Interpreter data structure Project Report

An analysis of project design and implementation

Project team :

* Omar Samir
* Yehia Salah

Lecturer: Dr. Saleh Elshehaby

**Table of Contents**

1. **Abstract**

Implementation steps

1. **Teamwork methodology**
2. **Functionality**
3. Overview
4. File load
5. BST
6. Evaluator
7. Heaps
8. Stacks
9. Console interface (main)
10. **Features**
11. Containerization
12. Error handling
13. Global var and extern
14. **Results**

1. **References**

**Abstract:**

The Interpreter implementation is based on the use of functions (caller/callee concept-modules), data structure (BST, Stacks) and headers. Our main objectives were:

* Containerization of the code thus guarantee the reusability of the modules.
* Error handling and cover all the possible cases
* Reaching the maximum optimization and performance possible.

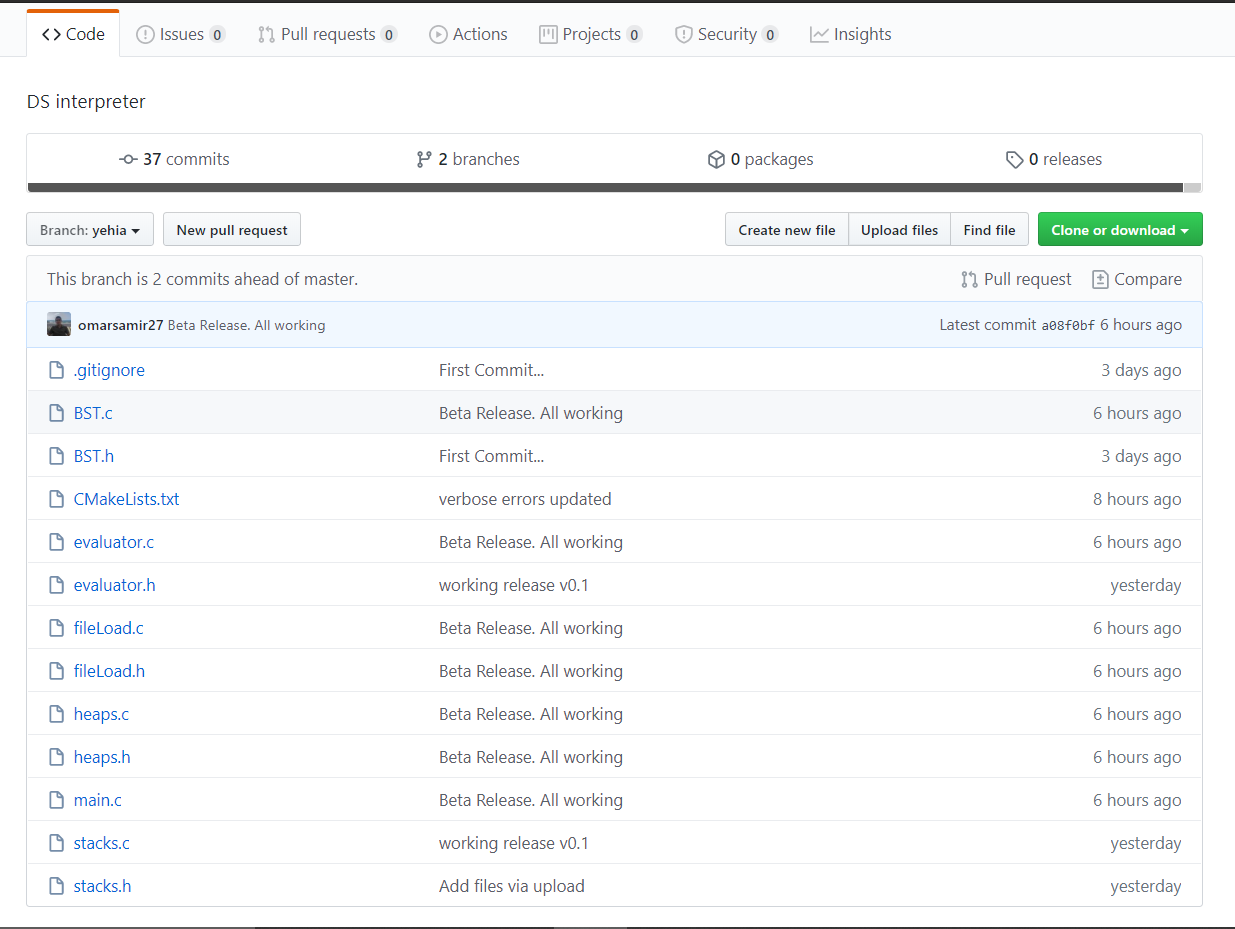
Implementation steps:

* Write pseudo code of the main modules and i/o charts.
* Split the project into main Files (sources + headers).
* The modules implementation of each file.
* Enhancing time complexity (reduced from O(n^2) to O(n\*logn )).
* Exceptions handling and covering all possible scenarios.
* Code annotation, improving program features and make it user friendly.

**Teamwork methodology:**

"Adding manpower to a late software project makes it later"

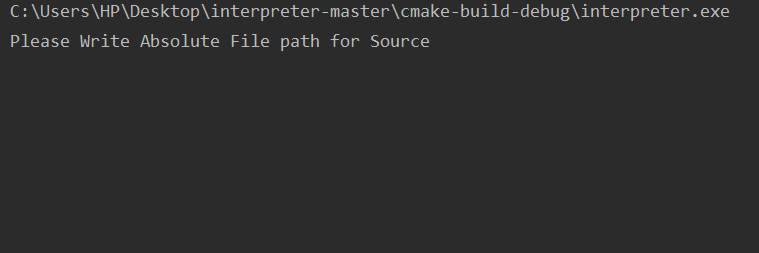
* Split code files on members (every member works individually on assigned file).
* Set the communications between the code containers and preset the functions prototypes.
* Combining all project files.
* Use of version control platform (Github) to deliver the code.

****

**Functionlity:**

***Overview:***

The programs id divided into six main files:

****

\* console picture

1. **Console interface (main)**

Contains the main of the program and the console interface code as pic. above.

1. **File load**

Load the data from the file and return a BST and heap to main.

1. **Evaluator**

Rebuild the RHS string and process an omni-check and evaluate it unless corrupted.

1. **Stacks**

Convert RHS from infix to postfix and evaluate it.

1. **BST**

Store the variable name and its key using bst, sorted alphabetically

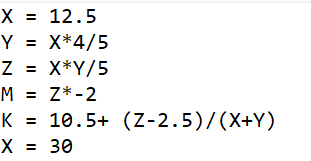
1. **Heaps**

Convert the bst into array and heapsort it.

***Files:***

File (.txt) is used as data buffer.

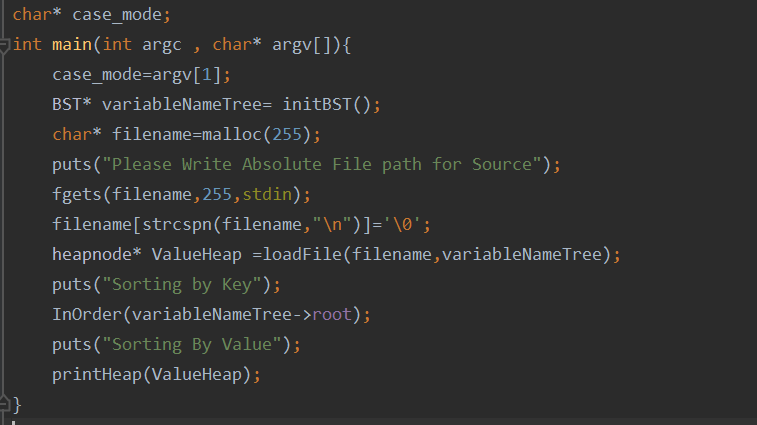
The Text contains the equations (data) that will be interpreted.



\*\* In-depth view of the code\*\*

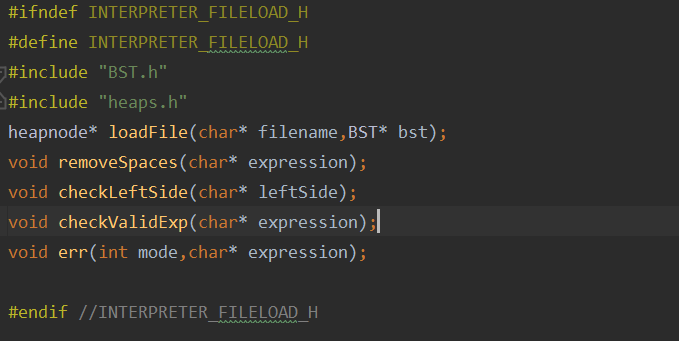
1. ***Console interface (main)***

The main is the start function it prints the user-console interface, take the file path and sends it to loadfile function… It takes argument ‘case’ from the user if he wants the variables to be case-sensitve.

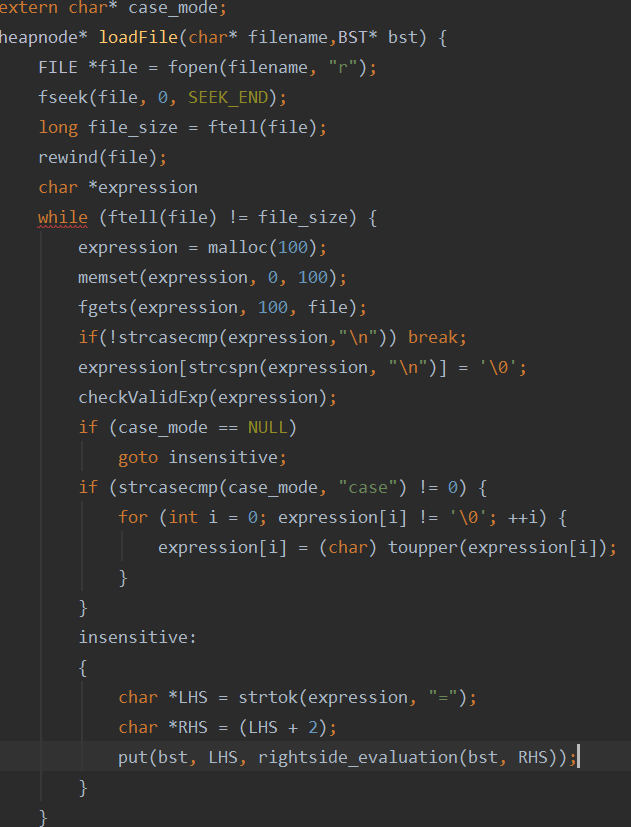


1. ***File load***

Fileload file has 5 functions, its objective is to open the .txt file, collects the data, evaluate LHS, fill the BST, and send the RHS to evaluator.



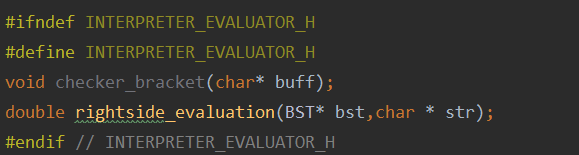
**The main function of the code block: -**



* Load file function return a heapnode pointer(heap array after being filled).
* It loops on the file line by line : check the validity of LHS then send it rightside\_evaluation function
* If LHS and RHS are valid then it create a tree node of the corresponding variable in LHS.
* After the completion of the tree it calls heapsort function to create the heap arr and sort it.

1. ***Evaluator:***

Evaluator file checks the RHS string, rebuild it and send it to be evaluated.

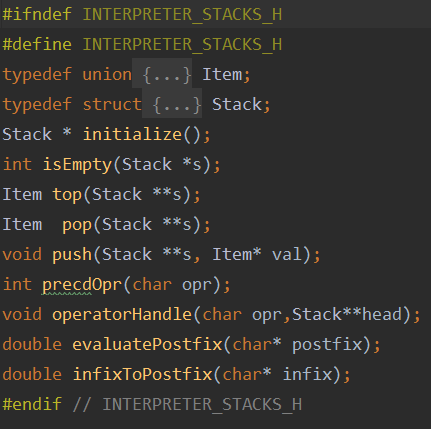


**The main function of the code block: -**



* Rightside\_evaluation function checks the RHS string if valid, then pass it to infixtopostfix function to be evaluated and return the result.
* The function structure is based on nested if condition to check each character of the string and compare it to the char around it.
* The function loops on every character, check it, and rebuilt a new string with comma delimiter.
* If space it skips, it calls error function if double operator, unbalance parentheses, radix point error, inappropriate varname are found.

1. ***Stacks***

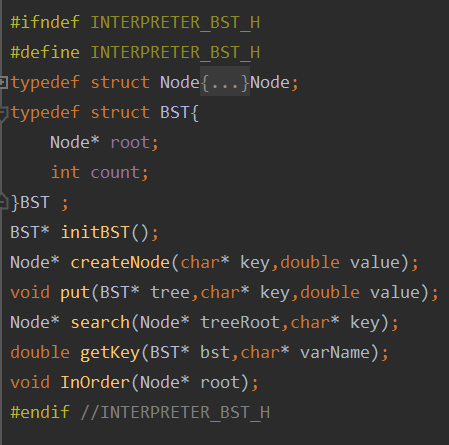
****

Stacks file transform the RHS-edited string from infix to postfix then it evaluates the postfix and returns the results.

**The main functions of the code block: -**

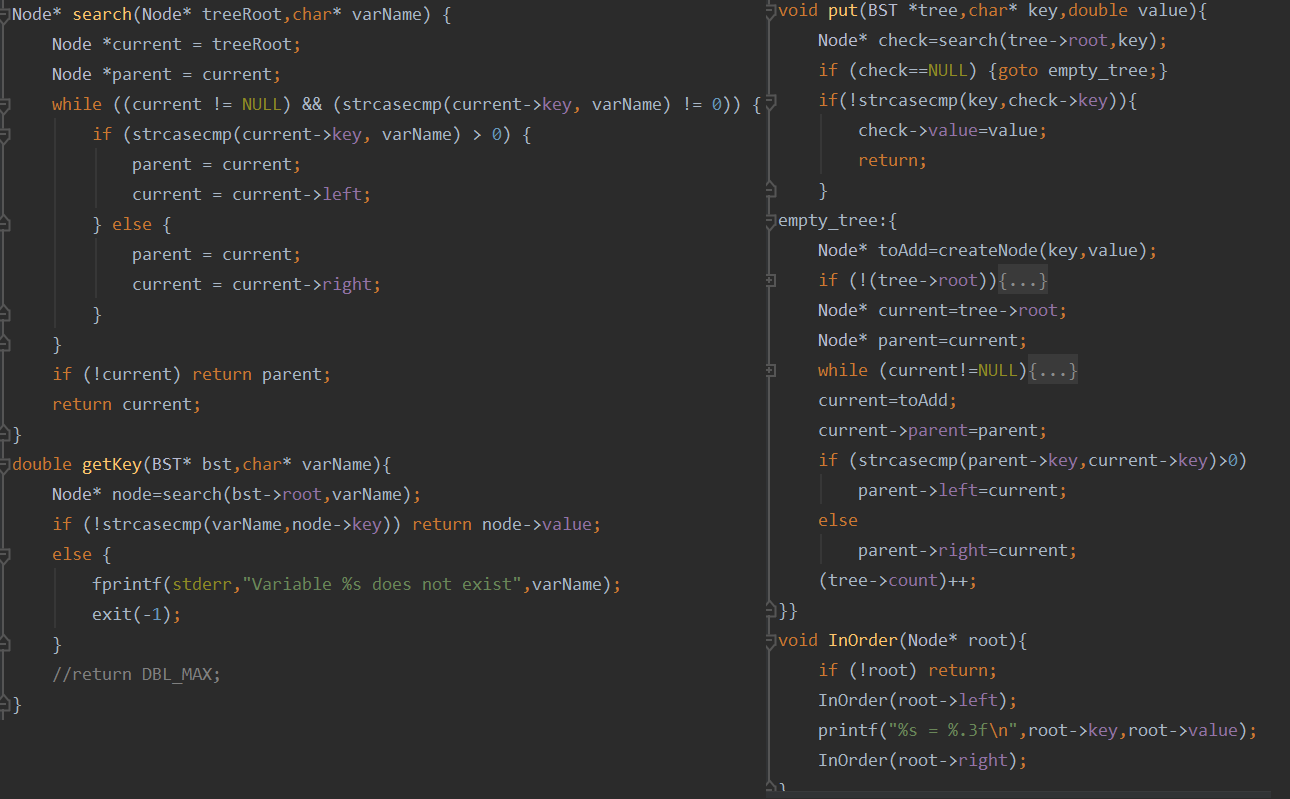


* Infixtopostfix function take a string in form of infix and transform it into postfix using shunt-yard algorithm.
* Evaluatepostfix take postfix and evaluate it using stacks and return the result.

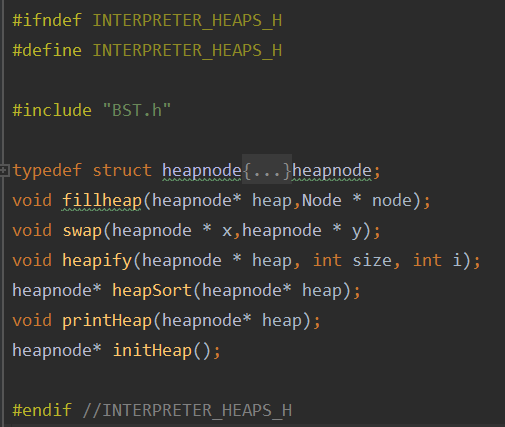
1. **BST**

BST file creates a bst tree, its objective is to insert a tree node for every variable, overwrite it if inserted for more than once. Search for a node and return the value of it

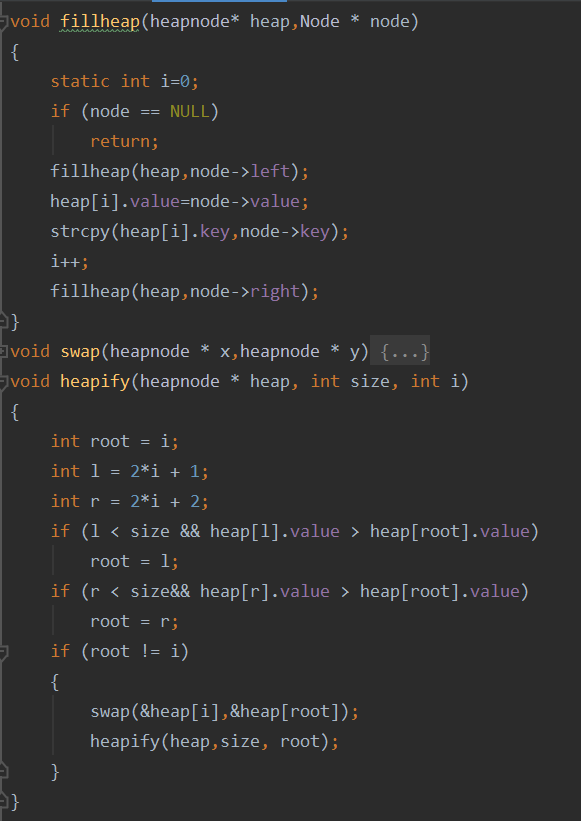
**The main functions of the code block: -**



* Put function create a tree node, it takes key and value as argument …if the node is found it overwrite it .
* Get key function return the value of the varnode searched for.
* Inorder function prints the node tree inorder traversel (from a to z)

1. **Heaps**

Heaps file create an array from bst nodes and sort it using heapsort

****The main functions of the code block: -**

* Fill heap fill the array using a recursive method (same as inorder traversal for traverse all nodes in tree).
* Static int is used to overcome recursive method and I value doesn’t change in recursion.
* Heapify function rearrange a heap to maintain the heap property, the key of the root node is more extreme (greater or less) than or equal to the keys of its children.

**Features:**

***Data formatting:***

* Fgets and strscpn:

Fgets is a function that reads a line from the specified stream and stores it into the string pointed to by str. It stops when either (n-1) characters are read, the newline character is read, or the end-of-file is reached, it’s also safe. The problem faced that the user most probably stops, and press enter before the string ends, that makes the last character “\n”. we have used strscpn to replace the \n by \0 character.

* Getchar and stdinclr function:

Getchar is a function reads a single character from the standard input stream stdin and return it. The problem: getchar stores the char in the buffer, most probably the user after typing the character press enter; we have then to clear the stdin after getchar. We have implementing a stdinclr function for that purpose.



* Space print

Spaceprint is a void function that print spaces for log print so the contact can appear in table form.

* Replace

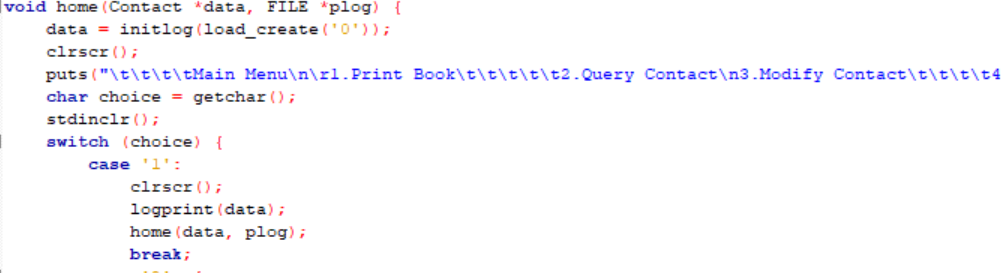
This function replaces a character in string by other character.

***User interface:***

* Home function:

We have implemented this function to act like the user interface function.

After the user choose to do any operation on the contacts the function call itself, back to main menu, until the user choose the exist option.



* Delay and sysclr:

We have used the delay and sysclr function to give the user a better user experience.

***The validation test:***

We have implemented several functions to check the validity:

* To check names: isstring

The isstring loop on the string to check whether there were any non-letters characters using isalpha.

* To check email: strstr

The strstr checks the existence of @.

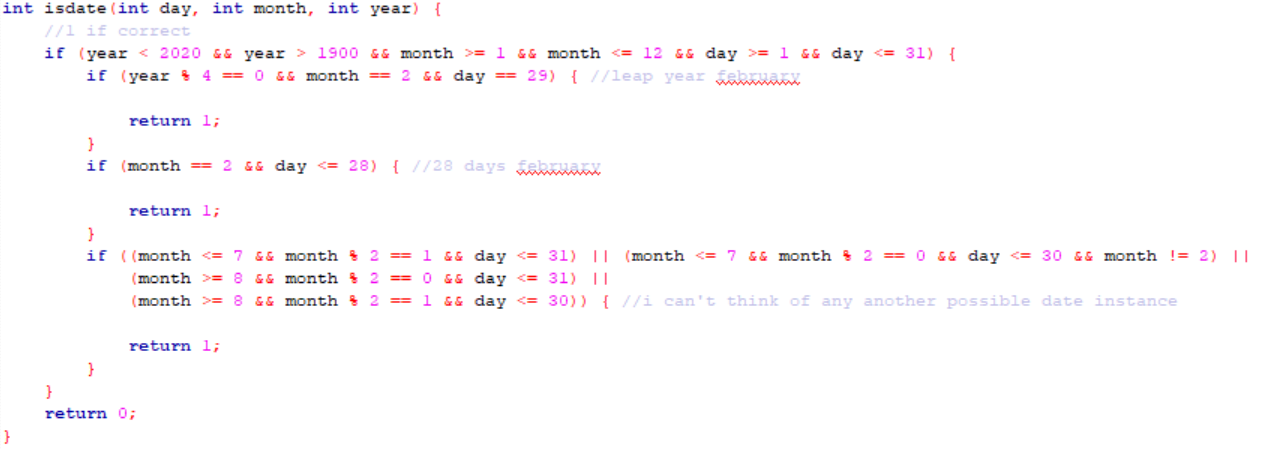
* To check phone number: isphone

The isphone loop on the string to check whether there were any non-digit characters using digit.

* To check date: is date

The isdate function checks if the year is between 1900 and 2020, the months are between 1and 12 and the days are between 1 and 31

Then we check the validity of dates with even and odd months.

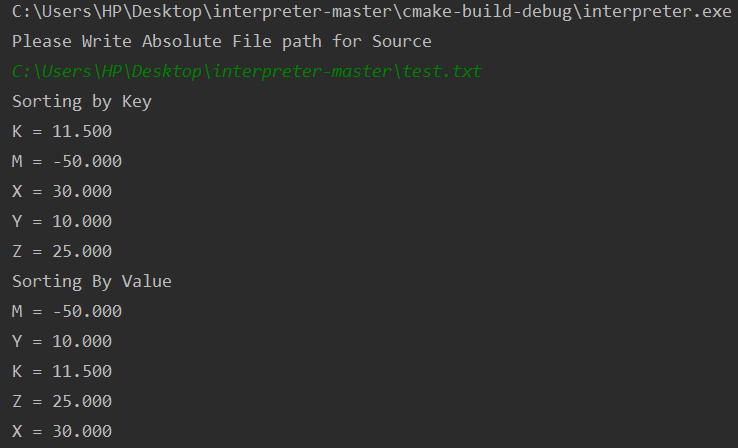


**Results: “FINALLY THE TESTING PART”**

The program works properly:

* ALL FUNCTION HAVE BEEN TESTED
* SPECIAL CASES HAVE BEEN TESTED

AND HERE IS A BRIEF OF THE PROTOTYPE FILE TEST:



**References:**

[**https://github.com/**](https://github.com/)

[**https://stackoverflow.com/**](https://stackoverflow.com/)

[**https://www.tutorialspoint.com/**](https://www.tutorialspoint.com/)