Page :
Date: / /

15/1/18

UNIT- I Fundamentals of Algorithms

ALGORITHM is a set of compitional (computable) instructions that receive some value as input which in return gives an output, which can be a single value or multiple values.

OR

It is a set of well-defined statements to solve a problem.

Characteristics:

- It has finite steps.
- It should be correct.
- It should be unambiguous.
- It should be efficient.
- It should have a proper beginning.
- It should eventually terminate with correct output.

Complexity of an algorithm is defined in 2 ways:

(i) Space complexity: utilisation of RAM ; space / memory occupied by a program / algorithm

eg. swapping of 2 variables → by using third variable
→ by not using third variable

(ii) Time complexity: time taken by a program for its execution

Complexity is divided into 3 parts:

- Best case
- Intermediate case
- Worst case

Types of algorithms: Different techniques to solve a problem

Algorithm is language independent.

1) Simple recursive
2) Backtracking
3) Divide and conquer
4) Dynamic programming
5) Greedy
6) Branch and bound
7) Brute force
8) Randomized

Scanned by CamScanner

Date: / /

Data Types: int, float, char
%d %f %c → format specifier

printf - to print the output
scanf - to take input
] - user defined functions

#include <stdio.h> → Pre processor directory
standard input output header file
(contains the declaration of already defined functions)

- main() function is present in all C programs. This is mandatory.

```
main()
{
    block of function
}
```

- ```
#include <stdio.h>
main()
{
    int a, b, c;
    printf("Enter value of a and b");
    scanf("%d", &a);
```
↳ location of variable for storage in memory
```
    scanf("%d", &b);
    c = a+b;
    printf("sum = %d", c);
```
format string
(consists of statement to be printed and format specifier)
```
}
```

- clrscr() and getch() are defined in <conio.h>.

clears the screen

- accepts 1 character as input but does not display it.
- ENTER key need not be pressed.
- Program terminates after inputting that 1 character.
- Holds output screen

Scanned by CamScanner
```

Page :
Date: / /

Every function returns some value.
void main() → won't return any value

Q. Write a program to calculate area of triangle.

Sol^n
```
# include <stdio.h>
main()
{
    int a, b, h;
    printf("Enter values of base and height");
    scanf("%d, %d", &b, &h);
    a = 1/2 * b * h

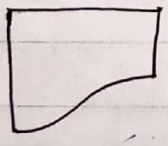
    printf("Area is = %d", a);
    return (0);
}
```

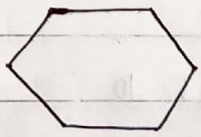
Flowchart

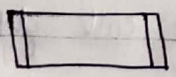
A flowchart is pictorial representation of an algorithm which represents the structure of algorithm, flow of instructions and involved calculations, in computation.

There are 2 types of flowcharts:

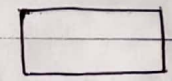
1) Program flowchart
2) System flowchart (for a particular process)

 → secondary storage

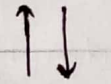
 → looping

 → sub-routines

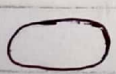
 → connectors

 → computational steps

 → decision-making

 → directional arrows

 → input / output

 → start / stop

Scanned by CamScanner

Page :
Date: / /

C tokens- smallest individual unit

- Operators
- Identifiers
- Keywords
- String constants
- Character constants
- Special symbols

Identifiers are used to create variable and function names.
Maximum length can be 31 characters.
It should only start with alphabets, with no spacing.

Integer constants should not contain coma. Also called numeric constants.
eg. 20,000 ✗ 20000 ✓
1) integer 2) float (real values)

String constants eg. "CAT" (in double quotes)

Variables are of 2 types:
1) local → scope limited to a block
2) global → scope not limited to a single block

eg.
```
int x = 20;
main()
{
    int x = 10;
    printf("%d", x);
}
```
Output : 10

NOTE: In C++, there is functionality which allows to call global variable in the scope of local variable.

There are 32 keywords in C 1989. After 1989, 5 more were added.

Scanned by CamScanner

Page :
Date: / /

A single character within inverted commas is character constant. (single quotes)

Operators are of 3 types:

1) Unary
2) Binary
3) Ternary – conditional operators

DATA TYPES

1) Primary / Basic datatypes - int, char, float
2) Derived datatypes
3) User defined datatypes

(i)

			signed (-ve to +ve)	unsigned (0 to +ve)
int -	short	long		
char -	short	long		
float -	double	long double		

Ranges:

-32768 to 32767 — signed int

0 to 65535 — unsigned int

] 2 bytes for 16-bit OS otherwise 4 byte

-128 to 127 — short signed int

0-255 — short unsigned int

-2147483648 to 2147483647 — long signed int

] 4 byte

char takes 1 byte.

4 byte [3.4e-38 to 3.4e37 — float

1.7e-308 to 1.7e308 — double float

Scanned by CamScanner

CONDITIONAL STATEMENTS

```
if (expression)
{
    Block          # If the expression evaluates to non-zero value then
}                    it is true else it is false.
else
{
    Block
}
```

eg. If we have to calculate $R = \frac{a-b}{c-d}$,

```
if ((c-d) != 0)
{
    R = (a-b)/(c-d)
    R;
}
else                    # executes when (c-d) = 0
{

}
```

Q. Write a program to find the roots of quadratic equation.

Sol^n

```
# include <stdio.h>
# include <conio.h>
# include <math.h>
void main()
{
    float a,b,c,d, root1, root2;
    clrscr();
    printf("Enter the values of a,b,c");
```

Scanned by CamScanner

Date: / /

```
scanf ("%f%f%f" &a,&b,&c);
if (a==0 || b==0 || c==0)
{
    printf ("Error")
}
else
{
    d = (b*b) - (4.0*a*c);
    if (d>0.00)
    {
        printf ("Roots are real and distinct");
        root 1 = -b + sqrt(d) / (2.0*a);
        root 2= -b - sqrt(d)/ (2.0*a);
        printf (" Root 1 = %f " , root 1);
        printf ("Root 2 = %f " , root 2);
    }
    else if (d<0.00)
    {
        printf ("Roots are imaginary");
        root1 = -b/(2.0*a) ;
        root2 = sqrt (abs(d)) / (2.0*a);
        printf ("Root 1 = %f" , root 1);
        printf (" Root 2= %f" , root 2);
    }
    else if (d= 0.00)
    {
        printf (" Roots are real and equal")
        root 1 = -b/(2.0*a);
        root 2 = root 1
        printf (" Root 1 = %f" , root 1);
        printf (" Root 2 = %f" , root 2);
    }
}
getch();
}
```

Scanned by CamScanner

OPERATORS

operators operate on operands.

Unary + → makes a value +ve eg. +5

Unary - → negates a value eg. -5

Binary + → add

Binary - → subtract

Increment operators (unary) → ++, --

% (modulus) → works on integers only

Relational operators → <, >, <=, >=, ==, !=

Assignment operator → =

Logical operator → && (and), || (or), ! (not) (depend on truth table)

Bitwise operator → works on bit level

Order of precendence : ( )
/ * %
+ -

System scans the expression from left to right.

If an expression involves multiple data types, the data type having highest range converts all other data types to its type.

eg. float a, int b;
a = 2.5;
b = 3;
# Result would be float.

eg. float a, int c, int b;
a = 2.5;
b = 3;
c = a+b
# Result would be 5.

Scanned by CamScanner

Page :
Date: / /

```
int a= 5;
int c;
c = ++a;   # first increment, then assign i.e. 6
c = a++;   # first assign, then increment i.e. 5
```

Conditional statements never end with a semi-colon.

Shorthand assignment:

$a+=b \Rightarrow a=a+b$

$a^* = (x+1) \Rightarrow a=a^*(x+1)$

Order of precedence:

```
(), [] , -> , . , ++ , --
+ , - , ! , ~ , ++ , -- , (type) , * , & , sizeof
* , / , %
+ , -
<< , >>
< , <= , > , >=
== , != 
&
^
|
&&
?:
```

- ++2.5 gives 3.500000 (precision value of float is 6)
  2.5-- gives 1.500000 (but first 2.5 is displayed if a command is given to print it and then the variable in which it is stored attains 1.500000 as its value.)

Scanned by CamScanner

- ```
  int a=10;
  printf("%d", a++);
  printf("%d", ++a);
  ```
  Output:
  10
  12

- ```
  int a=5;
  int c;
  c=++a;
  printf("%d", a);
  ```
  ## c = 6
  ## a = 6

- ```
  int a=5;
  int c;
  c=a++;
  printf("%d", a);
  ```
  ## c = 5
  ## a = 6

- ```
  int a;
  if(scanf("%d", &a))
  {
      printf("%d", a);
  }
  ```
  # error if the input is 0
  executes only when we enter a non-zero value

COMPILATION of C program

The components of compiler are:

1. Pre-processor
2. Compiler
3. Assembler
4. Linker

X- Loader (not a component of compiler but OS)

```
#define x=10;
```
→ macros

- Consider a program file sum.c.
- Source code is input to preprocessor.

Scanned by CamScanner

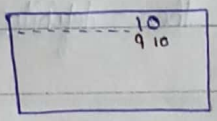
Date: / /

- Compiler converts source code into assembly code; .s file is created.
- Assembly code is an intermediate language; it is neither high level nor low level language. It is machine dependent.
- Assembler converts assembly code into machine code (0 and 1) and an object file is created.
- Linker links all the object files and makes a new object file. This new object file is exe file.

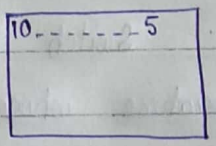
⇒ .c file (executable file) is machine independent.

Formatted input and output functions: printf, scanf
The format for taking input or generating output is specified.

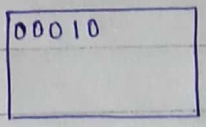
```
int c=10;
printf(" %10d",c);
```




```
int c=10;
int d=5;
printf(" %-10d",c);
printf(" %d",d);
```




```
int c=10;
printf("%10.5d",c);
```




```
float c=5;
printf("%0.3f",c);
```

5.000 (upto 3 decimals)

- getch() takes one value but does not display it.
- getche() takes one value and displays it. No need to press enter.
- getchar() takes one value and displays it. To take the input, we need to press enter.

Scanned by CamScanner

- scanf() has a limited scope as when it encounters space, it assumes input has been taken.

eg. CSE mashing
scanf() would only accept "CSE".
To overcome this, we use gets() and puts().
↓
output

÷ Explicit Conversion and Typecasting:

(int) x ;
c = (float) x / (float) b; → only for this statement, the type has been changed

- Header files contain the declaration of pre-defined functions.
- Library files contain the definition of pre-defined functions.

getch() and getche() are input functions.

Switch Statement

There are some operators whose associativity is from right to left.

switch (variable)
↳ only integer / character constant not float
{
case 1 :
=
break;
=
⇒ if case 1 is true, the commands under case 1 are executed and then control moves out of the block.
}

- Switch statement is used in menu-driven programs.

Scanned by CamScanner

Example:

```
int a,b,c, op ;
scanf ("%d %d", &a, &b);
printf ("1. Sum \n");
printf ("2. Substract \n");
printf ("3. Multiply \n");
printf ("4. Divide \n");
print ("Enter your choice: ");
scanf ("%d", &op);
switch (op)
{
        case 1:
            c= a+b;
            printf ("%d", c);
            break;
        case 2:
            c= a-b;
            printf ("%d", c);
            break;
        case 3:
            c= a* b;
            printf ("%d", c);
            break;
        case 4:
            c= a/b;
            printf ("%d", c);
            break;
        default :
            printf ("Wrong choice");
}
```

Scanned by CamScanner

Loops

- Entry controlled : test condition is in the beginning of the loop
  eg. while loop, for loop

- Exit controlled: test condition is at the end of the loop
  eg. do-while loop
  In do-while, execution of body would take place atleast once and then it checks the test condition.

÷ Structure of a loop:
→ Counter variable
→ Body of loop
→ Test condition
→ Counter variable (increment, decrement)

while loop:

```
counter var = 0;
while (Test condition)
{
        Body of loop
        counter var (inc/dec)
}
```

eg.

```
int = 0;
sum = 0;
while (i < 10)
{
        sum += i;
        i++;
}
printf ("%d", sum);
```

Scanned by CamScanner


```
int i=0;
char ch='y';
sum=0;
while (ch=='y')
{
    sum+=i;
    ch=getchar();
    i++;
}
printf("%d", sum);
```

for loop:

```
for(i=0; i<n; i++)
{
    Body of loop
}
```

OR

```
int i=0;
for(   ; i<n; i++)
{
    i++;
}
```

→ nested loop:

```
for(i=0; i<5; i++)      ⟶ outer loop
{
    for(j=0; j<5; j++)  ⟶ inner loop
    {
        <Body>
    }
}
```

continue → the following statements won't be executed.


```
for(i=0, j=0; x, y; i++, j++)
```

2 test conditions (Both should be true.)

Scanned by CamScanner

Page :
Date: / /

do while loop:

```
int i=0;
do
{
    <Body of loop>
} while (test condition);
```

- Nesting can be done till 15 levels.

- ```
  for(i=1; i<=5; i++){
      for(j=1; j<=i; j++){
          printf("%d", j);
      }
      printf("\n");
  }
  ```

Output:

1
1 2
1 2 3
1 2 3 4
1 2 3 4 5

Storage classes

Whenever a variable is declared, it is stored in either memory or CPU register.

With every variable, there is an associated scope (block of the variable) and lifetime (till the time it is valid (till a programs runs)).

- ```
  int a;
  printf("%d", a);
  ```
  → it would give a junk value

With every declared variable, there is an associated storage class which

Scanned by CamScanner

Page :
Date: / /

defines its memory, initial value, scope and lifetime.

Types of Storage classes:

1. Automatic] - initial value of variable is any garbage value
2. Register]
3. Static] - storage is in memory and initial value is 0 by default.
4. External

- Allocation of variable is in memory in case of automatic storage class.
- Allocation of variable is in CPU register in case of register storage class. It is easy to access a variable from CPU registers but these CPU registers are very limited so sometimes the allocation of variable goes in automatic storage class.

```
void main()
{
    int i=0;
    {
        int i=1;
        {
            int i=2;
            printf("%d", i);        // 2
        }
        printf("%d", i);            // 1
    }
    printf("%d", i);                // 0
}
```

- In C, it is not possible to access a global variable if another variable with same name exists in an inner block.
- auto int i; register int a; static int k;
  ↳ declaration

Scanned by CamScanner

```
void inc()
{
    static int i;
    i = i+1;
    printf("%d", i);
}
void main()
{
    inc();   // 1
    inc();   // 2
    inc();   // 3
}
```

- In static, only once the initial value is 0, after that it retains the value.

```
void inc()
{
    int i=0;
    i = i+1;
    printf("%d", i);
}
void main()
{
    inc();   // 1
    inc();   // 1
    inc();   // 1
}
```

- In usual declaration (rather initialization) of variables, the value is not retained.

- Static variables have more lifetime than usual variables as they retain their value.

- Constant variable is the one whose value is fixed and can't be changed.

```
# c = printf("CSE");
  printf("%d", c);
```

```
# 3
```
↳ no. of characters being displayed

- The default returntype of printf is int.

Scanned by CamScanner

Page :
Date: / /

Arrays

Array is a collection of similar items.
It is linear.

eg. int a[ ] = {1,2,3,4,5} → compile time initialisation

```
int a[10];
int i=0;
for (i=0; i<10; i++){          → run time initialisation
        scanf("%d", &a[i]);
}
```

Q.1) Write a program to count no. of digits.
Q.2) Write a program to check whether a no. is prime or not.
Q.3) Write a program to check whether a no. is palindrome or not.
Q.4) Write a program to check if a no. is Armstrong no. or not. (eg. 153)

Sol.n 1)

```
# include < stdio.h>
int main()
{
        int n, count=0;
        scanf("%d", &n);
        while (n>0){
                n=n/10;
                count+=1;
        }
        printf("%d", count);
        return 0;
}
```

Sol.n 2)

```
# include<stdio.h>
int main()
{
        int n, p=0;
        scanf("%d", &n);
        for (int i=2; i<n; i++){
                if (n%i==0)
                {
                        p+=1;
                        break;
                }
        }
        if (p==0)
        printf("%d is prime", n);
```

Scanned by CamScanner

Page :
Date: / /

Sol.n 3)
```
# include < stdio.h>
int main()
{
    int n, r, s=0;
    scanf ("%d", &n);
    while (n!=0) {
        r = n % 10;
        s = s * 10 + r;
        n = n / 10;
    }
    if (n == s)
    printf ("%d is palindrome", n);
    return 0;
}
```

Sol.n 4)
```
# include < stdio.h>
int main()
{
    int n, r, sum=0;
    scanf ("%d", &n);
    while (n!=0) {
        r = n % 10;
        s += r * r * r;
        n = n / 10;
    }
    if (n == s)
    printf ("%d is Armstrong number", n);
    return 0;
}
```

Scanned by CamScanner

÷ Array:

Address of first index of array is called base address of an array.

Q. WAP to find min. and max. no. in an array, alongwith index value.

Sol^n

```
# include < stdio.h>
int main()
{
        int arr [100];
        int max, min
        for (int i=0 ; i < 100 ; i++){
              scanf ("%d", &arr [i]);
        }
        max = arr [0];
        for (int j= 1 ; j < 100 ; j++){
             if ( arr [j] > max)
             max = arr [j];
        }
        for ( int i=0 ; i< 100 ; i++){
             if ( arr [j] == max)
             printf ("%d %d", arr [j], j);
             break;
        }
        min = arr [1];
        for ( int j=1 ; j< 100 ; j++){
             if (arr [j] < min)
             min = arr [j];
        }
        for ( int i=0 ; i< 100 ; i++){
             if ( arr [i] == min)
             printf ["%d %d ", arr [i] , i );
             break;
        }
return 0;
```

Scanned by CamScanner

Functions

Function is a set of related instructions to solve a problem.
In C, procedure and function mean the same thing.

Execution always starts from main().
Compilation starts from preprocessor.

Declaration is only a prototype.
Definition is the body; it is a complete block.
USES:

- Readability improves.
- We don't have to write the same code.
- Functionality can be changed.

If we are defining a function after main(), then declaration is mandatory.

Declaring a function:
Every function must have a return type. By default, the return type is int.

return-Type function name < list of parameters > ;

While declaration, writing variable names is optional.

Eg. void sum (int a, int b); // declaration
↳ formal parameters

```
void main ()
{
        x, y ;
        sum (x, y);          // function call
          ↳ actual parameters
}
```

Scanned by CamScanner

Page :
Date: / /

```
void sum (int a, int b) {                    // definition
    int c;
    c = a+b;
    printf ("%d", c); }
```

- In C, there is only static linking i.e. linking happens at compile time.

```
# void sum (int , int);
  void main()
  {
      x, y, z;
      sum (x, y, z);                         // ERROR
  }
```

```
# void sum (int , int , int);
  void main()
  {
      x, y;
      sum (x, y);                            // It will take default
  }                                              value for 3rd variable.
```

- We can declare a function within a function but we can't define a function within a function.

POINTERS

A pointer variable is a variable which stores the address of another variable.

```
int * ptr;                    (int) * ptr;
int i;                        (float) i;
ptr = &i;                     ptr = &i;        // ERROR
      ↳ address of operator
```

Scanned by CamScanner

Page :
Date: / /

```
int * ptr;
ptr = &i;
int i;                    // ERROR
```

- Variable declaration must take place before assigning its address value to pointer variable.

```
float * ptr;
float i;
ptr = &i;
printf ("%d", &i);
                  ↳ address of i
printf ("%d", * (&i));
                  ↳ value at address of i
printf ("%d", * ptr);
                  ↳ value at address of i
```

-
```
void sum (int a, int b)
{
    int c = a+b;
    printf ("%d", c);
}
main ()
{
    int a,b;
    sum (a,b);          // Linking of sum (a,b) with above defined function is binding.
}
```

While calling a function, a copy of the actual parameters is generated and based on the copy, output is obtained.

Scanned by CamScanner

```
void swap (int, int);
void main ()
{
        int a,b;
        swap( &a, &b);
}
void swap (int *a, int *b)
{
        int temp;
        temp = *a;
        *a = *b;
        *b = temp;
}
```

Call by reference is actually call by address in C.

```
int *a, i;
a = &i;
scanf ("%d", &i);
printf ("%d", *a);
```

```
int *a;
int i=3;
a= &i;
printf ("%d", *a);
```

Q. WAP to sort an array of 10 elements.

Sol^n

```
# include < stdio.h>
void main()
{
        int i,j,a,n, number [10];
        scanf ("%d", &n);
        for (i=0; i<n ; ++i){ scanf ("%d", & number [i]); }
        for (i=0 ; i<n ;++i){
        for (j=i+1 ; j<n ; ++j){
        if (number [i] > number [j]){
        a= number [i];
        number [i] = number [j];
        number [j] = a;
        } } }
        for (i=0; i<n ; i++) printf ("%d\n", number [i]);
}
```

Scanned by CamScanner

Page :
Date: / /

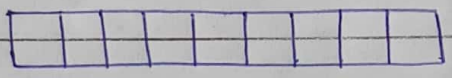
2-d array

row column

int a[0] [0]
a[0] [1]
a[0] [2]
a[1] [0]
⋮

(0,0)
→ ① 2 3
4 5 6
7 8 9

But memory is 1- dimensional.



→ 1-d memory

Storage can be row-wise or column-wise.

÷ Defining an array:

int a[3][3] = {1,2,3,4,5,6,7,8,9};

OR

int a[3][3] = {{1,2,3},{4,5,6},{7,8,9}};

OR

int a[3][3] = {{0},{0},{0}};

By default, values of array elements are garbage values.
All matrix operations can be performed on a 2-d array.

• int a[ ][3];
↳ by default, the no. of rows will be equal to the no. of inputs we are giving.

Scanned by CamScanner

Q. Write a program to find an element in array.

Sol.n
```
# include < stdio.h>
int main()
{
        int b,a,n, arr[100];
        for( int i=0 ; i<n ; i++){
                arr[i] = scanf(" %d", &b);
        }
        for( int i=0; i<n; i++) {
                if (arr[i] == a) printf(" %d %d", arr[i] , i);
        }
        return 0;
}
```

Q. Write a program to calculate factorial of a number using function.

Sol.n
```
# include< stdio.h>
int fac(int);
int main();
{
        int a;
        scanf(" %d", &a);
        int factorial;
        factorial = fac(a);
        return 0;
}
int fac(int a)
{
        int i, fact = 1;
        if(a>0) for( i = a ; i>0 ; i--)
        {
        fact = fact * i; }
        return fact;
}
```

Scanned by CamScanner

Page :
Date: / /

Calling a function within a function is called nesting of function. Calling a function within a function call is called nesting of function calls.

eg. add (add (a,b) , c) ;

Recursive Functions

When a function calls itself, we call it a recursive function.

eg.
```
int fact (int a)
{
    int f;
    if (a==1)  return 1;
    else {
    f= a* fact (a-1);
    return f; }
}
```

We use recursive functions where we want to solve the problem recursively. Sometimes, recursive functions may be infinite.

When we write void in formal parameters, it means we are not giving any parameter to the function.

•
```
int *ptr;
int a , b= 25 ;
ptr = &a;
*ptr = 10;                    // dereferencing
printf ("%d" , ptr);          // address of a
printf ("%d" , *ptr);         // 10
ptr = &b;
*ptr+ = 10;
printf ("%d", ptr);           // address of b
printf ("%d", *ptr);          //35
```

Scanned by CamScanner

- * ptr means deferencing in which we are working with the value at the specific address.

Structure

↓

User defined datatype

A structure stores heterogenous data items, which should be logically related.

- We can pass an array as a parameter to a function either as a whole or element by element.


```
void display (int a)
{
    printf ("%d",a);
}
void main()
{
    int a[5] = {1,2,3,4,5};
    int i;
    for(i=0; i<5; i++)
    display(a[i]);
}
```


```
void display (int a[], int n)
{
    int i;
    for(i=0; i<n; i++)
    printf ("%d", a[i]);
}
void main()
{   int n;
    scanf("%d", &n);
    int a[n];
    display(a, n);
}
```

Scanned by CamScanner

ptr = ptr + 10;

if ptr = 2000 then ptr + 10 = 2000 + (10 * 2) = 2020

↓ size of integer

-
```
void main()
{
    int a[5] = {1,2,3,4,5}, int i;
    for (i=0; i<5; i++) {
        printf ("%d", *(a+i));
    }
```

↳ base address of array

all elements would be printed

Nesting of functions:

```
int sum (int a, int b){
    c = a+b;
    return c;
}
int mult (int a, int b, int c)
{
    int d;
    d = c* sum(a,b);
    return d;
}
void main()
{
    int a, b, c, d;
    scanf ("%d %d %d", &a, &b, &c);
    d = mult (a,b,c);
    printf ("%d", d);
}
```

Scanned by CamScanner

Page :
Date: / /

÷ STRUCTURE:

User defined datatype

Grouping of different types of logically related data under a common name is called structure.

eg. struct st] Structure tag
{

DECLARATION:

```
int rollno;
char name[10];
int marks;
float per;
```

Structure members / elements

};

struct st st1, st2, st3;

st1, st2, st3 : variables

- Memory allocation of a structure takes place when we declare a variable for it.

```
struct st st1, st2, st3;
str1.rollno = 20;
strcpy(str1.name = "CSE");
st1.marks = 90;
st1.per = 90;
```

Continuous memory allocation means different datatypes in a structure are stored linearly.

```
struct stud
{
    int R;
    char name[20];
    int mks;
} st1, st2;        // another way of declaring variables
```

Structure members can't be initialised within the structure block.

Scanned by CamScanner

1. st1.R = 10; // initialising elements outside
st2.R = 20; structure block

2. struct stud st1 = {10, "CSE", 20};

3. scanf("%d", &st1.R);

# struct stud st[10]; // array of structures
for(i=0; i<10; i++){
scanf("%d", &st[i].R);
}

Array of structures is homogenous as it is storing elements of same datatype.

Q. Create a structure of employees containing elements employee id, employee name and employee salary. Declare array of structures initialising the elements and display it.

Sol:
```
struct emp
{
        int empid;
        char name[20];
        float salary;
} ;
struct emp arr[2];
for(i=0; i<2; i++){
        scanf("%d", &arr[i].empid);
        scanf("%s", &arr[i].name);
        scanf("%f", &arr[i].salary);
}
```

Scanned by CamScanner

Page :
Date: / /

÷ Working with functions in structures:

(a)
```
struct st
{
        int a;
        int b;
        int c;
};
void print (int a, int b, int c)
{
        printf ("%d", a);
        printf ("%d", b);
        printf ("%d", c);
}
void main()
{
        struct st st1, st2;
        st1.a = 10;
        st1.b = 20;
        st1.c = 30;
        print (st1.a, st1.b, st1.c);
}
```

(b)
```
struct st
{
        int a;
        int b;
        int c;
};
void main()
{
        int *p;
        struct st st1, st2;
        st1.a = 10;
```

Scanned by CamScanner

```
    st1.b=20;
    st1.c=30;
    p=&st.a;
    printf("%d", *p);          -10
    printf("%d", *(++p));      -20
    printf("%d", *(++p));      -30
}
```

(c)
```
struct st
{
    int a;
    int b;
    int c;
};
void main()
{
    int *p;
    struct st st1={10,20,30};
    struct st *st2;                    // Pointer structure
    st2=&st1;
    printf("%d", st2->a);
}
```

-> arrow operator (used in case of pointers)

(d)
```
struct st
{
    int *a;
    int *b;
    float *c;
};
void main()
{
```

Scanned by CamScanner

Page :
Date: / /

```
struct st st1;
int a1 = 20;          // initiasing structure variables
int b1 = 30;          with these values using
float c1 = 40;        pointers
```

Nesting of Structures

It means that a variable of one structure is present in another structure.

```
struct st
{
        int a;
        float b;
};
struct st2
{
        char name[10];
        struct st st1;
} st02;
```

Reference using &st02.st1.a

Scanned by CamScanner

Page :
Date: / /

23/3/18

File Handling

File is a collection of related data stored in secondary memory.
Till now, we were dealing with console (terminals).
But in case of files, we can also store our output in secondary memory.
To deal with files, we need an interface called stream.

- Text Stream (a sequence of characters stored line by line)
- Binary Stream (a sequence of bits and bytes)

(i) Sequential files: Access is sequential. Access time is greater.
(ii) Random access files: Access time is smaller. It can jump to a particular character.

- Every file should have a valid name, followed by extension.
- Every file should have a purpose.
- There must be a data structure for a file.
- FILE → defined in stdio.h
  ↓ a structure (pointer type)

→ fopen (" ", "w") // declared in stdio.h
  ↓ creates a file
  ↓ filename
  ↳ write mode

→ fclose (p) // declared in stdio.h
  ↳ pointer

Scanned by CamScanner

```
#include <stdio.h>
void main()
{
        FILE *p;
        p= fopen ("CSE", "w");
                    ↳ previous contents will be deleted and new text would be written in it.
        fclose(p);
        p= fopen ("CSE","r");
        if (p==NULL)    puts ("file does not exist");
        // NULL is a macro defined in stdio.h.
}
```

- EOF is a macro, which returns TRUE if pointer has reached end of file.

```
while (!feof(p))
{
        ==
} fclose(p);
```

-
```
#include <stdio.h>
void main()
{
        FILE *p;
        char c=' ';
        p= fopen ("CSE","w");
        while (c!='.')
        {
                putc (c,p);
        while (1){
                c= getchar();
                if (c=='.')   break;
                else          putc (c,p);
        }
        fclose(p);
}
```

Scanned by CamScanner

ASCII value of EOF is 25/26.

Page :
Date: / /

- putc(,) or fputc(,) is used to write a single character to a file.
- getc(p) or fgetc(p) is used to read a file character by character.
- puts(,) is used to write a string to a file.
- gets(c, 20, p) is used to read a file line by line.
  - c → declared string
  - 20 → no. of bytes

"a" → append mode

"w+" → both read and write mode

but first we write, then we can read.

After writing, pointer is at last so to read, the pointer should be at beginning. To bring the pointer at beginning, we use rewind(p).

```
#include <stdio.h>
void main()
{
    FILE *p;
    p= fopen("CSE","w");
    char ch= '';
    while(1){
        ch= getchar();
        if(ch!="\n") break;
        else    putc(ch,p);
    }
    fclose(p);
    p= fopen("CSE","r");
    while(!feof(p)){          // feof is a function.
        getc(p);
    }
    fclose(p);
}
```

Scanned by CamScanner

```
#include <stdio.h>
#include <string.h>
void main()
{
        FILE *p;
        p=fopen("CSE","a");
        char ch[20];
        scanf("%s",ch);
        for(int i=0; i<strlen(ch); i++)
        {
                putc(ch[i],p);
        }
        fclose(p);
        p= fopen("CSE","r");
        while(!feof(p)){
                getc(p);
        }
        fclose(p);
}
```

Formatted file I/O

fscanf(p,"%d",&a);
for writing into a file → fprintf
for reading from a file → fscanf

Q. Open a file in write and read mode (w+) and use the datatypes char name; int m_1, m_2; float per.

Sol:

```
#include <stdio.h>
void main()
{
        FILE *p;
        p= fopen("CSE","w+");
        char name[10];
```

Scanned by CamScanner

int m_1, m_2;
float percent;
scanf("%s", name);
scanf("%d %d", &m_1, &m_2);
scanf("%f", &percent);
puts(name, p);
~~put~~

- We create a structure and the above datatypes are its members.

```
fprintf(p, "%s %d %d %f", name, m1, m2, percent);
rewind(p);
fscanf(p, "%s %d %d %f", name, &m1, &m2, &percent);
```

- fread() and fwrite() functions are only used for binary files.

- fwrite(&stud, sizeof(s), 2, p);
  - &stud → structure name
  - sizeof(s) → array of structures
  - 2 → no. of records

STRINGS

A string is an array of characters.

char a[10]
↳ 9 characters followed by a null character ('\0')

char a[10] = "CSE"
char a[10] = {'C', 'S', 'E', '\0'}

- gets(a) → input
- puts(a) → output

Scanned by CamScanner

strlen - returns the no. of characters in string
d = strlen(a);
It also includes '\0'.

strcpy - initialising a string with an already initialised string
char text1[5] = "CSE";
char text2[5];
strcpy(text2, text1);

strcat - concatenation of strings
strcat(text2, text1);

strncat - concatenates specific no. of characters to another string
strncat(text2, text1, n)
↳ no. of characters of text1 to be concatenated with text2

strncpy - copies / overwrites specific no. of characters to another string
strncpy("CSE", "EEE", 1)
Output is ESE

strcmp - checks if strings are identical or not (not case sensitive)
strcmp("CSE", "cse"); → returns 0

* strcmp("CSE", "cse"); → case sensitive so returns false

strrev - reverses a string
strrev("CSE"); → output is ESC

strdup - uses pointers

Scanned by CamScanner

Date: / /

- int d;
  char a[5] = "1981"
  d = atoi(a); // converts "1981" into numeric value

Scanned by CamScanner