Array Rotation

1. Left Rotation

Rotate array elements left by one position.

```cpp
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    cin >> n;
    int a[n];
    for (int i = 0; i < n; i++)
        cin >> a[i];

    int val = a[0];

    for (int i = 0; i < n - 1; i++)
    {
        a[i] = a[i + 1];
    }
    a[n-1]=val;

    for (int i = 0; i < n; i++)
        cout << a[i] << " ";
    return 0;
}
```

2. Right Rotation

Rotate array elements right by one position.

```cpp
#include <bits/stdc++.h>
using namespace std;
int main()
{
```

```
    int n;
    cin >> n;
    int a[n];
    for (int i = 0; i < n; i++)
        cin >> a[i];

    int val = a[n - 1];

    for (int i = n - 1; i > 0; i--)
    {
        a[i] = a[i - 1];
    }
    a[0] = val;

    for (int i = 0; i < n; i++)
        cout << a[i] << " ";
    return 0;
}
```

---

3. Left Rotation by K times

3.1. Using mod

```
#include <bits/stdc++.h>

using namespace std;

int main()

{

    int n;

    cin >> n;

    int a[n];

    for (int i = 0; i < n; i++)

        cin >> a[i];

    int k;

    cin >> k;
```

```
    int res[n];
    for (int i = 0; i < n; i++)
    {
        int idx = (i + n - k) % n;
        res[idx] = a[i];
    }
    for (int i = 0; i < n; i++)
        cout << res[i] << " ";
    return 0;
}
```

3.2. Using reverse

Reverse the first K items, reverse the rest of the item, reverse the whole array.

```
#include <bits/stdc++.h>
using namespace std;
void reverse(int *a, int i,int j)
{
    while(i<j) {
        swap(a[i], a[j]);
        i++;
        j--;
    }
}
int main()
{
    int n;
    cin >> n;
    int a[n];
    for (int i = 0; i < n; i++)
        cin >> a[i];

    int k; cin>>k;

    reverse(a,0,k-1);
```

```
    reverse(a,k,n-1);
    reverse(a,0,n-1);

    for (int i = 0; i < n; i++)
        cout << a[i] << " ";
    return 0;
}
```

---

4. Right Rotation by K times

4.1. Using mod

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    cin >> n;
    int a[n];
    for (int i = 0; i < n; i++)
        cin >> a[i];
    int k;
    cin >> k;

    int res[n];
    for (int i = 0; i < n; i++)
    {
        int idx = (i + k) % n;
        res[idx] = a[i];
    }

    for (int i = 0; i < n; i++)
        cout << res[i] << " ";

    return 0;
}
```

4.2. Using reverse

Reverse the whole array, reverse the first K items, reverse the rest of the items.

```
#include <bits/stdc++.h>
using namespace std;
void reverse(int *a, int i,int j)
{
    while(i<j) {
        swap(a[i], a[j]);
        i++;
        j--;
    }
}
int main()
{
    int n;
    cin >> n;
    int a[n];
    for (int i = 0; i < n; i++)
        cin >> a[i];

    int k; cin>>k;

    reverse(a,0,n-1);
    reverse(a,0,k-1);
    reverse(a,k,n-1);

    for (int i = 0; i < n; i++)
        cout << a[i] << " ";
    return 0;
}
```

Dynamic Memory Allocation

Dynamic memory allocation in C++ allows you to allocate memory during runtime using the `new` keyword and release it using `delete`.

---

1. Dynamic Memory Allocation

C++ uses the `new` keyword to allocate memory from the heap at runtime.

```
int *p = new int; // allocates memory for a single integer
*p = 10;
delete p;
```

---

2. Create a Dynamic Array

You can create an array of `n` integers at runtime like this:

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n = 5;
    int *arr = new int[n];
    for (int i = 0; i < n; i++)
    {
        arr[i] = i + 1;
    }
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }

    return 0;
```

```
}
```

---

3. Increase or Decrease Size of Dynamic Array

Since raw arrays cannot be resized, you have to:

- Create a new array of the desired size.
- Copy old elements to the new array.
- Delete the old array.

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int oldSize = 5;
    int *oldArr = new int[oldSize];
    for (int i = 0; i < oldSize; i++)
        oldArr[i] = i;

    // Resize to new size
    int newSize = 8;
    int *newArr = new int[newSize];
    for (int i = 0; i < oldSize; i++)
        newArr[i] = oldArr[i];

    // Optional: Initialize new space
    for (int i = oldSize; i < newSize; i++)
        newArr[i] = i;

    delete[] oldArr;
    for (int i = 0; i < newSize; i++)
    {
```

```
        cout << newArr[i] << " ";
    }

    return 0;
}
```

---

4. Delete a Dynamic Array

```
delete[] oldArr; // use delete[] for arrays
```

---

5. Create Dynamic Object

Suppose you have a class:

```
class Student
{
public:
    string name;
    Student(const string &n)
    {
        name = n;
    }
    void show()
    {
        cout << "Name: " << name << endl;
    }
};
```

You can create a dynamic object like this:

```
int main()
{
```

```
    Student *s = new Student("Rahat");
    s->show();

    return 0;
}
```

---

6. Delete a Dynamic Object

```
delete s; // delete the dynamic object
```

✅ **Note:**
Always pair `new` with `delete`, and `new[]` with `delete[]`.