Arrays and Pointers Assignment

Mandatory

1. What does the code below refer to? Extend the code and demonstrate the use of ptr to access the contents of a 2D array.

   int (*ptr)[4];

[Refer the sample code in “array_ptr_simple.c”]

user60@trainux01: ~/Batch17OCT2024_175/Assignments/Day06/Arrays_and_Pointers_Assignment

```
#include <stdio.h>
int main() {
    int arr[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};
    int (*ptr)[4];
    ptr = arr;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 4; j++) {
            printf("ptr[%d][%d] = %d\n", i, j, ptr[i][j]);
        }
    }

    return 0;
}
```

```
user72@trainux01:~/splint$ vi aar.c
user72@trainux01:~/splint$ gcc aar.c
user72@trainux01:~/splint$ ./a.out
ptr[0][0] = 1
ptr[0][1] = 2
ptr[0][2] = 3
ptr[0][3] = 4
ptr[1][0] = 5
ptr[1][1] = 6
ptr[1][2] = 7
ptr[1][3] = 8
ptr[2][0] = 9
ptr[2][1] = 10
ptr[2][2] = 11
ptr[2][3] = 12
```

2. Refer the code snippet below. Implement the function search_insert() as mentioned in the code.

```
#define MAX 80

//search for the given char and if found, then create space for 1 character and insert ‘_’ after the searched character. Let the remaining characters in the input be placed after ‘_’.
int search_insert(char name[], char search_char);
```

```
int main()
{
        char name[MAX]=”ABC”;
        char *ptr = name;
        int ret = search_insert(name, search_char);

       if (ret == SUCCESS)
        {
                //display updated string
        }
}
```

user60@trainux01: ~/Batch17OCT2024_175/Assignments/Day06/Arrays_and_Pointers_Assignment

```c
#include <stdio.h>
#include <string.h>
#define MAX 80
#define SUCCESS 1
#define FAILURE 0
int search_insert(char name[], char search_char) {
    int len = strlen(name);
    for (int i = 0; i < len; i++) {
        if (name[i] == search_char)
            for (int j = len; j >= i; j--) {
                name[j + 1] = name[j];
            }
            name[i + 1] = '_';
            return SUCCESS;
        }
    }
    return FAILURE;
}
int main() {
    char name[MAX] = "ABC";
    char search_char = 'B';
    int ret = search_insert(name, search_char);
    if (ret == SUCCESS) {
        printf("Updated string: %s\n", name);
    } else {
        printf("Character not found.\n");
    }
    return 0;
}
```

```
user72@trainux01:~/splint$ vi arp.c
user72@trainux01:~/splint$ gcc arp.c
user72@trainux01:~/splint$ ./a.out
Updated string: A_C
```

3. Refer the program “array_ptr_repr_partial.c”. Implement the functions below which are yet to be implemented in code.

   int func1(int (*ptr)[3]); // pointer to array, second dimension is explicitly specified

   int func2(int **ptr); // double pointer, using an auxiliary array of pointers

user60@trainux01: ~/Batch17OCT2024_175/Assignments/Day06/Arrays_and_Pointers_Assignment

```c
#include <stdio.h>
int func1(int (*ptr)[3]) {
    for (int i = 0; i < 3; i++) {
        printf("%d ", ptr[0][i]);
    }
    return 0;
}
int func2(int **ptr) {
    for (int i = 0; i < 3; i++) {
        printf("%d ", ptr[i][0]);
    }
    return 0;
}
int main() {
    int arr[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    int *pArr[3] = {arr[0], arr[1], arr[2]};
    func1(arr);
    printf("\n");
    func2(pArr);
    return 0;
}
```

```
user72@trainux01:~/splint$ vi point.c
user72@trainux01:~/splint$ gcc point.c
user72@trainux01:~/splint$ ./a.out
1 2 3
1 4 7
```

4. Refer the program “array_dbl_pointers_function_partial.c”. Implement the missing functionality in the code marked with TBD1, TBD2…..

user60@trainux01: ~/Batch17OCT2024_175/Assignments/Day06/Arrays_and_Pointers_Assignment

```c
#include <stdio.h>
#include <stdlib.h>

void func1(short mat[][3]);
void func2(short (*ptr)[3]);
void func3(short *mat);
void func4(short **ptr);
void func5(short *ptr[3]);

/*******************************************************************************
*
* FUNCTION NAME: main
*
* DESCRIPTION: Main function for 2D array - function program
*
* RETURNS: Nothing
*******************************************************************************/
int main()
{
    short mat[3][3], i, j;
    for (i = 0; i < 3; i++)
        for (j = 0; j < 3; j++)
        {
            mat[i][j] = i * 10 + j;
        }

    printf("Initialized data to: ");
    for (i = 0; i < 3; i++)
    {
        printf("\n");
        for (j = 0; j < 3; j++)
        {
            printf("%5.2d", mat[i][j]);
        }
    }
    printf("\n");
    func1(mat);
    func2(mat);
    func3((short *)mat);
    func4((short **)mat);
    func5((short *)mat);

    return 0;
}

/*
Method #1 (No tricks, just an array with empty first dimension)
================================================================
```

user60@trainux01: ~/Batch17OCT2024_175/Assignments/Day06/Arrays_and_Pointers_Assignment

```
51 void func1(short mat[][3])
52 {
53     register short i, j;
54 
55     printf("Declare as matrix, explicitly specify second dimension:\n");
56     for (i = 0; i < 3; i++)
57     {
58         for (j = 0; j < 3; j++)
59         {
60             printf("%5.2d", mat[i][j]);
61         }
62         printf("\n");
63     }
64     printf("\n");
65 }
66 
67 /*
68 Method #2 (Pointer to array, second dimension is explicitly specified)
69 ======================================================================
70 */
71 void func2(short (*mat)[3])
72 {
73     register short i, j;
74 
75     printf("Declare as pointer to column, explicitly specify 2nd dim:\n");
76     for (i = 0; i < 3; i++)
77     {
78         for (j = 0; j < 3; j++)
79         {
80             printf("%5.2d", mat[i][j]);
81         }
82         printf("\n");
83     }
84     printf("\n");
85 }
86 
87 /*
88 Method #3 (Using a single pointer, the array is "flattened")
89 ============================================================
90 With this method you can create general-purpose routines.
91 The dimensions don't appear in any declaration, so you can add them to the formal argument list.
92 */
93 void func3(short *mat)
94 {
95     register short i, j;
96 
97     printf("Declare as single-pointer, manual offset computation:\n");
98     for (i = 0; i < 3; i++)
```

user60@trainux01: ~/Batch17OCT2024_175/Assignments/Day06/Arrays_and_Pointers_Assignment

```
 99      {
100          for (j = 0; j < 3; j++)
101          {
102              printf("%5.2d", *(mat + i * 3 + j));
103          }
104          printf("\n");
105      }
106      printf("\n");
107  }
108
109  /*
110  Method #4 (Double pointer, using an auxiliary array of pointers)
111  ================================================================
112  With this method, you can create general-purpose routines if you allocate "index" at runtime.
113  */
114  void func4(short **mat)
115  {
116      short i, j, *index[3];
117      for (i = 0; i < 3; i++)
118      {
119          index[i] = mat[i];
120      }
121
122      printf("Declare as double-pointer, use auxiliary pointer array:\n");
123      for (i = 0; i < 3; i++)
124      {
125          for (j = 0; j < 3; j++)
126          {
127              printf("%5.2d", *(index[i] + j));
128          }
129          printf("\n");
130      }
131      printf("\n");
132  }
133
134  /*
135  Method #5 (Single pointer, using an auxiliary array of pointers)
136  ================================================================
137  */
138  void func5(short *mat[3])
139  {
140      short i, j, *index[3];
141      for (i = 0; i < 3; i++)
142      {
143          index[i] = mat[i];
144      }
145
146      printf("Declare as single-pointer, use auxiliary pointer array:\n");
```

```
147      for (i = 0; i < 3; i++)
148      {
149          for (j = 0; j < 3; j++)
150          {
151              printf("%5.2d", *(index[i] + j));
152          }
153          printf("\n");
154      }
155      printf("\n");
156  }
157
```

5. Refer the program "pointer_example.c". Fix the warning issue.
6. Consider an array of strings as below.

   char arr[][10]={"Word", "Excel", "PowerPoint", "Pdf", "Paint"};

a. Implement a function read_displaystring() to read a row index from the user, access the string, store in a char * variable and using this, traverse every alternate character in the string and display in console.

```
void read_displaystring(char *arr[][10], int row);
```

b. Reverse the string read at the index in a) using a function of prototype as below. Caller to read the returned string and display the reversed string. [Ensure that the input source array is not corrupted and remaining elements are intact]

```
char *reverse(char *arr[][10], int row);
```