**Bharat Ratna Puratchi Thalaivar Dr.MGR Governtment Arts and Science College**
**Palacode – 636808**

## B.Sc. MATHEMATICS
## SEMESTER – V
## SKILL BASED ELECTIVE COURSE - III
## C PROGRAMMING

### UNIT - I

Constants and variables: Introduction - Character set - Constants - Keywords and Identifiers - Variables - Data Types - Declaration of Variables - Assigning values to variables - Defining symbolic constants.(Sections:2.1 to 2.8, 2.10, 2.11)

### UNIT - II

Arithmetic operators - Relational operators - Logical operators - Assignment operators - Increment and Decrement operators - conditional operators - Special operators. Arithmetic expressions - Evaluation of Expressions ( Sections 3.2 to 3.7, 3.9, 3.10, 3.11)

### UNIT - III

Managing Input and output operations: Reading a character - Writing a character - Formatted input and output Decision making and Branching: Decision making with IF Statement - Simple IF Statement - IF ELSE Statements - Nesting of IF …ELSE Statement - ELSE IF Laader ( Sections 4.1 to 4.5)

### UNIT - IV

Switch Statement - ? Operator - GOTO Statement - Decision Making and Looping: WHILE Statement - Do Statement - FOR Statement - Jumps in Loops - Simple Programs. ( Sections 5.2 to 5.9, 6.2 to 6.5)

### UNIT - V

Arrays: Introduction - One Dimensional array - Declaration of one and two dimensional arrays - Initiating of one and two dimensional arrays - Declaring and initializing string variables - Reading
strings from terminal - writing sting on the screen-Arithmetic operations on characters - simple
problems. ( Sections 7.1 to 7.6,8.1 to 8.5 )

### TEXT BOOK:

1. E. Balagurusamy, Reprint 2006, Programming in ANSI C, Tata McGraw Hill Publishing Company Ltd.,
New Delhi, 3rd Edition.

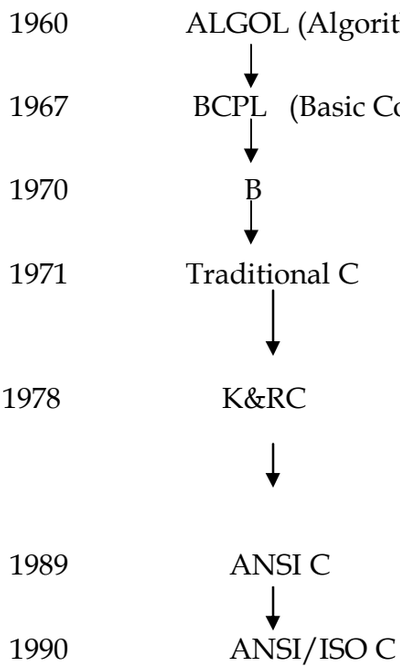
### REFERENCE BOOKS

1. Peter Aitken and Bradley L Jones, Teach Yourself C in 21 Days, Tech Media, New Delhi, 4th Edition.
2. Tony Zhang, Teach Yourself C in 24 Hours, Sams Publications, 1st Edition, 1997.
3. Ram Kumar and Rakash Agrawal, Programming in ANSI C, Tata McGraw Hill Publishing Company
Ltd., New Delhi, 1993.

# UNIT - I

**History of C:**

- C is one of the most popular computer languages.
- It is structured, high level, machine independent language.
- It is platform independent
- ALGOL is the first structured programming language used in Europe.
- In 1967, Martin Richards developed a language called BCPL (Basic Combined programming Language)
- In 1970, ken Thompson created a language using BCPL features called B.
- C was evolved from ALGOL, BCPL and B by Dennis Ritchie at Bell Laboratories in 1972.
- Unix Operating system, which was developed at Bell Laboratories, was coded almost entirely in C
- C is running under a variety of operating system and hardware platform.
- C proved to be an excellent programming language for writing system programs.
- "The C programming language" by Kernighan and Ritchie, 1977 is known as "K&RC"
- ANSI, in 1983 standard for C was introduced it is called an ANSI C.




**Chronological order of the Development of C**

**Features of C:**

**C is attractive and popular because**

1.General-purpose language
2.Structured Language
3.Flexible and powerful language
4.System programming Language
5.Fast running and efficient Language
6.Supports limited data types.
7.Commands may be inserted anywhere in a program
8.Programs are made up of functions
9.More built-in functions
10.Permits recursion.

**Importance of C:**

- It is a robust language

- It is well suited for writing both system software and business packages.
- Programs written in C are efficient and fast
- It is many times faster than BASIC
- Several standard functions are available
- It is highly portable
- C language is structured and it makes program debugging, testing and maintenance easier.
- It is a collection of functions that are supported by C library.
- We can add own functions
- The programming task becomes simple using functions

**Advantages**

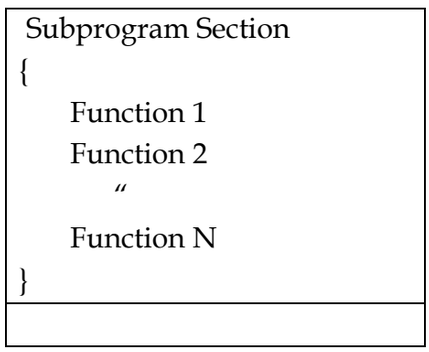
– C is a real world language, widely available and popular with professional
– C is a small, efficient, powerful and flexible language
– C has been standardised, making it more portable than some other languages
– C is close to the computer hardware revealing the underlying architecture
– C provides enough low level access to be suitable for embedded systems
– C is a high level language allowing complex systems to be constructed with minimum effort
– C's modular approach suits large, multi-programmer projects
– C's use of libraries makes it adaptable to many different application areas
– The Unix operating system was written in C and supports C
– C gave birth to C++, widely used for applications programming and more recently Java which was based upon C++
– Many other languages borrow from C's syntax: e.q. Java,JavaScript, Perl etc

**Disadvantages**

- C is not really a language for novices; it was designed for professional users
- There are many things that can go wrong if you're not careful
- C lacks much of he automatic checking found in other high level languages
- Small typing errors can cause unwanted effect
- Does not support modern concepts such as object orientation and multi threading.

**BASIC STRUCTURE OF C PROGRAM:**

| Documentation Section |
|---|
| Link Section |
| Definition Section |
| Global Declaration Section |
| Main( ) Function Section<br>{<br>Declaration Part ;<br>Executable Part ;<br>} |




**Documentation Section:**

It contain of set of comment lines giving the name of the program,author etc.comment lines should be given between /*…….*/

**Link Section:**

It provides instruction to the compiler to link function from the system library.

**Definition Section:**

- Define all symbolic constant.
- It must be in uppercase.

eg type def #define pi 3.14.

**Global Declaratoin Section:**

The global variables are declared in this section that is outside of all the function.

**Main Function Section:**

It contain is 2 part :

1.declaration part.
2.executable part.

**Declaration Part:**

In the part, declare all variables used in the executable part.

**Executable Part:**

- There is at least one statement in the executable part.
- These two parts appear between the opening and closing braces &all statements end with a semicolon.

**Subprogram Section:**

The subprogram section contains all the user-defined function that are called in the main function.

## CONSTANTS,VARIABLES & DATA TYPES

**Program:**

A sequence of instruction is used to perform a particular task. The instructions are formed using the character set.

**Character Set:**

The character sets are grouped into the following categories.

1. Letter (A….Z)&(a….z).

2. Digits (1…9).
3. Special characters
4. White spaces.

**Special Characers:**

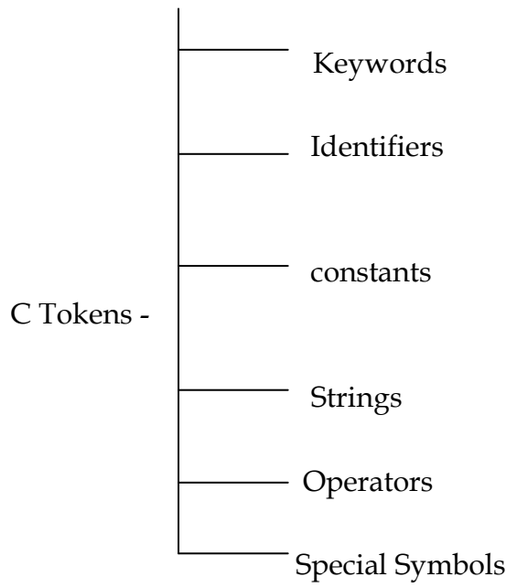
| SYMBOL | NAME | SYMBOL | NAME |
|---|---|---|---|
| , | comma | \ | back slash |
| . | period | - | under score |
| ; | semicolon | $ | dollarsign |
| : | colon | % | percentage sign |
| ? | question mark | & | ampersand |
| ‘ | apostrophe | ^ | caret |
| “ | quotation mark | * | asterisk |
| ! | exclamation mark | - | minus sign |
| \| | vertical bar | + | plus sign |
| / | slash | < | less sign |
| \ | greater than sign | ] | right bracket |
| > | left parenthesis | { | left brace |
| ( | right parenthesis | } | right brace |
| ) | right parenthesis | # | number sign |
| [ | left bracket | | |

**White Space:**

Black space.
Horizontal tab
Carriage return.
New line
Form Feed.

**C Tokens:**

Individual words ,punctuation mark, text are called tokens.




**Keyword:**

- All keywords have fixed meaning & these meanings cannot be changed. It must be written in lower case.
- ‘C‘ support 32 keywords only.

Eg Auto int
Void long
Goto float
Do Char

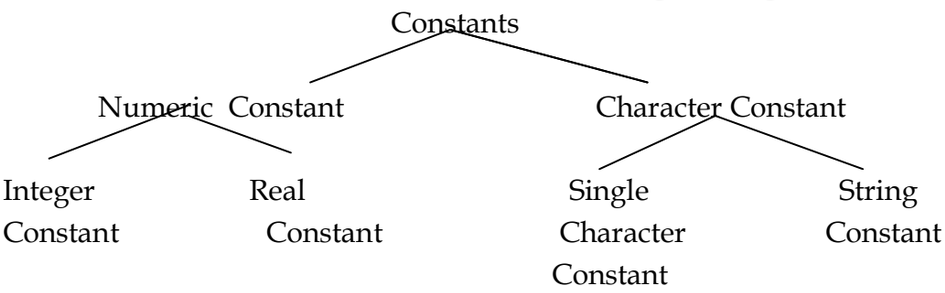
**Identifiers:**

It refer to the name of variables, functions & arrays. Both lowercase & uppercase are permitted. It cannot use a keyword.

Eg: Name, area, sum…..

## CONSTANT:

It refer to fixed data value that do not change during the execution of a program.




**Integer Constant:**

It refer to a sequence of digit.

Three types of integers

1. Decimal
2. Hexadecimal
3. Octal

**Decimal Integer:**

It consist of digit from 0 to 9. eg 1789.

**Hexadecimal :**

It must begin with either 0x or 0x and then followed by any digits from 0 to 9.

eg-0x123,0xabcd.

**Octal Integer Constant:**

It contain the digit from 0 to 7 eg 0777.

**Real Constant Or Floating Point Constant:**

The numbers are represented by fractional part like 17.548. This is called floating point constant. eg 0.86

A real number may also be expressed in exponential notation.

General Format:

| Mantissa e exponent; |
|---|

The mantissa may be either decimal or integer.

The exponent is an integer number with an optional plus or minus sign. The letter 'e' separating the mantissa.

Eg.0.65e4

**Single Character Constant:**

It is single character enclosed with single quotes.

Eg.'k'

**String Constant:**

A string constant is an sequence of character enclosed with in double quotes.

Eg. "hello".

**Variables:**

It is a data name that may be used to store a data value. A variable name can be chosen by the programmer.

Eg: total
Sum.

**Rules:**

- The must be begin with letter and remaining character may be alphabets or digits.
- It should not be a keyword.
- White space is not allowed.
- A maximum of 8 should be allowed.
- Upper case and lower case are significant.

**Declaration Of Variables:**

The declaration statement tells the complier 2 things.

1)It tells the complier what the variable name is.
2)It specify what type of data variable name is.

General format:

| **Data type v1,v2……vn:** |
|---|

Eg: int count;
Int number, total;

**Assigning Values The Variables:**

Value can be assigned to variables using the assignment operator '='.

General format:

| **Data type variable name = constant(data value);** |
|---|

Eg:

Int X=10;
Int N= 'a';

- The left-hand side of an assignment statement is variable name and right-hand side of an assignment statement is value.
- It is possible to use multiple assignment operators.

Eg a=b=10;

**DATA TYPES:**

C allows 3 classes of data types.

1)primary

2)user defined data types.
3)derived data types.

**Primary Data Type**

| **Integer** | | **character** | **floatingpoint** |
|---|---|---|---|
| Signed | unsigned | signed char | float |
| Int | unsigned int | unsigned char | double |
| Short int | unsigned shortint | | long double |
| Long int | unsigned int | | |

| Type | Memory | range |
|---|---|---|
| Char | 8 | -128 to 127 |
| Unsigned char | 8 | 0 to 255 |
| Int | 16 | -32,768 to 32,767 |
| Unsigned int | 16 | 0 to 65535 |
| Short int | 8 | -128 to 127 |
| Unsigned short int | 8 | 0 to 255 |
| Long int | 32 | -2,147,483,648 to 2,147,483,647 |
| Unsigned long int | 32 | 0 to 4,294,967,295 |
| Float | 32 | 3.4E-38 TO 3.4E +38 |
| Double | 64 | 1.7E-38 TO 1.7E + 308 |
| Long double | 80 | 3.4E-4932 TO 1.1E+4932 |

**User Defined Data Declaration:**

This allows the user to define an identifier that would represent an existing data type.

General format:

```
Typedef  type  identifier
```

Type ⟶ refer to an existing data type.

identifier ⟶ refer to the new name given to the data type.

**Eg Program:**

```
#include<stdio.h>
#include<conio.h>
 void main()
{
typedef   int marks;
Marks m1,m2,m3,tot;
Char name;
Scanf("%d%d%d%s",&m1,&m2,&m3,&name);
Tot=m1+m2+m3;
Printf("%d",tot);
Getch();
}
```

**Enumerated Data Type:**

General format:

```
Enum identifier {value1,value2……valuen}
```

identifier ⟶ is a user defined data type.

Eg enum day{Monday,Tuesday……Sunday}

- The complier automatically assign integer digits beginning with 0 to all the enumeration constants.
- ie) the enumeration constant value1 is assigned 0,value2 Is assigned 1 and so on.
- The automatic assignments can be overridden by assigning values.
- Eg enum day{Monday=1,Tuesday=2……..Sunday=n}

**Declaration Storage Classes:**

It provides the information about their location and visibility. The storage classes are

- auto
- register
- static
- extern

**auto**

- automatic variables are declared inside a function. It also referred to as local or internal variables.
- We may declare and use the same variable name in different function in the same program with out causing any confusion the complier.

**External Variable:**

- It is also known as global variables.
- It is declared as outside a function.
- Once a variable has been declared as global any function can use it and also change its value.
- The keyword is extern.

**Static Variable:**

It is a local variable which exists and retains its value even after the control is transferred to the calling function.

**Register:**

Local variables which is stored in the register.

# UNIT - II

## OPERATOR:

It is a symbol that tells the computer to perform certain mathematical or logical manipulations.

- Arithmetic operators
- Relational operators
- Logical operators
- Assignment operators
- Increment & decrement operators
- Conditional operators
- Bitwise operators
- Special operators

**Arithmetic Operators:**

The operators that works on numbers are called arithmetic operators.

+ addition
- subraction
* multiplication
/ division
% modulo(It is used to give the reminder value )

**Integer Arithmetic:**

We use integer values in the arithmetic expression is called integer arithmetic.

Eg:
A=5,b=3

| **Expression** | **value** |
|---|---|
| A+b | 8 |
| a-b | 2 |
| a*b | 15 |
| a/b | 1 (decimal point truncated) |
| a%b | 2 |

**EXAMPLE PROGRAM**

```
/*calculate months &days*/
Void main()
{
Int months, days;
Printf("enter the days");
Scanf("%d",&days);
Months=days/30;
Days=days%30;
```

Printf("months=%d days=%d",months,days);
}

**Real Arithmetic:**

The arithmetic operation involves only real numbers is called real arithmetic.

A=5.5,b=2.0

| **Expression** | **value** |
|---|---|
| A+b | 7.0 |
| a-b | 3.0 |
| a*b | 10.0 |
| a/b | 2.75 |

**Mixed mode Arithmetic**:

When one of the operands is real & other is integer .

Eg 15/10.0=1.5

**Relational Operators:**

These operators are used to compare two or more quantities. the result is either true or false.

| **Operators** | **Meaning** |
|---|---|
| < | is less than |
| <= | is less than or equal to |
| < | is greater than |
| >= | is greater than or equal to |
| = = | is equal to |
| != | is not equal to |

**General format:**

| **Operand1 relationaloperator operand2;** |
|---|

**Program**

```
#include<stdio.h>
#include<conio.h>
Void main()
{
int a,b;
clrscr();
scanf("%d%d",&a,&b);
if(a>b)
printf(" the largest value is %d",a);
else
printf(" the largest value is%d",b);
getch();}
```

**Logical Operator:**

The logical operator is used to test more than one condition.

| **Logical Operator** | **Meaning** |
|---|---|
| && | logical AND |
| \|\| | logical OR |
| ! | logical NOT |

**Working of AND,OR:**

| **Op1** | **op2** | **op&&op2** | **op1\|\|op2** |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 |

**Working of OR:**

| Op1 | !op1 |
|---|---|
| 0 | 1 |
| 1 | 0 |

**Assignment Operator:**

The assignment operator is used to assign the values to the operand.

General format:

```
Variable operand = expression;
```

**Shorthand Assignment:**

The shorthand assignment operator is op=.

Op->operator.

| 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; |

**Advantages:**

- The statement more concise and easier to read.
- The left-hand side variable need not to repeated on the right-hand side.
- The statement is more efficient.

**Increment And Decrement Operator:**

- The increment is operator (++) add one and decrement operator(--) is subtract one.
  Eg.
  ++m =>m=m+1=>m+=1.
  M++=>m=m+1=>m+=1.
- In general m++,++m are same but when they are used in assignment statement they have different meanings.
- If the operator placed before the operand it is known as preincrement operator.if the operator placed following the operand it is known as post increment operator.
  For eg.

| | |
|---|---|
| M=5 | m=5; |
| Y=++m; | y=m++; |
| Y=6,m=6. | y=5,m=6. |

- A prefix operator first adds 1 to the operand and then the result is assigned to the variable on left.
- A postfix operator first assign the value to the variable on left and then increment the operand.

**Conditional Operator:**

It is a ternary operator and takes three operands. The "?:" is joinly known as conditional operator.

General format:

```
Exp1?Exp2:Exp3;
```

Exp -> expressions.

- Exp1 is evaluated and if it is true,exp2 is evaluated and the value is assign to variable.
- Exp1 is false exp3 is evaluated and the value is assign to variable.

**Bitwise Operator:**

It is used for manipulation of data at bit level.
It may not be applied tp float or double.

| Operator | meaning |
|---|---|
| & | bitwise AND |
| \| | bitwise OR |
| ^ | bitwise exclusive OR |
| << | shift left |
| >> | shift right |

**Special Operators:**

1. Comma(,)
2. Sizeof.
3. Pointer(&.*).
4. Member selection(. And->).

**Comma Operator:**

- It is used to link the releted expression together.
- It is evaluated from left to right.

Eg.
Value=(n=1,m=10,n+m);

**The Sizeof Operator**:

It is compile time operator and when it is used in operand , it the number of bytes the operand occupies.

Eg
M=sizeof(sum).
N=sizeof(longint).

## EXPRESSION:

An expression is combination of operator and operands.
Three types of expressions.

- Arithmetic expression.
- Relational expression.
- Logical expression.

**Arithmetic Expression.:**

An arithmetic expression involving only arithmetic operators is known as arithmetic expression

```
Exp1 Arithmetic Operator Exp2;
```

**Precedence Of Arithmetic Operator:**

**First Priority:()**

The quantity inside the parenthesis have highest priority.

**Second Priority:*/%**

The * and / have equal priority and are evaluated from left to right in the order of occurrence.

**Third Priority:+-**

The + and - have equal priority and are evaluated from left to right in the order of occurrence.

**Arithmetic Expression Are Of Three Types** :

1. integer expression
2. real expression
3. mixed mode expression

**Integer Expression:**

An arithmetic expression involving only integer operands is known as an integer expression.

Let a=2,b=3,c=4,d=5,e=6

**Eg 1)**

(a+c)/c*d

Step 1): (2+4)/4*5

Step 2): 6/4*5

Step 3):1*5

Step 4): 5

**Eg 2)**

(a+c)/(c*d)

Step 1). (2+4)/(4*5)

Step 2). (6)/(4*5)

Step 3). (6)/(20)

Step 4). 0

**Real Expression:**

If all the operands in an expression are of real type the expression is known as real expression.

Where a,b, c, d, are real variables

Eg

Let a=.0 ; b=15.0;c=2.5

=10.0+15.0*2.5

=10.0+37.5

=47.5

**Mixed Mode Expression:**

If the operands in an expression are of different types, the expression is known as mixed mode expression for evaluating a mixed mode expression, the lowest type operands is converted to highest type and the result is of highest type.

Eg:

a+b*c

let a=10,b=15,c=2.5

=10+15*2.5

=10+37.5

=47.5

**Relational Expression:**

- An expression containing relational operators is known as the relational expression.
- It is also used to combine two arithmetic expression.
- The result of a relational expression is either true or false. True is taken as one and false is taken as zero.

  The general form is

| Exp1 relational operator Exp2; |
|---|

E.g.: a=5,b=6

a>b = false (0)

a!=b = True (1)

**Logical Expressions:**

- An expression containing logical operators is known as the logical expression.
- It is used to combine two or more relative expression.
- The result of a logical expression is either true or false.

  The general form is

| Exp1 logical operator Exp2; |
|---|

E.g. 1:

(test 1 > 40) | |(test 2 > 40)

Mark = 5

E.g. 2:

Mark 1 > 40)&& (mark 2 >40) && (mark 3 > 40)

R = 'p'

l NOT (!)

Second priority: logical AND (&&)

Third priority: logical OR (| |)

**Type Conversions In Expressions**:

There are two types conversions preformed on expressions.

1. Automatic type conversion.
2. Casting type conversion.

**Automatic Type Conversions:**

- This is done be the when the operands in an expression are of different types.
- The lower type is automatically converted into higher type before the operation proceeds.
- The result is always of higher type.

E.g.:

```
int i , k;
float f;
double d;
long int l;
x = l / i + i * f - d
```

**Rules In Automatic Type Conversion**:

1 .If one of the operands is long double other is converted to long double and the result is of long double.

2. If one of the operands is double the other is converted to double and the result is of double.

3. If one of the operands is float the other is converted to float and the result is of float.

4. If one of the operands is unsigned long int the other is converted to unsigned long int and the result is of unsigned long int.

5. If one of the operands is long int the other is unsigned int then either of the two things can be perform

- The unsigned int is converted long int and the result is of long int.
- Both the operands are converted to unsigned long int and the result is of unsigned long int.
- If lone of the operands is long int the other is converted to long int and the result is of long int.

6. If one of the operands is long int the other is converted to long int and the result is of long int.

7. If one of the operands is unsigned int the other is converted to unsigned and the result is of unsigned int.

The final result of an expression is converted to the type of the left hand side variable.

- Float to into causes, truncation of fractional part.
- Double to float causes rounding of digits.
- Long int to int causes dropping of excess higher order bits.

**Casting Values:**

This is done by the user. In some cases the user may force type conversion in a way that is different from automatic conversion.

For example: If the user wants to calculate the ratio of the number of females to the number of males The result is in integer if the number of females and the number of males are integer. In above

case the ratio is a wrong estimate due to the truncation of fractional part. This problem can be solved by converting locally one of the variables to float that is

Ratio = (float) no females / no males;

- The operator (float) converts the number of females to float
- This process of local conversion is know as casting the general is (type name )exp
- Where type name is the standard date type and exp is the variable or expression

Eg int (6, 7) = 6

(float) 7 = 7.000000.

## Operator precedence and associativity:

- The operator at the higher level of precedence are evaluated first.
- The operator of the same precedence are evaluated from left to right

| Operator | Description | Associativity | Rank |
|---|---|---|---|
| ()<br>[] | Function call<br>Array element refer | Left to right | 1 |
| +<br>-<br>++<br>--<br>!<br>~<br>*<br><br>Size of<br>(type) | Unary plus<br>Unary minus<br>Increment<br>Decrement<br>Logical negation<br>One's complement<br>Pointer reference<br>Address<br>Size of object<br>Type cast | Right to left | 2 |
| *<br>/<br>% | Multiplication<br>Division<br>modulos | Left to right | 3 |
| +<br>- | Plus<br>minus | Left to right | 4 |
| <<<br>>> | Shift left<br>Shift right | Left to right | 5 |
| <<br><=<br>><br>>= | Less than<br>Less that or equal to<br>Greater than<br>Greater than or equal to | Left to right | 6 |
| ==<br>!= | Equality<br>Inequality | Left to right | 7 |
| & | Bitwise AND | Left to right | 8 |
| ^ | Bitwise XOR | Left to right | 9 |
| \| | Bitwise OR | Left to right | 10 |
| && | Logical AND | Left to right | 11 |
| \|\| | Logical OR | Left to right | 12 |
| ?: | Conditional expression | Right to left | 13 |
| =<br>*= /= %=<br>+= -= &=<br>^= \|= | Assignment operators | Right to left | 14 |

| <<= >>= | | | |
|---|---|---|---|
| , | Comma operator | Left to right | 15 |

**BUILT IN FUNCTIONS (OR) LIBRARY FUNCTIONS**

- In general most of the programmers require mathematical.
  E.g. sin(x),cos(x), tan(x),…….
- These functions are known as built in functions or library functions. Functions, it is necessary to include the header file math.h
- C, provides, the below library functions for performing mathematical computations.

**TRIGNOMETRIC FUNCTIONS;**

- Sin(x) – used to find the sine valve of x . x in radians
- Cos(x)- used to find the cosine value of x.
- Tan (x)- used to find the tangent value of s.
  a sin (x) = sin -1 (x)
  a cos (x)= cos-1 (x)
  a tan (x)= tan-1 (x)

**OTHER FUNCTIONS:**

Ceil (x)- >If rounds the number to the next highest integer ceil (10.6)=1, ceil (10.3)=11

- Floor(x)-> it rounds the number to the next lowest integer floor (10.6)= 11, floor (10.3)= 10
- Exp(x)->e to the power.
- Fabs(x)-absolute value of x.
- Sqrt(x)->square root of x.
- Log(x)->natural log of x.

UNIT III

**Formatting input and output**

scanf function is used for reading formatted input and printf statement is used for printing the output in a particular format.

The general syntax of printf and scanf are as follows

s**canf("control string",&arg1,&arg2,&arg3….&argn);**

arg1….argn - specify the address of the locations where the data is stored. It is generally preceded by an ampersand(&) symbol.

control string - %d (integer)
%f (float)
%c (char)

**printf("control string",arg1,arg2,arg3….argn);**

Control string specifies the field specifications consisting of the % symbol, data type, and an optional number specifying the field width. It also includes blanks, tabs, new lines.

***Conversion specifications for scanf***

a) The field specification for reading an integer number is %wd where w stands for the width. This width should be large enough to obtain the input data size.

b) Input items are separated by spaces or tabs and the format specifications should match the arguments in order.

c) An input value can be skipped by specifying * in the place of width

Example:

printf("%d %*d %d",&a,&b);

Let the data be as follows

2 8 10

then a=2 and b=10(since 8 is omitted).

d)The specification for integer %d
long int %ld
float %f or %e

double %lf
single character %c.
string %s.
octal %o
unsigned int %u
hexa decimal %x

e) Any unread data items will be considered to be the input for the next scanf.

**Example:**

```
char a[10],b[10];
……..
scanf("%s",a);
scanf("%s",b);
……
printf("%s %s",b,a);
```

Then if New York is the input given, the output of the printf statement will be **York New**

***Conversion specifications for printf:***

1. The format specification in the printf statement has the following form %w.p type specifier where

w – is an integer number that specifies the total number of columns for the output value

p - specifies the number of digits to the right of the decimal point (in a real number) or the number of characters to be printed from a string.

2. For integer numbers width can be specified. The number is usually right justified. It can be left justified by preceding the type specifier with a minus sign.
3. It is also possible to fill the leading blanks by zero using %0d
4. The output of a real number is enabled by the format %w.p f

where

w - total number of positions
p - number of digits after the decimal point.

5. Can also display in the exponential form using %w.p e
6. To print a single character use %c

**Example:**

| Statement | Output | |
|---|---|---|
| printf("%d",9876) | 9876 | integers |
| printf(%-6d"9876) | 9876 | |
| printf("%06d",9876) | 009876 | |
| printf("%7.4f",98.7654) | 98.7654 | real numbers |
| printf("%7.2f",98,7654) | 98.77 | |
| printf("%-7.2f",98,7654) | 98.77 | |
| printf("%10.2e",98.7654) | 9.88e+01 | |
| printf("%s","New York") | New York | strings |
| printf("%10s","New York) | New York | |
| printf("-10s","New York) | New York | |
| printf("10.5s","New York") | New Y | |
| printf("-10.5s","New York") | N | |

## DECISION MAKING AND BRANCHING

**If and If-Else statements**

There are a lot of situations where the order of execution has to change. This can be done by two ways

- By repetition

- By condition

There are four decision making or control statements as follows

**1.If statement**

2.Switch statement

3.Conditional operator statement

4.Goto statement

**If statement**

It is a powerful decision making statement to control the flow of execution. It is used as two-way decision statement.

*Syntax:*

```
if (expression)
```

It allows the system to evaluate the expression first and then, depending on whether the value is true or false it transfers the control to the particular statement.

***Different forms of IF statement***

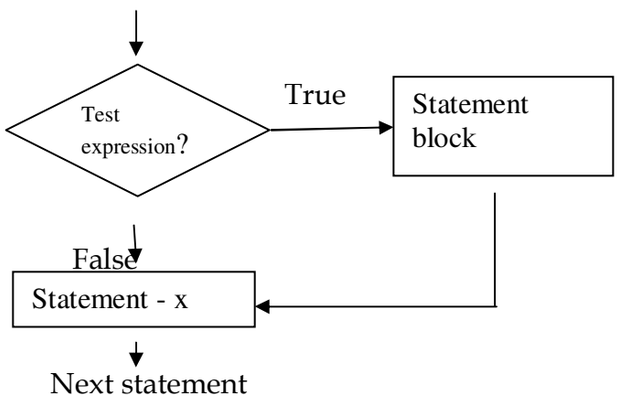
1. Simple if statement
2. If …else statement
3. Nested if … else statement
4. Else if ladder

**Simple if statement**

Syntax:

```
if (expression)
{
statement block;
}
statement-x;
```

In the above statement if expression is true then statement block will be executed otherwise statement block will be skipped.




**Flowchart for Simple If**

**Example:**

```
// Program to        find if a number is divisible by 7 or not
#include<stdio.h>
main()
{
        int n;
        printf("Enter the number");
        scanf("%d",&n);
                if ((n%2)== 0)
                        printf("The number is an Even number \n");
}
```

**Sample Output 1:**

Enter the number 2

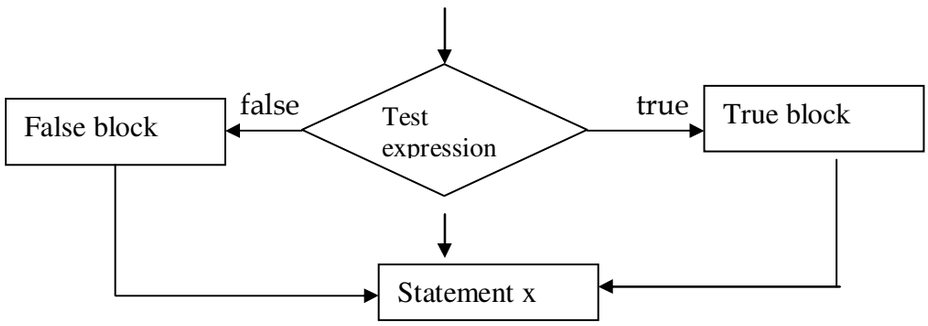
The number is an Even number.

**If …else statement**

**Syntax:**

```
if (expression)
{
true-block statement(s)
}
```

In the above statement, if the expression is true then true block is executed; otherwise the false block is executed. In either case either true block will be executed or false block will be executed but not both.




**Flowchart for If-Else**

**Example:**

```
// Program to find the greatest  of 2 numbers
#include<stdio.h>
void main()
{     int a,b;
        printf("Enter 2 numbers);
        scanf("%d %d",&a,&b);
        f(a>b)
              intf("%d is greater",a);
        else
              printf("%d is greater",b); }
```

**Output:**

Enter 2 numbers 2 3

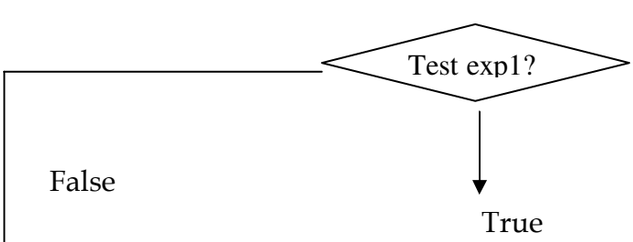
3 is greater

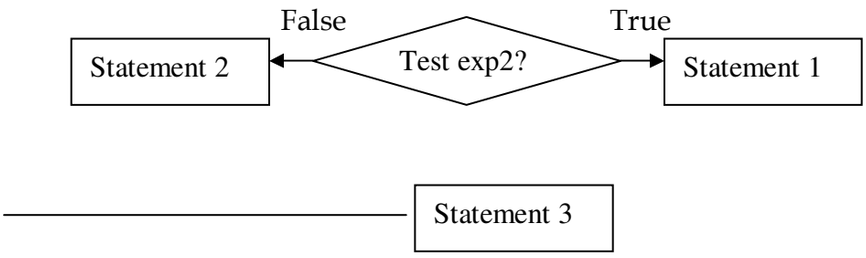
**Nesting of if ... else statement**

Syntax:

```
if (condition 1)
{
        if (condition 2)
        {
          statement 1;
        }                                  inner If
      else
        {
          statement 2;
        }
  }
else
{
        statement 3;
}
     statement –x;
```

In the above statement if test condition 1 is true then test condition 2 will be tested and if it is also true then statement 1 will be executed. If the second condition is false then statement 2 will be executed. If the first condition is false then statement 3 will be executed.






**Flowchart for Nested If**

**Example:**

```
//Program to find the greatest of 3 numbers
#include<stdio.h>
void main()
{
        int a,b,c;
        printf("Enter the 3 numbers");
        scanf("%d %d %d",&a,&b,&c);
        if( a>b)
        {
                if(a>c)
                printf("a is greater");
        else
                printf("c is greater"); }
else
{
        if(b>c)
                printf("b is greater");
        else
                printf("c is greater");
}
}
```

**Sample Output;**

```
Enter the 3 numbers  2   3   5
c is greater
Enter the 3 numbers  5   3   2
a is greater
Enter the 3 numbers  2   5   3
b is greater
```

**The Else … if ladder**

Syntax:

Else If ladder is used for multi path decisions (ie) a chain of if in which the statement associated with each else is another if. The condition are evaluated from the 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 statement x. When all the 'n' conditions become false then the final else containing the default-statement will be executed.

**Else If Ladder**

**Example:**

```
//Program to find the grade
#include<stdio.h>
```

```
void main()
{
        int marks;
        char grade;
        printf("Enter marks);
        scanf("%d",marks);
        if(marks>89)
                grade='O';
        else if(marks>79)
        grade='A';
        else if(marks>69)
        grade='B';
        else If(marks>40)
        grade='C';
        else
        grade='F';
        printf("Marks is %d and the grade is %c",marks, grade);
}
```

**Sample Output:**

Enter marks 89

Marks is 95 and grade is O

# UNIT - IV

**While and Do-While statements**

**Loop constructs**

It is used to repeat the block of statements for n number of times.

## *Process involved in looping statement is as follows:-*

1. Setting and initialization of counter.
2. Execution of the statements of the statements in the loop.
3. Test for a specified condition for execution of the loop.
4. Incrementing the counter.

**Types:**

1. While statement
2. Do statement
3. For statement

1) **While statement**

a. The while is an entry- controlled loop statement.

b. The test -condition is evaluated first and if the condition is true, then the body of loop is executed. Again the test-condition is tested and body of the loop will be repeated until the condition becomes false

c. If at the entry condition itself the test-condition is false then body of the loop will be skipped.

Syntax:

```
while ( test condition)
{
body of the loop;
}
```

**Example:**

```
#include <stdio.h>
void main()
{
     int  i=1;
     while (i>1)
     {
           printf("The number is one");
           i--;
     }
           printf("Thank you");
}
```

**Sample Output:**

Thank you

**2) The do statement:**

a. The do is an exit- controlled loop statement.

b. The body of loop will be executed once and the condition is tested if the condition is true then body will be repeated or else exit from the loop.

*Syntax:*

```
do
{
body of the loop
}
while (test-condition);
```

**Example:**

```
//program using the do statement
#include<stdio.h>
void main()
{
    int I=1;
    do
    {
        printf("the number is one\n");
    }while(I>1);
    printf("thank you");
}
```

**Sample output:**

The number is one

Thank you

**For Loop**

If we know the exact number of times for repetition of block of statements, then for statement can be used. It is entry controlled control loop.

Syntax:

```
for (initialization; test condition; increment)
{
  body of loop;
}
```

***Execution of 'for' statement as follows:***

1. Initialization of the control variable is done first using an assignment statement. The variables used for initialization are called control variables.
2. The value of the control variable is tested using the test condition. If the test condition is true then the block is executed otherwise the loop is terminated and the execution continues with the statement immediately following the loop.
3. When the body of the loop is executed, then the control is transferred back to the For loop. The control variable will be incremented by using the increment and again tested using the test condition. This will be repeated until the test condition becomes false.
4. It allows negative increments. In that case the initial value should be greater than the final value.
5. In normal situations initial value is lesser than the final value.
6. It allows more than one variable initialization as well as more than one variable increment.

**Examples:**

for (i=1; i<=10; i++) (positive increment)

for (i=10; i>0; i--) (negative increment)

for (i=0,j=8; i<=j; i++,j--) (multiple initializations, increment and decrements)

7. For statement is not followed by semicolon(;).
8. Nested loop is also possible. (ie.,) loop within a loop.

**Example :**

```
---------------
---------------
for (i=1;i<=n;i++)
{
for(j=1;j<=m;j++)
{
------------------
                  ------------------
}
}
------------------
```

**Example:**

```
//program to find the factorial of a number
//factorial of a number (n) = 1*2*3*…..*n
#include<stdio.h>
void main()
{
int n, i,fact =1;
printf("Enter the number…");
scanf("%d",&n);
for(i = 1; i<=n; i++)
fact = fact * i;
printf("\n The factorial of the given number %d is %d",n, fact);
}
```

**Sample Output:**

Enter the number…3
The factorial of the given number 3 is 6.

**Switch Statement**

Switch statement is used as an alternative for else-if ladder. The major disadvantage with else –if ladder is that when there are more number of alternatives, the else-if ladder grows. This reduces the readability of the program. Thus switch can be used as an alternative for else-if ladder for multi path decisions. The value of any variable or expression is used to select one branch among several alternatives. The switch tests the value of given variable (or) expression against a list of case values and when a match is found, a block of statements associated with that case is executed.

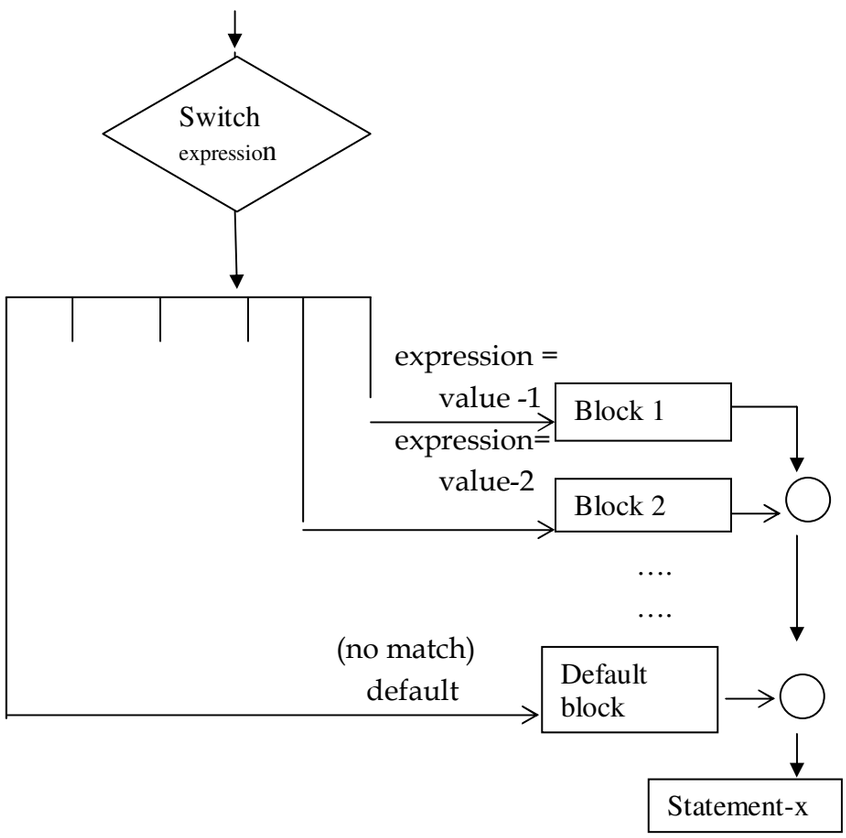
*Syntax:*

```
switch ( expression )
{
        case value-1:   block-1;
                        break;
         case value-2:  block-2;
                        break;
         …….

         default:       default-block;
                            break;
 }
 statement-x
```

In the above syntax the expression is an integer expression or characters. The value-1,value-2,…. are constants or constant expressions and are known as case labels.

Each of these values should be unique within a switch statement. block-1, block-2,….. are statements contain zero or more statements. There is no need to put braces around these blocks. The case labels must end with a colon. When a switch is executed, the value of expression is compared with case values, if matches, then the block of statements that follows the case are executed. The break statement is used to exit from the switch statement. Default is an optional case. When default is present, it will be executed if the value of the expression does not match with any of the case values. If it is not present then no action takes place and the control is transferred to statement –x.




**Flowchart for switch statement**

**Example:**

```
// program to implement a simple calculator
#include<stdio.h>
void main()
{
int a, b, c, choice;                    // c is a temporary variable
printf("1. Addition \n");
printf("2.  Subtraction \n");
printf("3.  Multiplication \n");
printf("4.  Division \n");
printf("Enter your choice……..");
scanf("%d",&choice);
printf("\n Enter 2 values");
switch(choice)
```

```
{
case 1: c=a+b;
          break;
case 2: c=a-b;
          break;
case 3: c=a*b;
          break;
case 4: c=a/b;
          break;
}
printf("The value = %d\n",c);
}
```

**Sample output:**

1. Addition
2. Subtraction
3. Multiplication
4. Division

Enter your choice…….1
Enter 2 values 3 4
The value = 7

**Conditional operator**

The conditional operator is used for two-way decisions. The operator is a combination of ? and : and takes three operands. The operator is otherwise called as the ternary pair.

Syntax:

| Conditional expression ? expression1; **:expression2** |
|---|

The conditional expression is evaluated first. If it is non zero then expression 1 is evaluated else expression 2 is evaluated. It is an alternative for the If-else statement.

**Example:**

```
//program to find the greater of 2 numbers
#include<stdio.h>
void main()
{
int a, b, great;
printf("Enter 2 numbers……");
scanf("%d %d",&a,&b);
great = (a>b) ? a : b;
printf("the greater value is %d",great);
}
```

**Sample output:**

Enter 2 numbers…….10 23
The greater value is 23

**Goto statement**

C supports the goto statement that branches unconditionally from one point of the program to another. The goto requires a label in order to identify the place where the branch is to be made. A label is a valid variable name that has to be followed by a semicolon. The label is placed before the statement where the control needs to be transferred.

Syntax:

| Goto label; |
|---|

**Example:**

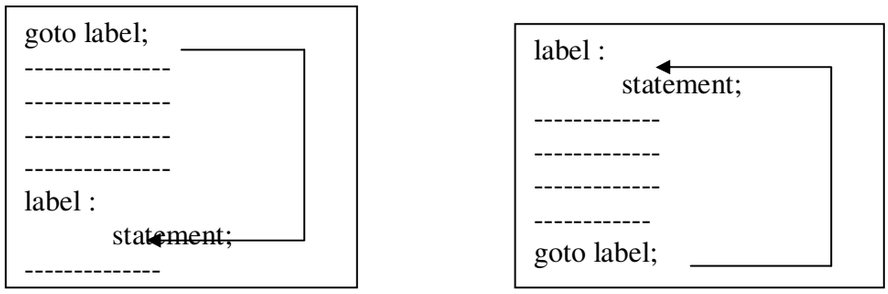
```
/*program that illustrates the use of goto statement */
#include<stdio.h>
void main()
{
int a, b;
a=16;
b=17;
if(a==b)
a++;
else
goto ERROR;

ERROR:
printf("Fatal error, exiting \n");
}
```

***Sample output:***

Fatal error,exiting

If the label is placed before the statement 'goto' then it is called a backward jump. If the label is placed after the statement 'goto' then it is called a forward jump.




Forward jump Backward jump

**Figure 2 – Types of goto**

# *UNIT V*

***Arrays***

***Definition :***

An array is a group of related data items that share a common name.

***Example:***

An array name ***salary*** can be used to represent a set of salaries of a group of employees.

A particular value in an array is called an element and can be written as follows

```
array_name [index];
```

salary [10] represents the salary of the 10th person.

**Types**

1. One dimensional arrays / Single subscripted array
2. Two dimensional array
3. Multi dimensional array

**One Dimensional Array**

***Definition :***

A list of related data items with one variable and only one subscript is called single- subscripted array.

**Declaration of arrays:**

***Syntax:***

```
Type variable_name[size] ;
```

type - int,float,double,long int,char

variable_name - valid c programming variable

size - no. of elements

***Example:***

1. int a[10];
2. float height[20];
3. char name[10];

***Explanation:***

1. In the first declaration we can store atmost 10 elements of type integers.
2. The second declaration can hold 20 elements of real numbers.

3. In third we can store atmost 9 characters. While declaring character array the last character must be a null character('\0'). This character is automatically included while assigning values to these variables. While mentioning about the size we must include one extra space with maximum number of characters.

**Initialization of arrays:**

***Syntax:***

```
static type array_name[size] = {list of values}
```

We can initialize the array variable like ordinary variables when they are declared. In the above list of values are separated by comma. Suppose the value is not given then automatically it will be assigned as zero.

***Example:***

a. *static float total[5] = {0.5,15.63,-20};*

In the above the first three elements are set to 0.5, 15.63, -20 and the remaining elements are set to zero.

b. static int count[]= {1,1,1,1};

In the above the size is omitted. The compiler itself allocates enough space for all initialized elements.

**Draw backs in initialization of arrays:**

1. There is no convenient way to initialize only selected elements.
2. There is no shortcut method for initializing a large number of array elements like the one available in FORTRAN.

**Two dimensional arrays**

***Definition***

A list of related data items with one variable and subscripts representing the row value and column value is called two dimensional array.

**Declaration of arrays**

***Syntax:***

```
Type  variable_name[row_size][column_size];
```

where

type - int, float ,double, longint,char

variable_name - valid c programming variable

size - no. of elements

***Example:***

1. int a[10][10];
2. float height[20][2];
3. char name[10][10];

***Explanation:***

1. In the first declaration we can store atmost 100 elements of integers.
2. The second declaration can hold 40 elements of real numbers.
3. In third we can store almost 10 names of 9 characters. While declaring character array the last character must be a null character('\0'). This character is automatically included while assigning values to these variables. While mentioning about the size we must include on extra space with maximum number of characters.

**Initialization of arrays**

***Syntax:***

```
static type array_name[row_size] [column_size] = {list
```

We can initialize the array variable like ordinary variables when they are declared. In the above list of values are separated by comma. Suppose the value is not given then automatically it will be assigned as zero.

***Example:***

a. static float mat[2][3] = {{0.5,15.63,-20},{10.2}};

In the above the first row elements are set to 0.5 , 15.63 , -20 and in the second row the first element is 10.2 and the remaining elements are set to zero.

**Multi dimensional arrays**

***Definition :***

A list of related data items with one variable and more than 2 two subscripts is called multi-dimensional array.

**Declaration of arrays**

***Syntax:***

```
Type array_name[s1][s2][s3]…[sn];
```

where

type -- int , float , double , long int , char

array_name -- valid c programming variable

**si -- size of ith dimension where i = 1,2,3…n**

**SAMPLE PROGRAM USING ARRAY:(ascending order)**

```
#include <stdio.h>
#include <string.h>
main()
{
int i, a[10], n, t;
/*INPUT THE NUMBER OF ELEMENTS*/
printf("Enter the total no of elements \n");
scanf("%d",&n);
/*ENTER THE VALUES TO BE SORTED*/
printf("Enter  elements \n");
for(i=1;i<=n;i++)
scanf("%d",&a[i]);
/*BUBBLE SORT*/
                for(i=1;i<=n-1;i++)
{
for(j=i;j<=n;j++)
{
if (a[i]>a[j])
{
t=a[i];
a[i]=a[j];
a[j]=t;
}
}
}
/*PRINT THE RESULT*/
printf(" sorted  elements \n");
for(i=1;i<=n;i++)
printf("%d",a[i]);
```

```
getch();
return(0);
}
```

## STRINGS AND CHARACTER ARRAYS

## INTRODUCTION

***Definition:***

String is an array of characters. Any group of characters enclosed between double quotes is a string constant.

***Example:***

"Where there is will there is way"

**Operations performed on strings:**

1. Reading and writing strings.
2. Combining strings together.
3. Copying one string to another.
4. Comparing strings for equality.
5. Extracting a portion of string.

**Declaration and initialization of string variables:**

***Syntax:***

```
char string_name[size];
```

***Example:***

1. char city[10];
2. char name[30];
3. static char city[9]= "new york";
4. static char city[9]={'n','e','w',',','y','o','r','k','\o'};

In the above 3 and 4 initializations are same. In character array the compiler automatically includes the null character at the end of the string.

If the size is not mentioned then the compiler itself defines the size of the array variable depending upon number of elements in the initialization statement.

**Reading strings from the terminal**

The scanf function can be used with the control string %s to read in a string of characters.

***Example:***

```
char name[15];

scanf("% s",name);
```

In the above example & is not required before the variable name.

***Drawback:*** We cannot read a line of text from the terminal.

***Reading a line of text:***

We can read the entire line of text using the function getchar().By using the above function it terminates the input process whenever it receives the newline character.

```
/* program to read a line of text from terminal*/
#include <stdio.h>
main()
{
char line[81],character;
int c;
c=0;
    printf("enter text .press <enter> at end\n");
do
{
character=getchar();
line[c]=character;
c++;
}
while (character!='\n');
    c=c-1;
    line[c]='\0';
    printf("\n%s\n",line);}
```

**Writing strings to the screen**

Printf statement with control string %s is used to display the string value.

***Example:***

printf("%s",name); (or) printf("%Total columns.Total characters s",name);

**Effect of various %s specification**

1. When the field width is less than the length of the string , the entire string is printed.
2. The integer value on the right side of the decimal point specifies the number of number of characters to be printed.
3. When the number of characters to be printed is specified as zero, nothing is printed.
4. The minus sign in the specification causes the string to be printed left- justified.

**Arithmetic operations on characters**

C allows us to manipulate the characters in the same way as numbers. Whenever a character constant or character variable is used it is automatically converted into integer value by the system. The integer value depends on the local character set of the system.

***Example:***

```
X='a';
printf("%d\n",x);
```

The output of the above statement will be 97.

Similarly it is also possible to perform all the arithmetic operations like addition ,subtraction , multiplication , division etc. To convert integer to ascii as well as ASCII to integer is done by using library functions like itoa(integer),atoi(string).