

# Competitive Programming

## 234900

Lesson 1 - C++ and STL reminder

# Introduction

- Competitive programming makes intensive use of C++ and STL
- Good knowledge of STL can be very helpful! We encourage you to experiment and play 😊
- This presentation provides a quick reminder
- Some slide contains input and output examples:

• Input

• Output

- More information can be found in the C++ reference website:

- <http://www.cplusplus.com/reference/>

# C/C++

Data types, input and output

# Data types

- Max int value  $\sim 2 \cdot 10^9$
- If unsure that the values will be small, better use long long
- For ease of use:

```
typedef long long ll;
```

- Similarly, use long double

# Input/Output

- Input and output is always done through stdin, stdout.
- scanf is faster than the << operator
  - This might be an issue with problems of linear complexity
- Useful specifiers:
  - %d – int
  - %lf – double
  - %c – char
  - %s – C string (char\*)
- If you want to read a whole line use getline()

# Input

- scanf returns the number of values read, or EOF if the file ended.
- Input can have several formats:
  - Number of test cases is given first:

```
int TC;  
scanf("%d", &TC); // cin >> TC;  
while (TC--) {}
```

- Read until EOF:

```
int a, b;  
while (scanf("%d %d", &a, &b) != EOF) {}
```

- Read until the values are 0:

```
int a, b;  
while (scanf("%d %d", &a, &b), (a || b)) {}
```

# Output

- Some problem require printing headers such as case number
- Some problems require printing a blank line between cases
  - In this case, a blank line at the end is considered an error

```
int a, b, c = 1;
while (scanf("%d %d", &a, &b) != EOF) {
    if (c > 1) printf("\n"); //2nd/more cases
    printf("Case %d: %d\n", c++, a + b);
}
```

- Some problems require printing a floating point with a specific precision

```
float num = 0.9375;
printf("%.2f", num); \\ prints 0.94
```

# STL

Using STL, data structures and algorithms



# Using STL

- Programs that use STL need to include the appropriate headers:

```
#include <vector>
```

- STL interfaces are implemented inside the std namespace:

```
using namespace std;
```

- The data type is specified when creating an instance:

```
vector<int> v;
```

# pair

- `pair<int, string> a = {2, "what"};`
- `cout << '(' << a.first << " , "`  
`<< a.second << ')' << endl;`
- `pair<int, pair<int, int>> tuple;`  
`// pairs can also be nested`

- `(2, what)`

# C string vs. C++ string

- It is sometimes easier to read and print a C string, but work with a C++ string

```
char c[100];  
scanf("%s", c);  
string cpp(c);  
  
printf("%s", cpp.c_str());
```

# C++ string

```
#include <string>
// strings work like vectors
string s = "hello";
s.pop_back();
cout << s << endl;
s.push_back('l');
cout << s << endl;
s += " world";
cout << s << endl;
```

- hell
- helll
- helll world

# String functions

- STL includes useful functions to handle strings:
  - ***tolower / toupper*** - turns all the English characters to lower/upper case
  - ***isalpha*** – checks whether a character is an alphabet character
  - ***isdigit*** – checks whether a character is a digit

# vector

```
vector<int> arr1(1e5, 0); //init - o(n)
arr1[0] = 1;
vector<int> arr2 = arr1; //copy - o(n)
arr2[0] = 2; //arr1[0]=1
arr1.push_back(3);
cout << "arr1[1e5]="
<< arr1[arr1.size () - 1] << endl;
arr1.pop_back();
```

- arr1[1e5]=3

# STL Algorithms

- STL includes useful algorithms that operate on iterable objects:
  - ***sort*** – Sort an array/vector in  $O(n \log(n))$  time
  - ***binary\_search*** – Search sorted array in  $O(\log n)$  time
    - And related: ***lower\_bound, upper\_bound*** (next week)
  - ***nth\_element*** – Find the  $n$ th element in a given array in  $O(n)$  time
  - ***merge*** – Merge two sorted arrays/vectors
  - ***unique*** – Remove consecutive repeating elements
  - And many more.
    - Most of them are trivial and relatively easy to implement (max, min, count, reverse...)

# STL Algorithms – cont.

- Pairs are sorted by p.first, and then by p.second
- When dealing with non-primitive objects (structs, classes), STL algorithms can still be used by overloading the < operator:

```
struct fraction {  
    int n, d; // (n/d)  
    bool operator < (const fraction& f) const {  
        return n*f.d < f.n*d;  
    }  
};  
vector<fraction> v;  
sort(v.begin(), v.end());
```



# Algorithm - sort

- `arr.clear();`
- `arr.push_back(3);arr.push_back(1);`  
`arr.push_back(1);arr.push_back(7);`
- `sort(arr.begin(),arr.end());` //  $n \log(n)$
- `for (auto& x : arr)`
  - `cout << x << " ";`
  - `cout << endl;`
- `vector<pair<int, int>> vp;`
- `sort(vp.begin(), vp.end());` // compare by the first and then by the second
- Example- $(1,7)(2,1)(2,2)(2,7)(3,1)$

- 1 1 3 7

# Algorithm – All permutations

- `//All permutation`
- `int myints[] = { 1,2,3 };`
- `sort(myints, myints + 3; //must before next permutation`
- `cout << "The 3! possible permutations with 3 elements:\n";`
- `do {`
- `cout << myints[0] << ' ' << myints[1] << ' ' << myints[2] << '\n';`
- `} while (next_permutation(myints, myints + 3));`
- `cout << "After loop: " << myints[0] << ' ' << myints[1] << ' ' << myints[2] << '\n';`

- The 3! possible permutations with 3 elements:
- 1 2 3
- 1 3 2
- 2 1 3
- 2 3 1
- 3 1 2
- 3 2 1
- After loop: 1 2 3

# Algorithm - Iterators

- Most stl algorithms rely on iterators
- A.begin(), A.end(), A.rbegin(), A.rend(), A.back\_insert\_iterator()  
And more...
- Iterators operations:
- \*it - returns the iterator value
- it++ - increment the iterator
- it1==it2 - does the two iterators point to the same position?

## A good review:

## 105 STL

## Algorithms in

## Less Than an

Hour

# In Practice

- Teams in ICPC have a predefined header they type once and paste in every file to save time

```
#include <algorithm>
#include <bitset>
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <deque>
#include <functional>
#include <iostream>
#include <map>
#include <queue>
#include <set>
#include <stack>
#include <string>
#include <vector>
#include <numeric>
using namespace std;
typedef long long ll;
typedef unsigned long long ull;
typedef vector<int> vi;
typedef pair<int, int> pii;
```

# Lesson instructions

# Submission instructions

- We will be using vJudge for the class competitions:
  - <https://vjudge.net/>
- Submission instructions:
  - Submit the solution to the webcourse.
  - You should only submit the problems you managed to solve (== got accepted by the online judge using the C++11 compiler).
  - Your submission will be checked using an automated system. Please make sure to name your files according to the following pattern: [uva\\_####.cpp](#), where #### is the problem number (i.e. **Prob** column in vJudge).
- One team member can submit for each team (when working in teams).
  - Please include the IDs of your team members.
- You can also submit your solution after the class for a partial credit.



# General Tips

- Read all the questions
- Make sure your solutions has the correct time complexity (Rule of thumb:  $\sim 10M$  operations)
- Have fun!





also we used sort like this anonymous function

```
    cout << i << endl;
} while (next_permutation(i.begin(), i.end(), comp: [](const char& lhs, const char& rhs){
    char first = tolower(lhs);
    char second = tolower(rhs);
    if(first < second){
        return true;
    }
    if(first == second){
        return lhs < rhs;
    }
    return false;
}));
}
```