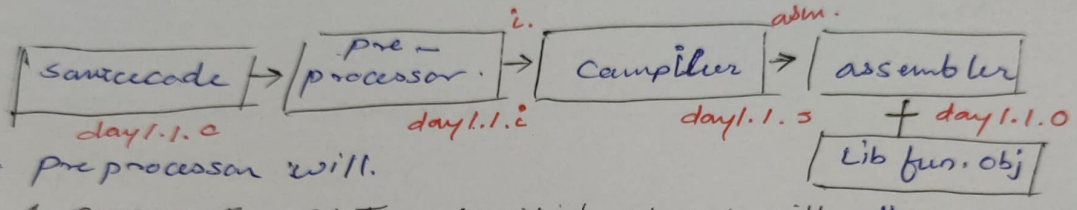
Date: 08/06/2023.

1. # include <stdio.h>.
// header file included
// declaration of data types.
// definition of symbolic constants.
// ~~def~~ declaration of function
// typedefination.

// gcc Day1_1.c -save-temps

* Commands.
cmd> gcchellow.c
cmd> ./a.exe.

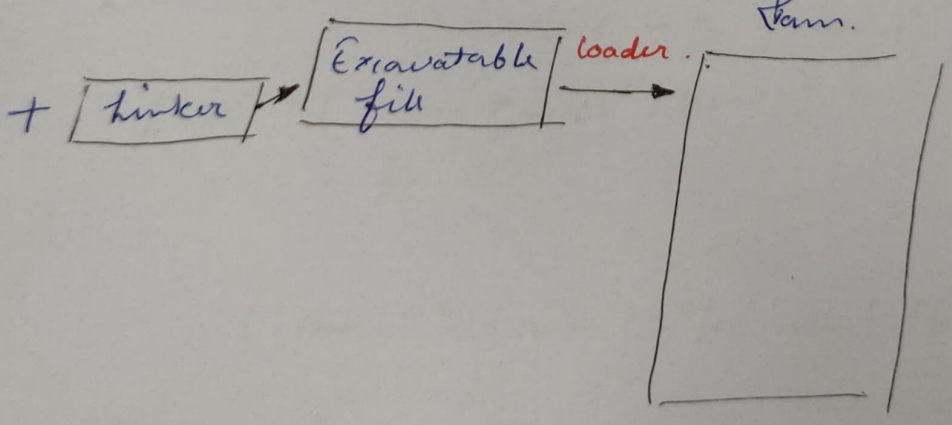


* preprocessor will.
1. process the statement which started with #
2. Removed comment.

* Compiler converts high level language instruction in assembly instructions.

* Assembler will assemble as assembly instruction & will generate object code (machine understanble code)

* Linker: links used objects code with library function. object code & generate executable



* Loader which is part of OS loads application into RAM and application starts it execution

// main. is entry. point funcation. Called.

int main ( ). // main is user defined. funcation, by the system.

// int here indicate return type of funcation

// and main is ~~de~~ indentifer to block.

{ printf ("welcame to pre-Cat - CH04 Batch \n")

// printf will show given msg.

return 0;.

}

Date: 9/06/2023.

1. ~~#~~ include <stdio.h>.

// data section

// bss code segment

// Stack section // local variable.

// ~~Heap~~ section // dyanic. memeny.

Note: Stack section whenever. we get memeny there is going to be garbage value.

int main ( ).

{. int num; // request 4 bytes. memeny // 4 bytes. are initialised with garbage.

printf ("num" = %d", num).

//

&& => address operator.

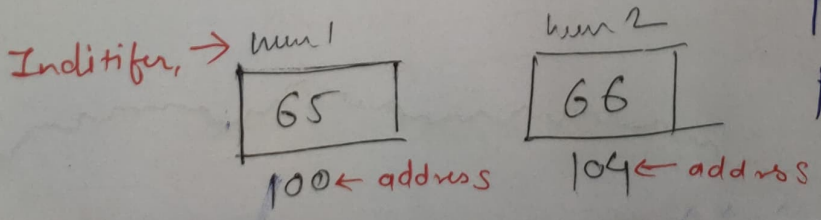
6422 => Stack section.

("num1 = %d num1 = %c num1 = %0 num1 = %x. \n",

num1, num1, num1, num1);

pf(num1) Int. value

pf(&num1) address.



(\n) - new Line. — 10 ascii value.

\ is a escape characters help to escape original // meaning of followed by character.

(\t) → add a horizontal tabspace. — 9
(\r) → it move carriage to the beining of line — 13
(\b) → it move carriage one character back — 8
(\v) → add vertical tap space.
(\a) → add a alert beep.
(\f) → add form feed.

```
int main ()
{  printf ("ascii value of \n = %d \n", '\n');
                          \t = %d \n", '\t');
                          \r = %d \n", '\r');
                          \b = %d \n", '\b');
```

```
int main ()
{  signed char ch1 = 'A'
   signed char ch2 = 'B'
   unsigned char ch3 =
   ch3 = ch1 + ch2.
   printf ("%d + %d = %d \n"
```

Size = %u
Σ operator.

```
int main ()
{  float fvar = 45.67
   printf ("fvar = %f size = %u \n", fvar, sizeof (float));
   printf ("fvar = %.2f size = %u \n");
   printf ("fvar = %10.2f size = %u \n");
   printf ("fvar = %-10.2f size = %u \n");
}
```

Note: hardcoded character constant is alway treated as type in the compiler
hardcode float constant → type of double compiler.

Increment / Decrement Operator's.

Types:

1) Pre-increment = int a = 10
x = ++a.
pf("x") = 11.

2) post increment = int a = 10
x = a++.
pf("x") = 11
("a++") = 10

3) Pre-decrement = int a = 10
x = --a.
pf = ("x") = 9.

4) Post decrement int a = 10
x = a--
pf(,x) = ~~10~~ 9
(a--) = 10.

Date: 12/06/2023.

1 Operators In C.

* C don't have ballian data type's.
// if any expression result non zero value then its considered as True. nothing but 1. OR its considered as false nothing but 0

^ - operator - either is one True is on and otherwise off

```
  1 0 1 1
  1 1 0 1
 ---------
  0 1 1 0
```

// exp1 && exp2 = <result>

exp1	exp2		result
1	1	=	1
1	0	=	0
0	1	=	0
0	0	=	0

// <exp1> O.R. exp2 = result

exp1		exp2		result
1		0	=	1
0		1	=	1
0		0	=	0
1		1	=	1

If bitwise operator's is applied on negative number -12>>3.

→ Step to be followed,
1. find the binary of positive value
2. Apply 1's compliment on binary... invert bits.
3. Apply 2's → 1 — add 1 into binary.
4. Process right shift operator.
5. Apply 1's compliment on binary... 1 invert bits.
6. Apply 2's → 1 — add 1 into binary
7. find decimal.

```
int main ()
{ int a = 2
  ++a ; // preincrement     a = a + 1
  a++; // post increment    — // —
  --a; // predecrement      a = a - 1
  a--; // post decrement    — // —
```

Rule's:
when ++/-- operator is applied, before the operand.
1. apply ++/-- operation on operand.

```
d = -9 || 4 << 1
d = 2, << 3 ||
```

=

when comma operator is in precedence following both rule's are applicable.
1. Solve each expression seprated commas from left to right
2. Result of right most expression has to associate to sequence pt

Number Conversion

1. Convert Binary No to Decimal No.
2. Convert Decimal No. to Binary No.

* Decimal Number base = 10 . Eg => $( )_{10}$

Binary Number base = 2 Eg => $( )_2$

Formula : N. . . . . 16 8 4 2 1.

Eg (Binary to Decimal)

1. $(1000)_2$.

8	4	2	1
1	0	0	0

8 Ans = 8.

2. $(10011)_2$.

16	8	4	2	1
1	0	0	1	1

16 + 2 + 1 = 19 Ans = 19.

3. $(111001)_2$

32	16	8	4	2	1
1	1	1	0	0	1

32 + 16 + 8 + 1 = 57 Ans = 57

4. (110010)

32	16	8	4	2	1
1	1	0	0	1	0

32 + 16 + 2 + 0 = 50 Ans = 50

Eg (Decimal to Binary).

1. $(47)_{10}$

32	16	8	4	2	1
1	0	1	1	1	1

Ans = $(101111)_2$

2. $(65)_{10}$.

64	32	16	8	4	2	1
1	0	0	0	0	0	1

Ans = $(1000001)_2$

3. $(86)_{10}$

64	32	16	8	4	2	1
1	0	1	0	1	1	0

Ans = $(1010110)_2$

4. $(98)_{10}$

64	32	16	8	4	2	1	0
1	1	0	0	0	1	0	

Ans = $(1100010)_2$

5 $(102)_{10}$

64	32	16	8	4	2	1
1	1	0	0	1	1	0
				1	0	0

~~Ans = 1 101000~~

$(1100110)_2$.

6). $(19)_{10}$

16	8	4	2	1
1	0	0	1	1

10011.

Date: 13/06/2023.

Rules:

Typedef :- Rename.

1. Help to give another to existing Data Type.
2. Improve readability of source code
3. Help to port code across multiple architecture / platform.

Enum:

enum <tag name> {(<enumated filed>)}

1. Help to define new data. types.
2. Help to improve readability of Source Code.
3. Collection of enumerate fields.
4. Each enumated field represent integer

★ Break is one jump Statement.

Switch Case:

1) Each case should by followed by integer constant
2. Cannot add duplicate case.
3. Use of deafault case optional

*** Loop:**

1. On-Entry check - while

```
int main()
{ int i = 3, 2, 1, 0, -1
  while(i--)
    printf("Inside loop i = %d \n = i);
  printf("outside loop i = %d \n = i);
}
```

3, 2, 1, 0, -1

2. On Exit. Level.

for loop:

for (<initial Statement>; <expression>; <modification statement>

H.W

Conditional Statement:

1) If else:

```
if (condition) {
  // do something if TRUE
}
else {
  // do something if FALSE
}
```

Else if:

```
if (condition 1) {
  // do something if True.
}
else if (condition 2) {
  // do something if 1st is FALSE
  & 2nd is TRUE
}
```

Conditional Operators.

1. Ternary

Condition ? do something if True : do something if false.

(? = check, : = नहीं तो)

2. Switch:

```
switch (number) {        // number, character
case C1 : // do something
          break;
case C2 : // do something
          break;
default : // do something
}
```

Properties:

a. Cases can be in any order

b. Nested switch (switch inside switch) are allowed.

x=5

* static int = 5, 6, 7
  int = 5, 5, 5.

Loop Control Instruction:

1. for 2. while 3. do while

1. For Loop

```
for (initialisation; condition; updation)
    { // do something }
```

int i = 1; i <= 5 i = i+1

i = 1

i++	++i
1, 2	1, 1
i--	--i
1, 0	0, 0

2. While Loop

```
while (condition) { // do something )}
```

condition always be out of the block (at starting) - declration

3. do while Loop.

```
do { // do something } while (condition)
```

* Break Statement: → Exit the code.
* continue Statement → skip to next iteration.
* factorial → n=5 1×2×3×4×5 = 120

---

functions: block of code that performs particular task.

Take argument → Do work → Return Result

it can be used multiple times.
it increase code reusability

Function's properties:

1. Execution always starts from main
2. A function get called directly or indirectly from main
3. There can be multiple function's in program

Types:

1) Library function: scanf(), printf()
2) User defined: declared & defined by program

Syntax 1:

function Prototypes
void printHello();
↓
Result nothing

Syntax 2:

function Defination
void printHell(){
print("Hello")
}

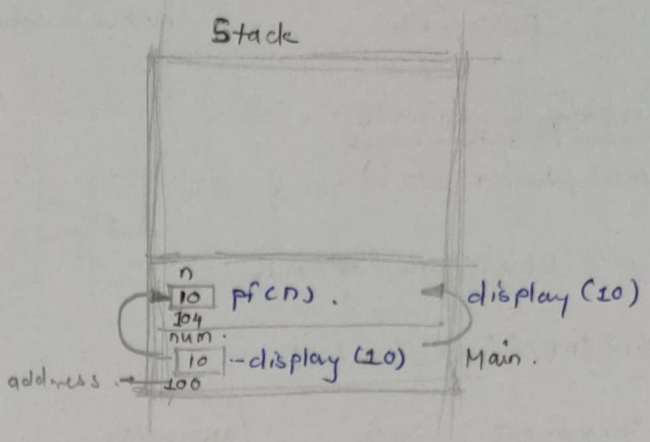
Syntax 3:

function Call
int main()
printHello();
return 0
}

Date: 14/06/2023.

* Do Entry check,
  Do Exit check.

```
#include <stdio.h>
int main (void)
{ int i = 0;
  for (i=0; i<5; i++);
{ if (i<=5)
  { printf ("Sunbeam \n");
  }
else
  break;
```



Function activated Record. - Stackframe.

* User's define function:
  LOC = Line of code.

(...) j => declaration.

(...) => defination.

* Call is doesnot contain any intruction (dont show data types)

Pass by Value:

* Auto, Register, Extern, Static
  (~~Strege~~ Storage classes)

Scope:- To whom reasources is know.
- who can asseee reasources.

Life: Time span how long bytes memory is retained in live mode / in use mode.

Eg:

```
Int main ()
{ int num = 10;
  display number (num);
  return 0;
}
void display number (int n)
{ printf ("Number %d", n );
```

keyword	Scope	life	memory fram.	deafult value.	
Auto	block ()	block ()	stack	garbage	
Register	block	block	~~Stack~~ cpu register	garbage	// not allowed to use here // ~~not~~ cannot take address of register
Extern	program	program	data seeth	~~garbage~~ 0	// extern is applicable only to global reasource
Static	block	program	data seeth	0	// memory recived, on first call of function in which it is declared // variable here is initialed, at the time of ~~variable~~ declaration

Function's :-

Function's can Take value & Same value.

parameter — return value

Passing Arguments :

void printHello().

void printTable(int n);

int sum(int a, int b);

Argument	V/S	Parameter
1) value that are passed in function Call		1) value in function declaration & definition
2) used to send value		2) used to receive value
3) actual parameter		3) formal parameter

Note :

a) Function can only return one value at a Time

b) Changes to parameter in function don't changes the value in calling function

Recursion :

When a function call itself it called recursion

Eg Sum of first n natural numbers.

n → 1 + 2 + 3 + 4 . . . . n-1+n.

n = 5 → 1+2+3+4+5 sum(4)+5

n = 4 → 1+2+3+4 sum(3)+4

n = 3 → 1+2+3 sum(2)+3

n = 2 → 1+2 = sum(n-1)+2

n = 1 → 1 ← Last value.

Properties of Recursion :

a. Anything that can be done with Iteration, can be done with recursion and vice-versa.

b. Recursion can ~~something~~ sometime give the most simple solution.

c. Base Case is the condition with which staps recursion.

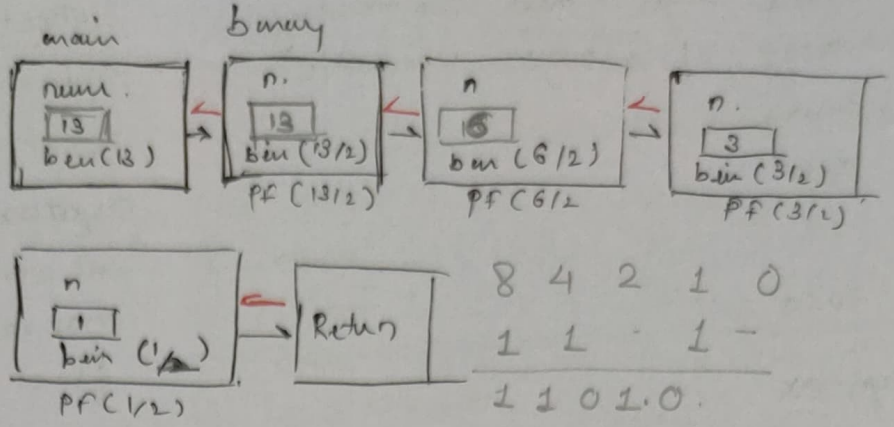
d. Iteration has infinite loop & Recursion has Stack overflow.

ctrl + c.

Date - 15/06/2023

* Recussion :

```
void binary (int n).
{ if (n==0);
    binary (n/2)
printf ("%d", n%2);
```

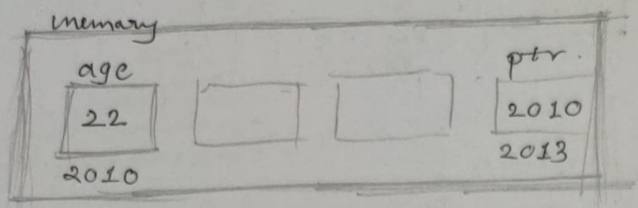


* Pointer.

```
int main ()
{ int num = 5
  printf ("before num = %d \n", num);
  test (&num).
  printf ("after call num = %d \n", num);
  void test (int *p).
{ printf ("*p = %d. \n", *p);
*p = 7
  printf ("*p = %d \n", *p);
```

Pointers

variable that stores the memory address of another variable.



Syntax

```
int *ptr = &age;
int age = *ptr;
```

* = value at address
& = address of

Declaring Pointers

```
int *ptr;
char *ptr;
float *ptr;
```

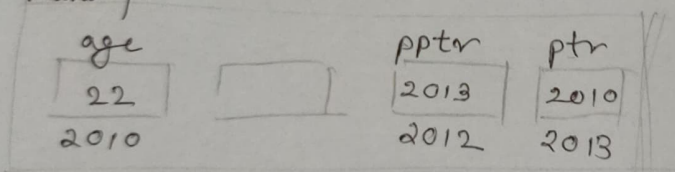
Format Specifier

o/p = hexadecimal
o/u = Unsigned Integer

```
printf ("%p", &age);
printf ("%p", ptr);
printf ("%p", &ptr);
```

Pointer to Pointer

A variable that stores the memory address of another pointer



Syntax.

```
int **pptr;
char **pptr;
float **pptr;
```

Pointer in function Call.

- Call by value: we pass value of variable as argument
- Call by Reference: we pass address of variable as argument

Arrays:

Collection of similar data types stored at contiguous memory location.

Syntax.

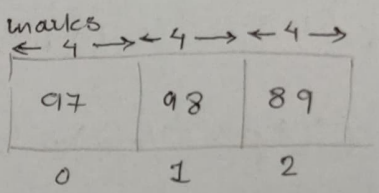
```
int marks [ ]
char name [ ]
float price[ ]
```

Input & Output:

```
scanf ("%d", &marks [0] );
printf ("%d", marks [0]);
```

Initialization of Array

```
int marks [ ] = {97,98,89}
int mark [3] ={ 97, 98, 89}
```



memory Reserved = 4x3 = 12 by

Pointer Arithmetic.

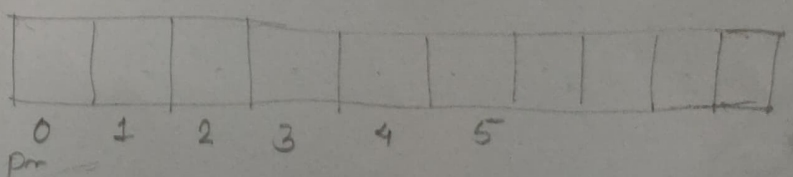
- pointer can be incremented & decrement
- we can also subtract one pointer from another
- we can also compare 2 pointers

Array is a Pointer

```
int *ptr = &arr [0];
int *ptr = arr;
```

Traverse an Array.

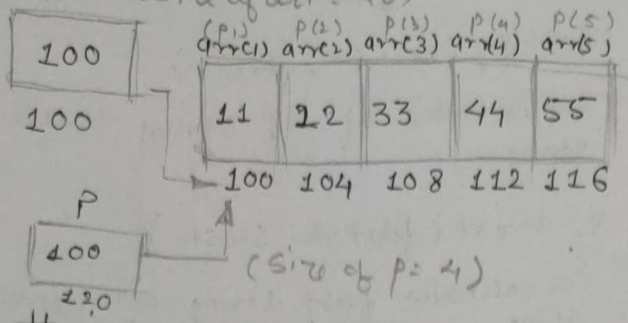
```
int addhar [10];
int *ptr = &addhar [0];
```



Date : 16/06/2023.

Pointer Arthicmatic.

arr (size of arr : 20)




```
#include <stdio.h>
int main()
{
  int arr[5] = {11,22,33,44,55}
  test(arr);  printf("arr = %u & arr = %u", arr, &arr, sizeof(arr));
}
void test(int p[5])
{ printf("p = %u & arr %u /n",
          p, &arr, sizeof(p));
```

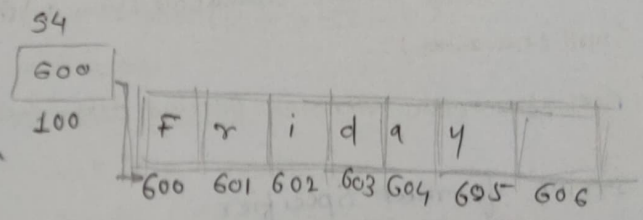
```
int num = 5
int *p = num.
++*p = ?   //*p = *p + 1  //num = 6.
*p++ = ?  //1. *p   // 2. p++
          // p = p + 1  //p = 104
(*p)++    //*p = *p + 1
*(p++)    //p = p + 1  // p = 104
```

String (1-D Array).

\0 OR NULL - are same

```
int main()
{ char S1[20] = "test"
  char S2[] = "friday".
  char S3[] = "frid\0day".
  char *S4 = "friday".
```

```
printf("size of %u \n", sizeof(S1));  20
                        (S2),  7
                        (S3),  8
                        (S4),  4
```




```
int main().
{ char *S4 = "friday"
  int num = 5;.
  printf("Hellaworld")
*S4 = f, // try to modify memory
          at D.S.R.O // causing
          Runtime Error.
  printf("check this ...")
  printf("S4 = %u & S4 = %u", S4, &S4)
  printf(S4); // friday
  printf(S4 + 3) // day
```

//strlen. //strcopy //strcat

//strchr //strstr //strcmp } main Imp

Arrays as functions Argument.

//Function Declaration

```
void printNumber(int arr[], int n)
        OR
void printNumber(int *arr, int n);
```

//Function Call

```
printNumber(arr, n);
```

Multidimensional Array.

2D Array

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

	0,0	0,1
arr[0][0]	1	2
arr[0][1]		
arr[1][0]	1,0	1,1
arr[1][1]	3	4

String's

A character array terminated by a '\0' (null character)

null character denoted string termination

String format Specifier

"%s"

```
char name[] = "Akash";
printf("%s", name)
```

Imp:

scanf() cannot input multiworded String with spaces

Here,

gets() & puts() came into picture

String functions.

gets(str)

input a string (even multiworded)

puts(str)

output string

fgets(str, n, file)

Stop when n-1 char input or new line is entered

Standard library function

<string.h>

2. strcpy(new str, old str).

copies value of old string to new string

3. strcat(first str, sec str).

Concatenates first string with second string

4. strcmp(first str, sec str).

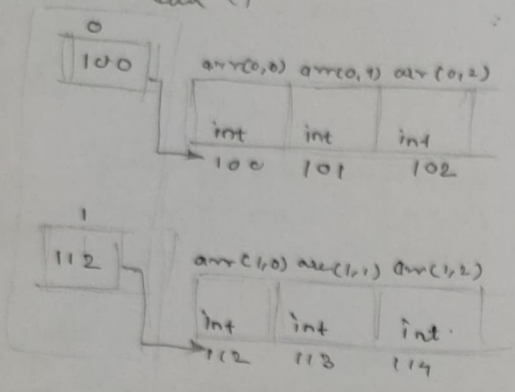
Compares 2 string & return value

Date 17/06/2023

2-D Array.

```
# include <stdio.h>.
int main ().
{ int arr [2][3] = {11, 22, 33, 44, 55} }
} // last dimention is compulsary
```

```
int main ()
```



// envhere will collect all system environment variable in the string format

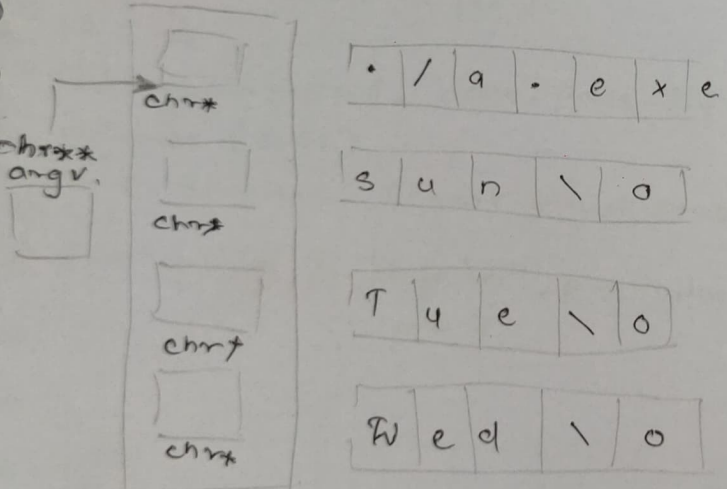
Dynamic alocations.

```
# include (stdio.h>
# include <stdlib.h>.
  void *— cdel malloc (size_t);
int main ().
{ int name = 567
  char ch = A
  void *uptr; // it is a generic pointer
      designed to store address
      of any type location.
```

Malloc.

```
# include <stdio.h>.
  void *malloc (size_t)
  int main
{ int neles;
  int *p = NULL;
  printf ("Specify No of element
          to process \n")
  scanf ("%d" & neless).
// 1. request memory at runtimes
  p = (int *) malloc (size of (int)*
      neles); //malloc reqted memory from garbage.
  if (p = Null)
{ printf ("Not enough memory \n")
  return 0.
// 2. If memory is received that
     use it
// 3. deallocate memory
```

```
# include <stdio.h>.
// argc here will catch count of
   argument's given to main
int main (int argc) argument count
```




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

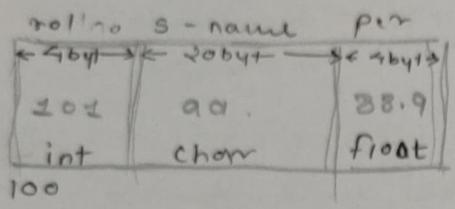
// argv here will catch all argument given to main in string format

argv : argument vector

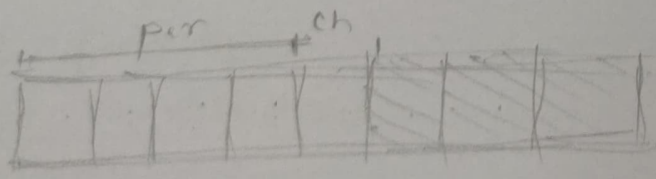
File Input / Output.

```
#include <stdio.h>
void fst ()
{ int *p = Null.
  p = (int *) malloc (size of (int)
// here an exit of function memory
gain for p will be deallocated.
// hence there will not any pointer
via. which we can.
```

Structure.




```
int main ()
{ struct student s1 = {101, "aa" 88.9"}
  struct date d1,
  printf ("size of u size=%u \n,
          size of (s1), size of (s2));
printf ("%d %s %.2f \n", s1.rollno,
        s1.sname s1.per).
s2 = s1
display_student_ data-pass-by-
address (&s1).
void f1 ——— (const struct student *p)
{ print f ("d %s %2.f \n", p → rollno, s-name,
                per ).
```



1. A stack is last in first out data structure
2. A queue is first in first out data structure
3. A linked list is dynamic data structure
4. Avg time complexity O(n)
5. Time complexity of an alogorithum/ program is excution time + compilation time
6. Sorting algorithum is efficient for array having small size insertion sort
7. Time complexties are genrally same coorst case & avg case
8. 823 ^/23*+51*= - +a*bc^^def
9. Binary search Tree is balanced. all nodes are balanced.
10. Time complexity for Searching an element in a binary search Tree O(logn)
11. Sum of degree of all vertical is 2E
12. Trawsel of graph is different from. Tree because
- There can be loop in graph sowe must maintain a visited flag for every vertex
13. Spanning Tree of graph contain minimum V-1 no of edges.
14. Edge of any vertex to itself it's called loop.
15. Separate chaning can be implemented by using linked list
16. Clustering problem may occured in linear Probing
17. Double Hashing Separate Chaining
18. An Improvement on Direct Access Table is called Hash Table
19. ~~Direct~~ Dynamic Implementation of Hash Table - Separate Chaining
20. Number of key inserted into Hash Table - Load factor
21. Callision Handling method - Open Address
22. Application of Hash Table Cache Memory

Data Structure (10 marks)

Date 19/06/2023.

Algorithm: Can be represent using the. pseudocode or flowchat.
- Logical flow of given Statement

Flowchart: logical flow represent in pitcorial format.

Efficiency: can be decided based on. timespan and memory consumed a specific algorithm.

Timespan: can be calculated using concept time Complexity
- Memory consumed by the. algorithum can be calculate using a concept of Space. Complexity.

Time Complexity is measured in 2 different ways.

1. Asymptoic Time Complexity
   a. Best Case - Ω Omega.
   b. Average Case - Θ Thita.
   c. Worst Case - O Big Oh.
2. Symptoic Time Complexity

Space Complexity is calculated based on 2 factors.

1. fixed Space
2. Variable Space

Binary Search use divide & Concur. algorithum.
- One of the fastest Searching algorithum
- pre-requisite of binary search. algrithum is data has be sorts. order
- If key is present in collection. it will be always at mid location

Types of Alogorithum

1. Divide and Conqur Alogorithum
2. In place Algorithum
3. Greedy Algorithum

Divide & Concur algorithum is used by.

1) Binary search.
2) Merge Search
3) Quick search.

0	1	2	3	4	5	6	7	8	9
11	22	33	44	55	66	77	88	99	110

key = 110

lo = 0 mid = 4
h = 9

4	5	6	7	8	9
55	66	77	88	99	110

$\frac{4+9}{2} = \frac{13}{2}$

key = 110

l = 4 mid = 6.
h = 9

6	7	8	9
77	88	99	110

key = 110
l = 6 mid = 7.
h = 9

7	8	9
88	99	110

key 110 mid 8
l = 7
h = 9

8	9
99	110

key = 110
l = 8 mid = 8
h = 9

9
110

l = 9
h = 9
mid = 9

0	1	2	3	4	5	6	7	8	9
11	22	33	44	55	66	77	88	99	110

l = 0
h = 9
mid = 4
key = 25

0	1	2	3	4
11	22	33	44	55

l = 0
h = 4
mid = 2
key = 25

0	1	2
11	22	33

l = 0
h = 2
mid = 1
key = 25

1	2
22	33

l = 02
h = 01
mid = 1
key = 25

Date : 20/06/2023

Selection Sort

in the case of selection sort lowest element of collection is placed in order after completion of first pass

Selection sort algorithum need to execute n - 1 passes.

```
for (i=0, i<size-1, i++).
{ for (j=i+1, j<size, j++).
  if (p[i] > p[j]).
  p[i] = p[i] + p[j]      a = a+b
  p[j] = p[i] - p[j]      b = a - b
  p[i] = p[j] - p[j]      a = a-b
```

In case of buble sort highest element from collection will be place in order

Bubble sort alogrithum exacts n - 1 passes

Time Complexity = (n-1) + (n+1) + n-3)..

j<5 -.

Stack Data Structure. -

last in first out

- used in Recursion.
- Expression Conversion & Evaluation
- Undo/Redo Operation
- forward/Backward web page
- Depth first Search.

Expression can be represented 3 types

Infix Expression : a+b left operand → operator → Right Operand
2 + 3 * 5

Prefix Expression : +ab operator → Left operand → Right operand.
+2 * 35

Postfix Expression: ab + → Left operand → Right operand → Operator.
23 5 * +

Operation. can be performed on stack

- push - add element. (is_full)
- pop - delet top element (is_empty)
- peek - retrieve top most element (is - empty)

Stack Application :

1. + -
2. * / %
3. $ ^
4. ( )

(Infix solved method) (a+b)

Eg 2+(3*5

= 2+(3*5) (2+(3*5)).

= 2+(15)

= 17

Infix to prefix. (+ab)

= 2+(3*5)

= 2+35*

Infix to postfix (ab+).

= 2+(3*5)

= 235*+

3+6*2/4-(5^(10$4))

3+((6*2)/4)-(5^(10$4))

(3+((6*2)/4))-(5^(10$4))

=(3+((6*2)/4))- 5^(104$)

=(3+((6*2)/4))- 5104$^.

=(3+(62*)/4)).-5104$^.

=(3+62*4/)-5104$^

= 362*4/+ - 5104$^

= 362*4/+5104$^-

+ ab

3 + ((6 * 2) / 4)) - ((5 ^ (10 $ 4))

3 + ((6 * 2) / 4)) - $ ^ 5104.

3 + ((6 * 2) / 4)) - $ ^ 5104

(3 + (* 62 / 4))) - $ ^ 5104

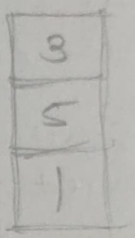
*) + 3624 - $ ^ 5104.

-+3/*624 ^ $

2 + 3 * 5 - 1

= -+2*351 → post

153*2+- → pre



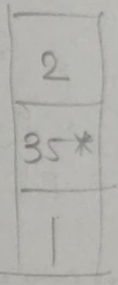
* is read in operator

1st pop = 3 = Left operand

2nd pop = 5 = right operand

After Conversion 35* is operand

Hence push



2 is read.

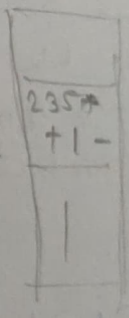
+ is operator.

1st pop () = 2 = left operand

2nd pop () = 35* = = right

after conversion 235*+

operand. Hence push



- is read is operator.

1st pop () = = 235*+ = left operand

2nd pop () = 1 = right

After Conversion 235*+1-

pop the els which is left in

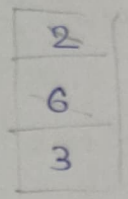
Stack is post for given prefix

235*+1-

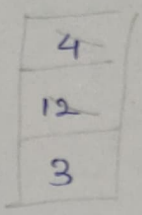
3 6 2 * 4 / + 5 10 4 $ ^ -

3 is read is operand. push on stack
6 " "
2 " "

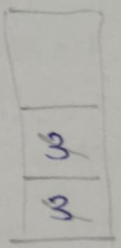
* is operator. is read pop() twice
1st pop() = = = 2 = = = right operand
2nd pop() = = = 6 = = = left operand.
After evalution. 6 * 2 = 12 as operand.
Hence push



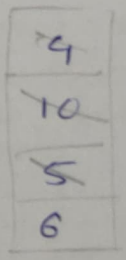
4 is read is operand push on stack
/ is read operator pop() twice.
1st pop() = = = 4
2nd pop() = = = 12
After evalution 12/4 = 3 as operand
push.



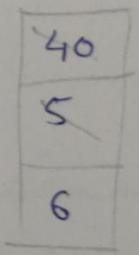
+ is read is operator = = pop() twice.
1st pop() = = = 3 = = = right operand.
2nd = " " = left operand.
After evalution 3 + 3 = 6 as operand.
Hence push.



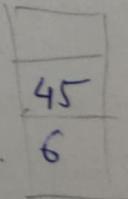
5 is read is operand. push on stack
10 " "
4 " "



$ is read is operator pop twice.
1st pop() = = = 4 = = = right operand.
2nd pop() = = = 10 = = = left operand.
After evalution. 10 $ 4 = 40 as.
operand Hence push



2nd pop() = = 5 = = left operand.
After evalution 5 ^ 40 = = 45



- Is read is operator = = pop() twice.
1st pop() = = 45 = = right operand
2nd pop() = = 6 = = left operand.
6 - 45 = 39

39

Date 21/06/2023

Queue Data Structure - FIFO

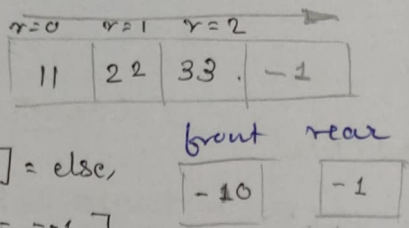
1. Enqueue / Add / Jaint / Insert
2. Dequeue / Delet / Leave / remove.
3. Peek
4. Pall

variations of Queue

- Linear Queue
- Circular Queue
- Priority Queue
- De Queue

Linear Queue.

```
If (rear == size - 1)
  Q is full
Else
{ Rear ++
  Else [rear] = else,
  If [ front == -1 ]
    front = 0
}
```



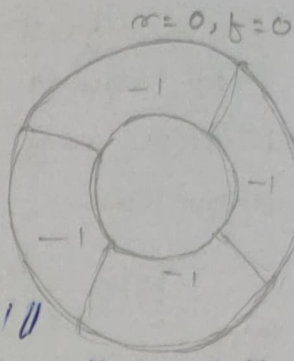
Delete Element from queue

```
If (rear == -1)
  Q is Empty !!!
Else
{ Else [front] = -1 //assume element is deleted
  front ++
}
```

OR.

```
If (front = rear) {
  { rear = -1;
    front = -1;
  }
  else
    front ++
}
```

- Circular Queue.



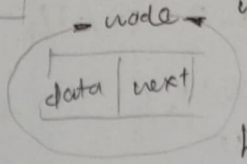
Delet Element from queue

```
If (rear == -1)
Else
{ else [front] = -1 //
```

assume element is delet

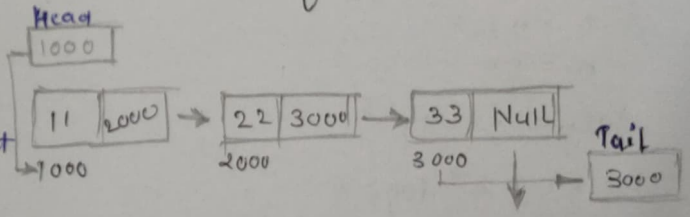
```
If (front == rear) {
  { Rear = -1;
    front = -1,
  } else
    front ++; }
```

Linked list: collection specially design. element call as node.



→ Node is element of Linkedlist which can have atleast 2 member

1. Data.
2. Address of another element.



Last node should be null.

Advantages:

1. optimized usage of memory
2. Size (no of element) of Linked list can be increased or descreased

Disadvantages:

1. In case of linked list each node maintain the address of another element. so memory given for that point which stored address is another against each node

Time Complexity:

It is defined as the number of steps required to solve the entire problem using same efficient algorithum

$$T(P) = C + tp(instance).$$

Here.

T(P) = time complexity of program
C = Compile Time
tp = Run time

* Drop Constant = O(n).

Name of Algorithum	Best Time Complexity	Worst Case Time Complexity	Avg Case
Linear.	$\Omega(1)$	$O(n)$	$\Theta(n)$
Binary	$\Omega(1)$	$O(\log n)$	$\Theta(\log n)$
Selection	$\Omega(n^2)$	$O(n^2)$	$\Theta(n^2)$
Bubble	$\Omega(n)$	$O(n^2)$	$\Theta(n^2)$
Insertion	$\Omega(n)$	$O(n^2)$	$\Theta(n^2)$
Quick	$\Omega(n\log n)$	$O(n^2)$	$\Theta(n\log n)$
Merge	$\Omega(n\log n)$	$O(n\log n)$	$\Theta(n\log n)$

Eg

4, 9, 15, 21, 34, 51, 68, 91 — search = 68

4, 9, 15, 21, 34, 51, 68, 91 - iteration 1 = $n/2$

34, 51, 68, 91 iteration 2 = $(n/2)/2 = n/2^2$

68, 91 iteration 3 = $(n/2)/2/2$

OR

$(n/2^2)/2 = n/2^3$

Iteration $k = n/2^k$

$\therefore 1 = n/2^k$

$n = 2^k$.

$\log_2 n = \log_2 2^k$

$\log_2 n = k * \log_2 2$

$k = \log n \rightarrow O(\log n)$.

$k \rightarrow \log_2 8 \rightarrow \log_2 2^3 \rightarrow 3 * \log_2 2$

$\rightarrow$ 3 iteration

Type of Linked list.

1. Single Linear Linked List
2. Single Circular Linked List
3. Doubly Linear Linked List
4. Doubly Circular Linked List

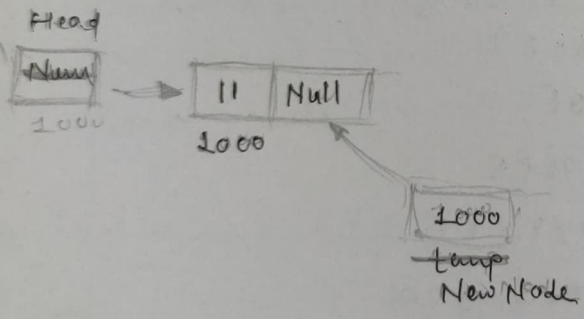
Operation, can be performed on Linked List

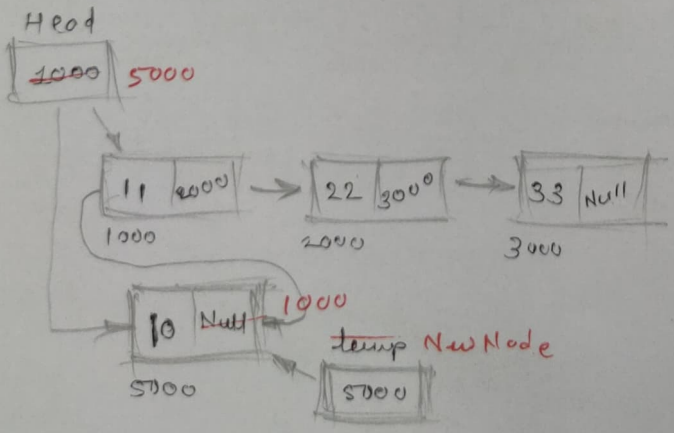
1) add first, add at last, add at pos
2. Del first, del at, del from pos
3. Travers
4. Reverse
5. Merge
6. Sort
7. Etc

Struct Node

```
{ int data,
  Struct node *next;
};
```

Self referential structure is a structure which has atleast one member as pointer which point self Type in which it is declared

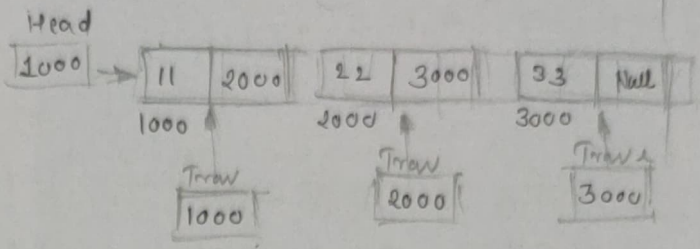




newnode → next = Head.

Head = newnode.

Travers:



node t *trav = head.

while (trav != Null)

{ printf ("%d --> " trav > data),

trav = trav -> next;

} when trav is equal to Null Then program will be stop

O(n) – average is not given

Infix to postfix Conversion.

operands: A, B, C, x, y, z ....
operators +, -, /, *, ^ ....

Rules:

1) priorities of operators:
   - ^ => highest
   - *, / => next
   - +, - => lowest
2. Now two operator of same priority can stay together in stack column
3. Lowest priority cannot placed befor highest priority.

Eg (A + B / C * (D + E) - F)

Convert Infix to postfix

Sym.	Stack	Postfix
(	(	
A	(	A
+	(+	A
B	(+	AB
/	(+, /	AB
C	(+, /	ABC
*	(+, *	ABC/
(	(+, *(	ABC/
D	(+, *(	ABC/D
+	(+, *, (+	ABC/D
E	(+, *, (+	ABC/DE
)	(+, *, (+)	ABC/DE+
-	(+, -	ABC/DE+*+
	some prepite.	
—	(-	ABC/DE+*+
F	(-	ABC/DE+*+F
)	(-)	ABC/DE+*+F-

Infix to Prefix Conversion.

A * B - C / D + E

=> A * B - C / D + E

= (A * B) - (C / D) + E

= (* AB) - (/ CD) + E

= (T1 - T2) + E

= (- T1 T2) + E

= T3 + E

= + T3 E

= + - T1 T2 E

= + - * AB / CD E

* AB = T1
/ CD = T2
- T1 T2 = T3

(A + B) / C * D - E

= + AB / C * D - E

= T1 / C * D - E

= [T1 / C] * D - E

= / T1 C * D - E

= [/ T1 C] * D - E

= T2 * D - E

= * T2 D - E

= (T3 - E)

= - T3 E

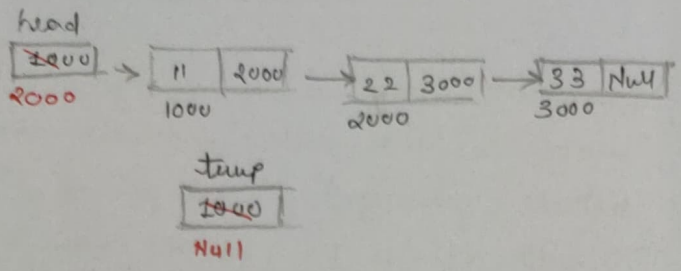
= - * T2 D E

= - * / T1 C D E

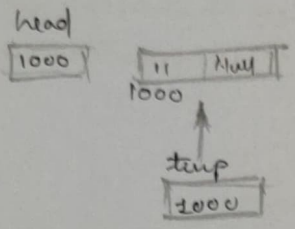
= - * / + AB C D E

+ AB = T1
/ T1 C = T2
* T2 D = T3

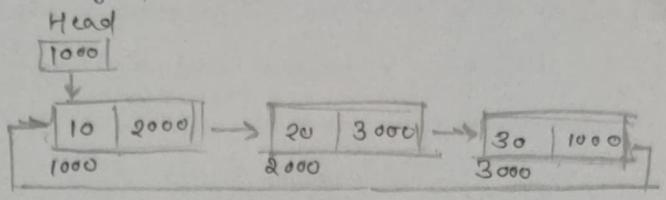
Date - 22/06/2023



node t * temp
temp = head.
head = head -> next
free (temp),
temp = Null

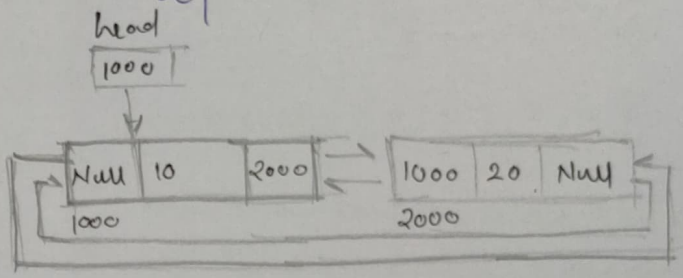


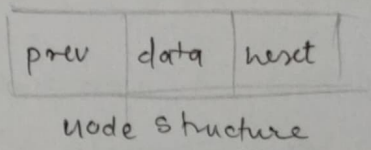
Singly Circular linked list.




```
struct node
{ struct node * prev,
  int data
  struct node * next
}
```

Doubly Circular linked list





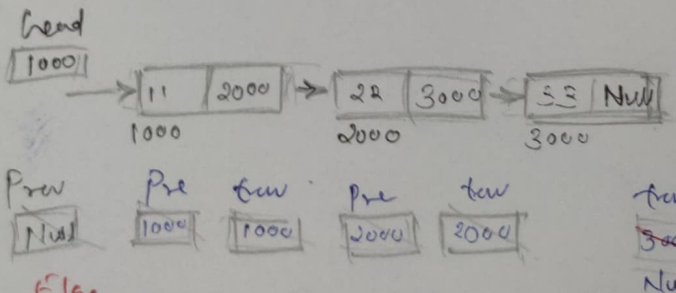
node structure

```
Struct node.
{ struct node * prev
  int data
  struct node * next,
```

Delet from position:

```
trave = head.
p = 1
while (p < pos - 1)
{ trav = trav -> next
  p++;
}
temp = trav -> next
trav -> next = temp -> next;
free (temp)
temp = Null;
```

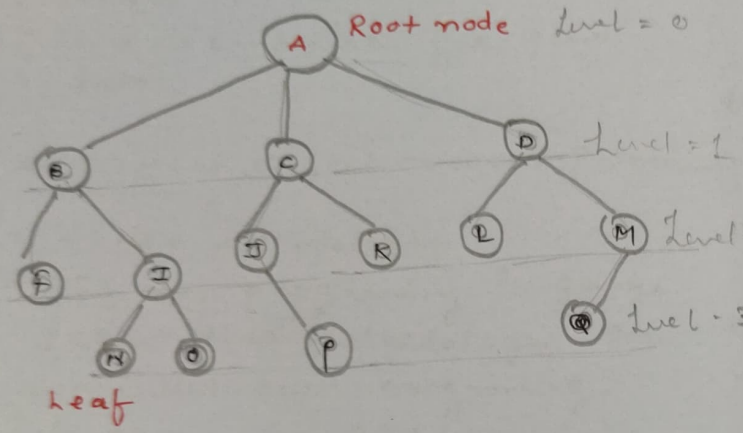
Last element delet.




```
Else,
while (trav -> next != Null).
{ trav = trav -> next
      prev = trav;
}
  prev -> next = Null
  free (trav)
  trav = Null;
If ( head -> next == Null)
    Head = Null
```

Tree and Graph.

Tree: it an non-linear, advanced data structure which is a collection of finite no. of logically related similar types of which element in which. Root element. remaining all element are connect to it in a hierarchial manner.



Root A = level 0.

D, C, B = present level + 1.

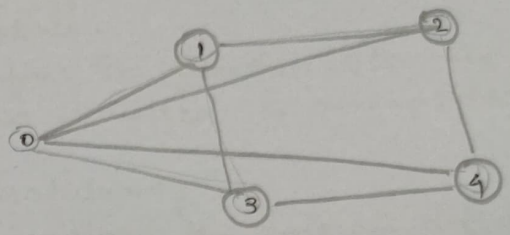
* Binary Tree has Maxm 2 Branches.

degree of Node – No of Branches connected

degree of Tree – A (max braches)

AVL = Balance Tree, Binary Tree

Graph: it is non linear / advanced. data structure, which is a collected of logically related similar and dissimilar types of element which contained set of finite no. of order / unorder refer. as a vertical. also called as node.



G(V, E): V = {0, 1, 2, 3, 4}

prim's & kruskal's are 2 algorithm to find minimum spanning tree of given graph

dijsktra's & bellam ford are the algarithum to find shartest path.

C ++ - 1960. - (9 marks)

Date : 23/06/2023.

1. Bjrane Stroustup.
2. C with classes
- CPP OR. C++.

Limitation of C

- POP > Procedure Oriented Programming
- As the code size increase its complex to maintain such.
- There is no security for your data.

OOP -> Object oriented Programing

- It is a methodology.
- If any programing langeuge follows this methodology then we called such programing langeuge as OOP Langeuge.
- Abstration, Encapsulation, Modularity, Hirarchy

1. Major pillar.
- Abrstration.
- Encapsulation
- Modularity
- Hirarchy.

- Minor pillar
- Typing / palymorphsim
- Concurrency
- Perisestence

* Following all the major pillar is compulsary for any programing langeuge That term itself as OOP language.

Abstration:
- knowing only the essential thing
Eg :) call given to the funcation
- print() , scanf().

Encapsulation.
- Binding the data and code together.
- define the function, structure, class is called as encapsulation.

Modularity.
- writing the code in different modules.
- Multiple function or multiple file

Hierarchy:
- classes

Polymorphsium (Typing)
- An entity that take multiple different forms is called polymorphsium.

Eg -> mobile.

Persistence.
- Storage.
- to persist the data permantely.
- Eg. file i/o.

Concurrency:
- the exection should be concurrent.

Datatype in CPP.
- bool -> boolean - 1 bytes
- wchar_t -> wide character - 2 bytes

Date: 24/06/2023

Structure in C.
- we cannot write function. inside structure

Structure in CPP
- we can write function. inside structure

Access specifer. in structure
- public.
  all member that are public. will be accessiable directly on the object outside of structure
- private
  all member that are private will not be accessiable outside. the structure
- By deafult member of the structure are public.

Name Space.
- It is a container that can store variable, function, structure classes.
- we cannot create object of the namespace
- To access the member of the Namespace we have to use an operator called as Scope resolution.

Cin and Cout
- Cin:
  it is an object of istream class.
  - (>>) extraction operator in Cin
- Cout
  - it is an object of ostream classes.
  - we need to use insertion (<<) operator with cout

Function Overloading
- it is an example of Compiler Time polymorphism
- To overload the function. we need to keep the name of the function same but sign. should be different
- No of parameter

How to use Cprog in C++.

//# include <stdio.h> // C prog.
OR
include <cstdio> // Cpp way.
include <iostream>.

Default argument (right to left)
- given in declaration or definition

Class and object.

What will be size of Empty class is → 1 bytes (8 bites).

-format specifer :- %p.

Class :- private.
Member function: public
which.

Type of member function
1. facility of input/output
2. ctor.
3. dtor
4. Mutator / Setter method.
5. Inspector / getter method.

Constructer
it is Special member of class having same name of the class. it doesn't have any return type not even of void.

It get automaticaly call. when the object is created.

There are 3 type of Constructer's

1) Default - storeage class
2) Parameter-less - no-argument cter
3) Parameterized.

If we donot write any constructor compiler give us a default constructor and Default constructor initiles data depending on the Storage class.

* Can you overload the Constructer :
Yes

Distructer :

distructor is the special member of the class which has same name of class. and started with "~"

It donot have any argument & return type it get automatically call. when the object going to be destroy.

The Constructor & distructor Call sequnce is exactly opposite for local object (local object create on stack and functionality is FILO / LIFO.

Stack lifo / fifo.

Mutator modify State of object (setter meth).

Inspector dont modify state object

// in C prog we can change the value of constant using pointer

* Can you overload Distructer.
NO!

friend can assess private / public / protected out side class.

* If you want function defination outside of class we have to use scope resolution operator. with following sytax.

{return-type classname::
member-fun-name (data-type parameters) }

Date : 26/06/2023.

// demo of constant data member, function and constant object.

```
#included <iostream>.
using namespace std;
namepase NConstDemo.
{
  class Demo.
  {
    // variable // data member / field.
    private:
      int a; // non-constant data mem
      int b;
      const int c; // constant data mem
      int const d;
    public
      ConstDemo ()
      {
        this -> a = 10
        this -> b = 20
        this -> c = 30
        this -> d = 40
      }
};
// end. of class Const Demo.
// end of namespace NConstDemo
```

// when we declear data member as constant it is compulsary to the initailzed. that data member inside Constructor. initializer. list

// No const can call both. constant data member, Non-constant data member

// constant used at Construction funcation.

// If you want to modified. state of object multiple

g++ demo.cpp

Static data member & Static member function.

Static member fun is designed to called on class name.

In cpp we can call. Static member fun on class name & object name. also

// which function have this pointer // all data function has the. pointer except static member function.

// which function donot have this pointer
1. Global function.
2. friend function.
3. Static member function.

// that static cannot be called on class name but we can called object name

// size of object is sizeof all. non-static

// when you static data global defn. is necessary

// static function aprate only the. static data.

Dynamic Memory Allocation.

valgrind is tool check memory leakage (Linux/mac).

Aggregation is tight coupling

There are some operator we can not overload There can we overload Then as friend function

→ Insertion & Extraction

Date: 27/06/2025

// g++ demo1.cpp -> a.exe.

// g++ Frienddemo

Inheritance

- default mode of inheritance is private.

If you are able to create object of a class it is called as Concrete class.

If you not able to create a object of class it is called. Abstract class.

RTTI : Run Time Type Info.

STL: Standard template libr.

// when we assign allready created object A1 to newly create a2 then internally copy ctor will be called for newly create object (a2)

// when we don't

custom Compiler used for this Run Copy Constructors

Computer fundamental. Date: 28/06/2023.

CPU
Central Processing Unit.

Input & Output devices.

Hardwar devices & Softwar.

Program:

1) User program:
democ, test.cpp, myprogram.
· Java, student.docx etc.

2) Application program:
VS code, chrome, word, excel, paint.

3) System.
Loader, Linker, deriwers. einterrupts.

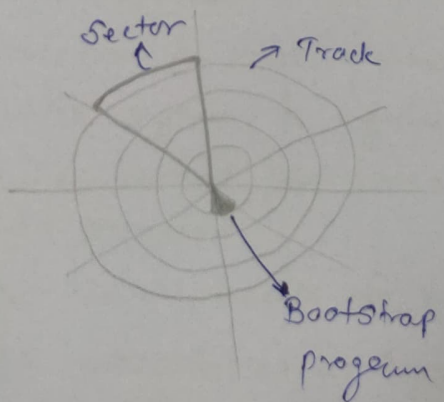
Booting
Program & Process:

Any ~~pro~~ memory program which is in the RAM is called as a process

Locating and Loading OS kernel in the main memory.

↓

Bootstrap program.



Booting process:

1) Machine Boot:
BIOS (Basic Input & Output System).
- micro-program.
- POST (power On Self Test)
- Bootstrap loader.

2) System Boot.
- Boot loader Program.
call/invoke to Bootstrap program (Bootstrap program located the kernal and load it into the main memory).

The Sector in which bootstrap program is present is called as BOOT SECTOR

The devices in which boot sector is present that devices is called. BOOTABLE DEVICE.

CPU - Central Processing Unit

CPU - Registers
- Limited
- intermedia. operation used by CPU
- faster.

Address.
1000 Start
Loop: add a, b
1002 mov a.
1003 add a, b
1004 Stop

Instruction Register: add a, b

Instruction Pointer: 1001

ALU (Arithmetic & Logical Unit)
- arithmetic, logical, bitwise, condional .. etc.

Control unit.
Instruction. Cycle.
1) fetch
2) decode
3) execute

Bus. (System Bus)
1) Control read / write.
2) Address 4000
3) Data 25

num. 25 | 4000

Programmed IO (Input Output)
- processor is checking status of devices again & again
- devices is not communicating with processor
- cpu involvement is more
- cpu utilization is less.

Interrupt. Drieven IO.
- devices send interupt signal to cpu
- cpu involvement is comparatively less
- cpu utilization is better than programed IO.

1) ISR (interupt Sevices Routine)
2) IVT (interupt Vector Table)

DMA (Direct Memory Access)
- CPU Utilization is less
- CPU involvement less

RAM - Random. Access. Memory
- used for Read / Write
- Volatile (Temporary Storeage)
- (SRAM & DRAM)

ROM - Read Only Memory
- we can only read & not write
- This type of memory is non-volatil
- A ROM Storage such instruction. that are required to start a. computer. This operation is refered to as boatstrap.

1) first Generation - Vaccum Tube
2) Second :- - Transistor
3) Third :- - Intergrated circuit
4) fourth :- - Microprocessor
- CPU.

Operating System.

Process management.
Memory management.
File management.
Disk Management.
Input Output Management (IO)

---

* Protection and Security.
* Networking
* User Interfacing.

An OS is system Software. which acts as the interface betn user and Hardware (collection of program)
- Resource manager
- Control program
- Resource allocters.
- System Softwar.

Kernel: it is core program / part of an OS which run continuously into the main memory does basic minimal function of it.
* Eg: * Linux, Window, ~~vmlimuz~~.
Linux: vmlimuz
Window: ntoskrnl.exe

Need of OS
- Data storage
- Data move

CUI - (Command User Interface / Command Line Interface).

GUI - Graphical User Interface.

gcc - GNOME / KDE
GNU Complier.

* Window - CUI => cmd.exe.
Linux CUI => shell / terminal

* Window GUI => explorer.exe.
Linux. GUI => GNOME / KDE

Dual Mode operation.

* file format of executable. file in Window is PE (Portable Executable)
* whereas file formate of an executable. file in Linux ELF (executable & linkable. format).

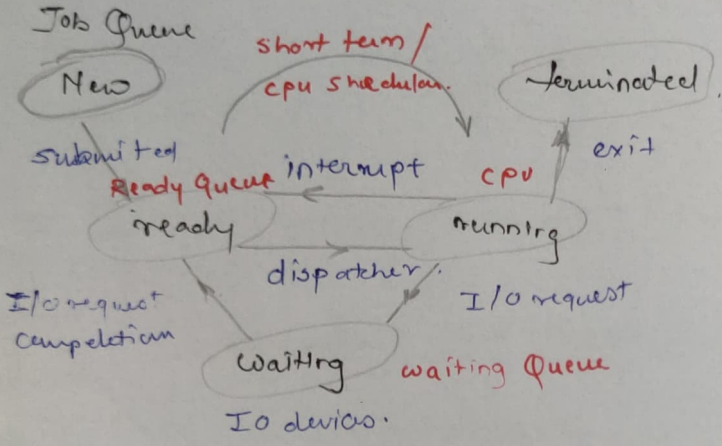
Date : 29/06/2023.

Process Management.

Program.

1) User program.
2) Application Program
3)

Process Control Block (PCB)



Process State Diagram

Context Switch

Context switch = State Save + State restore.

State save: to save execution context Suspended with PCB

State restore: to restore execution.

Process Scheduling:

- preemptive – Take down program
- Non-preemptive – Not Take down. program

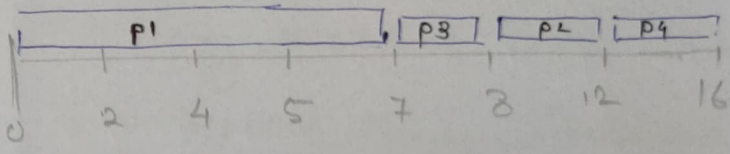
First come, first served.

CONVOY Effect:
Shorter process behind the long process

Shortest Job First.

Non-preemptive SJF

Process	Arival	Burst Time
P1	0.0	7
P2	2.0	4
P3	4.0	1
P4	5.0	4



P1	P1	P1	P1	P2=4	P2=4	P4=4
	P2	P2	P2	P3=1	P4=4	
		P3	P3	P4=4		
		P4	P4			

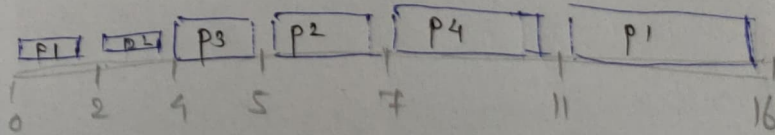
Waiting Time (TT - BT - AT)

P1 = 0 - 0 = 0
P2 = 8 - 2 = 6
P3 = 7 - 4 = 3
P4 = 12 - 5 = 7

Waiting (TT - BT - AT)

P1 = 11 - 0 - 0 = 0
P2 = 5 - 2 - 2 = 1
P3 = 4 - 4 = 0
P4 = 7 - 5 = 2

Preempit SJF.



P1=7	P1=5	P1=5	P1=5	P1=5	P1=5	P1=
	P2=4	P2=2	P2=2	P2=0	P4=0	
		P3=1	P3=0	P3=0		
			P4=4	P4=4		

Starvation : solution is aging

Starvation - low priority processes may never execute

Aging :- as time progress increase in the priority of process

Raund Rabin (Preemptive).
Time Quantam = [20]

Multilevel feedback queue. Scheduling.

Inter process Cammunicatian. (IPC)
- Independent and Cooperating

Process Syuchronizatian.
- Critical Sectian.
- Critical Sectian Problem (CSP)
- 1. Semaphare - binary, Counting
- 2. Mutex Object - lock and unlock.

Dead Locks:
- Mutual exclusian
- non-preemptian
- circular wait
- Hold and wait.

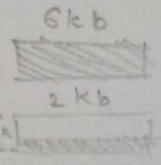
* Algorithum.
1. Banker's Algorithum
2. Reasource Allocatian Graph (RAG)
3. Dead Recavery
- process Terminatian.
- Reasaurce Pre-emptian

Method of Handling deadlock.
- Deadlock prevention
- Deadlock avaidance.

Memory Management

#Swapping is also knew as process af campacsian.

Fragementation.
External fragemantatian.
Internal fragmentatian



Memary Allocatian. Types
1) Contigauous Memary Allocatian
- fixed & Dynamic. Spartion.
2) Non-cantiguous Memary Allocatn.
- Paging
- Segmentatian.

Paging is used to avaid the External fragmentatian.

Segmantatian:
- avaid the internal fragmentatn

Virtual Memory.

Virtual memary is implemented using demand. paging or demand Segmentatian.

If any page is present in the main memary, then its entry in the page table in the page table is kept as "V" bit. Vindicate valid bit

Date: 30/06/2023.

If any page is not present in the memory it set as "i" bit invalid bit.

If we wish to swap out some page from main memory because main memory is full, then that page is called as VICTIM page. ~~then that page is~~ (Page Replacement policy)

Page Replacement Policy.

- FIFO.
- LRU, (Least Recently used)
- Optimal (future Refrence String)

File and Storage Management

File system is the way to store data on the disk in an orgained manner so that it can accessed efficiently

* UNIX : Ufs (UNIX file syst).
  Linux : Extended file sys. ext2, ext3
  Window : FAT, NTFS
  MAC OSX : HFS (Hierrarchical filesystem)

* FCB (file control Block)/ inode (index node)

each file is controlled by the exactly ONE inode

Inode that the contains the mode of all the file in the file system

File System Storage.

1. Boot Block / Boot Sector.
2. Super Block / Volume Control Block
3. Inode list / master file Table.
4. Data Block.

File Allocation on Disk

1. Contiguous
2. Linked
3. Block or indexed.

Contiguous:

- internal fragmentn: file is not given to used whole data block give on it.
- External fragmentatn: number of block needed for the file are available but not contiguous

Linked:
Each block contain data and address of its next block.

Indexed Allocatn:

* Disk Scheduling

1. First Come first Serve (FCFS)
2. Shortest Seek Time first (SSTF)
3. SCAN (Elevator).
4. Circular Scan (C-SCAN)