Associativity can be of two types:
① Left to Right
② Right to Left

9/1/17

Algorithm

Steps to solve a problem-
1.) Problem definition
⇒ I/p, o/p, identify requirements
2.) Analysis
Specify the operations, i.e. arithmetic and logical operations to be performed. Check memory execution time in this phase.
3.) Algorithms
It is a problem solving technique.
It is defined as step by step procedure to solve a particular problem.
It consists of english like statements

Characteristics of Algorithm

1.) Input
It may accept 0 or more inputs
2.) Output
It should produce at least one output
3.) Definiteness
Each and every instruction should be clear, precise

and there should not be any ambiguity.

4.) Finite
There should be a finite no. of steps.
Should end in a finite period of time.

5.) Effectiveness
Operations must be simple and carried out in finite time at one or more levels of complexity.

Notations -
1.) Name of algorithm
2.) Step number
3.) Explanatory comment [ ]
4.) Termination END/ STOP

Advantages of Algorithms
1.) It is simple to understand. Step by step solution of the problem.
2.) It is easy to debug.
3.) It is independent of programming language.
4.) It is compatible to computers in the sense that each step of algorithm can be easily coded into its equivalent high level language.

(#) Algorithm - Area of circle
Step 1 Read radius
Step 2 [Compute the area]
Area = 3.14 x radius x radius
Step 3 [Print the area]
Print "Area of circle =", Area
Step 4 [End of Algo]
STOP

Date:
Page No.

(#) Write an algo to perform the basic arithmetic operations such as addition, subtraction, multiplication and division.

Algorithm Basic Operations
Step 1 Read a, b
Step 2 [Addition]
add = a + b
Step 3 [Subtraction]
Sub = a - b
Step 4 [Multiplication]
mul = a x b
Step 5 [Division]
div = a / b
Step 6 [Print results]
Print add, sub, mul, div
Step 7 STOP

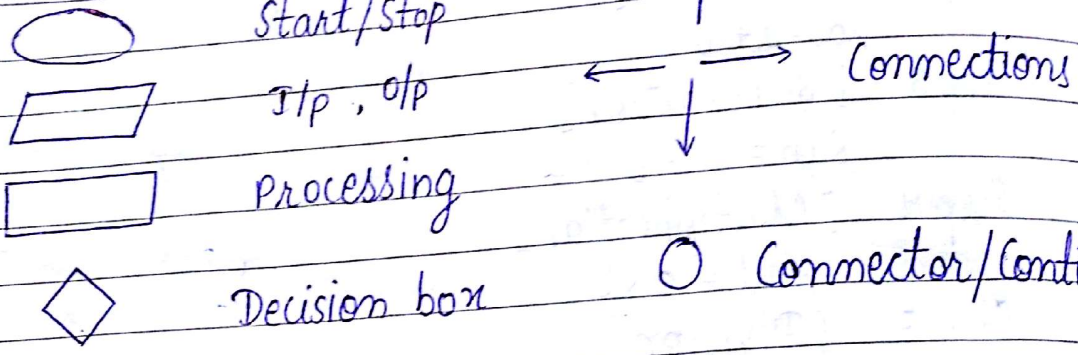
(#) Algo to find largest of three numbers
Step 1 Read a, b, c
Step 2 If a > b then go to step 3 else go to step 4
Step 3 If a > c, then print a, "is largest" else print c "is largest"
Step 4 If b > c then print "b is largest" else print "c is largest"

10/01/17

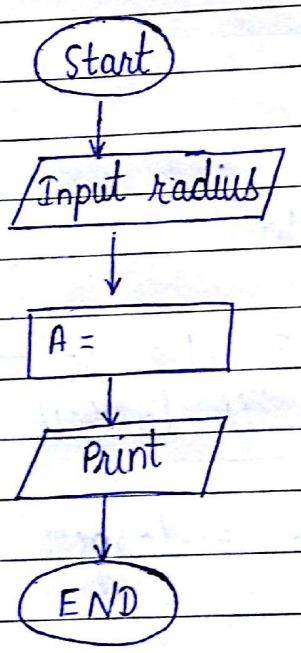
Flowchart

Pictorial representation of sequence of steps

Read top to bottom, right to left



Area of circle



Purpose of Flowchart

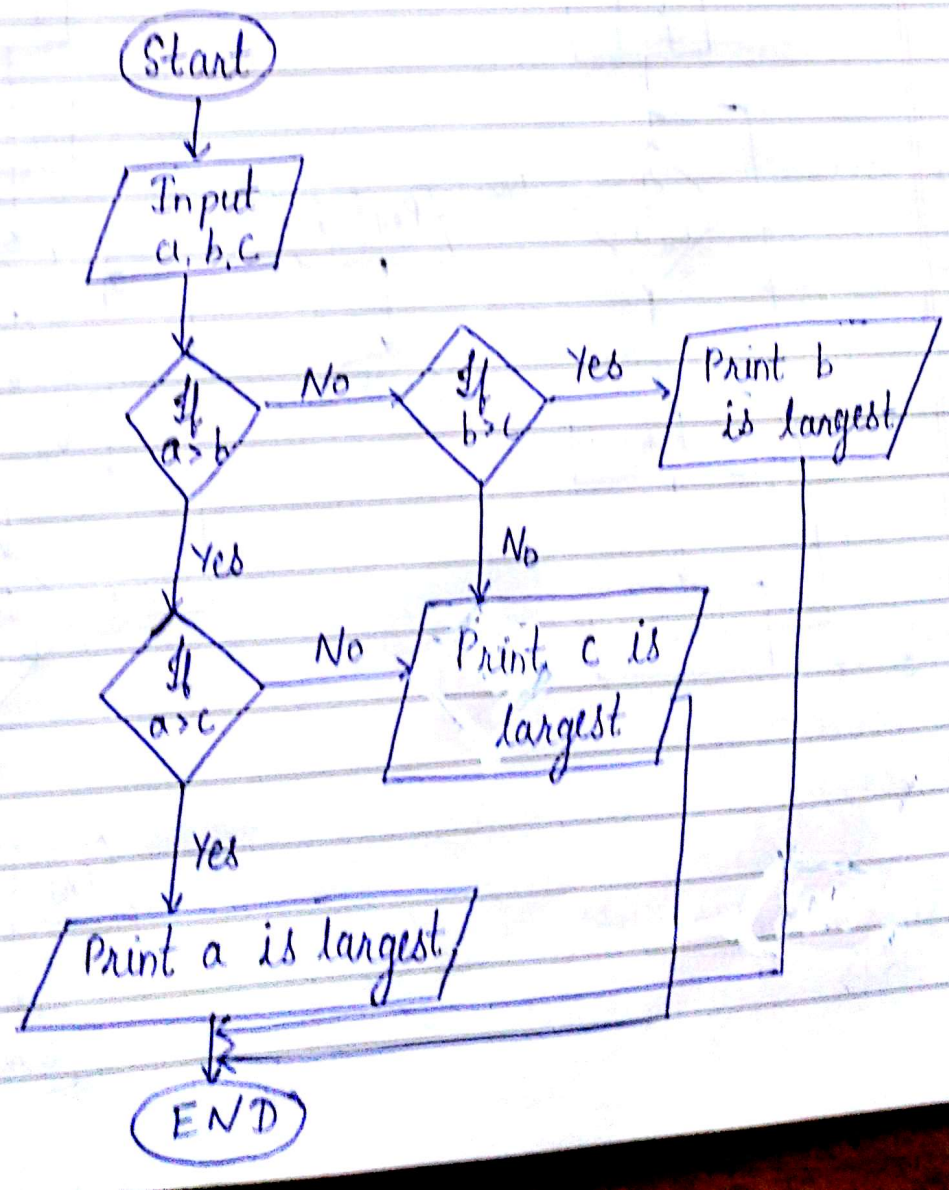
1.) Provides better communication
2.) Provides an overview of the complete problem
3) Helps in algorithm design / programming
4.) It is easy to check the program logic
5) Helps in coding

3) Modification becomes easy.
4) Better documentation is provided.

Three types of flowchart
1) System flowchart
2) Modular program flowchart
3) Detailed flowchart

Two levels of flowchart
1) Macro level flowchart - Shows main segment of program & less detail
2) Micro level flowchart - Shows more detail of any part of the program

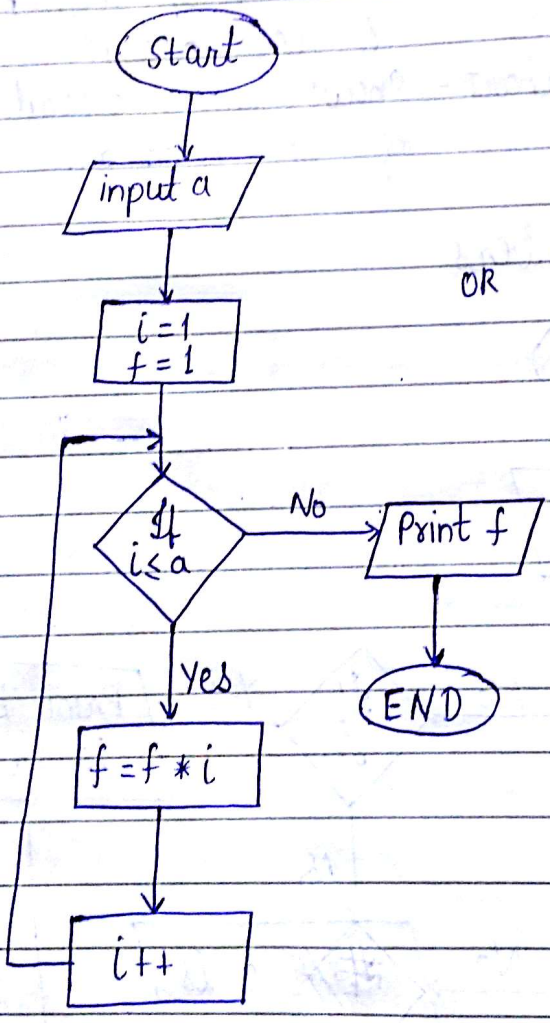
Largest of 3 numbers



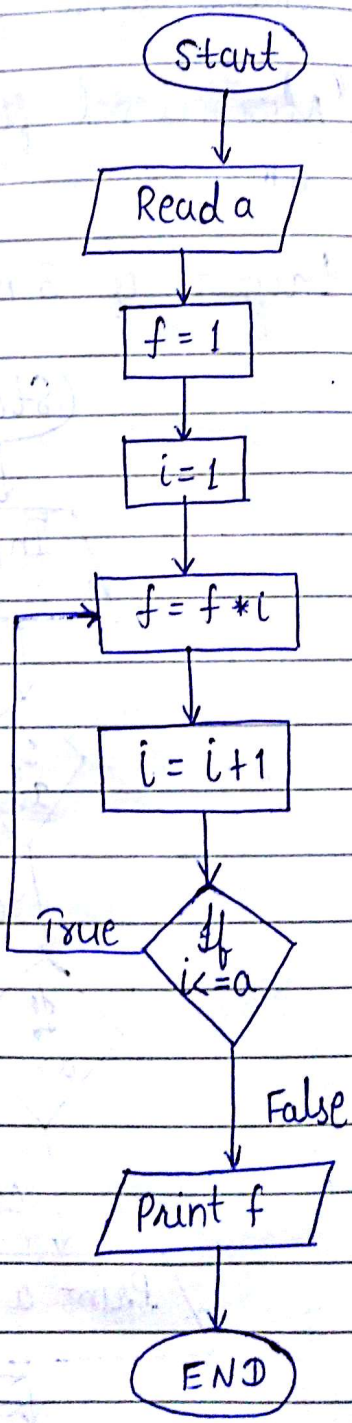
13/1/17

Compute factorial of a given number

Step 1 Start
Step 2 Read number
Step 3 [Computing Factorial]
i = 1, f = 0
while i ≤ a
f = f × i
Step 4 Print f
Step 5 End



OR



13/1/17

Date:
Page No.

7

Assembler, Compiler and Interpreter

Assembler is a program which converts assembly language to machine language

A computer can directly only execute machine language hence an assembly language program must be converted to its equivalent machine language program before A can be executed on the program. This translation is done with the help of translator program called assembler.

Compiler is a translator that translates source code (user writer program) to object code (machine language program)

Interpreter translates one line at a time.

An interpreter is a translating program for the purpose of converting HLL to machine language. Interpreter interpretes one line of HLL at a time After translation the statement is executed immediately.

This is in contrast with compiler which translates entire source program to object program. This object code is saved permanently for future use and used every time the program is to be executed. However in the case of interpreter the object code is not stored because the translation & execution takes place alternately. So if a statement is to be used again then it must

be interpreted again.

The advantage of an interpreter over a ~~complex~~ compiler is fast response. Moreover a compiler is complex program compared to interpreter. Interpreters are easy to write & does not require large memory space.

Interpretation is a time consuming process b'coz each statement must be translated every time it is executed from the source program. Thus a compiled machine language programs runs faster than an interpreted one.

Linker

In HLL some built-in header files or libraries are stored. These libraries are predefined and these contain basic fns which are essential for executing a program. These fns are linked to the libs by a program called linker.

If a linker does not find lib of a fn then it ~~inte~~ informs the compiler and the compiler generates an error.

Loader

A loader is a program that loads machine codes of a program into system memory. In computing a loader is a part of the OS that is responsible for loading ~~programmes~~ programs.

It is one of the essential stages in the process of starting a program because it places into the memory for execution.

Loading of a program involves reading the contents of executable program in memory. Once loading is complete the OS starts the program by passing control to the loaded program code. All operating systems that support program loading have loaders. In many OS the loader resides in the memory.

11/10/01/17

Compilation & Execution Process of C

The compilation and execution process of c include various steps:

① Pre-processing
Using a pre-processor program to convert c source code in expanded source code
"# include" &
"# define"
statements will be replaced & source code in this step.

② Compilation
Using a compiler program to convert c expanded source code to assembly source code

③ Assembly
Using a assembler program to convert assembly source code to object code.

④ Linking
Using a linker program to convert object code to executable code. Multiple units of object codes are linked together in this step.

⑤ Loading
Using a loader program to load the executable code to CPU for execution.

Here are examples of commonly used program for different compilation and execution on Linux system

hello.c - hello.i
Pre processor pre processing hello.c & saving o/p to hello.i

hello.i - hello.s
Compiler compile hello.i & saving o/p to hello.s

hello.s - hello.o
Assembler assembling hello.s & saving o/p to hello.o

hello.o - hello
Linker linking hello.o & saving o/p to hello.

Load hello
Loader loading hello & running hello.

C - Dennis Richies in 1972 at Bell Laboratories

Importance of C:
Reliable, simple & easy to use
Unix, Windows, Linux are programmed in C.

Types of C constants
Primary constant - integer (int), real, character
Secondary constant - arrays, pointers, structures, union, enum, etc.

Rules for constructing integer constant.
1) Space, comma or special characters are not allowed.
2) If there is no sign, the integer is considered +ve

Rules for constructing real constant:
1) Real const. ~~be~~ must have one digit
2) Can be +ve or -ve
3) No comma or blank spaces are allowed

Following rules must be observed while using exponential form
1) The ~~e~~ mantissa &
2) The mantissa part ~~must~~ may have +ve or -ve sign.
3) Exponent must have at least one digit
4) Range -3.4e38 to 3.4e38

Rules for constructing character constant:
1) A character const. is single alphabet or a single digit or a single special symbol b/w single inverted comma.

Types of C variable

Particular type of variable can hold particular type of constant.
1-31 alphabets, underscores ~~or~~ can be used
No commas or blanks are allowed within variable names.
No special symbol other than underscore can be used within variable names.

printf(); scanf();
↑ → std. i/p o/p header file

```
# include <stdio.h>  → console i/p o/p
  include <conio.h>  → getch();
  void main()
  { printf("Hello World");
    getch();
  }
```

Program is a set of instructions
3 types of instructions:

1.) Type Declaration Instruction
2.) Arithmetic Instruction - This inst. is used to perform arithmetic operation on constants & variable.
3.) Control Instruction - This inst. is used to control the sequence of execution of various statements.

1.) Type Declaration Instruction

```
int a, b;
float x, y;
char w, r;

int i = 5, j = 10;
float t = 5.6, z = 6.25;
float a = 1.5, b = 1.99 + 2.4 x 1.44;

int a = b = c = 10;          (invalid)
float a = 1.5, b = a + 3.1;
```

Arithmetic Operation

```
int a;
float b, c, d, e, f, g;
a = 3200,
b = 0.0056;
d = e x f / g + 3.2 x 2/5;
```

C arithmetic operator statements can be of 3 types:

1) ~~Arithmetic~~ Mixed mode arithmetic statement (some are integer some are real)
2) Integer mode arithmetic statement
   int a, b, ~~float c~~; c = a + b x 2 + 5;
3) Real mode arithmetic statement
   float p, q, r
   r = p/q x 5.25 + 1.628;

(1) Eg. int a, b;
   float c;
   c = a + b x 2.5 / 6.143 + 6 + 2;

a % b
8 % 2 = 0
-6 % 4 = -2
6 % -4 = 2

Char a, b, c;
a = 'F'; 47
b = 'T'; 84
d = 't'; 131
c = a + b

printf ("%c", c);
$

No operator is assumed to be present

b^8 → pow(b, 8)
Header file math.h should be included

Integer & float conversions

- $\frac{5}{2} = 2$

5.0/2 = 2.50000
5/2.0 = 2.50000
5.0/2.0 = 2.5000

```
int i;
float b;

i = 3.5;        Computer will take i as 3 & b as 30.0
b = 30;

float a,b,c; int s;
s = axbxc/100 + 32/4-3 x 1.1;
s will be integer
```

Hierarchy of operation

Priority order
1. Division, Multiplication, Modulus
2. Addition, Subtraction
3. Equal to

```
i = 2x3/4 + 4/4 + 8-2+5/8
    6/4 + 4/4 + 8-2+5/8
    1 + 1 + 8-2+0
    2+8-2
    10-2 = 8
```

9/1/17

Date:
Page No.

Associativity of Operators

```
i = 3/2 * 5
  = 1 x 5
  = 5
```

Associativity can be of two types:

1. Left to Right
2. Right to Left

24/01/17

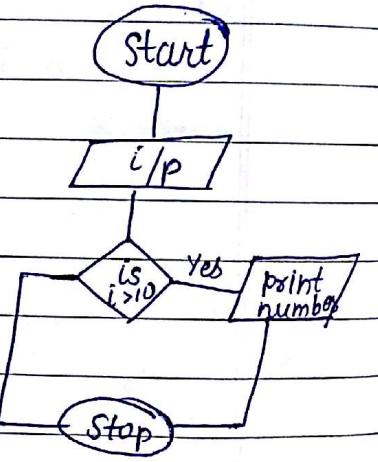
(#) If statement, else statement, nested if statement & use of logical operators

```
if (condition is true)
execute this statement;
```

x == y Comparison operator
x > y
x < y
x >= y
x <= y
x != y

x = y assignment operator

```
void main ()
{ int i;
scanf ("%d", &i);
  if (i>10)
printf("This is a number");
getch();
```



Q While purchasing certain items, 10% discount is offered on quantity greater than 1000 & price are input, write a program to calculate total expenses.

Ans
```
void main()
{ int qty, dis=0
  float rate, tot;
printf("Enter quantity & rate");
scanf("%d %f", &qty, &rate);
      if (qty>1000)
          dis = 10;
tot = rate x qty - (rate x qty x dis/100);
  printf ("%f", tot);
  getch();
  }
```

Date:
Page No.

Multiple statements within if

The current year and the year the employee joined are entered through keyboard.

Ans
```
void main ()
{ int bonus, cy, yoj, yos;
printf ("Enter current year and year of joining");
scanf ("%d %d", &cy, &yoj);
    yos = cy - yoj;
      if (yos > 3)
{ bonus = 2500;
printf ("%d", bonus);
 }
 getch ();
 }
```

Q If in a company employee is paid basic salary < 1500 then HRA = 10% of basic salary & DA = 90%. If salary > 1500 HRA = 500 DA = 90% of salary. ~~First~~ If the employee salary is i/p through keyboard. Write program for finding gross salary.

Ans
```
void main ()
{ float bs, da, hra, gs;
print ("Enter the basic salary");
scanf ("%f", &bs);
if (bs < 1500)
{ hra = bs x 10/100;
   da = bs x 90/100;
}
else
```

If block

```
else
{ hra = 500;                    ] Else block
   da = bs x 98/100; }
 gs = bs + hra + da;
printf ("%f", gs);
 getch ();
  }
```

Nested if & Nested else

```
void main ()
{ int i;
printf ("Enter the number");
scanf ("%d", &i);
 if (i==1)
{ printf ("This is number 1");
else
 { if (i=2)
    printf ("This is number 2");
   else
    printf ("This ~~in~~ is ~~number~~ out of scope");
 }
 getch ();
  }
```

Q. Marks obt. by students in 5 diff. subject are input through the keyboard. The student gets the division acc. to following table:

>=60%	1st div
50 - 59	2nd div.
40 - 49	3rd div
< 40%	fail

Date
Page No.

```
void main()
{ float a,b,c,d,e, per;
printf("Enter the marks");
scanf("%f,%f %f %f %f",&a,&b,&c,&d,&e);
per = a+b+c+d+e x 100 / 200
printf("%f", per);
  if (per >=60)
printf("First div");
   if (per >=50 && per <=59)
 printf("Second div");
 if (per >= 40 && per <=49)
 printf("Third div");
 if (per < 40)
 printf("Fail");
 getch();
  }
```

Priority

1. Not - logical not !
2. Multiplication, division, arithmetic operator, modulus
3. Add^n, Sub^n
4. <, >, >=, <=, ==, !=
5. And operator
6. or operator
7. Assignment operator

```
y = (x>5 ? 3:4);
```

The cond^nal operator can be nested.

Diff b/w conditional operator & if statement

```
int big a, b, c;
big = (a>b ? (a>c ? 3:4):(b>c?6:8));
```

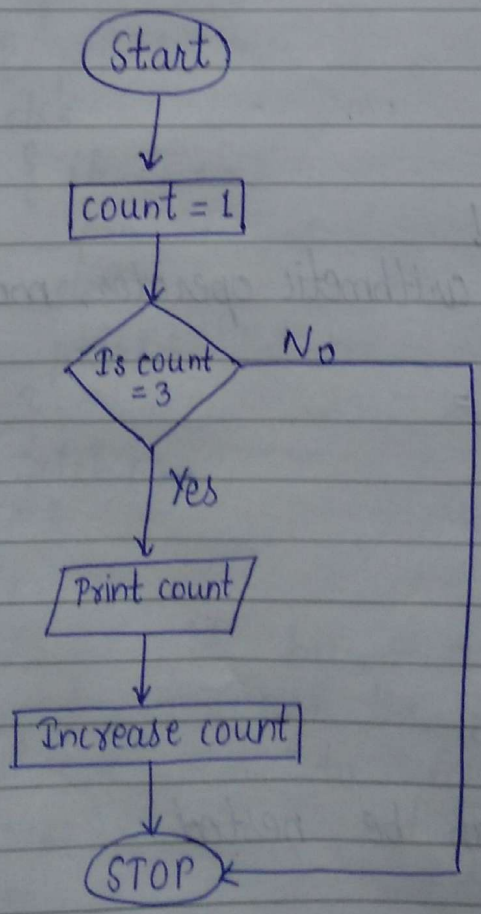
? : Cond^nal operators

It is not necessary that cond^nal operations can used in arithmetic operations

```
int i;
scanf("%d", &i);
(i==1 ? printf("Amit"): printf("ABC"));
```

Loops

For and while loop



Neelgagan
Date:
Page No.

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

Simple interest programming ~~to~~ in 'For' loop

```
for (i=1; i<=3; i++)
{ printf ("Enter value of p, t, r");
   scanf ("%d %d %f", &p &t &r);
   si = (p*t*r)/100;
  printf ("%f", si);
   getch ();
   }
```

Multiple initialization in 'For' loop

```
For (i=1, j=2; i<=3; i++)
```

Break statement

```
void main ()
```

```
}
getch();
}
```

To find simple interest of employees

```
# include <stdio.h>
# include <conio.h>
 void main()
{
 int p,t,i;
 float r, si;
  i=1
 while (i<=3)
 { printf("Enter value of p.t.r");
 scanf("%d %d %f",&p, &t, &t);
 Si = (p*t*r)/100;
printf("%f", Si);
      i = i++; }
   getch();
 }
```

General Syntax

Initialize loop counter

while (test loop counter using a condition)

```
 { statements;
      }
```

statements can also be given as

while ((i<=10) && (J>=15) :: (K==15))

break ~~command~~ statement is associated with if. By break statement we come out of the loop.

Date:
Page No.

30/1/17.

Multiple Initialization in 'For' loop

```
For (i=1, j=2; i<=3; i++)

void main()
{
 int num, i;
printf ("Enter a number");
scanf ("%d", &num);
i=2;
while (i<=num-1)
 { if (num%i==0)
{ printf (num%i==0) ("Not a prime number");
 break;
 } i++;
}
 if (i==num)
printf ("Prime number");
 getch(); }
```

Eg. for continue statement

```
 { int i, j;
  for (i=1; i<=2; i++);
{ for (j=1; j<=2; j++);
 { if (i==j);
  continue;
printf ("%d %d", i, j);
}
}
getch();
 }
```

(#) Do while loop

```
do
 { this;
  and this;
  and this;
} while (this condition is true);

void main ( )
{ char another;
   int num;
 do
{ printf ("Enter a number");
  scanf ("%d", & num);
printf ("Square of %d is %d" num, num * num);
printf (" want to enter another number y/n");
fflush (stdin);
 scanf ("%c", & another);
} while (another == 'y');
getch ( );
}
```

(#) Switch statement

```
Switch (integer expression)
{ case constant 1;
        do this;
 case constant 2;
        do this;
 case constant 3;
        do this;
      default;
   do this;
  }
```

Date:
Page No.

→
```
void main ()
{ int i=2;
 switch (i)
 { case 1 :
printf ("I am case 1");
  break;
case 2 :
printf ("I am case 2");
 break;
case 3:
printf ("I am case 3");
break;
case default :
 printf ("I am default");
} getch();
 }
```

→
```
void main ()
{ char c='x';
switch (c)
{ case 'v';
printf ("I am case 1");
  break;
case 'a';
printf ("I am case 2");
break;
case 'x';
printf ("I am case 3");
break;
default:
printf ("I am default");
} getch();
 }
```

→
```
void main ()
{ char c;
printf ("Enter any character");
Scanf &("%c," &c);
Switch(c)
{ printf("Welcome");   → This statement will not be executed
 case 'v':
 case 'V':
 printf ("I am case 1");
 break;
 case 'r':
 Case 'R':
 printf ("I am case 2");
 break;
case 'x':
 case 'X':
 print f ("I am case 3");
break;
default:
 printf("I am default");
} getch;
  }
```

→ Switch is the replacement of 'if' statement.
→ Disadvantage
We cannot write statement with case

```
case 3+7;          (Constants are used in 'case's' case)
switch (i+j*k)
switch (23+45 %4 * k)
switch (a<4 && b>7)
```

float is not used in switch

Neelgagan
Date:
Page No. 28

→ case's can never have variable expression.
→ Multiple statements cannot have same values

(#) Goto keyword

```
void main ()
{ int goals;
printf ("Enter the number of goals scored against India");
scanf ("%d", &goals);
if (goals <= 5)
  goto x;
else
{ printf ( )
  printf ( )
  printf ( )
}
x: printf ( )
```

31/01/17

int → 2 bytes of data
16 bit
$0 - 2^{15}$ - Data range 32768 - +32767
short int i
short i long i
long int takes 4 bytes of data

By default char, float, int are signed

shorts are ~~ver~~ never bigger than ints
~~lon~~ ints are never bigger than longs.

(#) void main ()
{
Char ch = 291;
printf ("%d %c", ch, ch);
getch();
}

Not possible. It can take from cannot take value -0 to 255 more than 127 if not unsigned
128 - +127

(#) void main ()
{ char ch;
for (ch=0; ch<=255; ch++)
printf ("%d %c", ch, ch);
getch();
}

Goes into infinite loop
It'll give values upto 127 then random

Corr. void main ()
{ unsigned char ch;
for (ch=0; ch<=254; ch++)
printf ("%d %c", ch, ch);
getch ();
}

Ans

Range of short double -1.7e308 ~~+3.4e~~ 1.7e308
" " long " -

Short - %d
int - %d
long - %ld
short unsigned int - %u

Signed char		%c
Unsigned char	0-255	%c
Short signed ~~is~~ int		%d
long signed int		%ld
long unsigned int		%lu
long double		- %Lf

3/2/17

① Q. What is the difference between break and continue statements?

② Q. Write the difference b/w while and do while statement?

③ Q. Define switch case statement with example.

④ Q. What will happen if we don't write break statement in switch case statement? Explain with example.

Ans ① The break statement is used to come out of the loop while the continue statement is used to go back to the loop.

② In case of do while statement first the syntaxc written after do is performed then the while statement is executed but it is not the case in using while statement.

③ The switch case statement is used to ~~the~~ jump to the case we want to be executed.

Eg.
```
# include <stdio.h>
# include <conio.h>
{
```

```
i = 2;
Switch (i);
Case 1:
printf ("I am case 1");
~~case 2~~ Break;
Case 2:
printf ("I am case 2");
Break;
Default:
printf ("I am default");
getch();
}
```

In this ~~program~~ case, the program will jump to case 2 directly rather than executing 1.

④ If we don't write break statement in switch case statement then all the syntax written after the case which is considered will be executed.

Eg:

```
{ # include <stdio.h>
  # include <conio.h>
{
  i = 2
Switch (i);
Case 1:
printf ("I am case 1");
~~Break~~;
Case 2:
printf("I am case 2");
Case 3:
printf("I am case 3");
```

```
default:
printf("I am default");
getch();
}
```

In this program, after the execution of case 2, case 3, and default will also be executed because break statement is not used.

6/9/17

Storage Classes in C

A Storage classes ~~in~~ C tells us:

1.) where the variable would be stored
2.) What will be the initial value of the factorial if initial value is not specifically defined then default value is set.
3.) what is the scope of the variable:
4.) in which fn the value of the variable would be available
4.) what is the life of the variable i.e. how long the variable would exist.

(1) Automatic Storage Class
(2) Registered " "
(3) Static " "
(4) External " "

(1) It will be stored in memory
(2) Default initial value: Garbage
(3) Local to the block in which the variable is defined.
(4) Till the control remains within the block in which the variable is defined.

} Automatic storage class

```
void main ()
{ auto int i,j;
  printf ("%d %d", i,j);
getch ();
}
```

It will give different values for i & j each time we run the program.

```
void main()
{   auto int i, i=1;
    { auto int i=2;
      { auto int i=3;
      printf("%d", i);
    }
    printf("%d", i);
    }
    printf("%d", i);
    getch();
    }
```

Output
3, 2, 1

First the internal block is executed.

(#) Registered Storage Class

→ Storage - CPU register
→ Garbage initial value
→ same as auto (3)
→ same as auto (4)

(#) Static Storage Class

→ Same as auto (1) ~~Value of~~
→ Default value 0
→ Same as auto (1)
→ ~~Same as auto (1)~~ Value of the variable persists b/w diff. fn calls

(#)
```
void main()
{ increment();
  increment();
  increment();
getch();
```

```
}  void increment ()
{
    static int i=1;
    printf ("%d", i);
     i = i+1;
}
```

O/P ⇒ 1, 2, 3

In place of static, if auto is used then
O/P ⇒ 1, 1, 1

(#) External Storage Class

→ Storage : Memory
→ Default value = 0
→ Global
→ As long as the program execution does not come to an end

(#)
```
int i;
void increment ();
void decrement ();
void main ()
{ printf ("i=%d", i);
increment ();
increment ();
decrement ();
decrement ();
getch ();
}
void increment ();
{ i = i+1;
printf ("i=%d", i);
}
void decrement ();
{ i = i-1;
```

O/P
i=0
i=1
i=2
i=1
i=0

OR !!
AND &&

Date:
Page No.

```
printf ("i=%d", i);
}
```

(#)
```
int x=21;
void main()
{ extern int y;
  printf("%d %d", x, y);
  getch();
}
  int y=31;
```

O/p 21, 31

If extern is not used it will show error since y is defined afterwards

The use of extern to mean that y is defined elsewhere

C preprocessor

1. Macro expansion
2. File inclusion
3. Compilation
4. Miscellaneous directives

} Preprocessor directive

Macro expansion

Define - Macro expansion

```
# include<stdio.h>
# include<conio.h>
# define PI 3.14.....
  void main
{ int r;
  float Area; scanf("%d", &r);
Area = PI * r * r;
```

Here
PI - Macro Template
3.14.. - Macro Expansion

```
printf ("%f." area);
getch ( );
 }
```

(#)

```
# #include <stdio.h>
 #include <conio.h>
# define UPPER  25
 void main ( )
{   int  r;
 for (r=1; r<=UPPER . r++)
 printf ("%d." r);
 getch ( );
  }
```

Here
UPPER - Macro Template
₹ 25 - Macro Expansion

(#)

```
# include <stdio.h>
# include <conio.h>
# define ARRANGE (a>25 AND a<50)
 void main ( )
 {     int  a= 30;
       if (ARRANGE)
       printf (" within range");
 else
   printf ("Out of range");
 getch ( );
  }
```

#define AND &&

(#)

```
# include <stdio.h>
# define FOUND printf("The signature found");
 void main ( )
{ char signature;
   if (signature == 'Y')
   FOUND
```

Date:
Page No.

```
else
 printf ("Not found");
getch();
}
```

(#)
```
#include <stdio.h>
#define AREA(x) (3.14*x*x)
 void main()
{ float r1=6.25, r2=2.5, a;
  a = AREA(r1);
printf ("Area of circle = %f," a);
 a = AREA(r2);
 printf ("Area of circle = %f," a);
 getch();
 }
```

Argument can be passed in macro

(#)
```
# include <stdio.h>
#include <conio.h>
# define ISDIGIT(y) (y>=48 && y<=57)
 void main()
{ char ch;
printf ("Enter any digit");
scanf ("%c," & ch);
if (ISDIGIT(ch));
printf ("You entered a digit");
else
printf ("illegal input");
getch();
}
```

Preprocesser takes more memory than function.

Date:
Page No. 37

(#)
```
# include <stdio.h>
# include <conio.h>
# define HLINE for(i=0; i<79; i++)\
        {printf("%c",196);\
         printf("%c",179);\
        }
```

File Inclusion

```
# include <stdio.h>
# include "abc.c"
```

1. A macro must always be written in capital letter. T
2) A macro should always be accomodated in a single line. F
3) After preprocessing when the program is sent for compilation the macros are removed from the expanded source code. F
4) Macros with arguments are not allowed. F
5) Nested macros are allowed.
6) In a macro call the control is passed to the macro.

7) How many # include directives in a given program?

8) What is the diff. b/w the following two # include directives

Date:
Page No.

9.) A header file is
a) A file that contains std. library fn
b) A file that contains definition & macros
c) A file that contains user defined fns
d) A file that is present in current working directory.

10) What is preprocessor directive?
a) A msg from compiler to the programmer
b) " " " " " " linker
c) " " " " " " preprocessor
d) " " " programmer " " microprocessor

①
```
{ float a = 13.5;
  double b = 3.5;
printf ("%f %lf", a, b)
```

②
```
int i=0;
void val();
{ printf
```

```
int i=0;
void val ();
void main ()
{ printf ("main's i= %d\n", i);
i++
val ();
printf ("main's i = %d \n", i);
val ();
getch ();
}
void val ()
} { i = 100;
printf ("val's = i = %d\n", i);
i++;
}
```

T/F

(1) Storage for registered storage class is allocated each time the control reaches the block in which it is present.

(2) An extern storage class variable is not available to the fns preee preceeds its defn unless the variable is explicitly declared in these fns

(3) The value of an automatic storage class variable persists b/w various fn invocations. ✗

(4) If the CPU registers are not available, the register storage class variable are treated as static storage class variables

(5) The reg. storage class variable cannot hold float values.

(6) If we try to use registered storage class for a float value the compiler will flash an error msg.

(7) If the variable x is defined outside all fns & a variable x is also defined as a local variable of some fn then the global variable gets preference over local variable.

(8) The default value for automatic variable is zero.

(9) The life of static variable is till the control remains in the block it is defined.

Neelgagan
Date:
Page No.

(10) If a global variable is to be defined then the extern keyword is necessary in its defn

(11) The address of register variable is not ~~avaible~~ accessible.

(12) A variable that is defined outside ~~ow~~ all fns can also have a static storage class.

(13) One variable can have multiple storage classes.