BUBT | BANGLADESH UNIVERSITY OF BUSINESS AND TECHNOLOGY
Committed to Academic Excellence

Lab-05 & 06

Course Title	: Algorithms Lab
Course Code	: CSE 242

Submitted By :

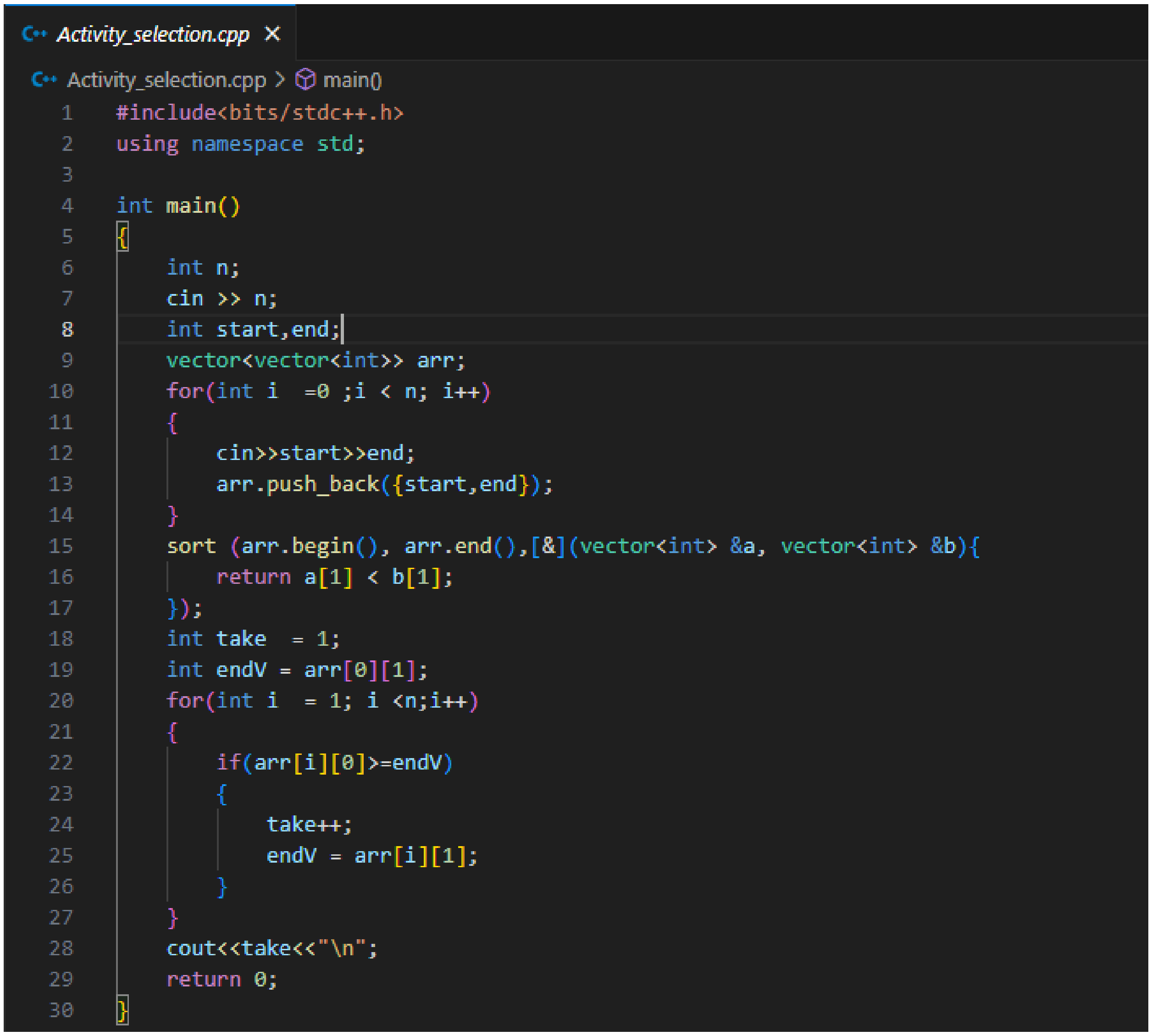
Name	: Md. Mursalin Hasan Nirob
ID No	: 21225103423
Intake	: 49
Section	: 10
Program	: B.Sc. Engg. in CSE

Submitted To :

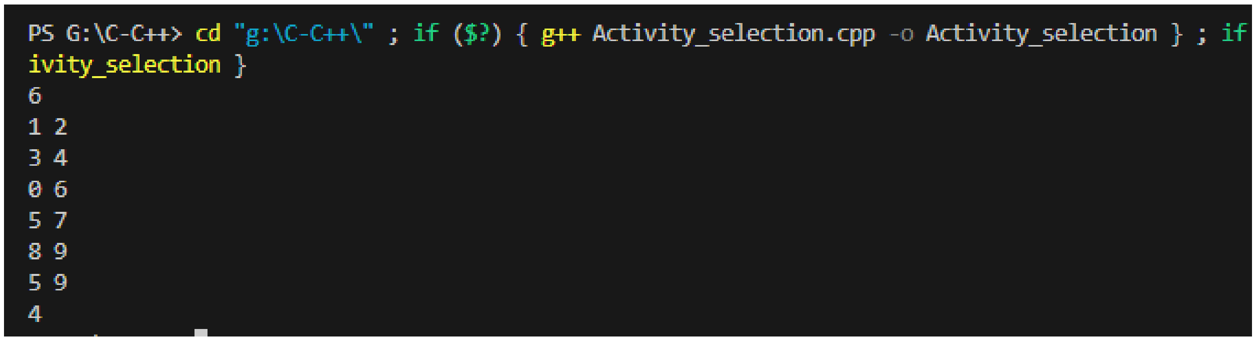
Name : Faria Binte Kader
Lecturer
Department of : CSE
Bangladesh University of Business & Technology

Date of Submission : 23/09/2023

1. You are given n activities with their start and finish times. Your program should output the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single move at a time.
Input: start[] = {1, 3, 0, 5, 8, 5}, finish[] = {2, 4, 6, 7, 9, 9};


Activity_selection.cpp ×
Activity_selection.cpp > main()

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

int main()
{
    int n;
    cin >> n;
    int start,end;
    vector<vector<int>> arr;
    for(int i  =0 ;i < n; i++)
    {
        cin>>start>>end;
        arr.push_back({start,end});
    }
    sort (arr.begin(), arr.end(),[&](vector<int> &a, vector<int> &b){
        return a[1] < b[1];
    });
    int take  = 1;
    int endV = arr[0][1];
    for(int i  = 1; i <n;i++)
    {
        if(arr[i][0]>=endV)
        {
            take++;
            endV = arr[i][1];
        }
    }
    cout<<take<<"\n";
    return 0;
}
```

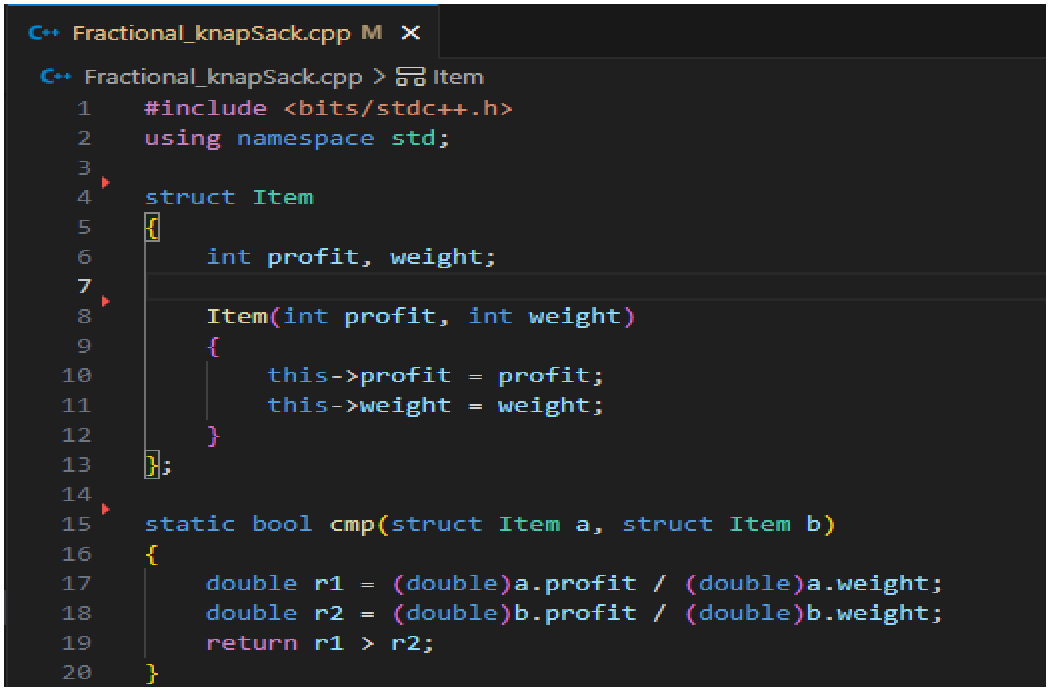


```
PS G:\C-C++> cd "g:\C-C++\" ; if ($?) { g++ Activity_selection.cpp -o Activity_selection } ; if
ivity_selection }
6
1 2
3 4
0 6
5 7
8 9
5 9
4
```

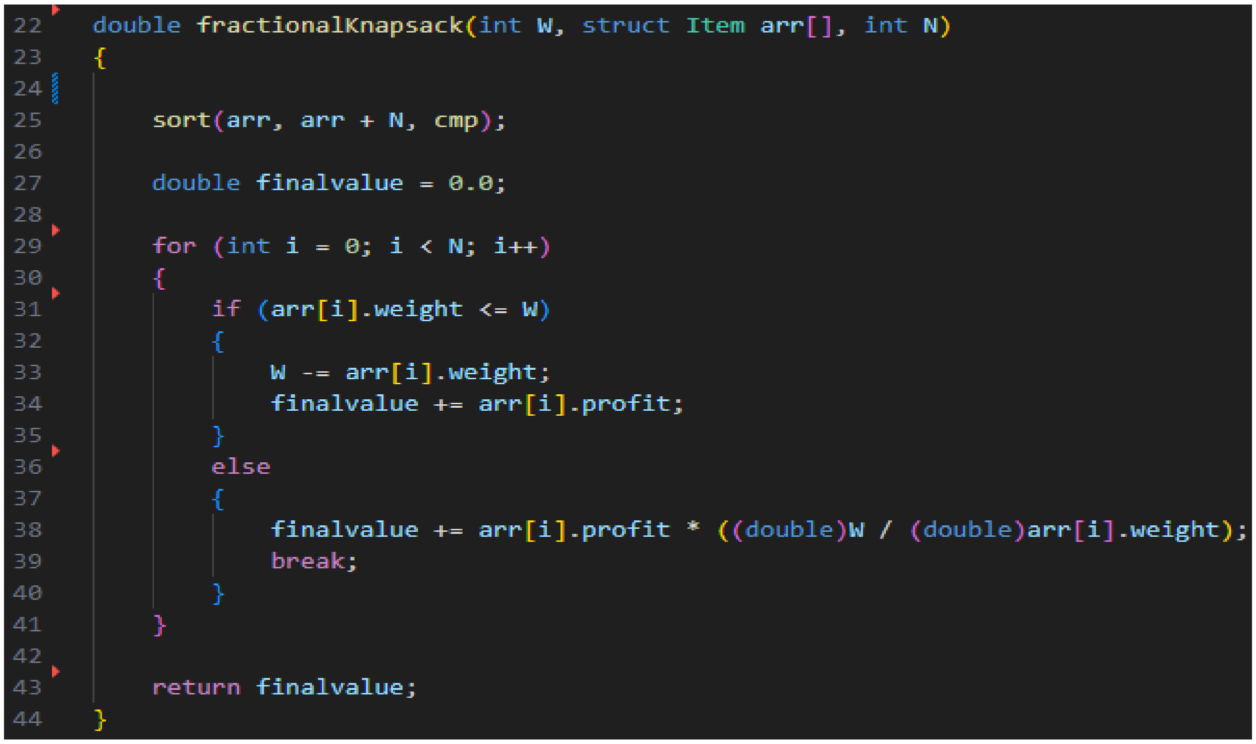
Explanation: I am finding out how many activities I can take by this code. I am count the first task always then check if the ending time is greater than the starting time of another task or not. If it's not Then I am counting on that also. By This logic, I find the number of tasks that can be taken.

2. Given the weights and profits of N items, in the form of {profit, weight} put these items in a knapsack of capacity W to get the maximum total profit in the knapsack. In Fractional Knapsack, we can break items to maximize the total value of the knapsack.

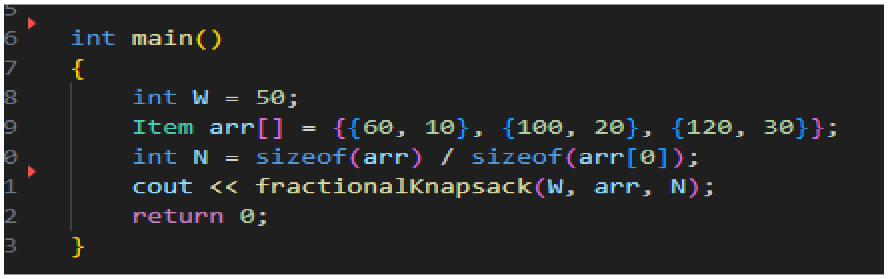
Input: arr[] = {{60, 10}, {100, 20}, {120, 30}}, W = 50



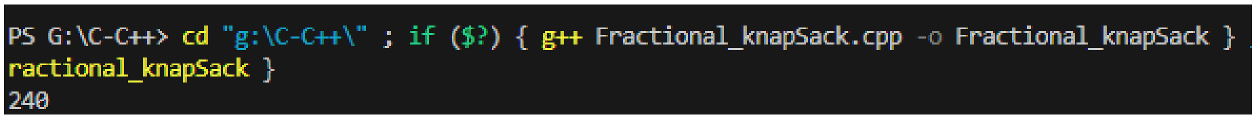
```cpp
#include <bits/stdc++.h>
using namespace std;

struct Item
{
    int profit, weight;

    Item(int profit, int weight)
    {
        this->profit = profit;
        this->weight = weight;
    }
};

static bool cmp(struct Item a, struct Item b)
{
    double r1 = (double)a.profit / (double)a.weight;
    double r2 = (double)b.profit / (double)b.weight;
    return r1 > r2;
}
```



```cpp
double fractionalKnapsack(int W, struct Item arr[], int N)
{

    sort(arr, arr + N, cmp);

    double finalvalue = 0.0;

    for (int i = 0; i < N; i++)
    {
        if (arr[i].weight <= W)
        {
            W -= arr[i].weight;
            finalvalue += arr[i].profit;
        }
        else
        {
            finalvalue += arr[i].profit * ((double)W / (double)arr[i].weight);
            break;
        }
    }

    return finalvalue;
}
```



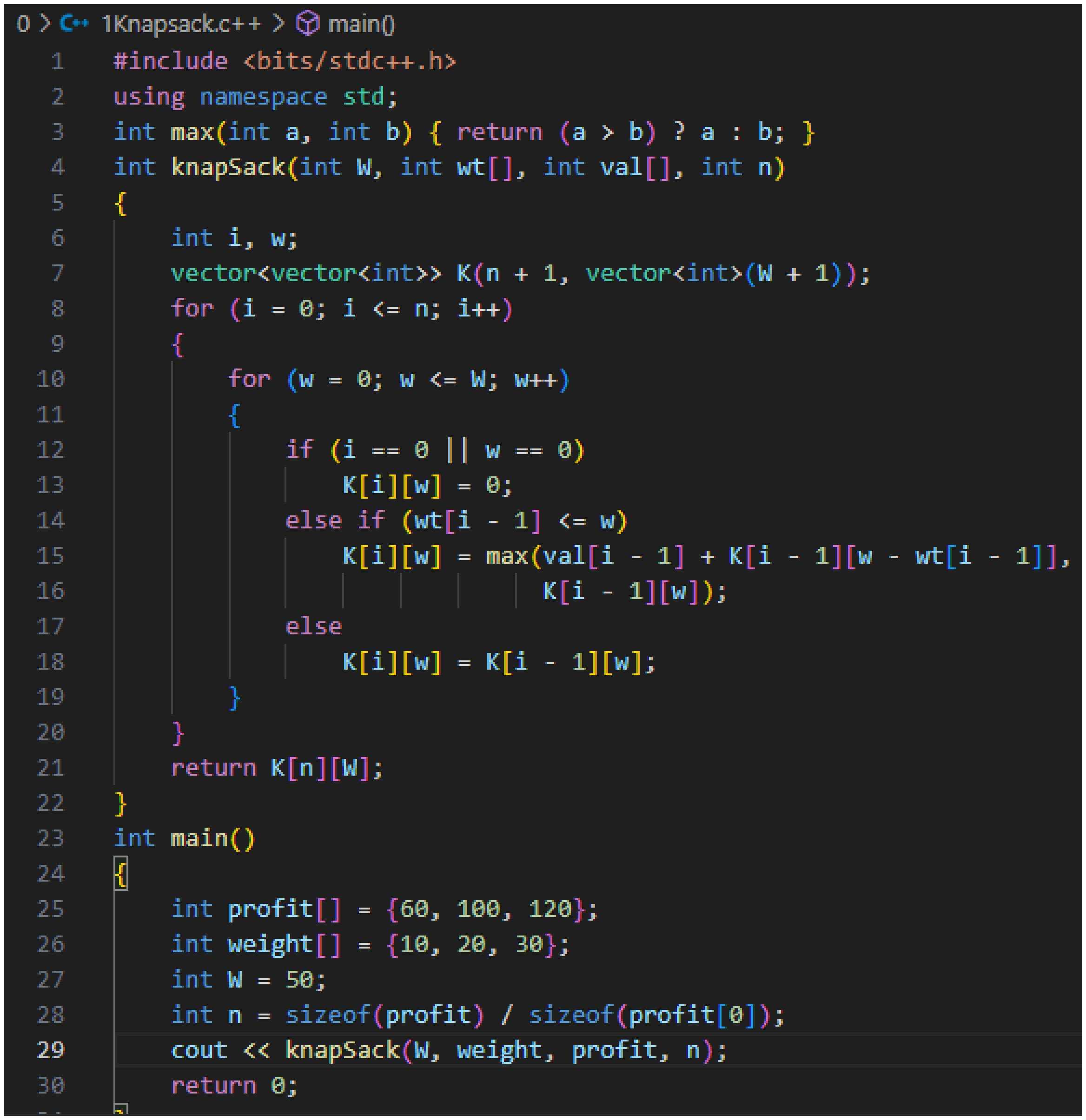
```cpp
int main()
{
    int W = 50;
    Item arr[] = {{60, 10}, {100, 20}, {120, 30}};
    int N = sizeof(arr) / sizeof(arr[0]);
    cout << fractionalKnapsack(W, arr, N);
    return 0;
}
```



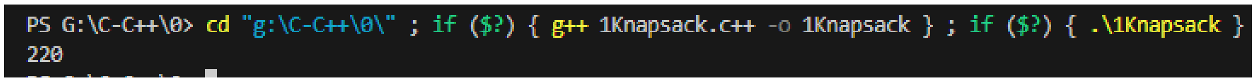
```
PS G:\C-C++> cd "g:\C-C++\" ; if ($?) { g++ Fractional_knapSack.cpp -o Fractional_knapSack }
ractional_knapSack }
240
```

Explanation: I am attempting to determine the maximum profit achievable by selecting items based on their maximum weight capacity. I begin by selecting items with the highest value-to-weight ratio until I reach the maximum weight limit. Then, I reduce the available weight capacity accordingly. Finally, I calculate the profit using the fraction of the item's weight taken and return the overall profit. This process is commonly referred to as the Fractional Knapsack problem.

3. Use the same greedy approach in the previous problem but this time the knapsack is not fractional meaning we cannot break the items.
Input: arr[] = {{60, 10}, {100, 20}, {120, 30}}, W = 50



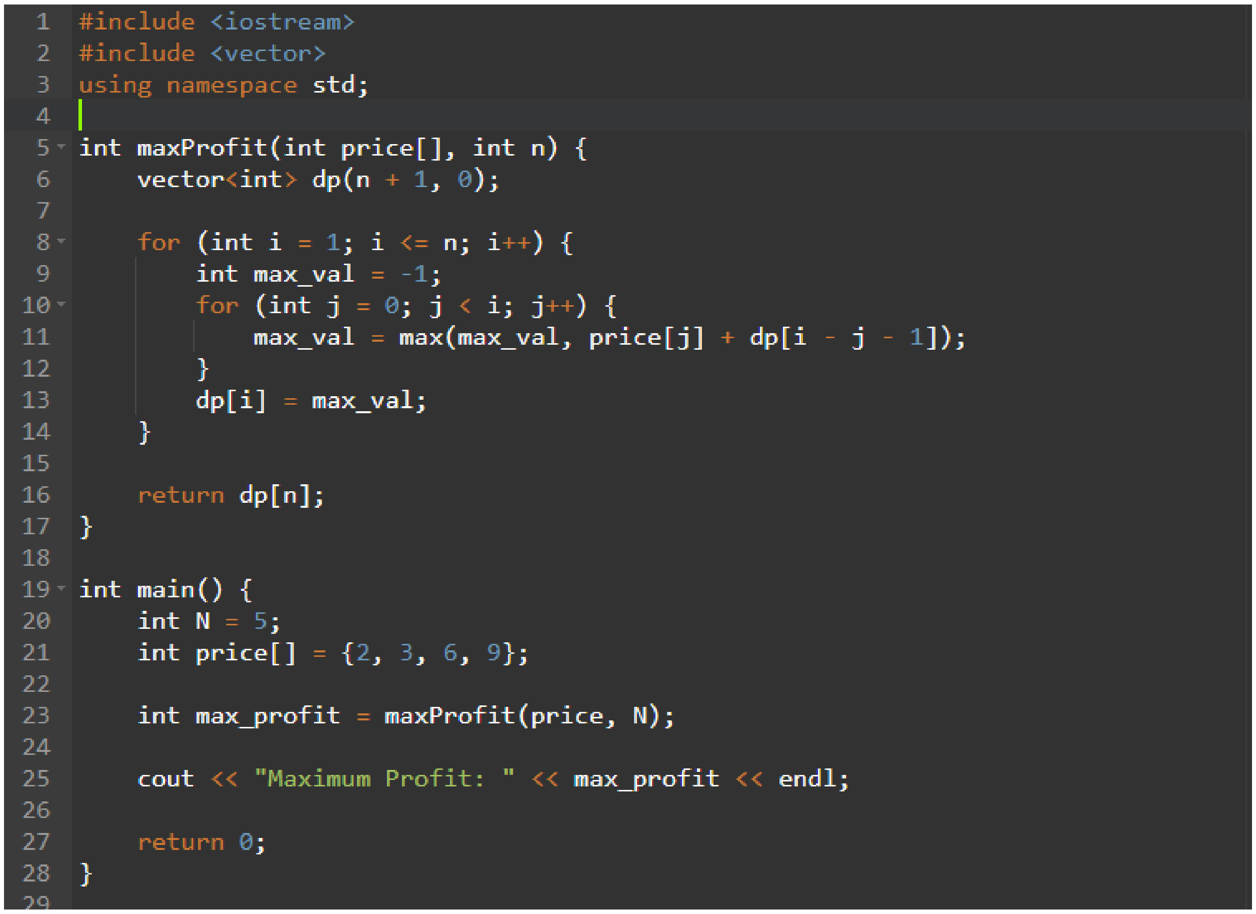
```cpp
#include <bits/stdc++.h>
using namespace std;
int max(int a, int b) { return (a > b) ? a : b; }
int knapSack(int W, int wt[], int val[], int n)
{
    int i, w;
    vector<vector<int>> K(n + 1, vector<int>(W + 1));
    for (i = 0; i <= n; i++)
    {
        for (w = 0; w <= W; w++)
        {
            if (i == 0 || w == 0)
                K[i][w] = 0;
            else if (wt[i - 1] <= w)
                K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]],
                                K[i - 1][w]);
            else
                K[i][w] = K[i - 1][w];
        }
    }
    return K[n][W];
}
int main()
{
    int profit[] = {60, 100, 120};
    int weight[] = {10, 20, 30};
    int W = 50;
    int n = sizeof(profit) / sizeof(profit[0]);
    cout << knapSack(W, weight, profit, n);
    return 0;
```



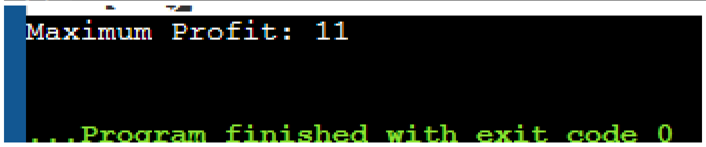
```
PS G:\C-C++\0> cd "g:\C-C++\0\" ; if ($?) { g++ 1Knapsack.c++ -o 1Knapsack } ; if ($?) { .\1Knapsack }
220
```

Explanation: I am striving to determine the maximum profit attainable while adhering to a strict binary choice: either an item is taken in its entirety or not at all. In the 0/1 Knapsack problem, I select items based on their value and weight, ensuring that the total weight does not exceed a specified limit. I aim to maximize the profit by choosing the right combination of items to include or exclude within this binary constraint.

4.You are given a rod of length N and a list of prices corresponding to different lengths of the rod that can be obtained by cutting it. Your goal is to determine the optimal way to cut the rod to maximize the total price.
Input: N = 5, price[] = { 2, 3, 6, 9 }



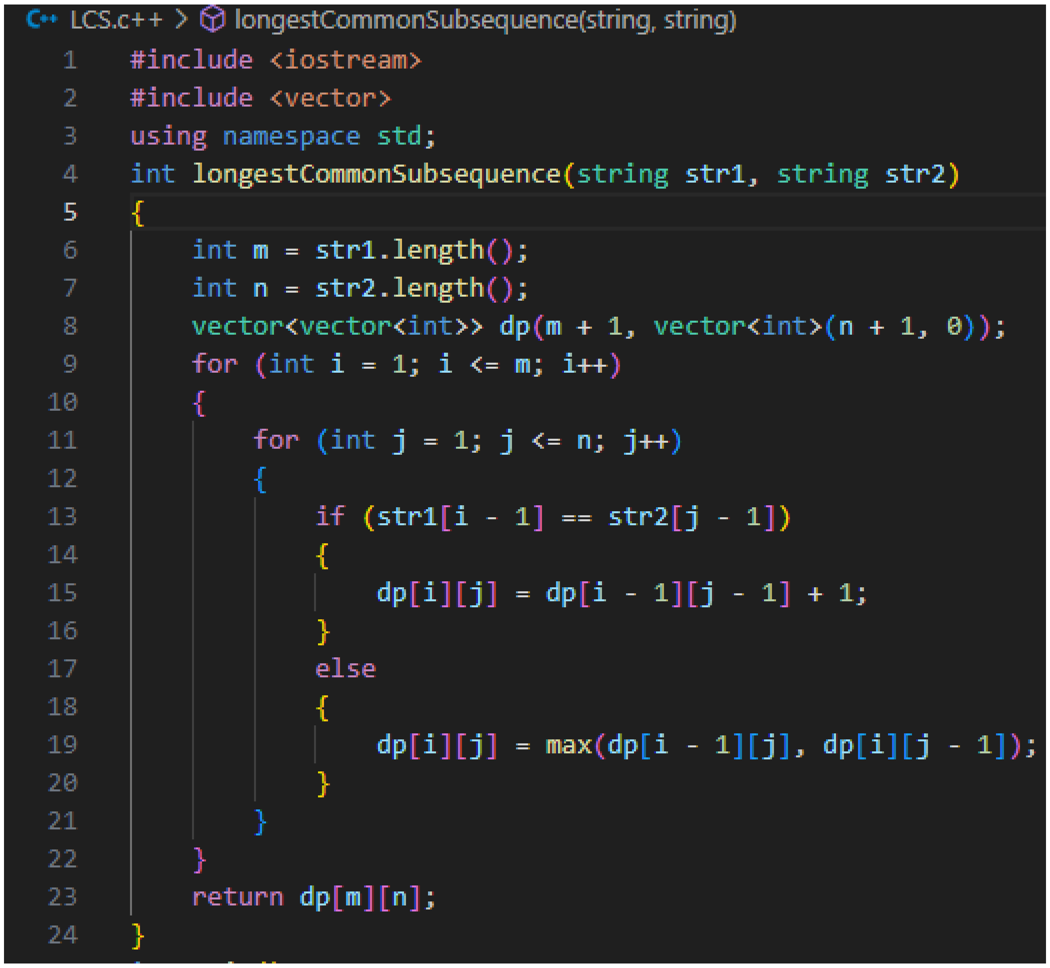
```cpp
#include <iostream>
#include <vector>
using namespace std;

int maxProfit(int price[], int n) {
    vector<int> dp(n + 1, 0);

    for (int i = 1; i <= n; i++) {
        int max_val = -1;
        for (int j = 0; j < i; j++) {
            max_val = max(max_val, price[j] + dp[i - j - 1]);
        }
        dp[i] = max_val;
    }

    return dp[n];
}

int main() {
    int N = 5;
    int price[] = {2, 3, 6, 9};

    int max_profit = maxProfit(price, N);

    cout << "Maximum Profit: " << max_profit << endl;

    return 0;
}
```



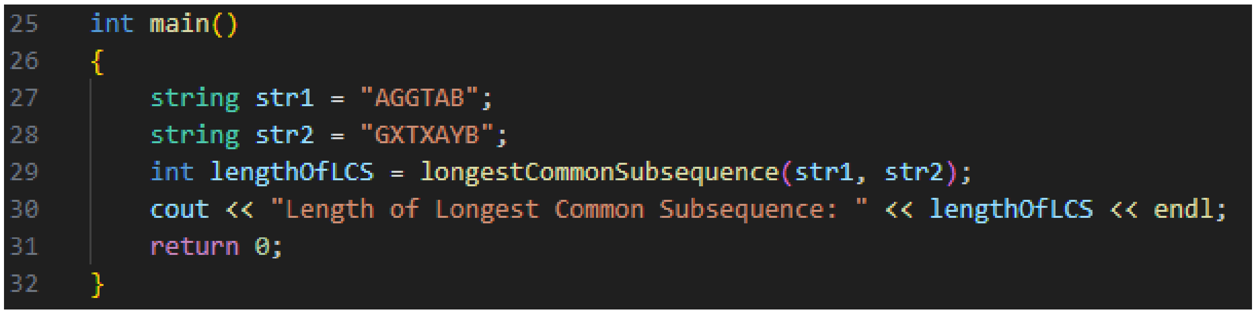
```
Maximum Profit: 11


...Program finished with exit code 0
```

Explanation: We define the function maxProfit to find the maximum profit, as before. We use a nested loop to iterate through all possible rod lengths and update dp in a compact way. In the main function, we provide the input values and directly print the result without storing it in a separate variable.

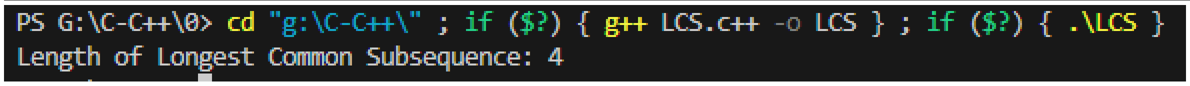
5.You are given two sequences of characters, “str1” and “str2”. Your task is to find the length of the longest common subsequence (LCS) between these two sequences.
Input: str1[] = AGGTAB str2[]=GXTXAYB



```cpp
#include <iostream>
#include <vector>
using namespace std;
int longestCommonSubsequence(string str1, string str2)
{
    int m = str1.length();
    int n = str2.length();
    vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));
    for (int i = 1; i <= m; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            if (str1[i - 1] == str2[j - 1])
            {
                dp[i][j] = dp[i - 1][j - 1] + 1;
            }
            else
            {
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
            }
        }
    }
    return dp[m][n];
}
```



```cpp
int main()
{
    string str1 = "AGGTAB";
    string str2 = "GXTXAYB";
    int lengthOfLCS = longestCommonSubsequence(str1, str2);
    cout << "Length of Longest Common Subsequence: " << lengthOfLCS << endl;
    return 0;
}
```



```
PS G:\C-C++\0> cd "g:\C-C++\" ; if ($?) { g++ LCS.c++ -o LCS } ; if ($?) { .\LCS }
Length of Longest Common Subsequence: 4
```

Explanation: This code uses dynamic programming to find the length of the Longest Common Subsequence (LCS) between two input strings, "str1" and "str2." It iterates through both strings, incrementing the LCS length when characters match, and calculates it by considering the maximum when they don't. The result, "Length of LCS: 4," for the input "AGGTAB" and "GXTXAYB," indicates the LCS is "GTAB" with a length of 4.