Basic Elements of C++

Chapter # 2

Instructor: Sadullah Karimi, Msc in CSE

Today Agenda

- How to Learn Programming
- The Basics of a C++ Program
- Comments
- Special Symbols
- Data Types
- Identifiers
- Discover how to use arithmetic operators
- Order of Precedence
- Type Conversion (Casting)

How to Learn Programming

- Learning a programming language is like learning to become a chef or learning to playa musical instrument.
- All three require direct interaction with the tools. You cannotbecome a good chef just by reading recipes. Similarly, you cannot become a musician by reading books about musical instruments.
- The same is true of programming.

The Basics of a C++ Program

- Programming language: A set of rules, symbols, and special words.
- A subprogram or a function is a collection of statements, and when it is activated, or executed, it accomplishes something.
- Some functions, called predefined or standard functions, are already written and are provided as part of the system.
- Every C++ program has a function called main. Thus, if a C++ program has only one function, it must be the function main.

Continue

- If you have never seen a program written in a programming language, the C++ programin Example 2-1 may look like a foreign language.
- To make meaningful sentences in a foreign language, you must learn its alphabet, words, and grammar.
- The same is true of a programming language.
- To write meaningful programs, you must learn the programming language's special symbols, words, and syntax rules.
- The syntax rules tell you which statements (instructions) are legal or valid, that is, which are accepted
- by the programming language and which are not.
- You must also learn semantic rules, which determine the meaning of the instructions.
- The programming language's rules, symbols, and special words enable you to write programs to solve problems.

Comments

- Typically, comments can be used to identify the authors of the program, give the date when the program is written or modified, give a brief explanation of the program, and explain the meaning of key statements in a program.
- for single line comment we use //
- for multi line comment we used */ */

The program in Example 2-1 contains the following comments:

```
//***********************************************************
// Given the length and width of a rectangle, this C++ program
// computes and outputs the perimeter and area of the rectangle.
//***********************************************************
/*
You can include comments that can
occupy several lines.
*/
```

Special Symbols

- The smallest individual unit of a program written in any language is called a token.
- C++’s tokens are divided into special symbols, word symbols, and identifiers.
- Following are some of the special symbols:

+	-	*	/
.	;	?	,
<=	!=	==	>=

Reserved Words (Keywords)

- A second category of tokens is reserved word symbols. Some of the reserved word symbols include the following:
- int, float, double, char, const, void, return

Identifiers

- Identifier: A C++ identifier consists of letters, digits, and the underscore character
- (_) and must begin with a letter or underscore.
- Some identifiers are predefined; others are defined by the user. In the C++ program
- in Example 2-1, cout is a predefined identifier and length is a user-defined identifier.
- Identifiers can be made of only letters, digits, and the underscore character; no other symbols are permitted to form an identifier.

TABLE 2-1 Examples of Illegal Identifiers

Illegal Identifier	Reason	A Correct Identifier
`employee Salary`	There can be no space between `employee` and `Salary`.	`employeeSalary`
`Hello!`	The exclamation mark cannot be used in an identifier.	`Hello`
`one + two`	The symbol + cannot be used in an identifier.	`onePlusTwo`
`2nd`	An identifier cannot begin with a digit.	`second`

Whitespaces: Proper utilization of whitespaces in a program is important. They can be used to make the program more readable.

Data Types

- Data type: A set of values together with a set of allowed operations.
- C++ data types fall into the following three categories:
  - Simple data type
  - Structured data type
  - Pointers

Simple Data Types

TABLE 2-2 Values and Memory Allocation for Simple Data Types

Data Type	Values	Storage (in bytes)
`int`	$-2147483648\ (=-2^{31})$ to $2147483647\ (=2^{31}-1)$	4
`bool`	`true` and `false`	1
`char`	$-128\ (=-2^{7})$ to $127\ (=2^{7}-1)$	1
`long long`	$-9223372036854775808\ (-2^{63})$ to $9223372036854775807(2^{63}-1)$	64

Floating-Point Data Types

TABLE 2-3 Examples of Decimal Numbers in Scientific and C++ Floating-Point Notations

Decimal Number	Scientific Notation	C++ Floating-Point Notation
75.924	$7.5924 * 10^{1}$	7.592400E1
0.18	$1.8 * 10^{-1}$	1.800000E-1
0.0000453	$4.53 * 10^{-5}$	4.530000E-5
-1.482	$-1.482 * 10^{0}$	-1.482000E0
7800.0	$7.8 * 10^{3}$	7.800000E3

- Float: The data type float is used in C++ to represent any decimal number:
  - between -3.4 * 10 ^ 38 and 3.4 * 10 ^ 38.
- The memory allocated for a value of the float data type is four bytes.
- double: The data type double is used in C11 to represent any decimal number:
  - between -1.7 * 10308 and 1.7 * 10308.
- The memory allocated for a value of the double data type is eight bytes.

Discover how to use arithmetic operators

1. Introduction to Arithmetic Operators:
   1)Perform basic mathematical operations.
   2)Common operators: +, -, *, /, %.

Example

- int a = 10, b = 5;
- int sum = a + b; // Addition
- int difference = a - b; // Subtraction
- int product = a * b; // Multiplication
- float quotient = (float)a / b; // Division
- int remainder = a % b; // Modulus

```
#include <iostream.h>
#include <conio.h>
void main() {
    clrscr();
    int a = 10, b = 5;
    cout << "Sum: " << (a + b) << endl;
    cout << "Difference: " << (a - b) << endl;
    cout << "Product: " << (a * b) << endl;
    cout << "Quotient: " << (float)a / b << endl;
    cout << "Remainder: " << (a % b) << endl;
    getch();
}
```

PEMDAS

- P: Parentheses – Solve expressions inside parentheses first.
- E: Exponents – Calculate exponents (powers and roots).
- MD: Multiplication and Division – From left to right.
- AS: Addition and Subtraction – From left to right.

BODMAS

- B: Brackets – Solve expressions inside brackets first.
- O: Orders – Refers to exponents (powers and roots).
- DM: Division and Multiplication – From left to right.
- AS: Addition and Subtraction – From left to right.

Order of Operations:

- PEMDAS/BODMAS Rule:
- Parentheses
- Exponents (not used in basic C++)
- Multiplication and Division (from left to right)
- Addition and Subtraction (from left to right)

Example Expression:

- int a = 10, b = 5, c = 2;
- int result = a + b * c - (a / b); // Evaluates to: 10 + 10 - 2 = 18

Evaluation Steps:

- Parentheses: Calculate (a / b) → 2.
- Multiplication: Calculate b * c → 10.
- Addition/Subtraction: Compute 10 + 10 - 2 → 18.

Understanding the String Data Type in C++

1. What is a String?

   A string is a sequence of characters used to represent text.

   - In Turbo C++, strings are typically handled as character arrays

2. Declaring Strings:

- char myString[50]; // Declaration with a fixed size
- strcpy(myString, "Hello, Turbo C++!"); // Copying a string

Common String Operations:

- Input and Output:
- #include <iostream.h>
- #include <string.h>

- void main() {
- char myString[50];
- cout << "Enter a string: ";
- cin.getline(myString, 50); // Input string with spaces
- cout << "You entered: " << myString;
- }

String Functions:

- strlen(myString): Returns the length of the string.
- strcat(string1, string2): Concatenates two strings.
- strcmp(string1, string2): Compares two strings.

Example Program:

```
#include <iostream.h>
#include <string.h>


void main() {
    char str1[20] = "Hello";
    char str2[20] = "World!";

    strcat(str1, str2); // Concatenates str2 to str1
    cout << str1; // Outputs: HelloWorld!
}
```

Arithmetic Operators, Operator Precedence,and Expressions

Arithmetic Operators:

+ (addition),

- (subtraction or negation),

* (multiplication),

/ (division),

% (mod, (modulus or remainder))

Example:

Given length in inches, we write a program that determines and outputs the equivalent length in feet and (remaining) inches.
Now there are **12** inches in a foot. Therefore, **100** inches equals **8** feet and **4** inches; similarly, **55** inches equals **4** feet and **7** inches.
Note that **100 / 12 = 8** and **100 % 12 = 4**; similarly, **55 / 12 = 4** and **55 % 12 = 7**.
From these examples, it follows that we can effectively use the operators **/** and **%** to accomplish our task.

- // Given length in inches, this program outputs the equivalent
- // length in feet and remaining inch(es).
- #include <iostream>
- using namespace std;
- int main()
- {
- int inches; //variable to store total inches
- inches = 100; //store 100 in the variable inches
- cout << inches << " inch(e s) = "; //output the value of

- //inches and the equal sign
- cout << inches / 12 << " fe et (foot) and "; //output maximum
- //number of feet (foot)
- cout << inches % 12 << " in ch(es)" << endl; //output
- //remaining inches
- return 0;
- }
- Sample run:
- 100 inch(es) = 8 feet (foot) and 4 inch(es)

Order of Precedence

- According to the order of precedence rules for arithmetic operators, *, /, and % are at a higher level of precedence than + and -

```
3 * 7 - 6 + 2 * 5 / 4 + 6
```

means the following:

```
   (((3 * 7) - 6) +((2 * 5) / 4)) + 6
= ((21 - 6) + (10 / 4)) + 6      (Evaluate *)
= ((21 - 6) + 2) + 6             (Evaluate /. Note that this is an integer division.)
= (15 + 2) + 6                   (Evaluate -)
= 17 + 6                         (Evaluate first +)
= 23                             (Evaluate +)
```

Type Conversion (Casting)

- Definition: Type conversion is the process of converting a variable from one data type to another.
- Importance: Necessary for operations between different data types and to avoid data loss.
- **Types of Type Conversion**
- Implicit Conversion: Automatic conversion by the compiler.
- Explicit Conversion (Casting): Manual conversion specified by the programmer.
- static_cast<dataTypeName>(expression)

Implicit Conversion

- Description: Happens automatically without programmer intervention.
- Example:
- int num = 10;
- double dNum = num; // int to double

Explicit Conversion (Casting)

- Description: Requires explicit specification by the programmer.
- Syntax: (target_type) expression
- Example:
  - double dNum = 9.78;
  - int num = (int)dNum; // double to int

static_cast

- Usage: Converts between compatible types.
- Example:
- double dNum = 10.5;
- int num = static_cast<int>(dNum);
-

EXAMPLE 2-9

Expression	Evaluates to
`static_cast<int>(7.9)`	`7`
`static_cast<int>(3.3)`	`3`
`static_cast<double>(25)`	`25.0`
`static_cast<double>(5 + 3)`	`= static_cast<double>(8) = 8.0`

`static_cast<double>(15) / 2`	`= 15.0 / 2`
	(because `static_cast<double> (15) = 15.0`)
	`= 15.0 / 2.0 = 7.5`

`static_cast<double>(15/2)`	`= static_cast<double>(7)` (because `15 / 2 = 7`) `= 7.0`

`static_cast<int>(7.8 +`
`static_cast<double>(15)/2)`
	`= static_cast<int>(7.8 + 7.5)`
	`= static_cast<int>(15.3)`
	`= 15`

`static_cast<int>(7.8 +`
`static_cast<double>(15/2))`
	`= static_cast<int>(7.8 + 7.0)`
	`= static_cast<int>(14.8)`
	`= 14`

- Any question are appreciating