CSE 06131223 ♦ CSE 06131224

Structured Programming

Lecture 7

The Essentials of C Programs (2)

Prepared by

Md. Mijanur Rahman, Prof. Dr.
Dept. of Computer Science and Engineering
Jatiya Kabi Kazi Nazrul Islam University, Bangladesh
www.mijanrahman.com

Contents

THE ESSENTIALS OF C PROGRAMS

- Basic Structure of C Program
- C Tokens
- Data Types
- Variables Declaration
- Operators
- Constant Declaration
- Statements and Expressions
- Input and Output Statements

Operators

- An *operator* is a symbol that instructs C to perform some operation, or action, on one or more operands. An *operand* is something that an operator acts on. In C, all operands are expressions. C operators fall into several categories:
- **List of C++ Operators:**

Type	**Operators**
Assignment Operator	=
Compound assignment Operator	+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |==
Arithmetic Operator	+,-,*,/,%
Increment/Decrement Operator	++,--
Relational Operator	==,>,>=,<,<=,!=
Logical Operator	&&, ||, !
Bitwise Operators	&, |, ^, ~, >>, <<
Conditional Operator	?:

Operators

Assignment Operator:

- **Assignment Operators** that are used to assign the operator on the left the value on the right. The basic assignment operator is the "=" operator.

Compound Assignment Operators:

- These operators are used modify the current value stored in a variable. Some of the compound assignment operators are +=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=.

Operators

Arithmetic Operators in C:

- Arithmetic Operators are used to do basic arithmetic operations like addition, subtraction, multiplication, division, modulus.
- The following table list the arithmetic operators used in C:

Operator	Action
+	Addition
-	Subraction
*	Multiplication
/	Division
%	Modulus

Operators

Increment and Decrement Operators in C:

- The following table list the increment and decrement operators used in C:

Operator	Symbol	Action	Examples
Increment	++	Increments the operand by one	++x, x++
Decrement	--	Decrements the operand by one	--x, x--

Operators

Relational / Comparison Operators:

- **Relational operators** are used to compare two values or expressions to evaluate the relationship. Following table lists the relational operators in C.
- The following table list the relational operators used in C:

Operator	Action
==	Equal to
!=	Not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

Expression	How It Reads	What It Evaluates To
5 == 1	Is 5 equal to 1?	0 (false)
5 > 1	Is 5 greater than 1?	1 (true)
5 != 1	Is 5 not equal to 1?	1 (true)
(5 + 10) == (3 * 5)	Is (5 + 10) equal to (3 * 5)?	1 (true)

Operators

Logical Operators:

- The **logical operators** are used to logically combine, compare Boolean conditions or expressions. The following table lists the operators.
- The following table list the logical operators used in C:

Operator	Action
!	NOT
&&	AND
||	OR

Operators

Bitwise Operators:

- **Bitwise operators** are AND, OR, XOR and NOT used to manipulate data at the bit level by shifting or testing bits.
- The following table lists the bitwise operators in C:

Operator	Action
~	Bitwise NOT
&&	Bitwise AND
||	Bitwise OR
^	XOR
<<	Bitwise Shift Left
>>	Bitwise Shift Right

Operators

Conditional Operator:

- **Conditional operator** is used to return a result based on a expression. This is the only operator that has three operands which also be used instead of "If else" statement for ease of use. Conditional operator is also known as "Ternary Operator".
- The conditional operator is C's only *ternary* operator, meaning that it takes three operands. Its syntax is:

  ***exp1* ? *exp2* : *exp3*;**

- If *exp1* evaluates to true (that is, nonzero), the entire expression evaluates to the value of *exp2*. If *exp1* evaluates to false (that is, zero), the entire expression evaluates as the value of *exp3*.

Operators

Conditional Operator:

- For example, the following statement assigns the value 1 to x if y is true and assigns 100 to x if y is false:

```
x = y ? 1 : 100;
```

- Likewise, to make z equal to the larger of x and y, you could write

```
z = (x > y) ? x : y;
```

- Perhaps you've noticed that the conditional operator functions somewhat like an if statement. The preceding statement could also be written like this:

```
if (x > y)
        z = x;
else
        z = y;
```

Constant

- Constants in C refer to fixed values that do not change during the execution of a program. C supports several types of constants:
  1. Numeric constants
     1. Integer constants. Example: 123, -321, 0, +876
     2. Real constants. Example: 0.00065, -0.95, +345.60, 456.75, 0.76e4, 12e-5, -1.2E-2.
  2. Character constants
     1. Single character constants. Example: ‘A’, ‘x’, ‘9’, ‘;’, ‘ ‘
     2. String constants: Example: “Hello!”, “X”, “2014”.

Constant Declaration

- Like a variable, a *constant* is a data storage location used by your program.
- Unlike a variable, the value stored in a constant can't be changed during program execution.

- C has two types of constants:

  (i) symbolic constants

  (ii) constant variables

Constant Declaration

Constant variables:

- The constant value cannot be changed by the program. A constant variable is declared and initialized in the variable declaration section of the program and cannot be modified thereafter.
- The type of value stored in the constant must also be specified in the declaration.
- For example, an integer constant can be declared as follows:

```
const int size = 100;
```

Constant Declaration

Symbolic constants:

- A *symbolic constant* is a constant that is represented by a name (symbol) in your program. Like a literal constant, a symbolic constant can't change. The actual value of the symbolic constant needs to be entered only once, when it is first defined.
- A ***symbolic constant*** is defined in the preprocessor area of the program and is valid throughout the entire program. A symbolic constant is defined as follows:

  #define N 100

- For example, we can define PI constant value as follows:

  #define PI 3.14159

- This symbolic constant with the name PI is used in the following expression:

```
circumference = PI * (2 * radius);
area = PI * (radius)*(radius);
```

Constant Declaration

Symbolic constants:

- The following rules apply to a **#define** statement which define a symbolic constant:
  1. Symbolic names have the same form as variable names.
  2. No blank space between the pound sign ‘**#**’ and the word **define** is permitted.
  3. ‘**#**’ is the first character in the line.
  4. A blank space is required between #define and symbolic name and between the symbolic name and the constant value.
  5. #define statement must not end with a semicolon.
  6. After definition, the symbolic name should not be assigned any other value.
  7. Symbolic names are NOT declared for data types.
  8. #define statements may appear anywhere in the program but before it is referenced in the program.

Expression and Statement

Expressions:

- An expression is a combination of constants, variables, and operators that are used to denote computations.
- For instance, the following:

  (2 + 3) * 10

  is an expression that adds 2 and 3 first, and then multiplies the result of the addition by 10. (The final result of the expression is 50.)
- Similarly, the expression 10 * (4 + 5) yields 90. The 80/4 expression results in 20.

Expression and Statement

Expressions:

- Here are some other examples of expressions:

Expression	Description
6	An expression of a constant.
i	An expression of a variable.
6 + i	An expression of a constant plus a variable.
exit(0)	An expression of a function call.

Expression and Statement

Statements:

- In the C language, a statement is a complete instruction, ending with a semicolon.
- In many cases, you can turn an expression into a statement by simply adding a semicolon at the end of the expression.
- For instance, the following

```
i = 1;
```

is a statement.

- Here are some other examples of statements:

```
i = (2 + 3) * 10;
i = 2 + 3 * 10;
j = 6 % 4;
k = i + j;
```

Expression and Statement

Statement Blocks:

- A group of statements can form a statement block that starts with an opening brace ‘{‘ and ends with a closing brace ‘}’. A statement block is treated as a single statement by the C compiler.
- For instance, the following

```
for(. . .) {
        s3 = s1 + s2;
        mul = s3 * c;
        remainder = sum % c;
}
```

- is a statement block that starts with { and ends with }. Here **for** is a keyword in C that determines the statement block.

Expression and Statement

Statement Blocks:

- A statement block provides a way to group one or more statements together as a single statement.
- Many C keywords can only control one statement.
- If you want to put more than one statement under the control of a C keyword, you can add those statements into a statement block so that the block is considered one statement by the C keyword.

Input and Output Statements

Input — *scanf*

- Getting a data value from input, i.e., from the keyboard.
- The following statement is used for getting a floating point number from input, i.e., from the keyboard.

```
scanf("%f", &num);
```

num is a variable of float type and %f is used for float.

- For integer number, we use “%d”, for character “%c”, etc.

Input and Output Statements

Output — *printf*

- Providing an output to the user.
- The following statement is used to display the result of a computation.

```
printf(“The average is %f", avg);
```

- In this statement:
  - “The average is %f" is the control string
  - avg is the variable to be printed
  - %f is a conversion specifier indicating that the type of the corresponding variable to be printed is floating-point number.

Sample Programs

Investment Program:

```c
#include<stdio.h>
#include<conio.h>
void main()
{
  int n, year;
  float amount, rate, value;
  printf("Enter amount, rate and year:\n");
  scanf("%f %f %d",&amount, &rate, &n);
  year =0;
  while(year<=n)
  {
    printf("%5d %.2f\n", year, amount);
    value = amount + (rate/100)*amount;
    year = year+1;
    amount= value;
  }
  getch();
}
```

Output:

```
Enter amount, rate and year:
                10000   14   5

                     11400.00
                     12996.00
                     14815.44
                     16889.60
                     19254.15
```

Sample Programs

Program: Area of a circle

```
#include <stdio.h>
#define PI 3.14159

Int main()
{
  float radius, area;
  printf("Enter the radius of a circle: ");
  scanf("%f", &radius);
  area = PI * radius * radius;
  printf("\nArea = %f", area);
  return 0;
}
```

?

THE END