UNIT-II

Module – 2: (Arithmetic expressions & Conditional Branching)

Arithmetic expressions and precedence: Operators and expression using numeric and relational operators, mixed operands, type conversion, logical operators, bit operations, assignment operator, operator precedence and associativity.

Conditional Branching: Applying if and switch statements, nesting if and else, use of break and default with switch.

Operators

Expression: An expression is a sequence of operands and operators that reduces to single value

Example: 10+25 is an expression whose value is 35
C operators can be classified into a no. of categories.

They include:

1. Arithmetic
2. Relational
3. Logical
4. Assignment
5. Increment and Decrement
6. Conditional
7. Bitwise
8. Special

ARITHMETIC OPERATORS: C provides all the basic arithmetic operators, they are +, -, *, /, %. Integer division truncates(shorten) any fractional part. The modulo division produces the remainder of an integer division.

Ex: a + b
a – b
a * b
a / b
a % b

Here a and b are variables and are known as operands.

Note: 1. Modulo (%) cannot be used for floating and double data types.

2. C does not have an operator for exponentiation.

INTEGER ARITHMETIC:

- When the operands in an expression are integers then the expression is an integer expression and the operation is called integer arithmetic.
- This always yields an integer value.
- For E.g. a = 14 and n = 4 then

a - b = 10

a + b = 18

a * b = 56

a / b = 3

- 14 % 3 = -2

a % b = 2

-14 % - 3 = 2

Note: During modulo division, the sign of the result is always the sign of the first operand (the dividend)

Example: Write a program to illustrate the use of all Arithmetic operator

```
#include<stdio.h>
#include<conio.h>
void main ( )
{
int sum,prod,sub, div,mod,a, b ;
clrscr();
printf("Enter values of a, b :");
scanf("%d %d", & a, & b);  sum
= a+b ;
printf("Sum = %d", sum);
sub = a-b;
printf("\tSubtract = %d", sub);
prod = a*b;
printf("\tProduct = %d", prod);
div = a/b;
printf("\tDivision = %d", div);
mod = a % b ;
printf("\tMod = %d",a % b);
getch();
}
```

Output:

```
Enter values of a, b:
10
4
Sum=14          Subtract=6      Product=40
Division=2      Mod=2
```

Real Arithmetic / Floating Pont Arithmetic:

Floating Point Arithmetic involves only real operands of decimal or exponential notation. If x, y & z are floats, then

x = 6.0/7.0 = 0.857143

y = -1.0/3.0 = 0.333333

z = 3.0/2.0 = 1.500000

% cannot be used with real operands

Mixed mode Arithmetic: When one of the operands is real and the other is integer the expression is a mixed mode arithmetic expression.

Example: 15/10.0 = 1.500000

15/10 = 1

10/15 = 0

-10.0/15 = -0.666667

RELATIONAL OPERATORS:

Relational operators are used to find the relation between two variables. i.e. to compare the values of two variables in a C program.

S.no	Operators	Example	Description
1	>	x > y	x is greater than y
2	<	x < y	x is less than y
3	>=	x >= y	x is greater than or equal to y
4	<=	x <= y	x is less than or equal to y

5	==	x == y	x is equal to y
6	!=	x != y	x is not equal to y

Example: Write a program in C to use various relational operators and display their return values.

```
#include <stdio.h>
#include<conio.h>
void main()
{
  int a = 21;
  int b = 10;
  int c ;

  if( a == b )
  {
    printf("Line 1 - a is equal to b\n" );
  }
  else
  {
    printf("Line 1 - a is not equal to b\n" );
  }

  if ( a < b )
  {
    printf("Line 2 - a is less than b\n" );
  }
  else
  {
    printf("Line 2 - a is not less than b\n" );
  }

  if ( a > b )
  {
    printf("Line 3 - a is greater than b\n" );
  }
  else
  {
    printf("Line 3 - a is not greater than b\n" );
  }
  /* Lets change value of a and b */
  a = 5;
  b = 20;
  if ( a <= b )
  {
    printf("Line 4 - a is either less than or equal to  b\n" );
  }
  if ( b >= a )
  {
    printf("Line 5 - b is either greater than  or equal to b\n" );
  }
}
```

Output:

```
Line 1 - a is not equal to b
Line 2 - a is not less than b
Line 3 - a is greater than b
Line 4 - a is either less than or equal to b
Line 5 - b is either greater than or equal to b
```

LOGICAL OPERATOR:

Logical Operators are used when we want to test more than one condition and make decisions. Here the operands can be constants, variables and expressions.

S.no	Operators	Name	Example	Description
1	&&	logical AND	(x>5)&&(y<5)	It returns true when both conditions are true
2	||	logical OR	(x>=10)||(y>=10)	It returns true when at-least one of the conditions is true
3	!	logical NOT	!((x>5)&&(y<5))	It reverses the state of the operand "((x>5) && (y<5))" If "((x>5) && (y<5))" is true, logical NOT operator makes it false

Where True=1 and False=0

Operator	Condition 1	Condition 2	Final Output
AND	0	0	0
	0	1	0
	1	0	0
	1	1	1
OR	0	0	0
	0	1	1
	1	0	1
	1	1	1
NOT	0	-	1
	1	-	0

Example:

```
#include<stdio.h>
void main()
{
  int num1 = 30;
 int num2 = 40;

if(num1>=40 || num2>=40)
   {
      printf("\nOr If Block Gets Executed");
   }
   if(num1>=20 && num2>=30)
   {
       printf("\nAnd If Block Gets Executed");
   }
   if( !(num1>=40))
   {
     printf("\nNot If Block Gets Executed");
```

Output

```
Or If Block Gets Executed
And If Block Gets Executed
Not If Block Gets Executed
```

```
}
getch();
}
```

ASSIGNMENT OPERATOR:

- Used to assign the result of an expression to a variable.
- = is the assignment operator.
- In addition, C has a set of short hand assignment operators of the form

int a=10;

Example: Short hand Expression

x+=2 ➔ x=x+2

x*=2 ➔ x=x*2

x-=2 ➔ x=x-2

x/=2 ➔ x=x/2

INCREMENT AND DECREMENT OPERATORS: ++ and --

The Operator + + adds 1 to the operand while -- subtracts 1, both are unary operators.

++x or x ++ == > x+=1 == > x=x+1

. -- x or x- - == > x-=1 == > x=x-1

- ✓ ++a or a++ is equivalent to a=a+1.
- ✓ The **difference between pre-increment and post-increment** lies in the point at which the value of their operand is incremented.
  - I. In case of pre-increment operator, firstly the value of its operand is incremented and then it is used for the evaluation of expression.
  - II. In case of post-increment operator, the value of operand is used first for the evaluation of the expression and after its use, the value of the operand is incremented.
  - III. Increment operator is a token i.e. one unit. There should be no white space character between two '+' symbols. If white space is placed between two '+' symbols, they become two unary plus (+) operators.
- ✓ --a or a-- is equivalent to a=a-1.
- ✓ The **difference between pre-decrement and post-decrement** lies in the point at which the value of their operand is decremented.
  - I. In case of pre-decrement operator, firstly the value of its operand is decremented and then used for the evaluation of the expression in which it appears.
  - II. In case of post-decrement operator, firstly the value of operand is used for the evaluation of the expression in which it appears and then, its value is decremented.

Example:

```
#include <stdio.h>
void main()
{
  int a = 21;
  int c ;
  c = a++;
  printf("Line 1 - Value of a++ is :%d",c);
  printf("Line 2 - Value of a is :%d",a);
  c = ++a;
 printf("Line 3 - Value of ++a is :%d",c);
  }
```

Output

Line 1 - Value of a++ is: 21

Line 2 - Value of a is: 22

Line 3 - Value of ++a is: 23

CONDITIONAL OPERATOR:

- It is used to check a condition and Select a Value depending on the value of the condition.

  Variable = (condition)? Value 1: Value 2;

- If the Value of the condition is true then Value 1 is e valued assigned to the Variable, otherwise Value2.

Example:

```
#include <stdio.h>
void main()
{
  int a=10,b,c;
  clrscr();
  b=(a == 1) ? 20: 30;
  c=(a == 10) ? 20: 30;
  printf( "Value of b is %d\n",b);
  printf( "Value of c is %d\n",c);
  getch();
}
```

Output

Value of b is 30

Value of c is 20

BITWISE OPERATOR:

- These operators are used to perform operations at binary level i.e. bitwise.
- These operators are used for testing the bits, or shifting them right or left.
- These operators are not applicable to float or double. Following are the Bitwise operators with their meanings.

Operator	Description	Example (A=60 and B=16)
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 16 which is 00010000
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 60 which is 00111100
^	Bitwise Exclusive – OR	Binary XOR Operator copies the bit if it is set in one operand but not both. (A^B) will give 44 which is 00101100
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 11000011 in 2's complement form due to a signed binary number
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000

>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111

```
#include <stdio.h>
void main()
{
  int A = 60,B = 16,C;
  C = A & B;
  printf("\nLine 1 - Value of C is %d\n", C );
  C= A | B;
  printf("\n Line 2 - Value of C is %d\n", C );
  C = A ^ B;
  printf("\nLine 3 - Value of C is %d\n",C );
  C = ~A;
  printf("\nLine 4 - Value of C is %d\n", C );
  C = A << 2;
  printf("\nLine 5 - Value of C is %d\n",C );
  C = A >> 2;
  printf("\n Line 6 - Value of C is %d\n", C);
}
```

Output

Line 1 - Value of C is 16

Line 2 - Value of C is 60

Line 3 - Value of C is 44

Line 4 - Value of C is -61

Line 5 - Value of C is 240

Line 6 - Value of C is 15

Sizeof Operator: It returns the size of a variable in bytes.

```
#include <stdio.h>
void main()
{
  int a = 4;
  float 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) );
}
```

Output

Line 1 - Size of variable a= 2
Line 1 - Size of variable b= 4
Line 1 - Size of variable c= 8

OPERATOR – PRECEDENCE:

Precedence is nothing but priority that indicates which operator has to be evaluated first when there are more than one operator.

OPERATOR-ASSOCIATIVITY:

- When there is more than one operator with same precedence [priority] then we consider associativity, which indicated the order in which the expression has to be evaluated.
- It may be either from Left to Right or Right to Left.

Sr.No.	Operator	Name of Operator	Associativity	Rank
1.	() []	Function call Array subscript	**L→R**	1
2.	+ - ++ -- & * sizeof (type)	Unary plus Unary minus Increment Decrement Address-of Deference sizeof Type cast (conversion)	**R→L**	**2**
3.	***** **/** **%**	Multiplication Division Modulus	**L→R**	**3**
4.	+ -	Addition Subtraction	**L→R**	**4**
5.	<< >>	Left Shift Right Shift	**L→R**	**5**
6.	< > <= >=	Less than Greater than Less than or equal to Greater than or equal to	**L→R**	**6**
7.	== !=	Equal to Not equal to	**L→R**	**7**
8.	&	Bitwise AND	**L→R**	**8**

9.	^	Bitwise X-OR	**L→R**	**9**
10.	|	Bitwise OR	**L→R**	**10**
11.	&&	Logical AND	**L→R**	**11**
12.	**||**	Logical OR	**L→R**	**12**
13.	?:	Conditional operator	**R→L**	**13**
14.	= *= /= %= += -= &= |= ^= <<= >>=	Simple assignment Assign product Assign quotient Assign modulus Assign sum Assign difference Assign bitwise AND Assign bitwise OR Assign bitwise XOR Assign left shift Assign right shift	**R→L**	**14**
15.	,	Comma operator	**L→R**	**15**

Example:

```
int a=4,b=2;
float x=2,y=3;
a=(x/y)/y;
b=(a*x)/y;
```

a=(2.0/3.0)+4/2
=0.6+2
=2.6

b= (4*2.0)/3.0
= 8.0/3.0
=2.66

TYPE CASTING OR TYPE CONVERSION:

- Normally before an operation takes pace both the operands must have the same type.
- C converts one or both the operands to the appropriate date types by Type conversion.
- This can be achieved in 3 ways.

IMPLICIT TYPE CONVERSION:

- In this the data type Variable of lower type (which holds lower range of values or has lower precision) is converted to a higher type (which holds higher range of values or has high precision).
- This type of conversion is also called "promotion".
- If a "char" is converted into "int" it is called as internal promotion

Example:

```
void main()
{
   int I;
   char C = 'A';
   I = C;
   printf("I=%d",I);
}
```

Output

I=65

Now the int Variable I hold the ASCII code of the char 'A'

a) An arithmetic operation between an integer and integer yields an integer result.

b) Operation between a real yield a real

c) Operation between a real & an integer always yields a real result

Example:

5/2 = 2	2/5 = 0
5.0/2 = 2.5	2.0/50. = 0.4
5/2.0 = 2.5	2/5.0 = 0.4
5.0/2.0 = 2.5	2.0/5.0 = 0.4

ASSIGNMENT TYPE CONVERSION:

If the two Operands in an Assignment operation are of different data types the right-side Operand is automatically converted to the data type of the left side.

Example: Let k" is an int var & a is a float var

```
int k;
float a;
```

k= 5/2=2	k=2/5=0	a = 5/2=2.0
k=5.0/2=2.5	k=2.0/5.0=0.4	a = 5.0/2=2.5

EXPLICIT TYPE CONVERSION:

- When we want to convent a type forcibly in a way that is different from automatic type conversion, we need to go for explicit type conversion.

```
(type name) expression;
```

- Type name is one of the standard data types.
- Expression may be a constant variable or an expression this process of conversion is called as casting a

value. Example: x = (int) 7.5;

A = (int) 21.3/ (int) 4.5;

Y = (int) (a + b);

P = (double) (sum)/n;

Format Specifier: Conversion specifications

Data Type	Conversion symbol
short int	%d
long int	%ld
float	%f
double	%lf
char	%c
string	%s

BRANCHING STATEMENTS

In the term software or computer programming, it has a set of instruction (in simple or complex form) called program. These instructions are also called statements, which occurs sequentially or in either conditional way or in the iterative (repeated) way. To handle such types of statements some flow controls required. These flow controls are called Control Statements

In other words, the control statements are used to control the cursor in a program according to the condition or according to the requirement in a loop. There are mainly three types of control statements or flow controls. These are illustrated as below:

- **Branching**
- **Looping**
- **Jumping**

Branching Statement

- **if statement**
  - **Simple if statement**
  - **if-else statement**
  - **nested if statement**
  - **else-if or ladder if or multi-condition if statement**
- **switch statement**
- **conditional operator statement**

Simple if statement:

The "if" statement is a powerful decision-making statement and is used to control the flow of execution of statements.

Syntax:

```
if (Condition or test expression)
{
Statement;
}
Statement-x;
```

- It is basically a "Two-way" decision statement (one for TRUE and other for FALSE)
- It has only one option.
- The statement as executed only when the condition is true.
- In case the condition is false the compiler skips the lines within the "if Block".
- The condition is always enclosed within a pair of parenthesis i.e. ( ).

- The conditional statement should not the terminated with Semi-colons (i.e. ;)
- The Statements following the "if"-statement are normally enclosed in Curly Braces i.e. { }. But it is good practice to use curly braces even with a single statement.
- The statement block may be a single statement or a group of statements.
- If the Test Expression / Conditions is TRUE, the Statement Block will be executed and executes rest of the program

Example: Write a program to check equivalence of two numbers. Use "if" statement.

```
# include<stdio.h>
# include<conio.h>

void main( )
{
int m,n;
clrscr( );
printf("\n Enter two numbers:");
scanf("%d %d", &m, &n);
if((m-n)= =0)
{
printf("\n two numbers are equal");
}
getch();
}
```

Output:
Enter two numbers: 5
5
Two numbers are equal.

if-else Statement:

- It is observed that the if statement executes only when the condition following *if* is true.
- It does nothing when the condition is false.
- In if-else either True-Block or False – Block will be executed and not both.
- The "else" Statement cannot be used without "if".

Syntax:

```
if ( Test Expression or Condition )
  {
        Statement-1; /*true block (or) if block */
   }
else
   {
        Statement-2; /* false block (or) else block */
    }
```

Example 1: Write a program to print the given number is *even* or *odd*.

```
#include<stdio.h>
#include<conio.h>
void main( )
{
int n;
clrscr( );
printf("\n Enter a number:");
scanf("%d", &n);
if( (n%2)==0 )
printf("\n The given number is EVEN");
else
printf("\n The given number is ODD");
getch();
}
```

Nested "if–else" Statement:

- Using of one if-else statement in another if-else statement is called as nested if-else control statement.
- When a series of decisions are involved, we may have to use more than one if-else statement in nested form.

Syntax:

```
if (Test Condition1)
    {
    if (Test Condition2)
      {
            Statement -1;
       }
    else
      {
            Statement -2;
       }
     }
else
    {
    if (Test Condition3)
      {
            Statement -3;
       }
   else
      {
            Statement -4;
       }
    } /* end of outer if-else */
```

- If Test Condition-1 is true then enter into outer if block, and it checks Test Condition-2 if it is true then Statement-1 executed if it is false then else block executed i.e. Statement-2.
- If Test Condition -1 is false then it skips the outer if block and it goes to else block and Test Condition- 3 checks if it is true then Statement-3 executed, else Statement-4 executed.

Example 1:

Program to select and print the largest of the three float numbers using nested "if-else" statements.

```
# include<stdio.h>
# include<conio.h>
void main( )
{
int a,b,c;
printf("Enter Three Values:");
scanf("%d%d%d",&a,&b,&c);
printf("\n Largest Value is:") ;
if(a>b)
 {
  if(a>c)
   printf("%d", a);
  else
   printf("%d", c);
 }
else
 {
  if (b>c)
   printf("%d", b);
  else
   printf("%d", c);
 }
getch( );
```

Output:

```
Enter three values: 45
78
145
Largest Value is: 145
```

```
}
```

(4) The "else – if" Ladder:

- This is another way of putting *if* else together when multiple decisions are involved.
- A multipath decision is a chain of *if else* in which the statement associated with each *else* is an *if*. Hence it forms a ladder called *else–if* ladder.

Syntax

```
if (Test Condition -1)
      Statement -1;
else if (Test Condition -2)
        Statement -2;
      else if (Test Condition -3)
            Statement -3;
                  :

                  :

          else if (Test Condition –n)
                Statement –n;
              else
                  default statement;
Statements-X;
```

- The above construction is known as *else if ladders*.
- The conditions are evaluated from top to bottom.
- As soon as a true condition is found, the statement associated with it is executed and the control is transferred to the Rest of the Program Statement–X (skipping rest of the ladder).
- When all the conditions become false, then the final else containing the default statement will be executed.

Example: Write a program to read three numbers and find the largest one by using "else-if" ladder.

```
# include<stdio.h>
# include<conio.h>
void main( )
{
int a, b, c;
clrscr ( ) ;
printf("Enter numbers a,b,c:");
scanf("%d%d%d", &a,&b,&c);
if ((a>b) && (a>c))
printf("\nHighest Number is: %d", a);
else if ((b>a) && (b>c))
printf("\nHighest Number is: %d", b);
else
printf("\nHighest Numbers is: %d", c);
getch( );
}
```

(5) The switch-case Statement:

- The *switch* statement causes a particular group of statements to be chosen from several available groups.
- The selection is based upon the current value of an expression which is included within the *switch* statement.
- The *switch* statement is a multi-way branch statement.
- In a program if there is a possibility to make a choice from a number of options, this structured selected is useful.

- The *switch* statement requires only one argument of *int* or *char* data type, which is checked with number of case options.
- The *switch* statement evaluates expression and then looks for its value among the *case* constants.
- If the value matches with *case* constant, then that particular *case* statement is executed.
- If no one *case* constant not matched then *default* is executed.
- Here *switch, case* and *default* are reserved words or keywords.
- Every *case* statement terminates with colon "**:**".
- In *switch* each *case* block should end with *break* statement, i.e.

Syntax:

```
switch(variable or expression)
    {
        case value-1:
              statement(s);
              break;
       case value-2:
              statement(s);
              break;
        _ _ _ _ _
        _ _ _ _ _
       case value-n:
              statement(s);
              break;
     default:
              statement(s);
    }
```

The switch() Execution:

- When one of the cases satisfies, the statements following it are executed.
- In case there is no match, the default case is executed.

Example: Write a program in C to evaluate addition, subtraction, multiplication, division based on user choice.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,c;
char ch;
printf("Press + addition, -subtraction, * multiply, / divsion, 5-exit");
scanf("\n%c",&ch);
printf("\nEnter two numbers");
scanf("%d%d",&a,&b);
switch(ch)
{
    case '+':
          c=a+b;
          break;
    case '-':
          c=a-b;
          break;
    case '*':
          c=a*b;
          break;
    case '/':
          c=a/b;
          break;
```

Output:

```
Press + addition, -subtraction, *
multiply, / divsion, 5-exit
+
Enter two numbers 10
30
Answer is 40
```

```
case 5:
                exit(1);
          default:
                printf("\nYou have entered wrong choice");
                exit(1);
     }
    printf("\nAnswer is %d",c);
    getch();
    }
```