Implementation in Competitive Programming

Oliver-Matis Lill

March 27, 2017
Lecture Structure

At the beginning I will hold a lecture
Rest of the lecture will be spent on problem solving

Subjects

- Work environment
- Standard library
- Implementation tips
My Programming Environment

Oliver-Matis Lill

Implementation in Competitive Programming
- Bash on Ubuntu on Windows - Linux command line on windows, more convenient than a virtual machine
- Notepad++ - Good text editor for windows
- Windows Explorer - You can run stuff from the address bar
- g++ - Compiles C++ files
- gdb - Debugging tool
Most languages have their own standard library
- Contains a lot of functions, objects and other stuff that give you a lot of power
- Accessible in almost any environment (including in programming contests)
- Very useful in contests
Elements from the Standard Library

Some useful C++ standard library functions/classes:

1. `vector` - resizeable array of objects
2. `sort` - sorting function. Can use your own predicate. $O(N \log N)$ complexity
3. `lower_bound`/`upper_bound` - binary search on sorted array. $O(\log N)$ complexity
4. `priority_queue` - heap. $O(\log N)$ insertion, deletion, maximum finding
5. `set`/`map` - very powerful self-balancing binary tree. $O(\log N)$ insertion, deletion, lookup. In practice, these operations are slower than binary search and heap operations

Problems 1, 2, 3 test your skill with the Standard Library
Learn to use some standard library reference. My favorite for C++ is: http://www.cplusplus.com/reference/

Experiment with the standard library as much as possible (especially in competitive programming)

Useful everywhere
Implementation is creative work

There are many ways to implement the same thing, some better than others

Focus on the readability and elegance of your code. Those attributes are useful for example:

1. For avoiding and fixing bugs
2. For implementing very complicated stuff
3. For focusing when writing code
4. When sharing code with others

The following tips should be helpful in achieving that
Locality

- Declare your variables in as small scope as possible
- Makes it clearer where and how the variables are used
- Allows you to better reuse variable names
- Helps you avoid mixing up variables

Example

```cpp
//... includes, etc ...
int ind, a, b;
long long dp[20][20];

int main() {
    //... some code ...
    if(something) {
        //... use the variables ...
    }
}

→
//... includes, etc ...
int main() {
    //... some code ...
    if(something) {
        int ind, a, b;
        long long dp[20][20];
        //... use the variables ...
    }
}
```
Scope can be created without keywords (like if/while)
Useful for creating locality

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>// ... includes, etc ...</td>
</tr>
<tr>
<td>int main() {</td>
</tr>
<tr>
<td>// ... some code ...</td>
</tr>
<tr>
<td>int x, y, dx, dy;</td>
</tr>
<tr>
<td>// ... use those variables ...</td>
</tr>
<tr>
<td>// ... some unrelated code ...</td>
</tr>
<tr>
<td>double xd, yd, dxd, dyd;</td>
</tr>
<tr>
<td>// ... use those variables ...</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>→</td>
</tr>
<tr>
<td>// ... includes, etc ...</td>
</tr>
<tr>
<td>int main() {</td>
</tr>
<tr>
<td>// ... some code ...</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>int x, y, dx, dy;</td>
</tr>
<tr>
<td>// ... use those variables ...</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>// ... some unrelated code ...</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>double x, y, dx, dy;</td>
</tr>
<tr>
<td>// ... use those variables ...</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>
Keyword static

- Enables you to declare global variables in local scope, giving them the benefits of locality

**Example**

```c
#include <stdio.h>

int dp1[1001][1001];
double dp2[101][50001];

int function1() {
    //... calculation on dp1 ...
    return dp1[1000][1000]
}

double function2() {
    //... calculation on dp2 ...
    return dp2[100][50000];
}

// ... rest of the code ...
```

→

```c
#include <stdio.h>

int dp[1001][1001];

int function1() {
    static int dp[1001][1001];
    //... calculation on dp ...
    return dp[1000][1000]
}

double function2() {
    static double dp[101][50001];
    //... calculation on dp ...
    return dp[100][50000];
}

// ... rest of the code ...
```
OOP (Object Oriented Programming) is a very powerful tool. Learn to use it!

- Allows you to logically connect variables, functions and etc.
- Gives you more options to create locality
- Allows you to declare functions locally
Example

//... includes, etc ...
vector<int> arc[2][100000];
int weight[2][100000];

void construct(int i, int seed) {
  //uses arc[i] and weight[i]
}

int calculate(int i) {
  //uses arc[i] and weight[i]
}

int main() {
  construct(0, 15);
  construct(1, 2017);
  cout<<calculate(0)<<' ';
  cout<<calculate(1)<<'
';
}

→

//... includes, etc ...

struct Graph {
  vector<int> arc[100000];
  int weight[100000];
  //constructor
  Graph(int seed) {
    //uses arc and weight
  }
}

int calculate() {
  //uses arc and weight
}

int main() {
  static Graph g1(15), g2(2107);
  cout<<g1.calculate()<<' ';
  cout<<g2.calculate()<<'
';
}

Oliver-Matis Lill
Implementation in Competitive Programming
Initializer List

- Gives you a very elegant way to initialize objects
- Can be used to initialize standard library objects

Example

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

using namespace std;

struct Object {
    int cnt, val, size;
};

int main() {
    Object cur;
    cur.cnt = 1, cur.val = 10;
    cur.size = 2;
    vector<int> arr(3);
    arr[0] = 2, arr[1] = 15;
    arr[2] = 52;
}

→

#include <iostream>
#include <vector>

using namespace std;

struct Object {
    int cnt, val, size;
};

int main() {
    Object cur = {1, 10, 2};
    vector<int> arr = {2, 15, 52};
}
```

Oliver-Matis Lill
Implementation in Competitive Programming
Lambda Functions

- Allows you to create one-time, nameless local functions
- Makes sorting with a function simpler and more readable

Example

```cpp
//... includes, etc ...
bool pred(Object l, Object r) {
    return l.cnt*l.val < r.cnt*r.val;
}

int main() {
    //... some code ...
    vector<Object> objects;
    //... construct objects ...
    sort(objects.begin(), objects.end(), pred);
}
```

→

```cpp
//... includes, etc ...
int main() {
    //... some code ...
    vector<Object> objects;
    //... construct objects ...
    sort(objects.begin(), objects.end(), [] (Object l, Object r) {
        return l.cnt*l.val < r.cnt*r.val;
    });
}
```
Pointers

- More convenient way to refer to objects than indices, cur->next[1]->next[3] is better than next[next[cur][1]][3]
- Useful for finding multiple orderings of an array of objects

**Example**

```cpp
vector<Object*> byVal(n);
for(int i=0;i<n;i++) byVal[i] = &object[i];
sort(byVal.begin(), byVal.end(),
    [](Object* l, Object* r) {return l->val < r->val;});

vector<Object*> odd(n/2);
for(int i=1;i<n;i+=2) odd[i/2] = byVal[i];
sort(odd.begin(), odd.end(),
    [](Object* l, Object* r) {return l->size < r->size;});

for(int i=0;i<n/2;i++) odd[i]->result += i;
```
Summary

- Competitive programming can help direct you to write a lot of code.
- Use it to develop your implementation skill and to learn to write more elegant and readable code.
- The aforementioned tips are only tools, use them intelligently. Don’t try to forcibly use something if it’s not helpful.
- Implementation is creative work and can be quite interesting.