S&T-1st C Programming Lab 8

2D Array

14 March 2025 Friday 6:50pm

Assignment 1

Unzip lab8.zip sent to you by email attachment. The content in **image.c** is related to a digitized image of an English letter. The image is composed of 8x8 small rectangles, and the content of each rectangle is a digit representing the light intensity. Assume that the range of the light intensity in the image is unknown.

```
6 6 6 6 6 6 6 6
6 2 2 2 2 2 2 6
6 3 1 0 1 1 2 7
6 6 5 0 1 5 6 6
6 6 5 3 3 5 6 7
7 6 5 3 2 6 6 7
7 6 6 2 2 5 6 7
7 7 7 7 7 7 7 7
```

(In our scale of light intensity, 0 represents no light (completely dark), and the light intensity increases or the rectangle becomes brighter for a larger value.)

What you have to do is to complete the program to convert the digitized image to a binary image and display the English letter on the screen. The following steps should be followed.

(i) Write a function to compute the threshold for constructing a binary image based on the contents of digitized image. The function header is given as follows:

```
float computeThreshold (int  image[8][8])
```

This function will first find the maximum (max) and the minimum (min) values of the light intensity from the 64 rectangles and return $\frac{(\max+\min)}{2}$ to its caller.

(ii) Write a function to convert the digitized image to a binary image based on a threshold value. The function header is given as follows.

```
void convertToBinaryImage (int image[8][8], float threshold)
```

This function will change the light intensity for all the rectangles to either 0 (dark) or 1 (bright).

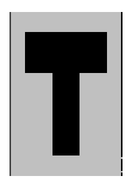
Suggested algorithm :

```
for each row
 for each rectangle in the row
 {
   if the light intensity of this rectangle > threshold
     change that intensity to 1 (bright);
   else
     change that intensity to 0 (dark);
 }
```

(iii) Write a `main` function that declares an int-array of dimension 8x8, and initializes the array by the digitized values. The `main` function will call the first function to compute the threshold for constructing the binary image, and call the second function to convert the digitized image to a binary image based on the threshold value. Finally, the `main` function will display the binary image on the screen by the following suggested algorithm:

```
for each row
{
  print a new line;
  for each rectangle in the row
  {
     if the binary value of this rectangle is 1
       print a rectangle (ASCII code = 219);
    else
       print a space to skip the rectangle;
  }
}
```

On the screen we should see a capital T.



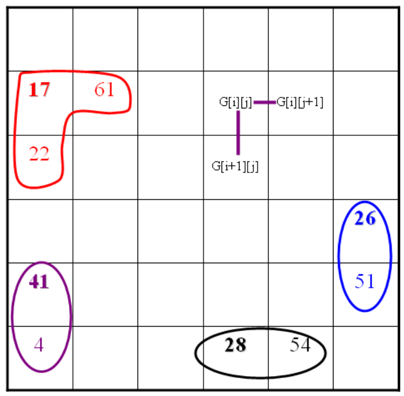
Use debugger whenever in doubt !!

Assignment 2

Given *n* x *n* unsorted data. Let **G** represent a square grid, where G[i][j] refers to the data on the grid location (i, j), $0 \le i < n$, and $0 \le j < n$. The root of grid/sub-grid refers to the left-top corner data on the grid/sub-grid region. The sons of G[i][j] refer to G[i][j+1] and G[i+1][j] if they exist. **G** is said to have heap property if for each G[i][j], G[i][j] $\le$ G[i][j+1] if G[i][j+1] exists, and G[i][j] $\le$ G[i+1][j] if G[i+1][j] exists. Assume n = 6, and **G** is loaded with random data initially.

	j=0	j=1	j=2	j=3	j=4	j=5
i=0	88	31	29	21	1	99
i=1	17	61	16	13	19	50
i=2	22	27	18	71	56	17
i=3	83	66	98	29	97	26
i=4	41	32	76	10	43	51
i=5	4	63	23	28	54	84

Initial Grid Contents with Random Data



The Father-and-Son Relationship in a Square Grid

The heap property in the square grid can be established by a pre-processing procedure where each row in **G** is sorted in ascending order, followed by each column in ascending order.

```
88 31 29 21  1 99
17 61 16 13 19 50
22 27 18 71 56 17
83 66 98 29 97 26
41 32 76 10 43 51
 4 63 23 28 54 84
```

Random Data

```
 1 21 29 31 88 99
13 16 17 19 50 61
17 18 22 27 56 71
26 29 66 83 97 98
10 32 41 43 51 76
 4 23 28 54 63 84
```

After Row Sort

```
 1 16 17 19 50 61
 4 18 22 27 51 71
10 21 28 31 56 76
13 23 29 43 63 84
17 29 41 54 88 98
26 32 66 83 97 99
```

After Column Sort
(Heap property established)

But G is said to be sorted only if G[0][0] ≤ G[0][1] ≤ … ≤ G[0][n-1] ≤ G[1][0] ≤ G[1][1] ≤ … ≤ G[1][n-1] ≤ … < G[i][0] ≤ ….. ≤ G[i][n-1] ≤ G[i+1][0] ≤ ….≤ G[n-1][n-1]. After the pre-processing procedure is performed, the data in the square grid is not sorted yet. The following data interchange and heap restoration procedure is able to arrange the data on **G** in ascending order.

For each data on (i_1, j_1), $0 \le i_1 \le n-2$, $1 \le j_1 \le n-1$, we move the data on **G** to their be-sorted location, which is the grid location the value should be placed, in row-wise order from the left-top corner to the right-bottom corner. To insert a sorted data to (i_1, j_1) we have to compare G[i_1][j_1] and G[i_1+1][0]. Data interchange is required if these two data are not in order, followed by the heap restoration of the affected sub-grid. To restore the heap property, we first restore the heap property of a cell of 3 data, which consists of the new data, denoted by u, at the root of the affected sub-grid and the two sons of u (or one son for boundary condition). Such a cell restoration is performed recursively until no data interchange is required.

1	**16**	17	19	50	61
4	18	22	27	51	71
10	21	28	31	56	76
13	23	29	43	63	84
17	29	41	54	88	98
26	32	66	83	97	99

G[0][1] and G[1][0] are compared.

1	4	17	19	50	61
16	18	22	27	51	71
10	21	28	31	56	76
13	23	29	43	63	84
17	29	41	54	88	98
26	32	66	83	97	99

G[0][1] and G[1][0] are exchanged.

1	4	17	19	50	61
10	18	22	27	51	71
16	21	28	31	56	76
13	23	29	43	63	84
17	29	41	54	88	98
26	32	66	83	97	99

Heap restoration continues …

1	4	17	19	50	61
10	18	22	27	51	71
13	21	28	31	56	76
16	23	29	43	63	84
17	29	41	54	88	98
26	32	66	83	97	99

Heap restoration for G[0][1] is completed as no interchange of data is needed.

Once the data interchange and heap restoration is completed for G[0][1], we have to do the same for G[0][2] and so on.

```
 1   4  17  19  50  61
10  18  22  27  51  71
13  21  28  31  56  76
16  23  29  43  63  84
17  29  41  54  88  98
26  32  66  83  97  99
```

G[0][2] and G[1][0] are compared.

```
 1   4  10  13  16  17
17  18  19  21  22  23
26  27  28  29  29  31
32  41  43  50  51  54
56  61  63  66  71  76
83  84  88  97  98  99
```

Sorted Data

Unzip lab8.zip to complete the program named as **gridsort.c** to sort 64 random numbers in an 8x8 integer array. Check the contents of grid after each procedure is performed. If your program runs correctly, the screen output will be as shown on the right:

```
Raw Data:
   94    3   98  164   92   34  163  174
    2  199   34   82   28  190  141   24
   17  139  151   78    6  183  128  111
   97   49  183  183   90  108   53   76
  163   55   31  165  191   71  115  173
   15   24  155  176   22   86  102   23
  109  188  163   94  163   46  158  117
   90  156   82   34   55  126   20   78
Checksum = 6420

After row sort:
    3   34   92   94   98  163  164  174
    2   24   28   34   82  141  190  199
    6   17   78  111  128  139  151  183
   49   53   76   90   97  108  183  183
   31   55   71  115  163  165  173  191
   15   22   23   24   86  102  155  176
   46   94  109  117  158  163  163  188
   20   34   55   78   82   90  126  156
Checksum = 6420

After column sort:
    2   17   23   24   82   90  126  156
    3   22   28   34   82  102  151  174
    6   24   55   78   86  108  155  176
   15   34   71   90   97  139  163  183
   20   34   76   94   98  141  164  183
   31   53   78  111  128  163  173  188
   46   55   92  115  158  163  183  191
   49   94  109  117  163  165  190  199
Checksum = 6420

After interchange and restoration:
    2    3    6   15   17   20   22   23
   24   24   28   31   34   34   34   46
   49   53   55   55   71   76   78   78
   82   82   86   90   90   92   94   94
   97   98  102  108  109  111  115  117
  126  128  139  141  151  155  156  158
  163  163  163  163  164  165  173  174
  176  183  183  183  188  190  191  199
Checksum = 6420   sorted

Press any key to continue . . . _
```

What is the time complexity of this Square Grid Sort Algorithm to sort ***n*** random data where ***n*** is a square number?

Use debugger whenever in doubt !!