Lesson 2 - 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:

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
</table>

• 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:

  ```c
  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*)
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:
    
    \[
    \begin{align*}
    \text{int } & \text{ TC;} \\
    & \text{scanf("%d", &TC);} \\
    & \text{while (TC--) \{}} \\
    \end{align*}
    \]

  • Read until EOF:
    
    \[
    \begin{align*}
    \text{int } & \text{ a, b;} \\
    & \text{while (scanf("%d %d", &a, &b) != EOF) \{}} \\
    \end{align*}
    \]

  • Read until the values are 0:
    
    \[
    \begin{align*}
    \text{int } & \text{ a, b;} \\
    & \text{while (scanf("%d %d", &a, &b), (a || b)) \{}} \\
    \end{align*}
    \]
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

```c
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

```c
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:

```cpp
#include <vector>
```

• STL interfaces are implemented inside the `std` namespace:

```cpp
using namespace std;
```

• The data type is specified when creating an instance:

```cpp
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

```c
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`
- `hellll world`
• `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`
unordered_map (hash table)

- unordered_map<string, int> ht;
- ht["key"] = 5;
- cout << "ht["key"] = " << ht["key"] << endl;
- if (ht.find("key") != ht.end())
  - cout << "key found" << endl;
- else
  - cout << "key not found" << endl;
- cout << "second=" << ht["second"] << endl;
  // Implicitly create an item (val=0) for non-existent keys.
- ht["third"]++; // Increase by 1 if the key already exists in the hash table. If not, insert the key and set the value as 1.
- for (auto& p : ht)
  - cout << "key=" << p.first << " " << "val=" << p.second << endl;

- ht[key]=5
- key found
- second=0
- key=key val=5
- key=second val=0
- key=third val=1
unordered_map (hash table)

• unordered_map - hash table in O(1)-keys must have hash function so int, string, long long and so on is OK
• but!! pair or struct aren’t OK because they don’t have hash function so.. you can implement or use map.
vector<int> arr = {5, 3, 6, 3, 2, 1, 1};
priority_queue<int> max_heap(arr.begin(), arr.end()); // built in o(n)
cout << "my max_heap" << endl;
while (!max_heap.empty()) // goes over the heap
{
    cout << max_heap.top() << endl;
    max_heap.pop(); // no return val
}
priority_queue<int, vector<int>, comparator> min_heap;
min_heap.push(6); min_heap.push(8); min_heap.push(1); min_heap.push(2); // log(n)
cout << "my min_heap" << endl;
while (!min_heap.empty())
{
    cout << min_heap.top() << endl;
    min_heap.pop();
}
priority_queue (min/max heap)

```cpp
struct comparator {
    // this already implement greater<int>()
    bool operator()(int i, int j) {
        return i > j;
    }
};
```
set (binary search tree)

- set<int> g;
- g.insert(5); g.insert(2); g.insert(8); g.insert(2); // log(n) each
- g.erase(2); // log(n)
- g.insert(1); g.insert(9);
- cout << "my set" << endl;
- for (auto& x : g)
  - cout << x << " ";
  - cout << endl;
- if (g.find(1) != g.end())//log(n)
  - cout << "1 has been found"<<endl;

- my set
- 1 5 8 9
- 1 has been found
set - lower/upper bound

- auto itr = g.lower_bound(1); // find the first element that does not compare less than val, return itr
  - cout << *itr << endl;
  - itr++; // need to check itr != g.end()
  - cout << *itr << endl << endl;
  - itr = g.lower_bound(2);
  - cout << *itr << endl;
  - itr++;
  - cout << *itr << endl << endl;
  - itr = g.upper_bound(1); // find the first element that greater than val, return itr
  - cout << *itr << endl;
  - itr++;
  - cout << *itr << endl << endl;
  - itr = g.lower_bound(2);
  - cout << *itr << endl;
  - itr++;
  - cout << *itr << endl << endl;

- 1
- 5
- 5
- 8
- 5
- 8
- 5
- 8
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
  • nth_element – Find the nth element in a given array by performing partition 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...
• 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:

```cpp
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()); // nlog(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)
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 {
• } while (next_permutation(myints, myints + 3));

• 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
In Practice

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

```cpp
#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 two online judges:
  • Live Archive: https://icpcarchive.ecs.baylor.edu
  • UVa Online Judge: https://uva.onlinejudge.org/

• Submission instructions:
  • 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:
    • For Live Archive problems, the file name will be icpc_##.cpp, where #### is the problem ID.
    • For UVa problems, the file name will be uva_##.cpp, where #### is the problem ID.
  • One team member can submit for each team (when working in teams).
    • Please include the IDs of your team members.
General Tips

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