UNIT II

OPERATORS AND EXPRESSION

Operator :

- 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.

Operand:

- The data items that the operators act upon are called operands.
- The operator that uses a single operand is called an unary operator, and which uses two operand is known as a binary operator.
- Any valid combination of constants, variables and operators constitute an expression.

Types of operator:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Increment and Decrement Operators
- Conditional Operators
- Bitwise Operators
- Special Operators

Arithmetic Operators:

It can be classified as

- Binary arithmetic operators
- Unary Arithmetic operators

Binary	Arithmetic	unary	Arithmetic
Operators	**Operation**	**Operators**	**Operations**
+	Addition	+	Unary Plus
-	Subtraction	-	Unary minus
*	Multiplication	++	Increment
/	Division	--	Decrement
%	Modulus		

- the modulo division operation produces the remainder
- The division operator / gives quotient value.
- Second operand of the operators % and / must be non-zero.

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

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

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.

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);
}
```

Real Arithmetic:

- An arithmetic operation involving only real operands is called real arithmetic.
  X = 6.0 / 7.0   = 0.857143

Relational Operators:

- Relational and equality operators are used to test or compare two numeric values or numeric expressions.
- They require two operands.
- The operands may be scalar data-types or pointers.

An expression such as

a <b or 1 < 20

- containing a relational operator is termed as a relational expression.
- The value of the expression is either one or zero.
- 1 represents true and zero if the relation is false.

Example :

10>20 is true 20<10 false

Relational operator :

operator	**meaning**
<	is less than
<=	is less than or equal
>=	is greater than or equal
>	is greater than
==	is equal to
!=	is not equal to

relational expression:	**expression result**
5<6	1(true)
X<=5	0(flase)
5!=6	1(true)

Logical operators:

'C' has threee logical operators:

&& meaning logical AND

|| meaning logical OR

! meaning logical NOT

The logical operators && and || are used when we want to test more than one condition and make decision

ex:a>b &&X==10

whichcombins two or more relational expression is termed as logical expression or as compound relational expression.

Truth table for && and||

Op1	**op2**	**op1&&op2**	**op1||op2**
Non-zreo	non-zero	1	1
Non-zero	0	0	1
0	non-zero	0	1
0	0	0	0

Truth table for !

condition	**result**
!false	true
!true	false

Example:

x=5 and y=0

x&&y =0 true &&false

x||y=1 true||false

!x=0 !true

Assignment operators:

- Assignment operators are used to assign the result of an expression to a variable

Three categories of operations:

1.Simpleassignement

2.Compound assignment

3.Assignment as expression

General forms:

lvalue=expression

General form2:

Lvalue operator =expression

is equivalent to

lvalue =lvalue operator expression;

where operator is + - * % >><<& ^ \

- **lvalue is the variable**
- expression may be constant, variable or a combination of both and operator
- = is the assignment operator

Examples:

a=a+1 a+=1

a=a-1 a-1=1

a=a*(n+1) a*=n+1

a=a/(n+1) a/=n+1

a=a&58 a&=58

a=a/b a/=b

- if a semicolon(;)is placed at the end of an assignment expression the result is an assignement statement

INCREMENT AND DECREMENT OPERATORS:

- C provides two more powerful and useful unary arithmetic operators.
- Increment operator ++ and the decrement operator—are also known as auto increment and auto decrement operators

++ adds 1to the operand,while – subtracts1

++m ;or m++;

--m; or m--;

is equivalent to

m=m+1; or m+=1;

m=m-1; or m-=1;

we use in for and while loops respectively

Rules :

1.The operand must be a variable,but not a constant or an expression.

2.The operator ++ or – may precede or succed the operand

Ex:

m=5

y=++m;

The value of y and m would be 6

Suppose if we rewrite

m=5;

y=m++;

then value of y would be 5 and m would be 6.

The prefix operator first adds 1 to the operand and then th e result is assigned to the variable on left on the other hand a postfix operator first assigns the value to the variable on left and then increment the operand.

Conditional operator :

The ternary operator pair ?:

Syntax:

exp1 ?exp2 :exp3

Where exp1,exp2,exp3 are expressions

Ex:

A=10;b=15;

X=(a>b)?a:b

Exp1 is evaluated first if it is nonzero then the expression exp2 is evaluated and become the result if exp1 is false,exp3 is evaluated and its value is the results.

BITWISE OPERATOR :

- Bitwise operator are used for manipulation of data at bit level.
- Direct manipulation of the bits stored in the memory.
- Bitwise operator are applied only to integer types such as char, short, int, signed, unsigned and long ,not applied for float and double.

Bitwise	**operators**
&	bitwise AND
|	bitwise inclusive OR
^	bitwise exclusive OR(XOR)
<<left shift	
>>right shift	
~	one's compliment (unary)

BITWISE AND (&)

- The operator & performs a bitwise AND between two operands.
- It computers each bit of the left operand with the corresponding bit of the right operand.

Compared bits	**result**
0&0	0
0&1	0
1&0	0
1&1	1

Ex:

a&b 00000101

```
&       00000011
                      __________________

                  00000001          result =1
```

BITWISE OR(|) INCLUSIVE :

- The | operator performs a bitwise OR between two operators.

Compared bits	result
0|0	0
0|1	1
1|0	1
1|1	1

Ex:

```
a|b                   00000101
                        00000011
                          __________________

                        00000111
given number 187          Binary Representation:   10111011
             15 ______                                        00001111 __________________
             91                                                 10111111
```

BITWISE EXCLUSIVE OR(^):

The operator ^ performs a bitwise exclusive OR between two operands.

COMPARED BITS	**RESULT**
0^0	0
0^1	1
1^0	1
1^1	0

ex:

```
a^b 00000101
   00000011          result(decimal)
   00000110            6
given number                214              11010110
selected second operand   31          ^  00011111
                              ________        ________________
                              201          11001001
```

SHIFT OPERATORS:

- The right shift operator >> and the left shift operator << more the bit patterns

To the right or less respectively,

Syntax: constant shift operator n

Variable or shift operator n

- Where shift operator is any one of the shift operator >. And << n is the number of bit positions to be shifted.

Ex:

Bit pattern 10101101 is to be shifed by 3 bit position

1	0	1	0	1	1	0	1

The bit patten after the right shift operation is given below

0	0	0	1	0	1	0	1

Left shift operator

0	1	1	0	1	0	0	0

Example:

```
+20 >> 2     00010100  >>  2                    100  >> 1   01100100 >> 1

                  00000101      5 Decimal                                00110010     50 Decimal

+20<<2                  00010100 <<2

                         01010000 ->80 (decimal)

100<<          1 01100100 <<1

               11001000 -> 200(decimal)
```

ONE’S COMPLIMENT OPERATOR:

- this operator converts all 1 bits to 0 and all 0 bit to 1

illustration of ~operator

Expression	Binary representation	one’s compliment	
		Binary	decimal
~17	~00010001	11101110	238

Special operators:

- Special operators such as comma operator,size of operator,pointer operators(& and *)
- And member selection operator (. And ->)

The comma operator:

- The comma operator can be used to link the related expression together.
- It is evaluated from left to right and the value of right most expression is the value of combined expression
- Value=(x=10,y=5,x+y)
- First assigns the value 10 to x then assigns 5 tpo y and finally assigns 15 to value.

The sizeof operator:

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

M=sizeof(sum)

N=sizeof(long int);

K=sizeof(235l);

Arithmetic expression:

- An arithmetic expression is a combination of variable,constant,and operator arranged as per the asyntax of the language.

Algebraic expression	**'C' Expression**
axb-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

<u>Evaluation of expressions:</u>

Expressions are evaluated using an assignment stement

Variable =expression

X=a*b-c;

Y=b/c*a;

<u>Precedence of arithmetic operators:</u>

An arithmetic expression without parenthese will be evalued from left to right using the rules of precedence of operators

<u>Arithmetic operator:</u>

+,-,*,/,%

high priority * /%

low priority + -

First the high priority operators are evaluated and then the low priority operators

x=a-b/3+c*2-1

when a=9,b=12 and c=3

x=9-12 /3+3*2-1

it is evaluated as

<u>step:</u> x=9-4+3*2-1

x=9-4+6-1

x=5+6-1

x=11-1

x=10

Rules:

1.First, parenthesized sub expression from left to right are evaluated

2.If parentheses are nested, the evaluation begins with the

3.Arithmetic expression are evaluated from left to right using the ri\ules of precedence.

4.When parentheses are used ,the expressions with in parenthese assume highest priority.

Type conversion in Expressions:

C converts one or both the operands to the appropriate data types by type conversions.

Type conversion can be

1. Implicit type conversion
2. Explicit type conversion

Expression combines of constant and variables of different types.

- ✰ C automatically converts any intermediate values to the proper type. This automatic conversion is known as implicit type conversion
- ✰ If the operands are of different types the lower type is automatically converted to the highest type before the operation proceeds.

Data type rank:

long double	1 (higher rank)
double	2
float	3
int	4
short int	5
char	6(lower rank)

Rules:

1. If one of the operand is long double, double, float & unsigned long int, the other will be converted to long double, double, float &unsigned long int and the result will be the same.
2. If either operand is long, the other operand is converted to long.
3. If either operand is unsigned, the other operand is converted to unsigned.
4. char and short are converted to int

Assignment type conversion:

1. Float to int cause truncation of the fractional part.
2. Double to float cause rounding of digits.
3. Long int to int cause dropping of the excess higher order bits.

If the two operands in an assignment operation are of different data type, the right side operand is automatically converted to the data type of the left side.

EXPLICIT TYPE CONVERSION:

Scalar data types can be explicitly converted into desired data types.

It is accomplished by using cast operator.

Syntax:

```
(data_type) expression
```

where expression is converted to the target data type enclosed with in the parentheses.

ratio=female/male

Female and male are declared as integers the decimal part of the result of the division would be lost and ration would represent a wrong figure.

ratio =(float)female/male

(float)converts the female to floating point for the purpose of evaluate of the expression

Ex:

x=(int)%5 result is x=7

a=(int)21.3/(int)4.5 result is 21/4=5

y=(int)(a+b) the result of a+b is converted to integer.

<u>Operator precedence and associatively:</u>

Operator	**description**	**associativity**
1.( )	function call	left to right
[ ]	array element reference	“
2. !	Logical Not	Right to left
~	one’s complement	“
-	Unary Minus	“
++	Increment	“
--	Decrement	“
&	Address of	“
*	Indirection	“
(data_type)	Cast Operator	“
sizeof	size of	“
3. *	Multiplication	Left to Right
/	Division	“
%	Modulus	“
4. +	Addition	“
-	Subtraction	“
5. <<	Left shift	“
>>	Right shift	“
6. <	Less than	“
>	Greater than	“
<=	Less than or Equal	“
>=	Greater than or Equal	“
7. ==	Equal to	“
!=	Not equal to	“
8. &	Bitwise AND	“
^	Bitwise XOR	“
|	Bitwise OR	“
9. &&	Logical AND	“
||	Logical OR	“

10. ?:	Conditional	Right to Left
= += -= *= /= %= >>= <<= &= ^= /=	Simple and Compound Assignment	“
11. ,	Comma	Left to Right

DECISION MAKING AND BRANCHING:

DECISION MAKING:

- ♣ IF Statement
- ♣ Switch Statement
- ♣ Conditional Operator Statement
- ♣ GOTO Statement

- ❖ These statements are popularly known as decision-making statements.
- ❖ These statements ‘control’ the flow of execution, they are also known as control statement.
- ❖ IF statement is a powerful decision-making statement and is used to control the flow of execution of statements.

if (**test_expression**)

Statements;

Condition Execution:

- ♣ The flow of execution may be transferred from one part of a program to another part based on the output of the conditional test carried out. This is known as a **conditional execution.**

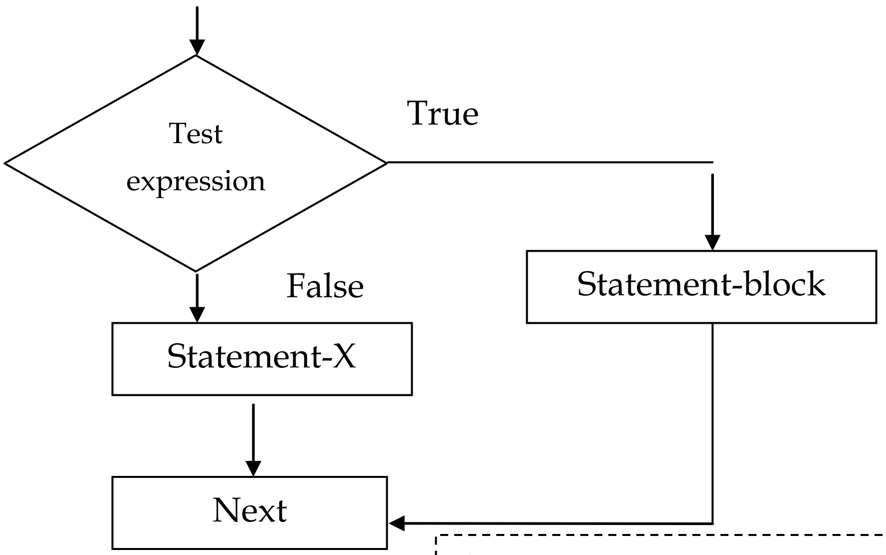
Simple IF Statement:

Syntax:

```
if (test_expression)
{
statement_block;
}
statement_x;
```

- ❖ The statement_block may be a single statement or a group of statement.
- ❖ If the test expression is true, the statement block will be executed, otherwise the statement_block will be skipped and the execution will jump to the statement-x.
- ❖ When the condition is true both the statement_block and the Statement_x are executed in sequence.

Flow Diagram:



Example:

```
#include <stdio.h>
main()
```

```
OutPut
Given number is 11
The given number 11 is ODD
```

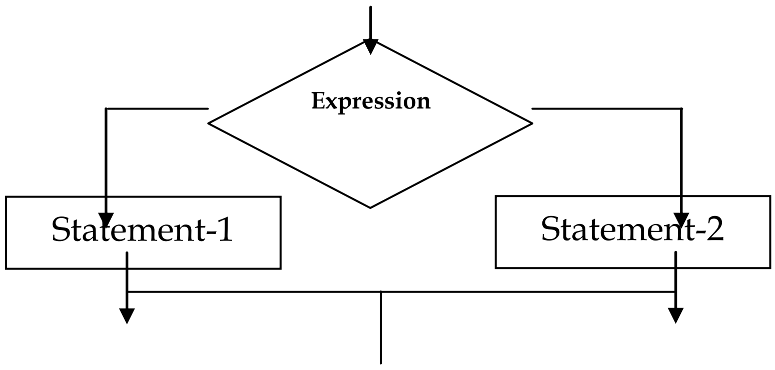
```
{
        intx,y;
        printf("enter an integer \n");
        scanf("%d",&x);
        printf("Given number is %d \n",x);
        y=x%2;
        if (y==1)
        printf("The given number %d is ODD \n ",x);
 }
```

IF-ELSE Construct:

- ❖ This construct is used to select a specific statement based on the specified condition.

Syntax: **Flow Chart**

```
        if (expression)
        {
                Statement_1;
        }
        else
        {
                Statement_2;
        }
```



- ❖ Where statement_1 and Statement_2 may be simple or compound statement.
- ❖ If there is a simple statement, the braces are optional.
- ❖ But the braces should surround the compound statements.
- ❖ The expression for the conditional test is always written within parentheses.
- ❖ The statement_1 will be executed if the expression returns a true value, otherwise statement_2 will be executed.

Example

```
/* Program to find whether an integer is odd or even */
        #include <stdio.h>
        main()
        {
                intx,y;
                printf("Enter an integer \n");
                scanf("%d",&x);
                printf("Given number is %d \n",x);
                y = x%2;
                if (y == 1)
                        printf("The given  number %d" is ODD \n",x);
                else
                        printf("The given number %d is Even \n",x);}
```

```
OutPut
Given number is 10
The given number 10 is EVEN
```

Nested-if-else:

- When an if-else construct appears as a statement in another if-else, it is known as nested-if-else construct.

Syntax:

```
          if(expression-1)
          {
          if (expression-2)
          {
          if(expression-3)
          {
          statement-1;
          }
          else
          {
          statement-2;
          }
          }
          else
          {
          statement-3;
          }
          }
```

- Where statement-1, statement-2 and statement-3 may be simple or compound statement.
- The statement-1 will be executed if the expression-3 is true; otherwise statement-2 is executed.
- If expression-1 is true then only expression-2 will be executed otherwise it skipped the subsequent statements.
- If expression-1 and expression-2 is true then the expression-3 will be executed; otherwise if expression-2 is false then the statement-3 is executed.

if-else-if Construct:

- The else if construct follows the if construct to have multiple decision making.

General form:

```
     if(expression-1)
     {
     statement-1;
     }
     else if(expression-2)
     {
          statement-2;
     }
     else if(expression-3)
     {
```

```
                         statement-3;
          }
          . . .
          else if(expression(n-1))
          {
          statement(n-1);
          }
          else
          {
          statement-n;
          }
```

- ❖ If any expression returns true value, the statement associated with that if construct is executed and then the next statement to the whole else if construct is executed.
- ❖ If no expression returns true value then the last else part is executed.

/* Program to check whether a char is vowel or not */

```
main()
{
charch;
printf(“Enter a character : \n”);
ch=getchar();
printf(“The character entered “);
putchar(ch);
if(ch>=’A’ &&ch<=’Z’ ||ch>=’a’ &&ch<=’z’)
 {
          if (ch == ‘a’ ||ch==’e’ || ch==’i’ || ch==’o’||
          printf(“ is a letter and vowel \n”);
          else if(ch==’A’ ||ch==’E’ ||ch==’I’||ch==’O’||ch==’U’)
          printf(“is a letter and vowel \n”);
else
          printf(“is a letter but not a vowel \n”);
    }
else
printf(“is not a letter \n”);
}
```

```
Output
Enter a character:   A
The character entered A is a letter and vowel.
Enter a character : w
The character entered w is a letter but not a vowel.
  Enter a character : 5
  The character entered 5 is not a letter.
```

Conditional Expression:

- ❖ The ternary operator ?: is used to form a conditional expression.
- ❖ It uses three operands and hence it is called as a ternary operator.

Syntax:

```
          Expression-1 ? expression-2 : expression-3;
```

- Parentheses are optional for expression-1
- A conditional expression is evaluated.
  1. The expression-1 is evaluated to find the logical value true or false.
  2. if the expression-1 is true, then the expression-2 is evaluated and that is the resulting value of the conditional expression.
  3. if the expression-1 is false, then the expression-3 is evaluated and that is the resulting value of the conditional expression.

```
/* program to find the largest of two numbers */
#include <stdio.h>
main()
{
        int x;
        float y;
        printf("Enter an integer and a real value \n");
        scanf("%d %f",&x,&y);
        printf("Given integer value = %d \n",x);
        printf("Given real value = %f\n",y);
        printf("Larger of these values = %f\n",(x>y)?x:y);
}
```

Output

```
Enter an integer and a real value
        5       2.3
Given integer value = 5
Given real value = 2.300000
Larger of these values = 5.000000
```

SWITCH STATEMENT

- The switch-case construct is an alternative to if-else or else-if construct.

Syntax:

```
        Switch(expression)
        {
                case constant-1:
                {
                        statement;
                        break;
                }
                case constant-2:
                {
                        statement;
                        break;
                }
case constant-n:
                {
                        statement;
                        break;
                }
default:
                {
                        statement;
                        break;
                }
        }
```

- ❖ The constants following the cases are known as case labels.
- ❖ A colon (:) must be placed at the end of each case label and default.
- ❖ The braces following the labels are optional.
- ❖ Break statement is not necessary in the default part.
- ❖ The expression must be an integer.
- ❖ The expression is evaluated first and its value is then matched against the case labels.
- ❖ When a match is found, the statement following that case will be executed.
- ❖ If there is no match, the statement following the default will be executed.
- ❖ The default part is optional.
- ❖ If, it is omitted, no statement within the switch-case construct will be executed.

/* program to print the digits in words */

```
main()
{
    int digit;
    printf("Enter a single digit number \n");
    scanf("%d",&digit);
switch(digit)
    {
    case 0:
        printf("The number entered is ZERO \n");
        break;
    case 1:
        printf("The number entered is ONE :\n");
        break;
    case 2:
        printf("The number entered is TWO:\n");
        break;
    default:
        printf("It is greater than two ");
        break;
    }
}
```

```
Output
Enter the Single Digit : 2
The number entered is TWO
Enter the Single Digit : 10
It is greater than two
```

Unconditional Execution:

1. Break Statement
2. Continue statement
3. goto statement

If the flow of execution is transferred from one part of a program to another part without carrying out any conditional test, it is known as an unconditional execution.

Break statement:

Syntax:

```
break;
```

- ❖ when a break statement appears inside a loop or in a switch-case control structure is terminates the execution at that point and transfer execution control to the statement immediately following the loop or switch-case construct statement.

Continue Statement:

The General format is

Continue;

- ❖ The continue statement is used within loops to end the execution of the current iteration and proceed to the next iteration.
- ❖ It provides a way of skipping the remaining statements in that iteration after the continue statement.
- ❖ Continue statement should be used only in loop constructs and not in selective constructs.

Example

```
/* Program to sum the given series */
main()
{
inti,sum =0;
for(i=5;i<=20;i++)
{
if(i%5!=0) continue;
sum+=i*i;
  }
printf("Sum is = %d\n",sum);
}
```

goto Construct:

- ❖ The goto construct causes an unconditional transfer of execution.
- ❖ Goto may be used for breaking more than one loop at a time.

Syntax:

```
label:
        {
        statement;
        }
goto label;
```

- ❖ Where label is an identifier and not a number .
- ❖ Identifier is a statement label and it is not declared.
- ❖ Statement label can also be used as a variable name in the same program.
- ❖ The statements to be executed follow the label ending with a colon (:).
- ❖ If these statements appear before a goto statement, the flow is in backward direction and it is known as backward goto.
- ❖ If the statements appear after a goto statement; this type of goto is known as forward goto.

- ❖ Same label cannot have in different places.

Example

```
/* Program to read a string using GOTO */
main()
{
charch;
printf(“Enter a string \n”);
read: ch=getchar();
if(ch!=’\n’)
        {
        putchar(ch);
        goto read;
        }
}
```

ARRAYS:

- ❖ An array is an ordered sequence of finite data items of the same data type that shares a common name.
- ❖ The common name is the array name and each individual data item is known as an element of the array.
- ❖ An array may be one-dimensional or multidimensional.
- ❖ A one-dimensional array can be used to represent a list of data items. It is also known as a vector.
- ❖ A two-dimensional array can be used to represent a table of data items consisting of rows and columns.
- ❖ It is also known as a matrix.
- ❖ An array is a fixed-size sequenced collection of elements of the same data type.

ONE-DIMENSIONAL ARRAYS:

- ❖ A list of items can be given one variable name using only one subscript and such a variable is called a single-subscripted variable or a one-dimensional array.
- ❖ Single subscripted variable xi can be expressed as

x[1] , x[2], x[3] ….. , x[n]

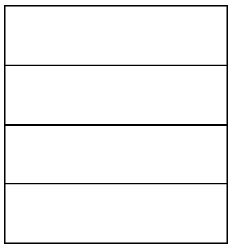
- ❖ The subscript can begin with number 0. That is x[0] is allowed.

Example:

- ❖ If we want to represent a set of five numbers, say (35,40,20,57,19), by an array variable number, then we may declare the variable number as follows

int number[5];

and the computer reserves five storage locations.



Number[0]

Number[1]

Number[2]

Number[3]

Number[4]

- The values to the array elements can be assigned as follows

```
number[0] = 35;
number[1] = 40;
number[2] = 20 ;
number[3]=57;
number[4] = 19;
```

- This would cause the array number to store the values

35	Number[0]
40	Number[1]
20	Number[2]
57	Number[3]
19	Number[4]

DECLARATION OF ONE-DIMENSIONAL ARRAYS:

- Like any other variable, arrays must be declared before they are used.

typevariable_name[size];

- The type specifies the type of element that will be contained in the array, such as int, float or char, and the size indicates the maximum number of elements that can be stored inside the array.

float height[50];

- declares the height to be an array containing 50 real elements.

char name[10];

- declares the name as a character array(string) variable that can hold a maximum of 10 characters.
- Suppose we read the following string constant into the string variable name.

"WELL DONE"

- Each character of the string is treated as an element of the array name and is stored in the memory.

Example Program

```
main()
{
int i;
float x[10],value,total;
/* …… Reading values into array …… */
printf("Enter 10 Real numbers \n");
for(i=0;i<10;i++)
{    scanf("%f",&value);
x[i]=value;}
total = 0.0;
for(i=0;i<10;i++)
total = total + x[i];
printf("\n");
for(i=0;i<10;i++)
printf("x [%2d]=%5.2f \n",i+1,x[i]);
printf("\n Total = %2f \n ",total);
}
```

INITIALIZATION OF ONE-DIMENSIONAL ARRAYS:

- Initialize the elements of arrays in the same way as the ordinary variables.

Syntax:

- Type array_name[size] = {list of values};
- The values in the list are separated by commas.

Ex: int number[3] = {0,1,2};

float total[5] = {0.0, 15.75,-10};

- will initialize the first three elements to 0.0,15.75 and -10.0 and the remaining two elements to zero.
- The size may be omitted.

Ex: int counter[] = {1,2,3,7};

- In such cases the compiler allocated enough space for all initialized elements.

char name[] = {'J','O','H','N','\0'};

declares the name to be an array of five characters, initialized with the string "John" ending with the null characters.

TWO-DIMENSIONAL ARRAYS:

- In two-dimensional array the array variables that can stored a list of values called tables.

Syntax:

Type array_name[row_size][column_size];

Example:

	Item1	Item2
Salesgirl#1	310	275
Salesgirl#2	210	190
Salesgirl#3	405	235

- The table contains a total of 12 values three in each line.
- Table as a matrix consisting of three rows and three columns.
- We represent a particular value in a matrix by using two subscript such as Vij.
- V denote the entire matrix.
- Vij refers to the value in the ith row and jth column.
- V22 refers to the value 190.

Column 0 Column 1

- Single Dimensional arrays, each dimension of the array is indexed from zero to its maximum size minus one.

- ❖ The first index selects the row and the second index selects the column within that row.

INITIALIZING TWO-DIMENSIONAL ARRAYS:

- ❖ It like as one-dimensional array.

  Syntax 1:

  int table[2][3] = {0,0,0,1,1,1};

- ❖ initializes the elements of the first row to zero and the second row to one.

Syntax 2:

int table[2][3] = { {0,0,0} , {1,1,1}};

by surrounding the elements of the each row by braces.

Syntax 3:

Array in the form of the matrix

int table[2][3] = {{0,0,0},{1,1,1}};

The array is completely initialized with all values, explicitly.

int table[ ][3]= {{ 0,0,0},{1,1,1}}; is permitted.

- ❖ If the values are missing in an initialized they are automatically set to zero.

  int table[2][3] = {{1,1},{2}};

- ❖ will initialize the first two elements of the first row to one, the first element of the second row to two, and all other elements to zero.

  int m[3][5] = { {0}, {0}, {0}};

  int m[3][5] = {0,0};

Multidimensional Arrays:

- ❖ C allows arrays of three or more dimensional.

Syntax:

typearray_name[s1][s2][s3]…[sn];

- ❖ Where si is the size of the ith dimension.

  int survey[3][5][12];

  float table1[5][4][5][3];

- ❖ survey is a three-dimensional array declared to contain 180 integer type elements.

Dynamic Arrays:

- ❖ An array created at compile time by specifying size in the source code has a fixed size and cannot be modified at run time.
- ❖ It is known as static memory allocation are called static arrays.
- ❖ Dynamic arrays are created using what are known as pointer variables and memory management functions such as malloc, calloc and realloc.
- ❖ These functions are included in the header file <stdlib.h>.