13/09/2021

INTRODUCTION TO C-PROGRAMMING

W5H

INPUT-STORAGE-PROCESS-OUTPUT

DATA

Raw form of information
Unorganized information
Unprocessed information
Collection of facts,figures or items

Information

Organized data or processxed data

Unit-I

Computer

An electronic device
Features of Computer

Speed
Accuracy
Diligence
Versatality
Storage Capacity

Input-----> CPU(Memory Unit,ALU,CU)---->Output---Monitor(CPU)
Keyboard

Types of Computers

Micro
Mini
MainFrames
Super Computer
Laptop
PC
Server Computer
Workstation Computer

```
        Computer Architecture
          /          \
     Software        Hardware
```

```
   --System--OS--Operating System
   --Application
         -Languages--structured program----basic,fortran,c,c++,java,...
         -Packages---Ready made program--
```

set of instructions(commands)----program

Operating System

DOS,MS-Windows,Unix(UNICS),Linux,Ubuntu,Android,Apple ios,Apple Mac....
Open source

Open source software
---
python,php scripting language,apache http web server etc.,

15/09/2021

List Types of OS
---
a)Batch OS
b)Distributed OS
c)Multitasking OS
d)Network OS
e)Real-time
f)CLI--Command Line Interface
g)Mobile OS
h)Multi-user
i)Time slicing

Functions of Operating System

Processor Management
Memory Management--Allocation and Deallocation of Memory
Device Management--Controls the working of Input-Output devices
File Management---Creation,deletion,copy,transfet etc.,Directory structure
Security--unauthorized access,Firewall active,memory,messages regarding the system vulnarabilities
Job Scheduling--Time allocated for each application
Error Detection--- external threat..alerts

Components of OS

Shell---user interactions
Kernel---core component of the OS...interface between applications and the hardware

Peripheral devices

Auxiliary device

Memory Unit

Bytes
1 byte=8 bits
bit=0 or 1

Kilo Bytes,Mega Bytes,Giga Bytes,Tera Bytes.....
Primary Secondary
-RAM Hard disk
-ROM

Types of Languages

Low-level or Machine-level
Middle-level or Assembly-level
High level

Translators

Compilers
Assemblers
Interpreters

System Unit

Motherboard,CPU,RAM etc.,

Algorithm
---
step-by-step procedure

independent of any language

Step 1:start
Step 2:Input
Step 3:Process
Step 4:Output
Step 5:stop

Algorithm to add two numbers
---
a=1,b=2
c=a+b

Step 1:Take two numbers
Step 2:Add numbers using + operator
Step 3:Display the result

17/09/2021

Characteristics of an algorithm
---
a)Clear and unambigious
b)Well-defined inputs
c)Well-defined outputs
d)Finiteness
e)Feasible
f)Language independent

Advantages
---
a)Easy to understand
b)Step-wise representation of a solution(problem)
c)Broken down into small pieces or steps

Disadvantages
---
a)Time consuming
b)Braching and Looping statements

Flow Chart

Pictorial or Diagrammatic
Flowcharting

Symbols

Oval---Start/Stop
Paralellogram---Input/Output
Diamond----Decision Making
Rectangle--Computation/Calculation
circle---connector
Arrows---indicate the flow of control(up,down,left and right)

a)Write algorithms to find the areas of circle,square,rectangle and triangle
Draw the flow char for the same
b)List and explain different applications of computers

INTRODUCTION TO C LANGUAGE

Dennis Ritchie,1972 at AT & T Bell Labs,USA

Simple,reliable and easy to use

ALGOL,BASIC,COBOL,FORTRAN,PASCAL,B,UNICS,C,C++,JAVA,C#,...

English

Alphabets--->words--->sentences--->Paragraph

Syntax and Semantics

C/C++/Java/C#

Character Set--->Variables,keywords,constants,operators,data types,escape sequences.--->Instructions-->Program

Character Set

Alphabets---a-z,A-Z
Digits--0 to 9
Special Characters----!,> < . ? / ' " ; : () {} [] * # etc.,

Variables--place holder--identifiers

Syntax

<data-type> <variable-name>;
int a;//dynamic initialization

int b=10;//static intialization..constant

a--variable name
10--value
346--address of the variable

Rules
---
a)Alphabet

a1//valid

b)Should not be lengthy ---32 characters

c) Meaningful

d)First character should not be a digit
1a //invalid
gross pay//invalid
gross-pay //invalid
gross_pay//valid

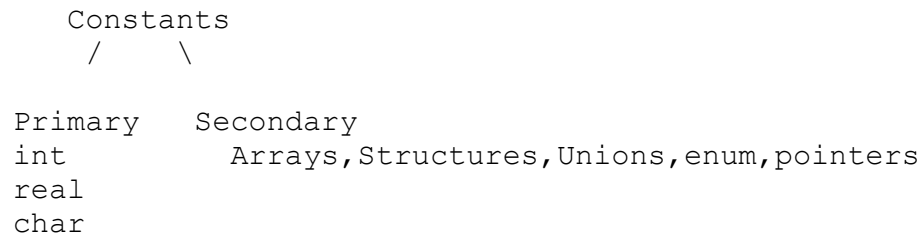
Keywords

Reserved words
Can't be used as variable names---32

if,for,else,int,break,continue,char,float,short ,long,for,while,do.....

Constants--literals

<data-type><variable-name>=value or expression
int a=10;



Integers
----
a)Zero,Positive or Negative
b)Digit
c)No decimal point
d)No sign is preceded before the digit..positive
e)No commas and blank spaces should be used

<data-type> <variable-name>=value;

int a=10;

b)Character Constants

Single alphabet,digit or a special character
Enclosed in single quotes ' '

'a','5','&'

c)Real Constants

Fractional

Decimal point

Positive or Negative

Exponent

3*10^2
3e+2

-3*10^-5

-3e-5

'e'
'E'

20/09/2021

BASIC STRUCTURE OF C PROGRAM

Data Types

Data storage format that a variable can store a data to perform specific operation

<data-type> <variable-name>;
int a;//dynamic initialization

int b=10;//static intialization..constant

Size of the variable,constant and array are determined by the data types

Data Types in C

Basic or Primary or Built-in------int,char,float,double

Supports unsigned and signed literals

Memory Size---32 bit to 64 bit

32-bit
------
signed char ---   1 byte--- -128 to 127
unsigned char--   1 byte   0 to 255
short---   2 bytes-- -32768 to 32767
signed short -   2 bytes -- "
unsigned short - 2 bytes --- 0 to 65535
int --   2 bytes ---- -32768 to 32767
signed int   -2 bytes ---- -32768 to 32767
unsigned int --- 2 bytes --- 0 tp 65535

long int   4 bytes
signed long   4 bytes
unsigned long   4 bytes
float   4 bytes
double   8 bytes
long double   10 bytes

BASIC STRUCTURE OF C PROGRAM

Built-in functions

Document section
Comments

//---Single Line
/*Text*/---Multi-Line

/** Document */

Escape Sequences

\n---new line
\t---horizontal tab
\v--vertical tab
\b--back space
\a--audible sound
\r--carriage return
\f---form feed
\\---back slash
\'--single quote
\"---double quote

```
     # include <stdlib.h>
       #include <stdio.h>
#include "file1.c"
        conio.h
        ctype.h
        string.h
        math.h
```

In-built library functions "stdio.h"
printf()
--ar---
Output
Syntax
---
printf("format-String",argument-list);

format String--Escape Sequence Characters,Text and formatting character

%d---int
%c--character
%f--float
%l---long
%s--string
%ld---long int
%lf---double
scanf()

Document Section
Global Variable declaration
Header file declaration
<return-type> main()
{
statement(s);
fun1();

```
fun2();
}
fun1()
{
}
fun2()
{
}
```

User-defined functions

Derived or Secondary or Referenced--array,pointers,structures,unions

Enumeration---enum

Void--- void (returns nothing)

22/09/2021

Data Types

```
char
short int
unsigned short int
signed short int
int
signed int
unsigned int
float
double
```

Operators

```
Arithmetic------------> + - * / %
Logical---------------> && || !
Relational------------> > < >= <= !=
Bitwise---------------> & | ^
Shift-----------------> << >>
Unary-----------------> + -
Increment/Decrement -----> ++ --
Conditional or Terenary----? ? :..if..else
Assignment            ---> =
Comparision           ---> ==
```

Escape Sequence Characters

```
\n---new line
\t---new tab
\v,\',\",\\,\r,\a,\f
```

Comments

```
//Text--single line comment
/* Text */---Multi-Line
/**Text */--Documentation
```

Home work

---
a)Write a C-program to find the sum,difference,product,quotient and remainder of two numbers(Initialize the variables)
b)Write a C-program to find the area of a rectangle and a square
c)Write a program to initialize two variables and interchange the variables
     a=10,b=20
     a=20,b=10
24/09/2021

Input function
---
scanf()

Syntax
---
scanf("format specifier",&var1,&var2,...&varn);

BODMAS
BEDMAS

ab
—

2

#include<math.h>

Roots of Quadratic Equation

-b+/-sqrt(b^2-4ac)/2a

d=b*b-4*a*c
e=2*a

r1=(-b+sqrt(d))/e
r1=(-b-sqrt(d))/e

Swapping using operators
---
a=1,b=2

a=a+b//a=1+2=3
b=a-b//b=3-2=1
a=a-b//3-1=2

27/0/2021

A-Z--->65-90
a-z--->97-122
0-9--->48-57

Type Conversion or Type Casting

 Converting one data type into another is known as type casting or type conversion

(type_name)expression

long double--->double--->float--->unsigned long---->long--->unsigned int---

>int

Implicit Conversion---Automatic

Explicit Conversion--forcible conversion
():
destination_datatype=(target_datatype)variable;

```
int x=20;
float y;
y=x;
```

Differences between Type casting and Type Conversion

Comaptible and Incompatible	Compatible
Destination type is smaller than the source	Destination type can't be smaller than the source
Narrowing Conversion	Widening
Implict	Explicit
More efficient	Less efficient

29/09/2021

Opertors of the same precedence are evaluated from

Left to Right

L->R

sizeof()

Compile-time Unary Operator used to compute the size of the operand
Returns the size of the variable
When sizeof() is used with data types it returns the amount of memory allocated for the data type

32-bit and 64-bit

a)Write a program to convert fahrenheit to celsius and vice-versa
b)Write a program to accept number of days and convert into days,weeks and months

01/10/2021

1.Write a C program to convert Centimeter into Meter and Kilometer.
2.Write a C program to compute the square and cube of a given number

04/10/2021

Bitwise Operators-- & | ^ ~

High(1)--ON and Low(0)-OFF

Truth Table

Bitwise AND

a	b	a&b

0	0	0
0	1	0
1	0	0
1	1	1

Bitwise OR

a	b	a|b
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise XOR(^)

a	b	a^b
0	0	0
0	1	1
1	0	1
1	1	0

Bitwise NOT(!)

a	!a
0	1
1	0

1. a=12,b=10

```
      2)12(6
        12
        -
        0-r

        2)6(3
          6
          -
          0-r


          2)3(1
            2
            -
            1
```

b=10

```
            2)10(5
              10---0
              2)5(2
                4
                --
                 1
                 2)2(1
                   2
```

```
                  -
                  0




       1100

       1010

    ---------
       1000---Binary form


      2^3    2^2    2^1  2^0
      1      0      0    0

  8*1+4*0+2*0+1*0=8(10)
```

Home work

Consider a=12,b=10, | ,^ and ~ using Truth table

UNIT-II

SELECTION AND DECISION MAKING

Structured Programming
---
sequence
Decision Making or Condition or selection--if,if-else,if..else-if--else,nested if,switch
Iteration or Looping or Repetition--while,do..while and for

paradigm

Decision Making--Conditional Flow of Control

Branching---Conditional(if) and Unconditional Branching(break,label and continue)

Bi-directional Branching
---
if statement
---
a)Simple if
b)if..else
c)if-else..if-else (else-if ladder)
d)Nested if
e)Switch-Case --Case Control Structure

a)Simple if

Performs an action if the condition is true
Otherwise the action is ignored
Syntax
---

if(condition)...logical and relational Operators
{
//statement(s);st1;
st2;
}

Write a program to check for single digit,two digit and three digits
b)if..else

Syntax
---
if(condition)
{
statement1;
}
else
{
statement2;
}

06/10/2021
---
c)if..else-if..else

elseif

Syntax
---
if(condition1)
{
statement 1;
}
else if(condition2)
{
statement 2;
}
else if(condition3)
{
statement 3;
}
else
{
statement 4;
}

Homework
---
a)Write a program to enter three angles of a triangle and check whether a triangle is possible or not.
if possible,then display whether it is an acute angled triangle,obtuse angled triangle and a right angled triangle.
Otherwise display triangle is not possible

b)Extension to the roots of the Quadratic Equation
d=-b+sqrt(b*b-4*a*c)
Display the value of the discriminant

if d>=0....display roots are real
if d<0 ...display roots are imaginary

Nested If
---
if statement placed within if statement

Syntax

```
if(condition 1)
{
if(condition 2)
statement 1;
else
statement 2;
}
else
{
if(condition 3)
statement 3;
else
statement 4;
}
```

Largest of 3 numbers using ternary operator..use nested if statement

08/10/2021

Case Control Structure/Menu Driven/User's Choice--switch..case
---
Multiple branching statement---transfers to a specific case to perform the defined job based upon the value of the switch variable

Syntax

```
switch(variable)
{
case constant 1: statement(s);
                 break;
case constant 2: statement(s);
                 break;

case constant 3: statement(s);
                 break;

case constant 4: statement(s);
                 break;

----
----
default:statement(s);


}
```

break---comes out of the switch block

Components of the Switch block
---
a)Switch Variable or Control Variable--numeric,char

Used along with the switch statement under brackets that decides the case

to be used for execution

b)Break Statement

Used at the end of the case block
Terminates the control out of the switch block after completion of a particular case

c)Default Case

If no case is available for the given value of switch variable,default case is used
Displays the message for the user indicating that wrong choice is opted

Write a program to accept eno,ename,basic,hra,da
Calculate gross
tax--5% on basic
Calculate net

11/10/2021

ITERATION THROUGH LOOPS

Conditional Repetitive Type flow

while
do..while
for

true or false

Looping structure parts
---
a)Control Variable:Initial Value..A variable,which starts with an initial value and determines the duration of the repetition
b)Body of the Loop:
A Set of statements,which are executed within the loop simultaneously

c)Test Condition
loop has to be terminated or executed,depending on the test condition
The control enters the body of the loop for execution till the test condition is true,otherwise terminates

d)Step Value-increment/decrement--updating(++/--)

Based on the iterations

Loops	
Fixed	UnFixed
for	while,do..while

Fixed Iterations

Statements are repeated for a fixed number of times.

For Loop

Conditional repetitive type of flow

Syntax

```
for(initialization;test-condition;step value)....++/--
{
//body of the loop or Loop Block
}
```

1 2 3 4 5

```
int i;
for(i=1;i<=5;i++)//i++......i=i+1
printf("%d",i);
```

1+1=2+1=3+1=4+1=5
I iteration
----
i=1--cv of i
II iteration
----
i=i+1
i=1+1=2--cv of i

III iteration
----
i=i+1
i=2+1=3--cv of i

IV iteration
----
i=i+1
i=3+1=4--cv of i

V iteration
----
i=i+1
i=4+1=5=--cv of i
18/10/2021

While loop

Unfixed iterations
Syntax

```
initialization;
while(test condition)
{
//body of the loop
step value;//increment/decrement
}
```

--Conditional Controlled Loop
--The control terminates when the test condition is false
--Entry controlled Loop

```
do..while
---------
--Unfixed iterations
--Exit Controlled Loop

Syntax
------
initialization;
do
{
//body of the loop
step value;//increment/decrement
}
while(condition);

Differences between while and do..while loop
--------------------------------------------
while loop                                  do..while loop
----------                                  --------------
a)Entry controlled                          Exit Controlled loop
b)condition is not satisfied                atleast once


Program to find the sum of n natural numbers
--------------------------------------------
int i,sum=0;
for(i=1;i<=5;i++)
{
sum=sum+i;//
}

Iterations
----------
I)sum=0+1=1
II)sum=1+2=3
III)sum=3+3=6
Iv)sum=6+4=10
v)sum=10+5=15

n*(n+1)/2
Fibonaccci Series
-----------------

    0   1    1    2   3   5   8....n

    f1  f2   f3
        f1   f2   f3
             f1   f2  f3
                  f1  f2  f3
                      f1  f2  f3
.................


int i,f1=0,f2=1,f3;

f3=f1+f2;//f3=1,2,3,5,8

f1=f2;//f1=3
f2=f3;//f2=5
```

Factorial of a given number

5

n*(n-1)*(n-2)*.....*1
5*4*3*2*1=120

Assignment
---
a)Write a program to print the multiplication table in the following format

n=5

5*1=5
5*2=10
5*3=15
5*4=20
5*5=25

5*10=50

20/10/2021

Write a program to check whether a given number is prime or not

2%1==0---1
2%2==0---1

```
int i,n,count=0;

for(i=1;i<=n;i++)
{
if(n%i==0)
count++;
}
if(count==2)
```

Explaination

I iteration:count=count+1-----count=0+1=1
II iteration:count=1+1=2

Nested for loop

A for loop within a for loop

```
 for(initialization;test-condition;increment/decrement)--outer loop
  {
    for(initialization;test-condition;increment/decrement)--inner loop
     {
      //body of the loop
     }
  }
```

Eg
--
```
for(i=1;i<=2;i++)//outer loop
```

```
{
for(j=1;j<=2;j++)//inner loop
{
printf("%d",j);
}
}
```

when i=1,j=1,2
when i=2,j=1,2

c)Generate all the prime numbers between 1 and n, where n is a value supplied by the user.
d)Find the sum of individual digits of a positive integer and find the reverse of the given number.

ARRAYS

Collection of similar(homogeneous) elements under a common name

Syntax

```
 <data-type> <variable-name>[value];
 int a[5];--subscript or index
 char ch[5];
```

subscripted variable or indexed variable

Array index starts with 0

Types of Arrays
----
a)One dimensional or Single dimensional Array--row or col
Syntax

```
 <data-type> <variable-name>[value];
 int a[5];--subscript or index
```

```
    0   1   2   3   4   (0 to n-1)

    67  45  12 -9  -87
```

b)Multi dimensional
- Two----matrix---row and col
- Three

22/10/2021

Find the sum of individual digits of a positive integer and find the reverse of the given number.

n=732 (10)

7+3+2=12

H	T	U
7	3	2

```
        2     3     7
  1  ----10^0

  10--- 10^1

  100---10^2

      int  n,sum=0,rev=0

 I iteration
------------

               10 )732(73--n
                   730
                   ---
                     2--rem

    sum=sum+rem----->sum=0+2=2
    rev=rev*10+rem;
   rev=0*10+2=2



II iteration
------------
                  10)73( 7--n
                     70
                     --
                      3--rem

sum=sum+rem-----2+3=5



III iteration
-------------

7/10,7%10----7

sum=sum+rem--->5+7=12

  while(n>0)
  {
  rem=n%10;
  sum=sum+rem;
  n=n/10;// n/=10
  }


  n=732

  rev=237


     n=12

      10^1  10^0
n=      1     2----12/10---1,12%10--2
rev=    2     1-----1/10,1%10---1
```

```
      rev=rev*10+d

rev=0*10+2=2

rev=2*10+1---20+1=21

25/10/2021
----------
                INTRODUCTION TO ARRAYS
                ----------------------
Defition
--------
<data-type> <variable-name>[value];
Types of Arrays
---------------
a)One dimensional or Single dimensional Array--row or col
Syntax
------
 <data-type> <variable-name>[value];
 int a[5];--subscript or index

    0   1   2   3   4   (0 to n-1)

    67  45  12 -9  -87

b)Multi dimensional
      - Two----matrix---row and col

Syntax:
------
<data-type> <variable>[value1][value2];
int a[2][2];

    Nested for loops
    ----------------

  for(i=0;i<2;i++)--outer loop
  {
  for(j=0;j<2;j++)--inner loop
  {
    //body of the loop
  }
}

I iteration
-----------
when i=0,j=0,1--2 times


II iteration
-----------
when i=1,j=0,1--2 times


    a[i][j]

       0   1---j

 i  0  11 12---I Row
```

```
    1  13 14 ---II Row

 Initializing an Array
----------------------

a[0]=12
a[1]=13
a[2]=34
a[3]=56
a[4]=78

int a[5];//Declaration of an array
int a[5]={12,13,34,56,78};//initialization

DDA--displayed in matrix format
---
int a[2][2];//Declaration
int a[2][2]={{12,13},{34,56}};//Initialization


        0     1
  0     11    12---12 will be printed

  1     13    14


I row
-----
a[0][0]=11
a[0][1]=12---


II row
------
a[1][0]=13
a[1][1]=14---


a)Write a program to find the sum of the diagonal elements of an array--
trace of a matrix
Square matrix

 rows and cols should be same

     int a[3][3];

i       0   1   2--j
0       1   2   3
1       4   5   6
2       7   8   9


1+5+9=15
3+5+7=15

a[i][j]


b)Consider 2 square matrices a and b with the dimension 3*3
```

Find the sum of the elements of a and b and store in c[3][3]

```
  c=a+b;
  c[i][j]=a[i][j]+b[i][j]
```

CHARACTER ARRAYS AND STRINGS

Alphabets,digits,special characters....

' '

Syntax

<data-type><variable-name>;
char ch; //declaration
char ch='a'; //initialization
%c
getchar()--accept the character
putchar()---display the character

include <ctype.h>

String---%s

1-D Array of characters terminated by '\0'--null character

'h','a','i'

  "hai"

include <string.h>---built in string functions

     'h' 'a' 'i' '\0'

Number of characters --3

char ch[]="Hello";

Standard String Library Functions--Built-in functions
---
strlen()----length of the given string(number of characters)
strrev()---reversed string
strcmp()---Compares two strings
strcat()---appends one string to another
strlwr()---converts an upper case string to lower case string
strupr()---converts a lower case string to upper case string
strstr()----Finds first occurance of a given string in another string
strdup()---dupicates a string
strcpy()--copies one string to another

ASCII CODES

A-Z----65-90
a-z---97-122
0-9--48-57
27/10/2021

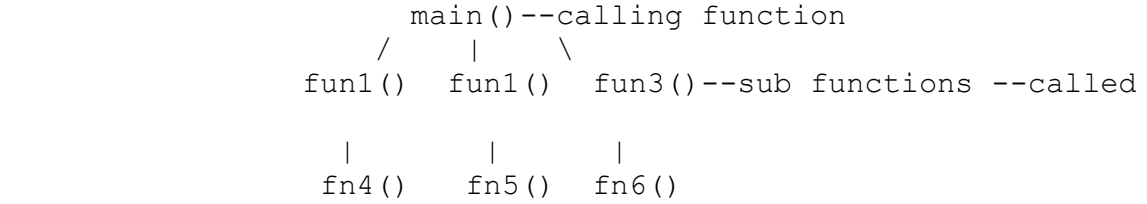
UNIT-III

Modular Programming

Functions

Built-in Functions or System defined Functions
User-defined Functions

```
char s1[]="Hello";

 strlen(s1);---5
```



Function

A function is a self-contained block of statements that performs a coherent task

Types of Functions

Built-in Functions or System defined Functions
User-defined Functions

Syntax for Function

```
<return-type>function-name(data-type variable1,data-type variable2,......)--argument list or parameter list
{
//body of the loop
}
```

A function can call itself again and again..recursion

29/10/2021

A program module(a part of a program) used simultaneously at different instances in a program to perform a coherent task,is known as a function

Syntax for Function

```
<return-type>function-name(data-type variable1,data-type variable2,......)-- parameter list
{
//body of the loop
}
```

{}--block

Features of functions

---
a)Reusablility
b)Hiding
c)Dividing a complex computational task into a collection of smaller functions that makes problem solving easier and modular

```
<return-type>function-name(data-type variable1,data-type
variable2,.....)//header/prototype
{
//body of the loop
}
```

Eg
--
```
int add(int a,int b)//header or prototype
{
int sum;
sum=a+b;
return(sum);
}
```

add(int,int)--signature

-return-type is the data type used before the function-name that indicates the type of the value as outcome/result

-Parameter-list

List of variables with the respective data types,required by the function for internal computation passed during the function call

Return Statement

The statement which sends back the value/result/outcome from a function to the caller program

--At the end of the function,as a function terminator
```
return <value>;
return;
```

```
--int add(int a,int b)//header or prototype
{
int sum;
sum=a+b;
return(sum);
printf("\n\t Hello...");//invalid
}
```

--Returns only a single value

```
return(s,p);//invalid
```

--A number of return statements may be included to terminate the method from a specific point
```
if(a>b)
 return(a);
else
return(b);
```

--In case of multiple functions,only one return statement can be activated to return the control
return(s);
return(p);

--Once control exits from the function,it can't reapper in the function again

```
for(i=1;i<=10;i++)
{
if(i>=5)
return(i);//exits from the function
printf("%d",i);
}
```

Parameter Passing Techniques
---
a)Function with no parameters and no return values
b)Funcion with parameters and return values
c)Function with parameters,but no return values
d)Function with no parameters,but return values

Actual Parameters and Formal Parameters

Actual Parameters

The parameters described in the caller definition(the values passed to the method) are known as Actual parameters.

Formal Parameters

The parameters, which are described in the function definition(receive the value) are called Formal
parameters.

```
       int main()//calling function
         {
         int a=5,b=6;
         int c=sum(a,b);//caller definition..Actual parameters
         printf("\n\t Sum of the two numbers:%d",sum(a,b));
         printf("\n\t Sum of the two numbers:%d",c);

         return 0;
         }
      int sum(int x,int y)//formal parameters
       {
        int add;
        add=x+y;      block of statement(s)
        return(add);
       }
```

01/11/2021
---
a)Write a program to demonstrate a function with no arguments,but return values

Techniques involved in passing elements to a function
---
a)Pass by value

b)Pass by reference(address)--&--hold

```
 int a=5;

  a--variable
  5--value
 4096--address

  &--Address of the variable
  *--value at the address
```

Storage Classes

Scope and life-time of variables and/or functions within a program

a)Where the variable would be stored?
b)What would be the default initial value of the variable?
c)What is the scope of the variable..ie., in which functions the value of the variable would be available
d)Life of the variabe

4 types of storage classes
---
a)Auto or Automatic
b)Static
c)Extern
d)Register

a)Auto
  ---
a)Storage   : Memory
b)Default Value: An unpredictable value,often called garbage value
c)Scope   :Local to the block in which the variable is defined
d)Life   :Till the control remains wihtin the block in which the variable is defined

Eg:
--
```
int main()
{
auto int i,j;
printf("%d %d",i,j);
return 0;
}
```

b)Register--The value stored in the CPU register can always be accessed faster than the one that is stored in the memory

a)Storage   : CPU-Registers
b)Default Value: An unpredictable value,often called garbage value
c)Scope   :Local to the block in which the variable is defined
d)Life   :Till the control remains wihtin the block in which the variable is defined

Eg:

Looping

```
register int i;
for(i=1;i<=10;i++)
printf("%d",i);
```

c)Static Storage Class
---
a)Storage : Memory
b)Default Value: Zero
c)Scope :Local to the block in which the variable is defined
d)Life :Value of the variable persists between different function calls

Static:

A reserved word which controls both lifetime as well visibility
Used with a constant variable or a method that is same for every instance of a class

d)Extern Storage Class
---
a)Storage : Memory
b)Default Value: Zero
c)Scope :Global
d)Life :As long as the program's execution doesn't come to an end
External variables are declared outside all the functions and are available to all the functions
Eg:
--
```
int a;//Extern
main()
{
}
```

Storage Class	Storage	Default Value	Scope	Life	Example
Auto					
Register					
Static					
Extern					

Recursion

A function calling itself repeatedly again and again,the process is known as recursion and the function ia called recursive function
The call---recursive call

03/09/2021

Pass by value
Pass by reference
&---address of the variable
*--value at the address
The process of passing the reference(address) of actual parameters to the formal parameters.
Any change made in the formal parameters will be reflected on the actual paramters

Towers of Hanoi--swapping

```
     small       A     B     C
     medium      |     |     |
      largest    |     |     |
```

```
int a=10,b=30;
int temp;

temp=a;
a=b;
b=temp;

//Program on function with arguments but no return values
void main()
{
char s1[15];
printf("\n\t Enter any String:");
gets(s1);//Built-In function
printf("\n\t The String is...");
display(s1);//user-Defined function
}

void display(char s2[])
{
printf("%s",s2);
return;
}
```

05/11/2021

---

a)Write a program to check whether a given character is a vowel or not using functions
b)Write a program to check the biggest of 2 numbers using a function
c)Write a program to pass elements into a function using call by reference
d)Write a program to pass a DDA into a function and display the values

Passing an Array to a function

Individual elements or an entire array can be passed to a function
a)If the values stored in an array are passed,single element at a time,it is called pass by value (or) call by value
b)Pass of the name of the array without any index,entire array at a time is called Array passing by reference (or) Call by reference.

```
    a       b
    10      20
    6896    6898

  & *

 a=10
 &a=6896

*(&a)---

*(6896)--10
```

```
add(&a,&b);

add(int *a,int *b)
```

const

Keyword to initialize the constant size value
Eg:
--
```
const int a=2;
                  row col
void display(int [] [n]);//col is constant value,row is optional
```

Recursion

A process where a function calls a copy of itself to work in a similar problem
The function is called a recursive function and the function calls are called recursive calls

Note:
---
initial and exit condition--otherwise infinite loop occurs

Features of Recursion

Decomposition--
Composition--
Base Value/Stopping Case----1

Eg:Factorial of a given number,Fibonacci Series,Towers of Hanoi

4!

 n*(n-1)*(n-2)*......*3*2*1

 n*(n-1)
 4*(4-1)*(4-2)*(4-3)
4*3*2*1=24
n-1-1
n-2
n-2-1
n-3
Iteration and Recursion

Iteration algorithm uses looping constructs
Recursion algorithm uses branching structures

Recursion is less efficient than Iteration in both space and complexity

```
int fact(int n)
{
if(n==0)
return 1;
else if (n==1)
return 1;
```

```
else
return n*(n-1)
}
```

10/11/2021

Explain Looping statements

Define Looping statements
List different looping statements
a)for
b)while
c)do..while
for

Explain in points

Syntax

Example

```
for(i=1;i<=10;i++)
{
//statement(s);
}
```

while
---

do..while

Programming

```
//comment
//inputs
//computation
//display
void main()
{
for(i=1;i<=10;i++)
{
}
}
```

Linear Search/Sequential Search

Searching of an item/element begins at the start of the array..0th position

Program

```
//Program to search for an element in an array
#include <stdio.h>
void main()
{
int a[10],n,ns,i,found=1;
printf("\n\t Enter n:");
```

```
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("\n\t Enter:");
scanf("%d",&a[i]);
}
printf("\n\t Enter the element to be searched for:");
scanf("%d",&ns);

for(i=0;i<n;i++)
{
if(a[i]==ns)
found=1;
break;
}

}
if(found==1)
printf("\n\t Search element found..");
else
printf("\n\t Search element not found..");
}
```

Program to display the reverse of a given number and check if it is palindrome or not

---

```
void main()
{
int n,rem=0,rev=0;
printf("\n\t Enter n:");
scanf("%d",&n);
while(n>0)
{
rem=n%10;
rev=rev*10+rem;//I)0*10+2=2,II)2*10+1=20+1=21,III)21*10+3=210+3=213
n=n/10;
}
printf("\n\t Reverse Number:%d",rev);
}
```

```
    100  10  1
    3    1   2
```

12/11/2021

Objective

Descriptive

Thoery

Write in points

Flow chart

Define

Mention the symbols

Step 1:Start

Step 2:

Step 3:
Step 4:
Step 5:Stop

GCC---GNU Compiler Collection--optimizing compiler supporting various programming languages,hardware architectures and Operating Systems

Key compnent of GNU ..Standard Compiler related to GNU and the Linux Kernel

GNU's not unix--used in the development of Unix like OS that comes with source code which can be copied.modified and distributed ..
recursive acronym,in which one of the letters stands for the acronymn itself

Explain different tokens in C language?

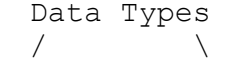
Individual Component

Identifiers--variables
Literals--constants
Operators--Explain different operators in c

Keywords--
Punctuators---, ; :..
Separators---{} () []...

Data Types

Define
Categories



Primary or System defined or Built-in Secondary or Reference or Derived

```
int a=2;//initialization
float f=6.5;
```

Type Converstion

Implicit and Explicit Conversion

```
a=2,b=3,c=4

(a*b)+c

header file
void main()
{
inputs
computations
print

}
```

Explain control statements
---

a)Decision Making (or)Conditional Control,case control structure..switch case
  Branching--Allow the flow of execution to jump to a different part of the program
   ---Conditional---if,if..else,if--else-if-else,nested if
   ---unconditional--goto label:
b)Iteration or Looping--for,while,do..while
c)Jumping--break,continue and goto

while	do-while
def	def
syntax	syntax
eg	eg

```
void main()
{
for(i=1;i<=10;i++)
{
if(i==5)
break;
}
printf("%d",i);
}
```

```
void main()
{
for(i=0;i<=10;i++)
{
if(i%2==0)
continue;
}
printf("%d",i);
}
```

Looping Statement

Definition
for--fixed--entry controlled loop
syntax
ex
while
syntax
ex
do..while
syntax
ex

Arrays

Definition
```
int a[5];//
int a[]={11,12,13,14,15};
```

Types of Arrays

Single or One dimensional Array---x or y

[]
int a[5];

Multi-Dimensional Array
    -Two-------Matrix---X and Y
     int a[][]={{1,2},{2,3}};

    -Three-----x,y,z

Strings
----

char ch;---'h','a','i'
char s1[10];---"hai"

String Functions
----
list the functions with def

strlen()
----
int s[]="Hello";
printf("\n\t Length of the string:%d",strlen(s);

Difference between if-else and switch case
----

if-else	switch
a)results in boolean type value	results in int/char type value
b)integers/floating points and char	not applicable for floating points
c)true/false..either of blocks are given switch value operated	specific case is operated for a
d)performs task on relational and logical expressions	test for equality
e)no default operation is not available for a given switch	A default case is applied,if case value

break

Used for unusal termination of a block
The control exits from the block

for(initial value;test-condition;inc/dec)--step value
{
if(condition is true)
break;
}

out of the loop block

while(condition)
{

```
if(condition is true)
break;
}

out of the loop block
```

```
Continue
--------
Opposite to break statement
The control skips rest of the statements for that value and resumes for the
next iteration

for(initial value;test-condition;inc/dec)--step value
{
if(condition is true)              |
continue;.........................
}
```

```
while(condition)
{
if(condition is true)
continue;.................
}

goto --Jumping statement..unconditional branching statement
----
label:
//code
goto label;
```

```
#include <stdio.h>
void main()
{
int n,i=1;
printf("\n\t Enter n:");
scanf("%d",&n);
table:
printf("%d*%d=%d\n",n,i,n*i);
i++;
if(i<=10)
goto table;
}
```

```
22/11/2021
----------
Write a program to compute the sum of the series

s=1+1/2+1/3+....+1/n

int i,n,s;

for(i=1;i<=n;i++)
```

```
{
s=(float)(s+1/i);
}
```

Program to check whether a given number is armstrong or not using function

---

```
n=153

1^3+5^3+3^3

1+125+27=153

void main()
{
int n;
printf("\n\t Enter n:");
scanf("%d",&n);
armstrong(n);
}

void armstrong(int n)
{
int temp,rem,sum=0;
temp=n;
while(n>0)
{
rem=n%10;
s=s+rem*rem*rem;
n/=10;
}
if(temp==sum)
printf("\n\t Armstrong Number..");
else
printf("\n\t Not an Armstrong Number...");
}
```

24/11/2021

```
scanf("%[^\n]", variable_name);
```

It is an edit conversion code.
The edit conversion code %[^\n] can be used as an alternative of gets.
C supports this format specification with a scanf() function.
This edit conversion code can be used to read a line containing characters like variables and even white spaces.
It means they cannot be used for reading a text containing more than one word, especially with white space.

Write a program using function to return the length of a string

26/11/2021

Recursion

A way to decompose a task into another smaller subtasks

Factorial

---

n*(n-1)*(n-2)*.....*1--base

4*3*2*1=24
n-1-1=n-2
n-2-1=n-3
3 Features

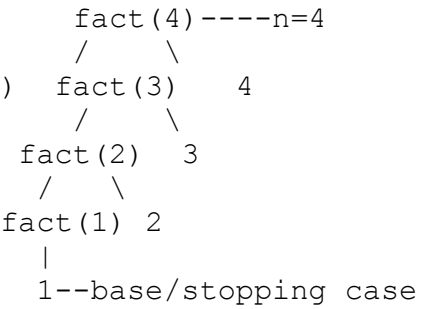
Recursive Design
---
a)Decomposition---(n-1)!
b)Composition---*n
c)Base/Stopping Case--1!

Iteration Vs Recursion

Iteration algorithm uses looping construct
Recursion algorithm uses a branching construct

Less efficient than iteration in space and complexity

Example



Fibonacci Series

0,1,1,2,3,5,....

Based on 10 digit number and a decimal point
Decomposition and Composition
---
fib(n)=fib(n-1)+fib(n-2)
(or)
fib(n+2)=f(n+1)+f(n)

Series begins with
---
fib(0)=0
fib(1)=1

Lab

Factorial of a given number using Recursion

Pass by value
Pass by reference

a---variable
10---value
6780---address

&,*

```
&--address of the variable
*--value at the address

void main()
{
int a=10,b=20,c;
c=sum(&a,&b);//pass by value---  -12,-14
printf("\n\t Sum:%d",c);
}


int sum(int *x,int *y)
{
int add;
add=x+y;
return(add);
}

*x=*(&a)=*(-12)---10
*y=*(&b)=*(-14)---20
```

Linear Search/Sequential Search

```
          a[10]

  a[0]  a[1]  a[2]  a[3]  a[4]
   11    12    13     14    15
```

input ----a[10],i,ns,found=0
search for the element in the array sequentially
if the search element is found ...search successful

Passing an array to a function

Pass by value

If the values stored in the array are passed,single element at a time...pass by value or call by value

Pass by reference

Pass the name of the array without any index,entire array at a time is called pass by reference or call by reference

29/11/2021

Arrays to Function

- Pass by value
- Pass by reference
- String as parameter

Programs

a)Write a program to pass a DDA to a function [][]

--Pass by value and Pass by reference

```
const int i=2;
```

b)Write a program to pass a string to a function and display the length of the string
--strlen() function,shouldn't be used

Fibonacci Series

0,1,1,2,3,5,8,.....n

Recursive Design

Decomposition
Composition

fib(n)=fib(n-1)+fib(n-2)

Base Value
fib(0)=0-first value
fib(1)=1--second value

f(n+2)=f(n+1)+f(n)

```
void main()
{
int n,fib;
printf("\n\t Enter n:");
scanf("%d",&n);
fib=fibo(n);
for(i=1;i<=n;i++)
printf("\n\t Fibonacci Series:%d",fib);
}

int fibo(int n)
{
int fib;
if(n>2)
{
fib=fibo(n-1)+fibo(n-2)//fib=fib(3-1)+fib(3-2)----fib(2)+fib(1)-----1+0=1
}
else if(n==2)
{
fib=1;
}
else
{
fib=0;
}

return fib;

}
```

Display the n Fibonacci series

C preprocessor is a program that processes a program before it is passed to the compiler

Before a C program is compiled,it is passed through another program known as "Source code".

The preprocessor works on the source code and creates "Expanded Source Code",as per the directives used in the program--source code

Each Preprocessor directive begins with the symobol '#'

Preprocessor Directives
---
a)File Inclusion
b)Macro Expansion
c)Conditional Compilation
d)Miscellaneous Directives

a)File Inclusion

#include "filename"

#include <filename>

--Common for the files with .h extension

h--header file,as it contains statements which are included to move to the head of the program

The prototypes of all the library functions are grouped into different categories and in different header files

"math.h"
"conio.h"

#include "filename.h"--will look for the file name in the current directory and specified list of directories

#include <filename.h>--Looks for the file in the specified list of directories

#include <stdio.h>
#include <math.h>

b)Macros

A macro is a fragment of code which has been given a name.

Whenever the name is used,it is replaced by the contents of the macro

Why macros are used?

Used to define constant values that are been used repeatedly in the program

Can macros accept arguments? NO

Macros are called Function-Like Arguments

A macro is a series of commands and instructions that are grouped together as a single command to accomplish a task

Simple Macro

```
#include <stdio.h>
#define PI 3.14159--Macro Expansion
int main()
{
float r=6.5,area;
area=PI*r*r;
printf("\n\t Area of a Circle:%.2f",area);
return 0;
}
```

//Macro Template

01/12/2021

```
#define PI--symbolic constant
```

Macro with Arguments

```
#include <stdio.h>
#define AREA(x)(3.14*x*x)

void main()
{
float r1=5.0,r2=6.0,a;
a=AREA(r1);
printf("\n\t Area of Circle:%.2f",a);
a=AREA(r2);
printf("\n\t Area of Circle:%.2f",a);
}
```

The preprocessor would replace every AREA(x) with(3.14*x*x)
x would be substituted in with the argument used in the macro

```
float a=AREA(r1);
float a=(3.14*r1*r1);
```

Examples

```
#define AND &&
#define RANGE1 (n>20 AND n<50)
```

Difference between Macro and Function
---

a)A macro is defined with a preprocessor directive	Function is not defined with a pre-processor directive
b)Macros are pre-processed	Functions are not pre-processed
c)Macro doesn't return any value	Function returns value
d)Macro doesn't check any compile time errors	Function checks compile time errors
e)Before compilation process,the macro name is	In a function call,transfer of function takes place

replaced by the macro value

Eg:

```
--
#define AND &&
```

Eg:

```
--
main()
{
func1();
func2();
}
```

CONDITIONAL COMPILATION

The compiler skips a part of the source code by inserting the preprocessing commands #ifdef and #endif

Syntax:

```
------
#ifdef macroname
statement1;
statement2;
statement3;
..........
..........
#endif
```

Programs

---

a)Write a program to pass a DDA to a function [][]

--Pass by value and Pass by reference

```
const int i=2;
```

b)Write a program to pass a string to a function and display the length of the string

--strlen() function,shouldn't be used

c)Write a program to find the maximum and minimum element in an array

d)Linear search.

e)Replace a character of string either from beginning or ending or at a location.

```
"Pello"

s1[10];

a[0] a[1] a[2] a[3]  a[4]
```

Towers of Hanoi

```
  T1        T3   T2


small
medium
large

bottom
```

Rules

---
a)Only one disk should be moved at a time
b)No larger disk should be placed on the smaller disk
c)Only top disk can be removed

03/12/2021
---
a[10],i,ns,found=0

a[0]	a[1]	a[2]	a[3]	a[4]
11	12	13	14	15

Write a program to find the largest and smallest element in an array

```
int a[10],i,n,lar,small;
Enter n
Accept the elements
lar=a[0]-11,small=a[0]-11
for(i=0;i<n;i++)
{
if(a[i]>lar)---11
lar=a[i];---33
if(a[i]<small)--11
small=a[i];
}
```

a[0]	a[1]	a[2]
11	22	33

```
char s1[10],i,ch1,ch2;
Accept the string
gets(s1);
```

06/12/2021
---
a)Write a C program to find out the smallest values among A,B and C
b)Explain the basic structure of C program
c)Give the syntax of for,while and do..while with examples
d)What is a preprocessor statement in C?
e)What is nested loop?
f)What are the basic datatypes in C?
g)Define a variable and constant with examples
h)Explain different operators in C
i)What is the difference between break and continue
j)Define string and explain different string functions

```
int a[][]={{"aaa","bbb","ccc"},{"ddd","eee","fff"}};

int a[3][3];

int a;
int a=10;
```

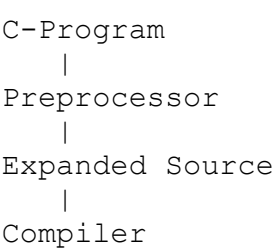
	0	1	2
0	aaa	bbb	ccc

```
1   ddd    eee    fff


Online Compilers
----------------
codechef
hackerrank
hackerearth


Different uses of #define
-------------------------
a)Macro to create symbolic constants
------------------------------------
#define PI 3.14

b)Macros for calculation
------------------------

#define AREA(x) (3.14*x*x)

c)Conditional and Looping
-------------------------
#define check(x)
if(x%2==0)
printf("\n\t Even...");
else
printf("\n\t Odd...");

d)Nested Macro
--------------
#define square(x) (x*x)
#define cube(x)  (square(x)*x)


#undef
------
Undefines symbolic constant created by #define directive

Syntax
------
#define macro-name
#undef identifier/macro-name

Eg
--
#define PI 3.14
#undef PI

Pass by reference---address
---------------------------
int a;

   a--variable name
  10--value
  3098--address

scanf("%d",&a);
```

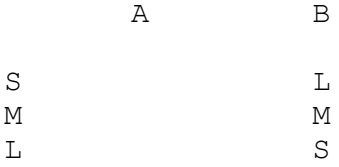
```
&--address of the variable
*--value at the address

main()
{
int a=10,b=20;
sum(&a,&b);//sum(a,b)
}
int sum(int *,int *)//int sum(int *,int *)
{
}

*a----*(&a)---*(-12)--10
*b---*(&b)---*(-14)--20


Compiler Directives
--------------------
Directives and Description
--------------------------
a)Macro Substitution
--------------------
#define---Substitutes a preprocessor macro
#undef--Undefines a macro
b)File Inclusion
-----------------
#include--Inserts a particular header file from another file

c)Conditional Compilation
-------------------------
#ifdef--Returns true if the macro is defined
#ifndef--Returns true if the macro is not defined
#if--Tests if a compile time condition is true
#else--The alternative for #if
#elif--#else and #if in one statement
#endif--Ends preprocessor conditional
d)Miscellaneous
---------------
#error--prints error message on the output
#pragma--Issues special commands to the compiler using a standardized
method

08/12/2021
----------
Compiler Control directives
---------------------------
Includes macro based on the condition

Syntax:
------
#ifdef identifier

Example
-------
#define PI 3.14
void main()
{
#ifdef PI
printf("\n\t Area can be calculated...");
#else
```

```
printf("\n\t Area can't be calculated...");
}

#ifndef
-------
Checks whether macro identifier is defined or not

Syntax
-----
#ifndef  identifier

#ifndef  PI

void main()
{
#ifndef PI
printf("\n\t Area can be calculated...");
#else
printf("\n\t Area can't be calculated...");
}

#error
------
Prints the error message on the standard output and displays user defined
message

#error "message"
Eg:
---
void main()
{
#ifdef PI

printf("\n\t Area can be calculated...");

#error  "Area can't be calculated"

#endif

}

Pragma
------
Gives special commands to the compiler using standardized method

Use to ON and OFF warnings

Syntax
------
#pragma warn + warning message notation
#pragma warn - warning message notation

+--turns on warning
- ->turns off the warning

Different Warning Messages
--------------------------
stu---undef structure
eff--code has no effect
rch---unreachable code
```

rvl--function should return a value
voi--void,function can't return a value
startup---before main
exit---just before program terminates
10/12/2021



Towers of Hanoi
---
n=3------ 2^n-1--------7



a)Only one disk can be moved at a time
b)The largest disk should not be placed on the smaller ones
c)Only the top disk should be moved

main()--concept--Ramya
{
int n;---Akash,

Nithyananda ---program
printf("\n\t Enter the number of discs...");
scanf("%d",&n);
TOH(n,'X','Y','Z');
}
void TOH(int n,char A,char B,char C)
{
TOH(n-1,A,C,B);
printf("\n %c to %c",A,B);
TOH(n-1,C,B,A);
}

UNIT-IV POINTERS

A variable which stores the address of another variable,where the variable can be int,char,arrays,functions and pointer
Note

The size of the pointer variable depends on the machine

32 bit----2 bytes

Eg:
---
int n=10;
int *p=&n;//variable p of type pointer is pointing to the address of the variable n of the type integer

The pointer is declared using an asterisk(*),known as indirection operator used to dereference a pointer

```
      normal                          pointer
  n---variable                     p ---variable
  10--value                       3406---value(Adddress of n)
 3406--address                   3408--address of p
```

Declaring a pointer

<data-type> *<variable-name>;

Advantages
---
a)Reduce the code and improves the performance
b)Return multiple values from a function
c)Makes to access memory location in the memory

Usage of pointer

DMA---Dynamic Memory Allocation
Arrays---Consecutive memory allocation

13/12/2021
---
malloc()
calloc()
dealloc()
free()

NULL POINTER

A pointer that is not assigned any vale but NULL is known as a NULL pointer

```
int *p=NULL;
```

a)Illustrate call by value --Aakhil
b)Illustrate call by reference ----Divyasri
c)Find the length of the given string using pointers--Poojitha
d)Reverse a string using pointers

```
       0   1   2
       'h' 'a' 'i'
      1001 1002 1003
int i;
char s[]="hai";
for(i=0;s[i]!='\0';i++)
{
}
printf("%d",i)
```

15/12/2021

Address Arithmetic or Pointer Arithmetic

32--2 bytes
64--4 bytes
p=p+2

1064----1066

1068

If p is a pointer to some element of an array,then p++ increments p to point to the next element
and p+=i(p=p+i),increments to point i elements.

Note

All the types of arithmetic operations are not possible with pointers

Valid Operations performed on pointers
---
a)Addition of an integer to a pointer and increment operation
b)Subtraction of an integer to a pointer and decrement operation
c)Subtraction of a pointer from another pointer of the same type

Invalid Operations
---
a)Addition,Multiplication and Division of two pointers
b)Multiplication of a pointer by a number
c)Division of a pointer by a number
d)Addition of float and double values to pointers

Eg
--
Suppose p is an integer pointer and x is an integer variable
Interpret the following
---
a)x=*p++

x=*p followed by p=p+1

b)x=(*)p++

x=*p followed by *p=*p+1

c)x=*++p

p=p+1 followed by x=*p

d)x=(*)++p

*p=*p+1 followed by x=*p

Pointer contain address
The result of an arithmetic operation performed on the pointer will also be a pointer

Increment
Decrement
Addition
Subtraction
Comparision
int n,*p,**p;

```
    n           *p1                                  **p2         ***p3
    10         1034--value-address of n         1038         1042
    1034       1038                             1042         1046
```

Pointer to Pointer(Double Pointer)

A pointer which stores the address of another pointer is known as pointer to pointer or a double pointer
Pointer reduces the access time of the variable

variable pointer pointer

value address address

```
    n           *p1                                  **p2                ***p3
    10         1034--value-address of n         1038                1042
    1034       1038 --address                   1042-address        1046--
address
```

```
void main()
{
int n=10;
int *p1,**p2;
p1=&a;//Pointer p1 is pointing to the address of a
p2=&p1;//Pointer p2(double-pointer) is pointing to the address of p1
}
```

17/12/2021

```
int a=10,*p;
p=&a;---Initializing a pointer variable
```

Pointers and Arrays

Single

```
int a[5];
int *p[5]=&a[0];//&a---pointer p is pointing to the address of integer a
```

```
    a[0]   a[1]  a[2]   a[3]  a[4]
     11    12    13     14    15
    1346   1350  1354   1358  1362
```

Multi
--Two
```
 int a[2][2];

     0  1
0   11 12
1   13 14


  a[0][0]  a[0][1]   a[1][0]  a[1][1]
    11       12       13        14
   <----I row------->   <----II row---->
int *p=&a[0][0];
```

--Three

int [5][2][3];

Functions

Pass by value
Pass by reference--address
void show(int);
void(*p)
(int)=&show;//Pointer p pointing to the address of a function

Structures

Pointers save the memory space
Execution of the pointers is faster,because of the direct access to the memory location
Memory is accessed efficiently
Allocation and Deallocation is easy
Used with Data Structures

Dangling pointer

If the programmer fails to initialize pointer with an invalid address,then this type of initialized pointer is known as dangling pointer

Occurs at the time of object destruction

A dangling pointer is a pointer that occurs at a time,when the object is deallocated from the memory,without modifying the value of the pointer

void pointer

A pointer that can point to any data type

Dynamic Memory Allocation

The process of assigning the memory space during the run-time or execution
DMA is managed by 4 functions
4 functions
---
malloc()----Memory allocation

(datatype *)malloc(sizeof(datatype))---int -4 bytes

p=100 1 byte 1 byte 1 byte 1 byte
890 100 200 300 400

when the pointer variable points to the de-allocated memory,it is known as the dangling pointer

ptr=NULL---It's not a dangling pointer

malloc()----memory allocation
calloc()---Continuous allocation
realloc()---Extended or Reduced
free()----Deallocates the memory
(deallocation)

20/12/2021

COMMAND LINE ARGUMENTS

Arguments that are specified after the name of the program in the system's command line are known as command line arguments

```
main(int argc,char *argv[])
```

argc----counts the number of arguments,file name as first argument
argv[]---Contains the total number of arguments
The first argument is the file

Eg
--

```
void main(int argc,char *argv[])
{
printf("\n\t Program name:%s",arg[0]);


}
```

UNIT-V

STRUCTURE

Array--similar elements under a common name---homegoneous

Structure----heterogenous

Name-------char[20]
ID---------int
DOB---------char[20]
Section--char
Percentage--float

ID	Name	DOB	Section	Per
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

Definition

A group of one or more variables of different data types organized together under a single name

A collection of heterogeneous(dissimilar) types of data grouped together under a single name

A structure can be defined to be a group of logically related data items,which may be of different types stored in contiguous memory allocation

Heterogeneous user-defined type

How to refer a structure?

Structure name
The individual components of a structure are called structure members

Structure Declaration

Specifies the grouping of various data items into a single unit

Syntax

```
struct <structure-name>---tag
{
data-type variable1;
data-type variable2;
data-type variable3;
--------------------
--------------------
};
```

Tag is used to create structure variables
Eg:
--
```
struct stud
{
int sno;
char sname[15];
float avg;
};
```

Note:

The individual data members can't be intialized with in the structure beginning of the program

with in the main
before main

Structure Variables

Declared using structure definition
I Method

```
struct stud
{
int sno;
char sname[15];
float avg;
}s1,s2;--structure variables
```

Ii Method

```
struct stud
{
int sno;
char sname[15];
float avg;
};
struct stud s1,s2;
```

III Method

```
struct
{
int sno;
char sname[15];
float avg;
}
s1={};
```
Structure Initialization

```
struct stud
{
int sno;
char sname[15];
float avg;
};
struct stud s1={1,"aaa",95.0};
struct stud s2={2,"bbb",96.0};

*structure_variable.datamember;

s1.sno=1;
s1.sname="aaa";
s1.avg=95.0;


s2.sno=2;
s2.sname="bbb";
s2.avg=96.0;
```

Structures and Arrays

Similar elements

a)Array of structures
b)Structures containing arrays or arrays within a structure
c)Array of structures contain arrays

Array of Structures

```
struct <struct-name>
{
data-type var1;
data-type var2;
.............;
--------------;
}variable[index];
```

```
struct <struct-name>
{
data-type var1;
data-type var2;
.............;
--------------;
};

struct <struct-name> struct-variable[index];


Eg:
---
struct stud
{
int sno;
char sname[15];
float avg;
}s[3];


  sno  avg   sname-i--s1
   1   95.0  aaa
   2   96.0  bbb
   3   97.0  ccc

for(i=0;i<3;i++)
{
scanf("%d",&s1[i].sno);1,2,3
scanf("%f",&s1[i].avg);95.0,96.0,97.0
scanf("%s",s1[i].sname);aaa,bbb,ccc
}

22/12/2021
----------
STRUCTURE AND FUNCTION
----------------------
Similar to the variables of the built-in types,structure variables can also
be passed to a function


Pointer to structures--complex structures
---------------------
Pointer to structure holds the address of the entire structure

How to access the structure members?
-----------------------------------
The members of the structure can be accessed using an arrow operator (->)

Declaration
-----------
struct structname *ptr;

Eg:
--
struct stud *s;
```

Accessing
---
s->struct-member;

24/12/2021
---
int n,*p;
p=&n;

s->sno

Self-Referential Structures

The structures that have one or more pointers,which point to the same type of structure,as their member

```
struct node<.............
{                        |
<data-type> member1;     |
<data-type> member2;     |
struct node *link;........
};
```

The structures pointing to the same type of structure are called self-referential structures

Eg:
--
```
struct node
{
int n;
char ch;
struct node *link;---pointer ,the structure node is a self-referential structure with the reference "link"

}
int main()
{
struct node ob;
return 0;
}
```

Types of Self-Referential Structures
---
a)Self-Referential Structures with single link

```
10|20|  ------> 30|40|X

  ob1           ob2
```

b)Self-Referential Structures with multiple links---contain more than one self-pointers

```
 10|20|  ------> 30|40|X----->50|60|x
         <------>       <----->
  ob1           ob2
```

UNIONS

```
struct
{
int sno;
char sname[20];
float avg;
}s1;
```

A union is a special data type that allows to store different data types in the "same memory" location.

```
Syntax
------
union <union-name>
{
<data-type> member1;
<data-type> member1;
.....................
}<union variables>;
Eg
--
union stud
{
int sno;
char sname[20];
float avg;
}u1;
```

```
   structure
   ---------
    sno   sname  avg
    4     20     4------28 bytes
```

```
typedef
-------
A keyword used to give a type a new name

Eg:
--
typedef unsigned int N;

N n1,n2;

Used for built-in types and also for user-defined types


Difference between #define and #typedef
---------------------------------------
a)#define is a directive used to define the alias names for various types
typedef is used to give symbolic names to types only(built-in or user-
defined types)

b)typedef interpretation is performed by the compiler
#define statements are processed by the pre-processor

27/12/2021
----------
enum keyword
------------
enumeration---enumerated type---user defined type,which consists of integer
```

values and provides meaningful names to the values

Syntax

```
enum flag{integer_constant1,integer_constant 2,....};
            0       1     2        3      4--default values
enum fruits{Mango,Papaya,PineApple,Grapes,Jackfruit};

enum status{true,false};

enum flag <tag-name>;
enum fruits f;
```

Bit Fields
----
1 bit---0 or 1

ON/OFF--toggle

Bitwise Operators

&
|
^
~
<<
>>

Conversion--Decimal to Binary

12
  (10)

```
       2|12
       2|6-0
       2|3-0
         1-1
```

Binary to Decimal

```
    1 1 0 0
            (2)

      2^3    2^2  2^1    2^0
       1      1      0     0
```

  =8*1+4*1+2*0+1*0
  =8+4+0+0
  =12

  << >>

```
  0  1  2  3
        1  0 --2 <<
     1  0  0 --4

>>          1-- 1
```

Bit Masking

FILE HANDLING

A file is a collection of related data records

Create
Open
read
write
update/append
close

```
FILE *fp;
```

```
     Memory
     ---
     ---
```

```
if(fp==NULL)
{
printf("\n\t File doesn't exist...");
exit();
}
```

File handling enables us to create,update,read and delete the files stored on the local file system

Types of Files

Text Files
Binary Files---0 and 1

Operations performed on files
---
a) Creating a new file
b) Opening an existing file
c) Reading from the file
d) Writing to the file
e) Deleting the file
f) Append contents to to the file
g) Closing the file

Functions
---
a) fopen()--Opens new or existing file
b) fprintf()--write data into the file
c) fscanf()--reads data from the file
d) fputc()--writes a character to the file
e) fgetc()--reads a character from the file
f) fclose()--closes the file
g) fseek()--sets the file pointer to a location
h) fputw()--writes an integer to the file
i) fgetw()--reads an integer from the file
j) ftell()--returns current position
k) rewind()--sets the pointer to the beginning of the file

Creating a file

FILE <file-pointer>;
FILE *fp;

Open a file---fopen()

FILE *fp;
fp=fopen(const char *filename,const char *mode);

filename----"c://folder/filename.ext"
mode---string

Mode

Text Files
---
r--Opens a text file in read mode
w--Opens a text file in write mode
a--Opens a text file in append mode

r+--Opens the text file in read and write mode
w+--Opens the text file in read and write mode
a+--Opens the text file in read and write mode

Binary Files
---
rb--Opens the binary file in read mode
wb--Opens the binary file in write mode
ab--Opens the binary file in read append mode

rb+--Opens the binary file in read and write mode
wb+--Opens the binary file in read and write mode
ab+--Opens the binary file in read and write mode

28/12/2021

File Handling

Copy the contents of one file to another

29/12/2021

Program to count the number of characters,digits,tabs,new lines from a file

Binary Files
---
rb--Opens the binary file in read mode
wb--Opens the binary file in write mode
ab--Opens the binary file in read append mode

rb+--Opens the binary file in read and write mode
wb+--Opens the binary file in read and write mode
ab+--Opens the binary file in read and write mode

fread()
fwrite(),which store numbers in binary format,where each number occupies

the same number of bytes on the disk
FILE *fp;

//Open the binary file in the write mode
fp=fopen("filename","wb");
//write the records to the file
fwrite(&e,sizeof(e),1,fp);

//Open the binary file in the read mode
fp=fopen("filename","rb");
//Read the records from the files
while(fread((&e,sizeof(e),1,fp)==1))

30/12/2021

CommandLine Arguments
---
int main(int argc,int *argv[])
{

}

0 1 2

The argument passed to the main are called commandline arguments

argc
*argv[]--

The arguments passed from the command line are called command line arguments
int main(int argc,int *argv[])
{

}

argc---counts the number of arguments.It counts the filename as the first argument
argv[]--contains the total number of arguments