WWW.KTUNOTES.IN

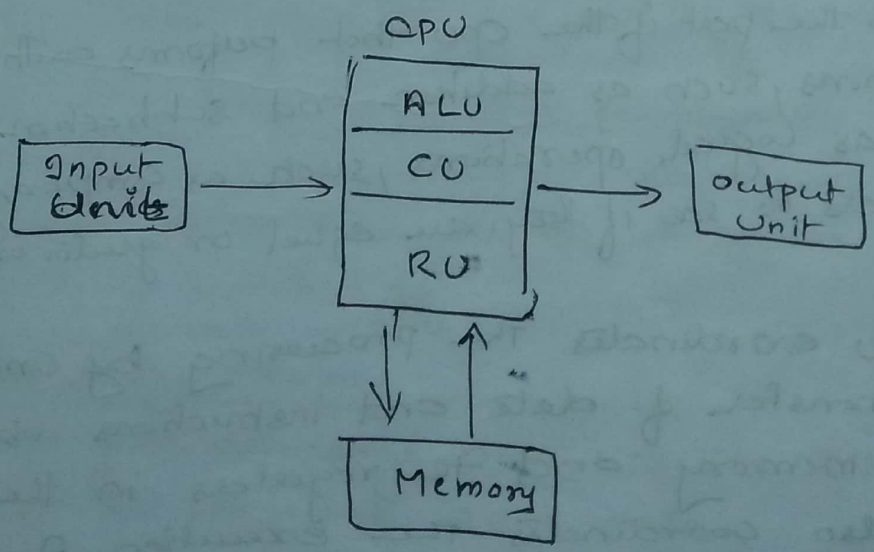
Computer

The oxford dictionary defines computer as "an automatic electronic apparatus for making calculations or controlling operations that are expressible in numerical or logical terms"

- A computer is a data processor

It can accept input, which may be either data or instructions or both. The computer remembers the input by storing it in memory cells. It then processes the stored input by performing calculations or by making logical comparisons or both. It gives out the result of the arithmetic or logical computations as output information. The computer accepts inputs and outputs data in an alphanumeric form. Internally it converts the input data to meaningful binary digits, performs the instructed operations on the binary data, and transforms the data from binary digit form to understandable alphanumeric form.

Functional parts of the computer



1) Input devices.

The data and instructions are typed, submitted or transmitted to a computer through input devices. Input devices are electronic or electro-mechanical equipment that provide a means of communicating with the computer system for feeding input data and instructions

eg: keyboard, Mouse, scanner, etc.

2) Output devices

Output devices are used to display information after processing the instructions by the computer.

eg: Monitor, printer, etc.

3) Central processing unit / CPU

CPU can be thought of as the brain of the computer. Most of the processing takes place in CPU. The CPU itself can be divided into different functional units which are described below.

(1) Arithmetic and logic unit (ALU)
(2) Control unit (CU) (3) Register unit (RU).

ALU :-

It is the part of the CPU that performs arithmetic operations, such as addition and subtraction as well as logical operations, such as comparing two numbers to see if they are equal or greater or less.

CU :-

The CU coordinates the processing by controlling the transfer of data and instructions b/w main memory and the registers in the CPU. It also coordinates the execution of ALU to perform operations on stored data stored

in particular registers.

Register unit:

CPU's current instructions and data values are stored temporarily inside a high-speed memory location called RU. It is also called Cache memory.

4) Memory unit:

It is used to store the data.

Memory unit contains an ordered sequence of storage locations called memory cells and each memory cell has a unique address that indicates relative position in memory.

- Data stored in a memory cell are called contents of the cell. Data are stored in memory as binary digits, called bits. Each memory location comprises of a single byte which is equal to eight bits
- Mainly two types of memory that are used in a computer systems.

(1) primary memory / main memory:

It stores programs, data and results.

Most common types of main m/m are RAM and ROM

RAM: It offers temporary storage of programs & data. It allows both read and write operations. RAM is a Volatile memory. Since, everything in RAM will be lost when computer is switched off

ROM: stores programs or data permanently. It allows only Read operation. Rom is non-volatile m/m, since the data stored there do not disappear when the computer is switched off.

(2) Secondary memory:

It provides large, non-volatile & inexpensive storage for programs and data.

eg: Hard disks, Floppydisks, CD, DVD.

Components of a computer system fall into two major categories:

- Hardware
- Software.

Hardware

Hardware is physical parts of the computer. The parts are possible to touch and are visible.

eg: keyboard, mouse, speakers.

Software:

S/w is a collection of programs. A program is a series of instructions which is intended to direct a computer to perform certain functions and is executed by the processor.

Software is classified into:

(1) System software

Is designed to facilitate & coordinate the use of the computer by making h/w operational.

eg: Language translators, OS, Loader, linker, etc.

(2) Application s/w

Is designed to perform specific usages of the users

eg: MS word, MS excel, photoshop, MS powerpoint

- programs are designed based on computer languages.

eg: pascal, fortan, C, C++, java.

Generally computer languages are classified into 3 types:

→ Machine Language

→ Assembly Language

→ High level Language.

Tokens in C

Character set in C

Expln: (Whenever we write any C pgm then it consists of diff strings. Each string is set of words and these words are constructed using characters from C character set)

Defn: It denotes any alphabet / digit or special symbol used to represent information.

Lower case letters - a-z
Upper case letters - A-Z
Digits - 0-9
Special characters - @, #, $, %
White Spaces - Tabs or Newline or space

→ C Language supports a total of 256 characters.
→ Every character in C language has its equivalent ASCII value
(ASCII - American standard code for Information interchange)

A - 65 ... Z - 90
a - 97 ... z - 122.

Variables:

Defn: Variable is an identifier (name) for a memory location in which data can be stored and subsequently recalled.

- Every variable is mapped to a unique m/m address.
- All variables have 3 important attributes:

① Datatype:- that is established when the variable is defined. once defined the type of a C variable cannot be changed.
eg: int, char, float, double

② name of the variable. Variable names are case sensitive i.e NUM & num are distinct.

③ Value:- that can be changed by assigning a new

Value to the Variable. The kind of values a variable can assume depends on its type.

→ In C, a variable must be declared before it can be used.

→ The number of characters that you can have in a variable name will depend upon your compiler. A minimum of 31 characters must be supported by a compiler that conforms to the C language.

Rules for Constructing Variable name.

(1) Characters allowed are:
- Underscore (_)
- Small letters (a-z)
- Capital letters (A-Z)
- Digits (0-9)

(2) Blank and commas are not allowed

(3) No special symbols other than underscore (_) are allowed

(4) First character should be a alphabet or underscore

(5) Variable name should not be reserved word.

(6) Variable names are case sensitive.

[we have declared variable means "we have created one empty container which will hold data"].

Variable declaration :-

Datatype variablename;

eg: int a; char b; float c;

→ Declaration is used for identification of Datatype

→ Redeclaration of variable will cause compile error.

Error
```
int ivar;
int ivar = 10;
```

→ We cannot use variable inside program expression or in program statements without declaring.

→ Declaring a variable just give rough idea to compiler that there is something which is of type "<Data Type Specified>" and will be used in program.

Variable Data Types:
- Integer
- Character
- float
- Double.

Initialization of a Variable:-

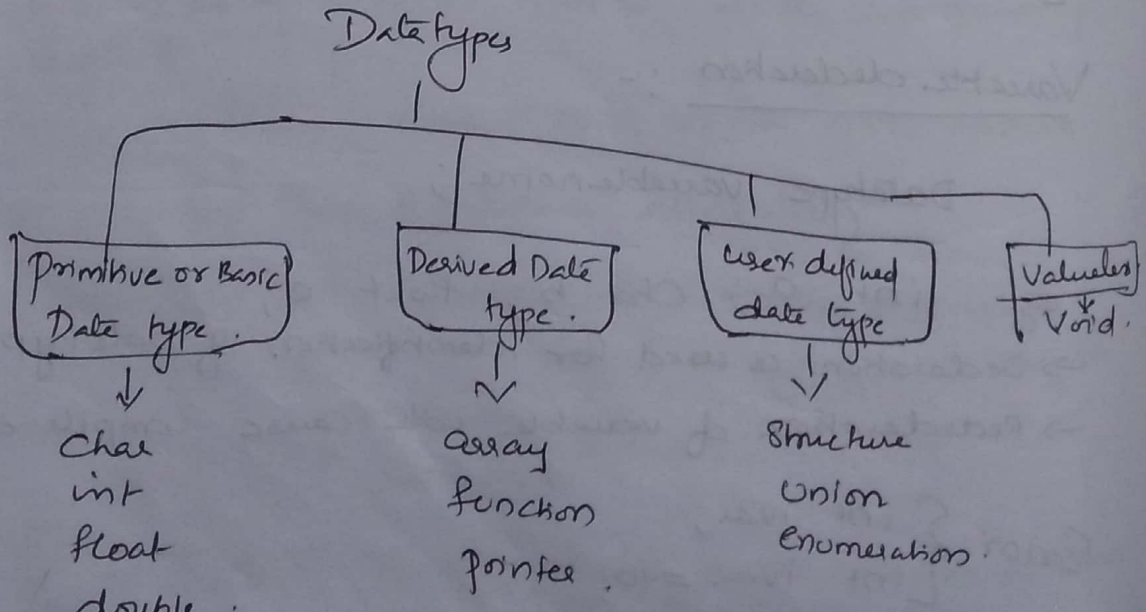
Variable name = Value;

int a = 10;

KTU 2016 Q

Data types in C

The type or data type of a variable determines the set of values that a variable might take and the set of operations that can be applied to those values.



For a 16-bit compiler

	Datatype	Size (byte)	Range
①	char	1 byte	−128 to 127
②	int	2 Bytes	−32768 to 32767
③	float	4 Bytes	1.17549×10^{-38} to 3.40282×10^{38}
④	double	8 bytes	2.22507×10^{-308} to 1.79769×10^{308}
⑤	void	1 byte	Valueless

Char: A char variable occupies a single byte that contains the code for the character. This code is a numeric value and depends on the character coding system being used. i.e: it is machine dependent.

Sign specifiers: - signed & unsigned

Int: - used to store whole numbers.

Sign specifiers: - signed & unsigned

Size specifiers: - short & long.

float: used to store real numbers. floats are stored in four bytes and are accurate to about seven significant digits. (used to store decimal numbers with single precision)

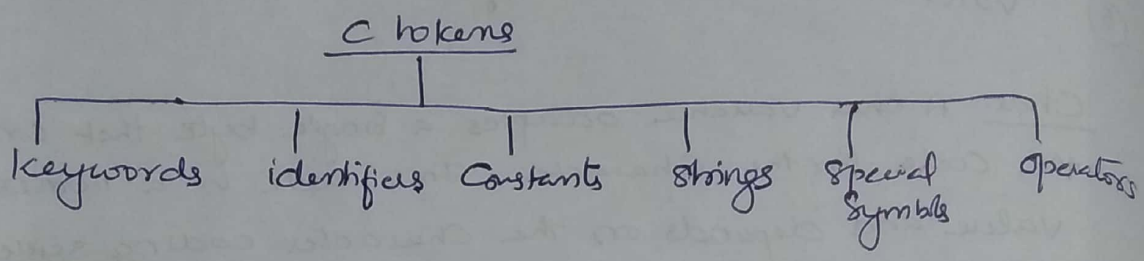
double: Allocates the storage of double precision floating point numbers that are held correct to 15 digits, & have a much greater range of definition than floats (used to store decimal numbers with double precision).

Size specifier: long double.

C tokens

In a C program the smallest individual units are known as C tokens.

C has six types of tokens:



Keywords

KTU 2016

They are reserved words. All keywords have fixed meanings and these meanings cannot be changed. Keywords serve as basic building blocks for program statements. All keywords must be written in lowercase.

eg: char, int, float, double, void, short, long, signed, unsigned, do, while, for, if, else, break, struct, typedef, etc.

Identifiers:

Identifiers refer to the names of variables, functions and arrays. These are user defined names and consist of a sequence of letters & digits, with a letter as a first character. The underscore character is also permitted in identifiers. It is usually used as a link between two words

in long identifiers.

Rules for identifiers

1. First character must be an alphabet (or underscore)
2. Must consist of only letters, digits or underscore.
3. only first 31 characters are significant.
4. Cannot use a keyword.
5. Must not contain white space.
7. Identifiers are case sensitive.

eg: Valid identifiers:

abc, SUM, SUM123, _abc, sum_nos

Invalid identifiers

1abc, *abc, MAT add

Constants :-

Constants in C refers to fixed values that do not change during the execution of a program.

C supports several types of constants:

→ Integer constants → Real constants
→ Character constants → String constants

- **Integer constants:-**

An integer constant refers to a sequence of digits. There are three types of integers

- Decimal integer constants
- Octal " "
- Hexadecimal " "

Decimal integers

consist of a set of digits, 0 through 9, preceded by an optional – or + sign.

eg: +123 -321 0 654321 +78

Embedded spaces, commas, and non digit characters are not permitted between digits.

Invalid: eg: 15 750 20,000 $1000

Octal Integer Constant

Consist of any combination of digits from the set 0 through 7, with a leading 0.

eg: 037, 0, 0435, 0551

Hexadecimal Integer Constant

A sequence of digits preceded by 0x or 0X is considered hexadecimal integer. They may include Alphabets A through F or a through f. The letters A through F represent the numbers 10 through 15.

eg: 0X2 0x9F 0Xbcd 0X

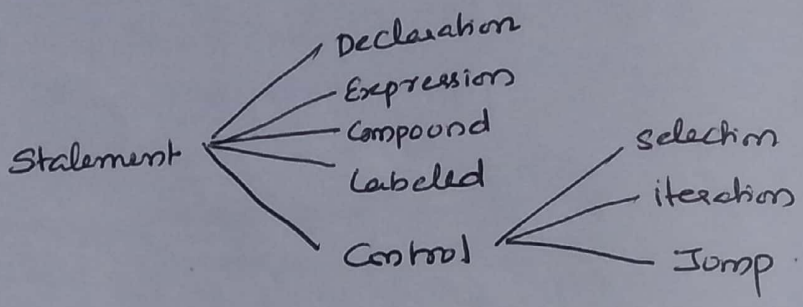
- Real constants:

WWW.KTUNOTES.IN

Program statement

A statement is a syntactic construction that performs an action when a program is executed.

- All C program statements are terminated with a semicolon (;)
- A program statement in C can be classified as shown below



Declaration

Is a program statement that saves to communicate to the language translator information about the name and type of the data objects needed during program execution.

eg: int x;

Expression Statement

Is the simplest kind of statement which is no more than an expression followed by a semicolon. An expression is a sequence of operators and operands that specifies computation of a value.

x == 4 is just an expression
but x=4; is a statement.

Compound Statement

Is a sequence of statements that may be treated as a single statement in the construction of larger statements.

Labeled Statements

Can be used to mark any statement so that control may be transferred to the statement by "switch" statement.

control Statement

Is a statement whose execution results in a choice being made as to which of two or more paths should be followed. In other words, the control statements determine the 'flow of execution' in a program.

(1) Selection Statements

Allow a program to select a particular execution path from a set of one or more alternatives. Various forms of the if, else statement belong to this category.

(2) Iteration Statement

Are used to execute a group of one or more statements repeatedly. while, for, and do..while statements fall under this group.

(3) Jump statements :-

Cause an unconditional jump to some other place in the program. goto, break and continue falls under this group.

Control Statements in C.

- Statements in a program are normally executed one after another until the last statement completes.
- A C application begins executing with the first line of the main() function and proceeds statement by statement until it reaches to the end of the main() function.
- Any sequence of statements can be grouped together to function as a syntactically equivalent single statement by enclosing the sequence in braces. This grouping is known as statement block or compound statement.

eg:
```
#include <stdio.h>
int main (void)
{
 int a=5;
 printf ("a = %d", a);
 /* A statement block follows */
 {
 int b=10;
  printf ("\n a = %d", a);
  printf ("\n b = %d", b);
 }
printf ("\n a = %d", a);
return 0;
}
```

Output

```
a=5
a=5
b=10
a=5
```

- The order in which statements are executed in a running program is called the flow of control.

Selection Statements

Selection is used to take a decision between one or two or more alternatives. Decision in a program is concerned with choosing to execute one set of statement over the others. Selection is also known as branching. Each decision is based on a boolean expression (also called a condition or test expression), which is an expression that evaluates to either true or false. The result of the expression determines which statement is executed next.

eg: if, if - - else, ~~else if~~; Nested if, if ... else if, switch

(1) if Statement

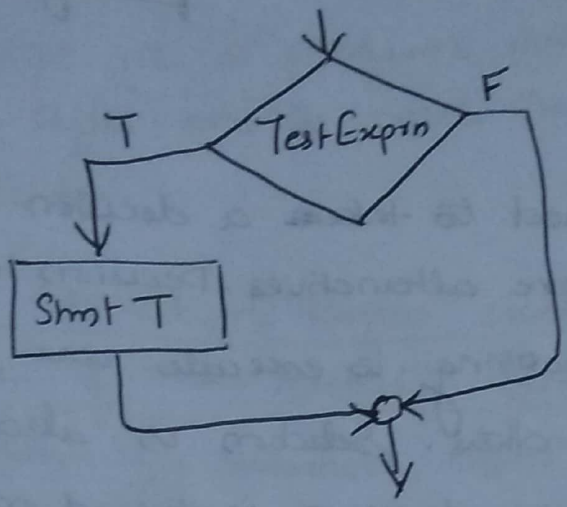
if statement ~~do~~ either do some particular thing or do nothing at all. The decision is based on a 'test expression' that evaluates to either true or false. If the test expression evaluates to true, the corresponding statement is executed. If the test expression evaluates to false, control goes to the next executable statement.

Syntax

```
if (TestExpr)
    Stmt T;
```

TestExpr is the test expression. Stmt T can be a single statement or a block of statements enclosed by curly braces { }

flowchart



Example

Write a program that prints the largest among 3 nos.

```
#include <stdio.h>
int main()
{
  int a,b,c,max;
  printf("Enter 3 nos:");
  scanf("%d %d %d",&a,&b,&c);
  max=a;
  if(b>max)
    max=b;
  if(c>max)
    max=c;
  printf("Largest no is:%d",max);
  return 0;
}
```

output

Enter 3 nos: 10 9 17

Largest no is: 17

if-else Statement

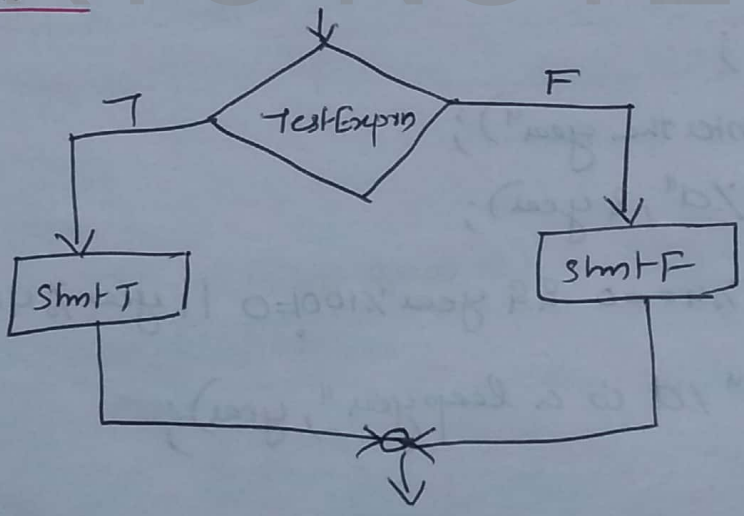
(2) if-else statements either do one particular thing or do another. The decision here is based on a test expression.

Syntax

```
if (TestExpr)
    StmtT;
else
    StmtF;
```

If the test expression TestExpr is true, StmtT will be executed; if the expression is false, StmtF will be executed. StmtT and StmtF can be single or a block of statements. Block of statements are enclosed within curly braces {}.

Flowchart



Example

Q. Write a program to check whether a given number is odd or even.

```
#include <stdio.h>
main()
{
    int n, r;
```

```
printf ("Enter the number");
scanf ("%d", &n);
    r = n % 2;
if (r == 0)
    printf ("Even");
else
    printf ("odd");
}
```

Output

Enter the number 7
odd
Enter the number 8
Even

Q. write a program that determines if a year is a leap year.

```
#include <stdio.h>
main ()
{
int year;
printf ("Enter the year");
scanf ("%d", &year);
if (year%4==0 && year%100!=0 || year%400==0)
printf ("%d is a leap year", year);
else
printf ("%d is not a leap year", year);
}
```

Output:-

Enter the year 1995

1995 is not a leap year.

Enter the year 2000

2000 is a leap year.

Enter the year 1800

1800 is not a leap year.

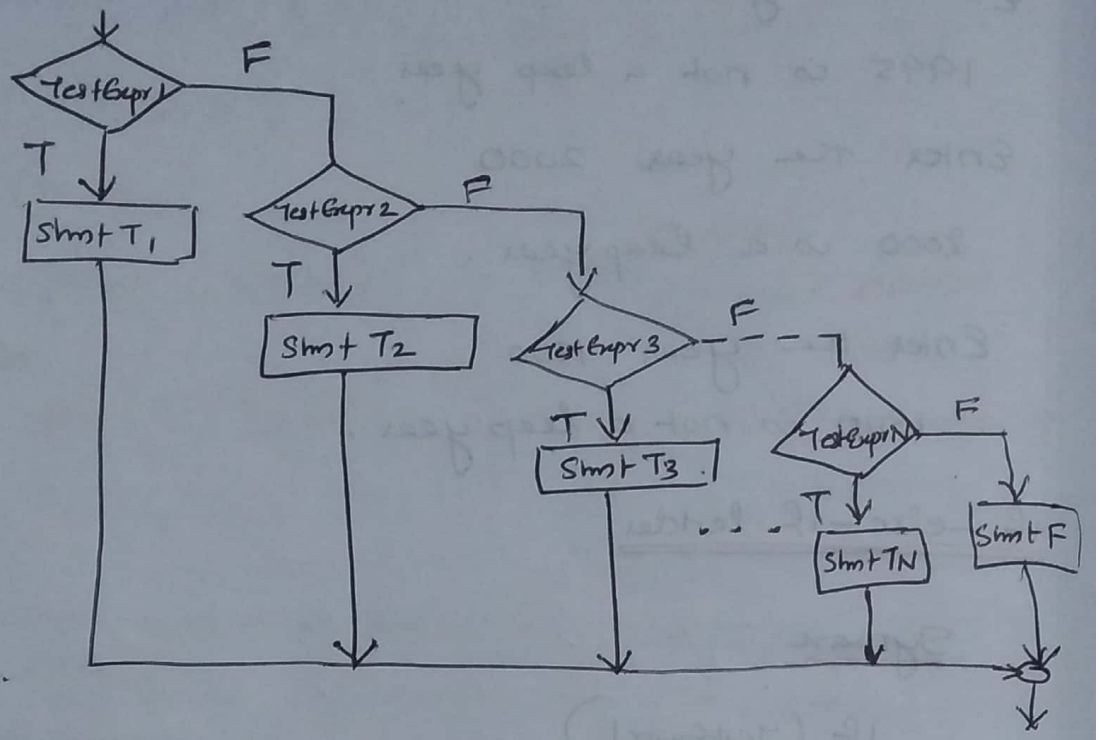
if-else-if ladder

Syntax

```
if (TestExpr1)
    Stmt T1;
else if (Test Expr 2)
    Stmt T2;
else if (Test Expr 3)
    Stmt T3;
    .
    .
    .
else if (Test expr N)
    Stmt TN;
else
    Stmt F;
```

If the first test expression TestExpr1 is evaluated to true, then stmt T_1 is executed. If the second test-expression TestExpr2 is true, then stmt T_2 is executed and so on. If none of the test expressions are true, then the statement Stmt F is executed.

Flowchart.



Example:

A program to check whether a number given by the user is zero, positive or negative

```
#include <stdio.h>
main ()
{
 int num;
 printf("Enter the Number:");
 scanf("%d",&num);
 if (num>0)
   printf("%d is positive\n",num);
 else if (num==0)
     printf("%d is zero\n",num);
 else
     printf("%d is negative",num);
}
```

Output

```
Enter the Number: 7
7 is positive
Enter the Number: 0
0 is zero.
```

- A program to print the grade according to the score secured by a student.

```
#include <stdio.h>
main()
{
  int score;
  Char grade;
  printf("Enter score:");
  Scanf("%d", &score);
  If (score >= 90)
      grade = 'A';
  else If (score >= 80)
      grade = 'B';
  else if (score >= 70)
      grade = 'C';
  else if (score >= 60)
      grade = 'D';
  else
      grade = 'F';
  printf("Grade is: %C", grade);
}
```

Output.

Enter score: 72.

Grade is: C

Nested if :-

When any if statement is written under another if statement this cluster is called a nested if.

Syntax :

```
if (TestExpr A)
    if (TestExpr B)
        Stmt BT;
    else
        Stmt BF;
else
    Stmt AF;
```

or

```
if (Test expr A)
    if (Test expr B)
        Stmt BT;
    else
        Stmt BF;
else
    if (Test Expr C)
        Stmt CT;
    else
        Stmt CF;
```

Example

program to find the largest of 3 nos:

```
#include <stdio.h>
main ()
{
int a,b,c;
printf("Enter 3 nos:");
scanf("%d %d %d",&a,&b,&c);
if (a>b)
    if (a>c)
        printf("\n Largest= %d", a);
    else
        printf("\n Largest= %d", c);
else
    if (b>c)
        printf("\n Largest= %d", b);
```

```
else
    printf ("\nlargest = %d", c);
}.
```

Output

Enter 3 nos : 10 9 17

largest = 17.

Conditional operator

Consider the situation in which there are two or more alternatives for an expression. The conditional operator of C is specifically tailored for such situations.

Syntax

expr1 ? expr2 : expr3.

It executes by first evaluating expr1, which is normally a relational expression, and then evaluates either expr2, if the first result was true, or expr3, if the first result was false.

example

Write a C program to find larger of two integer numbers using conditional operator.

```
#include <stdio.h>
main()
{
  int a,b,c;
printf ("Enter two numbers");
scanf ("%d %d ",&a,&b);
    c = a>b ? a : b;
```

```
printf ("\n larger number is : %d", c);
}.
```

Output:

```
Enter two numbers: 3   5
larger number is : 5
```

For finding max of 3 nos

```
#include <stdio.h>
main ()
{.
  int a, b, c, max;
  printf ("Enter the 3 numbers");
  Scanf ("%d %d %d", &a, &b, &c);
  max = a>b ? a>c ? a : c : b>c ? b:c;
  printf ("\n largest number is : %d", max);
  }.
```

Output:

```
Enter the 3 numbers :- 90   12   9
Largest number is : 12.
```

→ The use of conditional expression frequently shortens the amount of source code that must be written.

(3) **Switch Statement.**

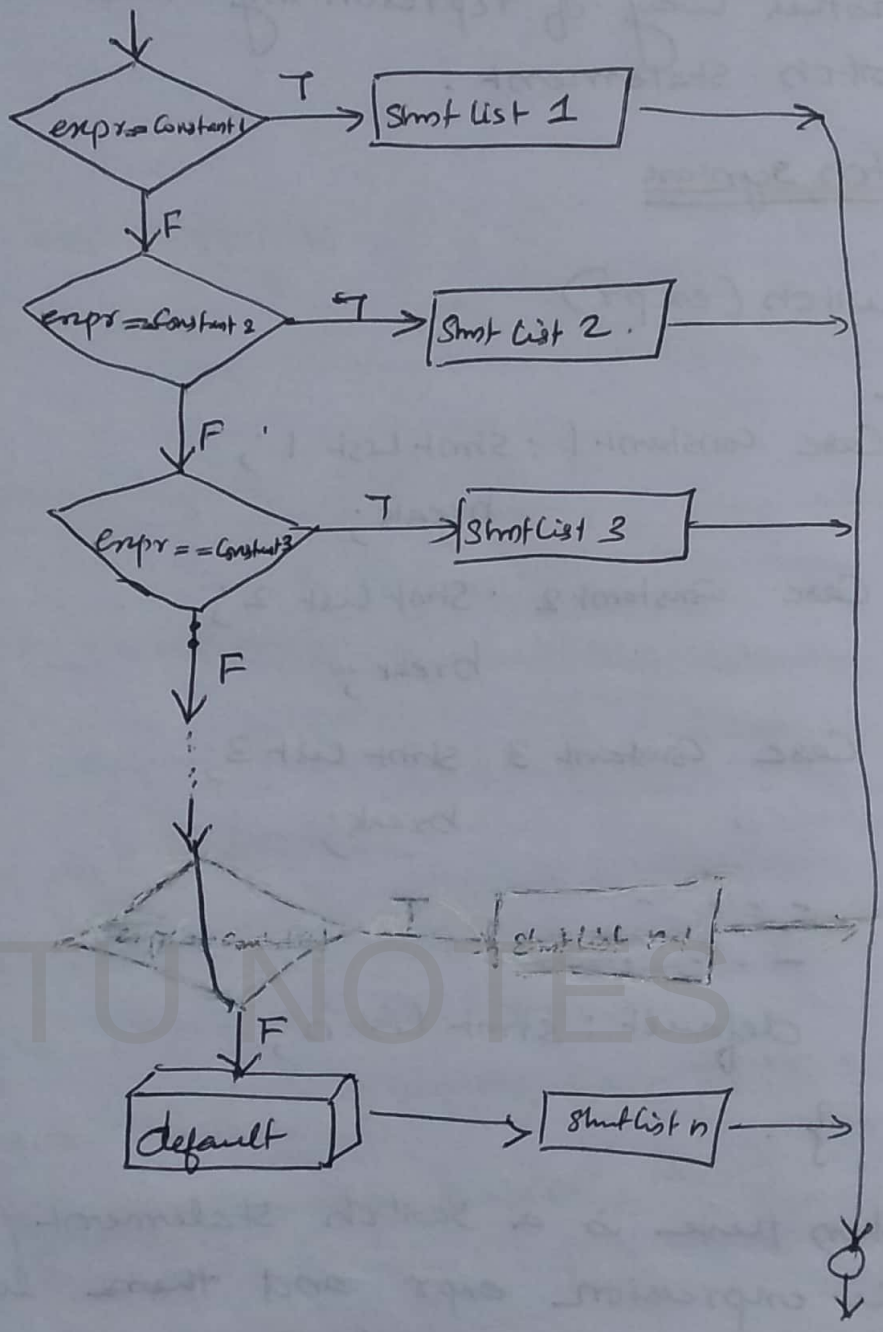
When there are a number of else alternatives another way of representing this is by the switch statement.

Switch Syntax

```
switch (expr)
{
 Case Constant-1 : Stmt List 1;
                   break;
 Case Constant-2 : Stmt List 2;
                   break;
 Case Constant-3 : Stmt List 3;
                   break;
  - - -
  - - -
 default : Stmt List n;
}.
```


When there is a switch statement, it evaluates the expression expr and then looks for a matching case label. If none is found, the default label is used. If no default is found, the statement does nothing. The break statement at the end of each block signals the end of a particular case and causes the exit from the switch statement.

flowchart



→ The break Statement must be used within each case if one does not want the following cases to execute after one case is selected. When the break statement is executed within a switch, C executes the next statement outside the switch construct.

example

Q. write a progeam to carry out the arithmetic operations addition, subtraction, multiplication and division between two variables :

```
#include <stdio.h>
main ()
{
 int value1, value2;
  char operator;
  printf ("Type in your expression\n");
   scanf ("%d %c %d", &value1, &operator, &value2);
   switch (operator)
   {
     case '+' : printf ("%d\n", value1 + value2);
                break;

     case '-' : printf ("%d\n", value1 - value2);
                break;

     case '*' : printf ("%d\n", value1 * value2);
                break;
     case '/' : printf ("%d\n", value1);

                If (value2 == 0)
                printf ("division by zero\n");
                 else.
                 printf ("%d\n", value1 / value2);
                 break;
               default : printf ("Unknown operator\n");
                 break;
   }
}
```

Q. Write a program that checks whether a character entered by the user is a vowel or not.

```
#include <stdio.h>
main()
{
char c;
printf("Enter a character");
scanf("%c", &c);
switch (c)
{
  case 'a' : case 'A' :
  case 'e' : case 'E' :
  case 'i' : case 'I' :
  case 'o' : case 'O' :
  case 'u' : case 'U' :
printf("%c is always a vowel \n", c);
break
default : printf("%c is not a vowel \n", c);
          break;
}
}
```

Note:
- The switch statement enables you to choose one course of action from a set of possible actions, based on the result of an integer expression.
- The case labels can be in any order and must be constants.

- No two case labels can have the same value.
- The default is optional and can be put anywhere in the switch construct.
- The case constant must be integer or character constants. The expression must evaluate to an integral type.
- The break statement is optional. If a break statement is omitted in any case of a switch statement, the program flow is followed through the next case label.

Iteration :-

A loop allows one to execute a statement or block of statements repeatedly.

- There are mainly two types of iterations or loops.
  - unbounded iteration or unbounded loop.
  - Bounded iteration or bounded loop.

Unbounded :-

There are many occasions when one does not know ahead of time, how many iterations may be required. Such ~~iterations~~ occasions require unbounded loops.

eg: while loop & do..while loop

These loops are also known as indeterminate or indefinite loops.

Bounded :-

Repetition is implemented by constructs that allow a determinate number of iterations. That is, bounded loops should be used when we know, ahead of time, how many times we need to loop.

eg: for loop.

- A loop can either be a pre-test loop or a post-test loop.

Pre-test loop :-

In a pre-test loop, the condition is checked before the beginning of each iteration. If the test expression evaluates to true, the statements associated with the pre-test loop construct are executed and the process is ~~one~~ repeated till the test expression becomes false. On the other hand, if the test expression evaluates to false, the statements associated with the construct are skipped and the statement next to the loop is executed. So for such a construct the statements associated with the construct may not be executed even once.

eg: while, for.

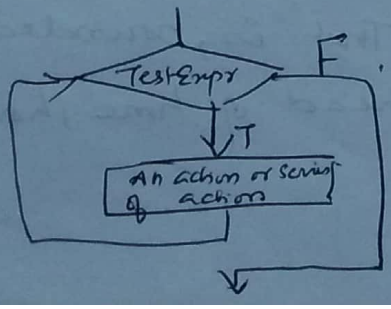
Post-test loop

In the post-test loop, the code is always executed once. At the completion of the loop code, the test expression is tested. If the test expression evaluates to true, the loop repeats; if the expression is false the loop terminates.

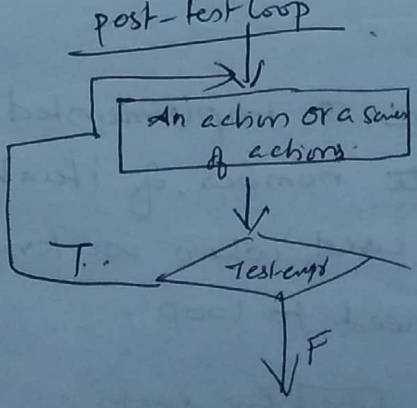
eg: do.. while.

Flowchart

Pre test loop



post-test loop



(4) ## While Construct

While statement is a pre-test loop. It uses a test expression to control the loop. Since it is a pre-test loop, it evaluates the test expression before every iteration of the loop.

Syntax

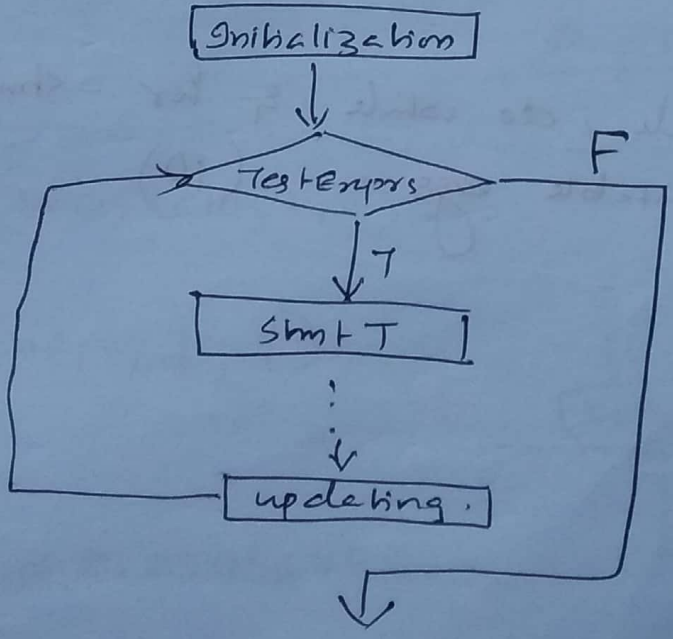
```
while (TestExpr)
{
  body of the loop
}
```

⟹

```
Initialization
while (TestExpr)
{
  stmt T
  . . .
  updating
}
```

To use the while statement, the test expression should contain a loop control variable. The initialization of the loop control variable has to be done before the loop starts and updating must be included in the body of the loop.

Flowchart



Program

Q. Write a C program to print the sum of digits of a number.

```
#include <stdio.h>
main()
{
  int n, t, d, S=0;
  printf("Enter the number:");
  scanf("%d", &n);
    t=n;
  while(n>0)
  {
     d=n%10;
     S=S+d;
     n=n/10;
  }
  printf("\n sum of digits of %d is %d", t, S);
}.
```


KTU 2016

Discuss while / do while & for stmt using suitable egs. (10)

(5) For Construct

used if the number of times a statement will be executed is known in advance.

Syntax

```
for (initialization ; Test-Expr ; updating)
    {
    body of the loop;
    } .
```

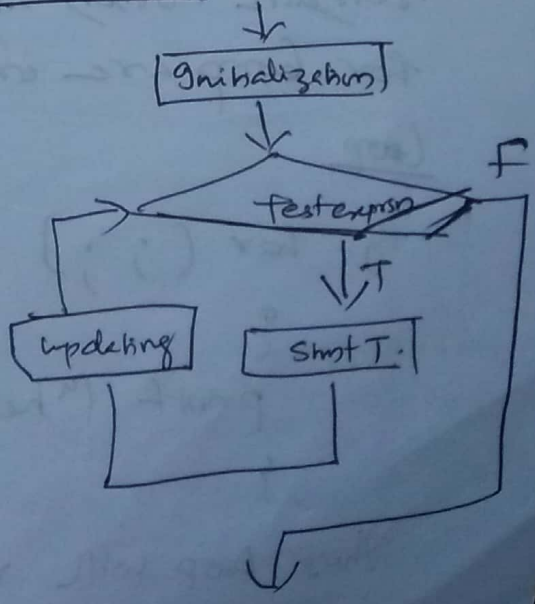
Initialization: This part of the loop is the first to be executed. The statement(s) of this part are executed only once. This statement involves a loop control variable.

Test-Expr:- represents a test expression that must be true for the loop to continue execution.

Stmt: is

updating:- The statements contained here are executed every time through the loop before the loop condition is tested. This statement also involves a loop control variable.

flowchart



Eg:

```
for(i=0; i<10; i++)
{
 printf("%d", i);
}.
```

Op: 0 1 2 3 4 5 6 7 8 9

- for (i=0 ; i<=10 ; i++)
  Can also be written as :

① i=0;
  for (; i<=10 ; i++)

②
```
for (i=0; i<=10; )
{
  ...
  i++;
}
```

③
```
i=0;
For( ; i<=10; )
{
  ...
  i++;
}
```


- Any or all of these expressions in a for loop can be omitted, but the two semicolons must remain. When all three expressions in a for loop are omitted, it acts as a <u>infinite loop</u>

eg:
```
for ( ; ; )
{
printf ("hello\n");
}
```
This loop will run forever.

- Multiple initializations should be separated with a comma operator :

eg: for (c=1, s=0 ; c<=n ; c++)

s=s+c;

- Multiple relational expressions in the test expression must be connected using logical operator && or ||

program to calculate sum of positive integers upto n.

1+2+3+...+n.

```
# include <stdio.h>
main ()
{
  int n, count, sum=0;
printf ("Enter the limit:");
scanf ("%d", &n);
for (count=1 ; count<=n; count++)
   {
      sum = sum + count;
   }
printf ("sum=%d", sum);
}
```

Output

Enter the limit = 10

sum = 55

1+2+3+4+5+6+7+8+9+10

factorial

```
#include <stdio.h>
main()
{
 int n, i;
 long int f = 1;
 printf ("Enter the number:");
  scanf ("%d", &n);
 for (i=1; i<=n ; i++)
  {
    f = f*i;
  }
printf ("factorial of the number=%ld");
printf ("factorial of the number= %ld", f);

}.
```

Do-while Construct

(6)

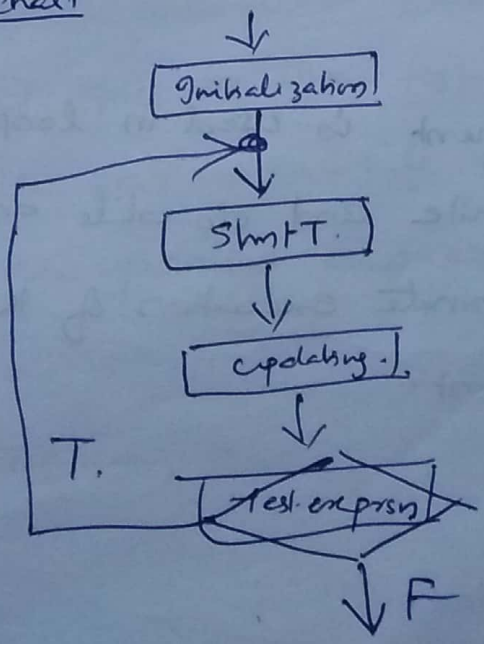
do-while loop is a control flow statement that executes a block of code atleast once, and then repeatedly executes the block, or not, depending on a given boolean condition at the end of the block.

Syntax

```
do
{
  Stmt T; /* body of statements could be placed here */
} while (Test Expr);
```

First, the code within the block is executed, and then the condition is evaluated. If the condition is true the code within the block is executed again. This repeats until the condition becomes false. Because do while loop check the condition after the block is executed, this control structure is often also known as post-test loop.

flow chart



Program:-

```
#include <stdio.h>
main ()
{
 int c=5;
  do
  {
   printf ("Hello");
    c++;
  }while (c<5);
}
```

Op : Hello

Special control statements:

There are certain control statements which terminate either a loop or a function. There are 3 such statements, namely, return, break and continue.

(1) break

KTU 2016

The break statement is used in loop constructs such as for, while and do-while and switch statement to terminate execution of the loop or switch statement.

Syntax

```
break;
```

After a break statement is executed within a loop or a case in a switch construct, execution proceeds to the statement that follows the loop construct or switch statement.

eg:
```
#include <stdio.h>
main()
{
int c=1;
while (c<=5)
{
 if (c==3)
  break;
 printf("\t%d",c);
  c++;
}
}
```


O/p 1 2.

(8) Continue statement

KTU 2006

The continue statement does not terminate the loop but goes to the test expression in the while and do-while statements and then goes to the updating expression in a for-statement

Syntax:

```
continue;
```

The Jumps by continue in different pre-test and post-test loops is as shown below:

```
while (testexpr)
{
  ==
  continue;
  ==
}
```

```
do
{
  ==
  continue;
  ==
} while (Testexpr);
```

```
for (initialization ; Testexpr ; updating)
{
  ==
  continue;
  ==
}
```

KTU 2016

	Break	Continue
1.	It helps to make an early exit from the block when it appears.	It helps in avoiding the remaining statements in a current iteration of the loop and continues with the next iteration.
2.	It can be used in all control statements including switch construct	It can be used only in loop Constructs

Programs with Continue

```
#include <stdio.h>
main()
{
int c=0;
while (c<=5)
{
 c++;
 if (c==3)
  continue;
  printf("\t%d", c);
}
}
```

O/p: 1 2 4 5 6

(9) GoTo Statement

- another type of Control Statement supported by C.
- The control is unconditionally transferred to the statement associated with the label specified in the goto statement

Syntax

goto label-name;

→ A statement label is defined in exactly the same way as a variable name, which is a sequence of letters and digits, the first of which must be a letter.

→ The statement label must be followed by a colon (:)

→ The goto statement ends with a semicolon (;)

eg: Find factorial of a number.

```
#include <stdio.h>
main()
{
int n, c;
long int f=1;
printf("Enter the number: ");
scanf("%d", &n);
If (n <0)
 goto end;
for(c=1; c<=n ; c++)
   f = f*c;
printf ("\n factorial is %d ", f);
end:
return 0;
}
```

→ Goto Statement is not considered a good programming statement when overused.

→ Because the goto statement can interfere with normal sequence of processing, it makes a program more difficult to read & maintain

---

Nested loops:

→ A nested loop refers to a loop that is contained within another loop.

→ In nested loop, the inside loop executes completely before the outside loop's next iteration.

→ Each innerloop should be enclosed completely in the outloop, overlapping loops are not allowed.

eg: print the pattern:

```
*
* *
* * *
* * * *
```

explanation of pblm: Here an outerloop is required to keep track of the number of rows to be printed. and in each iteration of the outerloop, an inner loop is required to keep track of the printing of stars that corresponds to the row.

eg: Program

```
#include<stdio.h>
main()
{
 int row,col;
```

```
for (row=1 ; row <= 4; ++row)
{
 for (col=1 ; col <= row ; ++col)
    {
       printf ("* \t");
       printf ("\n");
    }
}
```

eg: print the pattern

```
1
2 2
3 3 3
4 4 4 4
```

```
#include <stdio.h>
main ()
{
int row, col ;
for (row=1; row<=4 ; row++)
 {
 for (col=1 ; col<=row ; col++)
  {
   printf ("%d \t", row);
   printf ("\n");
  }
}
```

eg. print the pattern.

```
1
2 3
4 5 6
7 8 9 10
```

```
#include <stdio.h>
main ()
{
 int row, col ;k=1;
for (row=1; row <=4; row++)
 {
  for(col=1 ; col <=row; col++)
    {
     printf ("%d \t", k++);
     printf ("\n");
    }
 }
}
```

WWW.KTUNOTES.IN

Arrays

KTU 2016 Array is a derived data type.

Defn. → Array is a collection of individual data elements that are ordered, fixed in size and homogeneous.
Homogeneous – all elements have to be of same type.

→ Array enables the storing and manipulation of potentially huge quantities of data

- In C, each array has two fundamental properties.
  - → the datatype and
  - → the size.
- Individual array elements are identified by an integer index. In C, the index begins at zero and is always written inside square brackets.
- There are several forms of an array used in C:
  - → one dimensional array
  - → multidimensional array.

One-dimensional array:

There will be a single subscript or index whose value refers to the individual array elements which ranges from 0 to (n-1), where n is the total number of elements in the array.

Declaration of one-dimensional array:

When defining an array in a program, three things need to be specified:

(1) The type of data it can hold. i.e int, char, float, double

(2) The number of values it can hold i.e the maximum no of elements it can hold.

(3) A name.

→ One-dimensional array declaration is a data type followed by an identifier with a bracketed constant integral expression. The value of the expression, which must be positive, is the size of the array. It specifies the number of elements in the array.

The array subscripts can vary from 0 to (size-1)
The lower bound of array subscripts is 0 and the upper bound is (size-1).

KTU 2016 Q

Syntax

datatype array-name [size];

eg: int a [10];

This reserves space for 10 integers.

int a [100];

a can hold 100 integers.

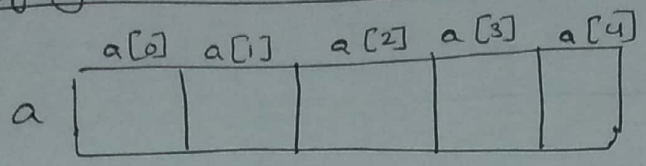
→ once declared, an array element can be referenced as:

<array-name> [<index>]

where index is an integer constant or variable ranging from 0 to <size> - 1.

→ A particular element of the array can be accessed by its index, a number that specifies which element is needed.

Identifying individual array elements



→ Each individual array element is called an indexed variable or a subscripted variable.

→ Both a variable name & an index or a subscript value must be used to reference the element.

→ Index or subscript value gives the position of element in the array.

→ At the time of declaration, the size of the array must be given; it is mandatory.

Initializing Arrays

KTU 2016

→ either one by one or using a single statement.

⇒
```
a[0] = 1;
a[1] = 2;
a[2] = 3;
```
or

⇒ `int a[3] = {1, 2, 3};`

Syntax

`type array-name [size] = {list of values};`

- The number of values b/w braces {} cannot be larger than the number of elements that we declare for the array between square brackets.

- If you omit the size of the array, an array just big enough to hold the initialization is created.

Automatic sizing eg: `double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};`

Similar to

`double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};`

Q. Program to read a one dimensional array and print it.

```
#include <stdio.h>
main()
{
int a[10], i, n;
printf("Enter the size of array:");
scanf("%d", &n);
printf("Enter the array elements:");
for(i=0; i<n; i++)
scanf("%d", &a[i]);
printf("The array elements are:");
for(i=0; i<n; i++)
printf("%d", a[i]);
}
```

Q. Program to read an array and print it in reverse order.

```
#include <stdio.h>
main()
{
int a[10], i, n.
printf("Enter the size of the array:");
scanf("%d", &n);
Printf("Enter the array elements:");
for(i=0; i<n; i++)
scanf("%d", &a[i]);
```

printf("The array in reversed order is:");
for(i=n-1; i>0; i--);
printf("%d", a[i]);

}

printf("The Array in reverse order is:");
for(i=n-1; i>=0; i--)
printf("%d\t", a[i]);

}

Q. write a program to read an array and print sum of the array elements.

```
#include <stdio.h>
main()
{
int a[10], i, s=0, n;
printf("Enter the size of the array:");
scanf("%d", &n);
printf("Enter the array elements:");
for(i=0; i<n; i++)
scanf("%d", &a[i]);
// finding the sum of array elements
for(i=0; i<n; i++)
{
  s=s+a[i];
  }
printf("The sum of array elements is: %d", s);
}
```

Linear Search

The idea is to simply search the array, element by element, from the beginning until the key is found or the end of the list is reached. If found the corresponding position in the array is printed; otherwise a message will have to be displayed that the key is not found.

Program:

```
#include <stdio.h>
main()
{
int a[30], n, i, key, f=0;
printf("Enter the size of the array:");
scanf("%d", &n);
printf("Enter the array elements:");
for(i=0; i<n; i++)
    scanf("%d", &a[i]);
printf("Enter the element to be searched:");
scanf("%d", &key);
//Searching operation.
for(i=0; i<n; i++)
{
if(a[i]==key)
 {
  printf("Element found at %d", i);
   f=1;
 }
```

```
if(f==0)
printf("Not found");
}.
```

Output

Enter the size of the array: 4
Enter the array elements: 5 4 6 7
Enter the element to be searched: 6

element found at 2.

Binary Search:

Binary search is an efficient algorithm for finding an element from an ordered list of items

In binary search,

1. Compare the key with middle element
2. If key matches with middle element, we return the mid index.
3. else if key is greater than the middle element, then key can only be in right half subarray after the middle element. so we search in right half.
4. else (x is smaller) search in the left half.

Algorithm: The algorithm determines the position of T in the List

1. Start
2. print "Enter the no of elements in the array"
3. Input N
4. I = 0
5. print "Enter array element in ascending order"
6. Input List(I)
7. I = I + 1
8. If I < N goto step 5
9. print "Enter the element to search".
10. Input T
11. High = N - 1
12. Low = 0
13. Found = 0
14. Mid = (High + Low) / 2
15. If T = List[Mid]
    Found = 1
    Else if T < List[mid]
    High = Mid - 1
    Else
    Low = mid + 1
16. If (Found = 0) & (High >= Low) then goto step 14.
17. If found = 0. Then print "Not Found".
18. Else print "Found at", mid.
19. Stop

Program

```
#include <stdio.h>
main()
{
int a[30], n, i, t, low, mid, high, found=0;
printf("\n Enter the no of elements in the array");
scanf("%d",&n);
printf("\n Enter the elements of the array:");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
printf("\n Enter the element to search");
scanf("%d",&t);
low=0;
high=n-1;
while(high>=low)
{
mid=(low+high)/2;
if(a[mid]==t)
{
found=1;
break;
}
else if(t<a[mid])
high=mid-1;
else
low=mid+1;
}
```

```
if (found==0)
printf ("\n Not found");
else
  printf ("\n found at %d", mid);
}.
```

output

Enter the ~~elem~~ no of elements in the array: 4

Enter the elements of the array:

5

7

8

9

Enter the element to search: 7

found at 1.

Sorting:

Bubble sort:- A bubble sort compares adjacent array elements and exchanges their values if they are out of order. In this the smaller values bubble to the top of the array while the larger values sink to the bottom of the array. This sort continues until no exchanges are performed in a pass. If no exchanges are made, then all pairs must be in order.

program:

```
#include <stdio.h>
main ()
{
int a[30], n, i, j, temp = sorted = 0;
printf ("Enter the size of array:");
scanf ("%d", &n);
printf ("Enter the array elements ");
for (i=0; i<n; i++)
scanf ("%d", &a[i]);

for (i=0; i<n-1 && sorted==0; i++)
{
  sorted = 1;
  for (j=0; j<(n-i)-1; j++)
   if (a[j] > a[j+1])
    {
     temp = a[j];
     a[j] = a[j+1];
     a[j+1] = temp;
     sorted = 0;
    }
}.
printf ("\n The nombers in sorted order \n");
for (i=0; i<n; i++)
printf ("\n %d ", a[i]);
}.
```

Output

Enter the size of array: 6

Enter the array elements: 42 60 26 55 34 28

The numbers in sorted order.

26
28
34
42
55
60

Multidimensional arrays :-

Array with more than one dimension are called multidimensional arrays.

Two dimensional array & Declaration

An array of two dimensions can be declared as follows:

Syntax: datatype Arrayname[size 1][size 2];

where datatype is the type of element that will be stored in the array. size 1 and size 2 are the sizes of array's first and second dimensions, respectively. Arrayname specifies the name of the array that follows rules as a valid identifier.

eg: int arr[8][8];

Initialization of two dimensional array

Method 1

Syntax:

datatype arrayname[size 1][size 2] = {list of values};

int K[2][3] = {11, 22, 33, 44, 55, 66}

11	22	33
44	55	66

Method 2

Syntax:

```
datatype Arrayname[size1][size2]= { {Row 1 values},
                                    {Row 2 values},
                                  };
```

eg: int k[3][3] = {{1,2,3},{4,5,6},{2,4,5}};

Method 3

While initializing list of values size1 may be omitted In such cases compiler allocates sufficient memory for all initialized elements.

int k[][3] = {{1,2,3},{4,5,6},{2,4,5}};

Method 4

At the time of initializing, even one element is initialized by default remaining elements are initialized with '0' by the compiler.

int k[2][3] = {1,2,3,4};

Multidimensional arrays

It uses three or more dimensions

Declaration

Datatype arrayname [size1][size2]...[sizen];

eg: int k[2][3][4];

above eg is a three dimensional array.

Initialization

datatype arrayname [size1][size2]...[sizen] = {list of values};

eg: int k[2][3][2] = {11,22,33,44,55,66,77,88,99,10,11,12};

Matrix:

The most important application of the 2D array is with a matrix.

Q. program to read a 2D array & print it

```
#include<stdio.h>
main()
{
int a[10][10], i, r, c, j;
printf("Enter the no of rows:");
scanf("%d\n", &r);
printf("Enter the no of columns:");
scanf("%d\n", &c);
printf("Enter the array elements:");
for(i=0; i<r; i++)
 for(j=0; j<c; j++)
   scanf("%d", &a[i][j]);
printf("The array elements are:");
for(i=0; i<r; i++)
{
  printf("\n");
  for(j=0; j<c; j++)
    {printf("%d", a[i][j]);
    }
}
```

Output:

Enter the no of rows: 2
Enter the no of colums: 2
Enter the array elements: 4 5 3 2
The array elements are: 4 5
3 2

Transpose of a matrix

The transpose of a matrix is found by exchanging rows for columns i.e., for.

KTU 2016 Q

$$\text{Matrix } A = (a_{ij})$$

the transpose of A is $A^T = (a_{ji})$, where i is the row number and j is the column number.

Program

```
#include <stdio.h>
main ()
{
int r, c;
int i, j, mat[10][10], transp[10][10];
printf ("Enter the no of rows: ");
scanf ("%d", &r);
printf ("Enter the no of columns: ");
scanf ("%d", &c);
printf ("Enter the elements of matrix: ");
for (i=0; i<r; i++)
for (j=0; j<c; j++)
  scanf ("%d", &mat[i][j]);

for (i=0; i<r; i++)
for (j=0; j<c; j++)
   transp[j][i] = mat[i][j];
printf ("The transpose of the matrix is: ");
for (i=0; i<c; i++)
  {
   for (j=0; j<r; j++)
```

```
    {
      printf ("%d", transp[i][j]);
    }
    printf ("\n");
  }
}
```

Matrix Addition.

```
#include <stdio.h>
main ()
{
int r1, c1, r2, c2, i, j;
int a[10][10], b[10][10], c[10][10];
printf ("Enter the no of rows & columns of matrix 1:");
scanf ("%d %d", &r1, &c1);
printf ("Enter the elements of matrix 1:");
 for (i=0; i<r1; i++)
 for (j=0; j<c1; j++)
  scanf ("%d", &a[i][j]);
printf ("Enter the no of rows & columns of matrix 2:");
scanf ("%d %d", &r2, &c2);
printf ("Enter the elements of matrix 2:");
  for(i=0; i<r2; i++)
   for(j=0; j<c2; j++)
     scanf ("%d", &b[i][j]);
```

S5 MEB.

```
if((r1==r2) && (c1==c2))
{
printf("The sum is:");
for(i=0;i<r1;i++)
 for(j=0;j<c1;j++)
    c[i][i]=a[i][j]+b[i][j];

for(i=0;i<r1;i++)
 {
  for(j=0;j<c1;j++)
   {
    printf("%d",c[i][j]);
   }
  printf("\n");
 }
}
else
 printf("\nAddition is not possible");
}
```

Matrix subtraction can be implemented in a similar way

Matrix multiplication

```
#include <stdio.h>
main()
{
int r1,c1,r2,c2,i,j,k,a[10][10],b[10][10],c[10][10];
printf("Enter the no of rows & columns of matrix 1 :");
scanf("%d%d\n",&r1,&c1);
printf("Enter the elements of matrix 1:");
for(i=0;i<r1;i++)
for(j=0;j<c1;j++)
  scanf("%d",&a[i][j]);

printf("Enter the no of rows & columns of matrix 2:");
scanf("%d%d\n",&r2,&c2);
printf("Enter the elements of matrix 2:");
for(i=0;i<r2;i++)
 for(j=0;j<c2;j++)
   scanf("%d",&b[i][j]);

if(c1==r2)
{
 for(i=0;i<r1;i++)
 for(j=0;j<c2;j++)
  {
     c[i][j]=0;
     for(k=0;k<c1;k++)
        c[i][j]=c[i][j]+a[i][k]*b[k][j];
  }
```

```
printf ("The product is:");
for (i=0 ; i<r1 ; i++)
{
for (j=0; j<c2 ; j++)
  {
   printf ("%d", c[i][j]);
  }
  printf ("\n");
 }
}
else
  printf ("Multiplication is not possible");
}.
```

Strings

A string is defined as "an array of characters".

→ Each string is terminated by the null character ('\0') which indicates the end of the string.

→ %s format specification is used to read & write a string.

→ The ampersand & symbol is not required before the variable name in scanf() function while reading strings.

→ Main problem with scanf() function is that it terminates its input on the first white space it found.

Declaration of a string:

```
Char stringname[size];
```

eg `Char str[10];`

String Initialization:

```
Char str[9] = "I like C";
```

which is the same as:

```
Char str[9] = {'I', ' ', 'l', 'i', 'k', 'e', ' ', 'C', '\0'};
```

also

```
Char msg[] = "Hello";
```

Q. program to read and print a string.

```
#include<stdio.h>
main()
{
char name[10];
printf("Enter the name: ");
scanf("%s", name);
printf("Name is: %s", name);
}
```

Output

Enter the name: Diana

Name is : Diana.

gets() and puts() functions:

- Are library functions used ~~for~~ to read & write an entire line of text including blank spaces.

Syntax

```
gets(string);
puts(string);
```

These functions are defined in "stdio.h" header file.

Q. program

```
#include <stdio.h>
main()
{ char str[50];
printf("Enter a line of text:");
gets(str);
printf("Entered text is:");
puts(str);
}
```

WWW.KTUNOTES.IN

Structures

Syllabus - declaration, definition & Initialization of Structures.

- **Structure** :- is a collection of related variables under one name. These variables can be of different types and each has a name which is used to select it from the structure. Structures are user defined or derived data types.

- **Structure declaration or Defining a structure**

A structure is declared by using the keyword Struct followed by an optional structure tag followed by the body of the structure. The variables or members of the structure are declared within the body.


```
Struct Structure_tag_name.
{
  datatype member_name1;
  datatype member_name2;
      :
} Structure_Variable1, Structure_Variable2, ... ;
```

Where the Structure_tag_name is the name of the structure, the Structure_Variables are the list of variable names separated by commas.

- Each of these structure_variable names is a structure of type structure_tag_name.
- The structure_variable is also known as an instance variable of the structure.
- Each member_name declared within the braces is called a member or structure element.

eg:
```
struct date
{
int day;
int month;
int year;
} order_date;
```

OR

```
struct date
{
int day;
int month;
int year;
};

struct date order_date;
```

// If one includes the structure with a semicolon after the closing brace, no variable is defined. Then the structure variable can be defined by specifying the following statement

```
struct tag_name variable1, variable2;
```

```
Struct book
{
 Char title [50];
 Char author [50];
 Char subject [50];
 int book_id;
} book;
```

Structure without or tag name

```
Struct
{
  Char title [50];
  Char author [50];
  Char subject [50];
  int book_id;
} book;
```


The proper place for structure declarations is in the global area of the program before main().

Multiple Variable declaration

```
eg: Struct personal_date
{
 Char name [100];
 Char address [200];
 int year_of_birth;
 int month_of_birth;
 int day_of_birth;
};
Struct personal_date manish, Venkat, naresh;
```

Different structures can have members with the same name, but the values of members of different structures are independent of one another.

Accessing the member of a structure

A structure member variable is generally accessed using a '.' operator.

Syntax:

```
structure_variable.member_name;
```

The dot (.) operator selects a particular member from a structure.


```
struct mystruct
{
  int a;
  int b;
  int c;
} s1, s2;
```

The first member can be accessed by the construct.

```
s1.a;
```

Structure initialization

Assigning constants to the members of the structure is called initializing of structure.

```
struct struct_name
{
 datatype member_name 1;
 datatype member_name 2;
} struct_variable = {constant 1, constant 2};
```

eg:

```
struct book
{
char name[50];
char author[50];
int price;
} b = {"Harry potter", "J.K. Rowling", 450};
```

or

```
struct book
{
char name[50];
char author[50];
int price;
} b;
```

```
b.name = "Harrypotter";
b.author = "J.K. Rowling";
b.price = 450;
```

To input values for data members of the structure variable.

for integer data members

```
scanf ("%d", &variable-name.member name);
```

for char variables

```
scanf ("%s", &variable-name.membername);
```

It must be noted that within the structure construct, no member is permitted to be initialized individually

eg:

Wrong declaration

```
struct games_ticket
{
  int value = 500;
  int seat_no = 52;
  int date, month, year;
} fan1;
```

is not allowed. This is invalid.

Correct declaration

```
struct games_ticket
{
  int value;
  int seat_no;
  int date, month, year;
} fan1 = {500, 52};
```

Here the members value and seat_num are initialized with the values 500 & 52 respectively.

Partial initialization feature is supported in C. The first few members are initialized and the remaining

uninitialized members are assigned default values (0).

Program:

Store information (name, roll no & marks) of a student and display it.

```
#include <stdio.h>
struct student
{
 char name[50];
 int roll_no;
 float marks;
}s;
main()
{
printf("Enter information:\n");
printf("Enter name:");
scanf("%s", s.name);
printf("Enter roll number:");
scanf("%d\n", &s.roll_no);
printf("Enter marks:");
scanf("%f\n", &s.marks);
printf("Displaying Information:\n");
printf("Name:");
puts(s.name);
printf("Roll:No: %d\n", s.roll);
printf("marks: %f\n", s.marks);
}
```

Nesting of Structures

A structure can be placed within another structure. In other words, structures can contain other structures as members.

→ A structure within a structure means nesting of structures.

→ In such cases, the dot operator in conjunction with the structure variables are used to access the members of the innermost as well as the outermost structures.

Syntax:

```
struct structure1
{ int a;
  --
};
struct structure2
{
  - - -
  - - -
  struct structure1 obj;
};
```

Accessing members from Inner struct

```
Outerstructure Variable.
Innerstructure Variable.
Inner member;
```

eg: S.obj.a;

eg:

```
#include<stdio.h>
struct Address
{
  char HouseNo[25];
  char city[25];
  char pincode[25];
};
```

```
Struct Employee
{
  int id;
  char name[25];
  float salary;
  Struct Address Add;
} E;

main()
{
  int i;
printf("Enter Employee id:");
 scanf("%d", &E.id);
printf("Enter Employee name:");
 scanf("%s", E.name);
printf("Enter Employee salary:");
 scanf("%f", &E.salary);
printf("Enter Employee house no:");
scanf("%s", E.Add.HouseNo);
printf("Enter Employee city:");
scanf("%s", E.Add.city);
printf("Enter Employee pincode:");
 scanf("%s", E.Add.pincode);
printf("Details of employees:");
printf("\n Employee id: %d", E.id);
printf("\n Employee name: %s", E.name);
printf("\n Employee salary: %f", E.salary);
printf("\n Employee House No: %s", E.Add.city);
 printf("\n Employee city: %s", E.Add.city);
```

```
printf ("Employee pincode :%s", E.Add.pincode);
}.
```

Output

```
Enter Employee id : 101
Enter Employee Name: Suresh
Enter Employee salary: 45000
Enter Employee HouseNo: 4598/D
Enter Employee city : Delhi
Enter Employee pin code : 110056.
```

Details of Employee

```
Employee id : 101
Employee Name: Suresh
Employee salary : 45000
Employee House No: 4598/D
Employee City : Delhi
Employee pincode: 110056.
```

Arrays of Structures

It means that the structure variable would be an array of objects, each of which contains the member elements declared within the structure construct.

Declaration of an array structure

```
struct structure_tag_name
{
  datatype member_name1;
  datatype member_name2;
  :
} structure_variable[index];
```

or

```
struct structure_tag_name structure_variable[index];
```

eg:

```
struct student
{
  char name[50];
  int rollno;
  float marks;
} s[10];
```

or

```
struct student
{
  char name[50];
  int rollno;
  float marks;
};
struct student s[10];
```

write a progeam to read a list of students information and print them in Ascending order of their average marks using array of structures.

```
#include <stdio.h>
struct student
{
  char name[50], branch[5];
  int sub1, sub2, sub3;
```

```
    float avg.
    }s[10];
main ()
{
    int i,j,n;
    struct student temp;
    printf ("Enter how many students:");
    scanf ("%d",&n);
    for(i=0;i<n;i++)
    {
    printf("Enter student %d name:", i+1);
    gets (s[i].name);
    printf("Enter student %d branch:", i+1);
    gets (s[i].branch);
    printf ("Enter marks");
    scanf ("%d %d %d", &s[i].sub1, &s[i].sub2,
                        &s[i].sub3);
    }
    for (i=0; i<n; i++)
     s[i].avg =(s[i].sub1+s[i].sub2 +s[i].sub3)/3.0;

for (i=1 ; i<=n; i++)
{
  for (j=i+1 ;j<=n;j++)
    {
     if (s[i].avg> s[j].avg)
       {
          temp= s[i];
          s[i]=s[j];
          s[j]=temp;
       }
```

```
    }
}
printf("Students details As per their average marks are:");
for(i=1; i<=n; i++)
    printf("\n%s\t%s\t%f", s[i].name, s[i].branch,
                           s[i].avg);
}
```

Union :-

A union is a structure, the members of which share the same storage. The amount of storage allocated to a union is sufficient to hold its largest member.
Union provide an efficient way of using the same memory location for multi-purpose.

Declaring a union & its members

A union is declared by using the keyword union

Syntax

```
union union_name
{
datatype var_name;
datatype var_name;
};
```

```
Union union_name variable1, variable2, ...;
```

OR

```
Union union_name
{
datatype var_name;
datatype var_name;
}variable1, variable2, ...;
```

The union tag is optional

All members in union are stored at the same address in memory. Therefore only one member can exist in a union at any instant of time.

Accessing & initializing members of a union

```
union tag-name
{
    member1;
    member2;
    :
    membern;
} Variable1, Variable2, Variable3, ... VariableX;
```

accessing : Variable1.member1;
Variable2.member2; etc.

initialization Variable1.member1 = constant

Difference between structure and union

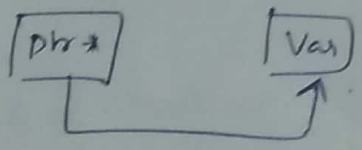
Dec 2016 KTU Q

Structure	union
1. A structure is a user-defined data type available in c that allows to combine data items of different kinds.	A union is a special data type in c that allows storing different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time.

Structure	Union
② keyword struct defines a ~~keyword~~ structure	keyword union defines a union.
③ ~~A Structure~~ Members of structure do not share memory. So a structure need seperate memory space for all its members.	A union shares the memory space among its members so no need to allocate memory to all the members.
④ All members of structure can be initialized	Only the first member of union can be initialized
⑤ Size of the structure is greater than or equal to sum of the each member's size	size of union is equivalent to the size of the member having largest size.
⑥ Struct Struct_name { datatype member1; datatype member2; : datatype membern; } Variable_name;	Union union_name { datatype member1; datatype member2; : datatype membern; } Variable_name;
⑦ In structures individual member can be accessed at a time.	In union only one member can be accessed at a time.

Pointers

defn: A pointer is a variable whose value is the address of another variable i.e direct address of the memory location.



→ for a 16-bit m/c the pointer type is two bytes and for a 32-bit m/c the pointer type is four bytes.

Declaration

The general form of a pointer variable declaration is

```
datatype *var_name;
```

where datatype is the pointer's base type; it must be a valid C data type.

and var_name is the name of the pointer variable and * used to declare a pointer.

eg: int *p;

Here p is a pointer variable ~~of type int~~ that can point to an integer variable.

eg: Char *ptr;

Here ptr is a pointer variable that can points to a character variable.

→ Any type of pointer occupies only 2 bytes of memory. Since pointer holds address of the variable

which is always an unsigned integer.

eg: int * x; x | 2 bytes

int *y; y | 2 bytes

int *z; z | 2 bytes

Initializing pointers;-

Once a pointer variable has been declared it can be made to point to a variable using an assignment operator as:-

Syntax: ptrvariable = &ordinary_variable;

eg: ptr = &x;

where x is an ordinary variable.

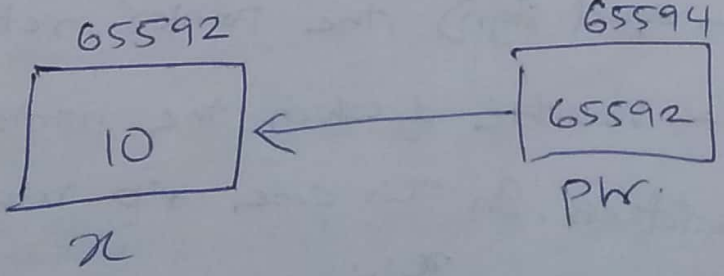
Here address of the variable x is stored in ptr. with this, a link is formed from pointer variable to ordinary variable.

---

eg: int x, *ptr;

x = 10;

ptr = &x;



Accessing Variable Value through pointer

The value of a variable can be accessed through pointer variable using a unary operator '*', usually known as indirection operator. The operator refers to "Value at address".

(*p = Value at the address contain in p)

eg:

```
main()
{
 int x;
 int *ptr;  // pointer declaration
 x=100;
 ptr = &x;  // pointer initialization.
 printf("Value in x: %d", *ptr);  // accessing Variable Value Through pointer
}
```


Output:

Value in x: 100.

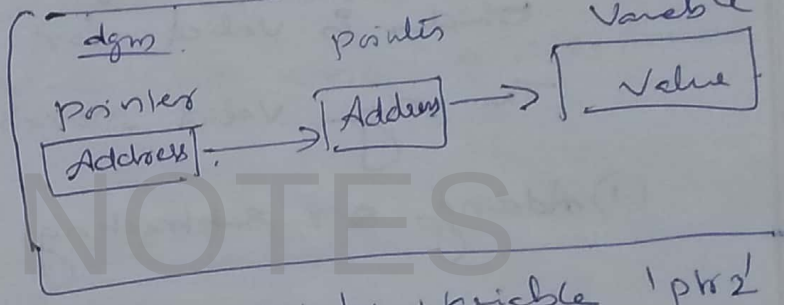
→ When the operator * is placed before a pointer variable in an expression (on the right-hand side of the equal sign), the pointer returns the value of the variable of which the pointer value is the address. In this case *p returns the value of the variable x, because p is the address of x.

Chain of pointers / pointers to pointers

It is possible to make a pointer to point another pointer, thus creating a chain of pointers. A variable that is a pointer to pointer must be declared as: datatype **var; for eg:

```
#include <stdio.h>
main()
{
int x=50;
int *ptr1 =&x;
int **ptr2= &ptr1;
printf ("*ptr1 =%d\n", *ptr1);
printf ("**ptr2 =%d", **ptr2);
}
```

Definition: A pointer is known as a variable containing address of another variable. The pointer variables also have an address. The pointer variable containing address of another pointer variable is called as chain of pointers. This chain can be continued to any extent



Output

*ptr1 = 50

**ptr2 = 50

In the above program, the pointer variable 'ptr2' contains the address of the pointer variable 'ptr1', which points to the location that contains the desired value. This is known as multiple indirections

- A variable that is a pointer to a pointer must be declared using additional indirection operator (*) symbols in front of the name. The declaration int **ptr2 tells the compiler that *ptr2 is a pointer to a pointer of int type.

Pointer Expressions

Like other variables, pointer variables can be used in expressions.

If p is declared as a pointer variable of any type and it has been initialized properly. Then just like a simple variable, any operation can be performed with *p. Because * implies value at address, working with *p means working with the variable whose address is currently held by p. Any expression, whether relational, arithmetic or logical, can be written, which is valid for a simple value variable.

The only valid operations on pointers are as follows:

(1) Adding or subtracting a pointer and an integer.

eg: ptr1 = ptr1 + 2

ptr2 = ptr2 - 2

(2) Subtracting or comparing two pointers (within array limits) that points to the elements of an array.

```
eg: int *p1, *p2, x, y, z;
    p1 = &x;
    p2 = &y;
    z = p1 - p2;
```

```
eg: int *p1, *p2, x, y;
    p1 = &x;
    p2 = &y;
    p1 > p2     /* valid */
    p1 == p2    /* valid */
    p1 != p2    /* valid */
```

3. Assignment of pointers to the same type of pointers.

eg: *a = *b;

4. Incrementing or Decrementing the pointers (within array limits) that point to the elements of an array. When a pointer to an integer is incremented by one, the address is incremented by two.

eg: p1++;
--p2;

- The following arithmetic operations on pointers are not feasible:
  1. Addition of two pointers. eg: ptr1 + ptr2 //invalid.
  2. multiplying a pointer with a no. p1*2 //invalid.
  3. Dividing a pointer with a no. eg: p1/3 //invalid

Pointer increment and scale factor

We can use increment operator to increment the address of the pointer variable so that it points to the next memory location. The value by which the address of the pointer variable will increment is not fixed. It depends upon the data type of the pointer variable. Thus the value by which the address of the pointer variable increments is known as Scale factor. The scale factor is different for different data types

as shown below:

data type	scale factor
char	1 byte
int	2 byte
short int	2 byte
long int	4 byte
float	4 byte
Double	8 byte
long double	10 byte

for eg: int *ptr;

ptr++;

It will increment the address of pointer variable by 2. So if the address of pointer variable is 2000 then after increment it becomes 2002.

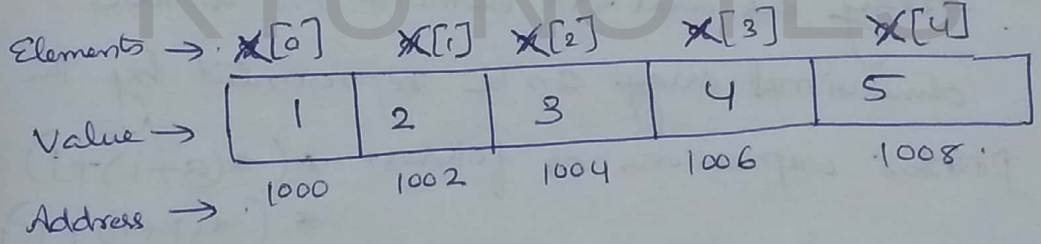
Pointers and Arrays

When an array is declared, the compiler allocates a base address and sufficient amount of storage to contain all the elements of array in contiguous memory locations. The base address is the location of the first element (index 0) of the array. The compiler also defines the array name as a constant pointer to the first element.

Suppose we declare an array x as follows:-

```
int x[5] = {1,2,3,4,5};
```

Suppose the base address of x is 1000 and assuming that each integer requires 2 bytes, the five bytes will be stored as follows:-



The name x is defined as a constant pointer pointing to the first element x[0] and therefore value of x is 1000, the location where x[0] is stored. That is,

$$x = \&x[0] = 1000.$$

The name of the array can be used as a pointer and an integer value can be used to represent an offset from the base address.

program

```
#include <stdio.h>
main()
{
int a[] = {10,20,30,40,50};
int i;
for(i=0; i<5; i++)
printf("%d", a[i]);
}
```

output

10
20
30
40
50

program using pointers

```
#include <stdio.h>
main()
{
int a[] = {10,20,30,40,50};
int i;
for(i=0; i<5; i++)
printf("%d", *(a+i));
}
```

output

10
20
30
40
50

a - base address

*(a+0) = a[0] = 10
*(a+1) = a[1] = 20
*(a+2) = a[2] = 30
*(a+3) = a[3] = 40
*(a+4) = a[4] = 50

two dimensional arrays

pointers can be used to manipulate two-dimensional array as well. An element in a two dimensional array can be represented by the following pointer expression as follows: *(*(a+i)+j) or *(*(p+i)+j)

~~a[i][j]~~

a[i][j] = *(a[i]+j) = (*(a+i))[j]

= *(*(a+i)+j).

program:-

```
#include <stdio.h>
main()
{
int a[2][3] = {10,20,30,40,50,60};
int i,j;
for(i=0; i<2; i++)
```

```
{
printf ("\n");
for(j=0; j<3; j++)
    printf ("%d\t", *(*(a+i)+j));
}
}
```

Output

```
10  20  30
40  50  60.
```

Q. Write a program in C to add two numbers using pointers.

```
#include<stdio.h>
main()
{
int fno, sno, *ptr1, *ptr2, sum;
printf ("Enter the numbers:");
scanf ("%d %d", &fno, &sno);

ptr1 = &fno;
ptr2 = &sno;
sum = *ptr1 + *ptr2;
printf ("sum of the numbers is: %d", sum);
}
```

WWW.KTUNOTES.IN

files

files are used for storing information that can be processed by the programs. Two types of streams - text and binary.

Declaration of a file pointer.

```
FILE *var;
```
{ typeFILE is predefined in stdio.h. }

file pointer — a pointer variable that points to a structure FILE.

Opening files :-

Before performing any type of operation, a file must be opened and for this fopen() function is used.

```
file-pointer = fopen ("FILE NAME", "Mode of open");
```

Here filename is a string literal, which you will use to name your file

eg:

```
FILE *fp = fopen ("text1.c", "r");
```


- access modes can have one of the following values.

mode	meaning
r	— open a text file for reading.
w	— Create a text file for writing
a	— append to a text file.
r+	— open a text file for read/write
w+	— Create a text file for read/write
a+	— Create or append a text file for read/write.
rb	— open a binary file for reading
wb	— open a binary file for writing
ab	— Append to a binary file.
r+b	— open a binary file for read/write
w+b	— Create a binary file for read/write
a+b	— Append a binary file for read/write.

Checking the result of fopen()

The fopen() function returns a FILE *, which is a pointer to structure FILE, that can then be used to access the file. When the file cannot be opened due to some reasons fopen() will return NULL

One may check to see whether fopen() succeeds or fails by writing the following set of statements.

```
fp = fopen ("data.dat", "r");
if (fp == NULL)
{
printf ("Cannot open data.dat\n");
exit (1);
}.
```

Or

```
FILE *fp;
if((fp = fopen ("data.dat", "r")) == NULL)
{
printf ("Cannot open data.dat");
exit (1);
}.
```

Closing the files:

After completing the processing on the file, the file must be closed using the fclose() function.

Syntax

```
int fclose (FILE *fp);
```

fclose (fp);

fclose() returns 0 on success or -1 on error.

All open streams except the standard ones (stdin, stdout, stderr) can also be closed by using the fcloseall( ) function.

Working with text files

C provides four functions that can be used to read text files from the disk. These are:

fscanf( )
fgets( )
fgetc( )
fread( ).

C provides four functions that can be used to write text files into disk. These are:

fprintf( )
fputs( )
fputc( )
fwrite( ).

Writing a character:

Characters are written using putc( ) or its equivalent fputc( )

Syntax

```
putc(ch, fp);
```

```
int putc (int ch, FILE *fp)
```

where ch is the character to be output.
If putc( ) is successful, it returns the character written otherwise it returns EOF.

Eg:

Reading a character

Characters are read using getc() or its equivalent fgetc().

```
ch=getc(fp);
```

```
int getc (File * fp);
```

getc() returns an EOF when the end of file has been reached

The following code reads a text file to the end.

```
do
{
ch= getc (fp);
} while (ch!=EOF);
```

Working with strings - fputs () & fgets ()

String oriented fns. that can handle an entire line as a string at a time.

```
int fputs (const char *str, FILE *fp);
char *fgets (char * str, int length, FILE *fp);
```

length - maximum size of the string.

fread() and fwrite()

- To read & write data types which are longer than one byte.

```
size_t fread (void * ptr, size_t size_of_elements,
    size_t no number of elements, FILE *a_file);
size_t fwrite (const void *ptr, size_t size of elements,
    size_t number of elements, FILE *a_file);
```

fprintf() & fscanf()

```
int fprintf (FILE *fp, const char * format, ...)
int fscanf (FILE *fp, const char * format, ...).
```

```
fscanf (filepointer, "control string", list);
fprintf (filepointer, "control string", list);
```

Q. write a program to read the data from a keyboard and write it into a file.

```
#include <stdio.h>
main()
{
  char ch;
  FILE *fp;
  fp = fopen("demo.txt","w");
  printf("enter Text");
  ch = getchar();
  while (ch != EOF)
  {
    putc(ch, fp);
    ch = getchar();
  }
  fclose(fp);
}
```

Q. write a program to read the data from a file and print it on monitor.

```
#include <stdio.h>
main()
{
Char ch;
FILE *fp;
fp= fopen("demo.txt","r");
if (fp==NULL)
{
printf("In file opening error");
exit();
}
printf("In file contents are");
ch=getc(fp);
while (ch != EOF)
{
 printf("%c", ch);
  ch= getc(fp);
}
fclose(fp);
}
```

Q. Write a program to append the data into a file.

```
#include <stdio.h>
main()
{
char ch;
FILE *fp;
fp = fopen("demo.txt", "a");
printf("\nEnter additional Data: \n");
ch = getchar();
while (ch != EOF)
{
putc(ch, fp);
ch = getchar();
}
fclose(fp);
}
```


Q. Write a program to copy the contents of one file into another file.

```
#include <stdio.h>
main()
{
char ch;
FILE *fp1, *fp2;
fp1 = fopen("demo.txt", "r");
if (fp1 == NULL)
{
printf("\n file opening Error");
exit();
}
```

```
fp2 = fopen ("exam.txt", "w");
while
    ch = getc (fp1);
    while (ch != EOF)
    {
      putc (ch, fp2);
      ch = getc (fp1);
    }
fclose (fp2);
fclose (fp1);
}
```

Q. write a program to merge two files into a third file:

```
#include <stdio.h>
main ()
{
char ch;
FILE *f1, *f2, *f3;
f1 = fopen ("demo1.txt", "r");
f3 = fopen ("result.txt", "w");
if (f1 == NULL)
{
printf ("\n file opening Error");
exit();
}
ch = fgetc (f1);
while (ch != EOF)
{ putc (ch, f3);
ch = fgetc (f1); }
```

```
fcloseall ();
f2 = fopen ("demo2.txt", "r");
f3 = fopen ("result.txt", "a");
if (f2 == NULL)
 {printf ("\n file opening error");
   exit ();
 }
ch = getc (f2);
while (ch != EOF)
{
  putc (ch, f3);
  ch = getc (f2);
}
 fcloseall ();
f3 = fopen ("result.txt", "r");
 if (f3 == NULL)
 {printf ("\n file opening Error");
   exit ();
}
printf ("\n merged contents are :\n");
   ch = fgetc (f3);
   while (ch != EOF)
   {
    printf ("%c", ch);
    ch = fgetc (f3);
   }
    fclose (f3);
 }
```

Q. write a program to read the data from a file and print number of lines in it.

```
#include <stdio.h>
#include <string.h>
main()
{
char ch[50];
FILE *fp;
int lines=0;
fp=fopen("check.txt","r");
if (fp==NULL)
{
printf("file opening error");
  exit();
}.
while ((fgets(ch,sizeof(ch),fp))!==NULL)
  {
    puts(ch);
    lines=lines+1;
  }
printf("\n No of lines:%d",lines);
  fclose(fp);
}.
```

Q. write a program to copy the contents of one file to another file line by line;

```
#include <stdio.h>
#include <string.h>
main()
{
char ch[50];
FILE *fp1,*fp2;
fp1=fopen("check.txt","r");
if (fp1==NULL)
{printf("file opening error");
 exit();
}
fp2=fopen("ptr.txt","w");
while ((fgets(ch,sizeof(ch),fp1)
                !=NULL)
fputs(ch,fp2);
fcloseall();
}
```