Advanced C programming

Ramtin Behesht Aeen

June 2024

Part I

Basic C Data Structures, Pointers, and File Systems

Chapter 1

Basic C Data Structures

1.1 Arrays

1.1.1 Concepts

In C: Array is a collection of consecutive objects with same data type

- Array is a variable
- Array has a data type and name with square bracket
- Within the brackets are the number of elements in the array

```
1
2     float best_score[3] = {
3         1.1, 2.1, 3.1, 4.1
4     };
```

- In C Arrays are non dynamic it means that their size be altered as the program runs.
- Arrays in C have no bounds checking, so it is possible to reference an element, which is not exist. calling element outside the Array declaration
- Arrays have a lot in commons with pointers
- It is possible to declare an arrays length as the program runs but it is mostly avoided, so jest set the value in the code and know that it can not be increased when program runs

Dynamic Arrays

Example of declare an array with use of pointers and still set its length at runtime:

- Code:

```
1
2
3       printf("Enter the size of the Array: ");
4       scanf("%d",&arraySize);
5
6       // Dynamically allocate memory for the array
7       myArray = (int *)malloc(arraySize * sizeof(int));
8
9       //Checking wheter rhe memory allocation was successful or
            not:
10      if(myArray == NULL) {
11              fprintf(stderr, "Memory allocation failed\n");
12              return 1;
13      }
14
15      //Initialize array with values:
16      for(int i = 0; i < arraySize; i++) {
17              myArray[i] = i * i;
18      }
19
20      //Print Arrays Values
21      for (int i = 0; i < arraySize; i++){
22              printf("myArray[%d] = %d\n",i, myArray[i]);
23      }
24
25      //Free the allocated memory
26      free(myArray);
27
28      return 0;
29      }
```

- Compile result:

```
1       Enter the size of the Array: 10
2       myArray[0] = 0
3       myArray[1] = 1
4       myArray[2] = 4
5       myArray[3] = 9
6       myArray[4] = 16
7       myArray[5] = 25
8       myArray[6] = 36
9       myArray[7] = 49
10      myArray[8] = 64
11      myArray[9] = 81
```

In this code, arraySize is determined at runtime based on the user's input. The malloc function is used to allocate the required amount of memory. It's important to free the allocated memory with free when you're done using the dynamically allocated array to prevent memory leaks. Keep in mind that dynamic memory allocation allows for flexible array sizes, but it also requires careful management of the allocated memory.

Example of declare an array with out useing pointers and still set its length at runtime(Using Variable Length Arrays (VLA)):

```c
#include <stdio.h>

int main(){
	int arraySize;

	//Ask the user for the array size:
	printf("Enter the size of the array: ");
	scanf("%d", &arraySize);

	//declare VLA based on the user input:
	int myArray[arraySize];

	//Intitialize array with values
	for (int i = 0; i < arraySize; i++){
		myArray[i] = i * i;
	}

	//Print array values:
	for (int i = 0; i < arraySize; i++){
		printf("myArray[%d] = %d\n", i, myArray[i]);
	}

	return 0;
}
```

- Compile result:

```
Enter the size of the Array: 5
myArray[0] = 0
myArray[1] = 1
myArray[2] = 4
myArray[3] = 9
myArray[4] = 16
```

In this code, myArray is a VLA whose size arraySize is determined by the user input at runtime. No pointers are used, and the array is directly accessed by its indices. Keep in mind that not all compilers support VLAs, and their use is controversial due to potential risks such as stack

overflow. Also, VLAs are not part of the ISO C++ standard, so they are not portable across all platforms or languages. For these reasons, dynamic memory allocation with pointers is generally preferred for arrays with sizes determined at runtime.

1.1.2 Working With Arrays

dublicating an array

- In this code the process of duplicated an array is demonstrated:

⚠Note: Duplicate must have the same or greater number of elements.(We as an Programmer must enforce this rule, because Compile wont check it)

```
#include <stdio.h>

int main(){
        int original_array[5] = {10, 20, 30, 40, 50};
        int duplicate[5];

        for ( int i = 0; i < 5; i++){
                duplicate[i] = original_array[i];
        }

        puts("Arrays Values \n");

        for ( int j = 0; j < 5; j++ ){
                printf("Element#%d %3d == %3d \n", j,
                    original_array[j], duplicate[j]);
        }


}
```

- Compile Result:

```
Arrays Values

Element#0 10 == 10
Element#1 20 == 20
Element#2 30 == 30
Element#3 40 == 40
Element#4 50 == 50
```

1.1.3 Passing an array to an function

passing the whole array

an entire array has been passed to the function. In the function, the argument is a character variable array with empty bracket and indicates that the entire array has passed. within the function the array has modified and output.

- Code:

```
1
2               #include <stdio.h>
3
4               void print_array_func(char input_array[]){
5
6                       for ( int x = 0; x < 6; x++){
7                               input_array[x]++;
8                               putchar(input_array[x]);
9                       }
10              }
11
12              int main(){
13
14                      char text[] = "Gdkkn ";
15
16                      print_array_func(text);
17                      putchar('\n');
18
19                      return(0);
20
21              }
```

- Compile Result:

```
1       Hello!
```

passing the arrays elements individually

- Code:

```c
#include <stdio.h>

void print_arrays_char( char a ){
	a++;
	putchar(a);
}

int main(){
	char text[] = "Gdkkn";

	for( int x = 0; x < 6; x++ ){
		print_arrays_char(text[x]);
	}
	putchar('\n');
}
```

- Compile Result:

```
Hello
```

retunring an array

⚠ Note: While individual array elements can be returned sequentially, to return an entire array created within a function, you must utilize pointers.

⚠ Wrong Way of returning an Array:

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

		int make_array_func(void){
			int array[5];
			for ( int x = 0; x < 5; x++){
				array[x] = rand() % 10 + 1;
			}
			return(array);
		}

		int main(){
			int r[5];
			r = make_array_func();
			puts("Here are your 5 random numbers:");
			for (int x = 0; x < 6; x++)
				printf("%d\n",r[x]);
		}
```

- Compile Result:

```
01_04_returning_an_array.c:12:15: warning: returning int *
    from a function with return type int makes integer from
    pointer without a cast [-Wint-conversion]
12 |       return(array);
|             ^
01_04_returning_an_array.c: In function main:
01_04_returning_an_array.c:19:11: error: assignment to
    expression with array type
```

return an array from a function by returning a pointer to the array

in C, you can return an array from a function by returning a pointer to the array. However, you need to ensure that the array you're returning is not a local array inside the function, as it will be destroyed once the function scope ends. One way to do this is by dynamically allocating the array on the heap using malloc. Here's how you can modify your function to return an array using a pointer:

- The asterisk * before make_array_func in the function declaration indicates that the function returns a pointer.

- Code:

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

int* make_array_func(void){
       //Dynamically allocate an array of 5 integers
       int *array = malloc( 5 * sizeof(int));

       if (array == NULL){
              //Handle memory allocation failure
              fprintf(stderr, "Memory allocation failed\n");
              exit(EXIT_FAILURE);
       }
       //Asigning values to the Array:
       for ( int x = 0; x < 5; x++){
              array[x] = rand() % 10 + 1;
       }

       return array;
}

int main(){
       int *r;

       r = make_array_func();
```

```
25
26              puts("Here are 5 random numbers:");
27              for ( int x = 0; x < 5; x++) {
28                      printf("%d\n", r[x]);
29              }
30              //Frreing up the memory!! Dont Forgot that
31              free(r);
32              return 0;
33
34      }
```

- Compile Result:

```
1       Here are 5 random numbers:
2       4
3       7
4       8
5       6
6       4
```

1.1.4 Multi-Dimensional Array

- Example code of multi-dimensional array for creating a three by three matrix, which is fulled randomly with X or O:

```
1               #include <stdio.h>
2               #include <stdlib.h>
3               #include <time.h>
4
5
6               int main(){
7                       char tic_tac_toe [3][3];
8                       char array[] = {'x', 'o'};
9
10                      //Initiallize the board
11                      for ( int i = 0; i < 3; i++ ){
12                              for ( int j = 0; j < 3; j++){
13                                      // Assign 'x' or 'o' randomly
                                            to the board
14                                      // rand() % 2 will Generate
                                            numbers between 0 and 1
15                                      tic_tac_toe[i][j] =
                                            array[rand() % 2];
16
17                              }
18                      }
19
20                      //Printing the board
21                      for ( int i = 0; i < 3; i++ ){
```

```
22                              for ( int j = 0; j < 3; j++){
23                                      printf(" %c ",
                                            tic_tac_toe[i][j]);
24                              }
25                              putchar('\n');
26                      }
27
28                      return(0);
29              }
```

- Compile Result:

```
1       o  o  x
2       x  o  x
3       x  x  x
```

1.1.5 Exercises

1.2 Structure

1.2.1 Concepts

- A structure is a container of multiple variable types.

- The variables can be different data types, the same data type, or mixed and matched in various quantities.

- All the variables relate to each other or describe a complex data structure like record of a Database.

- The key word 'Struct' will declare a structure, which is followed by name of the Structure.

- For using the structure we should define a variable with type of Struct

- Example Code:

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

struct animal {
        int id;
        int life_span;
        char sound[20];
        char class[10];
};

int main(){

        struct animal german_shepherd;
        struct animal chinchilla_persians;

        german_shepherd.id = 31415;
        german_shepherd.life_span = 20;
        //german_shepherd.sound = "Bark";
        strcpy(german_shepherd.sound,"Bark");
        //german_shepherd.class = "Dog"
        strcpy(german_shepherd.class,"Dog");


        chinchilla_persians.id = 31416 ;
        chinchilla_persians.life_span = 10;
        //chinchilla_persians.sound = "Meow";
        //chinchilla_persians.class = "Cat";

        printf("%d: German Shepherd is a %s breed, thus it
            will %s and live for %d ",
               german_shepherd.id,
               german_shepherd.class,
               german_shepherd.sound,
               german_shepherd.life_span
             );

        return(0);
}
```

- Compile Result:

```
31415: German Shepherd is a Dog breed, thus it will Bark
    and live for 20
```

1.2.2 Nesting Structures

1.2.3 Array of Structures

Sending a Structure to a function

1.2.4 Exercises

1.3 Union

1.4 Chapter Exercises