COMP 132: Advanced Programming Spring 2019
Self-Study Problems 10

Arrays in C

Question 1 (Selection Sort) A selection sort algorithm for a one-dimensional array has the following steps:

a) The smallest value in the array is found.

b) It is swapped with the value in the first position of the array.

c) The above steps are repeated for the rest of the array starting at the second position and advancing each time.

Eventually the entire array is divided into two parts: the sub-array of items already sorted which is built up from left to right and is found at the beginning, and the sub-array of items remaining to be sorted, occupying the remainder of the array. Write a program that sorts an array of 10 integers using this algorithm.

Question 2 (Union of Sets) Use one-dimensional arrays to solve the following problem. Read in two sets of numbers, each having 10 numbers. After reading all values, display all the unique elements in the collection of both sets of numbers. Use the smallest possible array to solve this problem.

Question 3 An n × m two-dimensional matrix can be multiplied by another ×m p m atrix to give a matrix whose elements are the sum of the products of the elements within a row from the first matrix and the associated elements of a column of the second matrix. Both matrices should either be square matrices, or the number of columns of the first matrix should equal the number of rows of the second matrix.

To calculate each element of the resultant matrix, multiply the first element of a given row from the first matrix and the first element of a given column in the second matrix, add that to the product of the second element of the same row and the second element of the same column, and keep doing so until the last elements of the row and column have been multiplied and added to the sum.

Write a program to calculate the product of 2 matrices and store the result in a third matrix.

Question 4 Write an iterative binary search in C. (`sizeof(arr) / sizeof(arr[0])` returns the length of the array, you will see such examples in next class)

Question 5 Write a recursive binary search in C. (`sizeof(arr) / sizeof(arr[0])` returns the length of the array, you will see such examples in next class)

Pointers in C

Example1: For each of the following, write a single statement that performs the specified task. Assume that double precision variables value1 and value2 have been declared and value1 has been initialized to 20.4568.

- **a)** Define the variable dPtr to be a pointer to an object of type double.
- **b)** Assign the address of variable value1 to pointer variable dPtr.

c) Print the value of the object pointed to by dPtr.
d) Assign the value of the object pointed to by dPtr to variable value2.
e) Print the value of value2.
f) Print the address of value1.
g) Print the address stored in dPtr. Is the value printed the same as value1's address?

Example1 Solution:

a) double *dPtr;
b) dPtr = &value1;
c) printf("%f\n", *dPtr);
d) value2 = *dPtr;
e) printf("%f\n", value2);
f) printf("%p\n", &value1);
g) printf("%p\n", dPtr); yes

Question 6 Implement the following functions in C and write a `main` method to test the correctness of your implementations.

a) Implement a function with the following prototype:

```
void swapRows(int d[][COL_SIZE], int i, int j);
```

that swaps the rows i and j. For each k in 0 to COL_SIZE-1, the function should swap d[i][k] with d[j][k]. Define and use a swapElement function for this purpose.

```
void swapElement(int *, int *);
```

b) Implement the following function without using any square brackets [ ] (i.e., without using array subscript notation) to access entries of the array. Instead, **use pointers and pointer arithmetic**.

```
int* findElement(char *arrayPtr, int arraySize, char elementToFind);
```

The findElement function takes a pointer to a character array, the length of the array and a character to search in the array. Traverse a pointer over the array to find the character. If the number is found, save the index of the corresponding entry into the return variable. If the number is not found, return NULL.

c) Implement a function that takes a char array and its size as arguments and converts the array into a palindrome by replacing the second half of the array with the reverse of the first half. A palindrome is a sequence of characters that reads the same backwards and forwards. Use pointers and observe how it eases the job compared to array index arithmetic.

Question 7 Implement a C function that performs a simple matrix multiplication. Prototype of the function is as follows:

```
void Multiply (int m[R_SIZE][C_SIZE], int *cVector, int *rVector);
```

This is rather the simplest of matrix multiplications. You are given a column vector (C_SIZE by 1) and a row vector (1 by R_SIZE) to multiply. Calculate each entry of the resultant matrix and save it in a two-dimensional array m. Remember this is multiplication of a column vector with a row vector, hence the resultant matrix will be of size (R_SIZE by C_SIZE).

Write a main function that tests the `Multiply` function with various column and row vectors.

Question 8 In this question, you are asked to implement a racing game using C arrays and pointers.
Use preprocessor directive to define `TRACK_LENGTH as 10.`
Define global integer pointers (**player1, player2**) and assign them the address of array **track**.
In the main method:

a) Define an integer array named **track** of size `TRACK_LENGTH`.
b) Randomly assign values between 1 and 9 to each element of the array **track**.
c) Set the value of the first element of the **track** to 0.
d) Define integer variables (**score1, score2**) to store players' scores, initialize them to zero.

Implement the following functions.

```
void printTrack(int *track);
void movePlayer(int *score1, int *score2);
void printPlayers(int *track);
```

Function **printTrack** prints the current values of the array **track** horizontally with the format:

```
|0|4|8|1|4|3|2|6|7|4|
```

Function **movePlayer** should perform the following and use pointer operations.
- Pick one of the players randomly and move it to the next step on the **track**.
- Add the value in the destination (element of the array **track**) to the score of the player.
- Modify the value in the destination step to the half of its current value.

Function **printPlayers** displays the current location of the players. The output of this function could be as follows. When player1 is at location 0 and player2 is at location 4.

```
|1| | | |2| | | | | |
```

Initially, when both players are at location 0.

```
|X| | | | | | | | | |
```

In the main method, implement the game that ends when one of the players reaches the end of the track.

A sample execution of the game is provided below.

```
|0|9|8|5|9|2|4|1|8|3|
|X| | | | | | | | | |
Scores: 0/ 0 Press ENTER for next step.
|0|4|8|5|9|2|4|1|8|3|
|2|1| | | | | | | | |
Scores: 9/ 0 Press ENTER for next step.
|0|4|4|5|9|2|4|1|8|3|
|2| |1| | | | | | | |
Scores: 17/ 0 Press ENTER for next step.
|0|4|4|2|9|2|4|1|8|3|
|2| | |1| | | | | | |
Scores: 22/ 0 Press ENTER for next step.
|0|4|4|2|4|2|4|1|8|3|
|2| | | |1| | | | | |
Scores: 31/ 0 Press ENTER for next step.
|0|4|4|2|4|1|4|1|8|3|
|2| | | | |1| | | | |
Scores: 33/ 0 Press ENTER for next step.
|0|4|4|2|4|1|2|1|8|3|
|2| | | | | |1| | | |
Scores: 37/ 0 Press ENTER for next step.
|0|4|4|2|4|1|2|0|8|3|
|2| | | | | | |1| | |
Scores: 38/ 0 Press ENTER for next step.
|0|2|4|2|4|1|2|0|8|3|
| |2| | | | | |1| | |
Scores: 38/ 4 Press ENTER for next step.
|0|2|4|2|4|1|2|0|4|3|
| |2| | | | | | |1| |
Scores: 46/ 4 Press ENTER for next step.
|0|2|2|2|4|1|2|0|4|3|
| | |2| | | | | |1| |
Scores: 46/ 8 Press ENTER for next step.
```

```
|0|2|2|2|4|1|2|0|4|1|
| | |2| | | | | |1|
Scores: 49/ 8 Press ENTER for next step.
|0|2|2|1|4|1|2|0|4|1|
| | | |2| | | | |1|
Scores: 49/10 Press ENTER for next step.
|0|2|2|1|2|1|2|0|4|1|
| | | | |2| | | |1|
Scores: 49/14 Press ENTER for next step.
Player1 wins!
Score of player1: 49
Score of player2: 14
```