C and Embedded C programming
Week 4 Day 2

Presented by :

Eng. **Mohamed Taha Gabriel**

Array of pointers

- In c we can create an array of pointers each element of the array is a pointer.
  - Syntax: data_type * ptr[number];
  - Ex: int * arrPtr[10]; // arrPtr is an array of 10 pointers to integers
- Example:

```
int x = 10 , y = -20 , z = 99;
int *p[3] = {&x , &y , &z};
printf("&x = %d , &y = %d , &z = %d\n" , &x ,&y , &z);
for (int i = 0; i < 3; i++)
{
   printf("&p[%d] = %d , p[%d] = %d , *p[%d] = %d\n" ,
        i, &p[i] ,i , p[i] , i , *p[i] );
}
```

Array of pointers Cont.

- In c we can create an array of pointers each element of the array is a pointer.
  - Syntax: data_type * ptr[number];
  - Ex: int * arrPtr[10]; // arrPtr is an array of 10 pointers to integers
- Example:

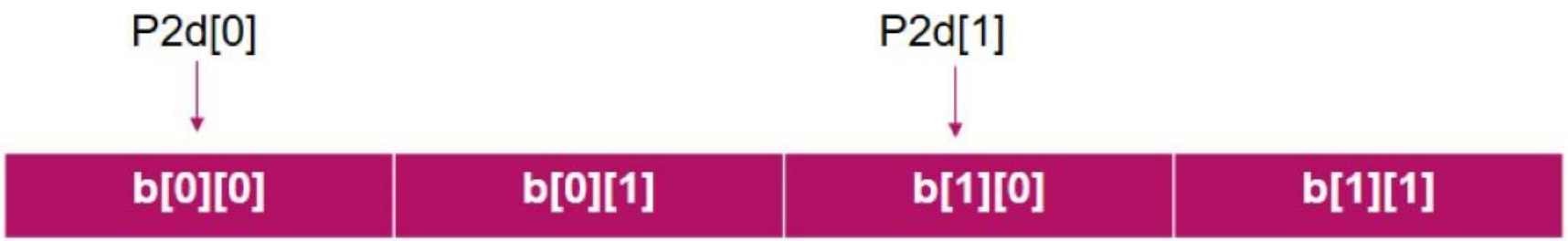
```
int x = 10 , y = -20 , z = 99;
int *p[3] = {&x , &y , &z};
printf("&x = %d , &y = %d , &z = %d\n" , &x ,&y , &z);
for (int i = 0; i < 3; i++)
{
   printf("&p[%d] = %d , p[%d] = %d , *p[%d] = %d\n" ,
        i, &p[i] ,i , p[i] , i , *p[i] );
}
```

```
&x = 6422040 , &y = 6422036 , &z = 6422032
&p[0] = 6422000 , p[0] = 6422040 , *p[0] = 10
&p[1] = 6422008 , p[1] = 6422036 , *p[1] = -20
&p[2] = 6422016 , p[2] = 6422032 , *p[2] = 99
```

Array of pointers Cont.

- Another Example:

```
int b[2][2] = {{0,1},{2,3}};
int *p2d[2] = {&b[0][0],&b[1][0]};
//same as
//int *p2d[2] = {arr[0] , arr[1]} ;
printf("%d,%d\n",p2d[0][0],p2d[1][1]); // output: 0,3
```



Pointer to Function

- The code of a function always resides in memory, which means that the function has some address.
- We can get the address of memory by using the function pointer.

```c
#include <stdio.h>
int main()
{
    printf("Address of main() function is %p",main);
    // 0000000000401710
    return 0;
}
```

- A pointer to function can be used to call the function.
- In Embedded systems pointers to functions mainly used for callback functions and state machines.

Declaration of a function pointer

Syntax of function pointer

```
return_type (*ptr_name)(type1, type2…);
```

Examples:

- **int** (*ip) (**int**); // ip is a pointer to function that takes integer and return integer.
- **float** (*fp) (**int** , **int**); // fp is a pointer to function that takes two integers and return a float

Pointer to function cont.

Assigning a pointer to function

```
float (*fp) (int , int);// Declaration of a function pointer.
float func( int , int );// Declaration of  function.
// Assigning address of func to the fp pointer.
fp = func; //or fp = &func;
```

NOTE: fp and func Arguments and return data type must be the same.

Pointer to function cont.

calling a pointer to function

- Syntax:

```
(*ptr_name)(arguments);
// or
ptr_name(arguments);
```

Example:

```
result = (*fp)( a , b);
//or
result = fp( a , b);
```

Pointer to function examples

- Try this:

```
void func()
{
    printf("hello NTI!!\n");
}

void main()
{
    void (*p)();
    p = func;
    printf("calling the function by its name :\n");
    func();
    printf("calling the function using a pointer :\n");
    p();
}
```

Pointer to function examples cont.

- It can make a function to call another function. (callback function)
- Func1 is used to call other functions.

```
void timer(void (*ptr)());
void LED_ON();
void LED_OFF();
void main()
{
    timer(LED_ON);
    timer(LED_OFF);
}
```

```
void timer(void (*ptr)())
{
    printf("delay 1 second\n");
    ptr(); // OR (*ptr)();
}
void LED_ON()
{
    printf("LED is now on\n");
}
void LED_OFF()
{
    printf("LED is now off\n");
}
```

Pointer to function examples cont.

- Try this:

```c
#include <stdio.h>
#include <math.h>
int add(int x,int y)
{
    return (x+y);
}
int mul(int x,int y)
{
    return (x*y);
}
void main()
{
    int (*p[2])(int ,int ) = {add,mul};

    int res=0;
    for ( int i =0; i < 2; i++)
    {
        res = p[i](4,5);
        printf("res = %d\n",res);
    }
}
```

Pointer to function examples cont.

- Try this:

```c
#include <stdio.h>
#include <math.h>
void cube(float,float *);
void sq_root(float, float*);
int main()
{
    float num=4 , cb, sqr;
    //creating array of pointers to function
    // that takes a float and float pointer and return nothing
    //and assigning it with cube and sq_root functions
    void (*ptr[])(float,float*) = {cube,sq_root};
    ptr[0](num,&cb);
    ptr[1](num,&sqr);
    printf("num = %.2f , cb = %.2f , sqr = %.2f",num,cb,sqr);
}
void cube(float a,float * aaa ){
    *aaa = a*a*a;
}
void sq_root(float a, float*sqa){
    *sqa = sqrt(a);
}
```

Pointer to function examples cont.

-State machine example:

To change from one state to another depend on some event.

```c
#include <stdio.h>
#include <math.h>
void f0(void)
{
    printf("state 0\n");
}

void f1(void)
{
    printf("state 1\n");
}
void f2(void)
{
    printf("state 2\n");
}
```

```c
void main()
{
    int state=0;
    void (*pa[3])(void) = {f0,f1,f2};
    while(getch()!=27)
    {
        switch (state)
        {
        case 0:
            pa[state]();
            state++;
            break;
        case 1:
            pa[state]();
            state++;
            break;
        case 2:
            pa[state]();
            state=0;
            break;
        default:
            printf("beck to state 0\n");
            state = 0;
            break;
        }
    }
}
```

Pointer to function examples cont.

-State machine example:

To change from one state to another depend on some event.

```c
#include <stdio.h>
#include <math.h>
void f0(void)
{
    printf("state 0\n");
}

void f1(void)
{
    printf("state 1\n");
}
void f2(void)
{
    printf("state 2\n");
}
```

-State machine example:

Other solution using array of pointer to functions

```c
void main()
{
    int state = 0;
    void (*p[3])(void)={f0,f1,f2};
    while (getch()!=27)
    {
        p[state]();
        state = (state+1)%3;
    }
}
```

Pointer to function examples cont.

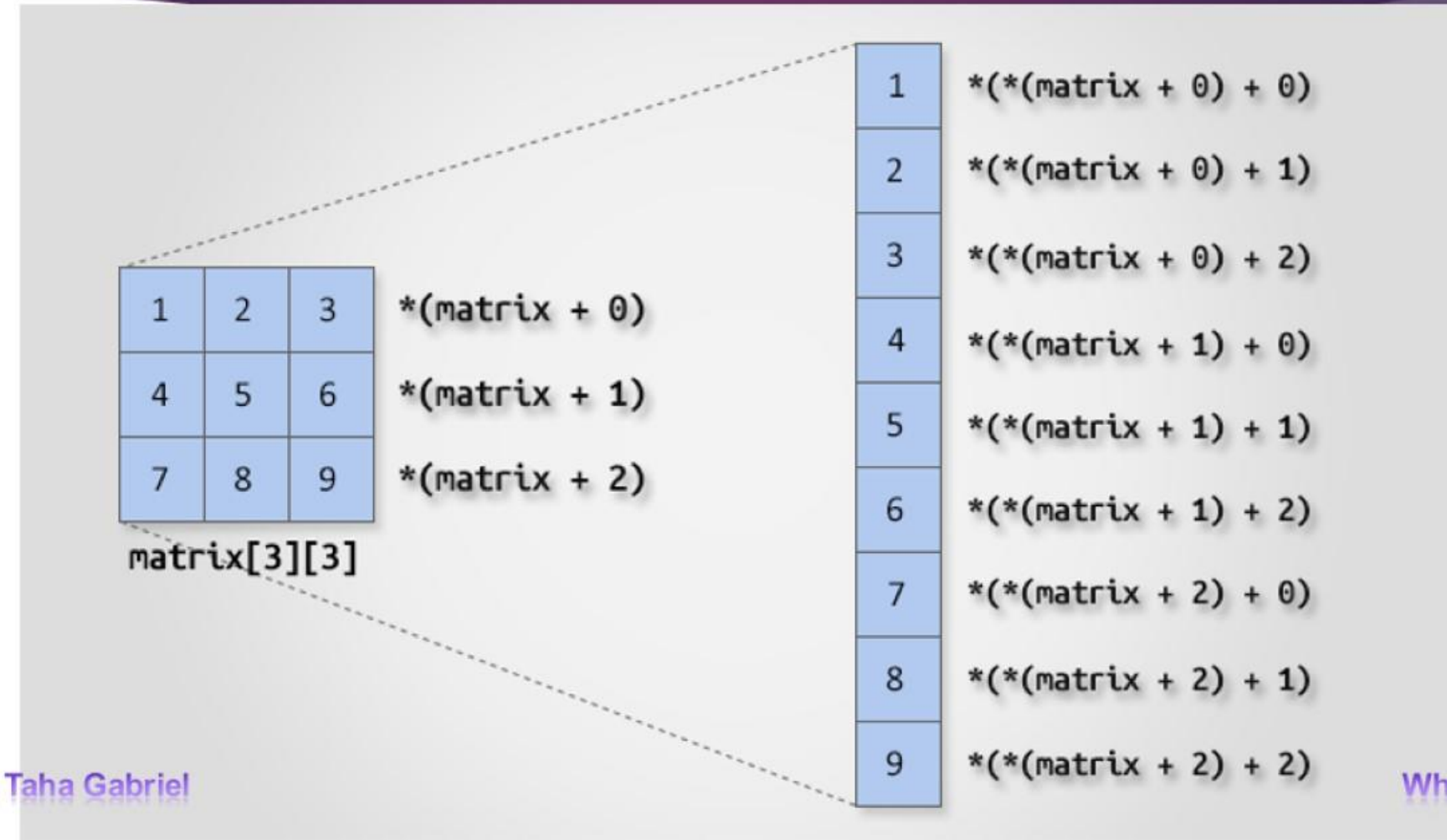
-Menu driven application example:

To calculate area or circumference

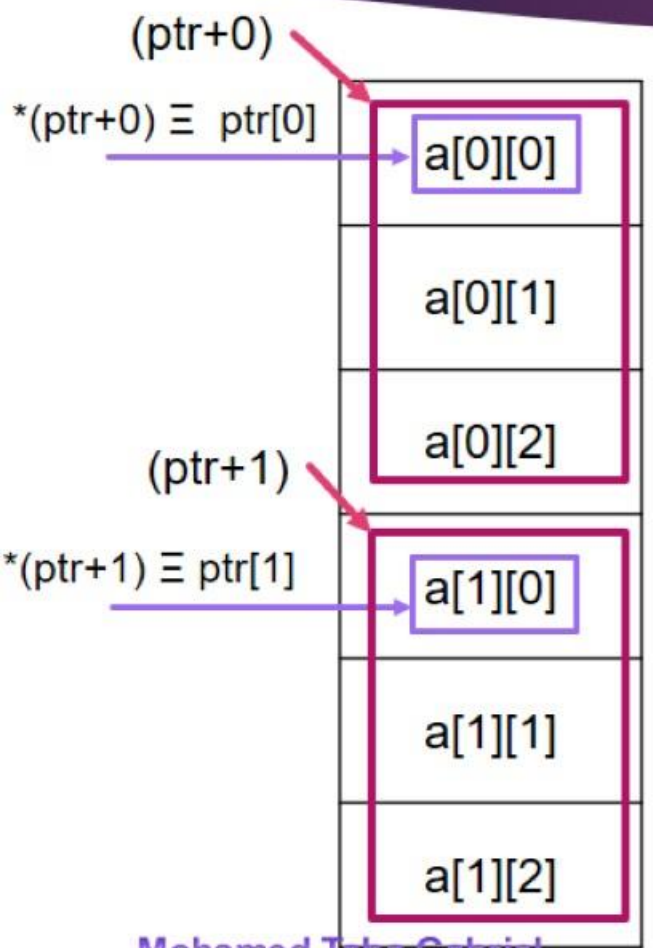
```c
#include <stdio.h>
#include <math.h>
float circ(int r){
    float result = 2*3.14*r;
    return result;
}
float area(int r){
    float result = 3.14*r*r;
    return result;
}
```

```c
void main()
{
    int r,op;
    float result;
    float (*p2f[])(int)={circ,area};
    printf("enter radius\n 1.for area.\n2.for circumference.\n");
    scanf("%d %d",&r,&op);
    switch (op)
    {
    case 1:
        result = p2f[0](r);
        printf("circumference = %f",result);
        break;
    case 2:
        result = p2f[1](r);
        printf("area = %f",result);
        break;
    default:
        printf("invalid input");
        break;
    }
}
```

Pointers to 2D arrays

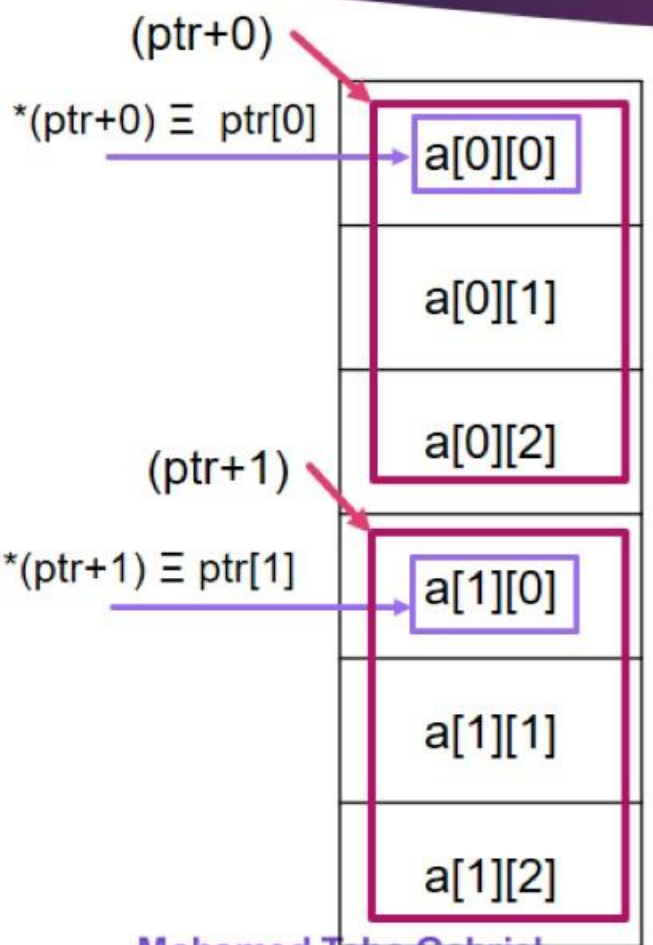


Pointing at 2D array using a pointer to 1D array




```c
#include<stdio.h>
void main ()
{
    int a[2][3]={{1,2,5},{3,4,6}}; //2D array
    int (*ptr)[3]; // pointer to 1D array declaration
    ptr =a; //assigning the pointer to the array
    ptr[0][1] = 10;
    printf("%d\n",ptr[0][1] ); // output: 10
     printf("%d\n",a[0][1] ); // output: 10
}
```

Pointing at 2D array using a pointer to 1D array cont.




```c
#include<stdio.h>
void main ()
{
    int a[2][3]={{1,2,3},{4,5,6}};
    int (*p)[3] = a;
    printf("p+0 = %d , p+1 = %d\n",p,p+1);
    printf("p[0] = %d , p[0]+1 = %d\n",p[0],p[0]+1);
    printf("p[1] = %d , p[1]+1 = %d\n",p[1],p[1]+1);
}
```

Output:

p+0 = 6421968 , p+1 = 6421980
p[0] = 6421968 , p[0]+1 = 6421972
p[1] = 6421980 , p[1]+1 = 6421984

Pointers to 2D arrays

- A pointer to 2D array :

```
int a[2][2]={{1,2},{3,4}}; //2D array
int (*ptr)[2][2]; // pointer to 2D array declaration
ptr =&a; //assigning the pointer to the array
printf("%d\n",(*ptr)[0][1]); // ouptue: 2
```

Passing a 2D array to function

- Example:

```c
#include <stdio.h>
void fnc(int p[][2],int rows,int cols);
void main()
{
    int a[2][2]={{1,2},{3,4}};
    fnc(a,2,2);
}
void fnc(int p[][2],int rows,int cols)
{
    for (int i = 0;i<rows;i++)
    {
        for(int j = 0;j<cols;j++)
        {
            printf("%d ",p[i][j]);
        }
        printf("\n");
    }
}
```

Passing a 2D array to function

- Example:

```c
#include <stdio.h>
void fnc(int p[][2],int rows,int cols);
void main()
{
    int a[2][2]={{1,2},{3,4}};
    fnc(a,2,2);
}
void fnc(int p[][2],int rows,int cols)
{
    for (int i = 0;i<rows;i++)
    {
        for(int j = 0;j<cols;j++)
        {
            printf("%d ",p[i][j]);
        }
        printf("\n");
    }
}
```

Note: The number of columns must be defined. And be the same as the passed array.

Passing a 2D array to function

- Another way:
- Since int p[][2] is the same as int(*p)[2].

```c
#include <stdio.h>
void fnc(int(*p)[2],int rows,int cols);
void main()
{
    int a[2][2]={{1,2},{3,4}};
    fnc(a,2,2);
}
void fnc(int(*p)[2],int rows,int cols)
{
    for (int i = 0;i<rows;i++)
    {
        for(int j = 0;j<cols;j++)
        {
            printf("%d ",p[i][j]);
        }
        printf("\n");
    }
}
```

Passing a 2D array to function

- Another indirect way:
- Using array of pointers.
- Each pointers points at the first element in each raw of the 2D array.
- Then pass the array name and the size of the 2D array.
- This method will not be required specifying the number of columns in the function.

```c
#include <stdio.h>
void arr2D_print(int **p , int rows , int cols);
void main()
{
    int arr[2][3] = {{1,2,3},{4,5,6}};
    int *ap[2] = {arr[0] , arr[1]} ;
    //same as
    //int *ap[2] = {&arr[0][0] , &arr[1][0]} ;
    arr2D_print(ap , 2,3);
}
void arr2D_print(int **p , int rows , int cols)
{
    for(int i = 0;i<rows ; i++)
    {
        for(int j =0;j<cols;j++)
        {
            printf("%d ",p[i][j]);
        }
        printf("\n");
    }
}
```

Wild pointer

It's a pointer that is not initialized or initialized with garbage value.

Example:

```
void main()
{
    int*wptr;
    printf("%d",wptr);
    *wptr = 10;
    printf("%d",*wptr);
}
```

Wild pointers are very dangerous. Using this pointer may cause a segmentation (runtime error) as it stores a garbage value as an address.

Null pointer

It is best practice to initialize a pointer when declaring it to avoid any unwanted change or access to the memory also to avoid the wild pointer.

If there is no variable to be pointed at we can initialize the pointer to zero or NULL.

A NULL pointer is considered a pointer that is not point at anything.

```
int*ptr = NULL;
```

Null pointer

- Don’t dereference a pointer in case it might be NULL.
- First you must check that it’s not a NULL pointer.

```c
#include <stdio.h>
int main()
{
    int *ptr=NULL;
    if(ptr!=NULL)
    {
        printf("value of ptr is : %d",*ptr);
    }
    else
    {
        printf("Invalid pointer");
    }
  return 0;
}
```

Dangling pointer

A pointer pointing to a memory location that has been deleted (or freed) is called dangling pointer.

There are three different ways where Pointer acts as dangling pointer.

- De-allocation of memory
- Variable goes out of scope
- Function Call

Dangling pointer examples

```c
int *fun()
{
    // x is local variable and goes out of
    // scope after an execution of fun() is
    // over.
    int x = 5;
    return &x;
}
int main()
{
    int *p=NULL;
     p = fun();
    // p points to something which is not
    // valid anymore
    printf("%d", *p);
    return 0;
}
```

```c
int main()
{
    int *p;
    if (1)
    {
        int x=5;
        p = &x;
    }
  // p points to something which is not
    // valid anymore
    printf("%d", *p);
    return 0;
}
```

Void pointer

- Void pointer is a specific pointer type – void * – a pointer that points to some data location in storage, which doesn't have any specific type.
- void pointers **cannot be dereferenced without typecasting.**
- So, it must be typecasted before dereferencing.
- Pointer arithmetic is not possible on pointers of void due to lack of concrete value and thus size of the pointed type.
- example:

```
int x=0xffff5555;
void *vp = &x;
printf("vp ch=%x \n vp int = %x \n", *(char*)vp, *(int*)vp);
```

Void pointer example

```c
#include <stdio.h>
void main()
{
    int x=0x44332211;
    void * vp;
    vp = &x;
    printf("\nvp ch=%x\nvp int = %x\n",*((char*)vp+2),*(int*)vp);
    printf("\nvp ch address = %x\nvp int address = %x\n",(char*)vp+2,(int*)vp);
    printf("\nvp ch address = %x\nvp int address = %x\n",(char*)vp+1,(int*)vp+1);
}
```

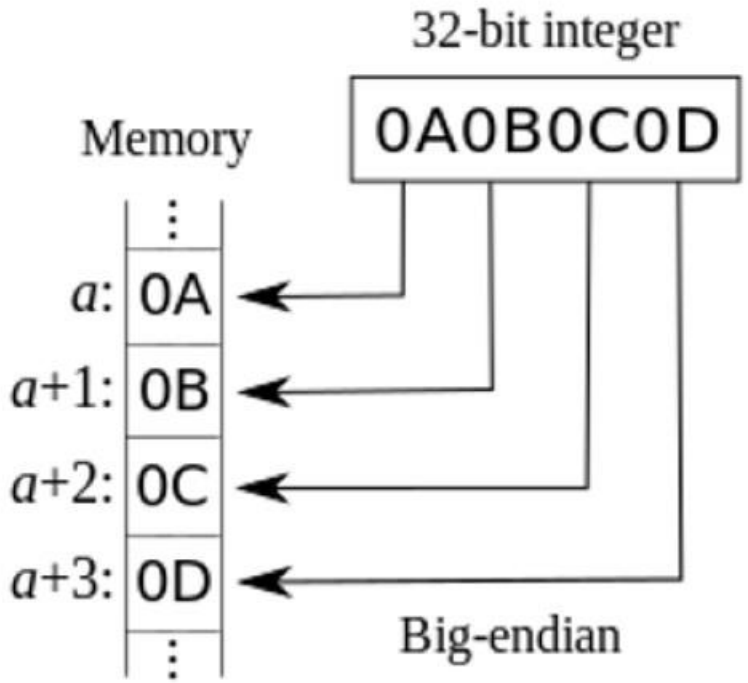
```
vp ch=33
vp int = 44332211

vp ch address = 61fde6
vp int address = 61fde4

vp ch address = 61fde5
vp int address = 61fde8
```

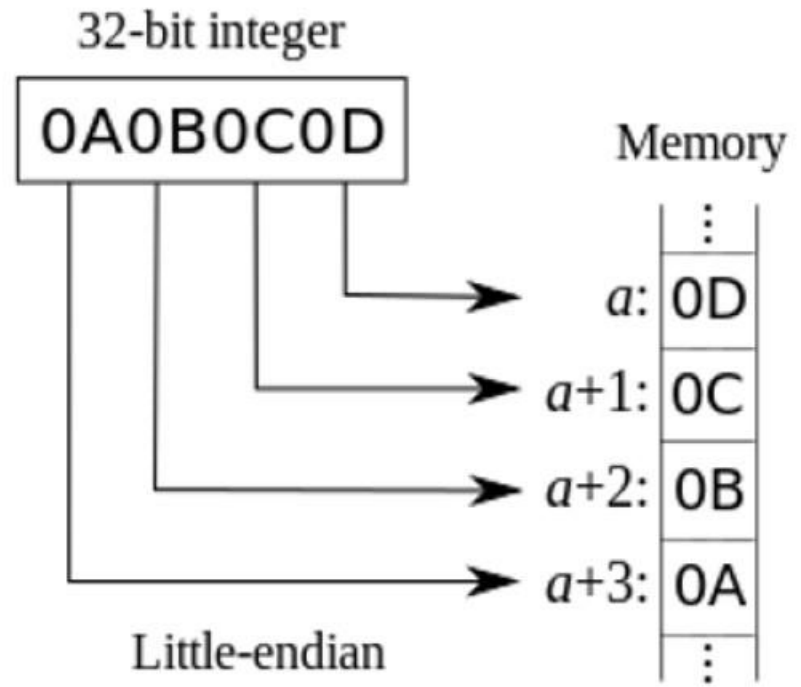
Big endian vs little endian

the big endian or small endian are two ways of storing bytes of data e.g. 32-bit integer in memory locations.

For Big Endian Representations, the Most Significant Byte (MSB) is stored at lower addresses.

For Little Endians, it is the opposite, the MSB is stored at higher addresses.





Labs

1. How to read the pointer: int (*p)(int (*)[2], int (*)(void))
2. Write a c program of a simple calculator that sum, subtract, multiply or divide two variables using pointer to functions.
3. Write a function that print a 2D array 2 *3 and return maximum, minimum and average value of its elements.
4. Write a function that gives the length of a string.
5. Write a c program to check if the memory sorting is big endian or little endian.