Alexandria University
Faculty of Engineering
Computer and Communication Department

Subject

Discussion Report for Final Project (Interpreter)

Done By

Name	Id	Group	Section
Mohamed Essam Eldin Mohamed	6367	3	2
Kareem Mohamed El Baroudy	6424	3	1
Loay Hesham Mohamed	6439	3	1
Bassel Ashraf Mohamed Elsheikh	6565	3	2
Omar Hossam Eldin Ebraheem	6667	3	1

Supervisor

Dr. Mohamed Salama

2019 – 2020

Introduction

In this report it is used to discuss and analysis of the implementation and simulation of the (Interpreter) Project , In computer science, an interpreter is a computer program that directly executes instructions written in a programming or scripting language, without requiring them previously to have been compiled into a machine language program the interpreter is located in all the IDE's of the programming languages , there are many programming languages that need the interpreter in order to read and execute there operations and commands , also the interpreter can executes instructions and print the result of execution , also the interpreter can compile and execute the read line by line , so the interpreter can play the important rule in compiling and executing the operations and commands of many programming languages .

Materials and procedures

- During implementing and simulating the interpreter project using (Code::Blocks) IDE by (C programming language) , it is used to separate the source code in five separated headers and six separated (.C) files , in order to make the code more readable and understandable .
- It has been using 34 functions to implement the interpreter project :
  1. int check(char* equation);
  2. char* remove_spaces(char* str);
  3. void interpretFile(char* filename);
  4. int checkfirstequation(char* input);
  5. char *replaceVariable(char *input,char *oldW,char *newW);
  6. char* swapvariables(char* rightside);
  7. int AddToArray(node* root, variable arr[], int i);
  8. node* newnode(char* key,float value);
  9. node* search(node* n,char *key);
  10. node* insert(node* node,char* key,float value);
  11. void put(char* key,float value);
  12. void InOrdertraverse(node *t);
  13. int size(node* root);
  14. int checkOperator(char input);
  15. char* cutTillOp(char* input);
  16. char* AddSpaces(char* input);
  17. int weight(char a,char b);
  18. float calc(char o, float op1, float op2);
  19. void infixToPostfix(char* in,char* post);
  20. float evaluatePostfix(char* post);
  21. void replaceNewLineBySpace(char *s);
  22. Item top (Stack *s);
  23. int isEmpty(Stack* s);
  24. int isFull(Stack* s);
  25. Stack* initialize();
  26. void push(Stack* s,Item a);
  27. Item pop (Stack* s);

28. int parent(int i);
29. int LeftChild(int i);
30. int RightChild(int i);
31. void MaxHeapify(variable arr[], int n, int i);
32. void BuildHeap(variable arr[], int n);
33. void SortHeap(variable arr[], int n);
34. void print(variable arr[], int n);

- All of the above 34 functions plays an important role in implementing the interpreter project.

Analysis and discussion

- In this section it is used to discuss function by function for it's input parameters , type of return and it's own functionality .

1. **int check(char* equation);**
   in check function , it is used to take an equation as a string in order to check the equation empty and clear from any syntax and math errors , also it is used to check any unknown symbols exists , more than one variable in the left hand side , and other many math and syntax errors , check functions used to return a flag for 1 if the equation is true and clear from any errors , otherwise it returns 0 .

2. **char* remove_spaces(char* str);**
   in this function , it is used to take the string read from the supported text file , it used to remove all the spaces between the variables , operators , operants and parenthesis , in order to make the mission more easily for the function (addspaces) to add only single space between any the variables , operators , operants and parenthesis , the remove_space functions used to return the modified string after removing all the spaces from it .

3. **void interpretFile(char* filename);**
   the interpretFile can act as a core brain in the project as it takes the file name , makes test cases and calla the suitable functions in order to implement the project well , it does not return any parameter .

4. **int checkfirstequation(char* input);**
   the checkfirstequation function used to take the first line read from the file as input , it checks the first line read if it is constant or not , because the first line must be constant in order to make the next equations depends on it , the checkfirstequation function return a flag which is 1 if the first equation is constant , otherwise it returns 0 .

5. **char *replaceVariable(char *input,char *oldW,char *newW);**
   in this function , it takes three parameters : the input string , the old string to be replaced , and the new string to be put instead , in this function it is used to replace and substitute the variable name with it's value in the equation in order to be calculated , it returns the new string after replacing and substituting the variable with it's own value .

6. **char* swapvariables(char* rightside);**
   in this function , it takes only the input string , in order to search on the BST for the variable key , to get it's own value in order to be substituted and passed to replaceVariable function , it returns the modified equations with the substituted values .

7. **int AddToArray(node* root, variable arr[], int i);**
   in this function , it takes three parameters : the root of the tree , the array to be filled with all the nodes of the tree , and the indexes of the array in order to be make the function called recursively .

8. **node* newnode(char* key,float value);**
   in this function it is used to take two parameters : the key of the variable (variable name) , and the value of the variable , in this function it used to make and factorize a new node in order to be inserted in the tree , it returns the node made .

9. **node* search(node* n,char *key);**
   in this function it takes two parameters : a node of the tree , and the key of the node , it used to search from the tree and look for a node for a specific passed key , it returns the node found from the search operation .

10. **node* insert(node* node,char* key,float value);**
    in this function it is used to take three parameters : the node to be inserted , the key and the value of the node , it is used to insert a node with specific passed parameters in the tree , it returns the node pointer of the node that has been inserted .

11. **void put(char* key,float value);**
    in this function it is used to take two parameters : key of a variable and the value of the variable , in this function it is used to insert the specific passing parameters in the tree , without passing the root of the tree , as it declared as a global variable , it returns no parameters .

12. **void InOrdertraverse(node *t);**
    in this function it takes the root of the tree, in order to print the keys and values of the tree in the in order traverse sequence (Left Root Right) i.e. : ascendingly order , it returns no parameters .

13. int size(node* root);

in this function it takes the root of the tree , it is used to get the size and the number of nodes of the tree , it returns the number of nodes of the tree .

14. int checkOperator(char input);

in this function , it is used to take an input , in order to check the input if it is an operator or not , it returns a flag of the checked operation .

15. char* cutTillOp(char* input);

in this function , it takes the a string , in order to cut the passed string till the operator , it returns the modified string after cutting process .

16. char* AddSpaces(char* input);

in this function , it takes a string as input , in order to add only a single space to the string in order to passed to infixToPostfix function , it returns the string after adding a single space .

17. int weight(char a,char b);

in this function it takes two parameters as operators , it is used to check the priority of the passed operators , it returns a flag of the checked priority .

18. float calc(char o, float op1, float op2);

in this function it takes three parameters : two operants with one operator , in order to calculate the equation and return the value as afloat value .

19. void infixToPostfix(char* in,char* post);

in this function , it takes two strings , one of them is equation written in infix principle and the other is empty in order to be filled with the postfix written principle . it returns no parameters as the function is called by reference .

20. float evaluatePostfix(char* post);

in this function it takes the string which written in postfix principle in order to be evaluated and return the value again .

21. void replaceNewLineBySpace(char *s);

in this function , it takes a string that read from a text file with fgets function , it is used to add a (‘\0’) in the end of the read line instead of the (\n) put by fgets , it returns no parameters .

22. Item top (Stack *s);

In this function it takes the pointer of the stack , in order to return the top element of the stack (first element) .

23. int isEmpty(Stack* s);

in this function it takes the pointer of the stack , in order to check if the stack is empty or not , it returns a flag with the checked operation .

24. int isFull(Stack* s);

in this function it takes the pointer of the stack as input . in order to check the stack if it is full or not to avoid the stack overflow problem , it returns a flag of the cheeked operation .

25. Stack* initialize();

In this function it takes no parameters , it is used to implement and initialize the stack in the first time , it returns a pointer of the implemented stack .

26. void push(Stack* s,Item a);

in this function , it takes two parameters : the pointer of the stack and the item to be pushed and inserted in the stack , it is used to insert and push the passed items in the stack , it returns no parameters .

27. Item pop (Stack* s);

In this function it takes the pointer of the stack as input in order to pop the first element of the stack , and return the poped item .

28. int parent(int i);

in this function it takes an index as input parameter in order to get the parent of the passed index on the heap based on building the heap from index 0 , it returns the index of the detected parent .

29. int LeftChild(int i);

in this function it takes an index as input parameter in order to get the left child of the passed index on the heap based on building the heap from index 0 , it returns the index of the detected left child .

30. int RightChild(int i);

in this function it takes an index as input parameter in order to get the right child of the passed index on the heap based on building the heap from index 0 , it returns the index of the detected right child .

31. void MaxHeapify(variable arr[], int n, int i);

in this function it takes three parameters : array to be inserted in the heap , size of the array , and the index to be inserted in , in this function it is used to build the heap in max heap principle and characteristics , it returns no parameters .

32. void BuildHeap(variable arr[], int n);

in this function it takes two parameters : array to be inserted and the size of the array , it is used to call the MaxHeapify function in order to build the heap well , it returns no parameters .

33. void SortHeap(variable arr[], int n);

in this function it is used to take two parameters : array to be sorted and the size of the array , it is used to sort the array build by max heap principle in ascending order , it returns no parameters .

34. void print(variable arr[], int n);

in this function it takes two parameters : the sorted array by heap sort algorithm and the size of the sorted array , it returns no parameters .

- Note that by inserting lower cases and upper cases variables in the supported text file , the interpreter deals with them as different variables , so it is forced to use strcmp function instead of strcasecmp function while inserting and searching from the tree , so the askii code of the lower case variables is more the askii code of the upper case variables , so that it is right and expected to make the lower case variables be the last and the big sorted variables .

ASCII printable characters

32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

Figure 1

Data

In this section there are some displayed screen shots for the sample runs of the implemented interpreter project .

- **screen shot of sample run 1**

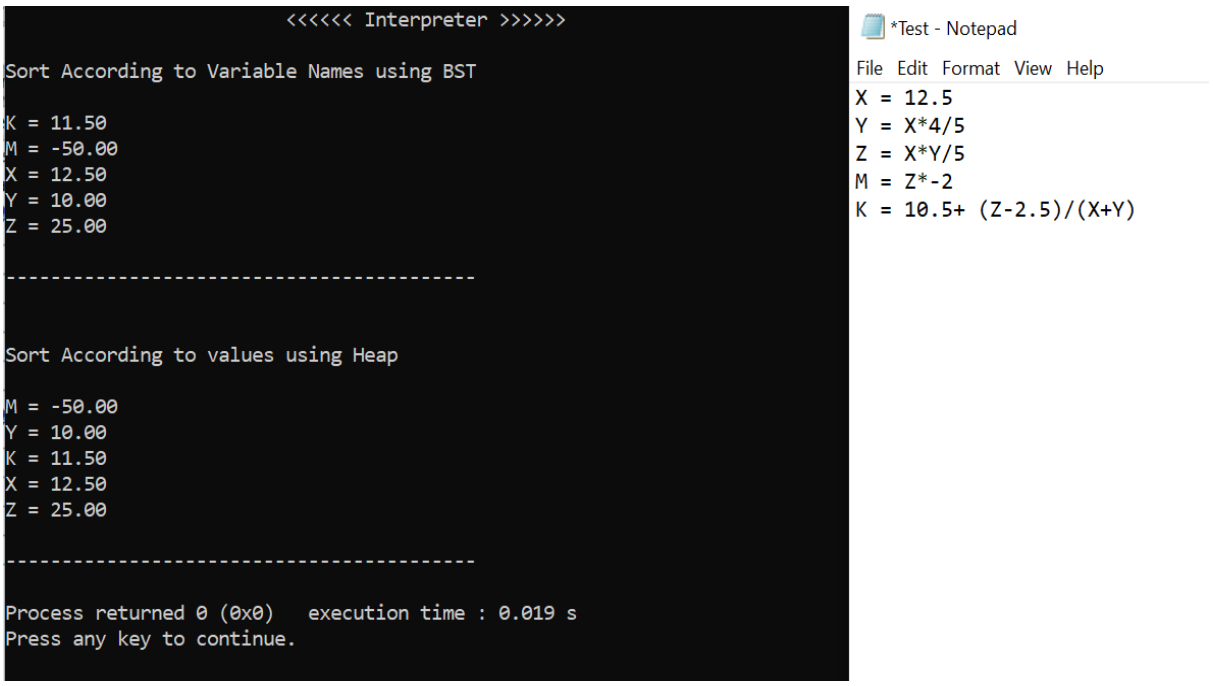

Figure 2

- **screen shot of sample run 2**

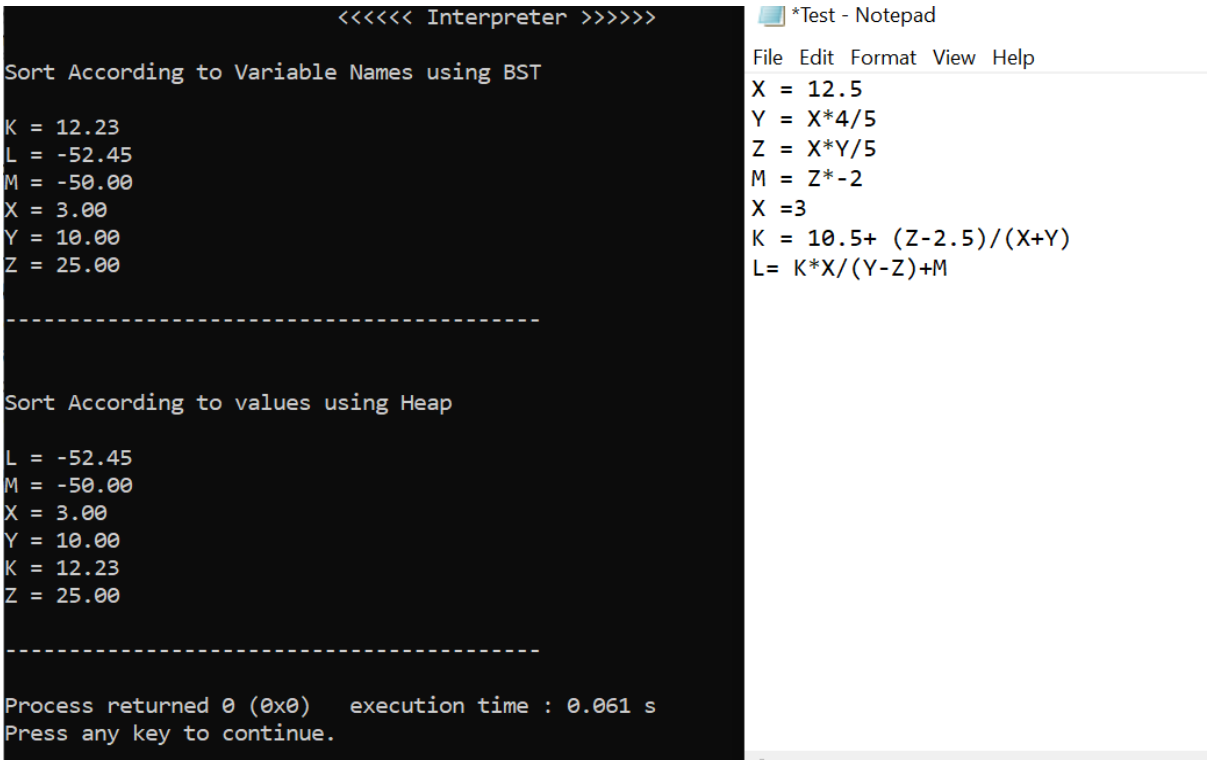

Figure 3

- **screen shot for sample run 3**

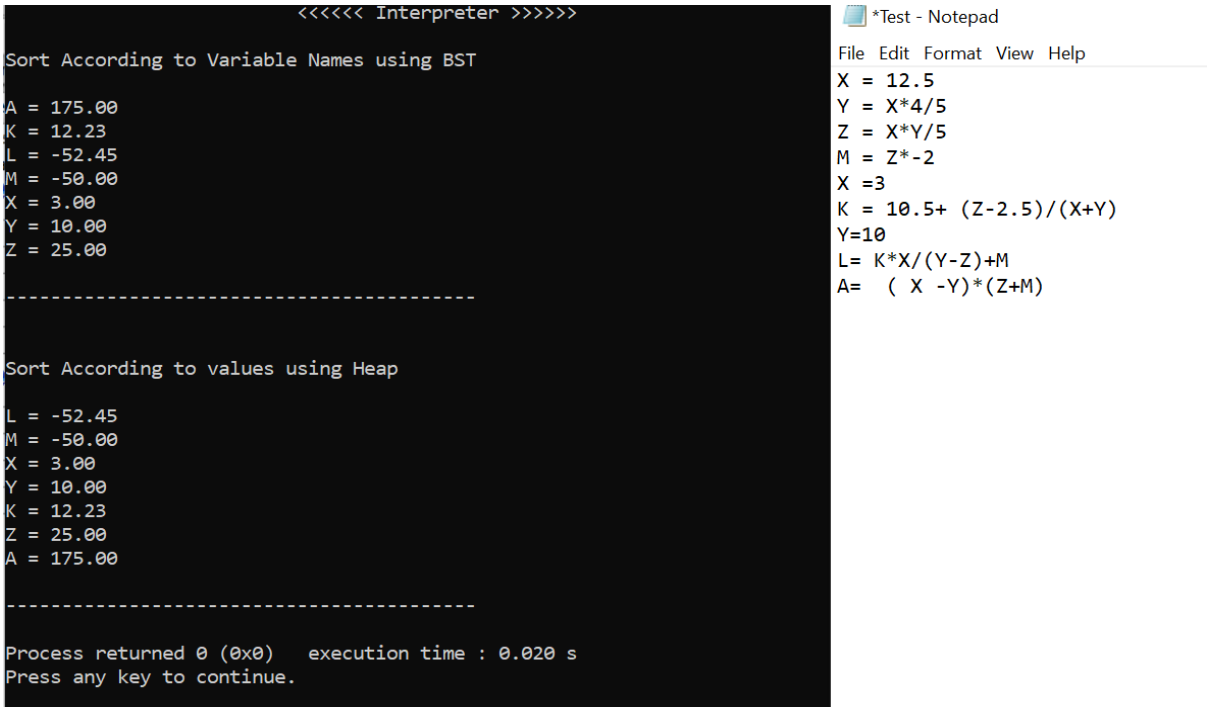

Figure 4

- the above sample runs are some runs with more varieties to show the well performance of the implemented interpreter .

conclusion

- The interpreter is a computer program that directly executes instructions written in a programming or scripting language, without requiring them previously to have been compiled into a machine language program , the interpreter is located in all the IDE's of the programming languages , there are many programming languages that need the interpreter in order to read and execute there operations and commands.

- Finally , the interpreter implementation took much time and more effort from the team in order to implement the project well , and deliver a good simulation of the interpreter program , but in the other hand , the all members of the team were very excited , and have a such enjoyable time implementing the interpreter , hope from God to gain admire of the supervisor , instructors and teaching assistants .

References

1. Code::Blocks IDE
2. https://classroom.google.com/c/NjExMTM3ODA5NjNa
3. https://en.wikipedia.org/wiki/Interpreter_(computing)
4. https://theasciicode.com.ar/ascii-printable-characters/capital-letter-x-uppercase-ascii-code-88.html
5. http://sticksandstones.kstrom.com/appen.html
6. https://www.youtube.com/watch?v=gyA7uDlazkc&t=322s
7. https://www.youtube.com/watch?v=1EeetMPACMI
8. https://www.youtube.com/watch?v=IUylyTdX_8A