→ Previous greater element :-

In this problem we have given an array, we have to print closest(position wise) greater element than given element in it's left, if there is no such element, print -1.

This problem is much like previous stock span problem, approach is same, we make.

we make a stack, with one initial value at arr[0];

traverse array, and for given element, remove all smaller element than it from stack, then top element give required element, and if stack is empty print -1.

push current element in stack.

Code :-

```
void prevgreater(int arr[], int n)
{
    stack<int> S;
    S.push(arr[0]);
    cout<<"-1 ";
    for(int i=1;i<n;i++)
    {
        while(!S.empty() && (arr[i]>=S.top())
            S.pop();

        int pg= (S.empty())? -1 : S.top();
        cout<<pg<<" ";
        S.push(arr[i]);
    }
```

→ Next grater element :-

In this problem we have given an array, and we have to return an vector in which we have to store next greater element for each value of that array.

next grater element means, number grater than given number and nearest on basis of distance in right side of it.

Ex -

arr[] = {7, 8, 6, 15, 10, 9, 30, 12, 20},

Output :-

vec = {8, 15, 15, 30, 30, 30, -1, 20, -1}

-1 for those element, for which, no greater element in right side of it.

It is similar problem to previous greater element, we just have to traverse the array from backward and instead of pinting the element we store them in vector and rotate the vector before returning.

If we not rotate it, then result will be wrong, as we are traversing the array in reverse.

Code :-

```
vector<int> greater(int arr[], int n)
{
    stack<int> s; vector<int> v;
    s.push(arr[n-1]);
    v.push_back(-1);
for(int i=n-2; i>=0; i--)
{
    while(!s.empty() && (arr[i]>=
                         s.top()))
    {
        s.pop();
    }
    int ans=(s.empty())? -1 : s.top();
    v.push_back(ans);
    s.push(arr[i]);
}
reverse(v.begin(), v.end());
return v;
}
```

→ Delete middle element of a stack :-

approach 1 :- using extra space :-

we have a stack and we have to remove middle element of it then print it.

If we use extra space, we will use another stack in which, we pop top element from given stack and push it in temp. stack until we reach middle elem. we pop it that ele. from stack but not push it in temp stack.

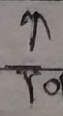
After that we pop element one by one from temp stack and push them in given array.

ex.-

stack <int> s = { 1, 3, 7, 8, 12 }

result :-

{ 12 , 8 , 3, 1 }



result is not reversed but while printing stack, it prints from top to bottom, so

Code:-

```
void(stack<int> &s; int size)
{
    stack<int> t;
    int n = ceil(size/2);
// n is element number which we have to
// remove, if size = 5, n = ceil(5/2) = ceil(2.5)
// which is 3, we have to remove 3rd elem.

for(int i=1; i<=n; i++)
  {
    if(i==n)
    {
      s.pop();
    }
    else{
      int x = s.top();
      t.push(x);
      s.pop();
    }
  }
while(!t.empty())
  {
    int x = t.top();
    s.push(x);
    t.pop();
  }
}
```

Approach 2:- without using extra space

In this approach we will make recursive calls to avoid using extra stack. As we know, in recursion stack calls are made. So, we will use that.

Code:-

```
void deleteuntil(stack<int>&S, int n, int cur)
{
    if(cur == n/2)
    {
        S.POP();
        return;
    }

    int x = S.POP();
    S.POP();
    cur++;

    deleteuntil(S, n, cur);

    S.push(x);
}
```

deletemid

```
void deletemid(stack<int> &s, int size)
{
    deleteuntil(s, size, 0);
}

int main()
{
    stack<int> s;
    s.push(7);
    s.push(10);
    s.push(2);
    s.push(12);
    s.push(5);

    deletemid(s, s.size());

    for(!s.empty()){
        cout<<s.top<<" ";
        s.pop();
    }

    return 0;
}
```

Output:-

5 12 10 7

str b = abc
str = abc

→ Removing consecutive dublicates from String :-

We have given string, in which we have to ~~remove~~ modify the string such that it will remove consicutive dublicate characters.

ex. -

string s = "aaabbCCaadd";

after modification s = "abCad";

So, what we do is we make another string say (str) of same size as S and a stack (say t)

we will push S[0] in stack and run a loop from i = 1 to i < S.length().

Check every S[i], if it's equal to t.top() if yes continue if no

add top of t (t.top()) in second string str then push that S[i] in stack.

return str or copy it in S.

Code (1) same approach :-

```
String removeDublicate(String s)
{
    String str = "";
    Stack<char> t;
    t.push(s[0]);

    for(int i=1; i< s.length(); i++)
    {
        if(s[i] == t.top())
            continue;
        else{
            str = str + t.top();
            t.push(s[i]);
        }
    }
    str += t.top();
    return str;
}
```

Code (2) same approach :-

```
String removeDublicate(String s)
{
    String str = "";
    Stack<Char> t;
    str = str + s[0];
```

```
t.push(s[0]);
for(int i=1; i<s.length(); i++){
    if(s[i]==t.top())
        continue;
    else{
        str+=s[i];
        t.push(s[i]);
    }
}
return str;
}
```

input:- aaabbccd

output- abcd

→ Remove consicutive elements :-

We have given an array of integers in which we have also given two integers x and y. when we incounter two x or two y consicutive remove them from array.

2 1 2 2 1 5

Date
Page

ex:-

vector <int> v = {2,1,2,2,1,5} x=2, y=1

Output: vector v1 = {2,5}

as, first 2,2 in middle will be removed then v becomes {2,1,1,5} then {1,1} in the middle will be removed so final vector will be {2,5}

approach:-

we make second vector to store final result and return it, make stack(t) for operations

we store v[0] is stack, and run loop from i=1 to i = v.size;

if v[i] == x or y then check if v[i] is == t.top() if yes t.pop() if not t.push(v[i]);

if v[i] is not x or y then also t.push(v[i]);

after that copy stack to vector result, and return that.

We can store values in result vector while iterating, so we don't need to copy stack to vector.

Code 1:-

```
vector<int> remove(vector<int> v, int x, int y){
    stack<int> t;
    t.push(v[0]);
    for(int i=1; i<v.size(); i++){
        if(v[i]==x || v[i]==y){
            if(v[i]==t.top())
            {
                t.pop();
            }
            else{
                t.push(v[i]);
            }
        }
        else
        {
            t.push(v[i]);
        }
    }
    vector<int> v1;
```

```
while(!t.empty())
{
    int x=t.top();
    v1.push_back(x);
    t.pop();
}
return v1;
}
```

code 2:-

```
vector<int> remove(vector<int> v, int x, int y){
    vector<int> v1;
    stack<int> s;
    s.push(v[0]); v1.push_back(v[0]);
for(int i=1;i<v.size();i++){
    if(v[i]==x||v[i]==y){
        if(v[i]==t.top()){
            t.pop();
            v1.pop_back();
        }
        else{
            t.push(v[i]);
            v1.push_back(v[i]);
        }
    }
    else{
        t.push(v[i]);
        v1.push(v[i]);
    }
}
```

return V1;

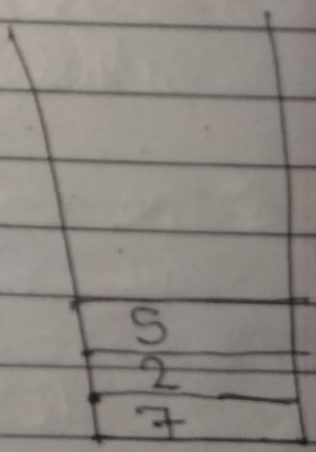
}

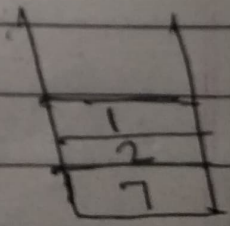
→ Get minimum element from stack :-

In this problem we have to make change in push and pop ~~such that when~~ and make another method getmin() such that when we call getmin() it will return minimum element from stack in O(1) time, also pop and push operation also need to work in O(1).

ex.-

```
stack<int> S;

S.push(7);
S.push(2);
S.push(5);

S.getmin(); // returns '2'

S.pop()

S.getmin(); // return '2'
S.push(1);
S.getmin(); // returns '1'.
```





Approach 1:-

In this approach we will use extra space in form of stack. We have our main stack and an auxilary stack. We use main stack for push and pop and use auxilary array for getmin().

Whenever push() check if pushed elemet is less than or equal to top of stack, Push that element also to aux. stack. Else only push element to main stack.

while poping if poped element is greater than top of auxilary stack only pop from main stack else, also pop from aux. stack.

when asked for minelement return top of auxilary stack.

code:-

~~void push(stack<int> &s, stack<int> &a){~~

```
Class stack_example{
    stack<int> s;
    stack<int> a;
public:
    void push(int x);
    int pop();
    int getmin();
};

Void stack_example::push(int x){
    if(s.empty()){
        s.push(x);
        a.push(x);
    }
    else if(x<=a.top())
    {
        s.push(x);
        a.push(x);
    }
    else{
        s.push(a);
    }
}

int stack_example::getmin(){
    if(s.empty()) return -1;
    else return a.top();
}
```

```
int stack_example::pop(){
    if(s.empty()) return -1;
    else if(S.top() > a.top()){
        int x = S.top();
        S.pop();
        return x;
    }
    else{
        int x = S.top();
        S.pop();
        a.pop();
        return x;
    }
}

int main(){
    stack_example Obj;
    Obj.push(3);
    Obj.push(2);
    Obj.push(10);
    Obj.push(1);
    cout<<"current min is "<<Obj.getmin()<<endl;
    Obj.pop();
    cout<<"current min is"<<Obj.getmin()<<endl;
    return 0;
}
```

Output :-

current min is 1
current min is 2

approach 2 :-

In this approach we will not use another auxilary stack. We use a variable to store current minimum. We modify the main stack such that it will store element which is pushed and previous minimum element.

when getmin() is called return minele element.

when pushing if stack is empty push x to the stack and assign minele = x.
if x is less than minele, push 2x - minele to stack. and assign x to minele.
else
just push x to the stack.

while poping
if stack is empty return -1.
else if(S.top() >= minele) no change in minele ; return S.top() and Pop it.

else
store current minele in variable say (y).
change minele as 2*minele - S.top;
pop S.top;
return y;

Code:-

```
Class stack_example{
    stack<int>S;
    int minele;
public:
    int getmin(){
        if(S.empty()) return -1;
        return minele;
    }

    void push(int x){
        if(S.empty()){
            S.push(x);
            minele=x;
        }
```

```
        else if(x < minele){
            s.push(2*x - minele);
            minele = x;
        }
        else{
            s.push(x);
        }
    }

    int pop(){
        if(s.empty()) return -1;
        else if(s.top() >= minele){
            int x = s.top();
            s.pop();
            return x;
        }
        else{
            int y = minele;
            minele = 2*minele - s.top();
            s.pop();
            return y;
        }
    }
};
```

```
int main(){
    stack_example obj;
    obj.push(2);
    obj.push(1);
    obj.push(-7);
    obj.push(-1);
cout<<obj.get min()<<endl;
    obj.pop();
cout<<obj.get min()<<endl;

 return 0;
}
```

Output :-

1

2