

**Department of AI&ML**

**II YEAR-II SEMESTER**

**Subject: Operating Systems Using C++**

**Subject code:** **MR23-1CS0224**

**Pongal Holidays Assignment**

**By 2311CS020200**

**Golla Prashanth**

**Aiml Omega**

**1. Conditional statements (Hacker rank)**

**Task**

Read numbers from stdin and print their sum to stdout.

**Input Format**

One line that contains space 3-separated integers :a ,b , and c .

# Constraints

 1<=a,b,c<=1000 **Output Format**

Print the sum of the three numbers on a single line.

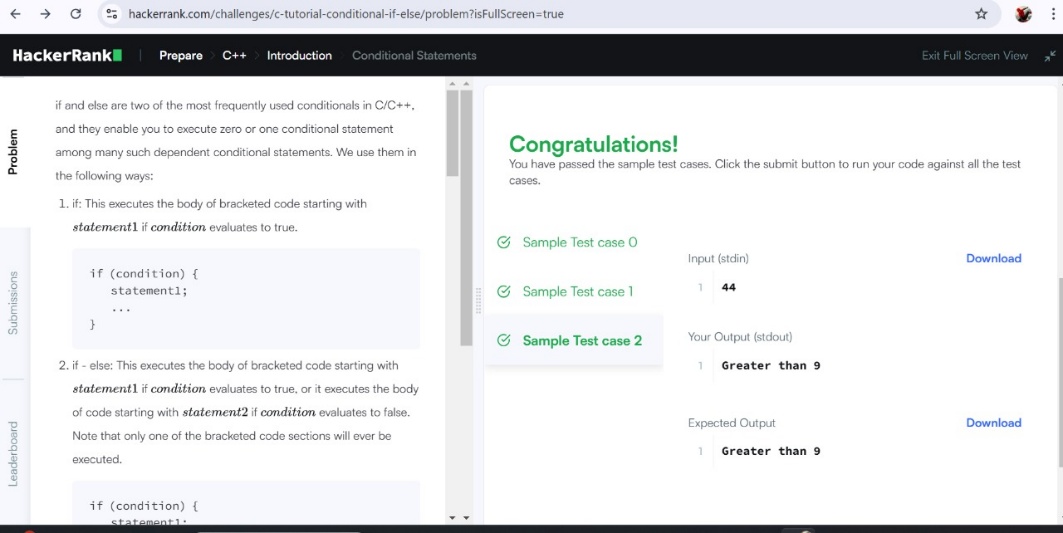
**Sample Input**

1 2 7

**Sample Output**

10

Ans.



**2. Functions (Hacker rank)**

**Input Format**

Input will contain four integers – a, b ,c, d.one per line.

# Output Format

Return the greatest of the four integers.

*PS:* I/O will be automatically handled.

# Sample Input

3

4

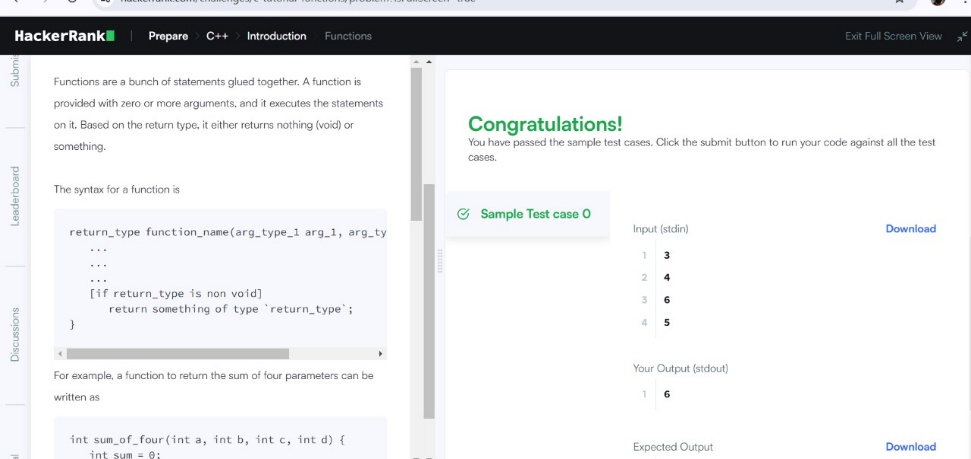
6

5

**Sample Output**

6

Ans.



**3. Pointer (Hacker rank)**

**Input Format**

Input will contain two integers a and b. separated by a newline.

# Sample Input

4

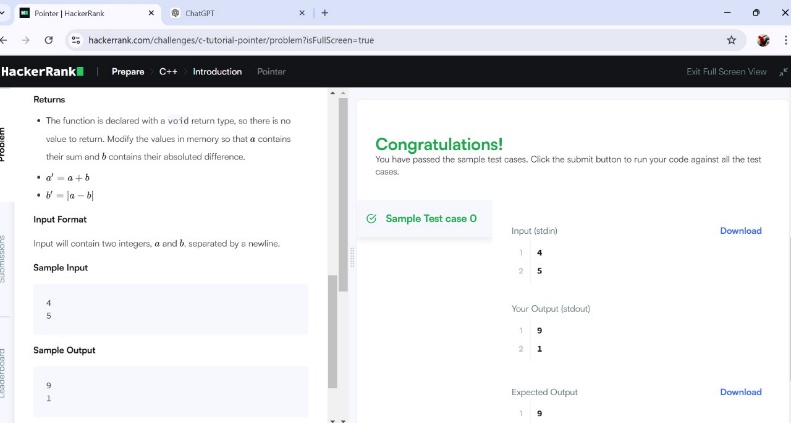
5

# Sample Output

9

1

Ans.



# 4. Structures (Hacker rank)

**You have to create a struct, named *Student*, representing the student's details, as mentioned above, and store the data of a student.**

# Input Format

Input will consist of four lines.

The first line will contain an integer, representing *age*.

The second line will contain a string, consisting of lower-case Latin characters

('a'-'z'), representing the *first\_name* of a student.

The third line will contain another string, consisting of lower-case Latin characters ('a'-'z'), representing the *last\_name* of a student.

The fourth line will contain an integer, representing the *standard* of student.

*Note:* The number of characters in *first\_name* and *last\_name* will not exceed 50.

# Output Format

Output will be of a single line, consisting of *age*, *first\_name*, *last\_name* and *standard*, each separated by one white space. *P.S.:* I/O will be handled by HackerRank.

# Sample Input

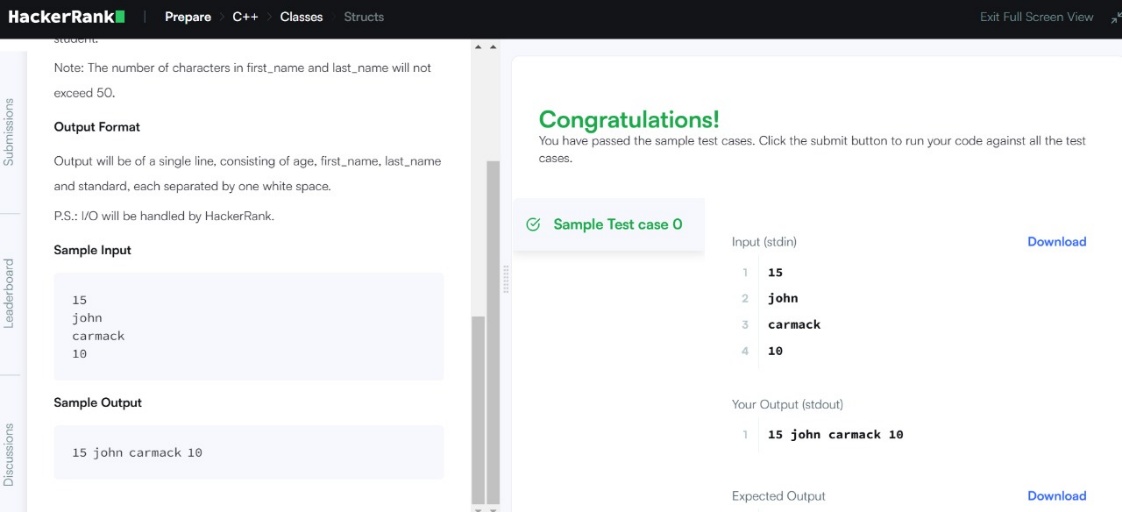
15 john carmack

10

**Sample Output**

15 john carmack 10

Ans.



**5. Strings (Hacker rank)**

# Input Format

You are given two strings a and b separated by a new line. Each string will consist of lower case Latin characters ('a'-'z').

# Output Format

In the first line print two space-separated integers, representing the length of a and b respectively.

In the second line print the string produced by concatenating a and b (a+b).

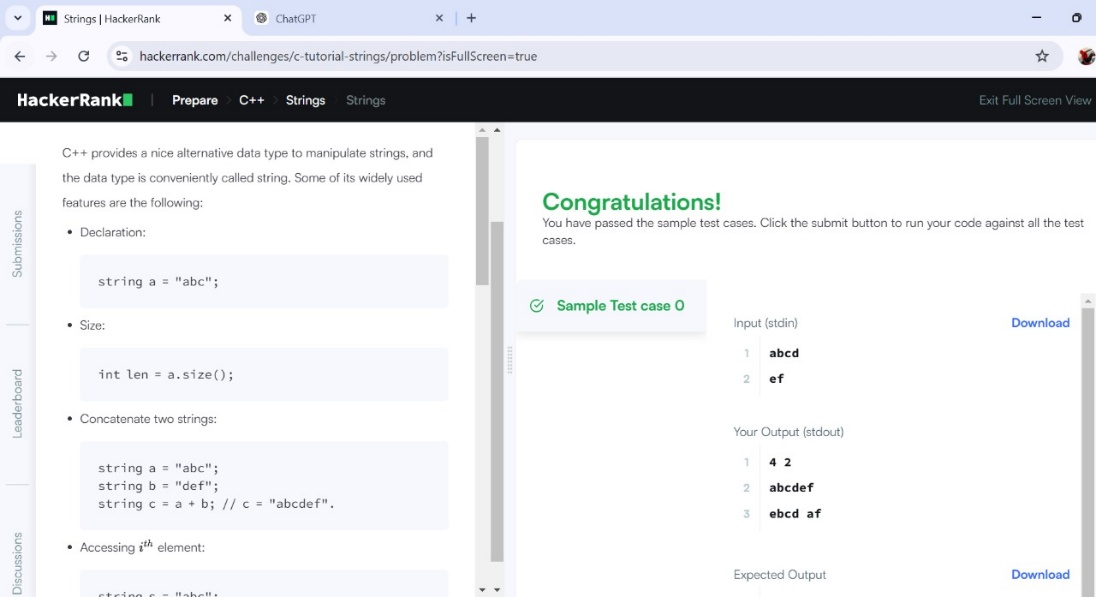
In the third line print two strings separated by a space a’ and b’ and a’ and b’ are the same as a and b , respectively, except that their first characters are swapped. **Sample Input**

abcd ef

# Sample Output

4 2 abcdef ebcd af

Ans.



**6. Recursion Problem** **(Hacker rank)**

**Task**

Write a recursive function to calculate a^b (a raised to the power of b).

Input Format

1. Two integers a and b (0 ≤ a ≤ 100, 0 ≤ b ≤ 10).

Constraints

* 0 ≤ a ≤ 100
* 0 ≤ b ≤ 10

Output Format

Print the result of a^b.

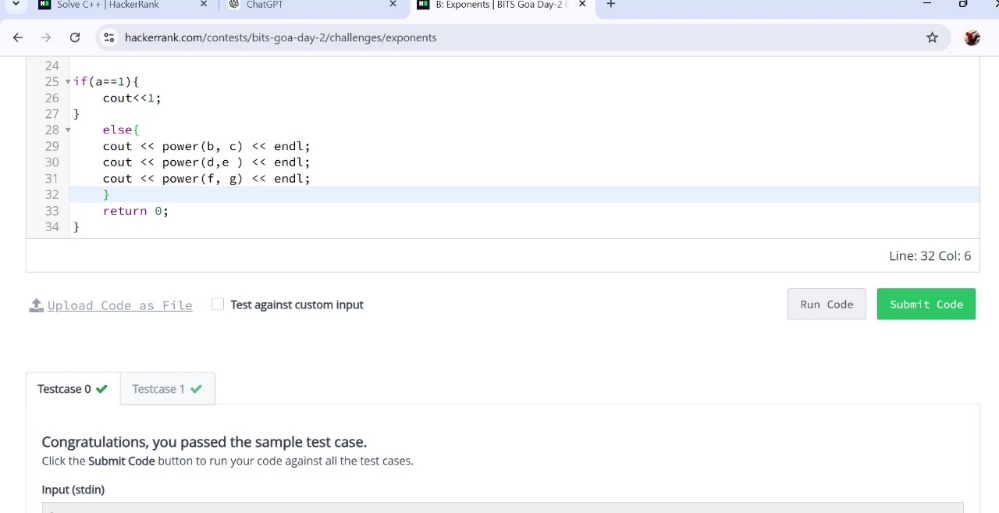
Sample Input

2 3

Sample Output

8

Ans.



1. **Exception Handling (Hacker rank)**

Problem Statement

Write a program that takes two integers a and b and calculates their division (a / b). If the divisor b is zero, the program should throw and handle an exception to avoid crashing.

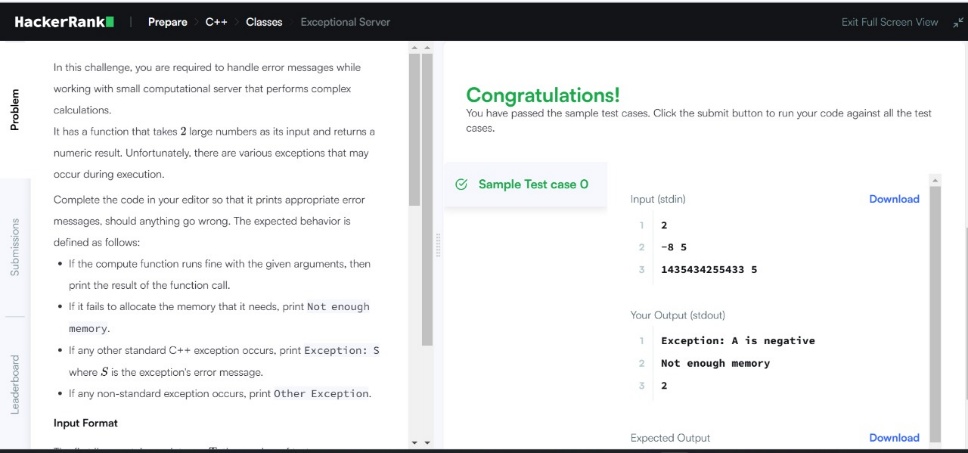
Input Format

1. Two space-separated integers a and b.

Constraints

 -10^9 ≤ a, b ≤ 10^9

Ans.



1. **For loop**

# Input Format

You will be given two positive integers, a and b (a<=b), separated by a newline.

# Output Format

For each integer in the inclusive interval[a ,b] :

* If 1<=n<=9 , then print the English representation of it in lowercase. That is

"one" for 1 , "two" for 2, and so on.

* Else if n>9 and it is an even number, then print "even".
* Else if n>9 and it is an odd number, then print "odd".

# Sample Input

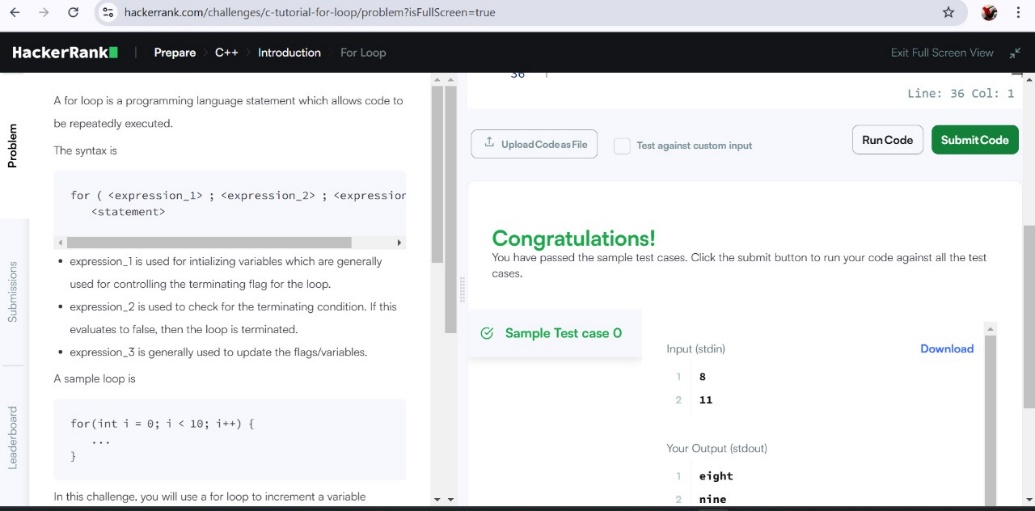
8

11

# Sample Output

eight nine even odd

Ans.



# 9. Arrays (Leetcode)

There are a total of numCourses courses you have to take, labeled from 0 to numCourses - 1. You are given an array prerequisites where prerequisites[i] = [ai, bi] indicates that you must take course bi first if you want to take course ai.

**Input format:**

For example, the pair [0, 1], indicates that to take course 0 you have to first take course 1.

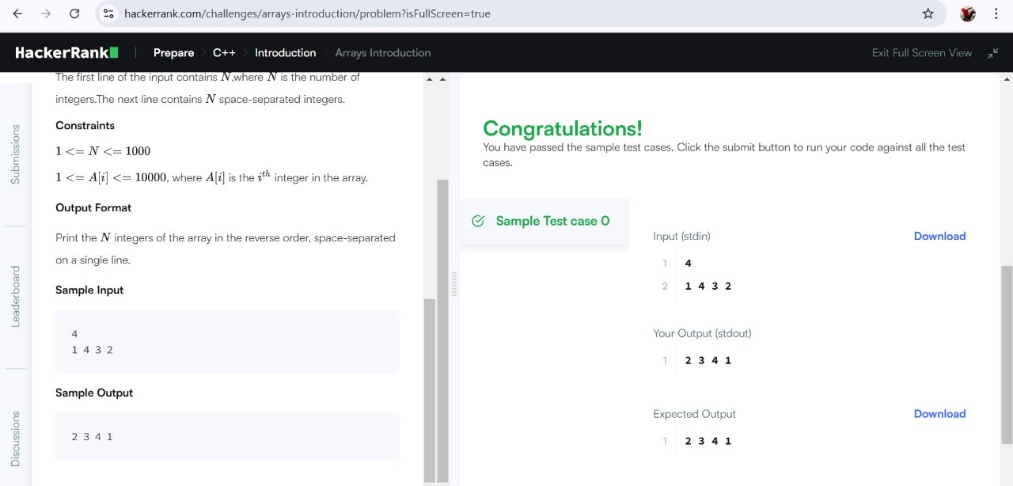
Return true if you can finish all courses. Otherwise, return false.

# Output Format

There are a total of 2 courses to take.

To take course 1 you should have finished course 0. So it is possible.

Ans.



1. **Case study: Implement First come first serve and Shortest job first CPU scheduling algorithms using c++.**

**Ans.**

**1. First Come First Serve (FCFS) Scheduling**

FCFS is the simplest CPU scheduling algorithm where the process that arrives first gets executed first.

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct Process {

int id;

int arrivalTime;

int burstTime;

int completionTime;

int waitingTime;

int turnaroundTime;

};

void findWaitingTime(vector<Process>& processes) {

processes[0].waitingTime = 0;

for (int i = 1; i < processes.size(); i++) {

processes[i].waitingTime = processes[i - 1].completionTime - processes[i].arrivalTime;

}

}

void findTurnaroundTime(vector<Process>& processes) {

for (int i = 0; i < processes.size(); i++) {

processes[i].turnaroundTime = processes[i].burstTime + processes[i].waitingTime;

}

}

void findCompletionTime(vector<Process>& processes) {

processes[0].completionTime = processes[0].arrivalTime + processes[0].burstTime;

for (int i = 1; i < processes.size(); i++) {

processes[i].completionTime = max(processes[i].arrivalTime, processes[i - 1].completionTime) + processes[i].burstTime;

}

}

void FCFS(vector<Process>& processes) {

findCompletionTime(processes);

findWaitingTime(processes);

findTurnaroundTime(processes);

cout << "Process ID\tArrival Time\tBurst Time\tCompletion Time\tWaiting Time\tTurnaround Time" << endl;

for (auto& process : processes) {

cout << process.id << "\t\t" << process.arrivalTime << "\t\t" << process.burstTime << "\t\t"

<< process.completionTime << "\t\t" << process.waitingTime << "\t\t" << process.turnaroundTime << endl;

}

}

int main() {

vector<Process> processes = {{1, 0, 5}, {2, 1, 3}, {3, 2, 8}, {4, 3, 6}};

FCFS(processes);

return 0;

}

**Explanation:**

* **Process Structure:** Contains the process ID, arrival time, burst time, completion time, waiting time, and turnaround time.
* **findWaitingTime():** Calculates the waiting time for each process.
* **findTurnaroundTime():** Calculates the turnaround time for each process.
* **findCompletionTime():** Calculates the completion time for each process based on FCFS logic.
* **FCFS():** Main function that calls other functions and prints the results.

**2. Shortest Job First (SJF) Scheduling**

SJF is a non-preemptive CPU scheduling algorithm that selects the process with the shortest burst time for execution next.

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct Process {

int id;

int arrivalTime;

int burstTime;

int completionTime;

int waitingTime;

int turnaroundTime;

};

bool compare(Process p1, Process p2) {

return p1.burstTime < p2.burstTime;

}

void findWaitingTime(vector<Process>& processes) {

processes[0].waitingTime = 0;

for (int i = 1; i < processes.size(); i++) {

processes[i].waitingTime = processes[i - 1].completionTime - processes[i].arrivalTime;

}

}

void findTurnaroundTime(vector<Process>& processes) {

for (int i = 0; i < processes.size(); i++) {

processes[i].turnaroundTime = processes[i].burstTime + processes[i].waitingTime;

}

}

void findCompletionTime(vector<Process>& processes) {

processes[0].completionTime = processes[0].arrivalTime + processes[0].burstTime;

for (int i = 1; i < processes.size(); i++) {

processes[i].completionTime = max(processes[i].arrivalTime, processes[i - 1].completionTime) + processes[i].burstTime;

}

}

void SJF(vector<Process>& processes) {

sort(processes.begin(), processes.end(), compare); // Sort processes by burst time (Shortest Job First)

findCompletionTime(processes);

findWaitingTime(processes);

findTurnaroundTime(processes);

cout << "Process ID\tArrival Time\tBurst Time\tCompletion Time\tWaiting Time\tTurnaround Time" << endl;

for (auto& process : processes) {

cout << process.id << "\t\t" << process.arrivalTime << "\t\t" << process.burstTime << "\t\t"

<< process.completionTime << "\t\t" << process.waitingTime << "\t\t" << process.turnaroundTime << endl;

}

}

int main() {

vector<Process> processes = {{1, 0, 6}, {2, 1, 8}, {3, 2, 7}, {4, 3, 3}};

SJF(processes);

return 0;

}

**Explanation:**

* **compare():** Function to sort processes by burst time (for SJF).
* **findWaitingTime(), findTurnaroundTime(), findCompletionTime():** Functions similar to FCFS, calculating the corresponding times.
* **SJF():** Main function for SJF scheduling that sorts processes and then calculates the times.

**Output Example:**

**For FCFS:**

**Process ID Arrival Time Burst Time Completion Time Waiting Time Turnaround Time**

**1 0 5 5 0 5**

**2 1 3 8 4 7**

**3 2 8 16 6 14**

**4 3 6 22 13 19**

**For SJF:**

**Process ID Arrival Time Burst Time Completion Time Waiting Time Turnaround Time**

**4 3 3 6 0 3**

**1 0 6 12 3 9**

**3 2 7 19 10 17**

**2 1 8 27 18 2**