Operating Systems

A (Soft) Introduction to C, Part IV

Vassilis Markos, Mediterranean College

Week 07

Contents

2D Arrays

(Reminder) Arrays In C

Can you guess what the following will print?

```c
// source/arrays_001.c
#include <stdio.h>

int main() {
    int arr[3];
    arr[0] = 4;
    arr[1] = 6;
    arr[2] = -5;
    for (int i = 0; i < 3; i++) {
        printf("arr[%d] == %d\n", i, arr[i]);
    }
}
```

(Reminder) Array Initialisation

We can also provide array elements all at once, as follows:

```c
// source/arrays_002.c
#include <stdio.h>

int main() {
    int arr1[5] = { 4, -2, 0, 4, 6 };
    int arr2[] = { 6, 5, 7, 9 };
    printf("arr1[3] == %d\narr2[1] == %d\n", arr1[3], arr2[1])
    ;
}
```

(Reminder) Dynamic Initialisation

We can also initialise the values of an array based on others' input (e.g., users, another process):

```c
// source/arrays_003.c
#include <stdio.h>

int main() {
    char arr[3];
    for (int i = 0; i < 3; i++) {
        printf("Please, enter a character: ");
        scanf(" %c", &arr[i]);
    }
    printf("%c%c%c\n", arr[0], arr[1], arr[2]);
}
```

(Reminder) Passing Arrays To Functions

What will this print?

```c
// source/arrays_011.c
#include <stdio.h>

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

int main() {
    int arr[] = {2, 6, 0, 1, 4};
    printArray(arr, 5);
}
```

(Reminder) Passing Arrays To Functions

What will this print?

```c
// source/arrays_012.c
#include <stdio.h>

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

int main() {
    int arr[] = {2, 6, 0, 1, 4};
    printArray(arr, 5);
}
```

(Reminder) Passing Arrays To Functions

What will this print?

```c
// source/arrays_013.c
#include <stdio.h>

void foo(int* arr, int length) {
    for (int i = 0; i < length; i++) {
        if (*(arr + i) == 0) {
            *(arr + i) = 4;
        }
    }
}

int main() {
    int arr[] = {2, 6, 0, 1, 4};
    printf("%d\n", arr[2]);
    foo(arr, 5);
    printf("%d\n", arr[2]);
}
```

(Reminder) Array Decay

- A common C catch–phrase is that “arrays decay into pointers”.
- This simply means that, whenever required, arrays are interpreted as pointers, as we have already discussed above.
- As a consequence, when passing an array to a function, we are actually passing a pointer.
- This means that an array is always **passed by reference**. So, in case we need to pass an array be value, we have to devise various tricks we shall see in upcoming lectures.

Declaring 2D Arrays

```c
// source/2d_arrays_001.c
#include <stdio.h>

int main() {
    int arr[2][3]; // Declare "rows" and "columns"
    arr[0][0] = 2; // Initialise each value separately.
    arr[0][1] = 5;
    arr[0][2] = -3;
    arr[1][0] = 0;
    arr[1][1] = 6;
    arr[1][2] = 7;

    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
}
```

Declaring 2D Arrays

```c
// source/2d_arrays_002.c
#include <stdio.h>

int main() {
    // Declare array at initialisation.
    int arr[2][3] = { { 2, 0, -3 }, { 4, 6, 7 } };

    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
}
```

Declaring 2D Arrays

```c
// source/2d_arrays_003.c
#include <stdio.h>

int main() {
    // Let the compiler make the splits.
    int arr[2][3] = { 2, 0, -3, 4, 6, 7 };

    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
}
```

What Will This Print?

```c
// source/2d_arrays_004.c
#include <stdio.h>

int main() {
    // Let the compiler make the splits.
    int arr[][] = { 2, 0, -3, 4, 6, 7 };

    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
}
```

2D Array Dimension Declaration

Hopefully, you see something like the following in your console:

```
2d_arrays_004.c:7:9: error: declaration of ‘arr’ as multidim
ensional array must have bounds for all dimensions except the
first
7 |     int arr[][] = { 2, 0, -3, 4, 6, 7 };
  |         ^~~
```

This is because, as the error says, when it comes to multidimensional arrays, **all but the first dimensions must be provided!**

How About This?

```c
// source/2d_arrays_005.c
#include <stdio.h>

int main() {
    // Let the compiler make the splits.
    int arr[][3] = { 2, 0, -3, 4, 6, 7 };

    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
}
```

2D Arrays And Functions

The same holds true when declaring arrays as function parameters:

```c
// source/2d_arrays_007.c
#include <stdio.h>

int foo(int arr[][3], int rows, int cols) {
    return arr[rows - 1][cols - 1];
}

int main() {
    int arr[][3] = { 2, 0, -3, 4, 6, 7 };
    int x = foo(arr, 2, 3);
    printf("%d\n", x);
}
```

What Will This Print?

```c
// source/2d_arrays_006.c
#include <stdio.h>

int main() {
    // Uncomment exactly 1 of the following 3 lines
    int arr[][3] = { 2, 0, -3, 4, 6, 7 };
    // int arr[][2] = { 2, 0, -3, 4, 6, 7 };
    // int arr[][6] = { 2, 0, -3, 4, 6, 7 };
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
}
```

2D Arrays Do Not Exist!

The previous code snippet was not expected to work, but it does for a single reason:

- 2D arrays **do not exist.**
- Indeed, what C does is to flatten the contents of a 2D array to consecutive memory locations.
- Thus, the `arr[i][j]` syntax does not actually mean “access the `arr` element at row `i` and column `j`”.
- But, then, how does C interpret `arr[i][j]`?

Rows And Columns

Using the following piece of code, can you figure what the C is doing behind the scenes when it comes to `arr[i][j]`?

```c
// source/2d_arrays_006.c
#include <stdio.h>

int main() {
    // Uncomment exactly 1 of the following 3 lines
    int arr[][3] = { 2, 0, -3, 4, 6, 7 };
    // int arr[][2] = { 2, 0, -3, 4, 6, 7 };
    // int arr[][6] = { 2, 0, -3, 4, 6, 7 };
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
}
```

2D Array Flattening

C flattens arrays as follows:

- All elements are put in memory first according to their row and then based on their column.
- So, essentially, each element could be the element of a one–dimensional array at index k, as shown next.
- Compute k as a function of row number i and column number j?

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29

2D Array Flattening

A flattened array's position index k is related to a 2D array's i, j indices by:

2D Array Flattening

A flattened array's position index k is related to a 2D array's i, j indices by:

$$k = i \cdot \#\text{columns} + j.$$

2D Array Flattening

A flattened array's position index k is related to a 2D array's i, j indices by:

$$k = i \cdot \#\text{columns} + j.$$

So, for instance, for a 2D array with 7 columns, the element at row 3 and column 4 (0–based indexing) should be placed at:

$$k = 3 \cdot 7 + 4 = 21 + 4 = 25,$$

i.e., at the 26$^{\text{th}}$ position.

2D Array Flattening

A flattened array's position index k is related to a 2D array's i, j indices by:

$$k = i \cdot \#\text{columns} + j.$$

So, for instance, for a 2D array with 7 columns, the element at row 3 and column 4 (0–based indexing) should be placed at:

$$k = 3 \cdot 7 + 4 = 21 + 4 = 25,$$

i.e., at the 26$^{\text{th}}$ position.
Can you see why we are allowed to omit (only) the first dimension in 2D array declaration?

2D Arrays Tips And Tricks

- 2D arrays are mostly used to make things conceptually easier for us (humans). They do not actually exist.
- So, use them whenever you need to make things easier to you.
- But, in general, this comes at a cost regarding memory de–allocation, as we shall see in the future, so be careful whenever you use 2D arrays!
- Alternatively, you can always use flattened 2D arrays, which should offload some memory management worries from you.
- Also, remember that you are allowed to not provide **only the first** dimension of a multidimensional array!

Strings Are Arrays

What will this print?

```c
// source/strings_001.c
#include <stdio.h>

int main() {
    char msg[] = { 'H', 'i', '!', '\0' };
    printf("%s\n", msg);
}
```

Strings Are Arrays

What will this print?

```c
// source/strings_001.c
#include <stdio.h>

int main() {
    char msg[] = { 'H', 'i', '!', '\0' };
    printf("%s\n", msg);
}
```

- This should print `Hi!`.

Strings Are Arrays

What will this print?

```c
// source/strings_001.c
#include <stdio.h>

int main() {
    char msg[] = { 'H', 'i', '!', '\0' };
    printf("%s\n", msg);
}
```

- This should print `Hi!`.
- The `\0` at the end of the array is a special character, the `NULL` character which indicates the end of the string.

Using Double Quotes

Strings can also be initialised using double quotes, in which case the compiler adds the `NULL` character, so we do not need to insert it manually.

```c
// source/strings_002.c
#include <stdio.h>

int main() {
    char msg[] = "Hi!";
    printf("%s\n", msg);
}
```

String Libraries

Since strings are arrays, we can manipulate them the same way we would with every other array, however we can also make use of the following libraries:

- `ctype`: character handling.
- `stdio`: input / output.
- `stdlib`: general utilities, some of them string–relevant.
- `string`: string manipulation.

String Cleanup

What will this print?

```c
// source/strings_003.c
#include <stdio.h>
#include <ctype.h>

int main() {
    char str[] = "t6H0I9s6.iS.999a9.STRING";
    char c = str[0];
    for(int i = 0; c != '\0'; c = str[++i]) {
        if(isalpha(c))
            printf("%c", (char)(isupper(c) ? tolower(c) : c));
        else if(ispunct(c))
            printf(" ");
    }
    printf("\n");
}
```

String Operations

What will this print?

```c
// source/strings_004.c
#include <stdio.h>
#include <string.h>

int main() {
  char fragment1[] = "I'm a s";
  char fragment2[] = "tring!";
  char fragment3[20];
  char finalString[20] = "";

  strcpy(fragment3, fragment1);
  strcat(finalString, fragment3);
  strcat(finalString, fragment2);
  printf("%s\n", finalString);
}
```

String Tips And Tricks

- When looping over strings, using the `NULL` character is a nice universal way to determine when all the string has been consumed. Thus, we need not pass string length as a parameter in string manipulation functions.
- Since each string contains the `NULL` character, all strings are by default non–empty!
- Remember that `char`s are declared using single quotes (’), while strings using double (”).
- Some string manipulation functions return strings while we would need a single `char`. In this case, we have to **cast the output** to a `char` before using it!

Fun Time!

In–class Exercise #001

Implement the following functions in C:

1. A function, `add()`, that takes as arguments two 3×3 `double` arrays and returns their sum.
2. A function, `transpose()`, that takes as argument a single 3×3 `double` array, `arr`, and returns its transpose, i.e., a 3×3 array whose rows are the columns of `arr`.
3. A function, `diag()`, that takes a 3×3 double array and computes and return the sum of its diagonal elements.

In–class Exercise #002

Implement a C function that:

- takes a one dimensional `int` array of length 25, and;
- prints the array's elements in a spiral order, i.e., starts from top left and, moving first right, then down, then left and then up, prints elements spiral–wise.

In–class Exercise #003

Implement tic–tac–toe in C as follows:

- Create a 3×3 matrix to represent the game board.
- Implement functions to make moves, check for wins, and check for draws.
- Play the game interactively.

In–class Exercise #004

The dot product of two vectors is computed as follows:

$$(x_1, x_2, x_3) \cdot (y_1, y_2, y_3) = x_1 y_1 + x_2 y_2 + x_3 y_3.$$

Write a C function that takes two 3 dimensional vectors as arguments and computes and returns their dot product.

In–class Exercise #005

We can multiply two square 2×2 matrices as shown below:

$$\begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix} = \begin{pmatrix} a_1 b_1 + a_2 b_3 & a_1 b_2 + a_2 b_4 \\ a_3 b_1 + a_4 b_3 & a_3 b_2 + a_4 b_4 \end{pmatrix}.$$

Write a C function that takes two 2×2 double arrays and computes their product.

In–class Exercise #006

The determinant of a 2×2 matrix is given by the following formula:

$$\det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc.$$

Write a C function that computes the determinant of a 2×2 double array.

In–class Exercise #007

The determinant of a 3×3 matrix is given by the following formula:

$$\det \begin{pmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{pmatrix} = a_1 \det \begin{pmatrix} a_5 & a_6 \\ a_8 & a_9 \end{pmatrix} - a_2 \det \begin{pmatrix} a_1 & a_3 \\ a_7 & a_9 \end{pmatrix} + a_3 \det \begin{pmatrix} a_4 & a_5 \\ a_7 & a_8 \end{pmatrix}.$$

Write a C function that computes the determinant of a 3×3 double array.

In–class Exercise #008: Part A

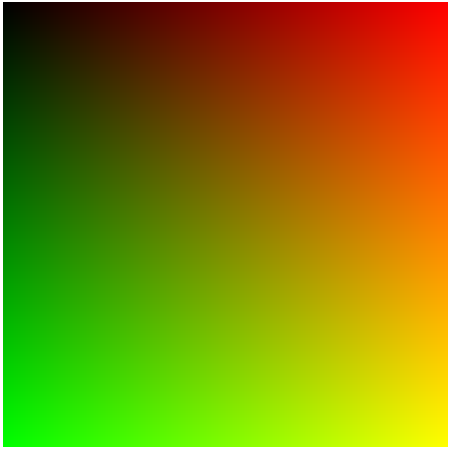
This is a three part self–study exercise. At first, read the following Wikipedia paragraph about how PPM image files are structured:

`https://en.wikipedia.org/wiki/Netpbm#PPM_example`

Then implement a C function that creates a 256×256 `.ppm` red image file.
Regarding C and file handling, either recall your C knowledge or look around the web!

In–class Exercise #008: Part B

As your first actual exercise with PPM images, try to generate an image like the one shown right. To do so, you might find useful to recall how red and green are represented in RGB.



In–class Exercise #008: Part C

Being sufficiently exposed to PPM images and C, you now have to implement the following C functions:

- a function, `flipX()` that flips an image across the horizontal axis;
- a function, `flipY()` that flips an image across the vertical axis;
- a function, `grayscale()` that turns an image into grayscale.

You can use the image generated in part B to test your functions.

In–class Exercise #009

Create the following functions in C:

1. An int function, `substrSearch()` that takes two strings, `needle` and `haystack` and looks for the first occurrence of `needle` in `haystack` and returns its starting index.
2. A boolean function `isPalindrome()` that takes a string and checks whether it is a palindrome, i.e., whether it reads the same right–to–left and left–to–right.
3. A boolean function `isAnagram()` that takes two strings, `ana` and `gram` and checks if `ana` is an anagram of `gram`, ignoring case and spaces.

In–class Exercise #010

Implement a C function `strComp()` that:

- takes as input a string, `str`;
- checks that it contains only characters in the range `a-z` or `A-Z`;
- spots any characters that repeat at consecutive positions, and;
- returns a string with consecutively occurring characters replaced by a single instance of that character followed by an integer indicating the number of repetitions.

For instance, for input `aaaabcccd` it should return `a4bc3d`.

In–class Exercise #011

Read about the Knuth–Morris–Pratt substring search algorithm:

`https://en.wikipedia.org/wiki/Knuth%E2%80%93Morris%E2%80%93Pratt_algorithm`

Then implement it in C with an appropriate function and any other required machinery.

Homework

1. Complete any in–class exercises you haven't so far.
2. Since this course's aim is to study socket programming in C, this homework is mostly oriented towards that direction, provided we have studied enough C so far. To get yourselves comfortable with sockets in C, study the tutorial found below:

   `https://beej.us/guide/bgnet/html/`

Share your comments and implementations at: `v.markos@mc-class.gr`

Any Questions?

Do not forget to fill in the questionnaire shown right!

https://forms.gle/dKSrmE1VRVWqxBGZA