Fall 2024 PF Mid II Solution

Programming Fundamentals(CS1002) Date: 11/4/2024
Course Instructor(s) Dr. Farrukh Shahid, Basit Ali, Farooq Zaidi, Fahad Hussain, Nauraiz Subhan, Kariz Kamal, Kashif, Sumaiya, Bakhtawar, Rafia, Zain Noreen, Iqra Fahad.
Sessional-II Exam
Total Time: 1hr
Total Marks: 30
Total Questions: 03

Q1: Write a C program for the requirements given below. [Marks 10, Est. Time 15 min]
You are given an integer array heights, where heights[i] represents the height of a hill at position i along a trail. You are planning a hiking trip and want to maximize the total height gain. The rules are:
1. You can only "gain" height by moving from a lower position to a higher one.
2. If the height decreases at any point, you stop gaining and start again from the next point.
3. Each time you stop, keep track of the total gain from the previous segment. In the end, select the segment with the highest total gain.
Your task is to find and return the maximum total height gain you can achieve by hiking up the hills on this trail.
Sample Input: Heights = [100, 180, 260, 310, 40, 535, 695]
Sample Output: 655

[Solution]

```c
#include <stdio.h>
int maxHeightGain(int heights[], int size) {
    int maxGain = 0, currentGain = 0;
    for (int i = 1; i < size; i++) {
        // Check if we are moving to a higher position
        if (heights[i] > heights[i - 1]) {
            currentGain += heights[i] - heights[i - 1];
        } else {
            // Compare and update the max gain if this segment is the highest
            if (currentGain > maxGain) {
                maxGain = currentGain;
            }
            // Reset current gain as we hit a lower position
            currentGain = 0;
        }
    }
    // Final check for the last segment
    if (currentGain > maxGain) {
        maxGain = currentGain;
    }
    return maxGain;
}
int main() {
    int n;
    printf("Enter size of array");
    scanf("%d",&n);
    int heights[n];
    for(int i=0;i<n;i++)
      scanf("%d",&heights[i]);
    int result = maxHeightGain(heights, n);
    printf("Maximum Total Height Gain: %d\n", result);
    return 0;
}
```

Fall 2024 PF Mid II Solution

```
Q2: [Marks 10, Est. Time 20 min]
Design a C function named Analysis(int array[10][10]) that takes a 2D array of size
10x10 as an argument. Each
cell in the array represents a type of parking space, defined as follows:
0 = Empty
1 = Occupied
2 = Reserved
3 = Disabled
4 = Maintenance
Requirements: The function should accomplish the following tasks:
1. Identify the row with the highest number of empty spaces.
2. Check whether at least 10% of the parking spaces are designated for disabled parking.
3. Find the largest contiguous block of empty spaces (either horizontally or vertically)
that consists of four
or more spaces.
```

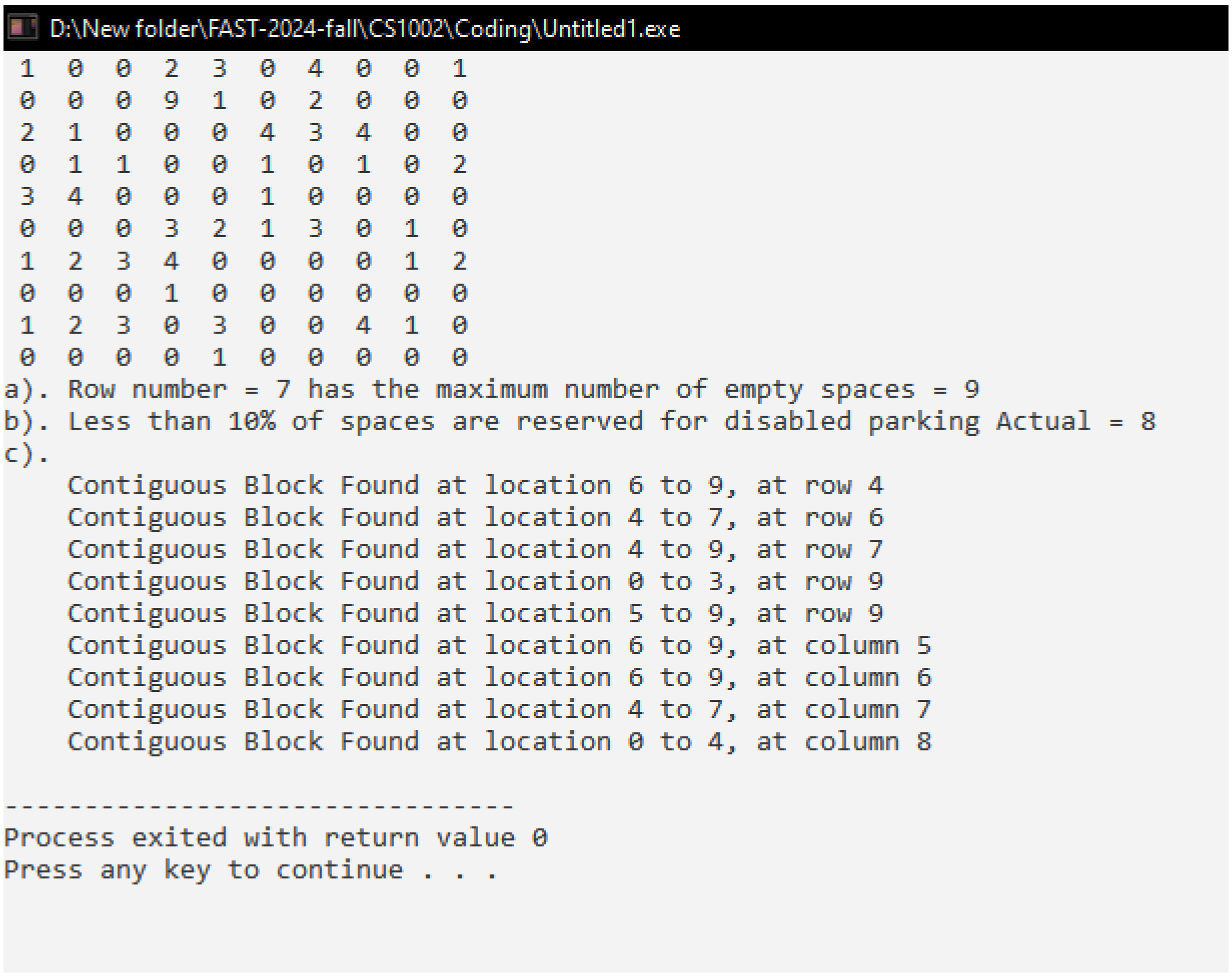
```
[Solved for horizontal (h=1) and vertical (h=0) search of contiguous spaces]
#include<stdio.h>
#include<stdlib.h>

#define n 10
#define prompt "Contiguous Block Found at location"


void analysis(int array[n][n]);
void printArray(int array[n][n]);
void findContigSpaces(int array[n][n], int h);

int main()
{
	int i, length;
	int ar[n][n] = {
	  {1, 0, 0, 2, 3, 0, 4, 0, 0, 1},
	  {0, 0, 0, 9, 1, 0, 2, 0, 0, 0},
	  {2, 1, 0, 0, 0, 4, 3, 4, 0, 0},
	  {0, 1, 1, 0, 0, 1, 0, 1, 0, 2},
	  {3, 4, 0, 0, 0, 1, 0, 0, 0, 0},
	  {0, 0, 0, 3, 2, 1, 3, 0, 1, 0},
	  {1, 2, 3, 4, 0, 0, 0, 0, 1, 2},
	  {0, 0, 0, 1, 0, 0, 0, 0, 0, 0},
	  {1, 2, 3, 0, 3, 0, 0, 4, 1, 0},
	  {0, 0, 0, 0, 1, 0, 0, 0, 0, 0}};

	printArray(ar);
	analysis(ar);
	return 0;
}
void analysis(int array[n][n])
{	int emptySpaces[n] = {0}, rowMaxEmpty = 0, rowMaxIndex, totalDisabled=0;
	int i,length;

	for(i=0; i<n;i++)
	{
		for(length=0; length<n;length++)
		{
			if (array[i][length] == 0)
			{
				++emptySpaces[i];
			}
```

Fall 2024 PF Mid II Solution

```
                    if (array[i][length] == 3)
                    {
                            ++totalDisabled;
                    }
            }
    }
    for(i=0; i<n;i++)
    {
            if(rowMaxEmpty<emptySpaces[i])
            {       rowMaxEmpty = emptySpaces[i];
                    rowMaxIndex = i;
            }
    }
    printf("a). Row number = %d has the maximum number of empty spaces
= %d\n",rowMaxIndex, emptySpaces[rowMaxIndex]);

    if(totalDisabled >=10)
    {
            printf("b). Yes at least 10%% of spaces are reserved for disabled parking
Actual = %d\n",totalDisabled);
    }
    else
    {
            printf("b). Less than 10%% of spaces are reserved for disabled parking
Actual = %d\n",totalDisabled);
    }
    printf("c). \n");

    findContigSpaces(array, 1);

    findContigSpaces(array, 0);
}

void findContigSpaces(int array[n][n], int h) // solved for horizontal search
{
    int i,length, val;
    for(i=0; i<n;i++)
    {
            length = 0;
            int csIndexEnd = -1;
            int csIndexStart = -1;
            // start searching for new rowIndex i
            while(length<n)
            {
                    val = h*(array[i][length])  + (!h)*(array[length][i]);
                    if(val == 0)
                            {
                                    csIndexStart = length;
                                    while(val == 0 && length < n)
                                    {
                                            length++;
                                    val = h*(array[i][length])  + (!h)*(array[length][i]);
                                    }
                                    csIndexEnd = length;
                                    if ((csIndexEnd - csIndexStart) >= 4)
                                    {
                                            (h == 1)? printf("    %s %d to %d, at
row %d\n",prompt, csIndexStart, csIndexEnd-1, i) : printf("    %s %d to %d, at
column %d\n",prompt, csIndexStart, csIndexEnd-1, i);
```

Fall 2024 PF Mid II Solution

```
                                    }
                            }
                    else {
                            length++;
                    }
            }
        }
}
void printArray(int array[n][n]){
        int i,length;
        for(i=0; i<n;i++)
        {for(length=0; length<n;length++)
                {printf("% d ", array[i][length]);}
                printf("\n");}}
```



D:\New folder\FAST-2024-fall\CS1002\Coding\Untitled1.exe

```
 1  0  0  2  3  0  4  0  0  1
 0  0  0  9  1  0  2  0  0  0
 2  1  0  0  0  4  3  4  0  0
 0  1  1  0  0  1  0  1  0  2
 3  4  0  0  0  1  0  0  0  0
 0  0  0  3  2  1  3  0  1  0
 1  2  3  4  0  0  0  0  1  2
 0  0  0  1  0  0  0  0  0  0
 1  2  3  0  3  0  0  4  1  0
 0  0  0  0  1  0  0  0  0  0
a). Row number = 7 has the maximum number of empty spaces = 9
b). Less than 10% of spaces are reserved for disabled parking Actual = 8
c).
    Contiguous Block Found at location 6 to 9, at row 4
    Contiguous Block Found at location 4 to 7, at row 6
    Contiguous Block Found at location 4 to 9, at row 7
    Contiguous Block Found at location 0 to 3, at row 9
    Contiguous Block Found at location 5 to 9, at row 9
    Contiguous Block Found at location 6 to 9, at column 5
    Contiguous Block Found at location 6 to 9, at column 6
    Contiguous Block Found at location 4 to 7, at column 7
    Contiguous Block Found at location 0 to 4, at column 8

--------------------------------
Process exited with return value 0
Press any key to continue . . .
```

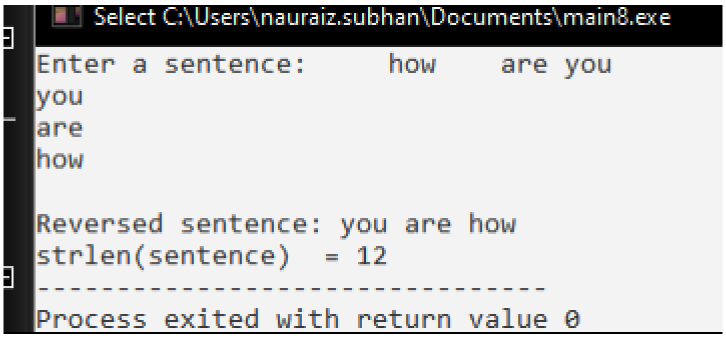
Fall 2024 PF Mid II Solution

```
Q3: [Marks 10, Est. Time 20 min] Complete the following C program that reads a sentence
and reverses the order of the words. Fill in the missing parts of the code where
indicated.
#include <stdio.h>
#include <string.h>
void reverseWords(char sentence[])
{ char reversed[100] = ""; // Array to hold the reversed sentence
char word[20]; // Temporary array to hold each word int length = 0;
// Missing code: Loop through the sentence to extract words and store them in reversed
// copy it back
reversed[strlen(reversed) - 1] = '\0';
// Remove the last space
strcpy(sentence, reversed); } I
nt main() {
char sentence[100];
printf("Enter a sentence: ");
fgets(sentence, sizeof(sentence), stdin);
reverseWords(sentence);
printf("Reversed sentence: %s\n", sentence);
return 0; }
```

[Solution]

```
#include <stdio.h>
#include <string.h>

void reverseWords(char sentence[]) {
char reversed[100] = ""; // Array to hold the reversed sentence
char word[20];
int length = strlen(sentence)-1;
int wordcount = 0;
int i=0;

      while (length>=0)
      {
          while(sentence[length]==' ' && length>=0)
          {
              length--;
          }
          i = length; // WordEnd postion = length
          // Missing code: Loop through the sentence to extract words and store them in
reversed order
          while(sentence[i]!= ' ' && i>=0)
          {
              i--;
          }
          //StartofWord at position  = i - 1
          // copying it backwards
          strncpy(word, &sentence[i + 1], length-i);
          word[length-i] = '\0';
          length = i;
          puts(word);
          strcat(reversed, word);
          strcat(reversed, " ");
      }
      reversed[strlen(reversed) -1] = '\0';
      strcpy(sentence, reversed);
}
int main() {
      char sentence[100];
```

```
        printf("Enter a sentence: ");
        fgets(sentence, sizeof(sentence)-1, stdin);
        sentence[strcspn(sentence, "\n")] = '\0';
        reverseWords(sentence);
        printf("Reversed sentence: %s\n", sentence);
        printf("strlen(sentence)  = %d", strlen(sentence) );
return 0;
}
```



Select C:\Users\nauraiz.subhan\Documents\main8.exe

```
Enter a sentence:     how    are you
you
are
how

Reversed sentence: you are how
strlen(sentence)  = 12
--------------------------------
Process exited with return value 0
```