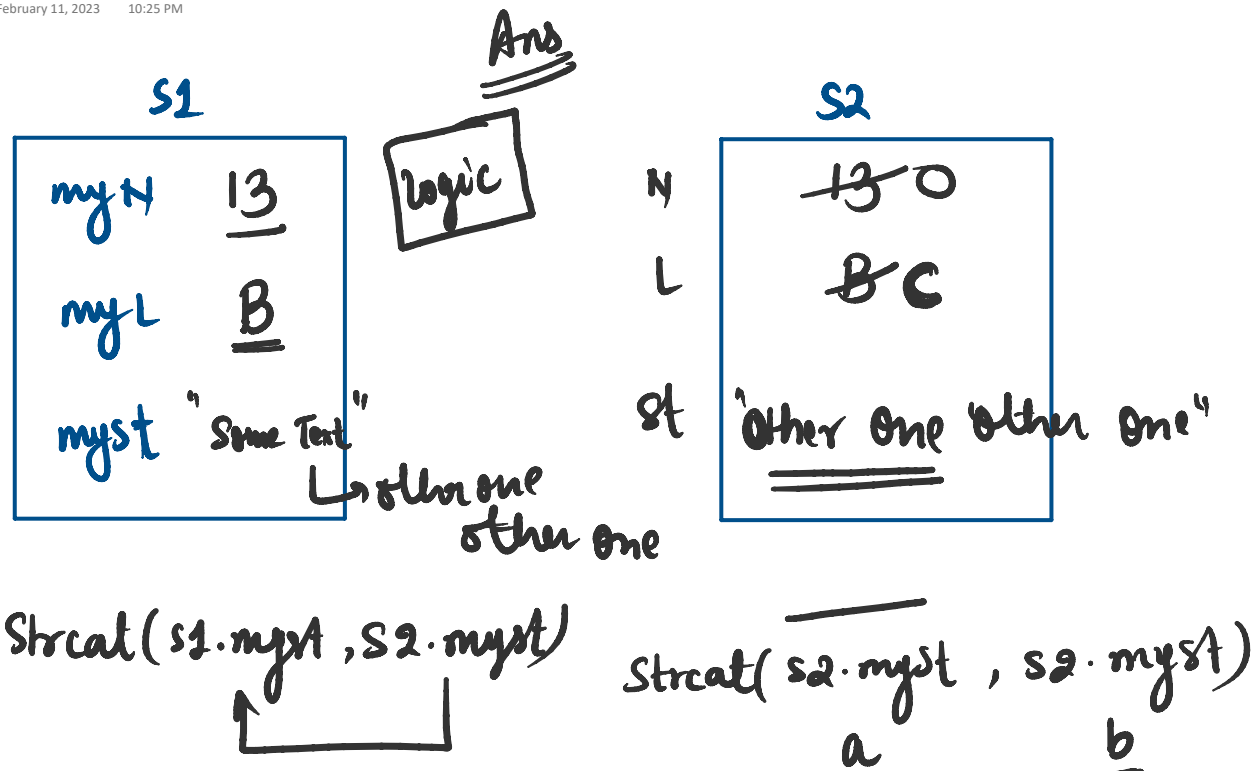
Test Questions

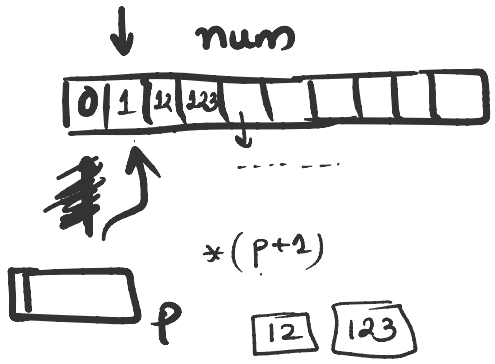
Saturday, February 11, 2023 10:25 PM

Q2

Ans



Q3.



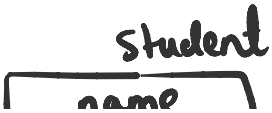
p = p+1

curr = prev * 10 + i

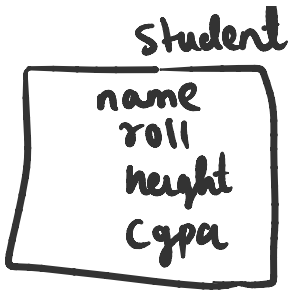
int *num = (int *)

⇒ malloc(10 * sizeof(int))

Q4.



Q4.




```
float avCGPA( student Stu[], int n)
{
    float sum=0
    for(int i=0; i<n; i++)
    { sum += Stu[i].cgpa;
    }
    return sum/n;
}
```

Ques 5

s1 = "Janardan"

s2 = "aabbaab"

change(s2)
change (s2)

A , B , C.

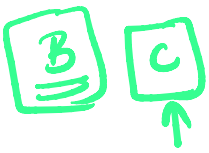
if (strlen(B) == 0) i=
locateSubstr(A, C);

7. Complete the following program, where the main function takes three strings *A, B, C* as input from the user and determines whether the string *A* contains the regular expression *B . C*, where . stands for any substring. For instance, if *A* = "abcdefg", *B* = "bc" and *C* = "ef", the function determines if an occurrence of "bc" followed (not necessarily immediately) by an occurrence of "ef" can be detected in "abcdefg". In this case, the occurrence *BC* is detected at index position 1 in *A*, and the main function gives this output. Either of *B* or *C* can also be null. The main function makes use of another function *locateSubstr* that checks whether a string *A* contains another string *B* as a substring, and if so, returns the match index of *B* in *A*. Thus, when *B* and *C* are non-empty, the main function first finds if *B* is a substring of *A*, and if that is the case, whether *C* is a substring for the remaining portion of *A*, where the match for *B* ends. Fill in the blanks.

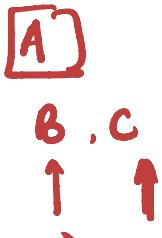
Each blank can have AT MOST one statement. **Marks:** 1+1+2+1+1+1+1+1+1+1 = 10

```
#include <stdio.h>
#include <stdlib.h>
#define MAXLEN 1024
int locateSubstr ( char A[] , char B[] )
{
        int i, j, match;
        if (strlen(B) == 0) return 0;
        for (i=0; i<=  strlen(A) - strlen(B)  ; ++i) if (A[i] == B[0]) {
            match = 1;
            for (j=0; j<  Strlen(B)  ; ++j) if (  A[i+j] != B[j]  )
                { match = 0; break; }
            if (match)  return i  ;
        }
        return -1;
}
int main ()
{
        char A[MAXLEN], B[MAXLEN], C[MAXLEN];
        /* Assume the code to input strings from the users is here.
           You need not write anything here */
        /* i should store the matching index of B*C in A*/
        int i,j,k;
        if (strlen(B) == 0) i = locateSubstr(A,C) ;
        else if (strlen(C) ==  locateSubstr(A,B) ;
        else{
            j = locateSubstr(A,B);
            if (j < 0) i=j;
            else k = locateSubstr(  &A[j+strlen(B)]  C  );
            if (k>=0) i=  j  ;
            else i =  -1  ;
        }
        if (i >= 0)
            printf("The pattern B*C is found in A at idx %d\n", i);
        else
            printf("The pattern B*C is not found in A\n");
        exit(0);
}
```

starting



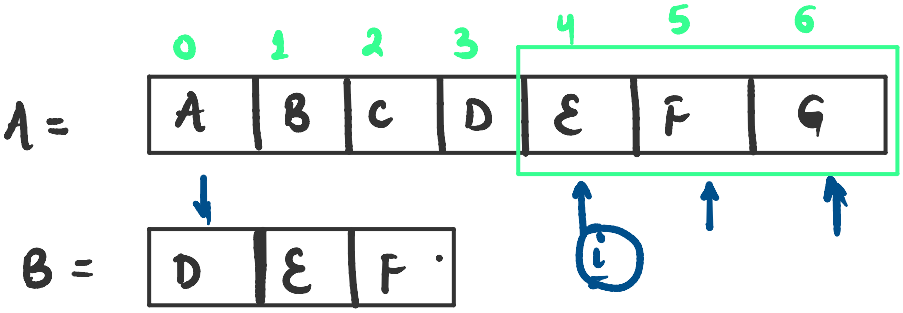
remaini



i+ len(B)

```
int locateSubstr ( char A[], char B[] )
{
    int i, j, match;
    if (strlen(B) == 0) return 0;
```

B



4

7 - 3

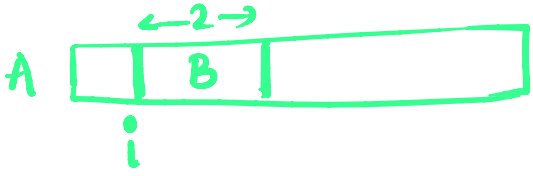
4

```
    for( i=0;  i<= len(A) - len(B) ;  i++)
    {
        if ( A[i] == B[0])
```

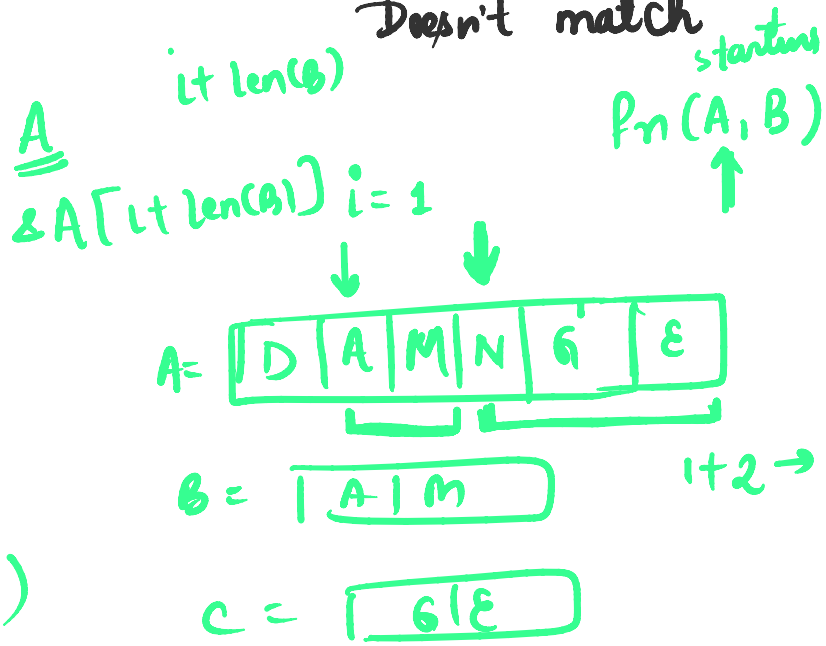
```
{   match = 1;
    for(j=0; j< len(B); j++)
    {
        if( A[i+j] != B[j] )
        {
            match = 0;
            break;
        }
    }
    if(match) return i;
}
}
return -1;
```

Doesn't match

```
}
```



locate (remaining, c)



pointer

pointer

$fn(char\ str[\,]) \Leftrightarrow fn(char\ *str)$

8. The following recursive function makes the base conversion. It reads two integers n and b from the terminal (with $n \geq 0$ and $b > 1$, both in base 10) and expresses n in base b. For example, the decimal

expansion of 345 in base 10 is $345 = 3 \times 10^2 + 4 \times 10 + 5$. Note that in this case 5 = 345%10 and 34 = 345/10; The output as printed by the program should be: (345) 10 = (3, 4, 5) 10. However, please note that for $n = 10$ and $b = 2$, the program should print (10) 10 = (1, 0, 1, 0) 2 and NOT (10)-2 = (1, 0, 1, 0)-2 .

Fill in the blanks. Each blank can have AT MOST one statement.

Marks: 2×5 = 10

```
#include <stdio.h>

void baseconv ( int n , int b )
{
   /* n is [illegible]ll. Simply print it and return. */
   if (    n<b    ) { printf("%d",n); return; }

   /* Recursively print the more significant digits */
      baseconv(n/b,b)      ;

   /* Finally print the least significant digit */
   printf("%d",    n%b    );
}

int main ()
{
   int n, b;

   printf("n = "); scanf("%d", &n);
   printf("b = "); scanf("%d", &b);

   if ((n < 0)||(b < 2)) {
      fprintf(stderr, "Error: Invalid input...\n");
      exit(1);
   }

   printf(" [illegible] (%d) 10 = (    ",n);
   [illegible]",b);

   exit(0);
}
```

call / print

for converting a decimal number n to a number in base b we use recursion to continuously divide the number by its base till its smaller than the base b and we display this first followed by the subsequent display of the previous remainders (n%b) in the previous recursively called functions in down to top approach. That is in simple words we keep dividing the original number n by b till the quotient left is less than b, we display this quotient first and then we backtrack the remainders we got by continuously dividing n by b and print them in the reverse order. This gives us a number in base b equivalent to n base 10.

2	10	0
2	5	1
2	2	0
2	1	1

We stop here because the quotient achieved i.e 1 is less than 2. So now we write the remainders in reverse order.

```
void baseconv (int n, int b)
{
```

1

formatting

fn(n, b);

n → decimal, b → base.

fn(23, 2);

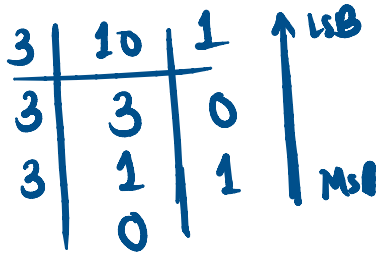
binary

2	23	1	LSB
2	11	1	
2	5	1	
2	2	0	
2	1	1	MSB
	0		

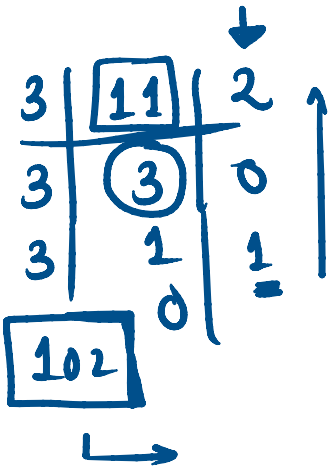
10111

fn(10, 3)

```
fn(n, b)
{
    // Base case
    if(n==0) return;

    // some work & Recursive call
    rem = n % b;
    quo = n / b;

    print(rem);      // last statement
    fn(quo, b);      // last statement
}
```



101

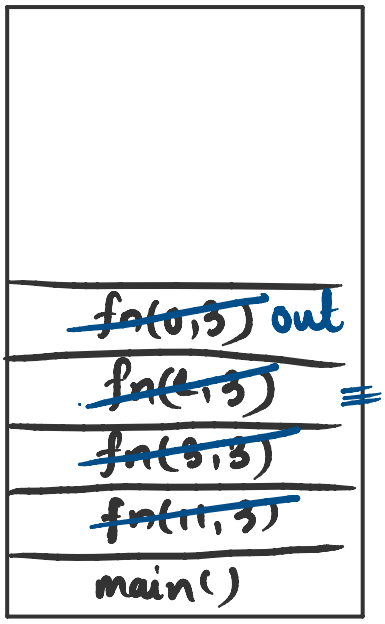


Call stack :

201 screen

We are printing digit by digit.

102



Num = 13425 Reverse using Recursion.

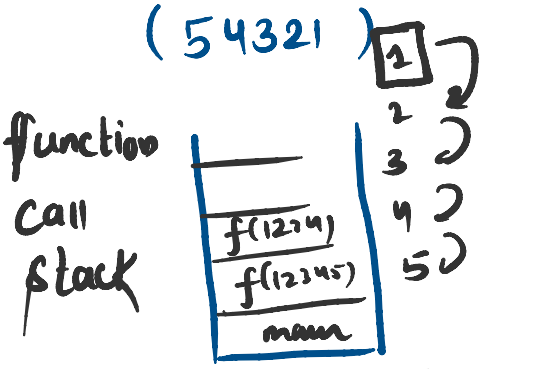
%d

```
void print (int n)
```

```
void print (int n)
{
    // base case
    if (n==0) return.

    digit = n%10
    remaining = n/10

    printf(digit)

    print (remai---)
}
```

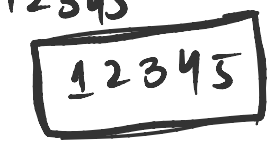
12345 =

(54321)



Reverse

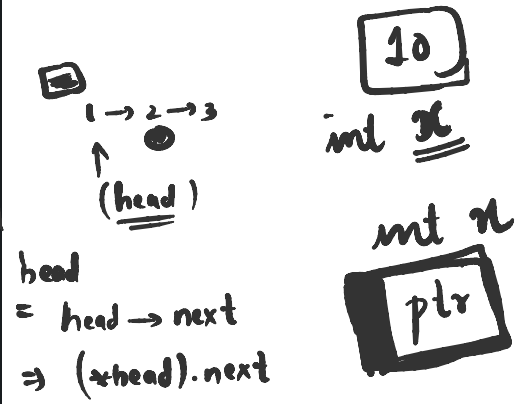
54321



3. Fill in the blanks to complete a C program that creates a singly linked list by repeatedly calling the function push(). It then counts the number of nodes present in the singly linked list recursively using the function getCount(). Each blank has at most one statement.

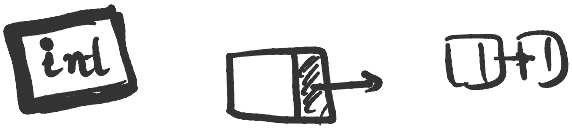
```
// Recursive C program to find length or count of nodes in a linked list
#include<stdio.h>
#include<stdlib.h>

/* Link list node */
struct Node
{
    int data;
    struct Node* next;
};

/* Given a reference (pointer to pointer) to the head of a list and
   an int as parameters, the function pushes a new node on the front
   of the list. */
void push(struct Node** head_ref, int new_data)
{
    /* allocate node */
    struct Node* new_node =
            (__struct Node*__) malloc(__sizeof(struct Node)__);

    /* put in the data  */
    new_node->data  = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    ___(*head_ref) = new_node___;
}

/* Counts the no. of occurences of a node
   (search_for) in a linked list (head)*/
int getCount(struct Node* head)
{
    // Base case
    if (___head == NULL___)
        return 0;

    // count is 1 + count of remaining list
    return 1 + ___getCount(head --> next)___;
}

/* the main function*/
int main()
{
    /* Start with the empty list */
    struct Node* head = NULL;
    int num;
    char flag = 'Y';

    // Use push() repeatedly to construct the list
    while(___flag == 'Y' (OR) flag != 'N'___)
        {
        printf("\n Enter the next number to be pushed into the stack: ");
        scanf("%d", &num);
            push(___&head___, num);
        printf("\n Do you want to push more numbers into the stack?
        (Answer Y to continue and N to stop pushing into the stack)");
        scanf("%c", &flag);
        }

     /* printing the size of the list created */
    printf("\n Number of nodes in the linked list is %d",
    ___getCount(head)___);
    return 0;
}
```

The Solution is Self Explanatory. Comments are Provided for proper understanding.



head = head → next

⇒ (*head).next



head = newNode

head

*head_ref = newNode;

fn(node * head)

fn(int n)

{

}

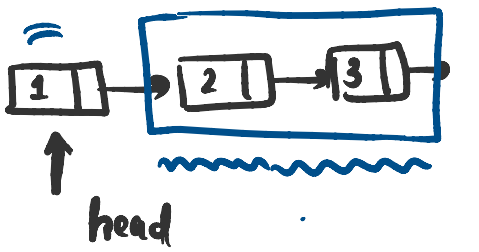
int x = 20

→ fn(n)

head

fn(head)

1+



len (node * head)

{

// Base case

if (head == NULL) return 0;

rem = len(head → next);

return rem + 1;

}

head = NUL

temp = head

cnt = 0

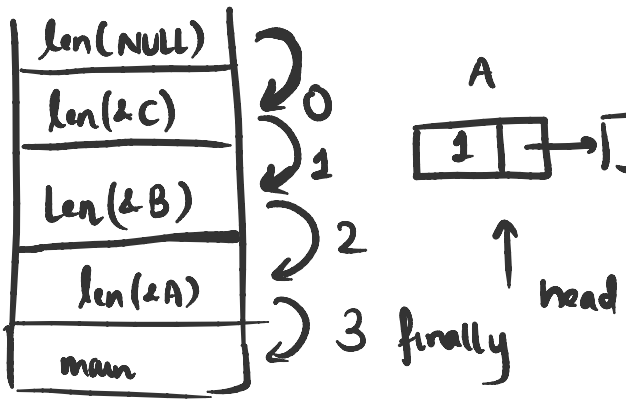
base case

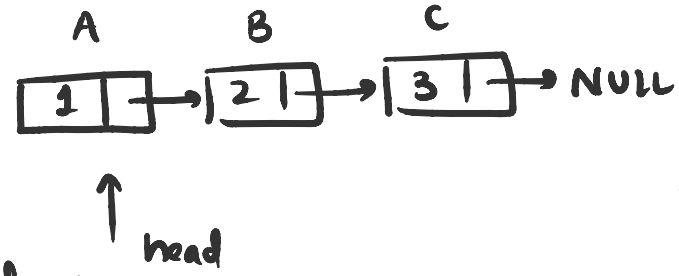
while (temp != NULL)

{ ...

```
    return rem + 1;
}
```

```
while (temp != NULL)
{   cnt++;
    temp = temp->next
}
```



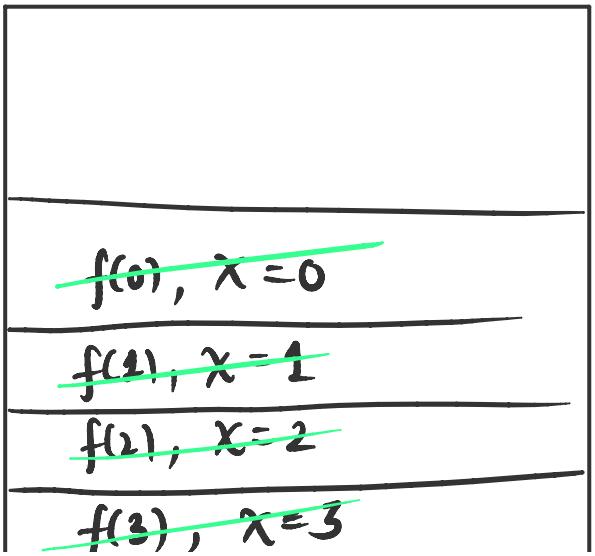


length of the linked list

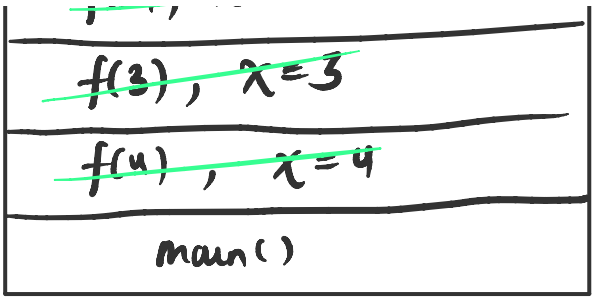
1 + Len (rem)

local variable of the fn

```
f(int x)
{   if (x==0) return;
    print(x);
    f(x-1);
}

main()
{
```



```
{
    f(4);
}
```



4321

f(2) ⇝ x = 2

f(9) ⇝ x = 9

f(10) ⇝ x = 10

} Manually

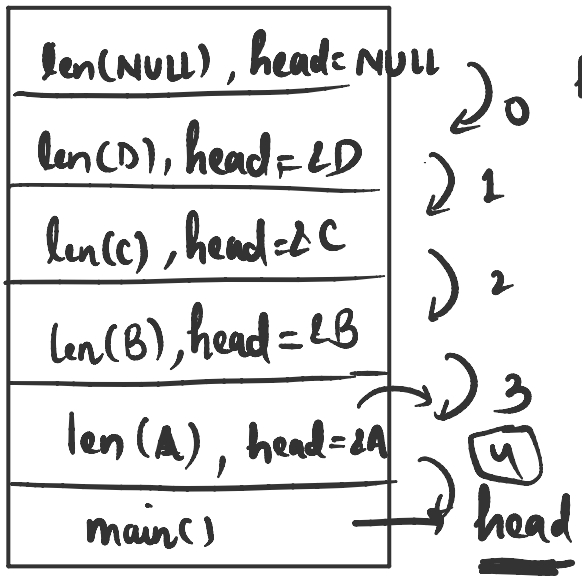
```
len(head)
{   if (head == NULL) return 0;
    return 1 + len(head->next)
}
```

call

5

```
main()
{   node *head = address of a list containig
    len(head)          4 nodes.
}
```

1 → 3 → 4 → 9 → NULL



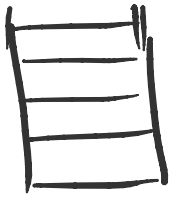
head = 10

re

pass by

head → next → next → next → next

this is one way list



Ques.

Stack operations:

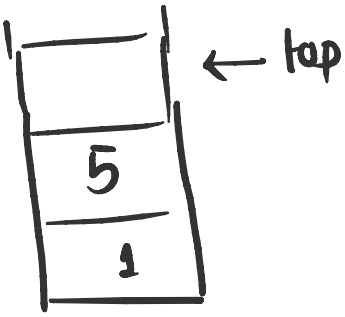
a) push – inserting at top

b) pop – deleting from top

c) peek – viewing what is at top.

d) size – current no. of elements in

e) empty



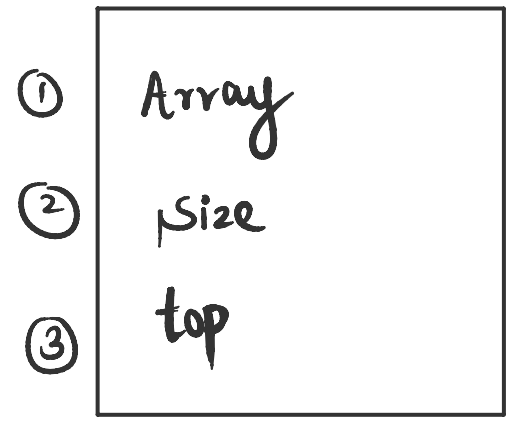

```
Stack S; // empty stack
S.push(1);
S.push(2);
S.push(3);
S.peek(); → 3 ))
S.pop();
```

```
S.pop();
S.push(a)
S.pop();
S.pop();
S.push(5);
S.size() → 2
S.empty() → 0  if not empty
          → 1  if empty
```

Implementing stack data structure

a) Using arrays

b) using linked list



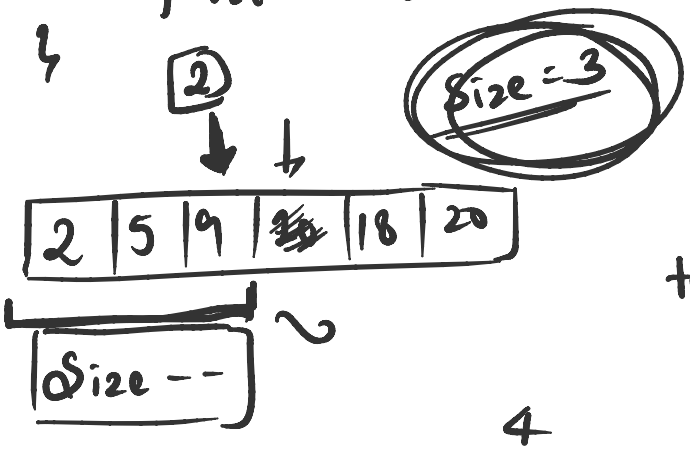
stack

```
Struct Node {
    int data;
    struct Node * next;
};
```

Default values

```
    struct Node * next;
};

main()
{
    struct Node first;
    // Default value
      first.data = 10
      first.next = NULL;
}
```

first



top=

Size = 3

push(5)

push(20)

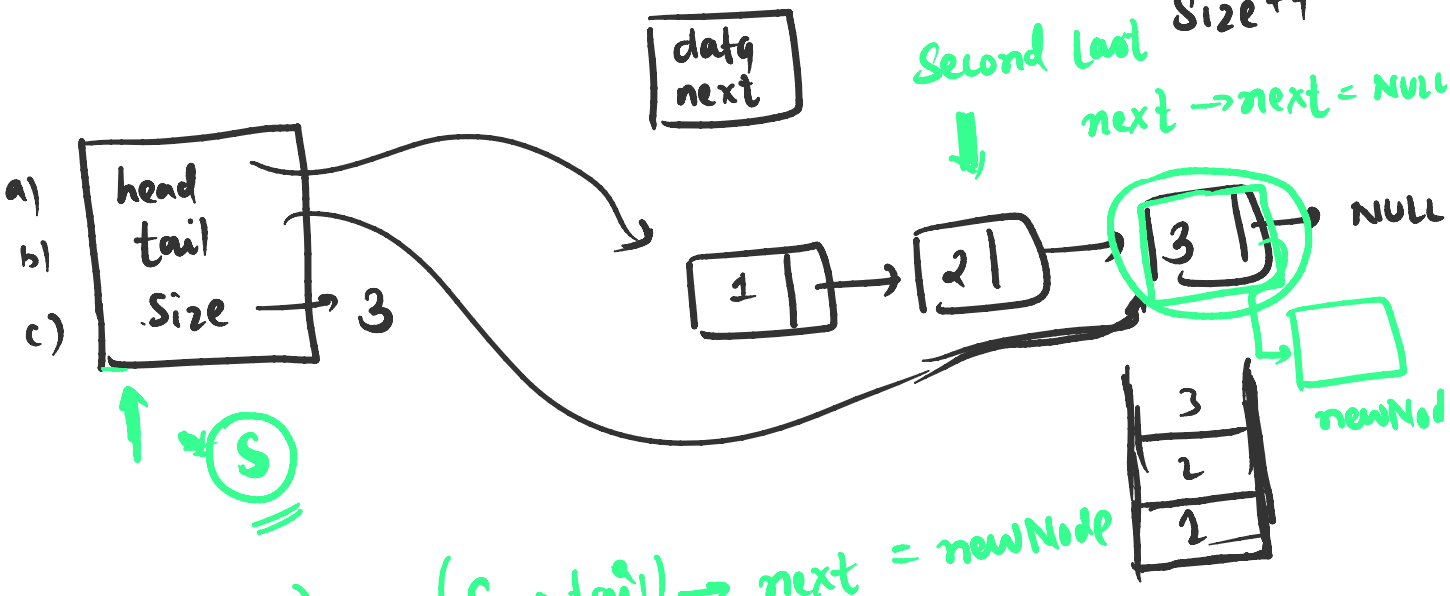
Size = 0

4 → 3

3 → 2

arr[size]

Size++



(S → tail) → next = newNode

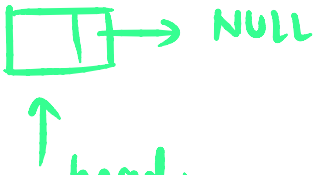
(S → member)

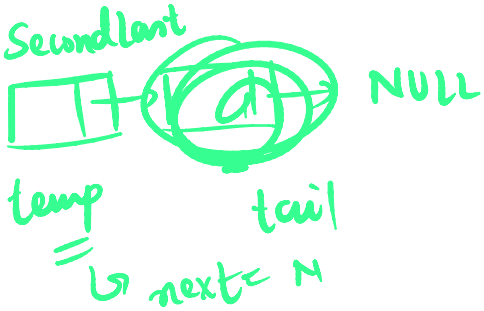
(S → member) (S → tail) → next - ...

node

node → member

Secondl ↳ next = tail





Bracket pair matching: / Balanced parenthesis

String of parenthesis

a) ( ) ( ) ( ) → Balanced

b) (( )) ( ) → B

c) (( )) ( ) → Unbalanced

d) ((( )) ( ) ( ) → U

e) ( )) ( ) (→ U

f) (( ) ( )) → B

Balanced : no of opening parenthesis == no. of closing brackets

no. of closing brackets

())(() → Unbalanced

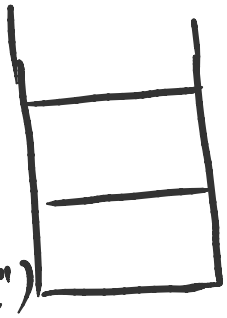
if(open != closed) print("Not balanced")

```
Stack S;

for(i=0; i< n; i++)
{
    if(str[i]=='(')
        push("(")
    else {
        if(s.empty()) print("Unbalan")
        else S.pop()
    }
}

if(s.empty()) print("Balanced")
else print("Unbalanced").
```

Ex:

Str = "(())()"

(closing)



((()))

[()]{}{[()()]()}

From <https://practice.geeksforgeeks.org/problems/parenthesis-checker2744/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article>

Ques

Str = "ABHISHEK"

Reverse using stack.

```
Stack S;
for (i=0;   i< n;  i++)
{
    s.push (str[i])
}

while (!s.empty())
{
    print (s.top());
    s.pop()
}
```

Reverse.

KEHSIHBA

Remaining -

a) Queue

b) Files