CP.Begin()

Coding Club, IIT Guwahati

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

int main()
{
    ios::sync_with_stdio(0);
    cin.tie(0);

    vector<string> CP = {"Week 1","Week 2","Week 3"};
    CP.push_back("Contest");

    cout << *(CP.begin());

    return 0;
}
```

>>> Week 1 <<<

What is Competitive Programming or CP?

- You are given a set of problems and you have to solve these problems in a fixed time.
- After solving, depending on your rank/percentile you will be awarded some virtual points.

Why should you do it?

- Mainly because it's competitive and programming :)
- The Competitive part gives you a feel of sports and the Programming part might help you in your **future courses, internships, and placements.**

CONTENTS

PREREQUISITES

(Expected Time: 1-2 days)

For this course, there are no prerequisites as such. But to start CP you will need a decent level of knowledge of at least one programming language.

- General Order of Preference:-

  C++ > Java > Python > Others
- It isn't that tough to switch between languages if you have a hold of at least one language.
- We recommend you start with C++ if it is going to be your first programming language.

If you already have knowledge of Python, we have linked resources for CP in python, so you can continue with the course. However, we recommend to learn C++ along the way as it's better for more difficult problems and will help you in your college curriculum as well.

SETTING UP AN IDE (Integrated Development Environment)

In simple words, an IDE is an environment that you will be using to write your code, compile it and view the output. A suitable and well-equipped IDE is very important, and not setting it up in the right way may cause trouble in the future.
(you may spend days fixing it).

Two of the most popular ones are:-

1. **VS Code**
   a. **Windows**
   b. **Mac**
2. **Sublime Text**

Don't mess up while setting it up otherwise you will easily waste days fixing it!!

STARTING WITH YOUR FIRST PROGRAMMING LANGUAGE

What we are going to recommend is the fastest and preferred way to learn C++ / Python. These are short videos, yet they are more than sufficient.

Resources for C++

1. **Bucky's Tutorial**: Watch lectures 1 to 41 and 71 to 73.

Resources for Python

1. **Programming with Mosh**: Watching the first three hours would be sufficient.
2. **Corey Schafer - STL**

Solve the First 10-20 problems from **hackerrank** just to get comfortable with the language you chose.

Note: **Do not waste much of your time learning the language** (getting into tiny details).

TOOLS FOR COMPETITIVE PROGRAMMING

Note: You don't need to set these up to begin with CP, however, along the way these tools become really helpful to your journey. You may revisit this when you get comfortable with CP.

1. **VS Code extensions:**
   To install these extensions, look for the "Extensions" button on the left sidebar. Then you can search for and install them
   - **Code Runner**: With this, you can directly run the code without having to compile in the terminal .

- **Competitive Programming Helper:** It automatically reads test cases from the website which you won't have to write again and again. To work with this you also have to install the **Competitive Companion Chrome extension** .

2. Sublime Text Packages:

Similar to the cph extension in VS Code, you can find **FastOlympicCoding** in Sublime which works with Competitive Companion .

3. User Code Snippets/Template code:

These are reusable code pieces that can help you avoid re-writing the same code. You can make them in both VS Code and Sublime Text.

4. Header Files

Including all libraries required can be a tedious task. So we prefer including *<bits/stdc++.h>* file instead, which already contains all the libraries. It is not used in usual C++ codes as it takes a long time to compile. However, that is not important in CP. It is available from **C++14 and above** and it is **not directly available for Mac users.**

Mac users can refer **this** video. The folder with include files may not be the same. To find that, you can make a C++ file, type *#include <iostream>* and Ctrl + Click on it to see the file location.

RESOURCES

1. **Handbook (CPH)**:- You can find 90% of the things you want to learn in this book, but reading this book completely and then starting to solve problems is not recommended. Just refer to the book when you come across a topic that you haven't heard about before.

2. **CSES**:- The author of the Handbook has also made a compilation of 300 really good and conceptual problems which covers almost all CP topics. But it doesn't contain problems sorted according to difficulty so you will have to estimate it using solve count.

3. **USACO Guide**:- This is literally the "one-stop shop". This website is generally referred to as USA computing Olympiad aspirants and it has everything in sorted order. And it has divisions like Bronze, Silver, and Gold. It will mostly refer to the Handbook and explain solutions to CSES problems as well.

4. **CP-Algorithms**:- Contains all CP topics but again not in sorted order of difficulty. So, refer to it only if you are stuck on a topic and couldn't find it in the Handbook

WEBSITES

Some websites from where you can learn and compete and some important competitions**.**

1. **Codeforces**:- Most popular and widely used platform.
    - Contests are mainly of the following types:- Div 4, Div 3, Div 2, Div 1, Div 1+2, Educational Rounds, and Global Rounds.
    - We recommend participating in all rounds without worrying about rating changes(ratings are temporary but the experience is unique).
    - Do try to solve all the problems whose ratings are <= your rating + 300. If you are not able to solve the problems, refer to their editorial. If you still don't understand the solution, discuss it with your friends or ask your doubts in the competitive-programming-discussion channel on our Discord.
2. **Atcoder**:- Educational and Weekly contests with quality problems.
    - 3 types of contests:- Atcoder Beginner/Regular/Grand contest
    - Must participate in Atcoder Beginner Contests (100 mins a week isn't that much)
3. **Codechef**:-
    - Monthly contests:- Long challenge and Lunch-time
    - Weekly contests named starters

It is highly recommended to start giving contests regularly once you have some basic knowledge about cp.

IMPORTANT YEARLY COMPETITIONS

* We will send you a reminder before all these contests when to participate.

1. **Google Contests**:- Don't miss any of them unless you have an exam at the same time.
    a. **Google Kickstart**:-
        i. 8 times a year. All 8 contests are independent.
        ii. Sometimes the problems are irritating but still, it's a google contest.
    b. **Google Codejam**:-
        i. Held once a year. Probably the biggest CP contest.
        ii. 5 rounds in increasing order of difficulty, with elimination in each Round.
        iii. Prelims->Round A->Round B->Round C-> world finals
    c. **Google Hashcode**:-
        i. Held once a year and is very different from normal CP contests.
        ii. You have to come up with a solution to a problem and improve it to gain more points. Problems couldn't be solved completely.
2. **Meta HackerCup**:- It is also a once-a-year contest in topological order just like Google Code Jam.
3. **ACM ICPC**:- Held once a year and the Biggest CP contest for college students. Also in topological order.
    a. Online -> Regionals -> World Finals
4. **Codechef SnackDown**:- Held once every two years. The Next will be in 2023.

IMPLEMENTATION

(Expected Time: 1 day)

After we have learned a language, we can now get started with actual problem-solving.

Problem-solving requires understanding the problem statement and designing an effective algorithm that successfully solves the task at hand.

The efficiency of algorithms is very important in competitive programming. Usually, it is easy to design an algorithm that solves the problem slowly, but the real challenge is to invent a **fast algorithm**. If the algorithm is too slow, it will get only partial points or no points at all.

Thus comes into the picture the concept of **time complexity**. By calculating the time complexity, we can find out whether the algorithm is fast enough without implementing it.

Here are some resources which will help you understand time complexity: -

1. **CPH** (Competitive Programmers' Handbook) Chapter 2.
   - Do read this chapter thoroughly without skipping to get a complete understanding of Big O notation etc.
2. **Big-O notation in 5 minutes**
3. This **blog** might help you in avoiding Time Limit Exceeded error

Some more aspects of a language become handy while doing CP problems. One such aspect, which we usually ignore when learning the language is that of range of numbers.

In Python3, the int type stores large enough numbers, so you don't have to worry about the value which will be stored in the variable.

In C++, **int** is a data type that takes up 32 bit of space and can store integers ranging from **-2,147,483,648 to 2,147,483,647**. However, the answers in the problems very often go over that range and give wrong output. This is called **integer overflow**. Hence, we can use **long long** data type over **int** in such cases. You can read more about it **here**. This can be really frustrating if not taken care of as the code gives wrong output even when your approach is correct.

Following are some challenging problems based mainly on implementation:-

- **Increasing Array (Solution)**
- **Permutations (Solution)**
- **Number Spiral (Solution)**
- **Chloe and the sequence (Solution)**
- **Viki and Squares (Solution)**
- **Trailing Zeroes (Solution)**
- **Masha and two friends (Solution)**

*Try to attempt a question for a minimum of 30 minutes before checking the solution.

*We have attached the codes with solutions. If you still have any doubts feel free to ask us in the discord channel.

*In general, the solutions to the problems can be found in the Editorial/ Tutorial section on websites like Codeforces, Atcoder etc. For CSES, most of the solutions can be found on google.

STL

(Expected Time: 1-2 days)

The **Standard Template Library (STL**) is a set of C++ template classes to provide common programming **data structures** and functions such as lists, stacks, arrays, etc.

At the core of the C++ Standard Template Library are the following three well-structured components – **containers, algorithms, and iterators**. Basic knowledge of STL is quite handy, and in fact, necessary for competitive programming.

Resources:

- **Standard Template Library in C++ - Great Learning**
- **The C++ Standard Template Library (STL) - GeeksforGeeks**
- **Watch this video to learn the basics of STL**
- Read Ch-4 of **CPH** (pg 35-pg 43)

Frequently used containers:

- **Array**: Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value. Vector is used more frequently compared to array since the former is dynamic in nature. However, **higher dimensional arrays** are simpler to declare than vectors.

- **Vector**: One of the most powerful and frequently used STL containers. Vectors are the same as dynamic arrays with the ability to resize themselves automatically when an element is inserted or deleted. Study these functions only:
  push_back(), pop_back(), sort(), upper_bound() , lower_bound()* , begin(), end(), rbegin(), size(), resize(), accumulate(), binary_search()* clear() and find()*.

- **Pair:** Pair is a container that contains two values. It is used to combine two values and associate them even if they are of different types. The first and second values of a pair 'p' can be accessed using *p.first* and *p.second* .

- **Set**: A container that keeps a **unique** copy of every element in **sorted** order. Study these functions only:
  insert(), erase(), begin(), end(), rbegin(), size(), lower_bound(), upper_bound()*, find()* and clear()*

- **Multiset**: Similar to a set, but it can also store multiple copies of an element.

- **Map**: Maps are containers that store elements in a mapped fashion. Each element has a key value and a mapped value. Study these functions only:
  insert(), erase(), begin(), end(), rbegin(), find(), clear(), size()
  lower_bound(), upper_bound(), and the operator []

*The use of these containers and their functions will be explained in the upcoming weeks

Not-so-frequently used containers:

- **Stack**: Stacks are a type of container adaptor that operate in a Last In First Out (**LIFO**) type of arrangement. Elements are added at one end (top) and are removed from that end only.

- **Queue**: Queues are a type of container that operates in a First In First Out (**FIFO**) type of arrangement, unlike Stack. Elements are inserted at the back (end) and are deleted from the front.

- **Priority Queue**: Priority queues are queues such that the first element is either the greatest or the smallest of all elements in the queue and elements are in nonincreasing order.

- **Deque**: Double-ended queues are a special case of queues where insertion and deletion operations are possible at both ends

- **MultiMap**: Multimap is similar to a map with the addition that multiple elements can have the same keys

STL EQUIVALENT IN PYTHON

Frequently used containers:

- **List**: Lists are used to store multiple values(they may not be of the same data type) in a single variable, instead of declaring separate variables for each value.

- **Dictionary**: It is equivalent to an unordered map of C++ STL. Maps are containers that store elements in a **mapped** fashion. Each element has a key value and a mapped value. It does not store elements in a sorted manner like C++ Map.

- **Sets**: A set is an unordered collection of items. Every set element is unique (no duplicates). Sets can also be used to perform mathematical set operations like union, intersection, etc.

Some more containers:

- **Deque**: It can be used to implement **queue** and **deque** similar to those given in C++ STL.

- **Heapq**: The property of this data structure is that each time the smallest heap element is popped. It is the python equivalent of a priority queue.

Frequently used Modules:

- **Math**: It contains all the basic Math operations ranging from ceil, and floor to factorial, pow, etc.

- **Sys**: It's generally used for some specific functions revolving around the std input and output. Eg: *sys.stdin.readline()* (which takes input faster than the normal input function), *sys.stdout.flush()* (used in interactive problems) , *sys.setrecursionlimit(int)* (to increase the recursion limit) , etc

- **Bisect**: Has 2 functions, *bisect_right(list, key)* which finds and returns the index of the minimum element in the list greater than the key, and *bisect_left(list, key)* which finds and returns the index of the maximum element in the list lesser than key both in *O(log(len(list))* complexity.

- **Random**: Has its use in some of the probability-based questions and in question generation. Some common functions of random:
  Random.randint(a,b) which generates a random integer between a and b .
  Random.shuffle(list) returns a shuffled version of a given list in O(len(list)) .

Following are some practice problems:-

- **Distinct Numbers (C++Solution) (Python-soln)**
- **Remove Duplicates (C++Solution) (Python-soln)**
- **Registration System (C++Solution) (Python-soln)**
- **T-Shirt Buying? (C++Solution) (Python-soln)**
- **Subarray Sums (C++Solution)(Python-soln)**
- **Scope (C++Solution)(Python-soln)**
- **Movie Festival (C++Solution) (Python-soln)**

MATHS

(Expected Time: 0-1 days)

There are several questions in CP, which can be solved only by observations and logical reasoning skills. No algorithm is needed as such to solve these kinds of problems. Such questions test the intuitive/ mathematical skills of the programmer, and not his/her coding skills.

Try to solve these given problems without using any known algorithms:-

- **Little Elephant**
- **Hard Calculation**
- **Who's Opposite**
- **Comma**

Additional Questions for practice :

- **Histogram Ugliness**
- **Pythagorean Triples**
- **Fraction Floor Sum**
- **Cat Cycle**

AD-HOC PROBLEMS *(Expected Time: 0-1 days)*

Ad hoc problems are those whose algorithms do not fall into standard categories with well-studied solutions. There are no general techniques to solve them. They are intuition-based problems, and you may understand better by solving these:

- **Teleportation**
- **Funny Permutation**
- **Three Doors**
- **Sleepy Cow Herding**

Additional Questions for Practice:

- **Exchange**
- **Do you know your ABCs?**
- **Sleepy Cow Sorting**
- **Difference of GCDs**