C++ Basics (Part 2)

Goal

To understand:

- Arrays and Vectors
- Strings
- Functions
- Range of datatypes
- Headers and Namespaces
- Competitive Programming tricks
- Basic C++ Template for CP

Arrays

An array is a collection of multiple items of the same datatype.

- Arrays are ordered.
- The size of an array cannot be changed.

Syntax:

```
datatype name[size]
```

Vectors

Very similar to arrays, but more convenient.

- Size of a vector can be changed.
- Supports many features arrays don't.
- Vectors are slower, and take more memory than arrays.

Syntax:

```
vector<datatype> name(size, default_value)
```

Multi-Dimensional Array/Vector

- Two dimensional array syntax: `type a[r][c];`
  For example `int arr[4][6];` creates a 4x6 grid of int

- Two dimensional vector syntax:
  ```
  vector<vector<type>> v(r, vector<type>(c, dv));
  ```

  For example:
  ```
  vector<vector<int>> v(4, vector<int>(6, -1));
  ```

  The above creates a 4x6 grid with default value -1

Strings

Similar to a vector of characters, but supports string operations as well.

- Can only store characters
- Has string-specific functions (e.g. string addition)
- Can be printed easily

Syntax: `string name(size, default_character)`

Functions

Functions are reusable blocks of code that can be run whenever called.

They can take in parameters (input) and return a value (output).

Syntax:

```
return_type name(d1 param1, d2 param2, …) {
    // result must be same as return_type
    return result;
}
```

Useful STL functions for Arrays/Vectors/Strings

- sort
- reverse
- min_element
- max_element
- to_string (Converts integers to string)

Check your understanding 1

1. Find the sum of the given array

2. Write a function to output the minimum and maximum element of an array

3. Create a grid of size NxM and fill it such that such that grid[i][j] is i*j (0-based indexing)

0, 1, 2, 3, ..., 2^31-1, - 2^31, -2^31 +1, -2^31+2,.....,-2,-3,-1

circle as like, int container

Range of integer types

If int has its maximum value and when we add more value then it give negative value as circular manner.
Ex:
int a= 2^31-1;
cout<<a+1; //output= - 2^31
cout<<a+3; //output= - 2^31+2 as like circular

- `int`: (-2^{31}) to $(2^{31} - 1)$
  `2^31` is a bit higher than $2*10^9$

0, i, 2, 3, ..., 2^31-1, - 2^31, -2^31 +1, -2^31+2...

- `long int`: Almost always same as int

as like circular

- `long long int`: (-2^{63}) to $(2^{63} - 1)$
  `2^63` is a bit higher than $9*10^{18}$

float
double
long double : use it good as compaire to double

- Limits of datatypes can be found in the header file <limits>

INT_MAX-> give max of int
LLONG_MAX ->give max of long

int a=-2^31;
cout<<a; //output=-2^31
cout<<a-1;//output=2^31-1
cout<<a-1-1; //output=2^31-2

Namespace

A namespace is a scope of the program that can store various useful functions and variables.

Two ways to use namespaces:

- Use scope resolution operator "::" (double colon) to use the values inside the namespace
- Type `using namespace name;` at the start of the file.

Namespaces are used to avoid conflicting names.

Header files

Header files store C++ variables, functions, etc. to be shared with multiple files

- Pre-existing header files:
  Files provided by the compiler for a variety of purposes.

- User-defined header files: Files written by the user.
  Can be used for templates, or to make code less complex.

Syntax:

```
#include <filename>
```

Header file for Competitive Programming

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

Pros:

- Includes every standard library and STL headers. Therefore, it saves time spent coding during contests.

Cons:

- Increases compile time (Doesn’t matter for CP)
- Does not work with compilers other than GNU C++

```
#include <iostream>
namespace asd {
   int a=4;
   float pi=3.14;}

int main(){
 cout<<a<<pi;
}

//output==4 ,3.14
```

for fixing decimal ke baad kitna digit ayega using

-> setprecision() function

```
Ex: float x=10.234234677543;
      cout<<setprecision(8);
     cout<<x; // give only 8 digit after decimal
```

Fast I/O

```
ios::sync_with_stdio(false);
```

Removes sync between cout and printf.

```
cin.tie(NULL);
```

Removes sync between cout and cin.

```
endl vs '\n'
```

"endl" forces the input buffer to flush.
When using fastio, use '\n' rather than endl

Avoid using fast I/O when debugging

Check your understanding 2

1. Given an NxM grid (use vectors), write a function to reverse each row and return the final grid

2. Find the indices of the “peaks” of a given array. An element which is greater than both adjacent elements is called a peak. (Ignore the first and last elements)

3. Given an array and a number K, rotate the array to the left K times (0 <= K < N)

   For example [1, 2, 3, 4, 5] and K = 2 -> [3, 4, 5, 1, 2]

My template:

```
#include <bits/stdc++.h>
using namespace std;

#define endl '\n'
#define int long long

const int MOD = 1e9 + 7;
const int INF = LLONG_MAX >> 1;

signed main() {
    ios::sync_with_stdio(false); cin.tie(NULL);

    int tc; cin >> tc;
    while (tc--) {

    }
}
```

Problems:

Array problems:

- https://codeforces.com/problemset/problem/110/A
- https://cses.fi/problemset/task/1083
- https://codeforces.com/problemset/problem/677/A

String problems:

- https://cses.fi/problemset/task/1069
- https://codeforces.com/problemset/problem/1619/A

Resources:

- https://www.tutorialspoint.com/cplusplus/cpp_namespaces.htm (for namespaces)
- https://www.programiz.com/cpp-programming/multidimensional-arrays (multi-dimensional arrays)
- https://usaco.guide/general/fast-io?lang=cpp (fast I/O)
- https://devdocs.io/ (docs for all in-built features)

Thanks for watching!