the value of a variable by 1. It has two forms:

  - **Prefix increment (++var)**: Increments the variable first, then returns the incremented value.
  - **Postfix increment (var++)**: Returns the current value of the variable, then increments it.

```
int a = 5;

int b = ++a; // Prefix: 'a' is incremented to 6, then 'b' is assigned the value 6
int c = a++; // Postfix: 'c' is assigned the current value of 'a' (6), then 'a' is
incremented to 7

cout << "a: " << a << endl;          // a: 7
cout << "b: " << b << endl;          // b: 6
cout << "c: " << c << endl;          // c: 6
```

6_increment_operator.cpp

Increment / Decrement Operators

- **Decrement Operator (--)**

  The decrement operator (--) decreases the value of a variable by 1. It also has two forms:

  - **Prefix decrement (- -var)**: Decrements the variable first, then returns the decremented value.
  - **Postfix decrement (var- -)**: Returns the current value of the variable, then decrements it.

```
int x = 10;

int y = --x; // Prefix: 'x' is decremented to 9, then 'y' is assigned the value 9
int z = x--; // Postfix: 'z' is assigned the current value of 'x' (9), then 'x' is
decremented to 8

cout << "x: " << x << endl;          // x: 8
cout << "y: " << y << endl;          // y: 9
cout << "z: " << z << endl;          // z: 9
```

7_decrement_operator.cpp

The Comment Statement

In C++, comments are used to explain code, making it easier to understand for anyone reading it. Comments are ignored by the compiler, so they don't affect how the program runs. C++ has two types of comments:

1. Single-line comments
2. Multi-line comments

1. Single-line Comments (//)

- Single-line comments start with // and continue until the end of the line. They're typically used for short explanations of code.

```
int x = 10; // This is a single-line comment explaining that x is initialized with the value 10
```

2. Multi-line Comments (/* ... */)

- Multi-line comments start with /* and end with */. They're used for longer explanations and can span multiple lines.

The Comment Statement

1. **Multi-line Comments (/* ... */)**
   - **Example**

```
/*
This is a multi-line comment.
It can be used for longer explanations
and spans multiple lines.
*/
int y = 20;
```

Thank You