„Programming”
Big Project

Made by: Lu Yingjie
Neptun code: AF35AN
E-mail: yingjielu2002@163.com

Course code: ???
Teacher’s name: vincze Dorottya

2022. December 10.

Content

User documentation

Task

A ticket inspector of the Budapest-Székesfehérvár train logged the number of passengers getting on and off at each of the stops. (There are no people getting off at Budapest, and no people getting on at Székesfehérvár. No people get on after getting off.) Write a program that tells whether it is economical to operate the train if a passenger pays N HUF for each traveled stop, and it costs M HUF to operate the train between 2 stops.

Runtime environment

Programs running on .NET system. An operating system capable of running cs files (eg. Visual studio).

Usage

Starting the program

The program can be found in the archive file, Program.cs. You can launch the program by double-clicking on the Peogram.cs file.

Program input

The program reads the input data from the keyboard in the following order:

#	Data	Explanation
1.	*length*	The count of the stops(1≤S≤1000).
2.	pays	The number of pay per stop (0<pays≤100).
3.	*coststops*	The spend on operation (1≤coststops≤100000).
4.	*On[i]*	The count of people getting on (0≤ON≤800)
……	……	
5.	*Off[i]*	The count of people getting off (0≤off ≤800)
……	……	

Program output

The program writes out should contain 1 if it is economic to operate the train, and 0 if not

Sample input and output

```
please input the count of stops:
6
please input the number of pay per stop and operation cost between 2 stops
100 1000
please input the count of people getting on and getting off:
0 15
10 30
0 32
48 0
20 27
26 0
1
```

Possible errors

The input should be given based on the sample. If count of stops is not an integer, or is not in the range 1..1000, this will cause a problem.
If passenger has to pay per stop is not a number, or is not in the range 0..100, this will also cause a problem.
If operation cost between 2 stops is not a number, or is not in the range 1..100000, this will also cause a problem.
If the number of people getting on or getting off at a stop is not a number, or is not in the range 0..800, this will also cause an error
If an error occurs, the program will display an error message or ask for a repeat entry.

Sample of running in the case of invalid data:

```
please input the count of stops:
10000
please input the count of stops:
7
please input the number of pay per stop and operation cost between 2 stops
1000 -1
please input the number of pay per stop and operation cost between 2 stops
100 1000
please input the count of people getting on and getting off:
900 900
```

Developer documentation

Task

A ticket inspector of the Budapest-Székesfehérvár train logged the number of passengers getting on and off at each of the stops. (There are no people getting off at Budapest, and no people getting on at Székesfehérvár. No people get on after getting off.) Write a program that tells whether it is economical to operate the train if a passenger pays N HUF for each traveled stop, and it costs M HUF to operate the train between 2 stops.

Specification

Input: sum∈N, length∈N, on[1...length] $\in N^{length}$,off[1...length] $\in N^{length}$
pays∈N, coststops∈N.

Output: count∈N
Precondition: 1≤length≤1000;0<pays≤100;1≤coststops≤100000
∀[i](1≤length≤1000):0≤on[i] ≤800,0≤off[i] ≤800
Postcondition:

∀[i](1≤length≤1000):0≤on[i] ≤800: sum = on[i] * pays ;

Count:=$\sum_{i=1}^{length} 1$
$sum > (coststops * length)$

Developer environment

An operating system capable of running exe files (eg. Visual studio).

Source code

All the sources can be found in the `B3` folder (after extraction). The folder structure used for development:

File	Explanation
`Program.cs`	Executable code
`B3test.txt`	input test file$_1$
`B3test.txt`	Output test file
`B3.docx`	documentation (this file)

Solution

Program parameters

Contants

```
length      : Integer(Console read)   [the count of stops]
pays  : Integer(Console read)   [the number of passenger has to pay per stop]
coststops   : Integer(Console read)   [the number of operation cost]
```

Types

```
ON              = Array(1..length:Integer)
Off             = Array(1..length:Integer)
```

Variables

```
length          : Integer
pays            : Integer
coststops       : Integer
ON[i]           : ON
Off[i]          : off
```

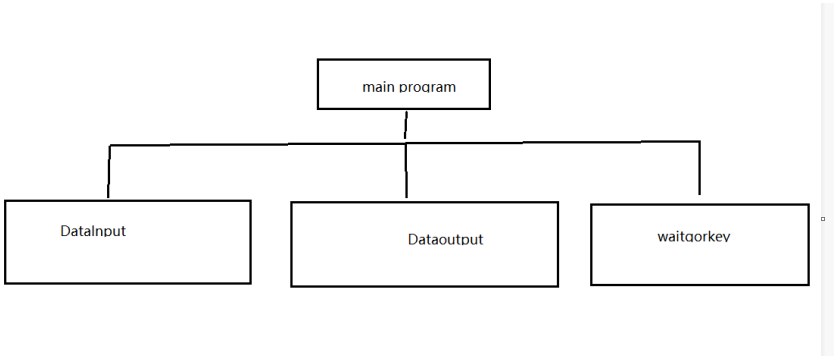
The structure of the program

The modules used by the program, and their locations:

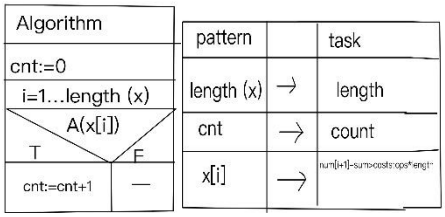
`Program.cs` – the program, in the source folder

`Main funcation` –Entry to the program

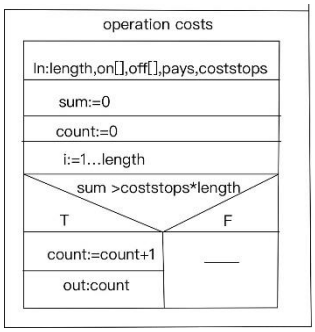
Structure of functions



The algorithm of the program



Main:



The code

The content of the `program.ca` file:

```
using System;
namespace B3
{
    internal class Program
    {
        static void Main(string[] args)
        {
            int count = 0;
            int length = 0;
            int sum = 0;
            int pays = 0;
            int coststops = 0;
            do
            {
                Console.WriteLine("please input the count of stops:");
                length = int.Parse(Console.ReadLine());
            } while (length<=1||length>=1000);
            do {
                Console.WriteLine("please input the number of pay per stop and operation cost between
2 stops");
                string all = Console.ReadLine();
                pays = Convert.ToInt32(all.Split(' ')[0]);
                coststops = Convert.ToInt32(all.Split(' ')[1]);
            } while (pays>100||pays<=0&&coststops>100000||coststops<1);
            int[] on=new int[length];
            int[] off=new int[length];
            Console.WriteLine("please input the count of people getting on and getting off: ");
            for (int i = 0; i < length; )
            {
                string input = Console.ReadLine();
                on[i] = Convert.ToInt32(input.Split(' ')[0]);
                off[i] = Convert.ToInt32(input.Split(' ')[1]);
                sum = on[i] * pays + sum;
                if (on[i] >= 0 || on[i] <= 800 && off[i] >= 0 || off[i] <= 800)
                {
                    i++;
                }
            }
            if (sum > (coststops * length))
            {
                count++;
                Console.WriteLine(count);
            }
            else
            {
                Console.WriteLine(count);
            }
            Console.ReadKey();
        }
    }
}
```

Testing

Valid test cases

1. test case: in1.txt

Input – *length*
please input the count of stops: 7
Output

2. test case: in2.txt

Input –the number of pay per stop and operation cost between 2 stops
please input the count of stops: 7 please input the number of pay per stop and operation cost between 2 stops 100 1000
Output

3. test case: in3.txt

Input –*the count of people getting on and getting off*
please input the count of stops: 7 please input the number of pay per stop and operation cost between 2 stops 100 1000 please input the count of people getting on and getting off: 0 15 10 30 0 32 48 0 20 27 26 0
Output
1

Invalid test cases

4. test case

Input – *wrong length*
Length=-1
Output
Asking again: please input the count of stops: length=

5. test case

Intput – wrong *height*
Length=7 Pays=1000 or coststops=-1
Output

Asking again: please input the number of pay per stop and operation cost between 2 stops pays= coststops=

6. test case

Input – *wrong length*
Length=7 Pays=100 coststops=1000 On[i]=-1 off[i]=-3
Output
Asking again: please input the count of people getting on and getting off: on[i]= off[i]=

Further development options

1. Data to be read from file
2. Detection of wrong file input, writing out the location and ID# of error
3. Capability to run multiple times after each other