C Programming

Arrays

Symbolic Names

In C, a symbolic name is used to avoid hard-coding values in a program.

A symbolic name looks like this:

#define symbolic_name data_value

e.g.,

#define SIZE 5

We normally use all uppercase characters for the names of symbolic names in order to visually differentiate them from regular variables in the program.

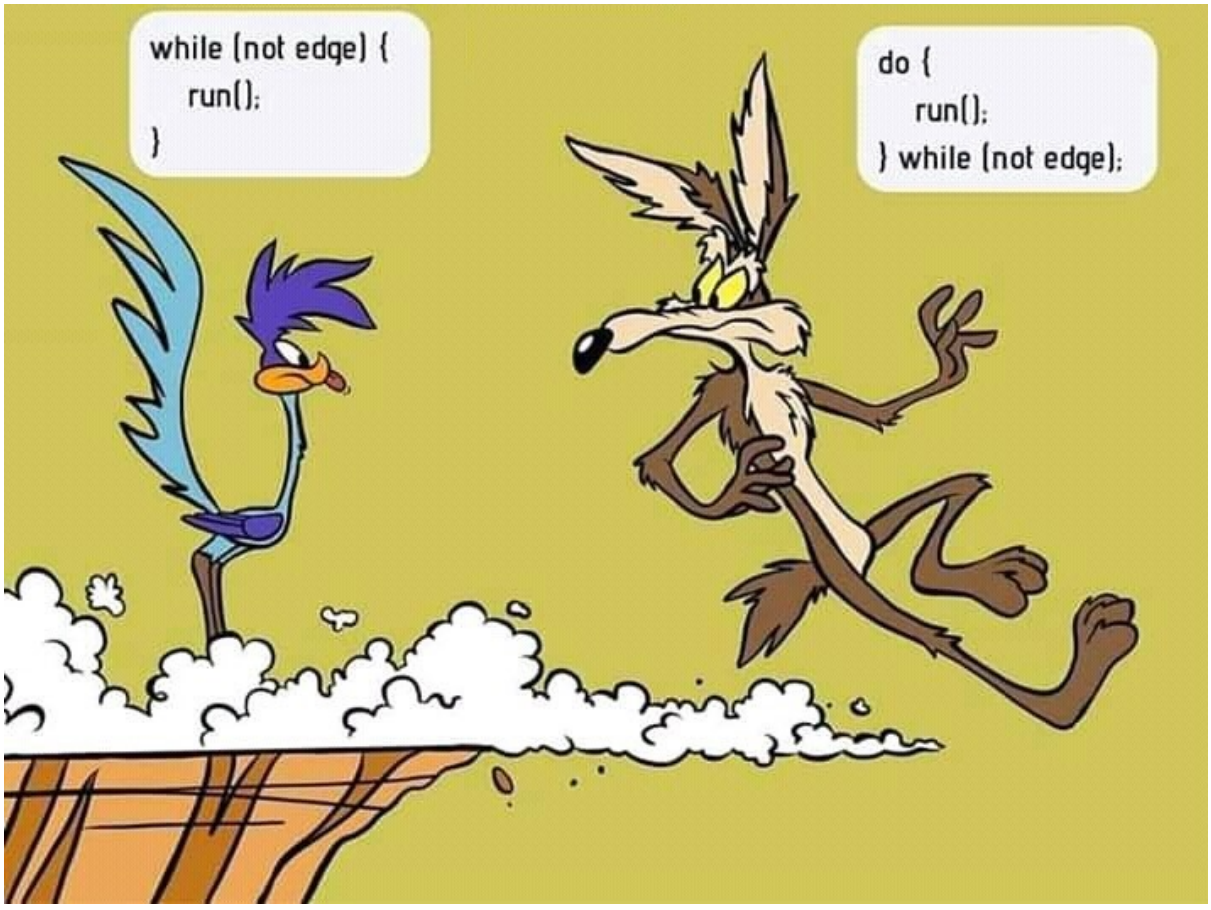
Let's substitute the above symbolic name SIZE into our previous program:

```
/*
Program that uses an array and calculates the average age of a set of
people

*/
#include <stdio.h>

// Symbolic name(s) are always listed below the last #include file
#define SIZE 5

int main()
{
	int ages[SIZE];
	int sum = 0;
	float average = 0;
	int i;
	int highest = 0;
	int lowest = 0;

	printf("Enter the ages of %d people\n", SIZE);
```

```
	// this loop will be used to enter an age into each element of
the array
	for(i = 0; i < SIZE; i++)
	{
		// enter an age
		scanf("%d", &ages[i]);

		// add the current value in sum to the age entered, i.e.,
keep a running sum total of ages
		sum = sum + ages[i];

	} // end for

	// Let's find the highest and lowest age entered
	// Assume the highest number in the array ages is inside element
	highest = ages[0];

	// Assume the lowest number in the array ages is inside element 0
	lowest = ages[0];

	/* this loop is used to go through each number in the array and
check if it is higher than the variable highest and lower than the
variable lower. If so, it replaces the values in highest and lowest
	*/
	for(i = 0; i < SIZE; i++)
	{
		// find the highest number in the array
		if(highest <  ages[i])
		{
			highest = ages[i];
		} // end if

		// find the lowest number in the array
		if(lowest > ages[i])
		{
			lowest = ages[i];
		} // end if
```

```
        } // end for

        // Display the highest and lowest number in the array
        printf("\n\nThe highest age is %d", highest);
        printf("\n\nThe lowest age is %d", lowest);

        // cast the variable sum to be a float just for this line of code
        average = (float)sum / SIZE;

        printf("\n\nThe average age is %.1f", average);

        return 0;

} // end main
```

Flashback: Comparison between a while loop and a do..while loop
(a do .. while loop guarantees to execute the code at least once, a regular while loop will not guarantee code execution, i.e., the condition may be false from the start)



Initialising an array

The contents of an array can be initialised just like any other standard variable,
i.e., int num = 0;

In an array, the contents of each element can also be initialised with data before the array is used in the program.

Have a look at the following code example, which initialises an array of size 12, i.e., there are 12 elements in the array. Each element in the array is initialised with a whole number before the array is used in the program.

```
/*
Program to show how to initialise an array

Author:
Date:

*/

#include <stdio.h>

#define NO_OF_MONTHS 12

int main()
{
	int days[NO_OF_MONTHS] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31,
30, 31};
	//int days[NO_OF_MONTHS];
	int month = 0;
	int i;

	for(i = 0; i < NO_OF_MONTHS; i++)
	{
		printf("%d ", days[i]);
	}

	printf("\n\nPlease enter a month, e.g., 1 = Jan, 2 = Feb, etc.,
\n");
```

```
	// user enters a number corresponding to the month they wish to
view the no. of days
	do
	{
		scanf("%d", &month);

		// check if valid month entered, i.e., 1 - 12 inclusive
		if (month < 1 || month > 12)
		{
			printf("\nInvalid month entered\n");
		} // end if

	} // end do
	while(month < 1 || month > 12);

	printf("\n\nThe number of days in month %d are %d", month,
days[month - 1]);

	return 0;

} // end main()
```

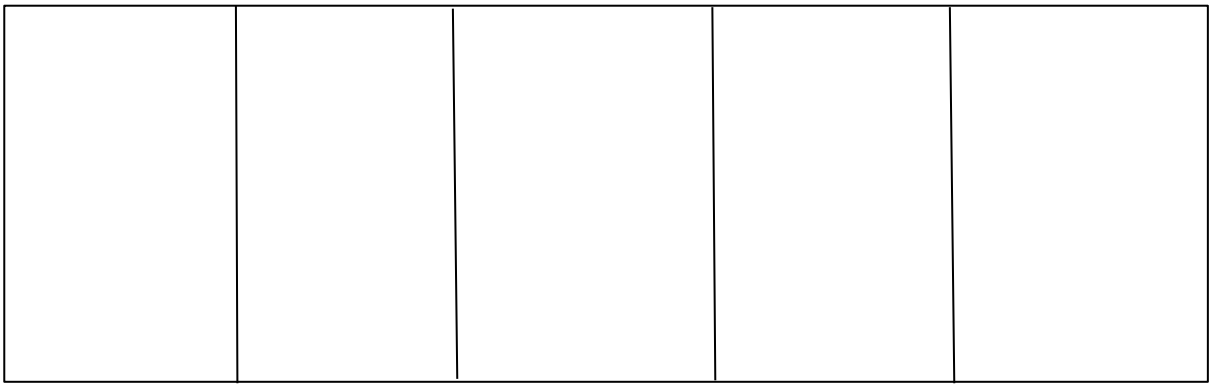
Repl 7.1: https://replit.com/@michaelTUDublin/71-Initialising-an-array#main.c

Multi-Dimensional Arrays

To date, we have been looking at what are called 1-Dimensional (1-D) arrays. For example,

int numbers[5];

numbers



numbers is a 1-Dimensional array

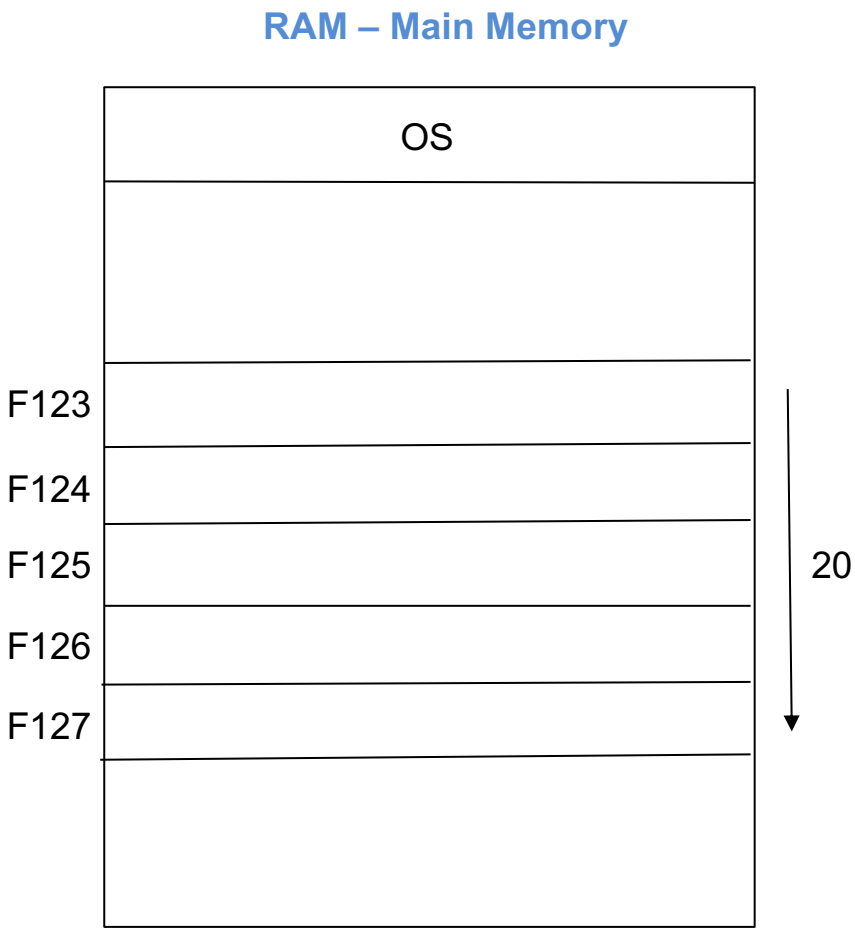
Arrays can also be created that have more than 1 dimension, hence called multi-dimensional arrays. We will now look at this.

Before looking at a multi-dimensional array, let's have a look at how a 1-D array is stored in memory

int numbers[5];

int numbers[5];

the array is placed in a **CONTIGUOUS** memory block in RAM, i.e., each array element is located after the previous element



2-Dimensional (2-D) arrays

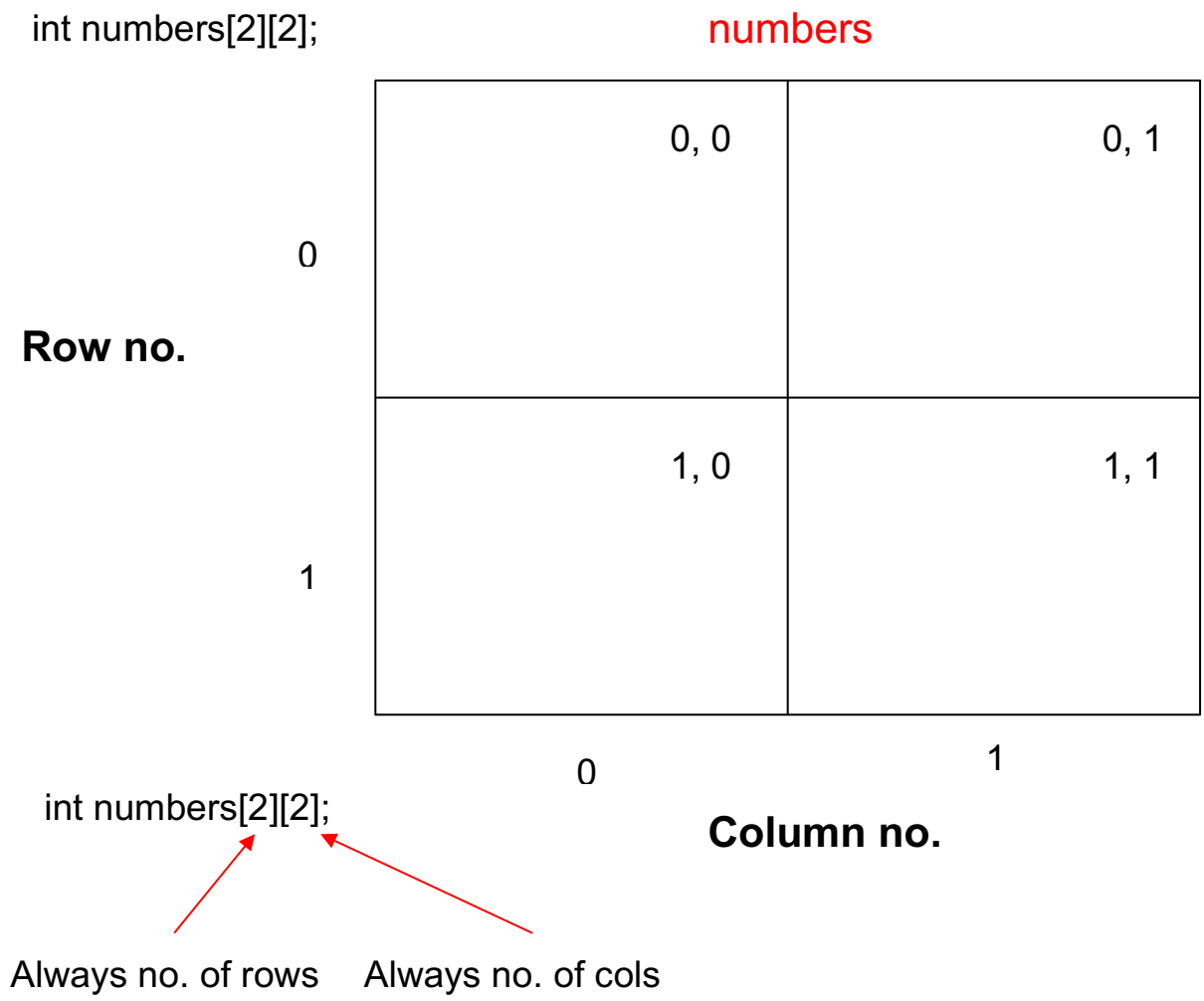
A 2-D array can be visualised as a hash-table / grid / etc.,

You create a 2-D as follows:

array_data_type array_name [no_of_rows] [no_of_columns];

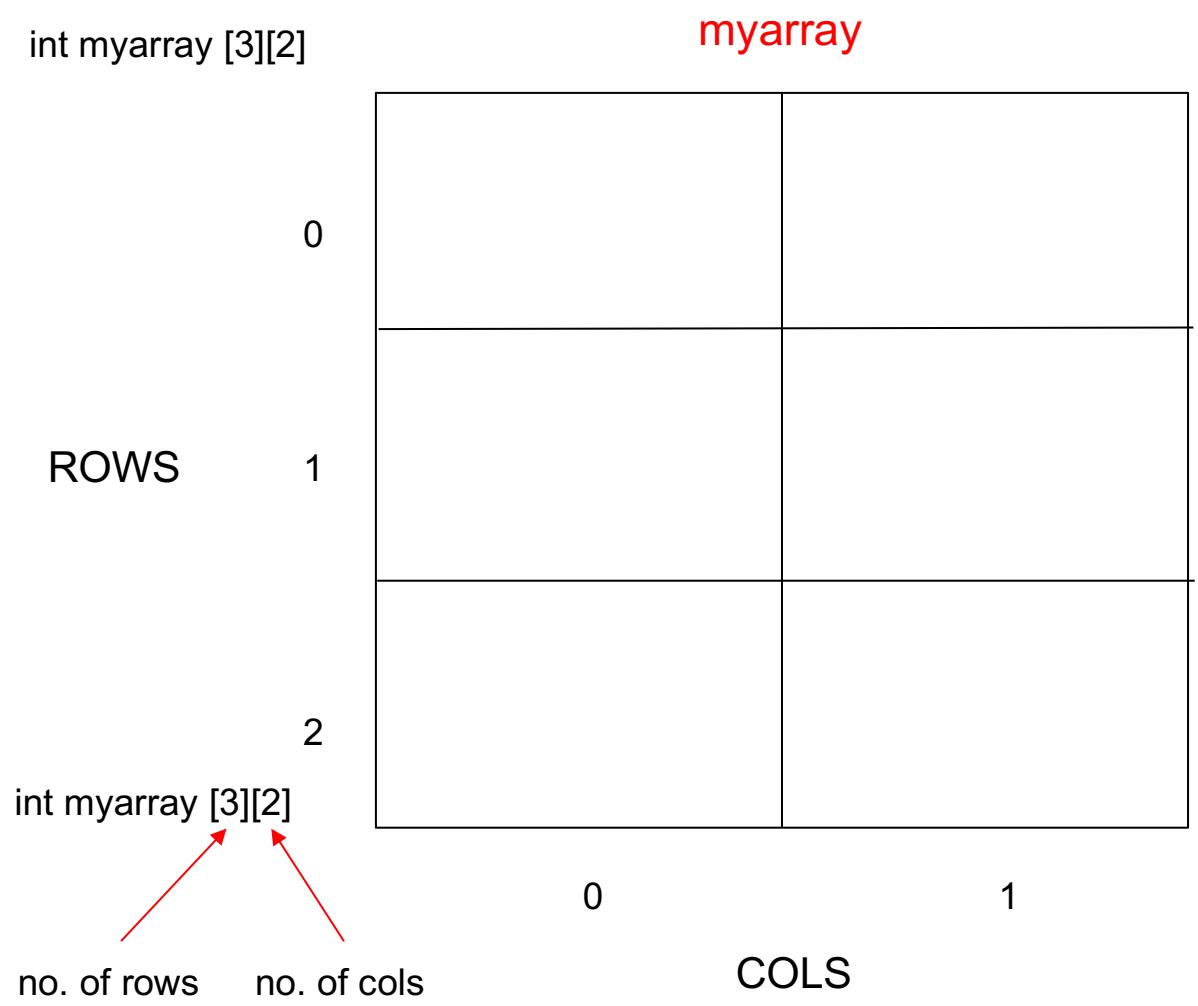
e.g.

int numbers [2][2];



Here's another example:

int myarray[3][2];



If you were to hard-code data into the above array, it would look like:

```
myarray[0][0] = 10;
myarray[0][1] = 15;
myarray[1][0] = 20;
myarray[1][1] = 25;
myarray[2][0] = 30;
myarray[2][1] = 35;
```

Let's now develop the code to enter and display data for the array myarray.

```
/*
Entering and displaying data in a 2-Dimensional array
*/

#include <stdio.h>
```

```c
// Use symbolic names for the Rows and Cols
#define ROW 3
#define COL 2

int main()
{
	int myarray[ROW][COL];
	int i, j;


	printf("Enter %d numbers\n", ROW*COL);
	// Enter data i.e., whole numbers, into the array
	for(i = 0; i < ROW; i++)
	{
		// Inner for that handles the Cols
		for(j = 0; j < COL; j++)
		{
			// read in data
			scanf("%d", &myarray[i][j]);

		} // end inner for

	} // end outer for


	// Display the data entered into the array
	for(i = 0; i < ROW; i++)
	{
		// Inner for that handles the Cols
		for(j = 0; j < COL; j++)
		{
			// display the data
			printf("\nRow %d, Col %d contains %d", i, j,
myarray[i][j]);

		} // end inner for

	} // end outer for
```

```
        return 0;

} // end main()
```

Repl 7.2: https://replit.com/@michaelTUDublin/72-2-D-array-p1#main.c