Digital image Processing

5. Array and Pointer

Dr.Weerayuth Suanpaga

Department of Civil Engineering
Faculty of Engineering , Kasetsart University
Bangkok, Thailand

https://pirun.ku.ac.th/~fengwys/ram/ge749/lect/lect5.pdf

Array 1

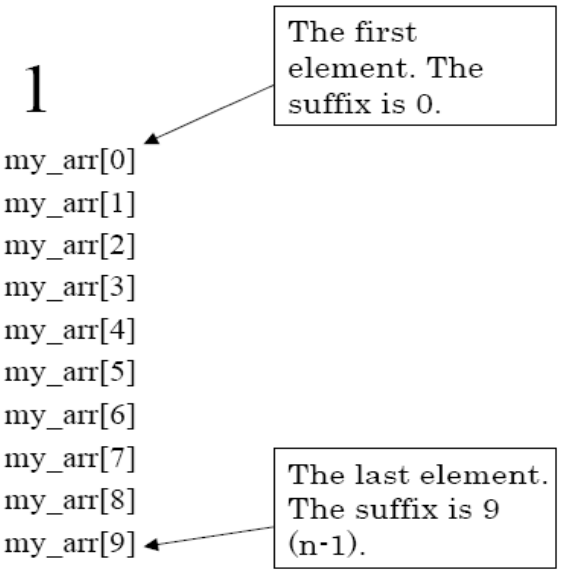
Array stores a series of data.
How to define an array is,

type name[size];

For example, the following definition creates an array named my_arr, which has 10 integer elements.

```
int my_arr[10];
```

Each element can be refereed as follows.

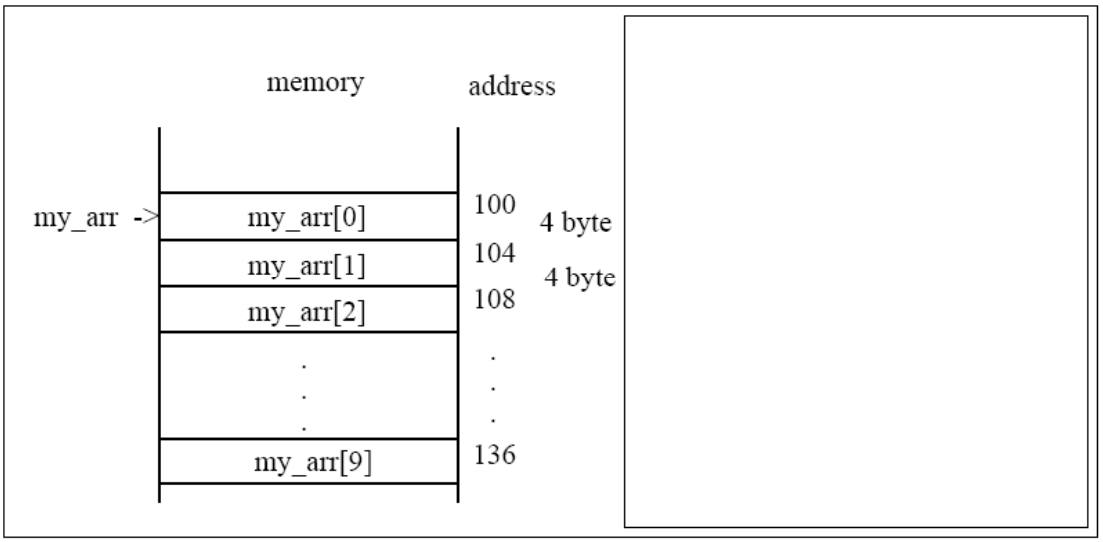


other examples

```
double   double_array[4000];
unsigned char     line_img[1024];
unsigned char     * img[1024];
```

Array 2

int my_arr[10];



Array 3 - example

```
#include <stdio.h>

void main( )
{
        double arr[5];
        int i;
        double total=0.0;

        for (i=0;i<5;i++){
                printf("input number%d=",i);
                scanf("%lf",&arr[i]);
                total+=arr[i];
        }
        for (i=0;i<5;i++){
                printf("OUTPUT number%d VALUE=%4.2lf\n",i,arr[i]);

        }

        printf("your result=%lf\n",total);
}
```

Array 4 – Setup a LUT

```
#include<stdio.h>
#define   LUT_SIZE                256
#define   LUT_MIN                 100
#define   LUT_MAX                 200
#define   LUT_VALUE(z)    (255*( (z) - LUT_MIN ) / ( LUT_MAX - LUT_MIN ) )

void main( void )
{
    unsigned char  lut[LUT_SIZE];
    int    i;

    for(i=0; i<LUT_SIZE; i++){
        if( i < LUT_MIN) {
            lut[i] = 0;
        } else if ( i > LUT_MAX) {
            lut[i] = 255;
        } else {
            lut[i] = LUT_VALUE(i);
        }
    }

    for(i=0; i<LUT_SIZE; i++)
        printf("%d¥t%d¥n", i, lut[i] );
}
```

Function to setup a LUT

Evaluated from left to right

Declaration of LUT as an array

Array 5 – Calculating average

```
#include<stdio.h>
#define  MDATA 1000
int   main( void )
{
    double          dat[ MDATA ];
    int             n, i;
    double          total, average;

    printf("Input number of data¥n");
    scanf("%d", & n );
    if( n <= 0 || n > MDATA ) {
        fprintf( stderr, "average: illegal n: %d: must be 1 - %d¥n", n, MDATA);
        return 1;
    }
    for(i=0;i<n;i++) {
        scanf("%lf", &dat[i] );
    }
    total  =0.;
    for(i=0; i<n; i++)
        total += dat[i];

    average = total / n;

    printf("average = %lf¥n", average );

    return 0;
}
```

Pointer

To declare a pointer,

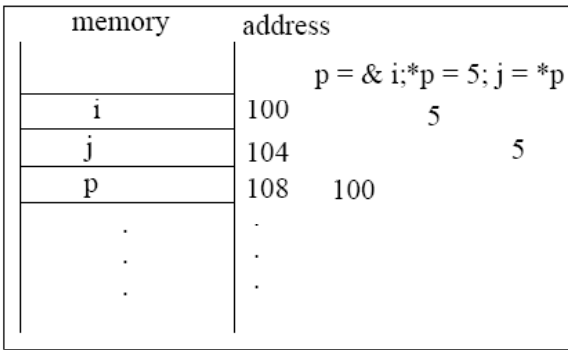
type * pointer_name;

```
int  i , j;
int   * p;          /*  p is a pointer to int. */
                    /*  *p is integer */

/* p has a address of i      */
p = &i;

/* Assign 5 to the integer which p is pointing */
*p = 5;

/* Assign a value of a integer which p is pointing to j, so j is 5 now */
j = *p;
```



7

Pointer & Array 1

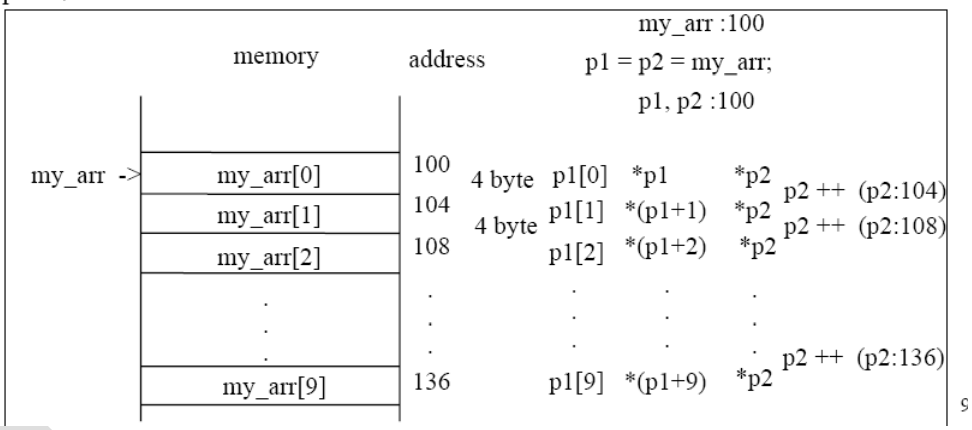
A pointer can represent an array. By using pointer, we can do every operation on arrays.

```
int i;
int my_arr[10];
int *p1, *p2;
for(i=0;i<10;i++)
    my_arr[i] = i*2;
    p1 = p2 = my_arr;
    printf("address is %d\t%d\t%d\n", my_arr, p1, p2);
for( i=0; i<10; i++ ){
    printf("%d\t%d\t%d\t%d\t%d\t%d\n", i,
    my_arr[i], p1[i], *(p1+i), *p2, p2);
    p2++;
}
```

Pointer and Array 2

Accessing to elements using pointers

```
p1 = p2 = my_arr;
printf("address is %d¥t%d¥t%d¥n", my_arr, p1, p2 );
for( i=0; i<10; i++ ){
    printf("%d¥t%d¥t%d¥t%d¥t%d¥t%d¥t%d¥n", i, my_arr[i],  p1[i],  *(p1+i),  *p2, p2 );
    p2++;
}
```



Pointer and Array 3

Passing an array to a function: 3 ways

```
#include<stdio.h>
#define  MDATA 1000
int get_data(double *dat, int n );
double   ave_data(double *dat, int n );
void main( void )
{
    double          x[ MDATA ];
    int             n;
    double          total, average;

    printf("Input number of data¥n");
    scanf("%d", &n );
    get_data( x, n );
    average = ave_data( x, n );
    printf("average = %lf¥n", average );
}
```

```
int     get_data( double dat[ ], int n )
{
        int     i;
        for(i=0;i<n;i++)
                scanf("%lf", & dat[i] );
}
```

```
int     get_data( double  *dat, int n )
{
        int     i;
        for(i=0;i<n;i++)
                scanf("%lf", & dat[i] );
}
```

```
int     get_data( double  *dat, int n )
{
        int     i;
        for(i=0;i<n;i++, dat++)
                scanf("%lf", dat );
}
```

Dynamic Memory Allocation

We cannot know how big the array should be before we run the program.
We do not need to use all of arrays always, which occupy memory space
-> Allocate minimum size of memory block dynamically only when it is necessary.

```
double   *data;
scanf("%d",&ndata)
/* Allocate a memory block */
data = malloc( sizeof(double)*ndata)
.....
/* Free the memory block after use */
free ( data );
```

```
#include <stdlib.h>

void *malloc(size_t size);

void free(void *ptr);
```

malloc( ) allocates size bytes and returns a pointer to the allocated memory. The memory is not cleared.

free( ) frees the memory space pointed to by ptr, which must have been returned by a previous call to malloc(), calloc() or realloc().

11

Dynamic Memory Allocation

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

int  get_data(double *dat, int n );
double   ave_data(double *dat, int n );

main()
{
    double          *x;
    int             n;
    double          average;

    printf("Input number of data¥n");
    scanf("%d", & n );

    x = (double * ) malloc( n * sizeof( double ) );

    if( x == NULL ){
        printf("ave: not enough memory¥n");
        exit(1);
    }

    get_data( x, n );
    average = ave_data( x, n );

    printf("average = %lf¥n", average );

    free( x );
}
```

12

Switching source code according to OS or any other environment

```
#include<stdio.h>
#if defined ( WIN32 ) || defined ( WIN64 )
   #warning being compiled for WINDOWS
   #include<stdlib.h>
   #include<malloc.h>
#elif defined( UNIX )
   #warning being compiled for UNIX
   #include<stdlib.h>
#else
   #error unknown OS
#endif
#define DEBUG
#if defined (DEBUG )
#else
#endif
```


Avoid using include files more than once

```
#include"utilHonda.h“
#include"utilHonda.h"
-----------------------------------------
utilHond.h
-----------------------------------------
#if !defined(UTIL_HONDA_H)
#define UTIL_HONDA_H
#if ! defined ( PI )
#define PI
   3.14159265358979323846264338327950288419716939937511
/ * Proto Types * /
.....
#endif
```

Exercise

- Confirm the output of examples of “Array 3”, “Array 4”, “Array 5”,

“Pointer & Array 1”,

- Add a function ave_data( ) to the example of “Pointer and Array 3”
- Prepare data in a text file and feed to the program.
- Add a function stddev( ) to calculate standard deviation
- Modify the program to allocate memory dynamically for the array

using malloc, like an example in “Dynamic Memory Allocation 2”

Reference:

Assoc.Prof.Dr.HONDA Kiyoshi, Lecture Note .RS and GIS Field of study,School of Engineering and Technology ,AIT Thailand.2005

Thank you for your kind attention