CSE 1201 "Structured Programming Language"

Operators and Expressions

[A. H. M. Saiful Islam]

Operator

- **An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operators are used in programs to manipulate data and variables.**

- C operators can be classified into a number of categories. They include:

1. Arithmetic operators
2. Relation operators
3. Logical operators
4. Assignment operators
5. Increment and decrement operators
6. Conditional operators
7. Bitwise operators
8. Special operators

- An expression is a sequence of operands and operators that reduces to a single value.

For example, 10 +15 is an expression whose value is 25.

Arithmetic operators

- C provides all the basic arithmetic operators. They are listed in the table.

Operators	Meaning
+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
%	Modulo division

Arithmetic operators

- The operators +, -, *, / all work the same way as they do in other language.

- These can operate in any built-in data type allowed in C.
- The unary minus operator, in effect, multiplies its single operand by-1 . therefore, a number preceded by a minus sign changes its sign.

- Integer division truncates any fractional part. The modulo division operation produces the remainder of an integer division. **Examples of use of arithmetic operators are :**

- a – b a + b
- a * b a / b
- a % b -a * b

- here **a** and **b** are variables and known as **operands**.

- The modulo division operator cannot be used on floating point data.

Integer arithmetic

- When both the operands in a single arithmetic expression such as a + b are integers, the expression is called an **integer expression** and the operation is called **integer arithmetic**.

- Integer arithmetic always yields an integer value. The largest integer value depends on the machine.

- In the above example, if a and b are integers, then **for a = 14 and b = 4** we have the following results:
- a - b = 10
- a + b = 18
- a * b = 56
- a / b = 3(decimal part truncated)
- a % b = 2 (remainder of division)

Integer arithmetic

- During integer division, if both the operands are of the same sign, the result is truncated towards zero. If one of them is negative, the direction of truncation is implementation dependant.

That is, **6/7= 0 and -6/-7 = 0**

- **But -6/7may be zero or -1. (Machine dependant).**

- Similarly, during modulo division, the sign of the result is always the sign of the first operand(the dividend). That is
- -14 % 3 = - 2
- -14 % - 3 = - 2
- 14 % - 3 = 2

The program in Fig.3.1 shows the use of integer arithmetic to convert a given number of days into months and days.

PROGRAM TO CONVERT DAYS TO MONTHS AND DAYS

- **Program**

```
main ()
{
   int   months, days ;

   printf("Enter days\n") ;
   scanf("%d", &days) ;

   months = days / 30 ;
   days   = days % 30 ;

   printf("Months = %d   Days = %d",
  months, days) ;
}
```

```
Enter days
265
Months = 8   Days = 25

 Enter days
364
Months = 12   Days = 4

Enter days
45
Months = 1   Days = 15
```

Fig. 3.1 ***Illustration of integer arithmetic***

Real Arithmetic

- An arithmetic operation involving only real operands is called real arithmetic.
- A real operand may assume values either in decimal or exponential notation.

- Since floating point values are rounded to the number of significant digits permissible, the final value is an approximation of the correct result.
- If x,y and z are floats, then we will have

- x = 6.0 / 7.0 = 0.857143
- y = 1.0 / 3.0 = 0.333333
- z = 2.0 / 3.0 = 0.666667

- **The operators % cannot be used with real operands.**

Mixed-mode arithmetic

- When one of the operands is real and the other is integer, the expression is called a mixed- mode arithmetic expression.
- If either operand is of the real type , then only the real operation is performed and the result is always a real number. Thus
- 15 / 10.0 = 1.5
- Whereas 15/10 = 1

Relation operators

C supports six relational operators .

Operator	Meaning
<	is less than
<=	Is less than or equal
>	Is greater than
>=	Is greater than or equal
==	Is equal to
!=	Is not equal to

Output ???

```
#include<stdio.h>
   int main()
   {
          int a=2,b=7,c=10;
          c = a == b;
           printf("%d",c);
            return 0;
}
```

Logical operators

- In addition to the relational operator, C has the following three logical operators.

• &&	meaning logical	AND
• ||	meaning logical	OR
• !	meaning logical	NOT

Output ???

```
int main()
{
    int num1 = 30;
    int num2 = 40;

    if(num1>=40 || num2>=40)
        printf("Or If Block Gets Executed\n");
    if(num1>=20 && num2>=20)
        printf("And If Block Gets Executed\n");
    if( !(num1>=40))
        printf("Not If Block Gets Executed\n");
    return(0);
}
```

Assignment operators

- Assignment operators are used to assign the result of an expression to a variable.
- C has a shorthand assignment operators of the form:

v op = exp;

v = variable,

exp = an expression

op = C arithmetic operator

- **v op = exp;** is equivalent to **v= v op (exp);**

- Consider the example, x + = y + 1;

this is same as the statement

x = x + (y+1);

Assignment operators

Shorthand assignment operators	
Statement with simple assignment Operator	**Statement with shorthand operator**
a = a + 1	a + = 1
a = a - 1	a - = 1
a = a *(n+ 1)	a * = (n+ 1)
a = a / (n+ 1)	a / = (n+ 1)
a = a % b	a % = b

USE OF SHORTHAND OPERATORS

Program

```
#define    N   100
#define    A   2

main()
{
    int  a;

    a  =  A;
    while( a < N )
    {
        printf("%d\n", a);
        a *= a;
    }
}
```

- **Output**

```
2
4
16
```

Fig. 3.2 Use of shorthand operator *=

Increment and decrement operators

- **C allows two very useful operators. These are the increment and decrement operators:**

++ and **- -**

- The operator ++ adds 1 to the operands, while - - subtracts 1. Both are unary operators and take the following form:

- ++m; m++
- --m; m- -
- ++m is equivalent to m = m+1;
- --m is equivalent to m = m – 1;

- We use the increment and decrement statements in **for** and **while** loop extensively.

Rules for ++ and -- operators

- Increment and decrement operators are unary operators and they require variable as their operands.
- When postfix ++ (or --) is used with a variable in an expression, the expression is evaluated first using the original value of the variable and then the variable is incremented (or decremented) by one.

Rules for ++ and -- operators

- When prefix ++ (or --) is used in an expression, the variable is incremented (or decremented) first and then the expression is evaluated using the new value of the variable.
- Increment and decrement operators are unary operators and they require variable as their operands.

Conditional operators

- A ternary operator pair “? :” is available in C to construct conditional expression of the form

exp1 ? exp2: exp3

where exp1, exp2, exp3 are expressions.

- The operator ?: works as follows: exp1 is evaluated first. If it is nonzero (true), then the expression exp2 is evaluated and becomes the value of expression. If exp1 is false, exp3 is evaluated and its value becomes the value of the expression.

Output ???

```
#include<stdio.h>
   int main()
  {
    int num;
    printf("Enter the Number : ");
    scanf("%d",&num);

   (num%2==0)?printf("Even"):printf("Od
   d");
    }
```

Bitwise operators

- C has a distinction of supporting special operators known as bitwise operators for manipulation of data at bit level.
- The operations are used for testing the bits, or shifting them right or left.
- Following table lists the bitwise operators and their meanings.

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	Shift left
>>	Shift right

Output:

```
#include <stdio.h>
int main()
{
    int a=5,b=3;
    printf("Output=%d",a&b);
    return 0;
}
```

Output???

```
#include <stdio.h>
int main()
{
    int num=2,i=1;
   printf("Right shift by %d: %d\n",i, num >> i);
    printf("\n");
   printf("Left shift by %d: %d\n",i,num << i);
    return 0;
}
```

1
4
????

Special operators

- C supports some special operators of interest such as comma operator, **sizeof** operator, pointer operators (& and *) and member selection operators(. and ->).

The Comma Operator

- The comma operator can be used to link the related expressions together. A comma-linked list of expressions are evaluated left to right and the value of right-most expression is the value of the combined expression.

For example, the statement

value = (x = 10, y = 5, x+y);

first assign the value 10 to x, then assigns 5 to y, and finally assign 15 (i.e. 10 + 5) to value. Since comma operator has the lowest precedence of all operators, the parentheses are necessary.

Output ???

```
#include<stdio.h>
    main()
   {
    int x;
    x=10,20,30;
    printf("%d",x);
    }
```

Output ???

```
#include <stdio.h>
main()
{
printf("Computer","Programming");
}
```

computer ????
programming ????
computer programming ????

Output ???

```
#include<stdio.h>
int main()
{
        int i;
        i = (1,2,3);
        printf("i:%d\n",i);
        return 0;
}
```

The sizeof operator

- The sizeof is a compile time operator and when used with an operand, it returns the number of bytes the operand occupies. The operand may be a variable, a constant or a data type qualifier.

- **Examples:**

- **m = sizeof (sum) ;**
- **n = sizeof (long int) ;**
- **k = sizeof (235L) ;**

- The sizeof operator is normally used to determine the lengths of arrays and structures when their sizes are not known to the programmers. It is also used to allocate memory space dynamically to variables during execution of a program.

Output???

```
main()
{
   int a ;
   short b;
   double c;

   printf("Line 1 - Size of variable a = %d\n", sizeof(a) );
   printf("Line 2 - Size of variable b = %d\n", sizeof(b) );
   printf("Line 3 - Size of variable c= %d\n", sizeof(c) );
   }
```

4
2
8

Arithmetic Expressions

- An arithmetic expression is a combination of variables, constants, and operators arranged as per the syntax of the language. To remember, C does not have an operator for exponentiation.

Expression	
Algebraic Expression	**C Expression**
a x b – c	**a * b - c**
(m + n) (x +y)	**(m + n)* (x +y)**
(ab/c)	**a*b/c**
$3x^2 + 2x + 1$	**3*x*x + 2*x + 1**
(x/y) + c	**x/y + c**

Evaluation of Expressions

- Expressions are evaluated using an assignment statement of the form
- **variable = expression;**

when the statement is encountered , the expression is evaluated first and the result then replaces the previous value of the variable on the left hand side. All variables used in the expression must be assigned values before the evaluation is attempted.

- Examples:
- x = a * b – c ;
- y = b / c * a ;
- z = a – b/ c + d ;
- The blank space around an operator is optional and adds only to improve readability.

Priority	Operators	Description
1st	* / %	multiplication, division, modular division
2nd	+ -	addition, subtraction
3rd	=	assignment

EVALUATION OF EXPRESSIONS

- **Program**

```
main()
{
    float  a, b, c, x, y, z;

    a = 9;
    b = 12;
    c = 3;

    x = a - b / 3 + c * 2 - 1;
    y = a - b / (3 + c) * (2 - 1);
    z = a -(b / (3 + c) * 2) - 1;

    printf("x = %f\n", x);
    printf("y = %f\n", y);
    printf("z = %f\n", z);
}
```

- **Output**

```
x = 10.000000
y = 7.000000
z = 4.000000
```

Fig.3.4 Illustrations of evaluation of expressions

In Fig.3.3, the program employs different kinds of operators. The results of their evaluation are also shown for comparison.

```
main()
{
    int a, b, c, d;

    a = 15;
    b = 10;
    c = ++a - b;

    printf("a = %d  b = %d  c = %d\n",a, b, c);

    d = b++ +a;

    printf("a = %d  b = %d  d = %d\n",a, b, d);

   printf("a/b = %d\n", a/b);
   printf("a%%b = %d\n", a%b);
   printf("a *= b = %d\n", a*=b);
   printf("%d\n", (c>d) ? 1 : 0);
   printf("%d\n", (c<d) ? 1 : 0);
}
```

Output

```
a = 16  b = 10  c = 6
a = 16  b = 11  d = 26
a/b = 1
a%b = 5
a *= b = 176
0
1
```

Fig. 3.3 Further illustration of arithmetic operators