Assignment 09| Advance Algorithms CE-092

Assignment submission for Advance Algorithms subject week 09.

nevilparmar24@gmail.com

Task 1:

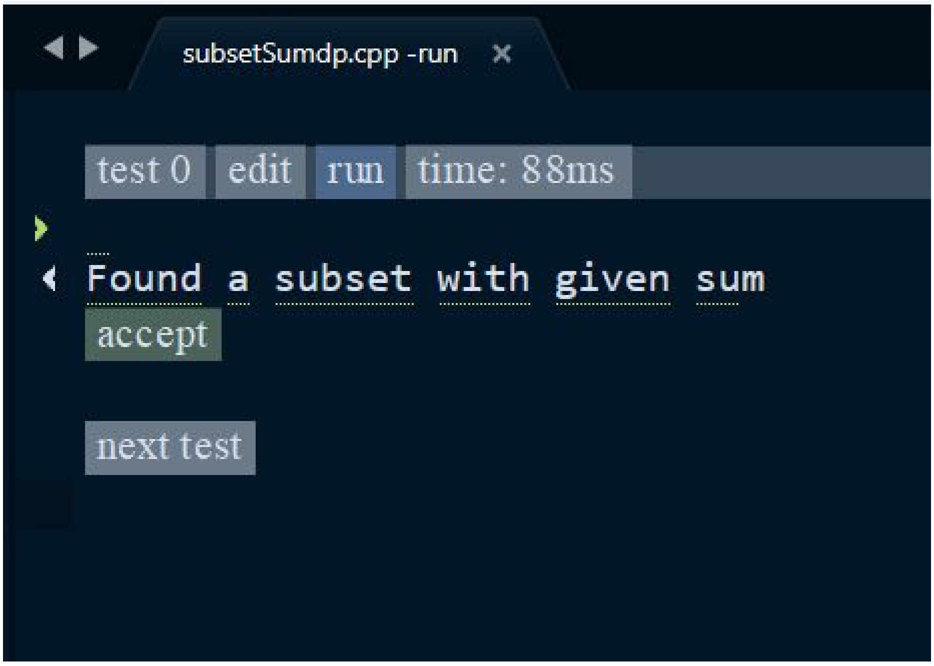
Subset sum using dynamic knapsack problem.

Code:

```
/*
* @Author: nevil
* @Date:   2020-10-31 14:11:40
* @Last Modified by:   nevil
* @Last Modified time: 2020-10-31 14:12:36
*/

// A Dynamic Programming solution
// for subset sum problem
#include <bits/stdc++.h>
using namespace std;


// Returns true if there is a subset of set[]
// with sun equal to given sum
bool isSubsetSum(int set[], int n, int sum)
{
    // The value of subset[i][j] will be true if
    // there is a subset of set[0..j-1] with sum
    // equal to i
```

```c
    bool subset[n + 1][sum + 1];

    // If sum is 0, then answer is true
    for (int i = 0; i <= n; i++)
        subset[i][0] = true;

    // If sum is not 0 and set is empty,
    // then answer is false
    for (int i = 1; i <= sum; i++)
        subset[0][i] = false;

    // Fill the subset table in botton up manner
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= sum; j++) {
            if (j < set[i - 1])
                subset[i][j] = subset[i - 1][j];
            if (j >= set[i - 1])
                subset[i][j] = subset[i - 1][j]
                            || subset[i - 1][j - set[i - 1]];
        }
    }

    return subset[n][sum];
}

// Driver program to test above function
int main()
{
    int set[] = { 3, 34, 4, 12, 5, 2 };
    int sum = 9;
    int n = sizeof(set) / sizeof(set[0]);
    if (isSubsetSum(set, n, sum) == true)
        printf("Found a subset with given sum");
    else
        printf("No subset with given sum");
    return 0;
}
```

Output:



```
Found a subset with given sum
```

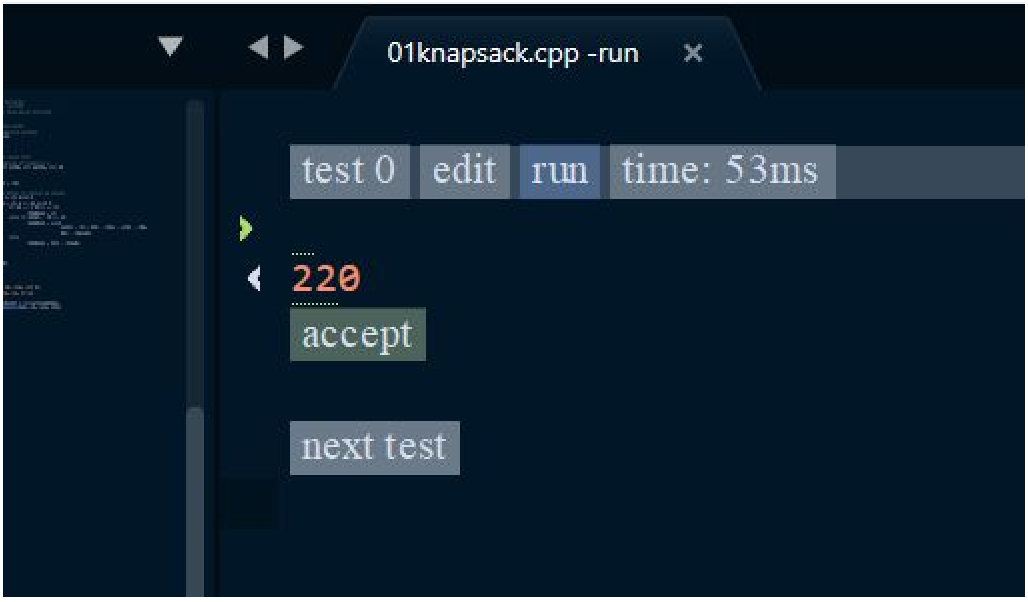
Task 2:

01 knapsack problem using dynamic programming.

Code:

```
/*
* @Author: nevil11
* @Date:    2020-10-25 17:30:26
* @Last Modified by:    nevil11
* @Last Modified time: 2020-10-25 17:31:42
*/

// A Dynamic Programming based
// solution for 0-1 Knapsack problem
#include <bits/stdc++.h>
using namespace std;


// Returns the maximum value that
// can be put in a knapsack of capacity W
int knapSack(int W, int wt[], int val[], int n)
{
    int i, w;
    int K[n + 1][W + 1];

    // Build table K[][] in bottom up manner
    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 val[] = { 60, 100, 120 };
    int wt[] = { 10, 20, 30 };
    int W = 50;
    int n = sizeof(val) / sizeof(val[0]);
    printf("%d", knapSack(W, wt, val, n));
    return 0;
}
```

Output:



```
220
```