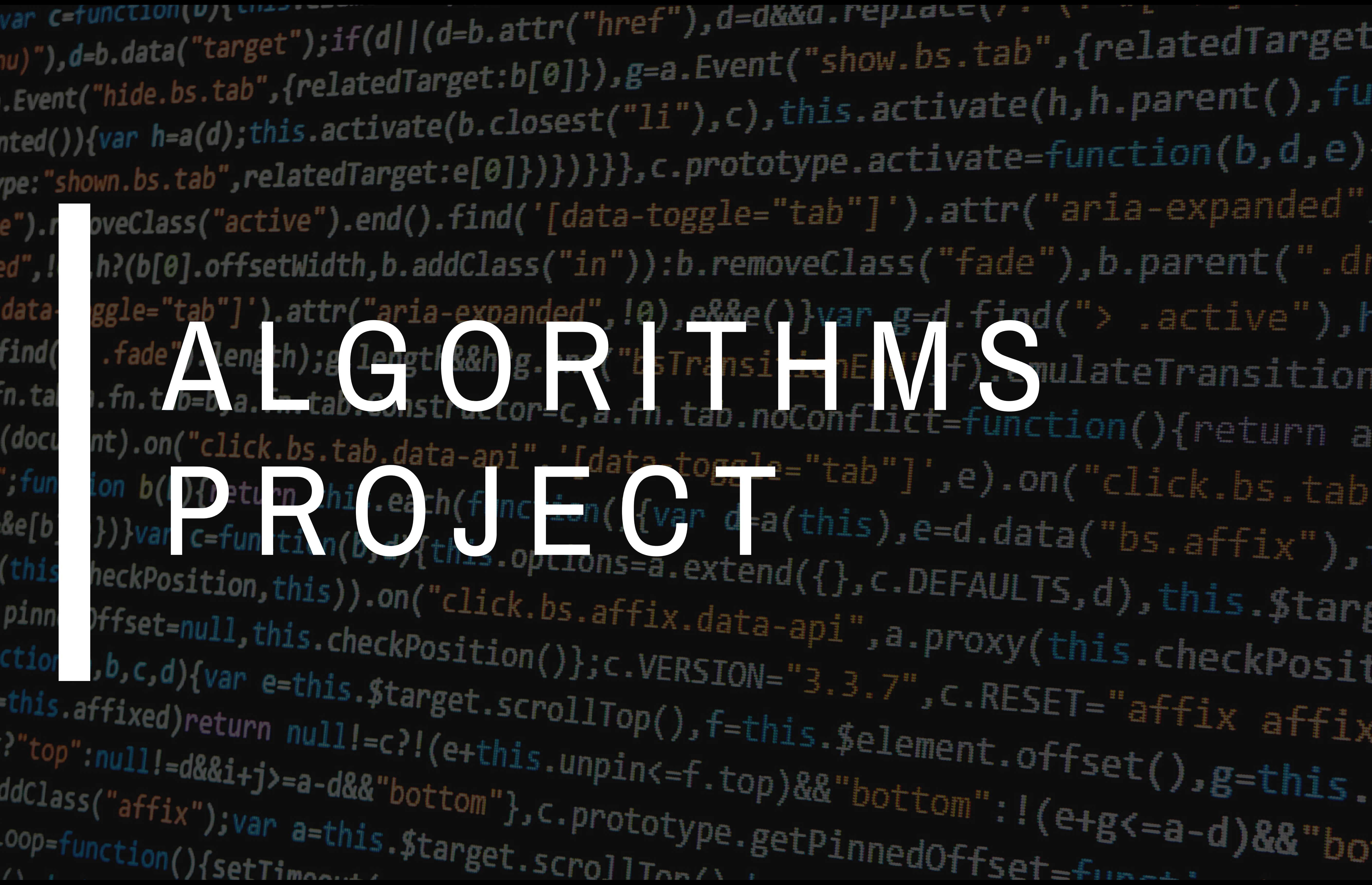
ALGORITHMS & DESIGN COURSE



ALGORITHMS PROJECT

TEAM (7)/ Thursday

ENG/ EZZ EL DEEN

TEAM MEMBERS & IDS

• Halla Ibrahim Ali	20201423108
• Habiba Mohamed Amin	20201378219
• Salma Mohamed AbdEl-Latif	20201378214
• Mariam Ahmed Ramadan	20201499017
• Mariam Ahmed Abu Baker	20201444727

TABLE OF CONTENT:

FIRST PROBLEM:

A. Given a set of N jobs where each jobi has a deadline and profit associated with it. Each job takes 1 unit of time to complete and only one job can be scheduled at a time. We earn the profit associated with job if and only if the job is completed by its deadline. Find the number of jobs done and the maximum profit. Jobs will be given in the form (Jobid, Deadline, Profit) associated with that Job.

Example 1:

Input

Jobs = {(1,4,20),(2,1,10),(3,1,40),(4,1,30)}

→ Steps(using greedy algorithm):

1) We will sort the jobs profit in descending order.

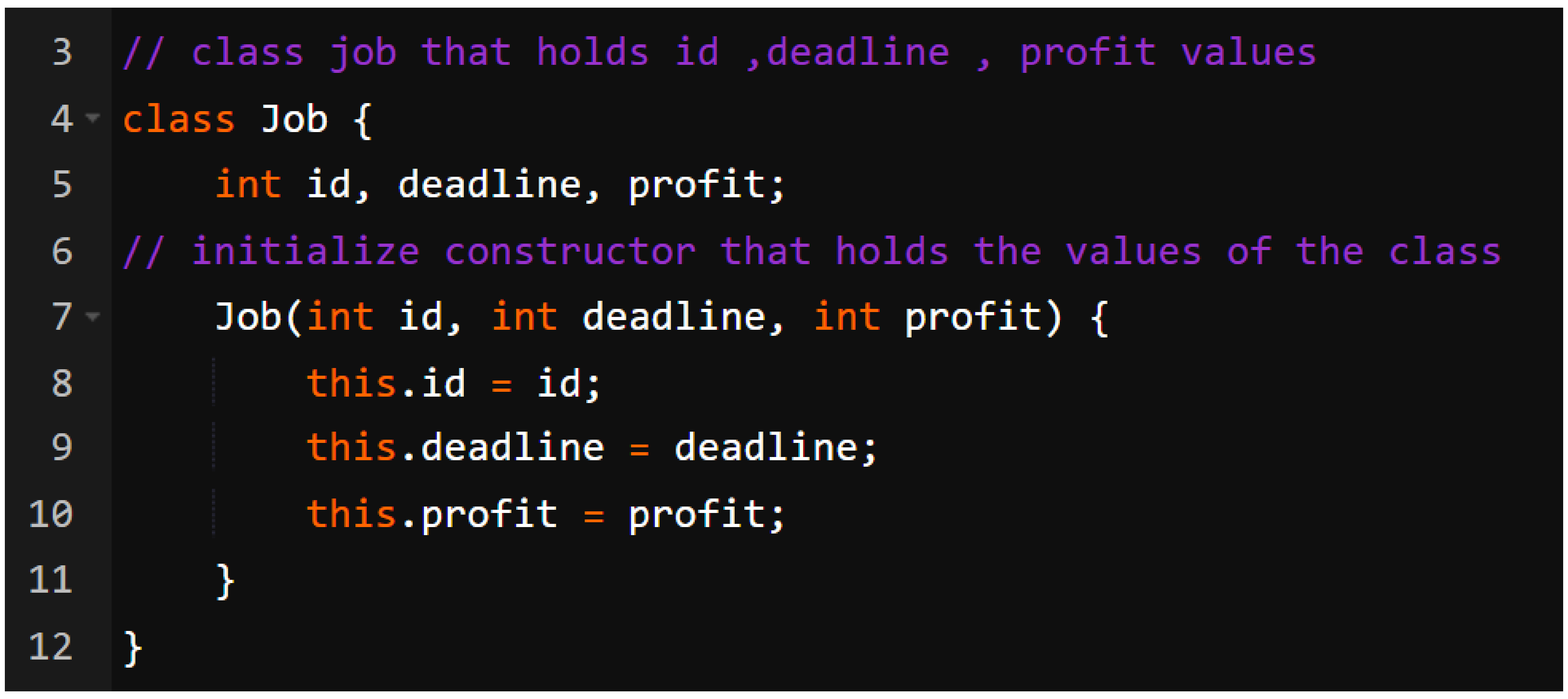
2) Initialize an array called "slots" of size equal to the maximum deadline among all jobs, and initialize all elements of the array to 0.

3) Iterate through the sorted jobs in order which if:

- **(i)** Starting from the job's deadline, find the first available slot in the "slot" array that is not already occupied. If a slot is found, schedule the job in that slot and mark the slot as occupied by setting its value to 1.
- **(ii)** If no available slot is found, skip the job.

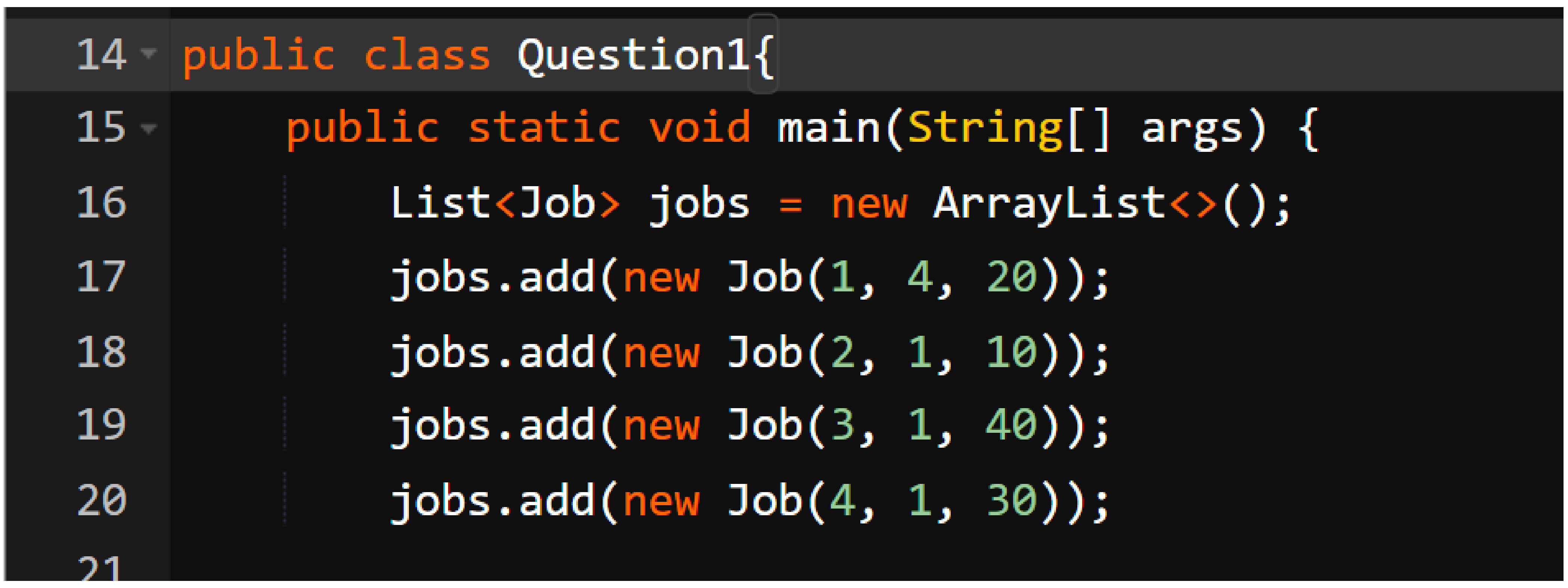
4) Count the number of jobs that were scheduled, and calculate the total profit earned by summing the profits of the scheduled jobs.

CODE & EXPLAINATION:



```java
// class job that holds id ,deadline , profit values
class Job {
    int id, deadline, profit;
// initialize constructor that holds the values of the class
    Job(int id, int deadline, int profit) {
        this.id = id;
        this.deadline = deadline;
        this.profit = profit;
    }
}
```

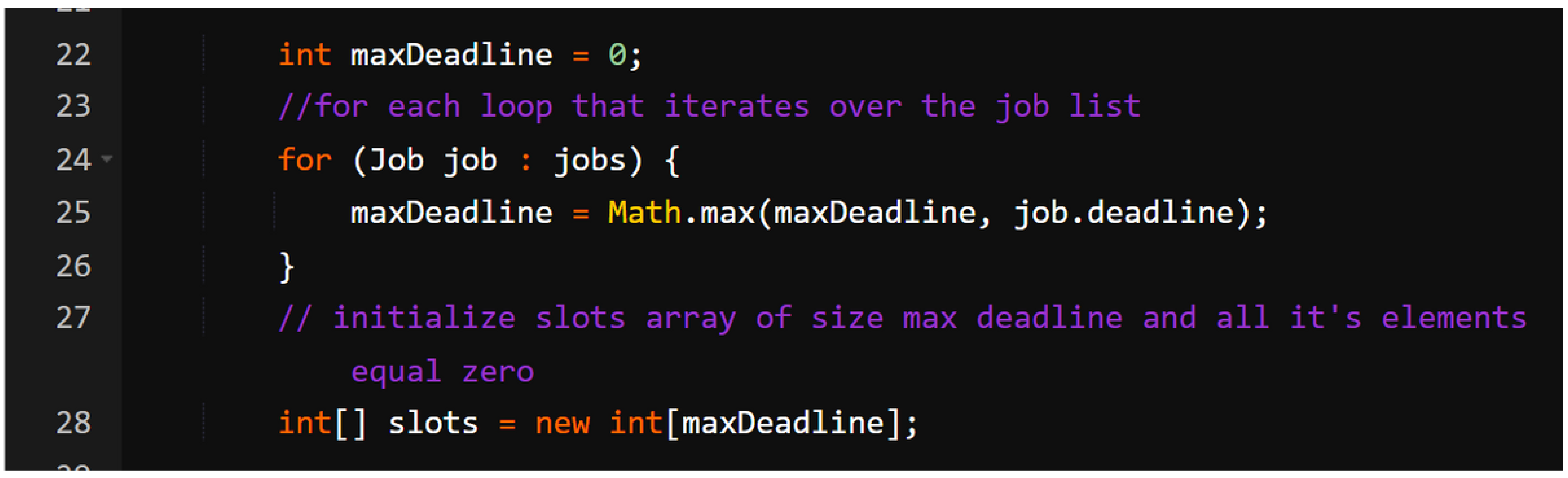
- The Job class has a constructor that takes three parameters - id, deadline, and profit.
- Initializes the corresponding instance variables with these values.
- This keyword is used to refer to the instance variables of the current object, also by defining a Job class in this way.
- We can create instances of the Job class to represent individual jobs with specific deadlines and profits.



```java
public class Question1{
    public static void main(String[] args) {
        List<Job> jobs = new ArrayList<>();
        jobs.add(new Job(1, 4, 20));
        jobs.add(new Job(2, 1, 10));
        jobs.add(new Job(3, 1, 40));
        jobs.add(new Job(4, 1, 30));
```

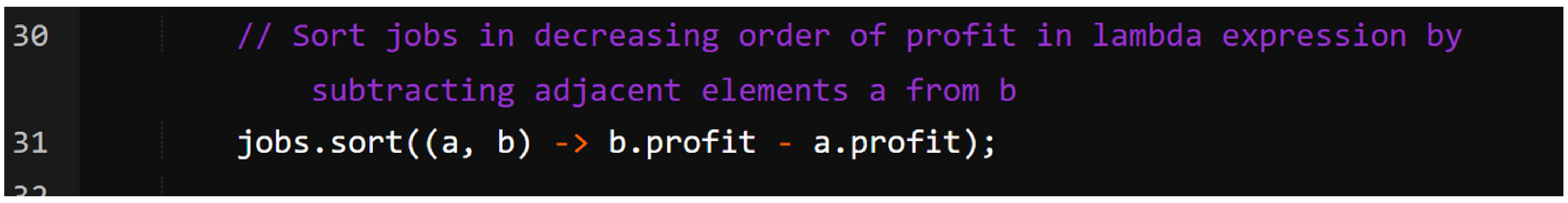
- creates a list of Job objects using an arraylist.
- adds four jobs to the list with specific values which are given in our question.

CODE & EXPLAINATION:



```
        int maxDeadline = 0;
        //for each loop that iterates over the job list
        for (Job job : jobs) {
            maxDeadline = Math.max(maxDeadline, job.deadline);
        }
        // initialize slots array of size max deadline and all it's elements
            equal zero
        int[] slots = new int[maxDeadline];
```

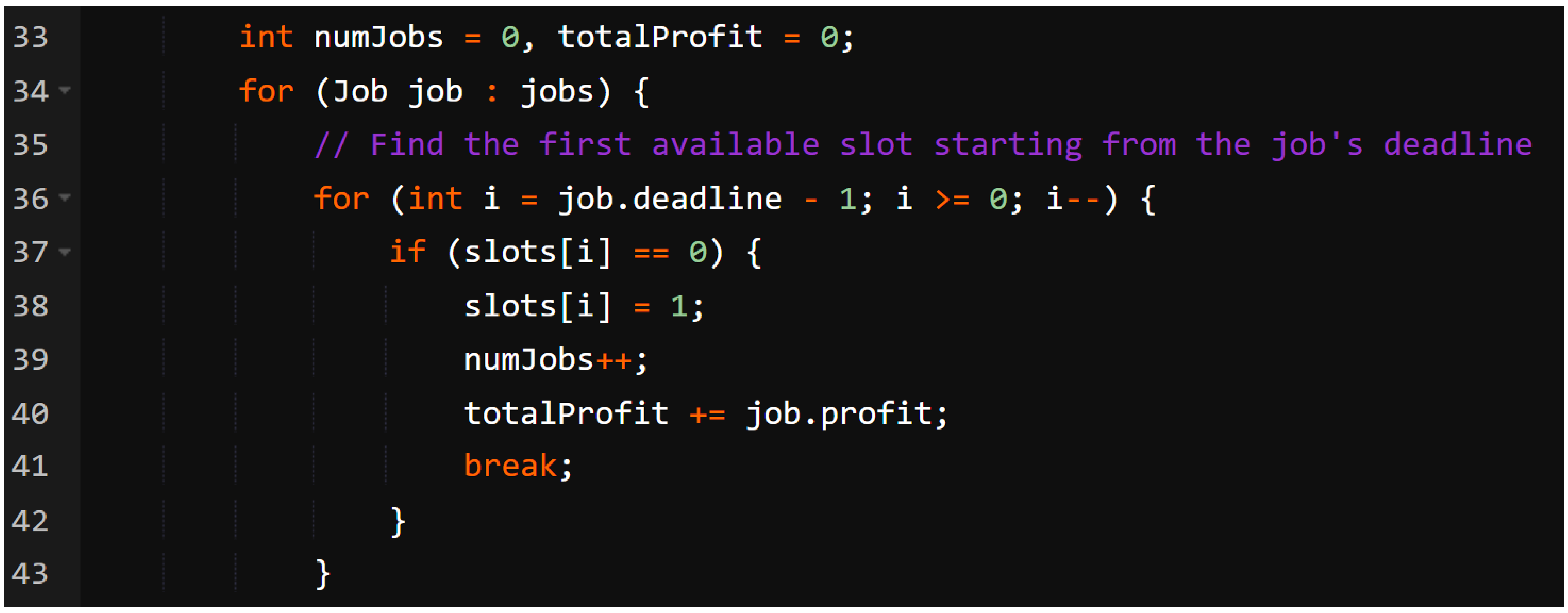
- Initializes an integer variable maxDeadline to 0.
- Iterates over the jobs list using a for-each loop to find the maximum deadline among all the jobs.
- Creates an integer array slots with a length equal to maxDeadline, and initializes all the elements of the array to 0.
- At the end of the loop, maxDeadline will contain the maximum deadline value among all the jobs in the jobs list.
- Math.max() method is used to update maxDeadline to the maximum value.
- Finally, the slots array is created with the appropriate length and initialized to all 0's.



```
        // Sort jobs in decreasing order of profit in lambda expression by
            subtracting adjacent elements a from b
        jobs.sort((a, b) -> b.profit - a.profit);
```

- Sorting the jobs list of Job objects in descending order of their profit, using a lambda expression as a comparator
- The comparator is defined using a lambda expression that takes two Job objects a and b, and returns the difference between their profit values.
- Compares the profit values of the two Job objects b and a, and returns a negative value if b.profit is less than a.profit, zero if they are equal, or a positive value if b.profit is greater than a.profit.

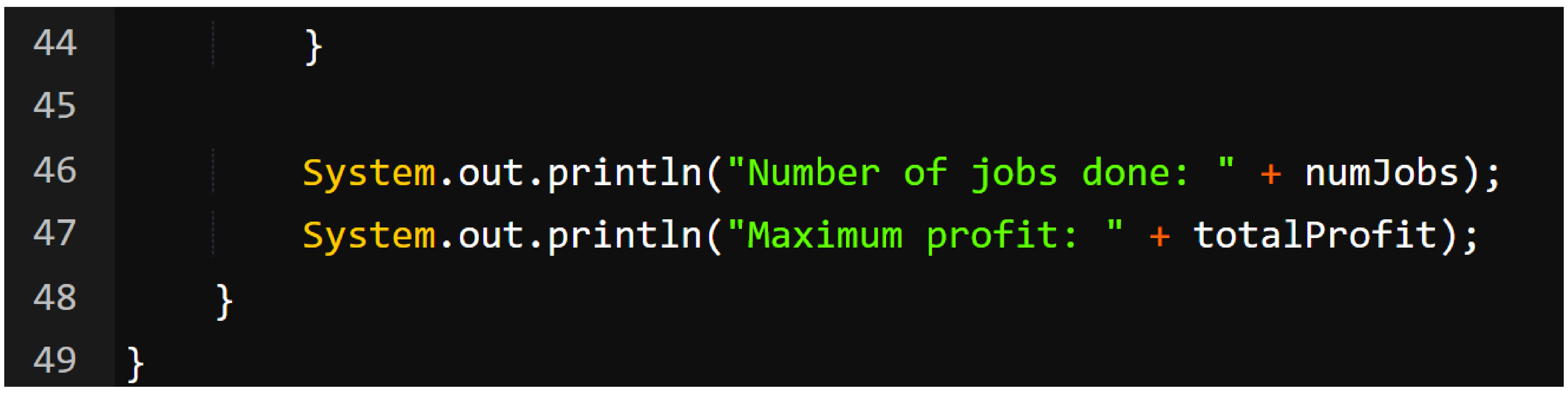
CODE & EXPLAINATION:



```java
        int numJobs = 0, totalProfit = 0;
        for (Job job : jobs) {
            // Find the first available slot starting from the job's deadline
            for (int i = job.deadline - 1; i >= 0; i--) {
                if (slots[i] == 0) {
                    slots[i] = 1;
                    numJobs++;
                    totalProfit += job.profit;
                    break;
                }
            }
```

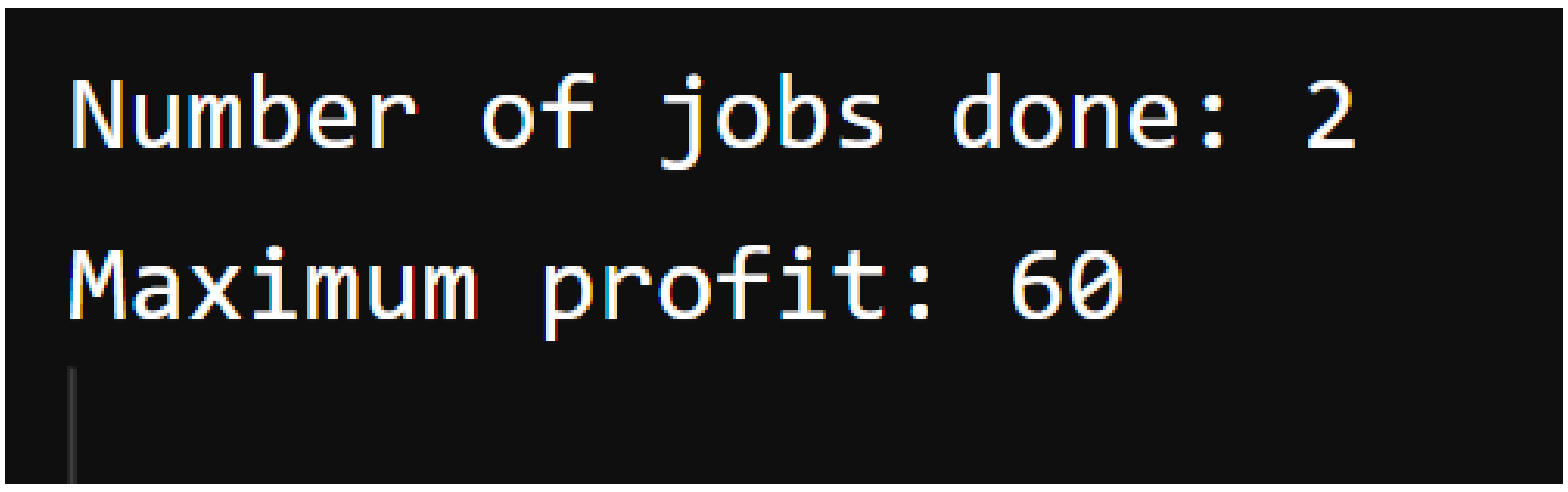
- using a nested for-loop to schedule the jobs and calculate the total profit that can be earned by scheduling the jobs subject to their deadlines.
- Two integer variables numJobs and totalProfit are initialized to 0.
- The outer for-each loop iterates over the jobs list, and for each Job object job in the list, it schedules the job in the first available time slot starting from the job's deadline.
- The inner for-loop starts from the index of the last time slot that is available for scheduling the job (i.e., job.deadline - 1) and iterates backwards to the first time slot (i.e., index 0). For each time slot i, the code checks if the time slot is available by checking if slots[i] == 0.
- If the time slot is available, the code schedules the job in that time slot by setting slots[i] = 1, and updates numJobs and totalProfit by incrementing them by 1 and by the profit of the job, respectively.
- Once a job has been scheduled, the inner loop is exited using the break statement, and the outer loop continues to the next job in the list.

CODE & EXPLAINATION:



```
            }

            System.out.println("Number of jobs done: " + numJobs);
            System.out.println("Maximum profit: " + totalProfit);
        }
    }
```

- Finally, the code prints out the number of jobs done and the maximum profit that can be earned.
- So the algorithm would schedule jobs 3 and 1, earning a total profit of 60.



```
Number of jobs done: 2
Maximum profit: 60
```

SECOND PROBLEM:

B. A.You are given a list of N jobs with their start time, end time, and the profit you can earn by doing that job. Your task is to find the maximum profit you can earn by picking up some (or all) jobs, ensuring no two jobs overlap. If you choose a job that ends at time X, you will be able to start another job that starts at time X.

Example 1

Input

Jobs = {{1, 6, 6}, {2, 5, 5}, {5, 7, 5}, {6, 8, 3}}

→ Steps(using dynamic programming):

(1) The main idea is to sort the jobs by their end times and then, for each job, compute the maximum profit that can be earned by scheduling that job.

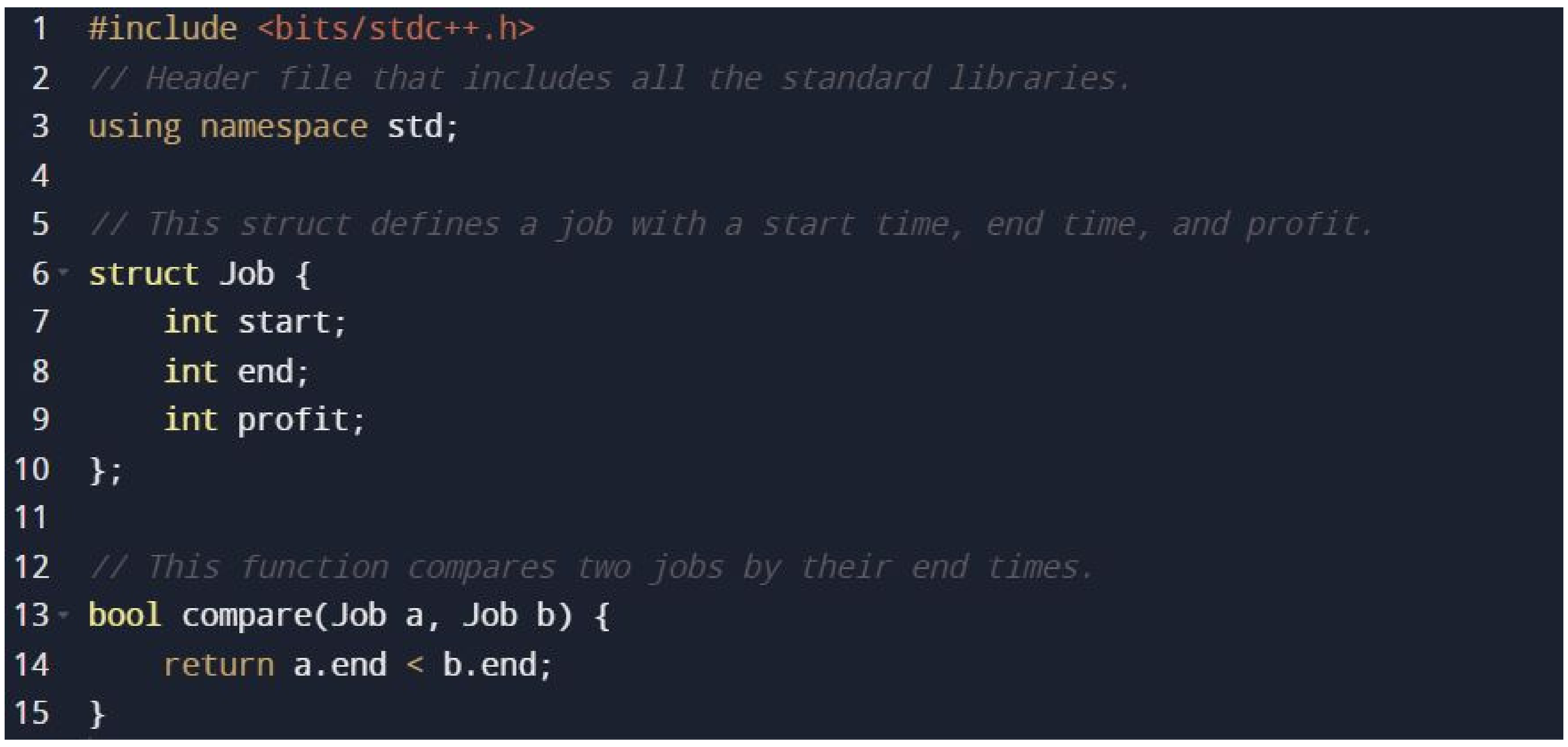
(2) creating a vector maxProfit that stores the maximum profit that can be earned by scheduling the first i jobs. The maximum profit for the first job is initialized to be its profit. It then iterates through all the jobs and for each job, it computes the maximum profit that can be earned by scheduling that job.

(3) To compute the maximum profit that can be earned by scheduling a job i, the algorithm looks at all the jobs that end before job i starts and selects the one that maximizes the profit.

(4) Finally, it updates the maximum profit for the current job to be the maximum of the current profit and the maximum profit earned by scheduling the previous jobs.

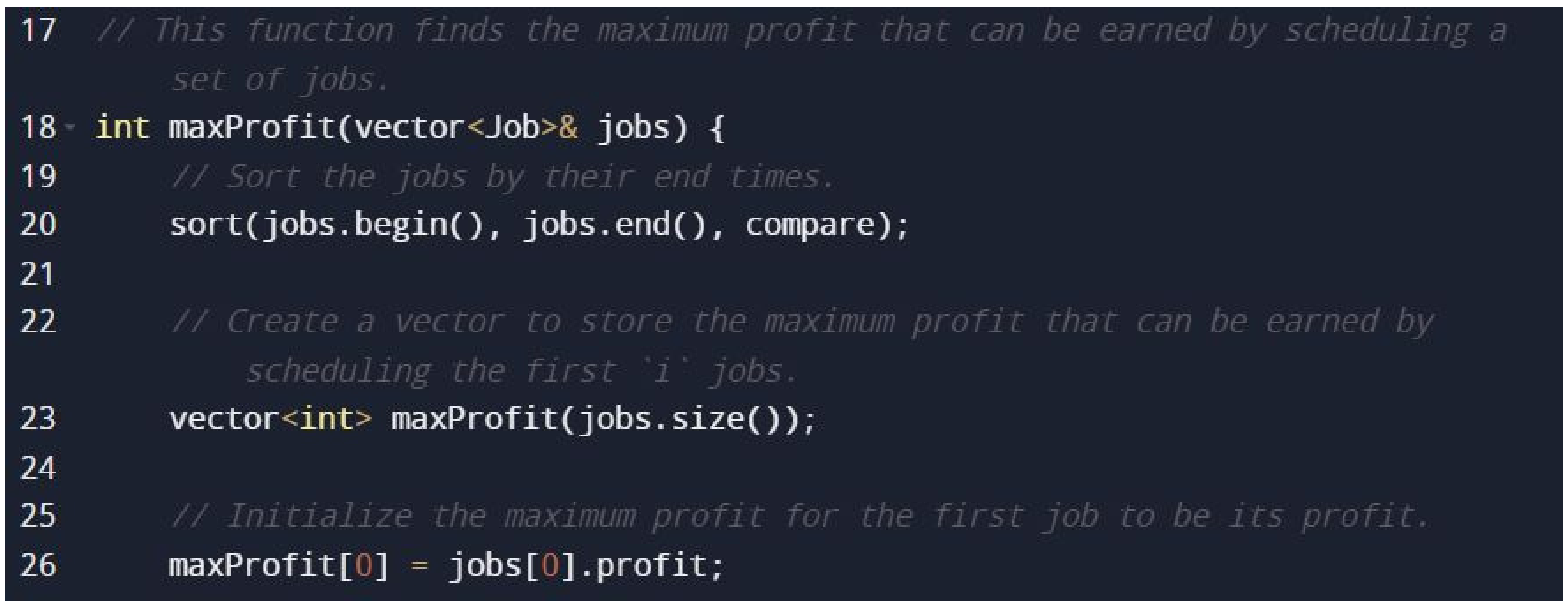
(5) using memoization to store the solutions to subproblems in the maxProfit vector.

CODE & EXPLAINATION:



```cpp
#include <bits/stdc++.h>
// Header file that includes all the standard libraries.
using namespace std;

// This struct defines a job with a start time, end time, and profit.
struct Job {
    int start;
    int end;
    int profit;
};

// This function compares two jobs by their end times.
bool compare(Job a, Job b) {
    return a.end < b.end;
}
```

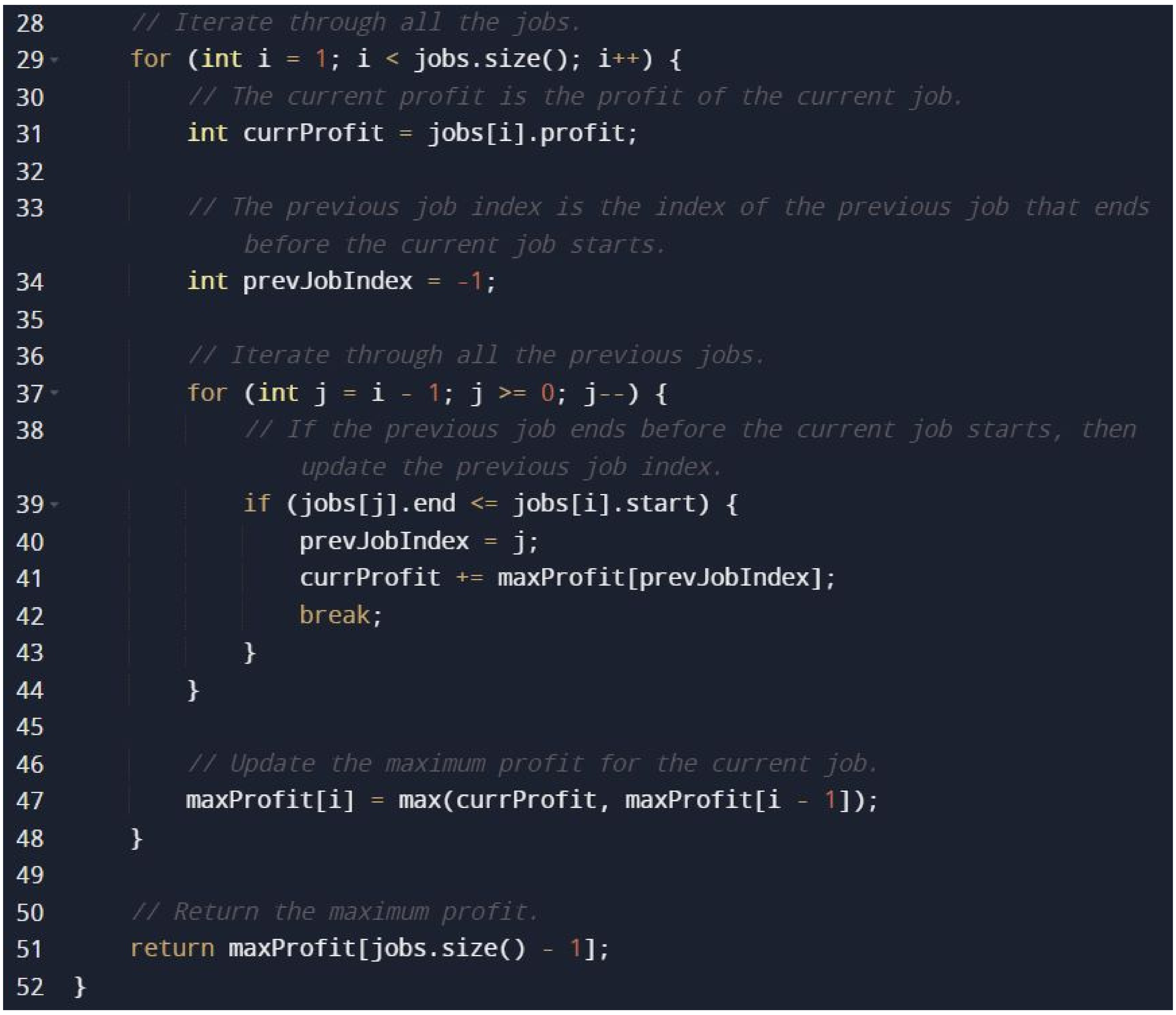
- First, we call all the standard header files in C++
- defines a struct called Job which has three fields: start, end, and profit. Each field is an integer.
- A function that takes two Job objects as input and returns true if the end time of the first job is less than the end time of the second job.(used as a comparator function to sort jobs ascendingly by its end time)



```cpp
// This function finds the maximum profit that can be earned by scheduling a
    set of jobs.
int maxProfit(vector<Job>& jobs) {
    // Sort the jobs by their end times.
    sort(jobs.begin(), jobs.end(), compare);

    // Create a vector to store the maximum profit that can be earned by
        scheduling the first `i` jobs.
    vector<int> maxProfit(jobs.size());

    // Initialize the maximum profit for the first job to be its profit.
    maxProfit[0] = jobs[0].profit;
```

- A function that calculates the maximum profit that can be earned from a set of jobs. It takes a vector of Job objects as input by reference so that the original vector can be modified.
- Sorting the jobs vector by end time using the compare function defined earlier.
- Gets the size of the jobs vector.
- Creating a vector called maxProfit which stores the maximum profit that can be earned up to a certain job index.

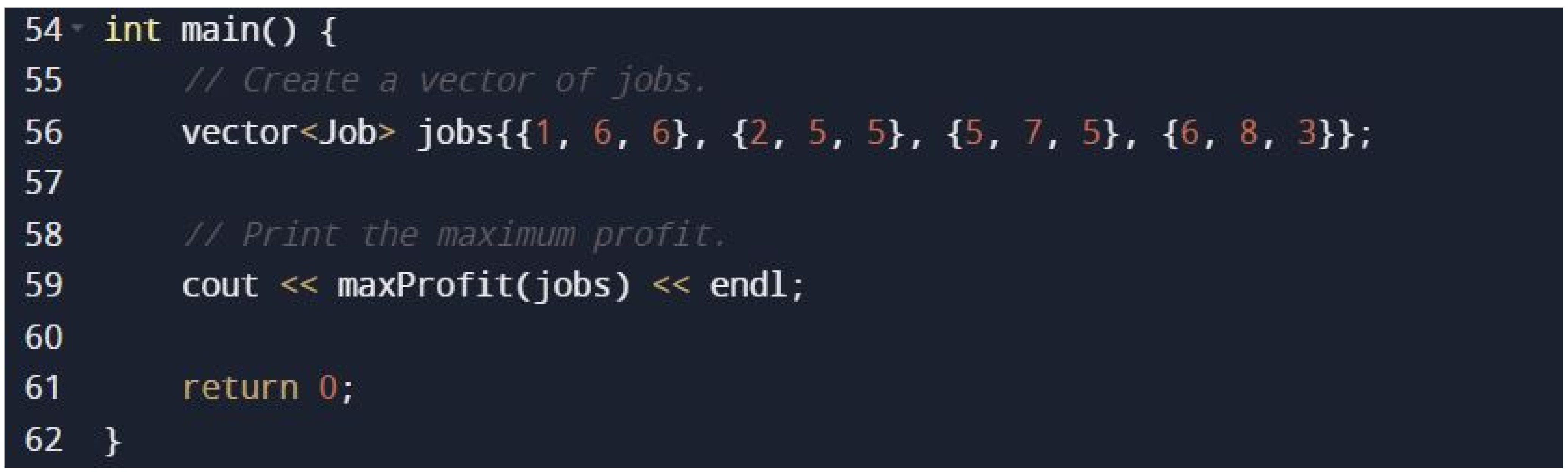
CODE & EXPLAINATION:



```
    // Iterate through all the jobs.
    for (int i = 1; i < jobs.size(); i++) {
        // The current profit is the profit of the current job.
        int currProfit = jobs[i].profit;

        // The previous job index is the index of the previous job that ends before the current job starts.
        int prevJobIndex = -1;

        // Iterate through all the previous jobs.
        for (int j = i - 1; j >= 0; j--) {
            // If the previous job ends before the current job starts, then update the previous job index.
            if (jobs[j].end <= jobs[i].start) {
                prevJobIndex = j;
                currProfit += maxProfit[prevJobIndex];
                break;
            }
        }

        // Update the maximum profit for the current job.
        maxProfit[i] = max(currProfit, maxProfit[i - 1]);
    }

    // Return the maximum profit.
    return maxProfit[jobs.size() - 1];
}
```

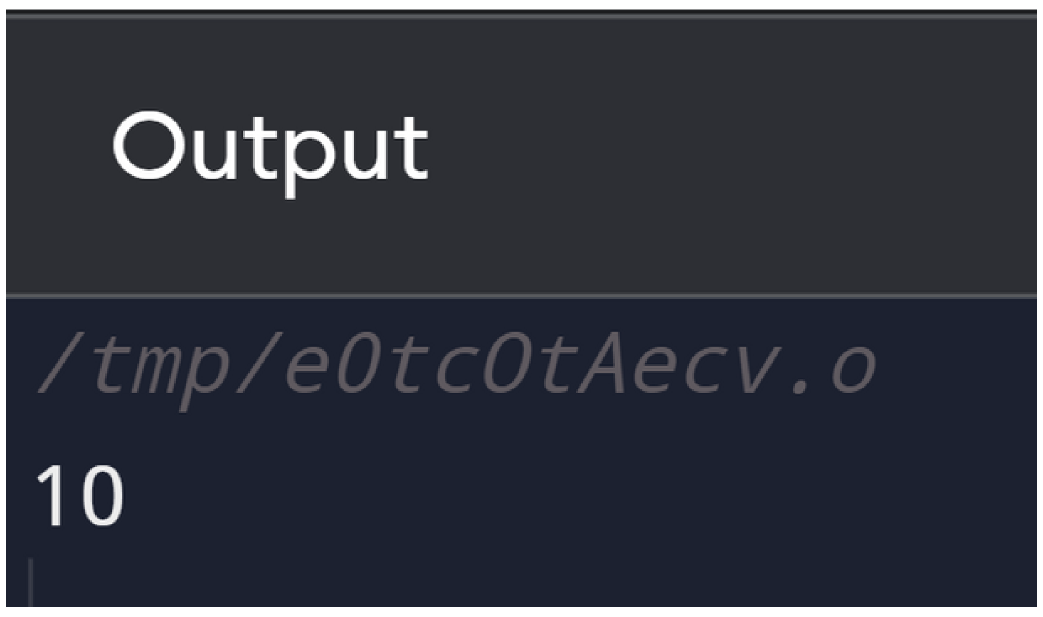
- This loop iterates through all the jobs and calculates the maximum profit that can be earned up to each job index. Starting with the second job, it checks all the previous jobs to see if they end before the current job starts. If there is a previous job that ends before the current job starts, the current job's profit is added to the maximum profit that can be earned up to the previous job's index. The maximum of this value and the previous maximum profit is stored in the maxProfit vector for the current job index

CODE & EXPLAINATION:



```
int main() {
    // Create a vector of jobs.
    vector<Job> jobs{{1, 6, 6}, {2, 5, 5}, {5, 7, 5}, {6, 8, 3}};

    // Print the maximum profit.
    cout << maxProfit(jobs) << endl;

    return 0;
}
```

- This is the main function that creates a vector of Job objects and calls the maxProfit function to calculate the maximum profit that can be earned from these jobs.



```
Output
/tmp/e0tc0tAecv.o
10
```

- This is the maximum profit that can be earned by scheduling the set of jobs provided in the main function.