1 A program reads data from a text file and stores it in a stack. The stack `Stack` is stored as a 1D array of up to 20 elements. The pointer `TopOfStack` stores the index of the last element stored in the stack.

(a) `Stack` is a global array of strings with all elements initialised to `"-1"`

`TopOfStack` is a global variable initialised to `-1`

Write program code to declare and initialise `Stack` and `TopOfStack`

Save your program as **Question1_J25**.

Copy and paste the program code into part **1(a)** in the evidence document.

[2]

(b) The function `Push()` takes a string parameter and attempts to store it on the stack.

The function returns `-1` if the stack is full.

The function returns `1` if the parameter is successfully pushed onto the stack.

Write program code for `Push()`

Save your program.

Copy and paste the program code into part **1(b)** in the evidence document.

[4]

(c) The function `Pop()` returns the next item from the stack.

The function returns `"-1"` if the stack is empty.

Write program code for `Pop()`

Save your program.

Copy and paste the program code into part **1(c)** in the evidence document.

[4]

(d) The text file `StackData.txt` stores numbers and mathematical operators. Each number and operator are on a new line in the text file.

The mathematical operators used in this program are:

mathematical operator	meaning
+	addition
–	subtraction
/	division
*	multiplication
^	power of

The procedure `ReadData()`:

- takes a string filename as a parameter
- opens the file and reads in each line of data
- uses the `Push()` function to insert each line of data onto the stack
- outputs `"Stack full"` if any data cannot be stored on the stack because the stack is full
- uses exception handling when opening and reading from the text file.

The procedure needs to work for a file that contains an unknown number of lines.

Write program code for `ReadData()`

Save your program.

Copy and paste the program code into part **1(d)** in the evidence document.

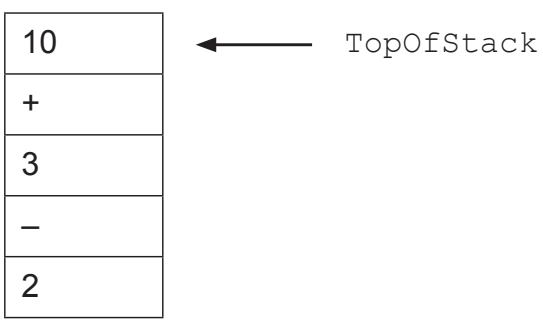
[6]

(e) The values stored in the stack are used to perform mathematical operations.

A total is initialised to the first value in the stack. The first value in the stack will always be a number.

The next value will be a mathematical operator and the next value will be the number to apply to the calculation. This is repeated until there are no values left in the stack. There will always be a number following a mathematical operator.

For example, the contents of a stack are:



The total is initialised to the first value in the stack: 10

total = 10

The next two values are: + 3

total = 10 + 3 = 13

The next two values are: – 2

total = 13 – 2 = 11

The final total is 11 and this is returned.

Write program code for the function `Calculate()` to:

- take each value from the stack
- calculate and return the final total.

Save your program.

Copy and paste the program code into part **1(e)** in the evidence document.

[7]

(f) (i) Write program code to extend the main program to:

- take a filename as input from the user
- call `ReadData()` with the input filename
- call `Calculate()`
- output the final total.

Save your program.

Copy and paste the program code into part **1(f)(i)** in the evidence document.

[2]

(ii) Test your program twice with the file names:

```
StackData.txt
SecondStack.txt
```

Take a screenshot of the output(s).

Save your program.

Copy and paste the screenshot into part **1(f)(ii)** in the evidence document.

[2]