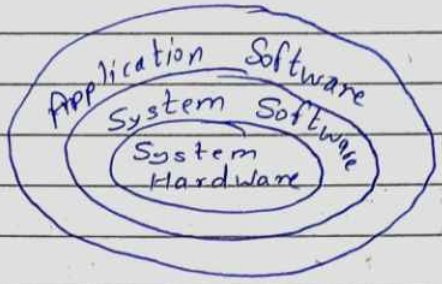
Class-Notes

13/09/23 Unit I:



Input → [System Transformation] → Output

Basic Terms: ~~Computer architecture~~

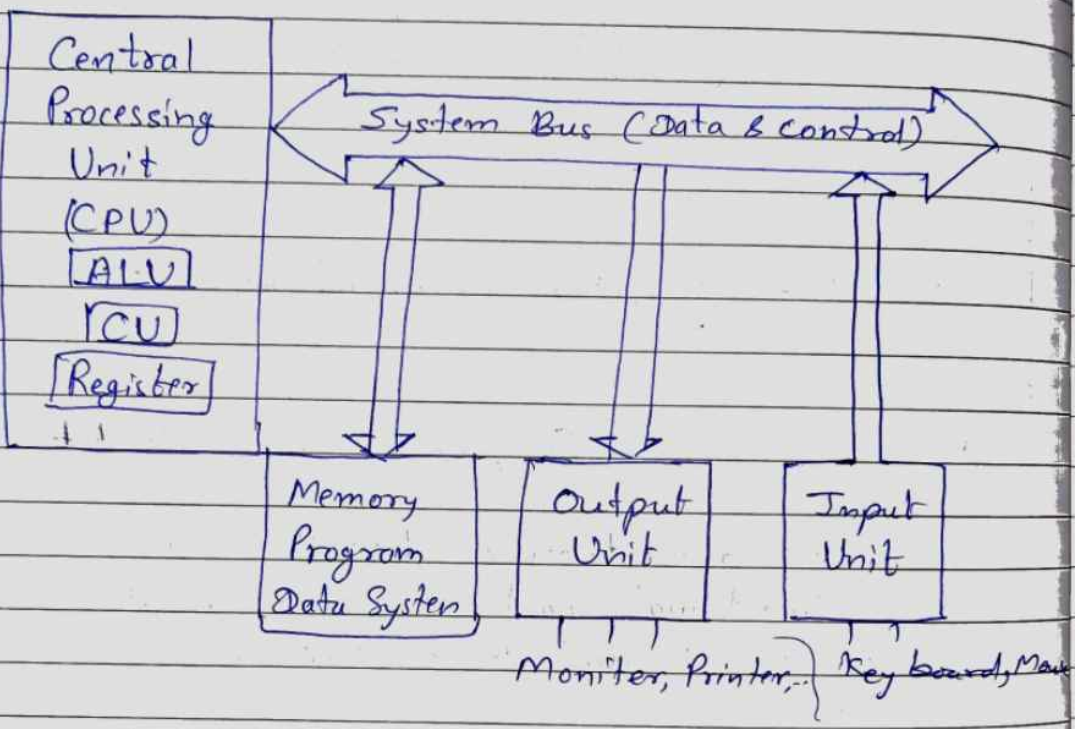
Computer architecture describes structure and basic functional part of a computer system at a logical level from the programmer's point of view.

Organization: Computer Organization describe how hardware component operate to meet the architecture

13/09/23

Organisation of Simple Computer System

Basic Organisation



A simple computer system is comprised of ~~these~~ three major component processor, memory input/output (i/o) that interact indirectly through the BUS.

System Component: Processor manupulate ~~processors, m~~ information (located in processors, memory, i/o) by executing instruction

Information in processer is held in Register

Processors are characterised by
1. Register Set
2. Instruction Set
3. Intruct mechanism

Memory hold information in cells (or locations)

1. A cell has an address and content
2. Memory support two operation read & write
3. i/o : i/o supports the information exchange between computer & connected device

Independent i/o component associated with each connected device.

Ports: exchange information between BUS & i/o component.

---

i/o : input/output
ALU : Arithmetic Logical Unit
CU : Control Unit
Register : Temporary storage

14/09/23

Number System

1. Binary
2. Decimal
3. Hexadecimal
4. Octal

→ operations

1. Interconversion
2. Binary Arithmetic

Binary information is stored in memory or processor register

Registers contain either ~~dat~~ data or control information

Data are numbers and other binary coded information

Control information is a bit or group of bits used to specify the sequence command signals

Data types found in the register of digital computers are numbers used in arithmetic computation,

Date / /

Data types found in the register of digital computers are numbers used in arithmetic computation, letters

letters → of the alphabet used in data processing and other dts

other discrete Symbol used for specific purpose

Number System

Base or Radix system uses distinct symbol for digits

① Decimal System / Base-10 System
0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Composed of 10 symbols ✓

② Binary system / Base-2 System

It consist of numerals (0, 1)

③ Hexadecimal / System / Base-16 System

It is composed of 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F (16) symbols

Octal / Base-8 System

O-1 (from Binary)

Date __/__/__

~~000, 001, ... 111.~~

000 → 0 Base - 8

001 → 1 bit → 3

010 → 2

011 → 3

100 → 4

101 → 5

110 → 6

111 → 7

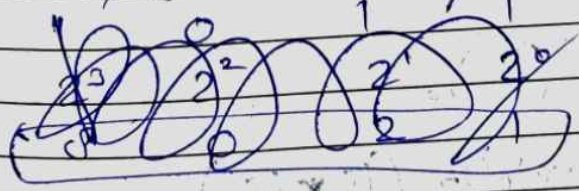
Interconversion

~~Binary to Decimal~~

BINARY to DECIMAL → ~~Base-8~~

1011 = ? 11

Binary decimal



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

1 0 1 1

$8 + 0 + 2 + 1 = 11$

1101 = 13 decimal

$1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0$

$8 + 4 + 0 + 1 = 13$

1 1 0 1

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

$8 + 4 + 0 + 1 = 13$

Page No.

DECIMAL to BINARY

Date ___/___/___

$37_{10} = (?)_2$

$37/2 = 18$
$18/2 = 9$
$9/2 = 4$
$4/2 = 2$
$2/2 = 1$
$1/2 = 0$

remainder

1
0
1
0
0
1

$37_{10} = 100101_2$

SHORTTRICK

$2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$
32 16 8 4 2 1
$32 + 4 + 1 = 37$

HEXADECIMAL to DECIMAL Conversion

$2AF_{16} = (?)_{10}$

$(2 \times 16^2) + (10 \times 16^1) + (15 \times 16^0)$

$512 + 160 + 15$

687_{10} Ans

DECIMAL to HEXADECIMAL Conversion

Date

$423_{10} = (?)_{16}$

Remainder

423/16 = 26 — 7

26/16 = 1 — 10 → A

1/16 = 0 — 1

$423_{10} = 1A7_{16}$ Ans

HEXADECIMAL to Binary Conversion

$9F2_{16} = ?_2$

8	4	2	1

9 → 1 0 0 1

F → 1 1 1 1

2 → 0 0 1 0

$1001\ 1111\ 0010_2$ Ans

BINARY to HEXADECIMAL

$0011\ 1010\ 0110_2$ → 0011 = 3, 1010 = A, 0110 = 6

$= 3A6_{16}$

Page No.

~~HW 0~~

20/09/23 Intro of Computing

Binary Arithmetic

(1) Binary Addition

~~Case~~ ~~A & B~~

Case	A+B	SUM	CARRY
1	0+0	0	0
2	0+1	1	0
3	1+0	1	0
4	1+1	0	1

16+8+2 = 26 +12

~~Decimal~~

$26 \leftarrow 0011010_2$ $\quad 26_{10}$

$12 \leftarrow 0001100_2$ $\quad 12_{10}$

0100110_2 $\quad 38_{10}$

$32 + 4 + 2 + 0$

(2) Binary Subtraction

case	A - B	Substract	BORROW
1	0 — 0	0	0
2	1 — 0	1	0
3	1 — 1	0	0
4	0 — 1	0	1

00111010_2
00011100_2
00001010_2

26
- 12
14

00111010_2
00011100_2
00001010_2

8 4 2 = 14 ✓

10 - 1 = 01 आगे जिससे 1 लिया था वो 0 हो गया अब फिर से 10 करेंगे और लास्ट वाला zero हो जायेगा।

(3) Binary Multiplication

Case	A × B	Multiplication
1	0 × 0	0
2	0 × 1	0
3	1 × 0	0
4	1 × 1	1

00111010_2
00011100_2
00001000

26
× 12
312

Date ___/___/___

```
  00110 10      26
  00011 00      12
  -------      ---
  00000000     312
   000000x
  0011010xx
  0011010xxx
  ---------
  010011000
```

Binary add^n

256 + 32 + 16 + 8 + 4 + 2 + 1

2^8
16
312

110 > 101 ⊗ → 0 (not divided)
110 < 1010 ✓ → 1 (divided)

(4) Binary division

```
        111
  000110 | 101010
           110
           ---
           001
           110
           ---
           110
           110
           ---
           000
```

```
      7
  6 | 42
      42
      --
      00
```

Program Translator

programme translator

is a computer programme that perform the translation of a programme written in a given programming language into functionally equivalent programme in a diff. computer language without loosing the functional or lexical str. of the original code.

① Compiler

a compiler is a computer programme that transforms source code into the another computer language that is the target language know as object code. The most common reason for converting a source code is to create an executable programme.

Compiler is performed following operation -

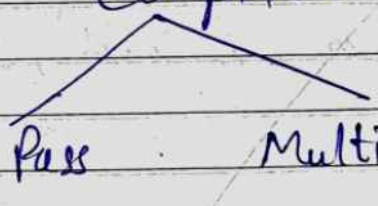
1) lexical analysis
2) preprocessing
3) parsing
4) semantic analysis
5) code generation / code optimisation

Compiler : a compiler is a comp. prog. that transform the source code into the another comp. lang. that is the target lang. known as object code

The most common reason for converting a source code is to create an executable prog.

The compiler is perform following operation
1. Lexical Analysis
2. Preprocessing
3. Parsing
4. Semantic analysis
5. Code generation / code optimization

Types of Compliter
① single pass Comptr
② multi pass comptr



Single Pas Cm : The ability to compile in SPC has benefite because it simplifies the job of writing a compiler and SPC generally perform compilations faster than MPC but it a disadvantage it does not perform code optimization needed to generate high quality code

Multi Pass Compiler: ~~MPC~~ While the typical MPC output machine

Date / /

~~code from it final pass there are~~ several other types a source to source compiler is a type of compiler that takes HLL as its input and output HLL

HLL: High level language

Intro of computing

Advantage of Compiler:

(1) Source code is not included therefore compiler code is more secure than interprate code

Task 2

(2) ~~Tend~~ to produce faster code than interpreter code

(3) Produce an executable file and therefore a program can be run without need of the source code

Disadvantage

(1) object code needs to be produced before ~~that executable file this can be a slow~~ process.

(2) The source code must be 100% correct for the executable file to be produced

(3) Interpreter: An Int is a comp program that directly executes or we can say an Int is a program that reads in as input a source program along with data and translates instruction by instruction

Adv

(1) Easier to debug than a compiler

(2) Easier to create multiplatform code as each diff would have an Intr to run the same code

(3) Useful for prototyping softwares and testing basic program logic

Disadvantages

(1) Source code is required for the prog to be executed and this source code can be red making insecure.

(2) Interpreters are generally slower than compilers due to the per line translation method.

(10) Assembler is a program that translate assembly lang. into machine code.

Date __/__/__

It create object code by translating combination of mnemonics ~~by translating~~ comb and syntax ~~and~~ for operations and addressing modes into there numerical equivalent

Advantage

(1) Very fast in translating assembly lang to machine code as 1-to-1 relationship

(2) Assembly code is very efficient bcz it is low level lang.

(18)

Disadv

(1) As lang is written for certain instruction set or processor

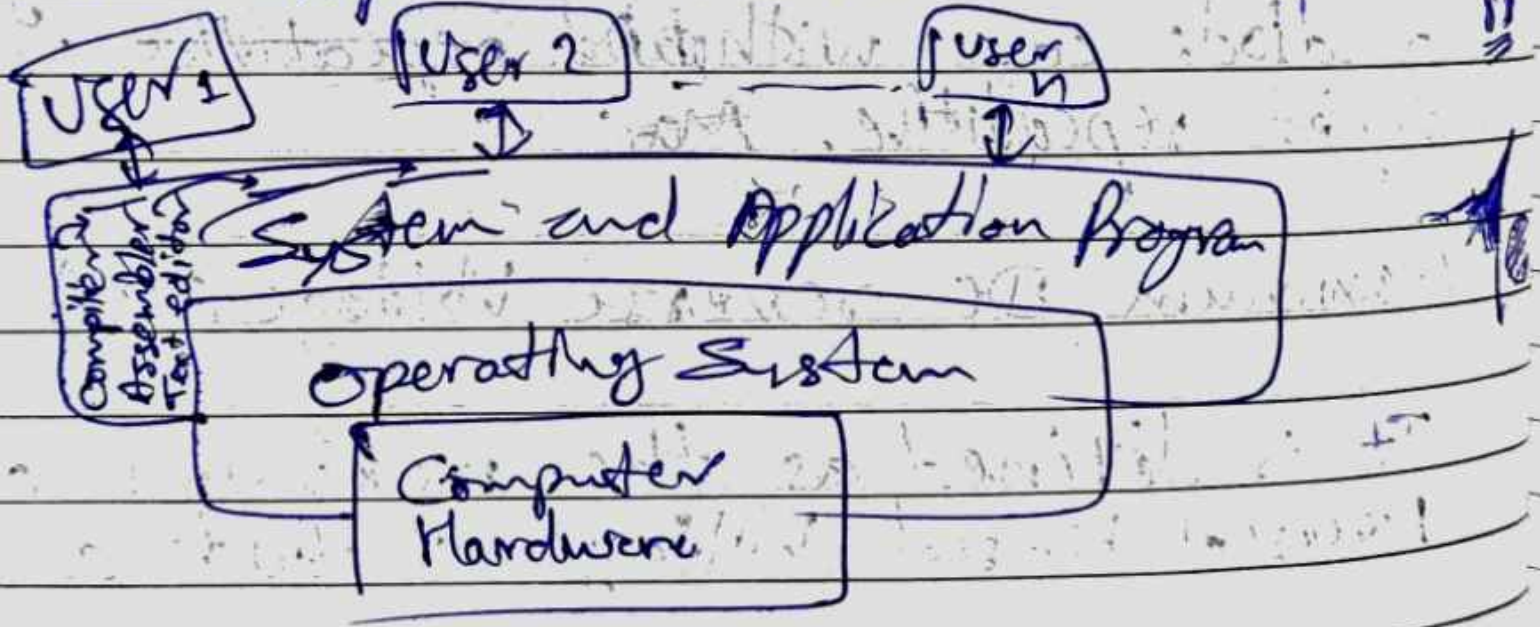
(2) Lots of ass code is needed to do relative simple task and complex program require lots of programming time

printf	comp	interp	assembly
def			
ifo			
etc			
etc			

27/09/23 Computer

Operating System (OS)

OS is a system software which act as an interface btw the end user and computer hardware

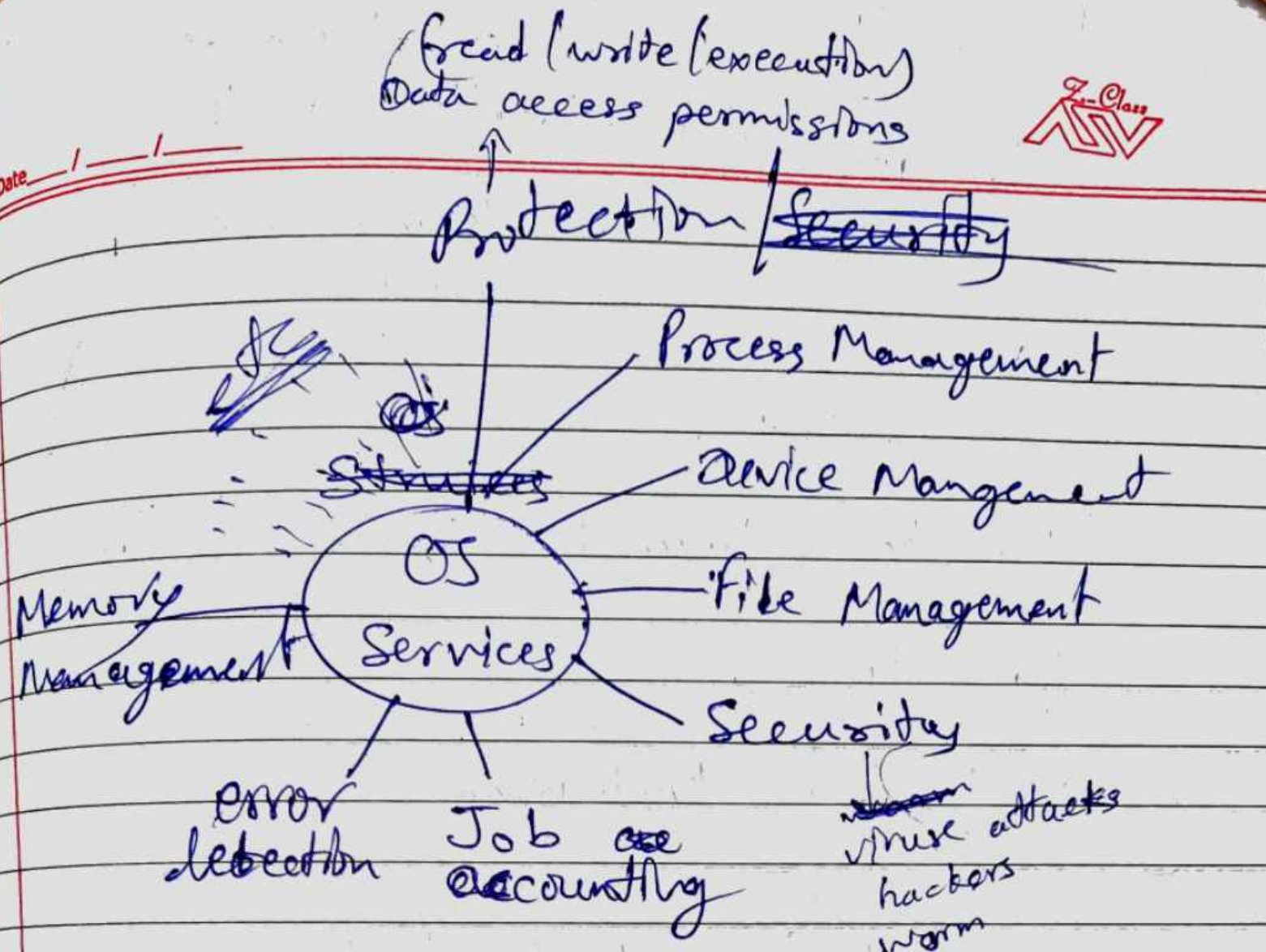


goals

↳ Resource allocation

↳ Control program → convenient & environment / user friendly

Date __/__/__



Types of OS — etc such bunch banake 0 is hole/zero, 0 & 1

→ Job card / Punch card

→ Batch OS (Coldest but used in whether forecast)
→ Multiprogram OS
→ Time sharing OS
→ Real time OS → Soft : Game; Hard : Aeroplane ATC (Air traffic control)
→ Distributal OS → Remotely controlled, cloud computing

Comp. → Unit II C. Prog. Lang.

Date __/__/__

Sample Program in C
C- character Set
Identifier & keywords
Data-Type

① Why we need computer programming language

—

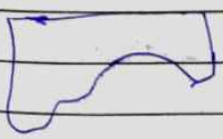
Flow chart:- FC is a type of diagram that represent a work flow and it can also be defined as a diagramatical representation of an algorithm, a step by step approach to solve the task

①  terminal star/stop

②  Input/output

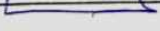
③  Process

④  Decision

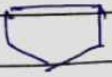
⑤  Document

⑥ → Flow

⑦  comment

⑧  stored Data

⑨ ○ connector/reference (on page)

⑩  off page connector (reference for next page connection)

Page No.

Types of flowchart

Date __/__/__

Process fc | Data fc | Business Process Modeling Diagram

Process fc: This type of flow chart shows all the activities that are involved in making a project

Data fc: It is used to analyse the data and it helps in analyzing the structured detail related to the project

BP: This fc helps to analyse the BP and also simplify the concept needed to understand the Business Activity

Adv of fc

(1) It is the most efficient way of communicating the logic of the system

(2) It acts as a blueprint during the program design

(3) It helps in the debugging process

(4) fc are good for documentation

Page No.

Disadvantage of FC

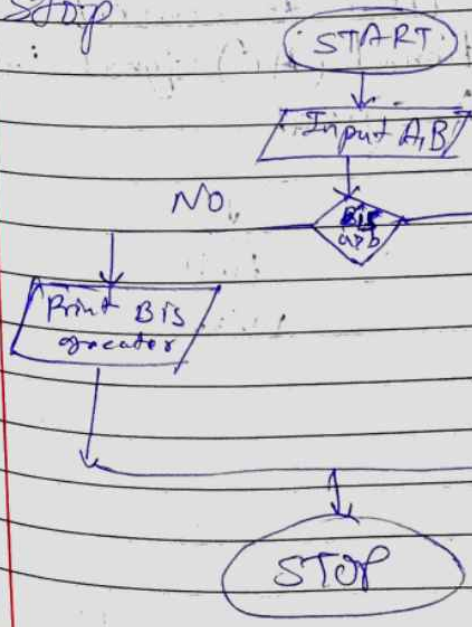
1. FC are challenging to draw for large and complex program
2. It doesn't contain the proper amt of information
3. FC are very diff. to modify

Ex.

(1) Draw a flow chart to find the greatest number among the to number

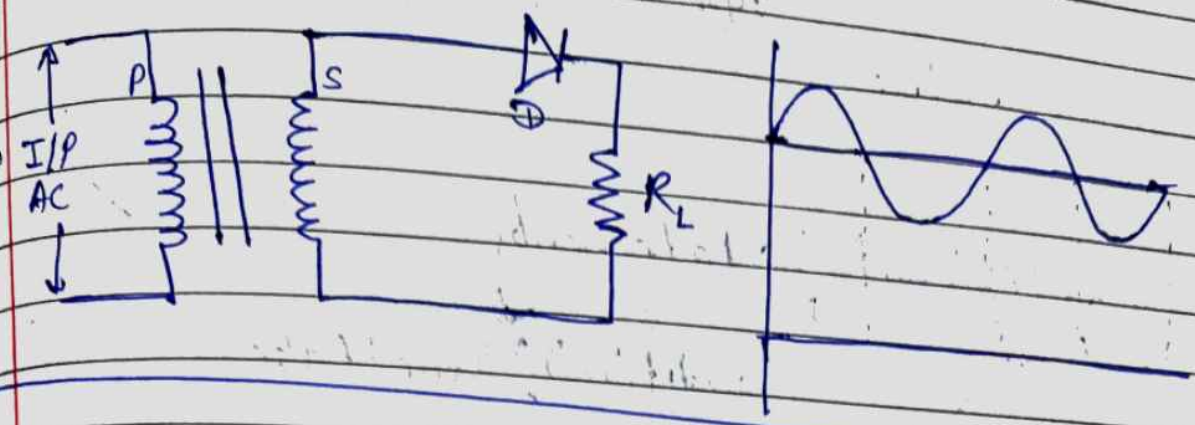
ALGORITM

1. START
2. Input 2 variable from user
3. check cond^n a > b
   go to step (4) else go to step (5)
4. Print a is greater, go to step (6)
5. Print b is greater, go to step (6)
6. Stop



Date __/__/__

Half Wave Rectr



Type Conversion

- Implicit conversion (automatic)
- Explicit conversion (manually)

```
int x = 5;
int y = 2;
int Div = 5/2;
printf("%d", Div);
```

o/p = 2; (X)

```
float DIV = (float)5/2
printf("%f", DIV)
```

```
float a = 9;
printf("%f", a);
```

o/p = 9.000

Implicit Conv^n: It is done automatically by the compiler when you assign a value of one type to another.

e.g. If we assign an int value to a float type

Date ___/___/___

Explicit Conv^n: It is done manually by placing the type in ( ) in front of the value

Conditional Statements

If, else, (condition)? True : False, Switch

If, If else, If else if, Nested if

If

Uses:

→ To specify a block of code to be executed if specified condi^n match

```
if (condition)
{
}
```

If. Else

Uses

→ Use else to specify a block of code to be exec. if the condi^n is false

```
if (cond$^n$)
{
}
```

Page No.

int x = 20
int y = 18

if (x>y)
{printf ("x is greater")
else
{printf ("b. is greater")
}

③ If else If
int time = 22
if (time<10)
{printf ("good morning");
}else if (time<20)
{printf ("good day");
}
else {printf ("good evening");
}

④ ~~5/10/23~~ ~~S.S.t.~~
9/10/23

If else
If condition)
{
}
else
{
}

```
int time = 20
if (time < 10)
{print ("Good day");
}
else
{print ("good evening");}
}
```

Page No.

short hand if else

Date (Ternary operator)

~~Symbols~~ Syntax

variable = (condition) ?
expression Time : expression false;

```
int time = 20;
(time < 20) ? printf ("good day");
printf ("good evening")
```

Switch Case

Syntax

```
Switch (expression)
{case x:
      // code block
    break;
 case y:
      // code block
    break;
 default:
      // code block
 }
```

```
int day = 4;
switch (day) {
 case 1:
  printf ("Monday");
  break;
 case 2:
  printf ("Tuesday");
  break;
 case 3:
  printf ("Wed");
```

break;
case 4:
printf("Thursday");
break;

}

Break Keyword

Break out of the switch block

Most Imp.

---

loops

loop can execute a block of code as long as a specified condn is true

loop

while

```
while (condition)
{ // code
}
```

do/while

```
do { //code
}
while (condition);
```

```
int i=0
do { printf("%d\n",i);
i++;
while(i<5);
```

for

most imp

Page No.

while Syntax

int i=0

while (i<5)

printf("%d \n", i));

i++;

}

~~do/while~~

~~for (~~

for (initialisation; condition;

{code

}

int i;

for (i=0; i<5; i++)

{printf("%d\n", i));

}

Nested loop

We can also place a loop inside another loop. this is called 'nested loop'

the inner loop will be executed one time for each iteration (repetation)

```
#include <stdio.h>
int main() { int i,j;
    // outer loop
for(i=1; i<=2; i++)
{printf("outer: %d\n", i);
  for(i=1; i<=3; i++)
  {printf("inner: %d\n", i);
  }
}
  return 0;
}
```

op

```
outer: 1
  inner: 1
  inner: 2
  inner: 3

outer: 2
  inner: 1
  inner: 2
  inner: 3
```

Break : This statement can be used to jump out of a loop.

```
int i;
for (i=0; i<10; i++)
{ If (i==4)
  { break;
  }
  printf ("%d\n", i);
}
```

o/p -

0
1
2
3

Continue : This statement breaks one iteration (In the loop), If a specified condition occure and continue with the next iteration in the loop

```
{
  int i;
for (i=0; i<10; i++)
  { if (i==4)
    {continue;
    }
    printf ("%d\n", i);
  }
  return 0;
}
```

Output

```
0
1
2
3
5
6
7
8
9
```

goto statement : It is a jump statement which is some times refered as an unconditional jump statement. The goto statement can be used to jump from anywhere to anywhere within a function.

Syntax ①

```
goto label;
:
:
label:
```

Syntax ②

```
label:
:
:
goto label;
```

Program

```
#include <stdio.h>

void check_even_odd (int num)
{ if (num % 2 == 0)
    goto even;
  else
    goto odd;

even: printf("%d is even", num);
      return;

odd: printf("%d is odd", num);
}

int main( )
```

```
int main ( )
{ int num = 26;
  check even odd (num);
  return 0;
}
```

O/P 26 is even

Disadv

(1) It makes the program logic very complex

(2) The use of goto makes tracing the flow of the program very difficult.

(3) The use of goto can be simply avoided by using break and continue.

(4)

Comma operator : Some times we assign multiple values to a variable using comma in case comma is known operator

```
a = (10, 20, 30
b = (10, 20, 30);
```

O/P : a = 10
b = 30

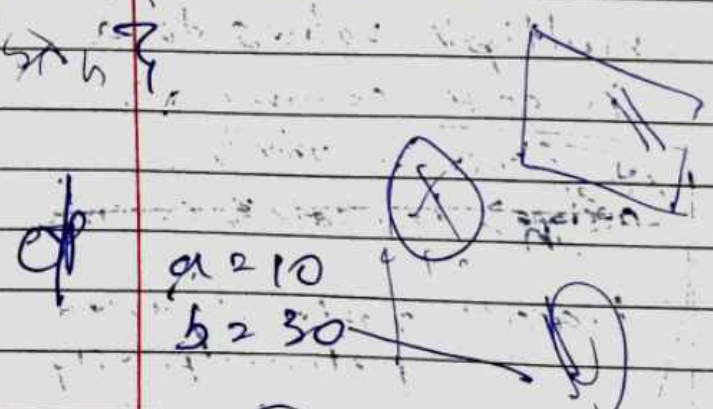
operator priority
~~assign > bracket > comma~~
bracket right most value preferred/priority
bracket > assign > comma

In the 1st statement value of a = 10 bcz assignment (=) has more priority than comma. In the 2nd statement value of b = 30 bcz all values are enclosed in ( ) and ( ) has more priority than =.

When multiple values are given with comma oper. within the ( ) then right most value is considered as a result of the operation.

( ) → braces / operator
= → assignment operator

```
int main()
{ int a, b;
  a = (10, 20, 30);
  b = (10, 20, 30);
  printf("a = %d, b = %d \n", a, b);
  return 0
}
```



18/05/23
Comp

Functions

Date

```
main()
{
    float square(float);
    float a, b;

    printf("Enter a number:");
    scanf("%f", &a);

    b = square(a); // calling function with actual parameter
    printf("square = %f", b);
}

float square(float x);
// function definition with formal argument
{ float y; /* local variable */
  y = x*x

  return(y);
}
```

- Function
  - Library fn
    - stdio.h
    - math.h
  - User-defined fn

Date ___/___/___

Passing a parameters to the fn

① call by value

call by reference sending the values of the argument to fn when a single value is passed to a fn while an actual argument the value of the actual argument is copied into the fn. Therefore the value of the corresponding formal argument can be altered within the fn but the value of the actual argument within the calling routine will not change

② call by reference

#include

main ()

Date

{ int a=2

printf ("In a = %d (" before calling func"), a);

modify (a);

printf ("In a = %d (from main after calling function"), a)

modify (int a)

{a* =3

printf ("In a = %d (from the function after modifying"), a);

output: a=2 (before calling func
a=6 (from the func after being modified
a=2 (from main after calling the function

The original value of a=2 when main is executed this value is then passed to the fn modify where it is multiplied by 3 and new value is displayed within the fn. Finally the value within the main is ——— after control it's transfer back ——

There results show that a is not altered in main even though the corresponding value of a is changed

Call by reference: It means sending the address of the argument to the called fn. In this method, the addresses of actual arguments in the calling fn's are copied into formal arguments of the called fn's. Thus using these addresses we would have an access to the actual argument and hence we would be able to manipulate them

```
int add (int*, int*)
void main()
{ int i; j, k;
~~void main()~~ )
printf("
```

Photo

case (2) for any value of m $\phi(m) = 0$ and $\phi(m) = 0$ then this curve has parallel asymptotes and these asymptotes are find by

$$\frac{1}{2}c^2 \phi_n''(m) + c\phi_{n-1}'(m) + \phi_{n-2}(m) = 0$$

FUNCTION

```
int add(int *, int *);
void main()
{
int i, j, k;
printf("Enter value");
scanf("%d", &i);
printf("Enter value");
scanf("%d", &j);
k = add(&i, &j);
```

printf("addition %d", add(

int add(int *a, int *b)

{ return *a + *b;

}

$\frac{1}{2}C^2\phi_3''(m) + C\phi_2'(m) + \phi_2(m) = 0$

Date ___/___/___

19/10/23
Computer

Prime Number & Armstong

```
1  #include <stdio.h>
2  #include <math.h>
3  type → name   formal parameter
4      int CheckPrimeNumber(int n);
5      int CheckArmstongNumber(int n);
6      int main()
7
8    { int n, flag;
9
10     printf("Enter a positive integer ");
11     scanf("%d", &n);
12
13     //Check prime no.
14
15     flag = CheckPrimeNumber(n);
16
17     if (flag == 1)
18
19     printf("%d is a prime number\n",n);
20
21     else
22         printf("%d is not a prime number");
23
24     //Check ArmstongNo
25     //Check armstong no.
26
27     flag = CheckArmstongNumbe(n);
28
```

Page No.

```
    if (flag == 1)
        printf ("%d is Armstong number", n);
    else
        printf ("%d is not an armstong naber");

    return 0;
}

int CheckPrimeNumber (int n);
{ int i, flag = 1, SquareRoot;

    // Calculate squarroot

    SquareRoot = sqrt (n);
    for (i = 2; i <= squareroot, i++)

    { if (n % i == 0)
        { flag = 0;
          break;
        }
    }
    return flag;

}
```

Program for Summation

Date ___/___/___

```
#include<stdio.h>

int sum();
int sub();
int main()
{ int a,b;
  printf("Enter a,b");
  scanf("%d %d",&a,&b);

  y=sum(5,10);
  printf("y %d");
}

int sum(int a, int b);
{ int res;
  printf("Enter a and b");
  scanf("%d %d",&a,&b);

  res=a+b;
}
```

25/10/23

Function

~~Recursion~~

Comp.

```
#include<stdio.h>
int myfunction (int x, int y)
{
    return x+y;
}

int main()
{ int result = myfunction (3,5);
  printf ("Result is = %d," Result);
  return 0;
}
```

O/P Result is 8

Page No.

Recursion fn

Date / /

Recursion is a ~~teching~~ technique of making a fn call itself.

Recursion ≡ Repeatition

```
int sum (int k);
int main()
{
    int result = sum(10);
    printf ("%d", result);
    return 0;
}

int sum (int k)
{if (k>0)
  {return k+ sum(k-1);
  }
 else
  {return 0;
  }
}
```

for factorial

```
#include <stdio.h>
int multiplying number (int n);
int main()
{   int n;
    printf ("Enter a number ");
    scanf ("%d", &n);
    printf ("Factorial %d = %d ", n,
            multiplying number (n) );
    return 0;
}
```

10 int multiplying (int n)
{ if (n>=1)
Date / return n* multiplying number (n-1);
else
return 1;
}

---

(1) Difference between fn and recursive fn explain it with the help of an example

one without recursion
one with recursion

---

fibonacci

(2) fibonacci series

1 · 2 · 3 4 ——

26/10/23

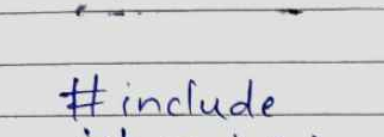
Arrays

1D 2D (multidimensional)

Syntax

datatype arrayname [size]

Arr[5]




```
1   #include
2   int main()
3   {
4       int arr[5] = {10,20,30,40,50};
5       int arr[] = {1,2,3,4,5};
6       float arr2[5];
7       for (int i=0; i<5; i++)
8           {
9           arr2[i] = (float) i*2.1;
10          }
11      return 0;
12  }
```

Try in Deepawali break

① Pattern wale programs

```
1
1 2 1
1 1 2 2 1
```

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

② fabonicca series

1 + 4 + 8 + ...

③ Mark sheet making

Name
number
%(calculated)

① Write a program to access the array

```
#include <stdio.h>
int main
{
  int arr[5] = {10,20,30,40,50};
  printf("Element at arr[2] %d\n", arr[2]);
  printf("Element at arr[4] %d\n", arr[4]);

  return 0;
}
```

O/P = 30
50

To Update
Syntax

```
arrayname[i] = new value;
arr[2] = 70;
```

~~Traversing~~ = Visiting

```
#include <          >
int main()
{
  int arr[5] = {10,20,30,40,50};
     arr[2] = 100;
  printf("Elements in array ");
  for (int i=0; i<5; i++)
    {
      printf("%d", arr[i]);
    }
  return 0;
}
```

O/P

10
20
100
40
50

```
char arr[4] = {'c', 'k', 'o', 'p', 's'}
int i=0
while (arr [i])
    {
    printf ("%c", arr (i++));
    }
```

2D

rows columns

```
arrayname [size1][size2]
```

```
#include <          >
int main()
{int arr[2][3] = {10,20,30,40,50,60};
 printf ("2D Array\n");
 for (int i=0; i<2; i++)
     {
      for (int j =0; j< 3; j++)
        {printf ("%d", arr[i][j]);
        }
      print ("\n");
     }
 return 0;
}
```

O/P : 10 20 30
40 50 60

Properties of array:

1. Fixed size
2. Homogeneous selection
3. Indexing - 0, ..., n-1
4. Dimension: Dimension is no. of indexes required to refer to an element in the array

(5) Contiguous Storage
(6) Random Access
(7)

Write a program to ~~print~~ ~~to~~ find the largest no. in an array

```
#include <stdio.h>
int getMax(int arr, int size)
{
    int max = arr[0];
    for (int i=1; i<size; i++)
    {
        if (max < arr[i])
        {
            max = arr[i];
        }
    }
    return max;
}

int main
{
    int arr[10] = {135, 165, 1, 16, 511, 65, 659, 659
                   169, 4.};
    printf("Largest Number in the array : %d",
           getmax(arr, 10));
    return 0;
}
```

30/10/23

String → group of characters

'a' → character → ' '
"abcd" → string ! → " "

string in C
geeksforgeeks

```
Char greeting[ ] = "Hello, World!"
printf("%s", greeting);

printf("%c", greeting[0]); → o/p - h

int i;
for(i = 0; i<11; i++){
    printf("%c\n", greeting[i]);
}

char greeting[ ] = "Hello. World";
char greeting[ ] = {'H', 'e', 'l', 'l', 'o', 'W', 'o', 'r', 'l', 'd', '\0'};
```

"We'

#include <string.h>

fn name	→ strlen( )
description	strcat(str1, str2)
	strcpy(str2, str1)
	strcmp( )

size should be greater or at least equal

#include <string.h>

(1) char alphabet[] = "ABCDEF....Z";
printf("%d", strlen(alphabet)); o/p in no.

(2) char str1[20] = "Hello";
char str2[] = "World";
strcat(str1, str2);
printf("%s", str1);

o/p = HelloWorld

(3) char str1[20] = "Hello World";
char str2[20];
strcpy(str2, str1);
printf("%s", str2);

(4) char str2[20] = "Hello";
char str2[] = "Hello";
char str3[] = "Hi";
printf("%d\n", strcmp(str1, str2));
printf("%d\n", strcmp(str1, str3));

o/p → 0 (same)
-4 (string are not equal)

(5) ~~int *ptr;~~
int myage = 43;
printf("%p", &myage);

Pointer

pointer is a variable that store the memory adress of another variable of as its value. Pointer variable points to the data type of the same type and it is created by * symbol

```
int myage=43;
int * ptr = &myage;                          o/p
printf ("%d\n", myage);   → 43
printf ("%p\n", &myage);  → no. may address of myage 0x77∞
printf ("%p\n", ptr);     → 0x77
                            memory address of
                            myage with pointers
```

```
int* my-name;
   or
int *myname;
```

declaration में है तो * pointer
अन्य fⁿ में/int main() में {} में
declaration में नहीं है तो * dereferen