Page No.
Date

(*) Pointer: It is a variable which consist of address of other variables or function.

Declaration:

```
Variable-type * Pointer-name;
```

Eg:- `int * int-ptr;`
`double * double-ptr;`
`char *char-ptr;`

`int *int-ptr { null-ptr };`

→ Initializing pointer variable to point 'nowhere'.

Note: If you don't initialize a pointer to point to a variable or function then you should initialize it to nullptr to make it Null.

Eg:- `int *P;`

Garbage value

```
cout<<" value of P" << P<<endl; // 0x61ff60
cout<<" Address of P"<< &P <<endl; // 0x61ff80
cout<<" Size of P " << sizeof(P) <<endl;
```

↳ 4

*In C++ addresses are represent in hexadecimal.

Page No.
Date

```
p = nullptr; // Set p to point nowhere
cout<<" value of p is :" << p << endl;
```
→ 0

(•) The compiler will make sure that the address stored in a pointer variable is of the correct type

```
int score{10};
double high-temp {100.70};

int *score-ptr {nullptr};
    score-ptr = score ; // ok
    score-ptr = high-temp // Compilation Error.
```

(*) Dereferencing a pointer :

↳ Access the data we're pointing to

(•) if score-ptr is a pointer and has a valid address.

(•) Then you can access the data at that address contained in the score-ptr using the dereferencing operator *.

Page No.
Date

Eg:- int score{100};

int *score_ptr{&score};

dereference operator

cout<< *score_ptr<<endl; //100

*score_ptr=200;
cout<<*score_ptr<<endl; //200

cout<<score<<endl; //200

*score_ptr;

? access to the value present at the address of value of that pointer.

Dereferencing

100
score

xyz
score_ptr

xyz
simple pointer.

Eg:- {
vector<string> stooges{"larry","moe","curly"

vector<string> *vector_ptr{nullptr};

vector_ptr=&stooges;

Pointer majority have size 8 byte.

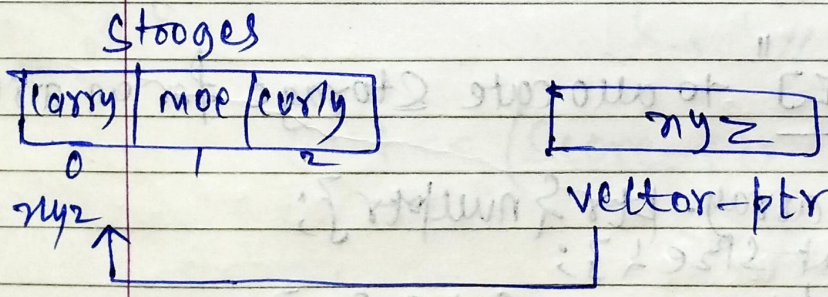
Page No.
Date

```
cout << "first stooge:" << (*vector-ptr).at(0) << endl;
```

→ larry.

```
cout << " Stooges: ";
for( auto stooges : *vector-ptr){
        cout << stooges << " ";
        cout << endl;
}
cout << endl;
}
```

→ Larry, moe, curly



};

Page No.
Date

(*) Dynamic memory allocation.

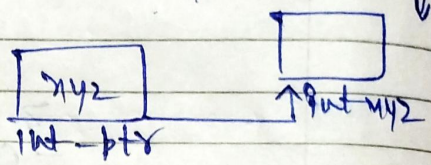
(•) using "new" to allocate storage.

Eg:- int *int_ptr{nullptr};

int_ptr = new int;

// allocate an integer on the heap.

... and so on



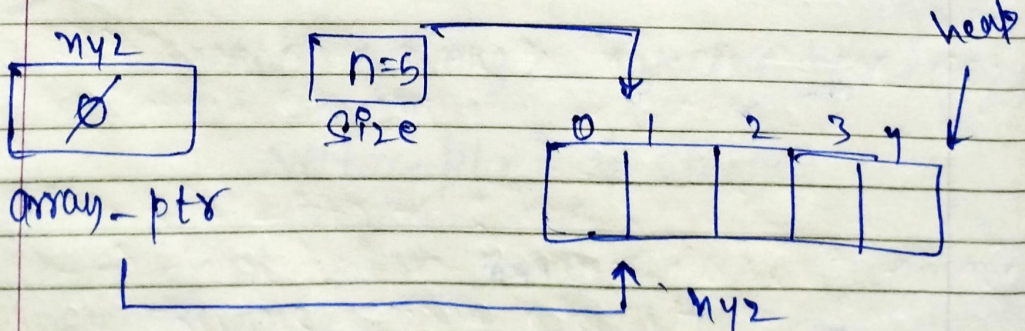
(•) "delete" to free the allocated memory

Eg:- delete int_ptr;

(•) using "new[]" to allocate storage for an array.

Eg:- int *array_ptr{nullptr};
int size{};
array_ptr = new int[size];

// allocate array on the heap



now can access array using array_ptr;

Page No.
Date

(*) using "delete []" to deallocate storage for an array.

Eg:- delete [] array-ptr ; // free allocated storage

(*) Relation b/w Arrays and Pointer :-

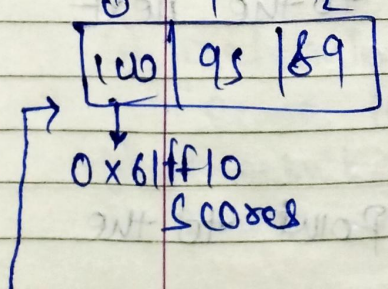
Eg:-

```
int score[] = { 100, 95, 89 };
int * score-ptr { scores };
```

(can put scores without a because it is an array.)

```
cout<< score-ptr <<endl;
        // 0x61ff10
cout<< (score-ptr+1) <<endl;
        // 0x61ff14
cout<< (score-ptr+2) <<endl;
        // 0x61ff18
```

+4 byte, +4



arr[i] = *(arr+i) ✓

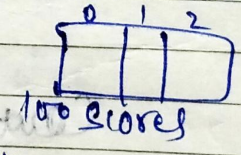
or

i[arr] = *(i+arr) ✓

Subscript Notation	offset Notation
array_name [index];	*(array_name + index);
Eg:- scores[0]; // 100	Eg:- *(scores +0); // 100
Pointer_name [index];	*(pointer_name + index);
Eg:- scores_ptr [0]; // 100	Eg:- *(scores_ptr +0); // 100

(*) Pointer arithmetic:

Eg:- int *int_ptr{scores



Eg:- int_ptr++; // 104

(++) increment a pointer to point to the next array element

Eg:- int_ptr--;

(--) decrement a pointer to point to the previous array element

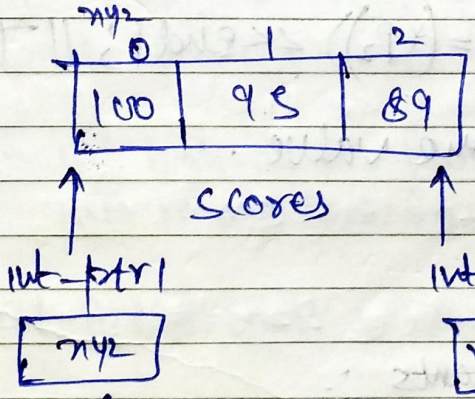
Page No.
Date

(*) Subtracting two Pointers :-

(•) Determine the no. of elements between the pointers

(•) Both pointers must point to same datatype

```
int n = int_ptr2 - int_ptr1;

int * int_ptr1 {scores};

int * int_ptr2 {scores[2]};
```

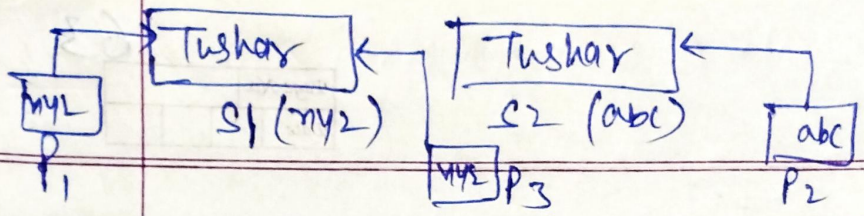



```
cout << (int_ptr2 - int_ptr1) << endl;
                ↳ 2
```

(*) Comparing two Pointers == and |=

(•) Determines if two pointers point to the same location.

(•) Doesn't compare the data where they point.



Eg:- Strong s1 {"Tushar"};

Stang s2 {"Tushar"};

strong *P1 {&s1};

strong *P2 {&s2};

Strong *P3 {&s1};

cout<< (P1==P2) << endl; // false

(out << (P1==P3) << endl; // True.

(out << (*P1)==(*P2)) << endl; // True

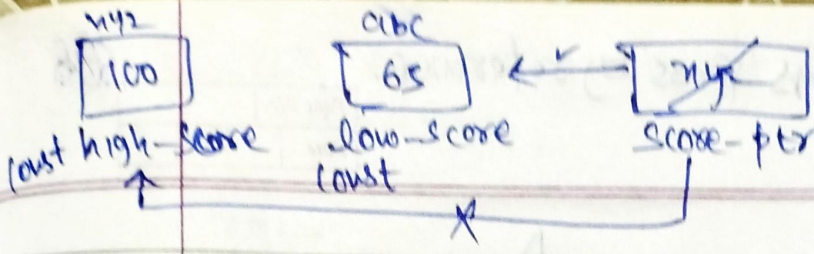
↳ checking the value.

(*) Pointer to constants :-

(i) The data pointed to by the pointers is constant and cannot be changed.

(ii) The pointer can change itself and can point somewhere else.

Eg:- int high-score {100};
int low-score {65};



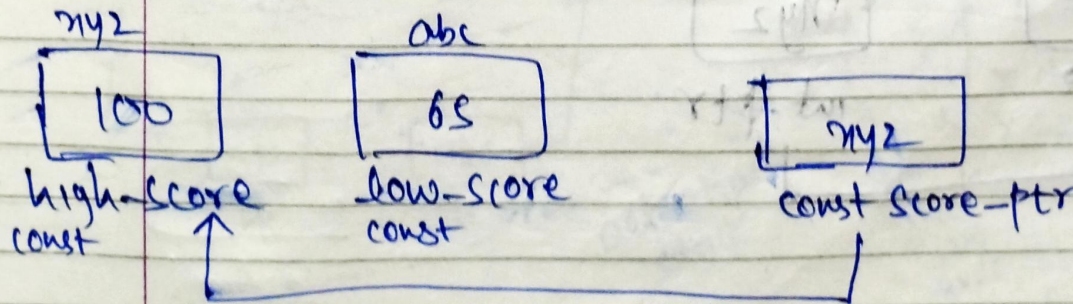

```
const int * score_ptr{ &high_score};

*score_ptr = 86; // Error ✗

score_ptr = &low_score; ✓
```

(*) Constant pointer to constants :

(•) The data pointed to by the pointer is constant and cannot be changed

(•) The pointer itself cannot change and point some where else.

Eg:-
```
int high_score {100};
int low_score {65};

const int * const score_ptr{ &high_score};

*score_ptr = 86; // Error ✗

score_ptr = &low_score; // Error ✗
```



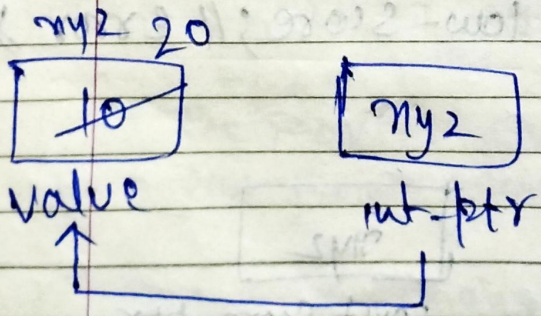
Note : Pointers are always Pass by reference.

Page No.
Date

(*) Passing Pointers to a function :

Eg:- Pass by reference to Pointer.

```
void double-data( int *int-ptr);

void double-data( int *int-ptr){
    *int-ptr *= 2;
}  // *int-ptr = *int-ptr *2 ;
int main() {
    int value {10};
  cout<< value << endl;  //10
    double-data(&value);
  cout<< value <<endl; //20
}
```

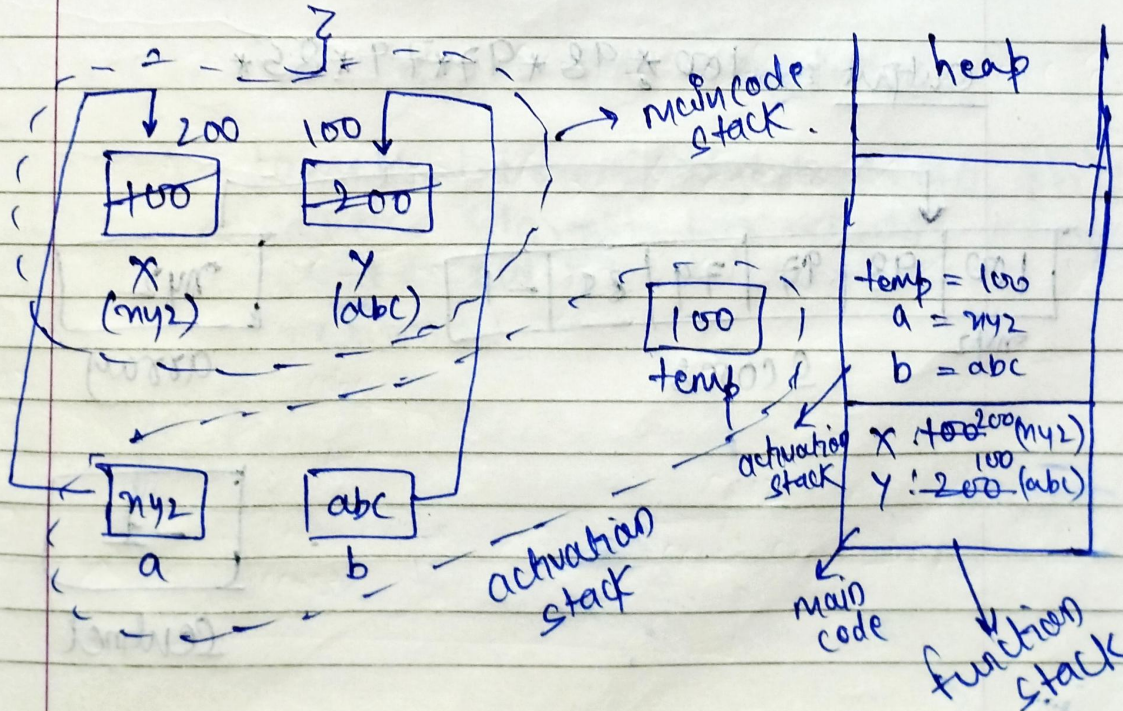


(4) Address of array can't be reasign.

Page No.
Date

Eg:-

```
void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}
int main(){
    int X{100}, Y{200};
    cout << "X:" << X << endl;  //100
    cout << "Y:" << Y << endl;  //200
    swap(&x, &y);
    cout << "X:" << X << endl;  //200
    cout << "Y:" << Y << endl;  //100
    return 0;
}
```

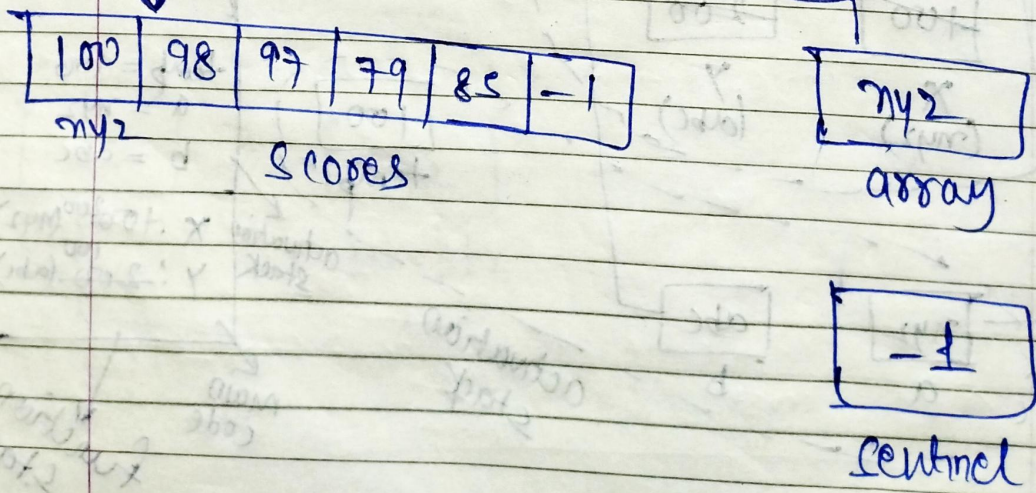


Page No.
Date

Eg:

```
void display( int *array, int sentinel){
    while(*array != sentinel){
        cout<<*array++<<"*";
    } cout<<endl;
}
int main(){
    int scores[] {100,98,97,79,85,-1};
    display(scores,-1);
    cout<<endl;
    return 0;
}
```

output : 100*98*97*79*85*



Page No. Date

(*) Returning a pointer from a function :-

(•) function can also return pointers

```
type *function();
```

Eg:-

```
int * largest_int(int *int_ptr1, int *int_ptr2)
{
  if (*int_ptr1 > *int_ptr2)
      return int_ptr1;
  else
      return int_ptr2;
}

int main() {
    int a{100};
    int b{200};

    int *largest_ptr {nullptr};

    largest_ptr = largest_int(&a, &b);
    cout << *largest_ptr << endl; //200
    cout << endl;
    return 0;

}
```

Page No.
Date

Eg:-

```
int *create_array(size_t size, int init_value=0)
    {
      int *new_storge{nullptr};
      new_storge=new int[size];
    for(size_t i{0}; i<size; i++){
      *(new_storge+i) = init_value;
      }
      return new_storge;
}
int main(){
    int *my_array{nullptr};
    my_array = create_array(100,20);
    delete[] my_array;
    return 0;
}.
```

Page No.
Date

(*) Dangling Pointer :

(i) Pointer that is Pointing to released memory.

Eg:- 2 Pointers pointing to same data
1 pointer releases the data with delete
The other pointer access the release data.

(ii) Pointer that points to memory that is Invalid.

(*) Leaking memory :

- Forgetting to release allocated memory with delete
- If you lose your pointer, there is no way to access the data again in the heap.

(*) References : Alias to a variable.

Eg:- using reference in a Range based loop

```
vector<string> stooges {"Larry", "moe", "curly"};

for(auto str: stooges)
    str = "funny"; // changes in its copy.
```

```
for( auto str : stooges)
    cout<< str<< " ";

// larry moe curly.
```

```
for ( auto &str: stooges)
    str= "funny";   // changes in orginal.

for ( auto str: stooges)
    cout<< str<< " ";

// funny funny funny.
```

Eg:-
```
int num{100};
int &ref{num};

cout<< num<<endl;   // 100
cout<< ref<<endl;   // 100

num=200;
cout<< num<<endl;   // 200
cout<< ref<<endl;   // 200

ref=300;
cout<< num<<endl;   // 300
cout<< ref<<endl;   // 300
```

Page No.
Date

(*) L-values

- values that have names and are addresable
- modifiable if they are not constant

Eg: int x{100}; // x is an L-value
x = 1000;
x = 2000;

String name; // name is L-value
name = "Frank";

(*) R-values:

- value that is not an L-value
- non-addresable and non modifiable

Eg: int x{100}; // 100 is R-value

int y {x+200}; // x+200 is R value

int max-sum = max(20,30); // max(20,30) is R value

Note: Rvalues are assigned to Lvalues explicitly

Array questions

imp (*)

leetcode 189

Q Rotate array.

Note : %n → of [0-(n-1)].

eg:- [1, 5, 9, 11, 13] K=2.

o/p = { 11, 13, 1, 5, 9 }

arr[(i+K) %.n] = arr[i]

eg:-

0	1	2	3	4	5	6
1	2	3	4	5	6	7

K=2

o/p =

0	1	2	3	4	5	6
6	7	1	2	3	4	5

for 6th element

arr[(6+2) %. 7] = arr[8 %. 7] = arr[1]

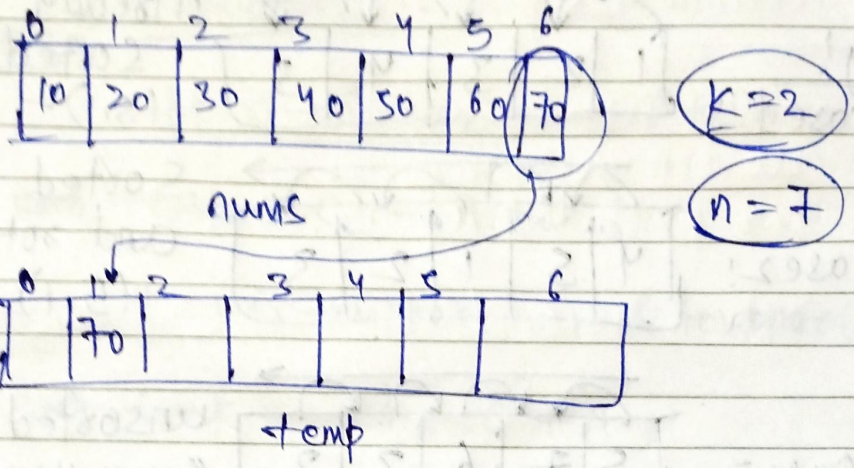
Code :

```
void rotate ( vector <int> & nums , int K ){
    vector <int> temp (nums.size());
    for( int i=0; i< nums.size(); i++){
        temp [ (i+k) % nums.size() ] = nums[i];
    }
    nums = temp ;
}
```

Shift the ith element by K

Page No.
Date

Eg:-



set (i=6)

temp ((6+2) % 7) = nums[6]

// Shift the i^{th} element by K.

leetcode 1752

Q. Sorted and Rotated array.

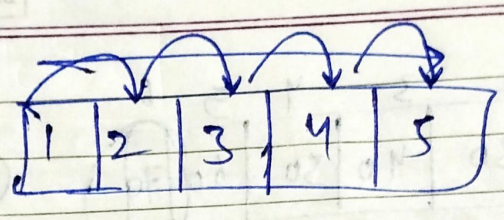
Code:

```
bool check(vector<int>& nums){
    int count=0;
    int n = nums.size();
    for(int i=0; i<n; i++){
        if(nums[i-1] > nums[i])
            count++;
    }
    if(nums[n-1] > nums[0])
        count++;
    return count<=1;
}
```

Page No.
Date

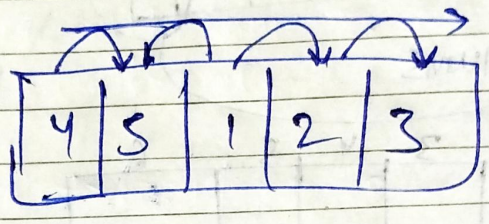
Eg!

case 1:



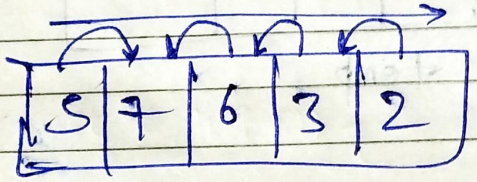
Already sorted.
(5,1)

case 2:



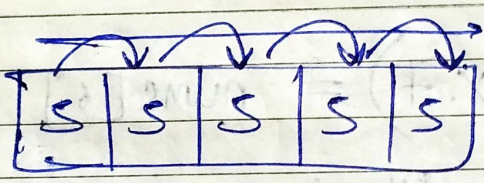
sorted and rotated.
(5,1)

case 3:



unsorted.
// more than 1 pair

case 4:



sorted & rotated.

Conditions:

```
nums[i-1] > nums[i];
// case 2;

nums[n-1] > nums[0];
// case 1;
```

Page No.
Date

Conversion :-

Note :

upper → lower // tolower();

Char temp = ch - 'A' + 'a';

lower → upper // toupper();

Char temp = ch - 'a' + 'A';

Header file : #include <cctype> // C++ style

or

#include <ctype.h>

// C style

The C++ <cctype> header file declares a set of function to classify (and transform) individual character

Eg:- isupper(), toupper(), isalpha(), ... etc.

Page No.
Date

Q leetcode 125

Convert uppercase to lowercase and remove all non-alphanumeric character then check for palindrome.

return true if it is palindrom, else false.

Code : class Solution {

private :

// checking for alpha numeric character

bool valid (char ch) {

if((ch>='a' && ch<='z') || (ch>='A' &&
ch<='Z') || (ch>='0' && ch<='9'))

{ return 1;

}

return 0;

}

// converting numeric alpha to lower case.

char tolowercase (Char ch) {

if((ch>='a' && ch<='z') || (ch>='0' && ch<='9'){

return ch;

}

Page No.
Date

```
else{
    // logic to convert upper to lower.
  char temp = ch - A + a;
    return temp;
}
}
// Checking for Palindrome.
bool checkpalindrome (string a){
    int s=0;
    int e= a.length()-1;
    while(s<=e){
        if( a[s] != a[e]){
            return 0;
        }
        else{
            s++;
            e--;
        }
    }return 1;
}
```

```
p){   public:
        bool ispalindrome (string s){
            string temp{};
```

```
for(int j= 0; j< s.length(); j++) {
        if (valid(s[j])){
            temp.push_back(s[j]);
        }
    }

for(int j= 0; j<temp.length(); j++)
    {
        temp[j] = tolowercase(temp[j]);
    }
    return checkPalindrome(temp);
  }
};
```

Page No.
Date

Q. leetcode 1910

Remove All occurences of substring.

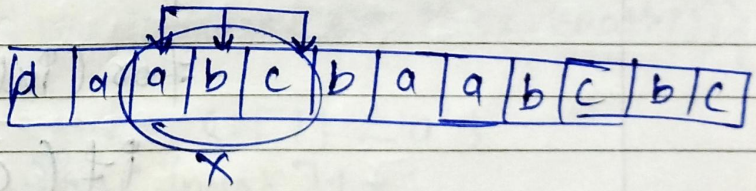
Eg:- S = "daabcbaabcbc"
part = "abc"

o/p = "dab"

{

Code:
```
while(s.length() != 0 && s.find(part) < s.length()) {
    s.erase(s.find(part), part.length());
}
return s;
}
```

Explanation:



= "d a b a a b c b c"
= "d a b a b c"
= "dab"

Removing the substring part from the string s.

Page No.
Date

*

Q. Leetcode 1047 Permutation In String

~~Remove all Adjacent Duplicate In String~~

Eg:- S1 = "ab"

S2 = "eidbaooo"

output: true

Explanation: S2 contains one permutation of S1 ("ba").

Code:

```
Class Solution {

    Private:
        bool checkEqual (int a[26], int b[26])
        {
        for (int i=0; i<26; i++){
            if( a[i] != b[i]){
                return 0;
            }
        }return 1;
    }
```

Page No.
Date

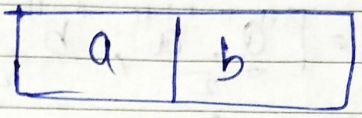
```
public:
    bool checkInclusion(string s1, string s2){
    // Character count array in s1
        int count1[26] {0};

    for(int i=0; i<s1.length(); i++) {
            int index = s1[i] - 'a';
            count1[index]++;
        }

    // traverse s2 string in window of size s1
        int i = 0;
        int windowSize = s1.length();
        int count2[26] {0};
[26])

    // running first window of s1 in s2
    while(i<windowSize && i<s2.length()){

            int index = s2[i] - 'a';
            count2[index]++;
            i++;
    }
    if( checkEqual(count1, count2)){

            return 1;

    }
```

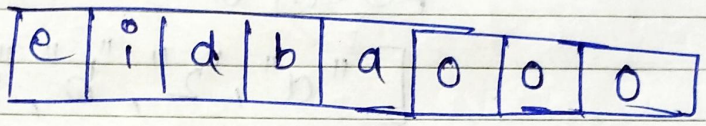
Page No.
Date

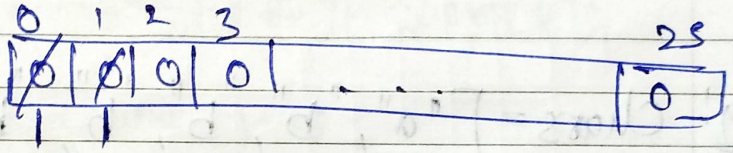
```
// Processing the further window
while( i< s2.length() ){

    // adding the new element to the window

        char newchar = s2[i];
        int Index = newchar - 'a';
        count2[Index]++;

    // delete the old element from window

        char oldchar = s2[i-windowsize];
        Index = oldchar - 'a';
        count2[Index]--;

    // updating the element position
        i++;

    if( checkEqual(count1, count2)){

            return 1;

        }

    } return 0;

}

};
```

Page No.
Date

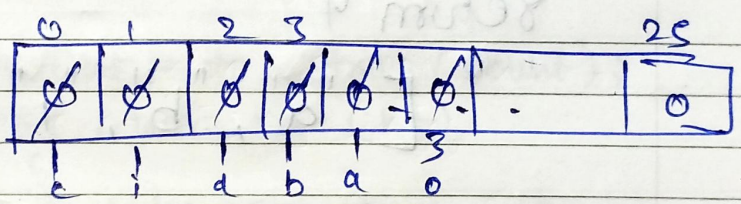
Explanation: $S1 =$ "ab"
$S2 =$ "eidba000"

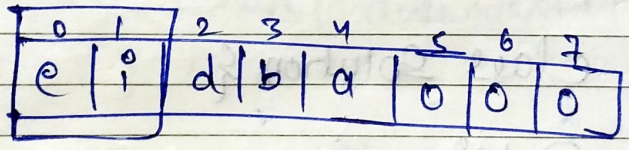
$S1 =$ 

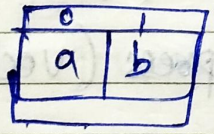
$S2 =$ 

count1 [26] = 

a=1
b=1 } one time

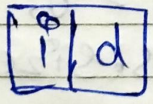
count2 [26] = 

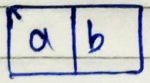
$S2 =$ 

$S1 =$ 

a==e X
b==i X

if not equal then delete the leftmost and add new element





again perform same, repeat loop.