COMP-273, Fall 2020, Assignment 3

School of Computer Science
McGill University
Available On: Saturday, October 31st, 2020.
Due Date: Friday, November 13th, 2020 by 11:59 pm.

By handing in your solutions using *mycourses*, you declare that you have followed the assignment submission instructions at the end of this assignment. **Late policy: 10% off of the total marks, per day late, for up to 2 days. If submitted 48 hrs or more after the deadline, your assignment will not be accepted.**

1 Question 1: 1D Array Manipulation (50 marks)

In this question you will manipulate the contents of a one dimensional array. To do so you will modify the template file 'array.asm'. In this file you are given a *beginarray*, whose contents are integers, and where the special integer -999 is placed at the end to mark the end of the array. In addition, 100 words of space (400 bytes) is allocated for *array* - we will assume that it will never store more than 100 integers. You are free to write additional helper subroutines, and to use them within your subroutines.

1.1 Helper Subroutine

Write a subroutine *le* of the array, not counting the spe

Write a subroutine *copyarray* that takes two pointers as it y and a pointer to the second array. It should copy the integer contents o array, including the special integer. It does not return anything.

Write a subroutine *printarray* that prints the contents o spaces, upto but not including the special integer.

Test the above functions by writing a main function that copies the contents of *beginarray* to *array*, and then prints the contents of *array*. **Save your code as q1a.asm**

From this point onwards, all your operations will be on *array*. For the remainder of this question you will build on the code you have so far, to implement the following four operations: Insert, Delete, Sort and Quit, by further modifying 'array.asm'. Each time you will use standard I/O to prompt the user to type a valid operation. The user should use one of the four keyboard inputs below:

- i: Insert
- d: Delete
- s: Sort
- q: Quit

If an invalid input is entered, your program should print an error and should prompt the user to enter a valid keyboard input. Once a valid input keybord character has been entered, the program should proceed to carry out that operation. The operations should be carried out on *array*, which

has by now been initialized. Indexing should start with a count of 0 representing the first element in the array. You can use the helper subroutines including *length, printarray* or any additional ones you have written. For each subpart you will be adding new functionality to your existing program, and then saving the code, allowing us to test it.

1.2 Insert Operation - 10 marks

This operation is triggered when the user inputs i. Then, the user should be prompted for the index to insert a value in the array. Your program must check if the index is valid, i.e., it is between 0 and length(*array*) - 1. If it is invalid, return an error and keep prompting the user until a valid index is entered. Once a valid index is entered, the user should be prompted for the integer value they wish to insert. Your program should insert that value at the correct position in the array and then print the result. **Save your code as q1b.asm**

For example, if *array* has the following contents: 5 3 4 1 2; the insert command should work as follows:

```
Enter a command : i
Enter an index: 3
Enter a value: 10
```


The current array is 5 3 4 10 1 2

In a case where an invalid ind


```
Enter a command: i
Enter an index: 8
Invalid index.
Enter an index: 4
Enter a value: 6
```


The current array is 5 3 4 10 6 1 2

1.3 Delete Operation - 10 marks

This operation is triggered when the user enters d. The user should then be prompted for the index at which a value would be deleted from the array. Your program must check if the index is valid i.e., it is between 0 and the length(*array*) - 1. If it is invalid, return an error and keep prompting the user until a valid index is entered. Once a valid index is entered, your program should delete the element at that index. **Save your code as q1c.asm**

For example, if *array* has the following contents: 5 3 4 1 2; the delete command should work as follows:

```
Enter a command: d
Enter an index: 2
```

The current array is 5 3 1 2

In a case where an invalid index is entered, the program should work as follows:

```
Enter a command: d
Enter an index: 7
Invalid index.
Enter an index: 3
```

The current array is 5 3 1

1.4 Sort Operation - 20 marks

This operation is triggered when the user enters *s*. Your program should display the sorted array. You are allowed to use *any* sorting algorithm for this operation e.g., selection sort, or bubble sort, so be sure to comment your code accordingly. **Save your code as q1d.asm**

For example, given the following array: 5 3 4 1 2; the sort command should work as follows:

```
Enter a command: s
```

The current array is: 1 2 3 4 5

1.5 Quit Operation - 2 ma

This operation is triggered [illegible]rd. This should be the only time your progra

2 Question 2: 2D Array Manipulation and Recursion (50 marks)

2.1 Warmup - 10 Marks

In this question you will gain some experience in I/O from a file, 2D array manipulation and recursion. The goal of this assignment is to create a program that can fill in a close contour in an image. The basic algorithm (the most robust one I know of) is a recursive one. For this question you can write helper subroutines and also introduce any additional variables, strings, or constants in the .data segment that you wish to use. Start by downloading and installing *gimp-2.10*, which is free software for image viewing and manipulation.

Using the template 'fileio.asm' as a starting point, write a MIPS procedure *readfile* that takes as an argument the address of a string that contains a valid filename. Then using appropriate syscalls, read from that input file and simply print the content of that file to screen.

In the template, there are two input files called test1.txt and test2.txt which you will be using for testing in this assignment. To accomplish this task, you are allowed to create a large buffer i.e., one that is larger than the expected number of ASCII characters (bytes) in the input file. The body of your code should work by calling the procedure you have written. The procedure should open the file, read its content as ASCII characters, store the content in the buffer, and then print the content and then close the file. Refer to 'mycourses->Content->Tutorials and Notes->MARS->syscalls' for handy routines there. Error statements should be printed if there are any errors in opening the file, reading from it or closing the file.

After reading the conte

procedure called *write* the following information to that file:


```
P1
50 50
```


Then, starting on a new line it should write out the content that was rea
then close the file. Error statements should be printed if there are any errors in opening the file, reading from it or closing the file.

Your main function should simply test your procedures by calling *readfile* and *writefile* in sequence, but with pointers to valid strings for the file names. **Save your code as fileio.asm**

If things are properly when you view test1.pbm with *gimp-2.10* (use the bottom left of the display window to zoom to 800%) you should see something interesting. For further testing you can repeat the above steps by taking test2.txt as input and generating test2.pbm as output.

2.2 Contour Filling - 40 marks

1. The template for this question is provided is provided in 'contourfill.asm'. In this file, both a buffer as well as space for a 2D array are defined. You are free to introduce other variables, strings, or constants in the .data segment. You can assume that the array will have dimensions 50×50.

2. Read the data in test1.txt into the buffer.

3. The data read into the buffer should now be converted to consecutive integers and then stored in a 2D array of length 50 * 50, i.e., one that has 50 columns and 50 rows. That 2D array will actually be represented as a 1D array*. Finally, take care to convert the entries in that buffer (which are in ASCII) to their numerical values in base 10 when storing them in the 2D array. For this assignment you can assume that the numerical values in the array will be either 1 or 0.

4. Write a procedure named *fillregion*, which works as follows. It takes 3 input arguments. The first is a pointer to the 50×50 2D array containing numerical values (1's or 0's) and the second and third arguments are the x and y integer coordinates of a seed point that lies within a contour, i.e., it is an interior point, with value 0. You can find such points by viewing test1.pbm or test2.pbm in *gimp-2.10*. The procedure then examines all 8 neighbors of this seed point and for each neighbor it:

   (a) Checks if that neighbor has a value of 0 in the 2D array, i.e., it is inside the contour.

   (b) If it does it changes that value to 1, and then calls itself again (this is the recursive part) with a pointer to the same 2D array but now with new arguments - the x and y coordinates of that neighbor.

5. Once your procedure has terminated you need to copy the ASCII value corresponding to each entry in the 2D array into a new buffer. Then open a file called testfill.pbm and write the following informa


```
P1
50 50
```


   Then, starting on a new line, write out the content that was wri
   then close the file. Error statements should be printed if there are
   file, or closing the file. If your procedure has worked properly testfill.pbm should contain an image which has a filled contour, corresponding to the region you chose via your selection of the seed point. **Save your code as contourfill.asm**

*[2D to 1D indexing]: Normally, in a language like Java, we would simply specify the respective array positions of i and j. (Let i represent the row we are currently at, and j represent the column we are currently at). In MIPS, however, our 2D array is stored as values in a 1D array. It is clear to see that for any position [i,j] in our 2D array, we can retrieve that position by simply calculating (i*width) + j. Since i represents rows, whenever we add a width (for this assignment, width is 50) we at are essentially going to the next row in our conceptual 2D array. j simply represents which column we are looking at.
Since we are currently looking at the correct range of data in our 1D array based on i*width, we simply determine which column to look at by adding j to that value. We now have the position of interest.

ASSIGNMENT SUBMISSION INSTRUCTIONS

Each student is to submit his or her own unique solution to these questions, electronically, in *mycourses*. By handing in this assignment, you declare that the work you are submitting is your own. We will be running Moss to carry out a pairwise comparison between each submission to check.

1. Submit your solution to myCourses before the due date.
2. Ensure that your code is well commented.
3. Zip your solution files - q1a.asm, q1b.asm, q1c.asm, q1d.asm, q1e.asm, fileio.asm and contourfill.asm, as well as sample output .pbm files (for question 2), into a single compressed file and rename it with your student ID number, e.g., 260763964.zip. Ensure that you use only the *.zip* format and no other compression software e.g., .rar, .7z, etc.
4. Submit this single compressed file on mycourses under Assignment 3.
5. Your code *must* assemble and run, even if the final solution is not quite working. Partial marks will be awarded for correct high-level control and use of conventions. If something is not working, comment-out the broken parts of code and write a comment or two about what you expect to happen, what is happening, and what the problem may be. *If your code does not assemble you will receive very few points.*
6. Hints, suggestions an *mycourses* as questions arise. Ev on board.
7. Make sure that you submit a single file (the zipped file), not many file
8. Once you have submitted your assignment, download the zip fi t it is indeed what you intended us to grade. This step is critical because a non-trivial number of you will submit the wrong zip file, or a corrupted version. You cannot submit a corrected file later, i.e., after the submission deadline and the two day "late" window have passed.