Chapter 2:

Introduction to C++

2.1 Parts of a C++ Program

// sample C++ program
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello, there!"
    return 0;
}

- **Comments**: Explain about the program, can appear anywhere, and are ignored by the compiler.
- **Directives**: Commands to the preprocessor to set up your source code for the compiler.
- **using name space**: Specifies where the names of the variables, functions, and other entities are declared.
- **int main()**: The beginning of the main function; The instructions are included within the pair of curly braces. It returns an integer.

- **Note**: C++ is case sensitive.
Special Characters

<table>
<thead>
<tr>
<th>Character</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>//</td>
<td>Double slash</td>
<td>Beginning of a comment</td>
</tr>
<tr>
<td>#</td>
<td>Pound sign</td>
<td>Beginning of preprocessor directive</td>
</tr>
<tr>
<td>&lt; &gt;</td>
<td>Open/close brackets</td>
<td>Enclose filename in #include</td>
</tr>
<tr>
<td>( )</td>
<td>Open/close parentheses</td>
<td>Used when naming a function</td>
</tr>
<tr>
<td>{ }</td>
<td>Open/close brace</td>
<td>Encloses a group of statements</td>
</tr>
<tr>
<td>&quot; &quot;</td>
<td>Open/close quotation marks</td>
<td>Encloses string of characters</td>
</tr>
<tr>
<td>;</td>
<td>Semicolon</td>
<td>End of a programming statement</td>
</tr>
</tbody>
</table>

2.2 The cout Object

- The cout Object
  - Represents the standard output stream, which is set to the console in most systems by default
  - Displays output on the computer screen
  - You use the stream insertion operator `<<` to send output to cout:
    ```cpp
    cout << "Programming is fun!";
    ```
    ```cpp
    cout << "Hello " << "there!";
    ```
    Or:
    ```cpp
    cout << "Hello ";
    cout << "there!";
    ```
    Both produce the same output as:
    ```cpp
    cout << "Hello there!";
    ```
The `cout` Object

- What if we need to display something in different lines?
- This produces one line of output:
  
  ```cpp
  cout << "Programming is ";
  cout << "fun!";
  ```

- `cout` does not start a new line unless told to do so

The `endl` Manipulator

- You can use the `endl` manipulator to start a new line of output. This will produce two lines of output:
  
  ```cpp
  cout << "Programming is" << endl;
  cout << "fun!";
  ```

- You do NOT put quotation marks around `endl`

- The last character in `endl` is a lowercase L, not the number 1.

  ```cpp
  endl
  ```

  This is a lowercase L
The \n Escape Sequence

- You can also use the \n escape sequence to start a new line of output. This will produce two lines of output:

```cpp
cout << "Programming is\n";
cout << "fun!";
```

Notice that the \n is INSIDE the string. Do not use forward slash (/).
The `#include` Directive

- Inserts the contents of another file into the program.
  - `#include <iostream>` – inserts the `iostream` header file, which contains information about `cout` and is needed for the compilation of programs using `cout` object
- This is a preprocessor directive, not part of C++ language
- `#include` lines will not be seen by the compiler
- Do not place a semicolon at end of `#include` line

Variables and Literals

- The program uses variables to store data and literals to represent data
- **Variable**: a storage location in memory
  - Has a name and a type of data it can hold
  - Must be defined before it can be used:

```cpp
int item;
```

Program 2-7

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

**Program Output**
The value in number is 5
Literals

- Literal: a value that is written into a program’s code.

  "hello, there" (string literal)
  12 (integer literal)
  "number" (string literal)
  "12" (string literal)

Note: integer 12 is different from string “12”.

```
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    apples = 20;
9    cout << "Today we sold " << apples << " bushels of apples.\n";
10   return 0;
11 }
```

Program Output
Today we sold 20 bushels of apples.

### Literals

- What will the following program segment print out?
  ```cpp
  number = 20;
  cout << "number = " << "number";
  ```

- Is the following program segment correct?
  ```cpp
  int grade;
  grade = "80";
  ```

### Identifiers

- An identifier is a programmer-defined name for some element of a program: variables, functions, etc.

### C++ Key Words

You cannot use any of the C++ key words as an identifier. These words have reserved meaning.

<table>
<thead>
<tr>
<th>C++ Key Words</th>
<th>(total=74)</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>continue</td>
</tr>
<tr>
<td>and_eq</td>
<td>default</td>
</tr>
<tr>
<td>asm</td>
<td>delete</td>
</tr>
<tr>
<td>auto</td>
<td>do</td>
</tr>
<tr>
<td>bitand</td>
<td>double</td>
</tr>
<tr>
<td>bitor</td>
<td>dynamic_cast</td>
</tr>
<tr>
<td>bool</td>
<td>else</td>
</tr>
<tr>
<td>break</td>
<td>enum</td>
</tr>
<tr>
<td>case</td>
<td>explicit</td>
</tr>
<tr>
<td>catch</td>
<td>export</td>
</tr>
<tr>
<td>char</td>
<td>extern</td>
</tr>
<tr>
<td>class</td>
<td>false</td>
</tr>
<tr>
<td>compl</td>
<td>float</td>
</tr>
<tr>
<td>const</td>
<td>for</td>
</tr>
<tr>
<td>const_cast</td>
<td>friend</td>
</tr>
</tbody>
</table>
Variable Names

• A variable name should represent the purpose of the variable. For example, instead of $x$, use:

  \textit{itemsOrdered}

  The purpose of this variable is to hold the number of items ordered.

Identifier Rules

• The first character of an identifier must be an alphabetic character or an underscore (_).
• After the first character you may use alphabetic characters, numbers, or underscore characters.
• Upper- and lowercase characters are distinct.

Valid and Invalid Identifiers

<table>
<thead>
<tr>
<th>IDENTIFIER</th>
<th>VALID?</th>
<th>REASON IF INVALID</th>
</tr>
</thead>
<tbody>
<tr>
<td>totalSales</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>total_Sales</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>total.Sales</td>
<td>No</td>
<td>Cannot contain .</td>
</tr>
<tr>
<td>4thQtrSales</td>
<td>No</td>
<td>Cannot begin with a digit</td>
</tr>
<tr>
<td>totalSale$</td>
<td>No</td>
<td>Cannot contain $</td>
</tr>
</tbody>
</table>

2.6 Integer Data Types
Integer Data Types

- There are many different types of data
- Integer variables can hold whole numbers such as 12, 7, and -99.
- Integer data type is defined using
  - `int`
  - `unsigned int`
- There are other integer data types, each representing integers with different range of values.

<table>
<thead>
<tr>
<th>Type Name</th>
<th>Bytes</th>
<th>Range of Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int</code></td>
<td>4</td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td><code>unsigned int</code></td>
<td>4</td>
<td>0 to 4,294,967,295</td>
</tr>
<tr>
<td><code>_int8</code></td>
<td>1</td>
<td>-128 to 127</td>
</tr>
<tr>
<td><code>unsigned _int8</code></td>
<td>1</td>
<td>0 to 255</td>
</tr>
<tr>
<td><code>_int16</code></td>
<td>2</td>
<td>-32,768 to 32,767</td>
</tr>
<tr>
<td><code>unsigned _int16</code></td>
<td>2</td>
<td>0 to 65,535</td>
</tr>
<tr>
<td><code>_int32</code></td>
<td>4</td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td><code>unsigned _int32</code></td>
<td>4</td>
<td>0 to 4,294,967,295</td>
</tr>
<tr>
<td><code>_int64</code></td>
<td>8</td>
<td>-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807</td>
</tr>
<tr>
<td><code>unsigned _int64</code></td>
<td>8</td>
<td>0 to 18,446,744,073,709,551,615</td>
</tr>
</tbody>
</table>

Defining Variables

- Variables of the same type can be defined
  - On separate lines:
    ```
    int length;
    int width;
    unsigned int area;
    ```
  - On the same line:
    ```
    int length, width;
    unsigned int area;
    ```
- Variables of different types must be in different definitions

Program 2-10

```cpp
// This program has variables of several of the integer types.
#include <iostream>
using namespace std;

int main()
{
    int checking;
    unsigned int miles;
    long days;
    checking = -20;
    miles = 4276;
    days = 189000;
    cout << "We have made a long journey of " << miles << " miles;";
    cout << " miles.\n";
    cout << "Our checking account balance is " << checking << " dollars;";
    cout << "\nAbout " << days << " days ago Columbus;";
    cout << "stood on this spot.\n";
    return 0;
}
```

This program has three variables: checking, miles, and days
Integer Literals

• An integer literal is an integer value that does not contain a decimal point. For example:

```cpp
itemsOrdered = 15;
```

In this code, 15 is an integer literal.

Integer Literals

• Integer literals are stored in memory as base 10 ints by default
• C++ allows integers to be represented in other bases
• Constants that begin with ‘0’ (zero) are base 8: 075
• Constants that begin with ‘0x’ are base 16: 0x75A

Program 2-10

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

int main()
{
    int checking;
    unsigned int miles;
    long days;
    checking = 450;
    miles = 4276;
    days = 109000;
    cout << "We have made a long journey of " << miles;
    cout << " miles."
    cout << "Our checking account balance is " << checking;
    cout << "About " << days << " days ago Columbus."
    cout << "stood on this spot.
"
    return 0;
}
```
The `char` Data Type

- Used to hold characters or very small integer values
- Usually 1 byte of memory
- Numeric value of character from the character set is stored in memory:

  CODE:
  ```
  char letter;
  letter = 'C';
  ```

  MEMORY:
  ```
  67 => 01000011
  ```

Character Literals

- Character literals must be enclosed in single quote marks. Example:
  ```
  char ans;
  ans = 'Y'; // correct
  ans = "Y"; // wrong
  ```

Character Strings

- A series of characters in consecutive memory locations:
  
  "Hello"

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

- Comprised of the characters between the " "

  \n  
  Program 2-13

```cpp
1  // This program uses character literals.
2  #include <iostream>
3  using namespace std;
4  
5  int main()
6  {
7      char letter;
8      
9      letter = 'A';
10     cout << letter << endl;
11     letter = 'B';
12     cout << letter << endl;
13     return 0;
14  }
```

Program Output

A
B
Character vs. String

- 'A' is stored as
  \[ A \]

- "A" is stored as
  \[ A \ 0 \]

```
char letter;
letter = 'A';     // This is OK
letter = "A";     // This will not work
```

The String Data Type

- Allows a sequence of characters to be stored in one memory location
- Actually a class and not a true data type built into the language
- Needs to include `<string>` header file

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

int main()
{
    string stName;
    stName = "Steven Moore";
    cout << "Welcome " << stName << endl;
    return 0;
}
```

2.8 Floating-Point Data Types
Floating-Point Data Types

- Used to define variables that hold real numbers
- The floating-point data types are:
  ```
  float
double
long double
  ```
- They can hold real numbers such as:
  ```
  123.45
  -0.38
  ```
- Stored in a form similar to scientific notation:
  ```
  1.2345\times10^2
  -3.8\times10^{-1}
  ```
- All floating-point numbers are signed

Floating-point Literals

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

Program 2.15

```c++
1 // This program uses floating point data types.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7   float distance;
8   double mass;
9   10   distance = 1.495979E11;
11   mass = 1.989E30;
12   cout << "The Sun is " << distance << " meters away.\n";
13   cout << "The Sun's mass is " << mass << " kilograms.\n";
14   return 0;
15 }
```

Program Output

The Sun is 1.495979e+11 meters away.
The Sun's mass is 1.989e+30 kilograms.
2.9

The bool Data Type

- Represents binary values that are true or false
- bool variables are stored as small integers
- false is represented by 0, true by 1:

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

2.10

Determining the Size of a Data Type

Program 2-16

```cpp
// This program demonstrates boolean variables.
#include <iostream>
using namespace std;

int main()
{
    bool boolValue;
    boolValue = true;
    cout << boolValue << endl;
    boolValue = false;
    cout << boolValue << endl;
    return 0;
}
```

Program Output

1
0
Determining the Size of a Data Type

The size of some data type may be different from machine to machine.
The `sizeof` operator gives the size of any data type or variable:

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

2.11 Variable Assignments and Initialization

• An assignment statement uses the assignment `=` operator to store a value in a variable.
  ```cpp
  item = 12;
  ```

• This statement assigns the value 12 to the `item` variable.

• Can also assign other data types
  ```cpp
  ans = 'A';
  ```

Assignment

• The variable receiving the value must appear on the left side of the `=` operator.
• This will NOT work:
  ```cpp
  // ERROR!
  12 = item;
  ```
Variable Initialization

• A variable just defined can have any value
• To initialize a variable means to assign it a value when it is defined:
  int length = 12;
• Can initialize some or all variables:
  int length = 12, width = 5, area;

Named Constants

• A named constant represents a value that cannot change while the program is running
• Declared with keyword const and capital letters
• Must be initialized when declared
  – For example:
    • const double PI = 3.1415926;
    • const int WIDTH; // ERROR! Must
    • const int WIDTH = 10; // initialize

Program 2-18

```
1 // This program shows variable initialization.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7   int month = 2, days = 28;
8   cout << "Month " << month << " has " << days << " days.\n";
9   return 0;
10 }
```
Scope

- The scope of a variable: the part of the program in which the variable can be used
- The rules are complex and will be introduced gradually
- A variable cannot be used before it is defined

---

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) \(\text{exp1 ? exp2 : exp3}\) (Only one in C++, will be covered in Chapter 4)
### Binary Arithmetic Operators

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>OPERATION</th>
<th>EXAMPLE</th>
<th>VALUE OF ans</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
<td>ans = 7 + 3;</td>
<td>10</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td>ans = 7 - 3;</td>
<td>4</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>ans = 7 * 3;</td>
<td>21</td>
</tr>
<tr>
<td>/</td>
<td>division</td>
<td>ans = 7 / 3;</td>
<td>2</td>
</tr>
<tr>
<td>%</td>
<td>modulus</td>
<td>ans = 7 % 3;</td>
<td>1</td>
</tr>
</tbody>
</table>

### Assignment Statement

- **Format:**
  
  ```
  Variable = Expression;
  ```

  It evaluates the value of the expression and assign it to the variable
  
  ```
  double area, radius;
  radius = 100.0;
  area = 3.14 * radius * radius;
  cout << "Area of the circle is " << area << endl;
  ```

### A Closer Look at the / Operator

- When an expression is used with `cout`, its value will be evaluated and the result will be inserted into `cout`
- `/` (division) operator performs integer division if both operands are integers
  ```
  cout << 13 / 5;    // displays 2
  cout << 91 / 7;    // displays 13
  ```

### A Closer Look at the % Operator

- If either operand is floating point, the result is floating point
  ```
  cout << 13 / 5.0;  // displays 2.6
  cout << 91.0 / 7;  // displays 13.0
  ```

- `%` (modulus) operator computes the remainder resulting from integer division
  ```
  cout << 13 % 5;    // displays 3
  ```

- `%` requires integers for both operands
  ```
  cout << 13 % 5.0;  // error
  ```
2.14 Comments

• Used to document parts of the program
• Intended 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:

```c
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 /*, end with */
• Can span multiple lines:
  /* This is the first line of the comment
     .......
  This is the last line of the comment */
• Can begin and end on the same line:
  ```c
  int area;        /* calculated area */
  ```
2.15 Programming Style

• The visual organization of the source code
• Includes the use of spaces, tabs, and blank lines
• Does not affect the syntax of the program
• Affects the look and readability of the source code

Common elements to improve readability:
• Braces { } aligned vertically
• Indentation and alignment of statements within a set of braces

```cpp
int main()
{
    int number; // not indented
    number = 12;
    cout << "Number = " << number << endl;
    return 0;
} // not aligned
```

```cpp
int fahrenheit = 98.7, celsius = 37;

cout << "The Fahrenheit temperature is "
    << fahrenheit
    << " and the Celsius temperature is "
    << celsius
    << endl;
```
A Wage Problem

- The regular work hours per week is 40. Any hours worked over 40 are considered overtime. An employee earns $18.25 / hr. for regular hours, and $27.78 / hr. for overtime hours. The employee worked 50 hours this week. Write a program to calculate and display the employee’s total wages for the week.

The Programming Process

1. Understand the problem
2. Develop an algorithm for solving the problem
3. Translate the algorithm into a computer language
4. Type the code, save it and compile it
5. Correct any syntax errors found during compilation. Repeat steps 4 and 5 as many times as needed
6. Run the program and check the result with test data
7. Correct any runtime and logical errors found while running the program. Repeat steps 4~7 as many times as needed

Algorithm

- An algorithm is a sequence of steps that shows how to solve a problem step by step.
- For example:
  - Driving direction
  - Cooking recipe
  - Directions for installing a device
Pseudocode Algorithm

- Pseudocode algorithm
  1. Prepare data: pay rates & work hours
  2. Regular wages = base pay rate \times \text{regular hours}
  3. Overtime wages = overtime pay rate \times \text{overtime hours}
  4. Total wages = regular wages + overtime wages
  5. Display the total wages

- The algorithm needs to be translated into C++
- In the beginning of program, define a variable for each value used in the algorithm (How many?)

Program 2-20

```c++
1 // This program calculates hourly wages, including overtime.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7    double regularWages,  // To hold regular wages
8        basePayRate = 10.25,  // Base pay rate
9        regularHours = 40.0,  // Hours worked less overtime
10       overtimeWages,  // To hold overtime wages
11       overtimePayRate = 27.78, // Overtime pay rate
12       overtimeHours = 10,  // Overtime hours worked
13       totalWages;  // To hold total wages
14
15 // Calculate the regular wages.
16    regularWages = basePayRate * regularHours;
17
18 // Calculate the overtime wages.
19    overtimeWages = overtimePayRate * overtimeHours;
20
21 // Calculate the total wages.
22    totalWages = regularWages + overtimeWages;
23
24 // Display the total wages.
25    cout << "Wages for this week are $" << totalWages << endl;
26    return 0;
27 }
```

Program Output

Wages for this week are $1007.0