

Must-Know C++ tips and tricks for Competitive Programming | Part -1



Dhruv Kothari

Follow

Jun 11 · 5 min read

Note from Author(Dhruv Kothari):- Hello readers! This article is part-1 of Must Know C++ tips and tricks for Competitive Programming. All the opinions expressed in this article represent my own views which I think is appropriate to keep in mind while doing Competitive Programming.



Before starting I would recommend you to please check out my previous article on beginners' roadmap to start Competitive Programming.

Complete Guide to kick-off Competitive Programming?

This article helps a beginner to get started with competitive programming. It guide's you through a perfect learning...

medium.com

1. Use <bits/stdc++.h> Header File

Let us start with the header file. <bits/stdc++.h> is a most commonly and widely used header file in competitive programming world. Basically, It is a header file that includes all the standard libraries.

```
1 //header files included in <bits/stdc++.h>
2 #include <iostream>
3 #include <algorithm>
4 #include <bitset>
5 #include <complex>
6 #include <deque>
7 #include <functional>
8 #include <limits>
9 #include <map>
10 #include <queue>
11 #include <set>
12 #include <stack>
13 #include <string>
14 #include <vector>
15 //and many more
```

headerfiles.cpp hosted with ❤ by GitHub

[view raw](#)

All these libraries are included in <bits/stdc++.h>. We all know that time is very precious in programming contests, hence using this header file would be a clever choice to save a lot of time and avoid mistakes.

Things to keep in mind while using <bits/stdc++.h>

- This header file contains a lot of header files which might be of no use in your code which may increase compilation time.
- It is not the standard header file of the GNU C++ library, hence compilers other than GCC might not be able to compile it. But don't worry, this won't happen in most cases 😊 😊.

2. “\n” versus endl

For most of us, both sound the same right but it's not the case. endl takes more execution time than “\n” which can cause TLE. Here is the reason:

`"\n"`: it inserts a new line.

`"endl"`: it inserts a new line and flushes the stream.

So `"\n"` is a character whereas endl is a manipulator.

`"endl"` is equivalent to `cout << '\n' << flush;`

Try problem **UNITGCD**: <https://www.codechef.com/problems/UNITGCD>. Using endl in place of “\n” will cause a TLE in the above question.

TIP:

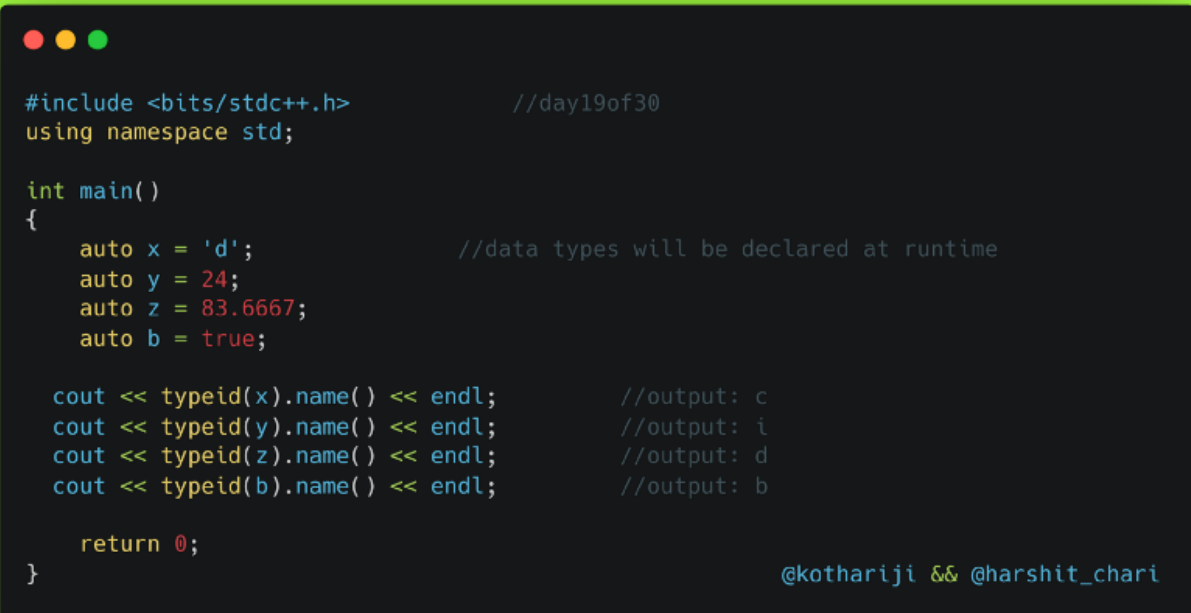
Hence to avoid this mistake we can use define a macro:

```
#define endl "\n"
```

3. auto(Keyword)

When I started with STL, iterators were one of the toughest things to understand as well as declaring-syntax was a difficult job, and maybe the same case is with you. Here's where auto(keyword) comes into the picture. Before C++ 11, data types were

mandatory to be defined during compile-time, but later on, the `auto` keyword was introduced which gave us the freedom to declare a variable at runtime. So we no longer need to define the datatype. This helps us to define iterators very easily.



```
#include <bits/stdc++.h> //day19of30
using namespace std;

int main()
{
    auto x = 'd'; //data types will be declared at runtime
    auto y = 24;
    auto z = 83.6667;
    auto b = true;

    cout << typeid(x).name() << endl; //output: c
    cout << typeid(y).name() << endl; //output: i
    cout << typeid(z).name() << endl; //output: d
    cout << typeid(b).name() << endl; //output: b

    return 0;
}
```

@kothariji && @harshit_chari

4. Range-based for-loops

Range-based for loops is an upgraded version of traditional for loops. They are quite similar to for loops which we use in Python. They were introduced in C++ 11. We can easily iterate over vectors, maps, and other STL containers with this for loop and `auto` keyword. It's really comprehensive and powerful.

Syntax:

for (range_declaration : range_expression)

- **range_declaration:** Here we have to write the data-type of the iterator which is going to iterate through the container. So we can use the “`auto`” keyword. It will detect the datatype during runtime.
- **range_expression:** we have to mention the container on which we have to iterate.

Also, we can reverse iterate the loop by using `boost::adaptors::reverse(v1)` which is included in `#include <boost/range/adaptor/reversed.hpp>` Header

```

#include <bits/stdc++.h> //day20of30
#include <boost/range/adaptor/reversed.hpp> //for reversing range based loop
using namespace std;

int main()
{
    string s = "kotharji"; //data types will be declared at runtime
    int y[] = {1, 2, 3, 4, 5, 6, 7, 8};
    int i = 1;
    vector<int> v1{1, 2, 3, 4, 5, 6, 7, 8};

    for(auto x: s)
        cout << x << " "; //output: k o t h a r i j i
    for(auto x: y)
        cout << x << " "; //output: 1 2 3 4 5 6 7 8
    for (auto x : boost::adaptors::reverse(v1))
        cout << x << " "; //output: 8 7 6 5 4 3 2 1
    return 0;
}

```

@kotharji

5. String To Integers and vice versa

Integer to String

- The `to_string()` function is used to convert a number into a string. I have listed 2 more ways to convert numbers to a string.
- For single-digit integer to string :

```

char c = char(num+48); // where num is single digit integer
                        // like 3 or 7. So char() typecast
                        // {num+48}from ASCII value to char

```

String to Integer

- **stoi()** is the function used for converting a string to an integer, for beginners it might be new, so I included that in this series.
- For **single-digit Integer**:

```
int x = int(c)-48;    // here c is a char like '3' or '7'.So int()
                     // typecast c and convert it into ASCII
                     // value and then we subtract 48(ASCII of
                     // digits start with 48) to get the integer.
```

6. isalnum(), isalpha(), isdigit() and many more !

These functions can be found in header `<cctype>`.

These functions are very useful in checking whether the character is an uppercase alphabet, lowercase alphabet, a number, and many more. These functions are really very useful while doing competitive programming. Here is the list of few must-know functions.

1. **isalpha()** // returns a non-zero value if a character is an alphabet else return 0;
2. **isalnum()** // returns a non-zero value if a character is an alphabet or numeric else return 0;
3. **isupper()** // returns a non-zero value if a character is an upper-case alphabet else return 0;
4. **islower()** // returns a non-zero value if a character is an lower-case alphabet else return 0;
5. **isdigit()** // returns a non-zero value if a character is a numeric digit else return 0;
6. **ispunct()** // returns a non-zero value if a character is a punctuation character else return 0;

Few more functions while use to change the case of alphabet characters.

1. `toupper()` // it is used to convert a lowercase alphabet to uppercase alphabet.
2. `tolower()` // it is used to convert a uppercase alphabet to lowercase alphabet.

7. C++ Code Visualizer

At the initial stage, we often found that our syntax is correct but the output is wrong, which means there is something wrong with the logic. And, sometimes it is very difficult to debug the logical error. In that case, we can use this tool through which we can visualize our code.

Link: <http://www.pythontutor.com/cpp.html#mode=edit>

The screenshot displays the C++ Code Visualizer interface. On the left, a C++ code snippet for a bubble sort algorithm is shown. The code includes a swap function, a bubbleSort function, and a print function. The current execution step is highlighted at line 7 of the swap function. On the right, a memory stack visualization is shown, illustrating the state of the program. The stack contains frames for 'main' and 'bubbleSort(int*, int)'. The 'main' frame shows an array 'arr' with values [24, 64, 25, 12, 22, 11, 90] and a variable 'n' with value 7. The 'bubbleSort' frame shows a pointer 'arr' pointing to the array in 'main', and local variables 'xp', 'yp', and 'temp' with values 24, 64, and 25 respectively. The 'Heap' section is empty.

```

C++ (gcc 4.8, C++11)
EXPERIMENTAL! known limitations
3 using namespace std;
4
5 void swap(int *xp, int *yp)
6 {
7     int temp = *xp;
8     *xp = *yp;
9     *yp = temp;
10 }
11
12 // A function to implement bubble sort
13 void bubbleSort(int arr[], int n)
14 {
15     int i, j;
16     for (i = 0; i < n-1; i++)
17     {
18         // Last i elements are already in place
19         for (j = 0; j < n-i-1; j++)
20             if (arr[j] > arr[j+1])
21                 swap(&arr[j], &arr[j+1]);
22     }
23 }
24 /* Function to print an array */

```

Print output (drag lower right corner to resize)

Stack Heap

main

array

0	1	2	3	4	5	6
int	int	int	int	int	int	int
24	64	25	12	22	11	90

arr

n

7

bubbleSort(int*, int)

arr

pointer

arr

n

7

i

0

j

1

swap(int*, int*)

xp

yp

temp

int

24

64

25

Step 18 of 165

C++ code visualizer



Photo by [Jon Tyson](#) on [Unsplash](#)

References

- https://gcc.gnu.org/onlinedocs/gcc-4.7.0/libstdc++/api/a01457_source.html
- <https://www.geeksforgeeks.org/endl-vs-n-in-cpp/>
- <https://en.cppreference.com/w/cpp/language/auto>
- <https://en.cppreference.com/w/cpp/header/cctype>
- <https://www.tutorialspoint.com/cplusplus11-reverse-range-based-for-loop#:~:text=To%20get%20the%20reversed%20range,for%20loop%20in%20reverse%20order.>

That's it from this article. Hope you liked this article.

For more such tips and tricks check out my github repo <https://github.com/kothariji/30-days-of-code-LinkedIn>

Thank You. See you in the next one!

Part-2 will be released soon.

Sign up for DSC DYPCOE Weekly

By DSC DYPCOE

Get awesome content, latest tech news, articles and awesome stuff straight to your inbox. Stay updated! [Take a look](#)

Your email

[Get this newsletter](#)

By signing up, you will create a Medium account if you don't already have one. Review our [Privacy Policy](#) for more information about our privacy practices.

[Programming](#)[Competitive Programming](#)[Coding](#)[Tips And Tricks](#)[Software Engineering](#)[About](#) [Help](#) [Legal](#)

Get the Medium app

