

# 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
10    not:
11    if(myArray == NULL) {
12        fprintf(stderr, "Memory allocation failed\n");
13        return 1;
14    }
15
16    //Initialize array with values:
17    for(int i = 0; i < arraySize; i++) {
18        myArray[i] = i * i;
19    }
20
21    //Print Arrays Values
22    for (int i = 0; i < arraySize; i++){
23        printf("myArray[%d] = %d\n",i, myArray[i]);
24    }
25
26    //Free the allocated memory
27    free(myArray);
28
29    return 0;
30    }
```

---

- 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 using pointers and still set its length at runtime(Using Variable Length Arrays (VLA)):

---

```
1      #include <stdio.h>
2
3      int main(){
4          int arraySize;
5
6          //Ask the user for the array size:
7          printf("Enter the size of the array: ");
8          scanf("%d", &arraySize);
9
10         //declare VLA based on the user input:
11         int myArray[arraySize];
12
13         //Intitalize array with values
14         for (int i = 0; i < arraySize; i++){
15             myArray[i] = i * i;
16         }
17
18         //Print array values:
19         for (int i = 0; i < arraySize; i++){
20             printf("myArray[%d] = %d\n", i, myArray[i]);
21         }
22
23         return 0;
24     }
```

---

- Compile result:

---

```
1      Enter the size of the Array: 5
2      myArray[0] = 0
3      myArray[1] = 1
4      myArray[2] = 4
5      myArray[3] = 9
6      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

#### duplicating 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)

---

```

1  #include <stdio.h>
2
3  int main(){
4      int original_array[5] = {10, 20, 30, 40, 50};
5      int duplicate[5];
6
7      for ( int i = 0; i < 5; i++){
8          duplicate[i] = original_array[i];
9      }
10
11     puts("Arrays Values \n");
12
13     for ( int j = 0; j < 5; j++ ){
14         printf("Element#%d %3d == %3d \n", j,
15             original_array[j], duplicate[j]);
16     }
17
18 }
```

---

- Compile Result:

---

```

1  Arrays Values
2
3  Element#0 10 == 10
4  Element#1 20 == 20
5  Element#2 30 == 30
6  Element#3 40 == 40
7  Element#4 50 == 50
```

---



### 1.1.3 Passing an array to an function

passing the whole array

passing the arrays elements individually

- Code:

---

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

---

- Compile Result:

---

```
1  Hello
```

---

### 1.1.4 Multi-Dimensional Array

## 1.2 Structure

### 1.2.1 Concepts

### 1.2.2 Nesting an Structure

### 1.2.3 Array of Structure

### 1.2.4 Sending a Structure to a function

## 1.3 Union