

# Programming For Engineers

## Pointers in C Programming: Part 02

by

Wan Azhar Wan Yusoff<sup>1</sup>, Ahmad Fakhri Ab. Nasir<sup>2</sup>  
Faculty of Manufacturing Engineering  
wazhar@ump.edu.my<sup>1</sup>, afakhri@ump.edu.my<sup>2</sup>



# 0.0 Chapter's Information

- **Expected Outcomes**
  - To further use pointers in C programming
- **Contents**
  - 1.0 Pointer and Array
  - 2.0 Pointer and String
  - 3.0 Pointer and dynamic memory allocation



# 1.0 Pointer and Array

- We will review array data type first and later we will relate array with pointer.
- Previously, we learn about basic data types such as integer, character and floating numbers. In C programming language, if we have 5 test scores and would like to average the scores, we may code in the following way.



# 1.0 Pointer and Array

```
#include <stdio.h>
int main()
{
    // declare and initialize values
    float score1 = 86.3;
    float score2 = 77.2;
    float score3 = 91.4;
    float score4 = 56.7;
    float score5 = 83.9;
    float average = 0.0;
    // determine the average
    average = (score1+score2+score3+score4+score5)/5;
    // display the results
    printf("The scores are\n");
    printf("Score1 = %4.1f\n",score1);
    printf("Score1 = %4.1f\n",score2);
    printf("Score1 = %4.1f\n",score3);
    printf("Score1 = %4.1f\n",score4);
    printf("Score1 = %4.1f\n",score5);
    printf("The average score is = %4.1f\n",average);
    return 0;
}
```



# 1.0 Pointer and Array

```
"C:\Users\user\Desktop\BHM2013 Programming for Engineers\Array_...
The scores are
Score1 = 86.3
Score1 = 77.2
Score1 = 91.4
Score1 = 56.7
Score1 = 83.9
The average score is = 79.1
Process returned 0 (0x0) execution time : 0.016 s
Press any key to continue.
```

- This program is manageable if the scores are only 5. What should we do if we have 100,000 scores? In such case, we need an efficient way to represent a collection of similar data type<sup>1</sup>. In C programming, we usually use array.
- Array is a fixed-size sequence of elements of the same data type.<sup>1</sup>
- In C programming, we declare an array like the following statement:



# 1.0 Pointer and Array

```
float score[5];
```

- The above statement tells the compiler that we want to group five float numbers in one array called score. Thus in computer memory, we organize the data like the following:

score[0]	four byte of a float number
score[1]	four byte of a float number
score[2]	four byte of a float number
score[3]	four byte of a float number
score[4]	four byte of a float number



# 1.0 Pointer and Array

- So the compiler has allocated 5 memory spaces for each float number. Notice that the subscript (the number inside the parenthesis) is from 0 to 4 not from 0 to 5 because we allocate only 5 memory spaces and we start our subscript from 0. Be aware that in an array, there is only one data type, in this case the float data type. Thus, the total bytes reserved for the variable `score[5]` are 20 bytes.
- Each data in an array is called an element of an array. In our case, `score[2]` or `score[4]` is an element of the array `score[5]`. Similarly, we can declare arrays for other data types.



# 1.0 Pointer and Array

`int number[100]` - array name "number" with 100 sequence of integer data (4000 bytes).

`char huruf[53]` - array name "huruf" with 53 sequence of character data (53 bytes).

`double points[1000]` - array name "points" with 1000 sequence of double data (12,000 bytes).

- The elements of an array can be initialized in several ways:

- Basic initialization - initialize all values in the array.

```
int number[5] = {3,7,1,1,21};
```

- In this case, `number[0] = 3`, `number[1] = 7`, `number[2] = 1`, `number[3] = 1` and `number[4] = 21`.





# 1.0 Pointer and Array

- Partial initialization - initialize some values in the array.

```
int number[5] = {3,7};
```

- In this example, the first two elements are initialized while the rest is zero. Specifically, `number[0] = 3`, `number[1] = 7`, `number[2] = 0`, `number[3] = 0` and `number[4] = 0`.

- Initialize to all zero - `int number[5] = {0};`

- To display the elements of an array we need to use repetitive structure (loop) such as the for-loop or the while-loop.
- To continue with our first example, we write a program using an array data type.



# 1.0 Pointer and Array

```
#include <stdio.h>
int main()
{
    // declare and initialize and array named "score" with five values
    float score[5] = {86.3,77.2,91.4,56.7,83.9};
    int i;
    // display the array elements
    printf("The scores are:\n\n");
    for (i=0;i<5;i++)
        printf("Score[%d] = %4.1f\n",i,score[i]);
    return 0;
}
```

C:\ "C:\Documents and Settings\user\Desktop\BHM2013 Programming fo

The scores are:

```
Score[0] = 86.3
Score[1] = 77.2
Score[2] = 91.4
Score[3] = 56.7
Score[4] = 83.9
```



# 1.0 Pointer and Array

- To sum and average a group of numbers, array is the appropriate data structure to use. In an example below, user can sum and average numbers regardless the quantity of numbers.



# 1.0 Pointer and Array

```
/* This program is to illustrate the use of an array to perform the sum and averaging number.
Written by: WAWY February 2012 */
int main()
{ /* Get the number of students.*/
  int num_of_students;
  printf("Please enter the number of students:");
  scanf("%d",&num_of_students); printf("\n");
  /* Key in all the marks into an array called marks[]
  Notice that the variable "num_of_students"
  determines the size of the array.*/
  float marks[num_of_students];
  int i;
  for (i=0;i<num_of_students;i++)
  { printf("Please key in the %d mark:",i+1); //i+1 is used so that the output start from 1 instead of 0
    scanf("%f",&marks[i]); }
  /* Now, determine the sum and the average of the marks */
  float sum = 0;
  float average = 0;
  for (i=0;i<num_of_students;i++)
  { sum = sum + marks[i]; }
  average = sum/num_of_students;
  /* Now, display the results */
  printf("_____ \n\n");
  for (i=0;i<num_of_students;i++)
  { printf("\tMark %d\t:%9.2f\n",i+1,marks[i]); } //i+1 is used so that the output start from linstead of 0
  printf("_____ \n\n");
  printf("\tSum\t:%9.2f\n",sum);
  printf("\tAverage\t:%9.2f\n",average);
  printf("_____ \n");
  /* End the program. */
  return 0; }
```



# 1.0 Pointer and Array

```
"C:\Users\user\Desktop\BHM2013 Programming for Engineers\Program Examp
Please enter the number of students:5
Please key in the 1 mark:66.98
Please key in the 2 mark:90.43
Please key in the 3 mark:87.63
Please key in the 4 mark:49.67
Please key in the 5 mark:78.60
-----
Mark 1 : 66.98
Mark 2 : 90.43
Mark 3 : 87.63
Mark 4 : 49.67
Mark 5 : 78.60
-----
Sum : 373.31
Average : 74.66
-----
Process returned 0 (0x0) execution time : 26.561 s
Press any key to continue.
```

- To determine the highest and the lowest numbers from a group of numbers, we can also use an array.



# 1.0 Pointer and Array

```
#include <stdio.h>
int main()
{
    /* Get the number of students.*/
    int num_of_students;
    printf("Please enter the number of students:");
    scanf("%d",&num_of_students);
    printf("\n");
    /* Key in all the marks into an array called marks[]
       Notice that the variable "num_of_students"
       determines the size of the array.*/
    float marks[num_of_students];
    int i;
    for (i=0;i<num_of_students;i++)
    {
        printf("Please key in the %d mark:",i+1); //i+1 is used so that the output start from 1
        scanf("%f",&marks[i]); } /* Now, determine the highest of the marks */
    float highest = 0; //Set the variable "highest" to the smallest number we think possible
    for (i=0;i<num_of_students;i++)
    {
        if (marks[i] > highest)
            highest = marks[i]; } /* Now, determine the lowest of the marks */
    float lowest = 100000000.0; //Set the variable "lowest" to the biggest number we think possible
    for (i=0;i<num_of_students;i++)
    {
        if (marks[i] < lowest)
            lowest = marks[i]; } /* Now, display the results */
    printf("_____ \n\n");
    for (i=0;i<num_of_students;i++)
    {
        printf("\tMark %d\t:%9.2f\n",i+1,marks[i]); } //i+1 is used so that the output start from 1
    printf("_____ \n\n");
    printf("\tLowest\t:%9.2f\n",lowest);
    printf("\tHighest\t:%9.2f\n",highest);
    printf("_____ \n");
    return 0; }
```



# 1.0 Pointer and Array

```
ca "C:\Documents and Settings\user\Desktop\BHM2013 Programming
Please key in the 1 mark:55.87
Please key in the 2 mark:56.98
Please key in the 3 mark:90.53
Please key in the 4 mark:86.21
Please key in the 5 mark:44.67
Please key in the 6 mark:82.30
Please key in the 7 mark:76.50

Mark 1 : 55.87
Mark 2 : 56.98
Mark 3 : 90.53
Mark 4 : 86.21
Mark 5 : 44.67
Mark 6 : 82.30
Mark 7 : 76.50

Lowest : 44.67
Highest : 90.53
```

- To represent tabulated data, we use 2 dimensional arrays. Below is a table of times versus speeds from an experiment. The first two columns are taken from experimental data while the data in the third column is calculated from the first two columns.



# 1.0 Pointer and Array

Times (seconds)	Speed (rad/sec)	Average Acceleration (rad/sec <sup>2</sup> )
0.00	0	$= \frac{5 - 0}{0.10 - 0.00} = 50$
0.10	5	$= \frac{13 - 5}{0.20 - 0.10} = 80$
0.20	13	$= \frac{19 - 13}{0.30 - 0.20} = 60$
0.30	19	$= \frac{27 - 19}{0.40 - 0.30} = 80$
0.40	27	$= \frac{35 - 27}{0.50 - 0.40} = 80$
0.50	35	.

- An example below shows how to declare and operates a 2-D array.





# 1.0 Pointer and Array

```
int main()
{
    /* Create the 2-D array and initialize */
    int row;
    int col;
    int row_size = 6;
    int col_size = 3;
    /*This how we initialize 2-D array. Later we will read from a text file. */
    float kinematics[6][3] = {
        0.00,0.00,0.00,
        0.10,5.00,0.00,
        0.20,13.00,0.00,
        0.30,19.00,0.00,
        0.40,27.00,0.00,
        0.50,35.00,0.00 };
    /*Calculate the third column data */
    for (row=0;row<row_size-1;row++)
        kinematics[row][2] = (kinematics[row+1][1] - kinematics[row][1])/(kinematics[row+1][0] - kinematics[row][0]);
    /* Now, display the results */
    printf("_____ \n\n");
    for (row=0;row<row_size;row++)
    {
        for (col=0;col<col_size;col++)
        {
            printf("%9.2f\t",kinematics[row][col]);
        }
        printf("\n");
    }
    printf("_____ \n\n");
    return 0;
}
```



# 1.0 Pointer and Array

```
"C:\Documents and Settings\user\Desktop\BHM2013 Programming"
0.00      0.00      50.00
0.10      5.00      80.00
0.20     13.00     60.00
0.30     19.00     80.00
0.40     27.00     80.00
0.50     35.00      0.00

Process returned 0 (0x0)   execution time : 0.063 s
Press any key to continue.
```

- What is an array has to do with pointer? The name of an array is a pointer to the first element of an array. The example is illustrated below.



# 1.0 Pointer and Array

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i;
    int a[5] = {1,2,3,4,5};
    //declare array of 5 elements
    for (i=0;i<5;i++)
    {
        printf("a[%d] = %d\t*(a+%d) = %d\n",i,a[i],i,*(a+i));
    }
    return 0;
}
```

```
F:\Wan-Azhar-Wan-Yusoff\WAWY-Programming-for-Engineers\WA...
a[0] = 1          *(a+0) = 1
a[1] = 2          *(a+1) = 2
a[2] = 3          *(a+2) = 3
a[3] = 4          *(a+3) = 4
a[4] = 5          *(a+4) = 5

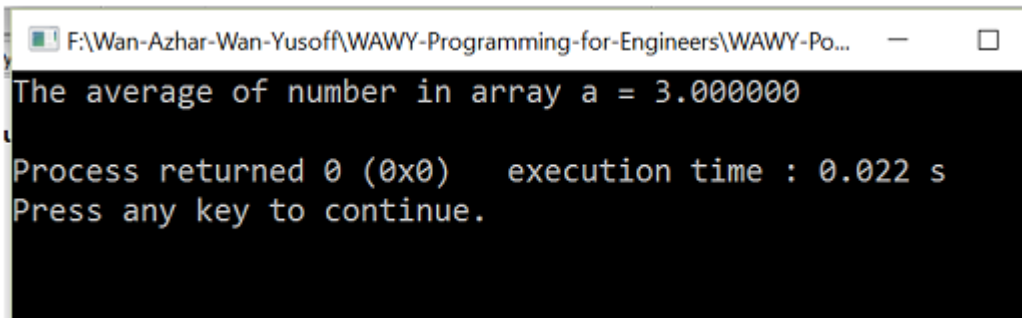
Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```



# 1.0 Pointer and Array

- In the previous example,  $*(a) = a[0]$ ,  $*(a+1) = a[1]$ ,  $*(a+2) = a[2]$  etc. We can say that the name of an array contains the address of the first array element. In fact, we can declare an array using pointer as shown below.

```
#include <stdio.h>
#include <stdlib.h>
double average(int a[], int length);
int main()
{
    int a[5] = {1,2,3,4,5};
    printf("The average of number in array a = %f\n", average(a,5));
    return 0;
}
double average(int* a, int length)
{
    int i;
    double sum =0;
    for (i=0;i<length;i++)
        sum += *(a+i);
    return sum/length;
}
```



```
F:\Wan-Azhar-Wan-Yusoff\WAWY-Programming-for-Engineers\WAWY-Po...
The average of number in array a = 3.000000
Process returned 0 (0x0) execution time : 0.022 s
Press any key to continue.
```



# 1.0 Pointer and Array

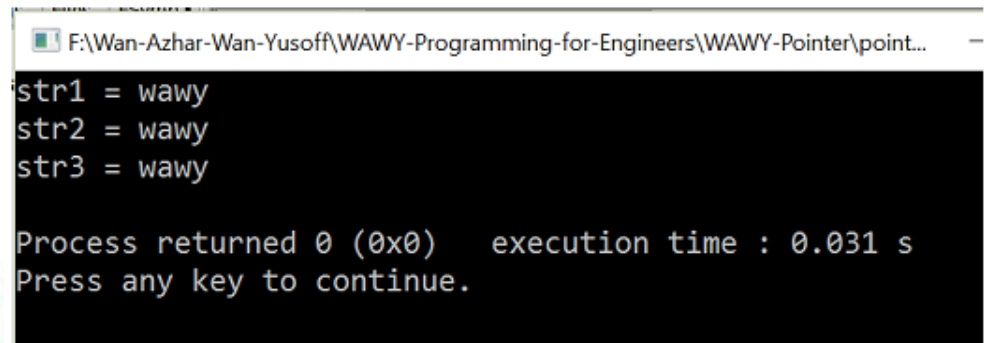
- Notice that, in the above program, passing an array is using a pointer. In this case, the declaration in the function parameter is a pointer to an integer, `int* a`. Actually, we pass the address of the first element of an array to the function. This is very important point – the size of an array is not relevant. We can have an array of 1000 elements but since we pass only the address, the function can work on an array without the need to copy all elements of the array. Another important point is to always pass an array to a function together with the length of the array. Otherwise, the function will not know the size of an array.



## 2.0 Pointers and String

- A string is an array of character with NULL character as the last element. The NULL character is the number 0. This is different from character zero ('0') which value is 48. For example the string "wawy" is an array of 5 bytes which are 'w', 'a', 'w', 'y' and '\0'. We provide a program below to illustrate the various declaration of a string.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char str1[5] = "wawy";
    char* str2 = "wawy";
    char str3[] = "wawy";
    printf("str1 = %s\n", str1);
    printf("str2 = %s\n", str2);
    printf("str3 = %s\n", str3);
    return 0;
}
```



```
F:\Wan-Azhar-Wan-Yusoff\WAWY-Programming-for-Engineers\WAWY-Pointer\point...
str1 = wawy
str2 = wawy
str3 = wawy
Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```



## 2.0 Pointers and String

- Furthermore, we access string just like we access array and pointer. Study the example below:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    char ch;
    ch = "wawy"[1];
    printf("ch = %c\n", ch);
    char* str = "wawy";
    printf("str[3] = %c\n", str[3]);
    printf("str[2] = %c\n", *(str+2));
    printf("The NULL character number = %d\n", str[4]); //number 0
    printf("The NULL character character = %c\n", str[4]); //blank
    printf("What is after the NULL character! = %c\n", str[5]); //dangerous game!
    return 0;
}
```

```
F:\Wan-Azhar-Wan-Yusoff\WAWY-Programming-for-Engineers\WAWY-P...
ch = a
str[3] = y
str[2] = w
The NULL character number = 0
The NULL character character = 
What is after the NULL character! = s

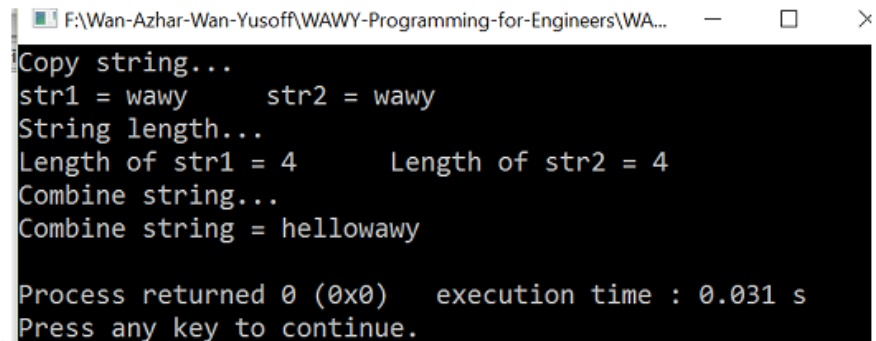
Process returned 0 (0x0)   execution time : 0.018 s
Press any key to continue.
```



## 2.0 Pointers and String

- We can use string library to manipulate string data. The library is `<string.h>`. In the following example, we perform 3 string functions: (1) copy string, (2) determine the string length and (3) combining string.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
    char *str1 = "wawy";
    char str2[5];
    strcpy(str2, str1); //str2 destination and str1 source
    printf("Copy string...\n");
    printf("str1 = %s\tstr2 = %s\n", str1, str2);
    printf("String length...\n");
    printf("Length of str1 = %d\tLength of str2 = %d\n", strlen(str1), strlen(str2));
    printf("Combine string...\n");
    char str3[20] = "hello";
    printf("Combine string = %s\n", strcat(str3, "wawy"));
    return 0;
}
```



```
F:\Wan-Azhar-Wan-Yusoff\WAWY-Programming-for-Engineers\WA...
Copy string...
str1 = wawy      str2 = wawy
String length...
Length of str1 = 4      Length of str2 = 4
Combine string...
Combine string = helloworldwawy

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```



## 3.0 Pointer and Memory Allocation

- Data structure of C program is fixed. For example, in our program we define the data type such as integer, double, array of integer, string etc. before the program runs. Once we run the program, we cannot define a new variable. Is there any way for us to define a new variable during the running of the program?
- C use dynamic data storage allocation to create new space for data. This function is available in `<stdlib.h>`. We will show you two important functions: `malloc()` and `calloc()`. We use `malloc()` to allocate (create) a block of memory and does not initialize. We use `calloc()` to allocate a block of memory and clears it. We provide a program example to clarify this idea.



# 3.0 Pointer and Memory Allocation

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i;
    char* p;
    //allocate 1000 bytes
    //address of pointer p is the first byte
    p = malloc(1000);
    if (p==NULL)
        printf("fail to allocate memory.\n");
        //Display the first 10 memory of p
    for (i=0;i<5;i++)
        printf("Address: %p\tValue: %d\n",p+i,* (p+i));
    printf("\nNow, change memory value.\n");
    for (i=0;i<5;i++)
    {
        *(p+i)=i;
        printf("Address: %p\tValue: %d\n",p+i,* (p+i));
    }
    free(p); //return back memory to OS
    return 0;
}
```

```
F:\Wan-Azhar-Wan-Yusoff\WAWY-Programming-for-Engineers\...
Address: 00A90D88      Value: 104
Address: 00A90D89      Value: 35
Address: 00A90D8A      Value: -87
Address: 00A90D8B      Value: 0
Address: 00A90D8C      Value: 104

Now, change memory value.
Address: 00A90D88      Value: 0
Address: 00A90D89      Value: 1
Address: 00A90D8A      Value: 2
Address: 00A90D8B      Value: 3
Address: 00A90D8C      Value: 4

Process returned 0 (0x0)   execution time : 0.016 s
Press any key to continue.
```



## 3.0 Pointer and Memory Allocation

- In the previous example, we create 1000 bytes of extra memory space and get the address of the first byte and store it in pointer `p`. Knowing the `p` address, we can observe the first 5 bytes. Since `malloc()` does not initialize, we get garbage. Then, we change the value of the first 5 bytes by using indirection of pointer `p`. This means that we can have new memory space to store data. Lastly, we need to “free-up” the space after we use by calling the function `free()`;
- Our next example is to use `calloc()` function. We need to supply to information to `calloc()` (1) the number of elements and (2) is the size of one element. For instance, if we want to have 10 integer space, we should call `calloc()` something like this: `p = calloc(10, sizeof(int));`
- In the following example, we create 5 new spaces for double and give values to all of them.



# 3.0 Pointer and Memory Allocation

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i;
    double* p;
    //allocate 5 doubles
    //address of pointer p is the first byte
    p = calloc(5, sizeof(double));
    if (p==NULL)
        printf("fail to allocate memory.\n");
    //Display the first 10 memory of p
    for (i=0;i<5;i++)
        printf("Address: %p\tValue: %d\n",p+i,*p+i));
    printf("\nNow, change memory value.\n");
    for (i=0;i<5;i++)
    {
        *(p+i)=3.14*i;
        printf("Address: %p\tValue: %f\n",p+i,*p+i));
    }
    free(p); //return back memory to OS
    return 0;
}
```

```
F:\Wan-Azhar-Wan-Yusoff\WAWY-Programming-for-Engineers\WAWY-Point...
Address: 00BF0D88      Value: 0
Address: 00BF0D90      Value: 0
Address: 00BF0D98      Value: 0
Address: 00BF0DA0      Value: 0
Address: 00BF0DA8      Value: 0

Now, change memory value.
Address: 00BF0D88      Value: 0.000000
Address: 00BF0D90      Value: 3.140000
Address: 00BF0D98      Value: 6.280000
Address: 00BF0DA0      Value: 9.420000
Address: 00BF0DA8      Value: 12.560000

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```



## 3.0 Pointer and Memory Allocation

- Notice that, by using `calloc()`, we manage to initialize to zero value. Also, do not forget to free the space by calling `free()` after we finish using it.
- In summary, we have use pointers in many ways.
  - We use pointer as a variable to the address of another variable.
  - We learn how to inspect the address and content of our memory space.
  - Third, we learn how to pass pointer to a function so that we can manipulate more than one variable solving the issue that a function will return only one value.
  - We know that array and pointer are closely related.



## 3.0 Pointer and Memory Allocation

- We know that string is an array of character terminated with null character. We also know that a string can be accessed through pointer.
- We can create memory space during execution of a program. We use dynamic memory allocation functions and receive the pointer to the address of the first byte of the memory block. We can store data but we also remember to free up the memory space after we use.

