C and Embedded C programming
Week 4 Day 1

Presented by :

Eng. **Mohamed Taha Gabriel**

Pointers in C

Mohamed Taha Gabriel
WhatsApp:
01004276101

Outlines

- What is a pointer?
- Why do we use pointers?
- Pointer declaration and definition.
- Pointer size
- Call by reference and call by value
- Pointer types
- Dangling pointer, null pointer, wild and void pointer
- Pointer to array
- Array of pointers
- Pointer to pointer
- Pointer to functions

What is a pointer?

The pointer in C language is a variable which stores the address of another variable.

This **pointed** variable can be of type int, char, array, function, even pointers or any other type.

Pointers are powerful features of C and C++ programming. As it allows low level memory accessing and manipulation.

Pointer declaration

- Syntax:

```
data_type * pointer_name;
```

  - The asterisk (*) is used to declare the pointer.
  - data_type: it's the data type of the pointed variable

- Example:

```
int x;        // declaring x as an integer variable
int * ptr2int; //declaring ptr2int as pointer to integer
ptr2int = &x;   //referencing the address of x to prt2int
```

```
//or
int x;
int *ptr2int = &x;
```

```
//or
int x,*ptr2int =&x
```

Pointer declaration

- Syntax:

```
data_type * pointer_name;
```

  - The asterisk (*) is used to declare the pointer.
  - data_type: it's the data type of the pointed variable

- Example:

```
int x;       // declaring x as an integer variable
int * ptr2int; //declaring ptr2int as pointer to integer
ptr2int = &x;   //referencing the address of x to prt2int
```

```
//or
int x;
int *ptr2int = &x;
```

```
//or
int x,*ptr2int =&x
```

ptr2int → 0x00f2

variable	Address	Data
ch	0x00f0	'g'
	0x00f1	0f
x	0x00f2	05
	0x00f3	02
	0x00f4	06
	0x00f5	00
	0x00f6	
ptr2int	0x00f7	f2
	0x00f8	00
	0x00f9	00
	0x00fa	00

Mohamed Taha Gabriel
WhatsApp: 01004276101

Pointer declaration cont.

- Other examples:

```
char ch ;
float fl;
short sh = 10;
```

```
char * ptr2ch = &ch;
float * ptr2fl = &fl;
short * ptr2sh = &sh;
```

	variable	Address	Data
ptr2ch → 0x00f0	ch	0x00f0	'g'
		0x00f1	f
ptr2fl → 0x00f2	fl	0x00f2	5
		0x00f3	f2
		0x00f4	ff
		0x00f5	ff
		0x00f6	
ptr2sh → 0x00f7	sh	0x00f7	0a
		0x00f8	00
		0x00f9	
		0x00fa	

Pointer size

Since pointer stores addresses, the size of a pointer is dependent only on the size of the address bus.

Its size does not depend on the pointed variable data type.

For example:

- In a 64-bit system the address consists of 64 bits means the pointer size = 8 bytes.
- In a 32-bit system the address consist of 32 bit means the pointer size = 4 bytes.

Pointer size example

```
#include <stdio.h>
void main()
{
    double x = 2.3;
    char ch ='a';
    double * p2dbl = &x;
    char * p2ch = &ch;
    printf("size of char = %d , size of ptr to char = %d\n”,
       sizeof(ch), sizeof(p2ch));
    printf("size of double = %d , size of ptr to double =%d\n”,
                                sizeof(x),sizeof(p2dbl));
}
```

- Output:

size of char = 1 , size of ptr to char = 8

size of double = 8 , size of ptr to double = 8

Notes

&var

- & (ampersand)
- it gives the variable **var** memory address

pointer = &var;

- This is called **referencing** var to pointer.
- That assigns the address of var to the pointer

* pointer

- **Dereference** operator *
- accessing the value stored at the address stored in the pointer.

* pointer = value;

- Assigns value at the address stored in the pointer.

Var2 = *pointer

- Assigns the value in the address in the pointer to Var2

Some Famous Errors

```
int c, *pc;

// pc is address but c is not
pc = c;  // Error

// &c is address but *pc is not
*pc = &c;  // Error

// both &c and pc are addresses
pc = &c;  // Not an error

// both c and *pc are values
*pc = c;  // Not an error
```

Notes

```
int * p1,p2;
```

- This declare p1 as a pointer and p2 as a normal integer variable.

```
int *p1,*p2;
```

- This declares p1 and p2 as pointers to integers.

Examples

- Try this:

```
#include <stdio.h>
void main()
{
    int x,*ptr = &x;
    *ptr = 10;
    printf("&x = %x , x = %d\n",&x,x);
    printf("ptr = %x , *ptr = %d\n",ptr,*ptr);
}
```

Examples

- Try this:

```
#include <stdio.h>
void main()
{
    int x,*ptr = &x;
    *ptr = 10;
    printf("&x = %x , x = %d\n",&x,x);
    printf("ptr = %x , *ptr = %d\n",ptr,*ptr);
}
```

output
&x = 61fde4 , x = 10
ptr = 61fde4 , *ptr = 10

Why do we use pointers?

The main reason for using pointer is accessing any memory location of (variable, array, function, data structure ..etc) without copying it.

The pointers transcends the scope. If we have the memory address of any (variable, array, function, data structure ..etc) , we can access it and change it even if the variable is not in your scope.

Advantages of pointers

Pointer reduces the code and improves the performance.

We can return multiple values from a function using pointers.

It makes you able to access any memory location in the computer's memory.

Dynamic memory allocation

- In c language, we can dynamically allocate memory using malloc() and calloc() functions where the pointers is used to store the location.

Mohamed Taha Gabriel

Pionters with Arrays, Functions, and Structures

- Pointers in c language are widely used in arrays, functions, and structures. It reduces the code and improves the performance.

Used with data structures like linked lists, queues, staks..etc

WhatsApp: 01004276101

Disadvantages of pointers

we can access restricted memory area.(segmentation fault).

A miss-use of pointers with data structures can change or ruin data permanently if used unproperly.

Pointers concepts

Concept	Description
Pointer arithmetic	There are four arithmetic operators that can be used in pointers: ++, --, +, -
Array of pointers	You can define arrays to hold a number of pointers.
Pointer to pointer	C allows you to have pointer on a pointer and so on.
Passing address to functions in C	Passing an argument by reference or by address enable the passed argument to be changed in the calling function by the called function.
Return address from functions in C	C allows a function to return a pointer to the local variable, static variable, and dynamically allocated memory as well.

Pointer arithmetic

There are four arithmetic operators that can be used in pointers:

- Addition +
- Increment ++
- Subtraction –
- Decrement --

Pointer arithmetic: addition

- Adding a number to a pointer.
  - Ex:

```
int a[]={0,1,2,3};
int *p1 = &a[0]; // or just a
p1 = p1+1; // now p1 is pointed at a[1]
```

- It depends on the size of the datatype the pointer points at.
  - In this example int is 4 bytes.
  - So, incrementing p1 which is a ptr to integer increase 4 bytes.

p1 → 0x00f2

P1+1 → 0x00f6

variable	Address	Data
	0x00f0	'g'
	0x00f1	0f
a[0]	0x00f2	00
	0x00f3	00
	0x00f4	00
	0x00f5	00
a[1]	0x00f6	01
	0x00f7	00
	0x00f8	00
	0x00f9	00
a[2]	0x00fa	02

Pointer arithmetic: addition cont.

- It depend on the size of the data type the pointer is pointed at.
  - In this example int is 4 bytes.
  - So incrementing p1 which is a ptr to integer increase 4 bytes.
  - Try this:

```
short x=0,y=2;
short *p1 = &x ,*p2=&y;
printf("%x , %x",p1,p1+1);
```

- Adding two pointers is not allowed:

```
int x=0,y=2;
int *p1 = &x ,*p2=&y;
unsigned long int z = p1+p2; // this is error
```

Pointer arithmetic: subtraction

- Same as increment we can decrement a constant from a pointer

```
int a[]={0,1,2,3};
int *p1 = &a[2];
p1 = p1-2; // now p1 is pointed at a[0]
printf("%d",*p1); // output: 0
```

- It depend on the size of the data type the pointer is pointed at.
- The difference is that subtracting two pointers (points at the same datatype) is allowed
  -

```
int a[]={0,1,2,3};
int *p1 = &a[2];
int *p2 = a;
printf("%d",p1-p2); // output: 2
```

Pointer arithmetic: increment

```
++ptr;
```

- Equals to: ptr +=1;

```
ptr2  =  ++ptr1;
```

- Equals to: ptr1 += 1; // ptr1 = ptr1+1
  - ptr2 = ptr1;

```
ptr2 = ptr1++;
```

- Equals to: ptr2 = ptr1;
  - ptr1 += 1;

Pointer arithmetic: decrement

```
--ptr;
```

- Equals to: ptr -=1;

```
ptr2  =  --ptr1;
```

- Equals to: ptr1 -= 1;
  - ptr2 = ptr1;

```
ptr2 = ptr1--;
```

- Equals to: ptr2 = ptr1;
  - ptr1 -= 1;

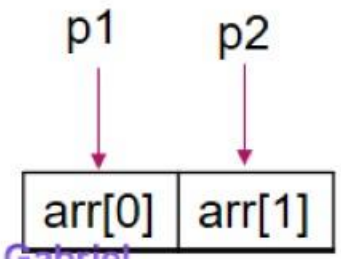
Notes

- Increment and decrement (++ and --) changes the address what the pointer is pointing at.
- Famous Error:
  - For ex:

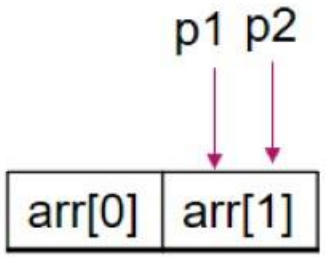
```
int arr[2]={0,1} , *p1, *p2;
```

  - If you want p1 to point at arr[0] and p2 to point at arr[1]:

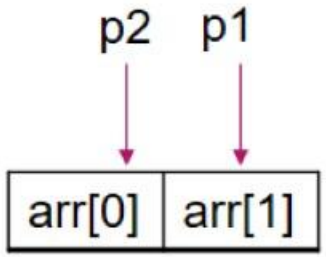
```
p1 = arr;
p2 = p1+1;
```




```
p1 = arr;
p2 = ++p1;
```




```
p1 = arr;
p2 = p1++;
```



Call by value and call by reference

Call type	Description
Call by value	This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.
Call by reference	This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

Call by value and call by reference cont.

No.	Call by value	Call by reference
1	A copy of the value is passed into the function	An address of value is passed into the function
2	Changes made inside the function is limited to the function only. The values of the actual parameters do not change by changing the formal parameters.	Changes made inside the function validate outside of the function also. The values of the actual parameters do change by changing the formal parameters.
3	Actual and formal arguments are created at the different memory location	Actual and formal arguments are created at the same memory location
Ex	pritnf()	scanf()

Call by reference

We use call by reference when we want to return more than one value.

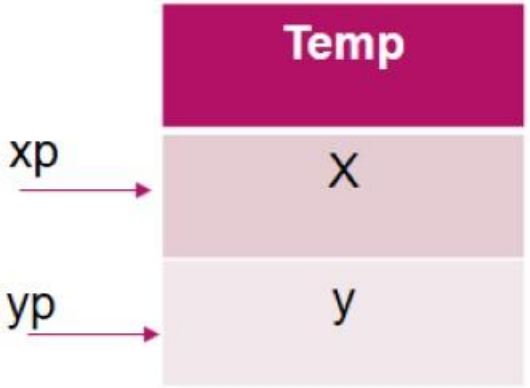
It does not copy actual value. That is faster and more memory efficient.

Call by reference example

- Swap the value of two variables using a swap function.

Call by reference example

- Swap the value of two variables using a swap function.




```c
void swap(int *, int *);
int main()
{
    int x, y;
    printf("Enter Value of x ");
    scanf("%d", &x);
    printf("\nEnter Value of y ");
    scanf("%d", &y);
    swap(&x, &y);
    printf("\nAfter Swapping: x = %d, y = %d", x, y);
    return 0;
}
void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}
```

Call by reference exercise

- Build a function that takes the value of the number and return its square, cube and the square root.

Call by reference exercise

- Build a function that takes the value of the number and return its square, cube and the square root.

```
void sq_cb_sqRoot(float x , float * px2 , float * px3 , float * pSq_x ,float * pMul3)
{
    *px2 = x*x;
    *px3 = x*x*x;
    *pSq_x = sqrt(x);
    *pMul3 = 3*x;
}
```

Pointers and Arrays

The array name is the base address of the array i.e., the address of the first element of the array.

Assigning the array address to a pointer enable us to use the pointer instead of the array name.

This also enables us to pass arrays to functions without coping all the data of the array.

That will reduce the size of the stack used for function calling.

In C:

```
arr[i] is the same as *(arr+i)
arr[]  is the same as *(arr)
p[]    is the same as *(p)
p[i]   is the same as *(p+i)
```

Pointers and Arrays

- You can assign a pointer to the base address of the array and use it as an array:
  - Ex:

```
int a[]={0,1,2,3};
int *p = a; // same as &a[0]
printf("%d,%d\n",p[0],p[1]); //output: 0,1
p[0] = 9; //you can use this to modify the value
*(p+1) = 5; // or this syntax
printf("%d,%d\n",p[0],p[1]); //output: 9,5
```

Pointers and Arrays exercise

- What will be the output of the program:

```
#include<stdio.h>
void main()
{
    int *p = NULL;
    int arr[5] = {1,2,3,4,5};
    p = arr;
    printf("size of P = %d, size of arr = %d\n",sizeof(p),sizeof(arr));
    printf("%d , %d\n",p,arr);
    printf("%d , %d \n",p[0], *(arr+0));
    printf("%d , %d \n",*(arr+1),arr[1]);
    printf("%d , %d \n",p[1], *(p+1));
}
```

Pointers and Arrays exercise

- What will be the output of the program:

```
#include<stdio.h>
void main()
{
    int *p = NULL;
    int arr[5] = {1,2,3,4,5};
    p = arr;
    printf("size of P = %d, size of arr = %d\n",sizeof(p),sizeof(arr));
    //size of P = 8, size of arr = 20
    printf("%d , %d\n",p,arr);//6421968 , 6421968
    printf("%d , %d \n",p[0], *(arr+0));//1 , 1
    printf("%d , %d \n",*(arr+1),arr[1]);//2 , 2
    printf("%d , %d \n",p[1], *(p+1));//2 , 2
}
```

Passing array 1D to function

- Syntax:

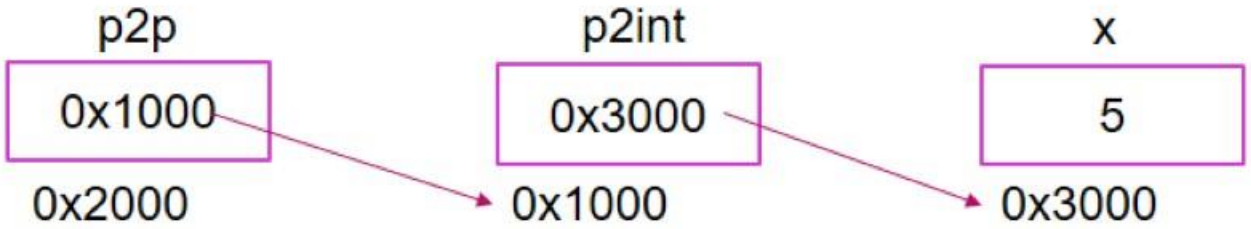
```
return_type function(type *array_name , type size)
```

- Example:

```
#include <stdio.h>
void print_arr(int *a,int size);
void main()
{
    int arr[]={1,2,3,4,5}; // array of 5 integers
    print_arr(arr,5); // only the array address and size
}
void print_arr(int * a,int size)
{
    for (int i =0;i<size;i++)
    {
        printf("arr[%d] = %d\n",i,a[i]);
    }
}
```

Pointer to pointer

- It's a pointer that holds the address of another pointer.



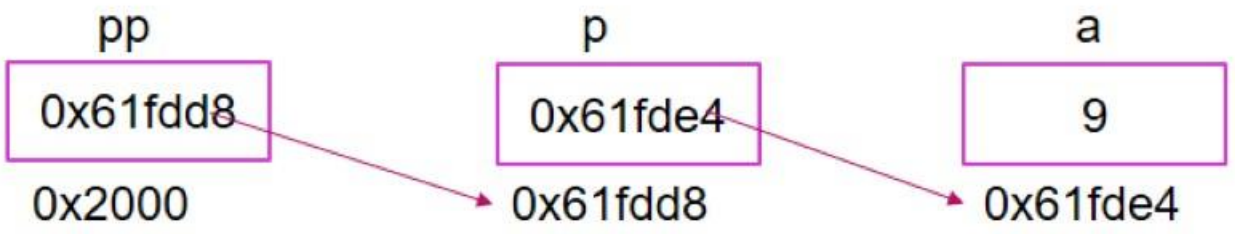
- Syntax:

```
data_type ** p2p;
```

Pointer to pointer example

```
#include<stdio.h>
void main ()
{
    int a = 9;
    int *p;
    int **pp;
    p = &a; // pointer p is pointing to the address of a
    pp = &p; // pointer pp is a double pointer pointing to the address of pointer p
    // Address of a will be printed
    printf("address of a: %x\n",p);
   // Address of p will be printed
    printf("address of p: %x\n",pp);
    // value stored at the address contained by p i.e. 9 will be printed
    printf("value stored at p: %d\n",*p);
    // value stored at the address contained by the pointer stored at pp
    printf("value stored at pp: %d\n",**pp);
}
```

Pointer to pointer example



Note

```
int * ptr1 [10];
```

- ptr1 is an array of 10 pointers to integers.

```
Int (*ptr2)[10];
```

- ptr2 is a pointer to array of 10 integers

```
int (*ptr2function)(int , int );
```

- This is a pointer to function that takes 2 integers and return an integer.

```
int *function(int , int );
```

- This a function that takes 2 integers and return pointer to integer.

Reading complex pointers

- the precedence and associativity of the operators which are used regarding pointers.
- (): This operator is a bracket operator used to declare and define the function.
- []: This operator is an array subscript operator
- * : This operator is a pointer operator.
- Identifier: It is the name of the pointer. The priority will always be assigned to this.
- Data type: Data type is the type of the variable to which the pointer is intended to point. It also includes the modifier like signed int, long, etc.).

Operator	Precedence	Associativity
identifier, ()	1	Left to right
[], *	2	Left to right
Data type	3	-

Pointer naming exercise

```
int *p2int;

int **p2p2int;

char* p1(float , int);

char (*p2) (float,int);

char* (*p2) (float,int);

float* p3[10];

float (*p4)[10];

int (*a[10])(int );
```

Pointer naming exercise

```
int *p2int; // pointer to integer

int **p2p2int; // pointer to pointer to integer

char* p1(float, int); // function takes float and int and returns a pointer to char

char (*p2) (float, int); // pointer to function that takes float and int and returns a char

char *(*p2) (float, int); // pointer to function that takes float and int and returns a pointer char

float* p3[10]; // array of 10 pointers to float

float (*p4)[10]; // pointer to an array of 10 floats

int (*a[10])(int ); // a is an array of 10 pointers to function that takes int and return and int
```

Labs

1. Write a function that takes the radius of a circle and return its area and circumference.
   - $Area = \pi r^2$, $circ = 2\pi r$
2. Write a function that takes an array of integers (1D) and return its max and minimum element also the average of the array.
3. How to read this pointer? ➔ int (*p)(int (*)[2], int (*)(void))