Faculty of Information Technology, University of Moratuwa

BSc. (Hons) in Information Technology

BSc. (Hons) in Artificial Intelligence

Programming Fundamentals – IN1101

Level 1 – Semester 1

Operators and Expressions

When writing programs in any language, arithmetic and logical expressions are often used. These expressions consist of variables, constants and operators. As we already discussed arithmetic and assignment operators, in this lab session, we discuss relational and logical operators and their precedence in C. The operators tell how the variables in an expression are manipulated. The C language is very much richer in operators. Expressions combine the variables and constants to produce new values.

Types of Operators

In C, operators can be classified into various categories based on their utility and action as follows:

- Assignment operators
- Arithmetic operators
- Relational operators
- Logical operators
- Bitwise operators
- Misc operators

Expression

An expression is a combination of variables, constants and operators written according to the syntax of the language. The following are some of the valid expressions.

a+b a+200*w total+x/3

Relational Operators

A relational operator is used to make comparisons between two expressions. The following relational operators are allowed in the C language. Each of these operators compares its left-hand side operand (operands can be variables or expressions) with its right-hand side operand. The whole expression involving the relational operator then evaluates to an integer. It evaluates to zero if the condition is false and non-zero value if it is true.

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

Consider the following program.

```c
#include<stdio.h>
int main() {
    int x = 12, y = 13;
    printf("x = %d\n", x);
    printf("y = %d\n\n", y);
    printf("x > y : %d\n", x > y);
    printf("x >= y : %d\n", x >= y);
    printf("x < y : %d\n", x < y);
    printf("x <= y : %d\n", x <= y);
    printf("x == y : %d\n", x == y);
    printf("x != y : %d\n", x != y);
    return 0;
}
```

Output:

```
x = 12

y = 13


x > y : 0

x >= y : 0

x < y : 1

x <= y : 1

x == y : 0

x != y : 1
```

Logical Operators

Sometimes you might need to ask more than one relational question at once. For example, "If it's 7.00 am and a weekday, ring the alarm".

Logical Operator	Usage
&&	AND
||	OR
!	NOT

Expression	What it evaluates to
(exp1 && exp2)	True only if both exp1 and exp2 are true; false otherwise
(exp1 || exp2)	True if either exp1 or exp2 is true OR both are true; false only if both are false
(!exp1)	False if exp1 is true; true if exp1 is false

You can create expressions that use multiple logical operators. For example, to ask the question "Is X equal to 2, 3 or 4?" you would write,

(x ==2) || (x == 3) || (x == 4)

It's important to be aware, however, that any numeric value is interpreted as either true or false when it is used in a C expression or statement expecting a logical value (that is, a true or false value). The rules are as follows:

- A value of zero represents false
- Any nonzero value represents true

Consider the following C program.

```
#include <stdio.h>
int main() {
    int num = 10;
    printf("%d\n", (num == 10 || num >= 5));
    printf("%d\n", (num >= 5 && num <= 50));
    printf("%d\n", (num != 10 || num >= 5));
    printf("%d\n", (num >= 20 && num != 10));
    return 0;
}
```

Output:

```
1
1
1
0
```

2.1 Increment and Decrement Operators

C language offers two unusual operators for incrementing and decrementing variables. These are ++ and -- operators and are known as increment and decrement operators, respectively. These two operators can only be applied to variables, not in expressions. These operators increase or decrease the value of a variable on which they operate by one.

The syntax for -- and ++ is either one of the following.

(operand) operator: postfix operator

operator (operand): prefix operator

If the prefix is used, the value is incremented or decremented before the variable is used. If the postfix is used, the value increases or decreases after the variable is used.

Consider the program given below.

```c
#include<stdio.h>
int main() {
    int a = 8, b = 2, x = 8, y = 2;
    int m1, m2, n1, n2, c, d;
    m1 = a + (++b);
    return 0;
}
```

In the expression given in the above program, the value of b is incremented by one, and then the result is added to a. This is the use of prefix,

n1=x+(y++);

According to this expression, values of x and y are added first, and then the increment is done. But the incremented value is not assigned to n1, and its value is only the sum of x and y. This is the use of postfix. Now, consider the following two expressions. The behaviours of the increment operator in these two are similar to the above.

m2 = a – (++b);

n2 = x – (y++);

But, in the first expression, the initial value of b is 3 (not 2), because it was incremented by one in the earlier expression (m1). Here, it is again incremented by one before the subtraction. At this moment, the value of b is 4.

Similarly, in the second expression, the value of y is 3 (not 2), as it was incremented by one in the earlier expression (n1). Then, the value of y is incremented again by one after the subtraction.

Consider the following program segment. Run the program and examine the result.

```c
m1 = a + (++b);
m2 = a + (++b);
n2 = x - (y++);
c = m1 / m2;
d = m1 % m2;
m1 = a + (++b);
m2 = a - (++b);
n2 = x - (y++);
c = m1 / m2;
d = m1 / m2;
```

Bitwise Operators

Operator	Usage
~	One's complement
>>	Right shift
<<	Left shift
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR (Exclusive OR)

Misc Operators

Operator	Usage
sizeof()	Returns the size of a variable
&	Returns the address of a variable
*	Pointer to variable
?:	Conditional expression

Conditional 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.

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;
```

Operator Precedence

As in arithmetic, the operators used in a program have their own priority order (precedence). The precedence when caring out these operators is as follows:

Operations	**Symbol**	**Precedence**
Increment & decrement	++ --	1
Logical NOT	!	2
Multiplication, division & modulus	*, /, %	3
Addition and subtraction	+, -	4
Relational less than, less than or equal	<, <=	5
Relational greater than, greater than or equal	>, >=	6
Relational equal, not equal	==, !=	7
Logical AND	&&	8
Logical OR	||	9

The highest operator in C is the increment and decrement operator. Then brackets will always have higher precedence than all these operators. If brackets are used, then the contents inside the innermost pair of brackets will be evaluated first. When several operators have the same precedence in an expression, then it will be evaluated from left to right.

Exercises

I. If the variable x has the value 10, what are the values of x and b after each of the following statements is executed separately?

   b = x++; b = ++x;

II. To what value does the expression 5 + 3 * 8 / 2 + 2 evaluate?

III. What is the output of the following program?

```c
#include <stdio.h>
int main() {
    int i = 3;
    printf("%d", (++i)++);
    return 0;
}
```

IV. Assume that the size of an integer is 4 bytes, and predict the output of the following program.

```c
#include <stdio.h>
int main() {
    int i = 12;
    int j = sizeof(i++);
    printf("%d %d", i, j);
    return 0;
}
```

V. Write the output following C programs.

1.

```c
#include <stdio.h>
void main() {
    const char var = 'A';
    ++var;
    printf("%c", var);
}
```

2.

```c
#include <stdio.h>
void main() {
    int x = 10;
    x += (x++) + (++x) + x;
    printf("%d", x);
}
```

3.

```c
#include <stdio.h>
void main() {
    unsigned short var = 'B';
    var += 2;
    var++;
    printf("var : %c , %d ", var, var);
}
```

4.

```c
#include<stdio.h>
int main() {
    int x = 2;
    (x & 1) ? printf("true") : printf("false");
    return 0;
}
```

VI. Write a C program to check whether the given number is even or odd using conditional operators.

VII. Write a C program to calculate the size of data type or variables using sizeof() method.

VIII. Write a C program to check whether the calendar year is leap year or not using conditional operators.

IX. Write the output of the following C program.

```c
#include<stdio.h>
void main() {
    int a = 0xFFFF;// 0x means value FFFF is in hexadecimal. Binary value is 1111111111111111
    char b = 0xAA;// 0x means value AA is in hexadecimal. Binary value is 10101010
    unsigned char c = 0xAA;// 0x means value AA is in hexadecimal. Binary value is 10101010
    printf("a in hexadecimal= %X, a in decimal= %d\n", a, a);
    printf("b in hexadecimal= %X, b in decimal= %d\n", b, b);
    printf("c in hexadecimal= %X, c in decimal= %d\n", c, c);
    printf("a & b in hexadecimal= %X, a & b in decimal= %d\n", (a & b), (a & b));
    printf("a & c in hexadecimal= %X, a & c in decimal= %d\n", (a & c), (a & c));
}
```