Personal Competitive Programming Notebook

Juan David Gaviria Correa

June 5, 2022

Contents

1 C++

1.1 C++ template

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

int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    cout << setprecision(20) << fixed;
    return 0;
}
```

2 Type Conversion

2.1 string to number

```cpp
// stof - float
// stod - double
// stold - long double

// stoi - int
// stol - long
// stoul - unsigned long
// stoll - long long
// stoull - unsigned long long

int x = stoi("789.19");

// Output: 789
```

2.2 number to string

```cpp
string x = to_string(475.1);
// Output: 475.100000
```

2.3 int to char

```cpp
// Any int
char x = 97;
// Output: a

// The int is a number from 0 to 9 and want to obtain the
    same number
char x = 5 + '0';
// Output: 5
```

2.4 char to int

```cpp
// Any char
char y = 'a';
int x = y;
// Output: 97

// The char is a number and want the same number as int
char y = '5';
int x = y - '0';
// Output: 5
```

3 Chars

3.1 Change Case

```cpp
char letter = tolower('A');
// Output: a

char letter = toupper('a');
// Output: A
```

3.2 Check Case

```cpp
islower('a');
// Output: true

isupper('A');
// Output: true
```

4 Strings

4.1 Substring

```cpp
string text = "Apple, Banana, Kiwi";
// Second param is optional, default text.lenght
text.substr(7, 6);
// Output: Banana
```

4.2 Replace

```cpp
bool replace(string &str, const string &from, const
    string &to)
{
    size_t start_pos = str.find(from);
    if (start_pos == string::npos)
```

```
        return false;
    str.replace(start_pos, from.length(), to);
    return true;
}

string text = "Apple, Banana, Apple";
replace(text, "Apple", "Banana");
// Output: Banana, Banana, Apple
```

4.3 Replace All Matches

```
void replaceAll(string &str, const string &from, const
    string &to)
{
    if (from.empty())
        return;
    size_t start_pos = 0;
    while ((start_pos = str.find(from, start_pos)) !=
        string::npos)
    {
        str.replace(start_pos, from.length(), to);
        start_pos += to.length();
    }
}

string text = "Apple, Banana, Apple";
replaceAll(text, "Apple", "Banana");
// Output: Banana, Banana, Banana
```

4.4 Change Case

```
string text = "ApPlE";

// To lower case
transform(text.begin(), text.end(), text.begin(), ::
    tolower);
// Output: apple

// To upper case
transform(text.begin(), text.end(), text.begin(), ::
    toupper);
// Output: APPLE

// Capitalize
transform(text.begin(), text.end(), text.begin(), ::
    tolower);
str[0] = toupper(str[0]);
// Output: Apple
```

4.5 Trim

```
void ltrim(string &s)
{
    s.erase(s.begin(), find_if(s.begin(), s.end(), !::
        isspace));
}

void rtrim(string &s)
{
    s.erase(find_if(s.rbegin(), s.rend(), !::isspace).
        base(), s.end());
}

string text = "    ' Apple '    ";
ltrim(text);
rtrim(text);
// Output:' Apple '
```

4.6 Split

```
vector<string> split(string str, string token)
{
    vector<string> result;

    while (str.size())
    {
        int index = str.find(token);

        if (index != string::npos)
        {
            result.push_back(str.substr(0, index)); //
                Push until token
            str = str.substr(index + token.size()); //
                Remove text from 0 to index + token size

            if (str.size() == 0) // Cause token was last
                content, it should be splitted
                result.push_back(str);
        }
        else
        {
            result.push_back(str);
            str = "";
        }
    }

    return result;
}

vector<string> tokens = split(" Apple Banana  Apple ", "
    ");
// Output: ["", "Apple", "Banana", "", "Apple", ""]
```

4.7 Compare Lexicographically

```
string a = "abcd";
string b = "bcde";

if (a < b)
{
    cout << "a is first";
}
else if (a < b)
{
    cout << "b is first";
}
else
{
    cout << "same";
}

// Output: a is first
// Note: You can order lexicographically a vector with
    sort
```

5 Vectors

5.1 Read

```
// Declaration: vector<type> arr(size, default); //
    params are optional
copy_n(istream_iterator<type>(cin), times, back_inserter(
    vec));
```

5.2 Print

```
copy(vec.begin(), vec.end(), ostream_iterator<type>(cout,
    " "));
```

5.3 Sort

```
// This works with any data type
vector<int> vecs = {2, 3, 1, 4};

sort(vecs.begin(), vecs.end());
// Ascending: [1, 2, 3, 4]

sort(vecs.begin(), vecs.end(), ::greater<int>());
// Descending: [4, 3, 2, 1]
```

5.4 Count

```
count(vec.begin(), vec.end(), 12); // Count how many 12's
    are in vec.
```

6 Useful Stuff

6.1 Swap values

```
int a = 1, b = 2;
swap(a, b);
// A will be 2 and b will be 1
```