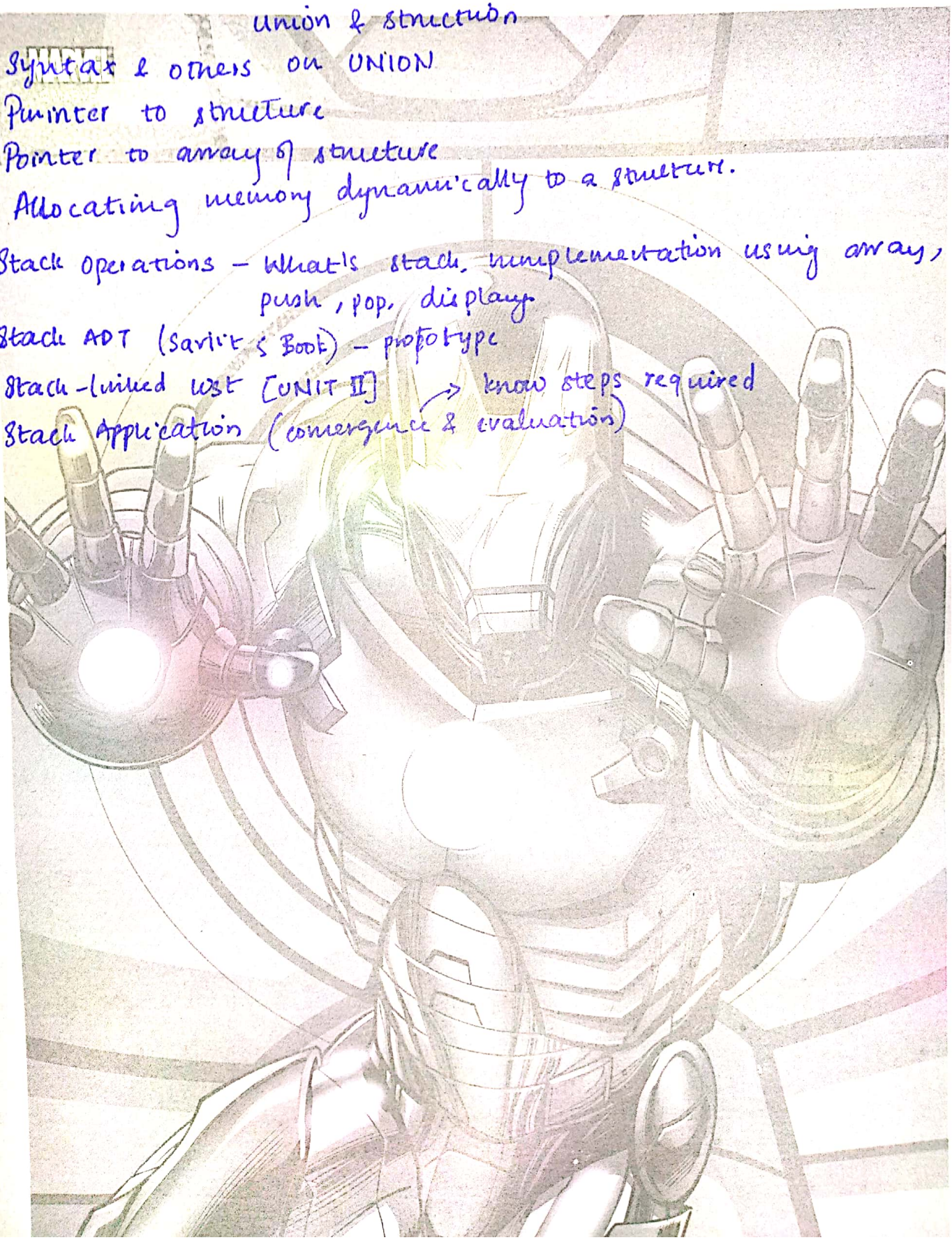
1) Difference b/w calloc() & malloc()
2) union & structure
3) Syntax & others on UNION
4) Pointer to structure
5) Pointer to array of structure
6) Allocating memory dynamically to a structure.
7) Stack Operations – What is stack, implementation using array, push, pop, display
8) Stack ADT (Savitch & Book) – prototype
9) Stack-linked list [UNIT II] → know steps required
10) Stack Application (convergence & evaluation)



__/__/__

UNIT I

Pointers - is a variable, that contains address of another variable (data variable) or address of another point variable

Declaring pointer → syntax : datatype *variable
Initializing → int *p1 (points value)
p1 = &a (points addres)

*p1, *p2 → o/p 5, 6
&p1, &p2 → addres
p1, p2 →

// WAP a to implement a simple calculator using pointers

```
#include<stdio.h>
void main()
{
  int a, b, *p1, *p2;
  printf(
  scanf(
  printf("Enter choice")
  scanf("%d",&n)
  p1 = &a, p2 = &b;
  switch(n):
```

* Every array acts as a pointer

__/__/__

// Write a C program to find sum and mean of given array elements using pointer

→
```
#include <stdio.h>
void main()
{   int n, ar[], sum=0, i;
  printf("Enter the size of array");
  scanf("%d", &n);
  printf("Enter the elements");
  for(i=0; i<n; i++)
    { scanf("%d", &a[n]); }
  float mean, *p;
  p = &a[0]  OR  p = a;
  for(i=0; i<n; i++)
  {
      sum += *p;
      p++;
  }
mean = sum/n;
printf("%d %d", sum, mean);
```

storing address of first element in p

value where p is pointing

p++; → incrementing to next value of the array.

fread (a, sizeof(char), 20, ptr)

fscanf (ptr, "%c", &a)

fprint

__/__/__

```
case 1 : *p1 + *p2 ; printf("   ");
         break;
case 2 : *p1 - *p2 ; printf("   ");
         break;
case 3 : printf("   ");
         *p1 * *p2 ;
         printf("   "); break;
case 4 : printf("   ");
         *p1 / *p2 ;
         printf("   "); break;
default : printf("   ");
}
```

① Dangling pointer : pointer having invalid address

② Null pointer : involving no add.

ex. ① int *p1, *p2

② int *p;

p = null

* Pointer & Arrays

1) Every array is a pointer
2) Value of array variable is equal to add. of 1st element of the array.
3) Elements of the array can also be accessed using pointer [*p1 + 1 ie index (1)]

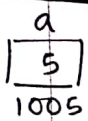
__/__/__

* Pointers & Functions

// WAP to swap contents of two variables using pointers

→

```
#include <stdio.h>
void exchange (int *m, int *n);   Declaration / prototype
void main()
{
    int a,b;
    printf("Enter 2 numbers")
    scanf("%d%d", &a, &b)
    exchange (&a &b);   Passing by address
    printf("%d %d", a, b);
}
                                  passing parameters by value
void exchange (int *m, int *n)
{
    int temp;
    temp = *m;
    *m = *n ;
    *n = temp;
}
```



*p → 5 → *p
p → 1005 → p = &a (addren of a)
&p → addren of p (pointing a pointer)

* Function returning a pointer

```
#include <stdio.h>
int *display (int *c);          because this function will return only a pointer
void main()
{
   int a; int *b;
   printf("Enter value of a")
   scanf("%d", &a)
   b = display(&a)              returns what c holds
   printf("%d", *b)
                                ↳ value will be printed ie 5
}
   int *display (int *c)
   {
      return c;  addren of a
   }
```

* Pointer to pointer

A variable that contains the addren of another Pointer variable is called as pointer to pointer

Syntax : int var, *p1, **p2;

```
var = 5;
p1 = &par;
p2 = &p1;
```

p2 → points value of var

To acces, *p1 → 5 → points value in p1

**p2 → 5

__/__/__

* Structures

A structure is a collection of one or more variables of same data type or dissimilar data type grouped under a single name

→ Declaring a structure

There are 3 ways : 1> Tagged structure
2> Type defined structure (Type-def using keyword)
3> Structure without tag

→ Tagged Structure

Declaring tagged structure consists of two parts
1> Structure defination
2> Defining or declaring struct. variables

* Syntax for struct. defination

```
struct tagname
{
   datatype1 variable1;
   datatype2 variable2;
    :
};
```

ex.
```
struct student
  { char name[50];
    char USN[10];
    int marks;
  };
```

__/__/__

* Syntax for defining or declaring struct.

struct tagname var1, var2 ... ;
↳ If variables are declared outside the structure

ex. struct student cse;
↳ memory is allocated ↳ it is of type structure then (44 bytes for 32 bit machine)

→ Type - def structure [typedef]
1> structure defination
2> Defining / declaring struct. variables

*Syntax for 1>
struct typedef
{
    datatype1 var1;
    datatype2 var2;
} tagname;

ex struct typedef
{
    char name[50], USN[10];
    int marks;
} STUDENT;

* Syntax <11>
tagname var1, var2 ... ;

ex
STUDENT cse, ise;

__/__/__

→ Structure Initialization

* Two types — ① Compile time ② Run time

* Syntax

for compile time →

```
struct tagname variablename = { value1,
        value2 ... valuen};
```

ex

```
struct STUDENT cse = {"Nikita", "1BU3915",
  98 };
```

run time initialization →

```
struct tagname variablename;
```

ex

```
struct STUDENT cse;
scanf ("%s", cse.name);
scanf ("%s"), cse.USN);
scanf ("%d", &cse.marks);
```

//WAP to store the following information & print the same using structure.

roll no., name, marks, grade

→

```
#include<stdio.h>
struct STUDENT {
        int roll no, marks;
        char grade, name[50];
};
struct STUDENT ise;
```

_/_/_

```
void main ()
{
    printf(" Entre the roll no. ");          } ①
    scanf( "%d" , &cse.rollno);              }
    printf (" Enter the name ");             } ④
    scanf( "%s", cse.name ");                }
    scanf ( "%d", &cse.marks ");             ②
    scanf ( "%c", cse.grade ");              ③
}
    printf("%d %s %s %d", &cse.rollno, &cse.marks, cse.grade,
                cse.name);
```

array & string don't have & while scanf

→ Accessing structure
    varname. memofstruct

→ Array of structure : struct STUDENT cse[100]

//WACP to read & display following details of 'n' students using structure - roll no, name, USN, marks, grade

```
→ #include<stdio.h>
struct STUDENT{ int rollno, marks;
                char grade, name[50];
              };
int n; ←
struct STUDENT cse [50];
void main()
{  printf(" Enter no. of students ");
   scanf ("%d", &n);
```

```
printf(" Enter values for the following rollno, marks,
 name, grade ")  → for(int i=0; i<n; i++)
scanf(" %d %d %s %c", &cse[i].rollno, &cse[i].marks, cse[i].name
    cse[i].grade)
printf("    ", );  → for(i=0; i<n; i++)
}
```

//WACP to maintain a record of 'n' employee details & print details — employee id, employee name & salary. of employees whose salary is above 5000.

```
#include <stdio.h>
struct EMPLOYEE {
                    int id, salary;
                    char name[50];
                  };
struct EMPLOYEE ibm[100];
void main()
{   int n, i;
    printf(" Enter no. of employees ");
    scanf(" %d", &n);
    printf ("Enter values for id, salary & name ");
    for( i=0; i<n; i++)
        { scanf ("%d %d %s", &ibm[i].id, &ibm[i].salary
          , ibm[i].name; }
    for(i=0; i<n; i++)
     if(ibm[i].salary >5000)
}      printf("    ", );
```

Difference b/w UNION & Structure.

typedef struct structname

__/__/__

* UNION

It is a collection of one or more variables of same datatype or dissimilar data type.

→ Syntax for declaring

```
union tgname {
        type1 var1;
        type2 var2;
        type3 var3;
};
```

Union	Structure
When a variable is associated w a union the compiler allocates the memory by cons-idering size of largest member. The size of union is equal to the size of largest memory.	When a variable / mem is associated w a structure the compiler allocates the memory for each member. The size of a structure is greater than or equal to then sum of sizes of its members
Memory allocated is shared by individual members of UNION.	Each member within a struct is assigned w unique storage area
The add. is same for all the members of structure.	The address is differe for each member of structure

int a in 32 bit machine 4 bytes

__/__/__

Altering the value of a member will alter the value of another.	Altering the value of a member will not alter the value of other mem.
Only one member can be accessed at a time.	Individual members can be accessed at a time.

⇒ Dynamic memory management in C

1> malloc() - It is used to allocate exact amount of memory needed during execution

Syntax: ptr = (datatype *) malloc (size)

ptr → pointer variable

size → size to allocate during runtime

→ always returns a void pointer or generic pointer by default.

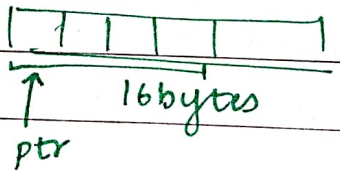
*It can return null value

ex: int *ptr;

ptr = (int *) malloc (4);

type casting/conversion

```
int *ptr;
ptr = (int *) malloc (4 * sizeof (int));
*ptr[0] = 5;
```



//WAP a C program to find sum of even and odd numbers in the given set of elements using malloc()

```
#include <stdio.h>  #include <stdlib.h>
void main()
{
```

Difference b/w malloc & calloc

* Not sufficient memory

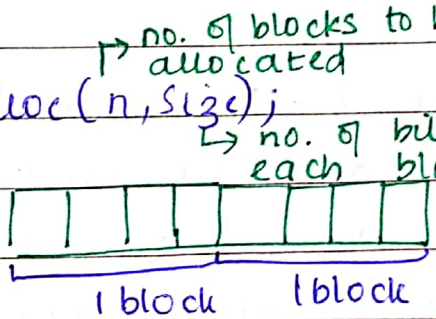
__/__/__

```
int n, ar[50], osum=0, esum=0, i;
printf("Enter the size of array"); printf("\n");
scanf("%d", &n)
int *ptr;
ptr = (int *) malloc (n * sizeof(int));
if(ptr == NULL)
printf("*")
exit(1);
}
printf ("Enter the array");
scanf ("%d", &ar[i]);
for(i=0; i<n; i++)
for ( i=0; i<n; i++)
{
    if (ptr[i] % 2 == 0)
        esum++; esum += *ptr[i]
    else
        osum++; osum += *ptr[i]
}
printf("The even sum is %d", esum);
printf("The odd sum is %d", osum);
}
```

2) calloc() : It used to allocate memory dynamically and used for allocating multiple blocks of memory. Mainly used for arrays.

syntax : ptr = (datatype *) calloc (n, size);

ex : ptr = (int *) calloc (2, 4)



__/__/__

//WAP to read & display marks of 3 subjects using calloc() function

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
   int s1, s2, s3, *ptr;
   printf("Enter the marks \n");
   ptr = (int*) calloc (3, 4)
   scanf ("%d %d %d", &ptr[0], &ptr[1], &ptr[2])
   printf ("The marks are %d %d %d ", *ptr[0], *ptr[1],
                                      *ptr[2]);
}
```

3> realloc() : Is used to extent the size of already allocated memory or to delete memory at the end of the block.

Before using this function the memory should be allocated using malloc() or calloc()

Syntax: ptr = (datatype*) realloc (ptr, size); → new size

4) free() : This function is used for deallocating the allocated blocks of memory. (specially allocated memory by malloc(), calloc() or realloc().

Syntax : free (ptr);

20/8/19

Malloc()	Calloc()
1) It doesn't initialize allocated memory	It initializes the allocated memory to zero.
2) It takes only one argument i.e., size	It takes two arguments, number & size.
3) Efficiency is higher than calloc()	Efficiency is less.
4) Allocates memory even if memory is not available continously	Allocates required blocks of memory contagiously

UNIT II - Files

* **Files**: A collection of data that is stored on secondary storage device.
  ex. BATCH file, DAT file, binary files, txt file.

* **Txt file**: The one in which the data is stored as sequence of characters that can be procesed sequencially.

* **Modes of files**: r, w, a (append)
  1) read mode - The file pointer will be pointing at the first character & the EOF() inbuilt function at the last character
  2) write mode - file pointer & EOF() will be pointing at the same first character
  3) append mode - both FP & EOF() will be pointing at the last character.

* s->USN, s->Name, &s->rollno, &s->marks

__/__/__

else (ch == '\t')
    t++;
}
}
printf("Characters: %d \n NewLine: %d \n Tab: %d",
    c, n, t);
}

Termwork 1

//WACP to copy the content of text file of integers into another file & print the same on the screen

Pointers to structure

//WACP to store details of student such as name USN, roll no & marks, using pointer to structure.

void main()
struct STUDENT {
    char USN[], Name[];
    int rollno, marks;
};
struct STUDENT obj; struct STUDENT *s;
void main()
{ s = &obj;
printf("Enter USN, name, rollno & marks: \n");
scanf("\r%s \r%s %d %d", obj.USN, obj.Name, &obj.rollno, &obj.marks);
printf("USN, Name, rollno, marks %s %s %d %d", s->USN, s->Name, &s->rollno, &s->marks);
}

* &s → rollno.obj[i] } can be written like this
* s → obj[i]
* (s+i) → name or &(s+1) → marks

Pointer to array of structure

// WAP to store details of 'n' students & print the same using pointer to array of structure concept.

```
Struct STUDENT {
        int rollno, marks;
        char USN[20], name[20];
    };
Struct STUDENT obj[20]; → 20×48
                          960 bytes
Struct STUDENT *s;
void main()
{
    int n; s = &obj[0];
    printf("Enter no. students: ");
    scanf("%d", &n);
    for(int i=0; i<n; i++)
    {
        scanf("%d%d%s
        s = &obj[0];   %s", &s→obj[i], &s→obj[i],
                       s→USN.obj[i], s→name.
                                     obj[i]
    }
    for(int i=0; i<n; i++)
    {
      printf("      ")
    }
}
```

```
int index=0
max = (ptr+0)→marks;
for(
if((ptr+i)→marks>max)
max = (ptr+i)→marks;
index = i;
}
```

```
printf("Highest mark");
printf("%s%s%d", (ptr+index)→name
        ...USN, ...roll)
```

```
scanf("Enter details: ");
```

__/__/__

Allocating memory systamatically to a structure

// WAPP to store details of a structure such as name USN marks using struct. Allocate memory dynamically to a struct.

```
#include <stdio.h>
struct STUDENT
{   char name[20], USN[20];
    int marks;
};
struct STUDENT *ptr;
void main()
{
  s = (struct STUDENT *) malloc (sizeof(struct STUDENT));
  printf ("Enter the student's name, USN, marks \n");
  scanf ("%s %s %d", ptr → name, ptr → USN, &ptr → marks);
  printf ("%s %s %d", ptr → name, ptr → USN, &ptr → marks);
```

// WAPP to store details of n students using structure & allocate memory dynamically to a structure

```
#include <stdio.h>
struct STUDENT
{
  char name[20], USN[20]; int marks, n i;
};
struct STUDENT *p;
void main()
```

notes → 3 topics are using
ptr → str
ptr arr of str allocating mem dy to str.

```
{
  p = (struct STUDENT *) malloc (n* sizeof(struct STUDENT));
  printf ("Enter no. of students ");
  scanf ("%d", &n);
  printf ("%d", &n); ("Enter details of students");
  for ( i=0; i<n; i++)
  {
     scanf ("%s%s%d", (p+i) -> name, (p+i) -> USN,
     (p+i) -> marks);
  }
  for ( i=0; i<n; i++)
  {
  printf ("%s%s%d", (p+i) -> name, (p+i) -> USN, (p+i) -> marks);
  }
}
//WAcp like above but print information of student w
 highest marks.
#include <stdio.h>
struct STUDENT
{
   char name [20], USN [20];
   int marks, n, i;
};
struct STUDENT s;
void main()
{
```

__/__/__

```
s = (struct STUDENT *) malloc (size of (struct STUDENT));
printf ("          ");
scanf ("%d", &n);
printf ("          ");
for (i=0; i<n; i++)
{
    scanf ("%s%d%d", (s+i)→name, (s+i)→USN, &(s+i)→ma);
    max = (s+0) → marks;
}
for (i=0; i<n; i++)
{
  if (max < (s+i) → marks)
  {  max = (s+i) → marks;
     index = i;
  }
}
printf ("%s%d%d", (s+index) → name, (s+index) → USN ...);
}
}
```

Data structure : A method of storing data so that it can be accessed & used efficiently.

__/__/__

check gallery

III Unit

STACK

It is a data structure where elements can be inserted from one end & deleted from the same end.

LIFO ; <1> push <2> pop <3> display

```
#include <stdio.h>
#define maxsize 3
int s[maxsize], top = -1;
void main ()
{
   int item;
   printf("Enter item to be inserted ");
   scanf ("%d", &item);
   if (top == (maxsize-1))
   {
      printf(" Stack is full");
      exit(0);
   }
   top + = 1;
   s[top] = item;
}
```

For stack = full for (i=0; i< maxsize+1 ; i++)

Stack ADT

Objects A

Functions

* Stack cre

maximu

* Boolean

* Stack

* Boolear

* Elemen

__/__/__

Stack ADT Objects:

~~Objects~~ A finite ordered list with 0 or more elements

Functions: for all stack ∈ stack, item ∈ element,
maxsize ∈ positive integer

* Stack creates (maxsize) ::= Create an empty stack whose maximum size is maxsize

* Boolean isFull (stack, maxsize) ::=
    if (no of ele in stack == maxsize)
        return TRUE
    else
        return FALSE

* Stack Push (stack, item) ::=
    if (isFull (stack)) stackfull
    else insert item into top of stack & return

* Boolean isempty (stack) ::=
    if (stack == creates (maxsize))
        return TRUE
    else
        return FALSE

* Element pop (stack) ::=
    if (isempty (stack)) return
    else remove & return element on the top of stack.

__/__/__

Applications of Stacks

1) Convergence of expression
2) Evaluation of expressions

Convergence of expression

* Precedence & associativity of operators

Operator	Priority	Associativity
^ , $	6	R → L
*	4	L → R
/	4	L → R
%	4	L → R
+	4	L → R
-	2	L → R

Precedence of an operator: The order in which the different operators are operatored.

* Converting exps from infix → postfix

Q) A + B * C
A + BC*
ABC*+

Q) A * B + C * D
AB* + CD*
AB* CD* +

Q) A + B + C + D
AB + CD + C + D
AB + CD + C + D +

Q) (A+B) * (C+D)
AB+ CD + *

Q) A + B * (C+D)
A + B * CD +
A + BCD + *
ABCD + * +

__/__/__

Q> (A + (B-C) * D) * B+C*D

(A + (BC-) * D) * B+C*D

(A + BC-D*) * B+C*D

(ABC-D*+) * B+C*D

ABC-D*+B* + CD*

ABC-D*+B*CD*+

<Q> ((A+(B-C)*D)^E+F)

(A+(BC-)*D))^E+F)

(A+(BC-D*))^E+F)

A+(BC-D*)

(A+(BC-D*))^

{ABC-D*+}^E+F

E(ABC-D*+)^+F

EF(ABC-D*+)^+

ABC-D*+E^+F

ABC-D*+E^F+

<Q> X^Y^Z-M+N+P/Q

X^ZY^-M+N+P/Q

XZY^^-M+N+P/Q

XZY^^-M+N+PQ/

XZY^^-MN++PQ/

XZY^^-MN+PQ/+

~~XZY^^MN+PQ/+-~~

XYZ^^M-+N+PQ/

XYZ^^M-N++PQ/

XYZ^^M-N+PQ/+

<Q> (A+b)*d/ + e/(f+a+d)+c

<Q> 4+3-5*6/2

<Q> a+b-(c*(a+b)/d)

<Q> (A+B-(C*(A+B)/D)))

__/__/__

Convert following infix expressions to post-fix using tabular methods.

1) x + y

Input	Stack	O/P
x		x
+	+	
y	+	xy
		xy+

2) x + (y * z)

I/P	Stack	O/P
x		x
+	+	x
(	+ (	x
y	+ (	xy
*	+ (*	xy
z	+ (*	xyz
)	+	xyz*
		xyz* +

3) ~~xy~~ *x*y+z

i/p	stack	o/p
x		x
*	*	x
y	*	xy
+	+	xy*
z	+	xy*
		xy* z+

__/__/__

4) (A+B) * (C+D)

I/P	Stack	O/P
(	(	
A	(	A
+	(+	A
B	(+	AB
)		AB+
*	*	AB+
(	*(	AB+
C	*(	AB+C
+	*(+	AB+C
D	*(+	AB+CD
)	*	AB+CD+
		AB+CD+*

5) A*B+C*D

I/P	Stack	O/P
A		A
*	*	A
B	*	AB
+	*+	AB*
C	*+	AB*C
*	*+*	AB*C*
D	*+*	AB*C*D
		AB*CD*+

__/__/__

*Points to be noted while converting exp from ifix pfix

1) Define priorities of operators on the stack & opr. in the i/p string
   (•) If the opr. are left to right anociated then stack precedence is higher than i/p string.
   (•) If the operators are R→L then the stack precedence is lower than the i/p string
2) Scan the i/p character ifix exp. one @ a time
   i) if the character is (push on the stack.
   ii) if the char is operand push to pfix string
   iii) if the char is operator & stack is empty push charac. on the stack.
   iv) if the precedence of the oper character on top of the stack is greater of equal to to the i/p character then pop the operator from top of the stack. & write it to the postfix string & push the i/p char to the stack.
   v) if less then push the operator to pfix string
   vi) if i/p char is) then pop element of stack till you reach the corresponding) (on stack.
3) Pop all the elements of the stack into postfix string once you reach the end of string.

//WAP
int
{
sum
{
int
{ sum
{

__/__/__

```
//WAP to convert infix exp to postfix.
int sp(char c)
{
   switch (c)
   {
      case '#' : return -1;
      case '(' : return 1;
      case '+' :
      case '-' : return 3;
      case '*' :
      case '/' : return 5;
      case '^' : return 7;
   }
}
int ip(char c)
{ switch (c)
   { case 'C' : return 6;
      case '+' :
      case '-' : return 2;
      case '*' :
      case '/' : return 4;
      case '^' : return 8;
   }
}
```

__/__/__

```
void convert (char infix[], char postfix[])
{
   int i=0, j=0;
   char symb;
   int top =0;
   char stk[30];
   stk[top] = '#';
   while (infix[i] != '\0')
   {
      symb = infix[i];
      i++;
      if ((symb >= '0' && symb <='9') || (symb >='a' && symb <='z'))
      {
         postfix[j] = symb;
         j++;
      }
      else
      {
         if(symb == ')')
         { while (stk[top] != '(')
           { postfix[j++] = stk[top--];
           }
           top--;
         }
         else
```

Scanned by CamScanner

```
{
    while (sp(stk[top]) >= icp(symb))
        postfix[j++] = stk[top--];
    stk[++top] = symb;
}
}
}
while (stk[top] != '#')
    postfix[j++] = stk[top--];
postfix[j] = '\0';
}
main()
{
    char infix[30], postfix[30];
    printf("Enter infix explaination");
    scanf("%s", infix);
    convert(infix, postfix);
    printf("%s", postfix);
}
```

__/__/__

Application 2 : Evaluation of postfix expression

Steps

I Scan one symbol from left to write

II If scanned symbol is an operand then push it on the stack.

III If scanned symbol is an operator pop 2 elements stored from stack. First popped Element into operand 2 & second popped element in operand 1

op2 = stk [top--];

op1 = stak [top--];

IV Perform the indicated operation

Result = op1 operation op2.

V Push the result on to the stack

VI Repeat the above procedure till the end of i/p is encountered

Q Evaluate following pfix exp., show evaluation steps using tabulation method. Also, write the final value

1> 632-5*+1$7+

Postfix Exp	Scanned symbol	op2	op1	Result = op1 op op2	Stack contents
632-5*+1$7+	6				6
32-5*+1$7+	3				6 3
2-5*+1$7+	2				6 3 2
-5*+1$7+	-	2	3	3-2=1	6 1
5*+1$7+	5				6 1 5
*+1$7+	*	5	1	5*1=5	6 5

+1$7
1$7
$7+
7+
+

Postfix
123+*32
23+*321
3+*321
+*321
*321
321-
21-+*
1-+
-+*
+*
*

12+3-2
2+3-2
+3-2
3-21
-21+
21+
1+3
+3

__/__/__

+1$7+	+	5	6	6+5 = 11	11
1$7+	1				11 1
$7+	$	1	11	11^1 = 11	11
7+	7				11 7
+	+	7	11	11+7	18

Postfix Exp	Scanned	op2	op1	Result op1 op op2	Stack Contents
123+*321-+*	1				1
23+*321-+*	2				1 2
3+*321-+*	3				1 2 3
+*321-+*	+	3	2	2+3 = 5	1 5
321-+	*	5	1	5*1 = 5	5
321-+*	3				5 3
21-+*	2				5 3 2
1-+*	1				5 3 2 1
-+*	-	1	2	2-1 = 1	5 3 1
+*	+	1	3	1+3 = 4	5 4
*	*	4	5	5*4 = 20	20

12+3-21+3$-	1				1
2+3-21+3$-	2				1 2
+3-21+3$-	+	2	1	2+1 = 3	3
3-21+3$-	3				3 3
-21+3$-	-	3	3	3-3	0
21+3$-	2				0 2
1+3$-	1				0 2 1
+3$-	+	1	2	2+1 = 3	0 3

+16
3
128

__/__/__

		opr	op1	result	stack
3$-	3				0 3 3
$-	$	3	3	3^3	0 27
-	-	27	20	0-27	-27
63/835+2**+	6				6
3/835+2**+	3				6 3
/835+2**+	/	6	3	6/3=2	2
835+2**+	8				2 8
35+2**+	3				2 8 3
5+2**+	5				2 8 3 5
+2**+	+	5	3	3+5=8	2 8 8
2**+	2				2 8 8 2
**+	*	2	8	8*2=16	2 8 16
*+	*	16	8	8x16=128	2 128
+	+	128	2	128+2	130

5) 53*5/4+25*2/-

6) 684*3+61-9+3-4+

//WAP evaluate given postfix expression

```
#sd include <stdio.h> #include <math.h>
#define 30 size 30
main ()
{
  char expr [30]; double b;
  printf (" Enter the exp: \n ");
  scanf(" %s", &expr[30]);
```

0 → 48 9 → 57 a = 97 A = 67

journal expr

do i++
i = ! NULL

```
postfix(expr);
printf("%df", postfix(expr));
}

double postfix(char expr[])
{tn
   double stk[size], op1, op2;  do {do
   char symb; int t = -1; int i=0; symb = expr[i];
   if(symb >= '0' && symb <= '9')
        stk[++t] = symb - '0';
   else
   {else
      op2 = stk[t--];
      op1 = stk[t--];
      switch(symb)
      {sw
         case '+': stk[t++] = op1 + op2; break;
         case '-': stk[t++] = op1 - op2; break;
         case '*': stk[++t] = op1 * op2; break;
         case '/': stk[++t] = op1 / op2; break;
         case '^': stk[++t] = pow(op1, op2); break;
         default: printf("Invalid choice");
      }sw
   }else
   i++; c = ! NULL;
} while(expr[i] != '\0');
}tn return stk[t];
```

proj → proj. opts →
cont ops →

__/__/__

#include <stdio.h>
#include <stdlib

QUEUE

It is a data structure where elements are inserted from one end and the elements are deleted from the other.
The end at which the new elements are inserted is called rear end & the end from which the element are deleted is front end

// WAP to implement queue using array

```
#include <stdio.h>      #define max 100
#include <stdlib.h>
void main()          → char queue[max];
{
  int front = 0, rear = -1, temp = 0;
  do {
     printf("Enter choice: ");   char ch; choice;
     scanf("%c", &ch);
     switch (ch)
     {
       case 1: printf(" ");
               push();
               break;
       case 2: printf(" ");
               pop(); break;
```

__/__/__

```
case 3: printf("          ");
        display();
        break;
default: printf("Invalid choice");
}
printf("do you still want to continue? ");
scanf("%c", choice);
} while (choice == 'y');
```