CHAPTER

OPERATORS AND EXPRESSIONS

Constants and Variables are the basic data objects manipulated in a program. Declarations list the variables to be used, their type and their initial values. An operator is a symbol that specifies an operation to be performed on them. The data items that operators act upon are called operands. Expressions combine variables and constants to produce new values. An expression is a sequence of operands and operators that reduce to a single value.

For example: 2+3, it is an expression whose value is 5. The value can be any type other than a void. Some operators require two operands. They are called binary operators. Some operators require one operand. They are called unary operator.

C includes a large number of operators which fall into different categories. They are:

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Assignment Operator
5. Increment & decrement Operator
6. Conditional Operator
7. Bitwise Operator
8. Special Operator

8.1 ARITHMETIC OPERATORS

To solve most programming problems, arithmetic operations are performed by writing arithmetic expressions. The basic arithmetic operators are addition(+), subtraction(–), multiplication(*), division(/) and modulo-division(%). Each operator manipulates two operands which may be constants, variables, or other arithmetic expressions. The arithmetic operators may be used with int or double data type operands. On the other hand, the remainder operator, also known as modulus operator, can be used with integer operands to find the remainder of the division.

- When both operands in an arithmetic expression are integers, the expression is called an integer expression, and the operation is called integer arithmetic.
- When both operands in an arithmetic expression are floating point numbers, the expression is called a floating point expression, or real expression, and the operation is called floating point arithmetic or real arithmetic.
- The result of integer arithmetic always has an integer value.
- The remainder operator requires both operands be integers and the second operand be non-zero.
- The division operator requires the second operand be non-zero.
- The result of integer division results in a truncated quotient (i.e., decimal portion of the quotient will be dropped).
- If a division operation is carried out with two floating-point numbers or with one floating-point number and one integer, the result will be a floating point quotient. For example, 9.5/5 = 1.9
- If one or both operands are negative, then the arithmetic operations will result in values whose signs are determined by the usual rules of algebra.
- The result of the remainder operation always gets the sign of the first operand.
- Operands that differ in type may undergo type conversion before the expression takes on its final value. In general, the final result will be expressed in the highest precision possible, consistent with the data type of the operands.

Operand 1	Operand 2	Result
short	int	int
int	float	float
double	float	double
double	int	double
int	long	long

EXAMPLE PROGRAM:

```
#include<stdio.h>
void main()
{
int a,b,c,d,e,f;
      a=10;
      b=7;
      c=a+b;
      d=a-b;
      e=a*b;
      f=a/b;
      printf("\n%d",c);
      printf("\n%d",d);
      printf("\n%d",e);
```

```
      printf("\n%d",f);
}
```

Output

```
17
3
70
1
```

8.2 RELATIONAL OPERATORS

Relational operators are used to compare values, typically in a conditional control statement. Operands may be variables, constants or expressions. The various relational operators are

Relational Operators	Meaning
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
Equality Operators	**Meaning**
==	Equal to
!=	Not equal to

The conditions are checked using relational operators. They return either 1 or 0 as true value or false value respectively. The result of the expression is of type integer, since true is represented by the integer value 1 and false is represented by the value 0.

The double equal to sign '==', used to compare equality, is different from the single equal to '=' sign which is used as an assignment operator.

EXAMPLE PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
printf("\nCondition   : Return Value");
printf("\n5!=5        : %d",(5!=5));
printf("\n10<11       : %d",(10<11));
printf("\n12>9        : %d",(12>9));
printf("\n55>=90      : %d",(55>=90));
printf("\n45<=91      : %d",(45<=91));
printf("\n78==78      : %d",(78==78));
getch();
}
```

```
Output
Condition   : Return Value
5!=5        : 0
10<11       : 1
12>9        : 1
55>=90      : 0
45<=91      : 1
78==78      : 1
```

8.3 LOGICAL OPERATORS

C has three logical operators for combining logical values and creating new logical values.

Operator	Meaning	Example	Return Value (Result)
&&	Logical AND	(4<6)&&(8==9)	0
||	Logical OR	(7<9)||(3<1)	1
!	Logical NOT	!(29>89)	1

NOT: The NOT operator (!) is a unary operator (operator that acts upon single operand). It changes a true value (1) to false (zero) and a false value (0) to true (1).

AND: The AND operator(&&) is a binary operator. Its result is true only when both operands are true; otherwise it is false.

OR: The OR operator(||) is a binary operator. Its result is false only when both the operands are false. Otherwise it is true.

EXAMPLE PROGRAM:

```
    #include<stdio.h>
    #include<conio.h>1
void main()
{
clrscr();
printf("\nCondition      : Return Value");
printf("\n5!=5&&6<9      : %d",(5!=5)&&(6<9));
printf("\n10<11||10==10  : %d",(10<11)||(10==10));
printf("\n!(12>9)        : %d",!(12>9));
getch();
}
```

```
Output
 Condition       : Return Value
 5!=5&&6<9       : 0
 10<11||10==10   : 1
 !(12>9)         : 0
```

8.4 ASSIGNMENT OPERATORS

It is used to assign a value of an expression on the right side of it to the variable on the left of it. The assignment expression evaluates the operand on the right side of the operator (=) and places its value in the variable on the left.

Syntax

Variable=expression (or) value
a=8; /* integer value is assigned to variable a*/
area=length * width /* value of length * width is assigned to variable area */

Important points regarding assignment operator:

- ❖ If the two operands in an assignment expression are of different data types, then the value of the expression on the right side of the assignment operator will automatically be converted to the type of the identifier on the left of assignment operator.
  - A floating value may be truncated if it is assigned to an integer identifier.
  - A double value may be rounded if it is assigned to a floating-point identifier.
  - An integer value may be changed if it is assigned to a short integer identifier or to a character identifier.

Compound Assignment

Operator	Example	Meaning
+ =	x+=y	x=x+y
– =	x–=y	x=x–y
* =	X*=y	x=x*y
/ =	x/=y	x=x/y

EXAMPLE PROGRAM

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b;
clrscr();
a=10; //Assignment Statement
b=6;
a+=b; // CompoundStatement
b-=a;
printf("\nThe value of a is %d",a);
printf("\nThe value of b is %d",b);
getch();
}
```

Output

```
The value of a is 16
The value of b is -10
```

8.5 INCREMENT & DECREMENT OPERATORS

The increment operator (++) adds 1 to the operand, whereas the decrement operator (--) subtracts 1 from the operand.These operators are called Unary operators. The increment and decrement operators can be used in two different ways, depending on whether the operator is written before or after the operand. For example,

a++ ; /* operator is written after the operand*/
++a ; /* operator is written before the operand */

If the operator precedes the operand, then it is called pre-increment operator and the operand will be changed in value before it is used for its intended purpose within the program. On the other hand, if the operator follows the operand, then it is called post-increment operator and the value of the operand will be changed after it is used.

Operator	Meaning
++x	Pre-increment
X++	Post increment
--y	Pre-decrement
Y--	Post decrement

EXAMPLE PROGRAM:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b;
clrscr();
a=10;
b=6;
printf("\nThe value of a is %d",a++);
printf("\nThe value of a is %d",++a);
printf("\nThe value of b is %d",b--);
printf("\nThe value of b is %d",--b);
getch();
}
```

Output

```
The value of a is 10
The value of a is 12
The value of b is 6
The value of b is 4
```

8.6 CONDITIONAL OPERATORS

Simple conditional operations can be carried out with the conditional operator. It is a ternary operator since it takes three operands.It checks the condition and executes the statements depending on the condition.The general form of conditional expression is,

Syntax

Test expression?exp1:exp2

If the condition is true, it evaluates the first expression otherwise it evaluates the second expression. Conditional operator works as follows:

- The test expression is implicitly converted to Boolean expression and then it is evaluated.
- If the result of the test expression is true (1), the expression 1 is evaluated and the conditional expression takes on value of expression 1.
- If the result of the test expression is false (0), the expression 2 is evaluated and the conditional expression takes on value of expression 2.

EXAMPLE PROGRAM

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,big;
clrscr();
a=10;
b=6;
big=a>b?a:b;
printf("\nThe biggest value is %d",big);
getch();
}
```

Output

```
The biggest value is 10
```

8.7 BITWISE OPERATORS

The bitwise operators are the bit manipulation operators.They are used to manipulate the data at bit level. They operate on integers and characters but not on floating-point numbers or numbers having double data type.

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	One's Complement
>>	Right Shift
<<	Left Shift

EXAMPLE PROGRAM

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,c,d,e;
clrscr();
a=10;
b=6;
c=a&b;
printf("\nThe value of c(AND) is %d",c);
d=a|b;
printf("\nThe value of d(OR) is %d",d);
e=a^b;
printf("\nThe value of e(XOR) is %d",e);
getch();
}
```

Output

```
The value of c(AND) is 2
The value of d(OR) is 14
The value of e(XOR) is 12
```

8.8 SPECIAL OPERATORS

C Language supports some special operators. They are:

Operator	Meaning
,	Comma operator
& and *	Pointer operator
sizeof	sizeof operator
cast	type cast

The comma operator is used to separate the variables. The pointer operator is used to get the address of the operand.

sizeof operator is used to calculate the size of various data types. The sizeof operator looks like a function but it is actually an operator that returns the length in bytes. It is used to get size of space of any data element/ data type occupied in memory. If type name is used, it always needs to be enclosed in parentheses, whereas variable name can be specified with or without parentheses.

TYPE CONVERSION – EXAMPLE PROGRAM

```
#include<stdio.h>
main()
{
int a=8,sum;
char c='4';
printf("Character c is %c \n",c);
printf("Numeric c is %d \n",(int)c);
sum=a+(int)c;
printf("Sum is %d \n",sum);
printf("integer %d\n", sizeofi); /* variable name*/
printf("character %d\n", sizeof c); /* variable name*/
printf("integer %d\n", sizeof(int)); /* type name enclosed in parenthesis*/
printf("float %d\n", sizeof(float));
printf("double %d\n", sizeof(double));
}
```

Output

```
integer       2
character     1
integer       2
float         4
double        8
```

The sizeof operator is used whenever the size of any datatype is needed. It is also used for dynamic memory allocation.

Type Conversion (Cast Operator)

Rather than letting the compiler implicitly convert data, the data can be converted explicitly from one type to another using cast operator. To cast data from one type to another it is necessary to specify type in parenthesis before the value that is to be converted.

For example, to convert the integer variable **count** to a float, the expression is given as below:

(float)count;

Since cast operator is a unary operator, the binary expression to be casted must be put in parenthesis to get the correct conversion. For example,

(float)(a+b);

One use of cast operator is to ensure that the result of a division is a floating point number,

EXAMPLE PROGRAM

```
#inlcude<stdio.h>
main()
{
int a=8, sum;
char c='4';
printf("character c is %c\n", c);
printf("Numeric C is %d\n",(int)c);
sum=a+(int)c;
printf("sumis %d\n",sum);
}
```

Output

```
Character C is 4
Numeric C is 52
Sum is 60
```

8.9 OPERATOR PRECEDENCE &ASSOCIATIVE OF OPERATORS

Operator precedence describes the order in which C evaluates different operators in a complex expression. For example, in the expression a=2 +b*2, which happens first, the addition or multiplication? The operator with the highest precedence is evaluated first.

The table below lists the operator precedence. If the two operators are on the same level in the table, the column on the right tells us the order they will be evaluated in. This is known as associativity property of an operator. In the above example, multiplication is performed before the addition.

The arithmetic operators are evaluated from the left to right using the precedence of operators, when the expression is written without the parenthesis.

Operator	Description	Associativity
( )	Parentheses (function call) (see Note 1)	left-to-right
[ ]	Brackets (array subscript)	
.	Member selection via object name	
–>	Member selection via pointer	
++ --	Postfix increment/decrement (see Note 2)	
++ --	Prefix increment/decrement	right-to-left
+ -	Unary plus/minus	
! ~	Logical negation/bitwise complement	
(*type*)	Cast (convert value to temporary value of type)	
*	Dereference	
&	Address (of operand)	
sizeof	Determine size in bytes on this implementation	

(Continued)

Operator	Description	Associativity
* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <=	Relational less than/less than or equal to	left-to-right
> >=	Relational greater than/greater than or equal to	
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
|	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
| |	Logical OR	left-to-right
? :	Ternary conditional	right-to-left
=	Assignment	right-to-left
+= -=	Addition/subtraction assignment	
*= /=	Multiplication/division assignment	
%= &=	Modulus/bitwise AND assignment	
^= |=	Bitwise exclusive/inclusive OR assignment	
<<= >>=	Bitwise shift left/right assignment	
,	Comma (separate expressions)	left-to-right

Rules for evaluation of expression:

1. Any expression within the parenthesis is evaluated first.
2. Arithmetic expression is evaluated from left to right using the rule of precedence.
3. Within the parenthesis, highest precedence operator is evaluated first.
4. If the operators have the same precedence, associativity is to be applied.
5. If the parenthesis is nested, the innermost sub-expression is evaluated first.

Example

Given expression 45 + 8 – 5 * 7

Step 1: 45 + 8 – 5 * 7

Step 2: 45 + 8 – 35

Step 3: 53 – 35

Step 4: 18

The result is 18.