„Programming”
Big Project

Made By :LinGuoHao[1]

Neptun code: IW3XV9
E-mail:
linguohao111@gmail.com

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

2019. December

Content

User documentation

Task

This program requires you to enter the number of items to be counted, the number of factories, and the number of candy types.

And then enter the factory that produced them, their type, and their price in rows

This procedure can be solved

1) Find the cheapest type of candy in the store and the factory ID which makes them

2) Factory ids that can provide up to different types of candy

3) How many different kinds of factories there are and how much the most expensive candy they can offer

4) How many different kinds of candy are there in the store

5) The types and quantity of candies provided by only one factory

Runtime environment

An IBM PC that is capable of running exe files, 32-bit operating system (eg. Windows 7). No mouse needed..

Usage

Starting the program

The program can be found in the archived file by the name **Condyshop.exe**.
You can start the program by clicking the **Condyshop.exe** file.

Program input

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

Program output

This procedure can output
1) Find the cheapest type of candy in the store and the factory ID which makes them
2) Factory ids that can provide up to different types of candy
3) How many different kinds of factories there are and how much the most expensive candy they can offer
4) How many different kinds of candy are there in the store
5) The types and quantity of candies provided by only one factory

Sample input and output

```
===Closest islands===
# of measurements [2..10000]:12
  1. measurement [0..9000]:3
  2. measurement [0..9000]:0
  3. measurement [0..9000]:2
  4. measurement [0..9000]:0
  5. measurement [0..9000]:4
  6. measurement [0..9000]:3
  7. measurement [0..9000]:0
  8. measurement [0..9000]:0
  9. measurement [0..9000]:3
 10. measurement [0..9000]:0
 11. measurement [0..9000]:2
 12. measurement [0..9000]:0
The closest islands are:
First: 3 3
Second: 5 6
```

Possible errors

The input should be given according to the sample. If the number of measurements is not a whole number, or it is not in the range 2..10000, it will cause a problem. If one of the measurements is not a number, or it is not in the range 0..9000, it also will cause a problem. In the case of an error, the program displays an error message, or asks for the repetition of the input.

Sample of running in the case of invalid data:

```
===Closest islands===
# of measurements [2..10000]:lot
# of measurements [2..10000]:1
# of measurements [2..10000]:1.1
# of measurements [2..10000]:2
  1. measurement [0..9000]:little
  1. measurement [0..9000]:-1
  1. measurement [0..9000]:9001
  1. measurement [0..9000]:0
  2. measurement [0..9000]:
```

Developer documentation

Task

We measure the height above sea level at equal distances during a flight. If we measured 0, it means we were flying above the sea, it we measured a positive height, it means we were above land. Three consecutive measurements are labelled `A`, `B` and `C`. Then `B` means left side of land, if `B>0 and A=0`;

rigth side of land, if `B>0 and C=0`.

Create a program that gives two islands that are the closest to each other among islands. If there are no such islands, the result should be 0.

Specification

Input: $N \in \mathbb{N}$, $Heights \in \mathbb{N}^*$

Output: $Exists \in \mathbb{L}$, $Is1, Is2 \in Island$, $Island = Left \times Right$, $Left, Right = \mathbb{N}$

Precondition: $N = Length(Heights) \land N \in [2..10000] \land \forall i \in [1..N]: Heights_i \in [0..9000]$

Postcondition: $cnt = \sum_{i=2}^{N-1} {}_{IslandBegin(i)} 1 \land$

$islands \in Island^{cnt} \land$

$\forall i \in [1..cnt]: (islands_i.Left \in [2..N-1] \land islands_i.Right \in [2..N-1] \land$

$islands_i.Left \leq islands_i.Right \land IslandBegin(islands_i.Left) \land IslandEnd(islands_i.Right) \land$

$\forall j \in [islands_i.Left..islands_i.Right]: Heights_j > 0) \land$

$cnt < 2 \rightarrow Exists = False \land$

$cnt \geq 2 \rightarrow Exists = True \land$

$\exists i \in [1..cnt-1]: Is1 = islands_i \land Is2 = islands_{i+1} \land$

$\forall i \in [1..cnt-1]: islands_{i+1}.Left - islands_i.Right \geq Is2.Left - Is1.Right$

Definitions: IslandBegin: $\mathbb{N} \rightarrow \mathbb{L}$

$IslandBegin(i) := Heights_i > 0 \land Heights_{i-1} = 0$

IslandEnd: $\mathbb{N} \rightarrow \mathbb{L}$

$IslandEnd(i) := Heights_i > 0 \land Heights_{i+1} = 0$

Comment: If there are less than 2 islands (Exists=False case) the program will write out a 0, and not the logical value (as it was required by the task).

Developer environment

IBM PC, an operating system capable of running exe files (eg. Windows 7). mingw32-g++.exe c++ compiler (v4.7), Code::Blocks (v13.12) developer tool.

Source code

All the sources can be found in the *A1B2C3* folder (after extraction). The folder structure used for development:

File	Explanation
A1B2C3\bin\Release\A1B2C3.exe	Executable code
A1B2C3\obj\Release\main.o	Semi-compiled code
A1B2C3\main.cpp	C++ source code
A1B2C3\test1.txt	input test file$_1$
A1B2C3\test2.txt	input test file$_2$
A1B2C3\test3.txt	input test file$_3$
A1B2C3\test4.txt	input test file$_4$
A1B2C3\test5.txt	input test file$_5$
A1B2C3\doksi\A1B2C3.docx	documentation (this file)

Solution

Program parameters

Contants

```
MaxN          : Integer(10000)     [the max of measurements]
MaxHeight     : Integer(9000)[the max height]
```

Types

```
THeights      = Array(1..MaxN:Integer)
TIsland       = Record(left,right:Integer)
```

Variables

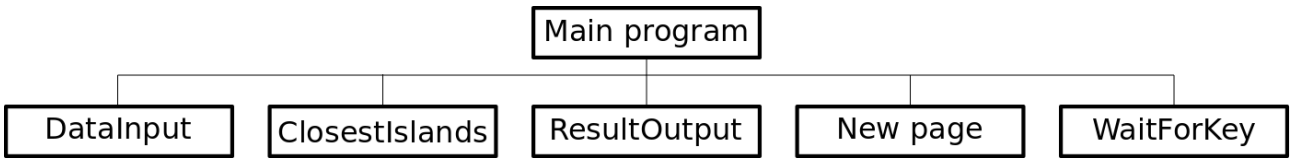
```
N             : Integer
Heights       : THeights
Is1,Is2       : TIsland
```

The structure of the program

The modules used by the program, and their locations:

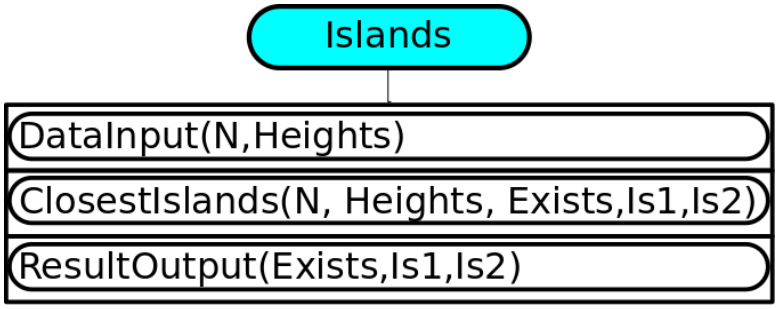
`main.cpp` – the program, in the source folder
`iostream` – keyboard and console management, part of the C++ system
`stdlib.h` – general routines, part of the C++ system

Structure of functions

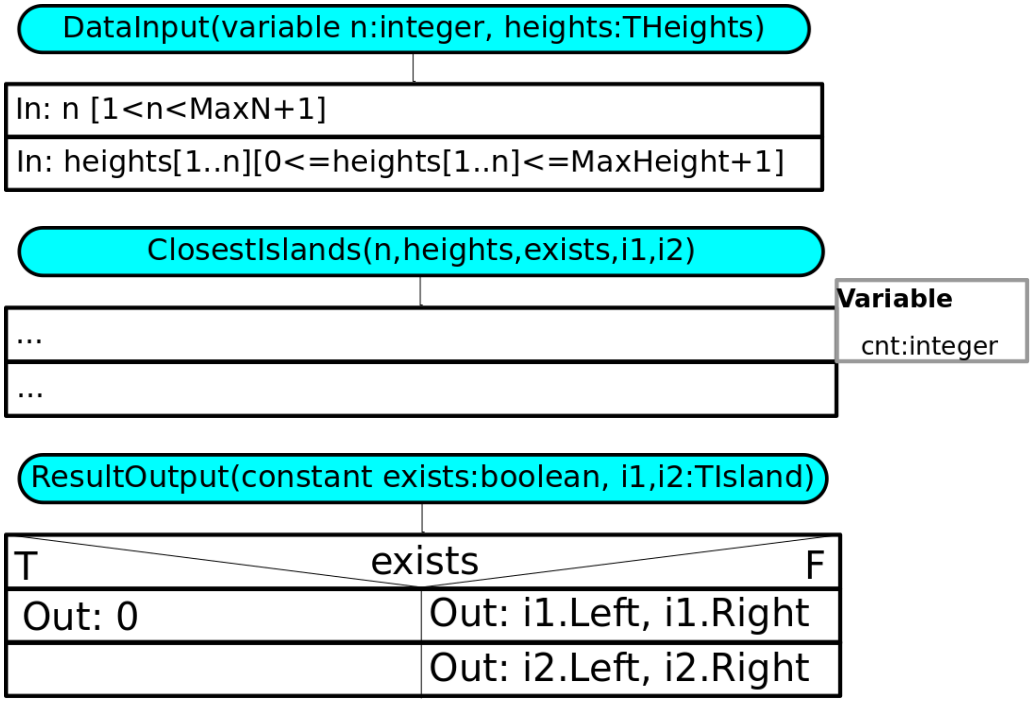


The algorithm of the program

Main program:



Subprograms:



The code

The content of the `main.cpp` file:

```
/*
  Created by: Thomas Tiny
  Neptun: A1B2C3
  E-mail: thomastiny@example.com
  Task: „Big Project” - Closest islands
*/
#include <iostream>
#include <stdlib.h>
```

```
using namespace std;

const string Title="Closest islands";
const int MaxN=10000;
const int MaxHeight=9000;
typedef int THeights[MaxN];
typedef struct{int left,right;} TIsland;
//Input:
int N;
THeights Heights;
//Output:
bool Exists;
TIsland Is1,Is2;

void DataInput(int& n, THeights heights);
void ClosestIslands(int n, const THeights heights,
                          bool& exists, TIsland& i1, TIsland& i2);
void ResultOutput(bool exists, TIsland i1, Tisland i2);
void WaitForKey();

int main()
{
    clog << Title << endl << endl;
    DataInput(N,Heights);
    ClosestIslands(N,Heights,Exists,Is1,Is2);
    ResultOutput(Exists,Is1,Is2);
    WaitForKEy();//should be commented in Biro version!!!
    return 0;
}

void DataInput(int& n, THeights heights)
{
    do{
      clog << "…"; cin >> …;
      …
    }While (…);
    …
}

void ClosestIslands(int n, const THeights heights,
                          bool& exists, Iisland & i1, TIsland& i2)
{
    …
}

void ResultOutput(bool exists, TIsland i1, TIsland i2)
{
    clog << "…"; cout << …;
    …
}

void WaitForKey()
{
  …
}
```

Testing

Valid test cases

1. test case: in1.txt

Input – *no island, lenght is minimal*
N = 2 $Height_1 = 0$ $Height_2 = 0$
Output
0

2. test case: in2.txt

Input – *starts with continent, there are at least 2 islands*
N = 12 $Height_1 = 3$ $Height_2 = 0$ $Height_3 = 2$ $Height_4 = 0$ $Height_5 = 4$ $Height_6 = 3$ $Height_7 = 0$ $Height_8 = 0$ $Height_9 = 3$ $Height_{10} = 0$ $Height_{11} = 2$ $Height_{12} = 0$
Output
Is1 = 3 3 Is2 = 5 6

3. test case: in3.txt

Input – *ends with continent, there are at least 2 islands*
N = … $Height_1$ = … …
Output
…

4. test case: in4.txt

Input – *no continent, only one island*
N = … $Height_1$ = … …
Output
…

5. test case: in5.txt

Input – *there are only continents*

N = …
$Height_1$ = …
…
Output
…

Invalid test cases

6. test case

Input – *wrong length*
N = eleven11
Output
Asking again: N =

7. test case

Intput – wrong *height*
N = 11 $Height_1$ = -1
Output
Asking again: $Height_1$ =

…

8. test case

…

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
4. Visual representation of input data, and emphasizing the result islands with different colors