Pointer

Jinwoo Choi

Pointer

pointer

noun [C]

UK /ˈpɔɪn.tər/ US /ˈpɔɪn.ɾɚ/

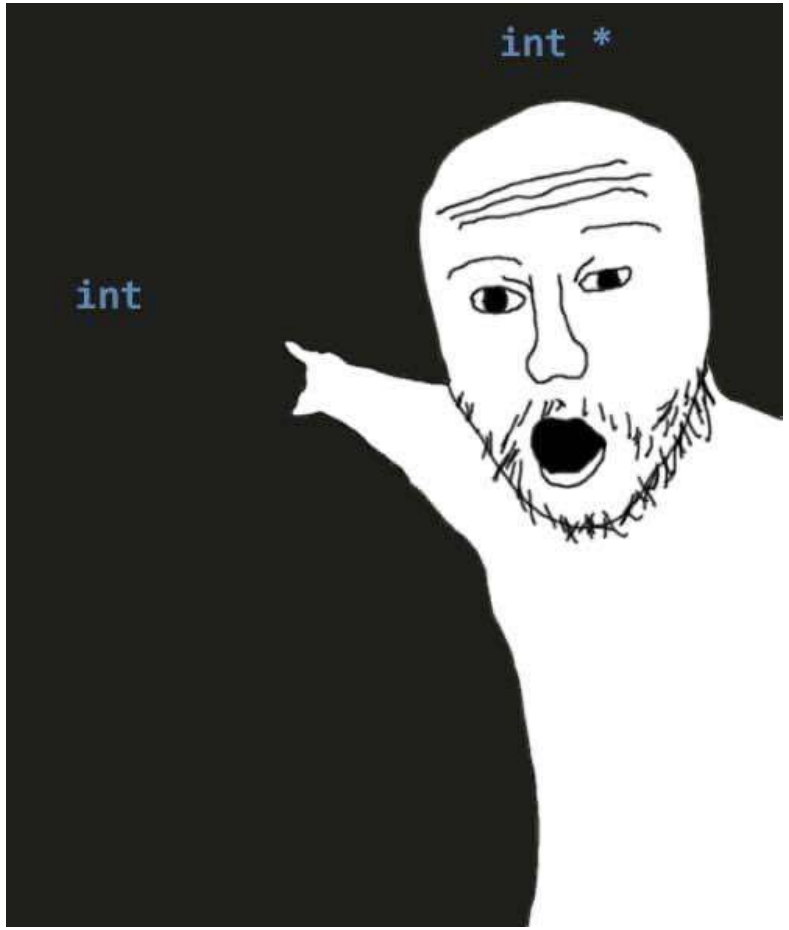
pointer *noun* [C] (STICK)

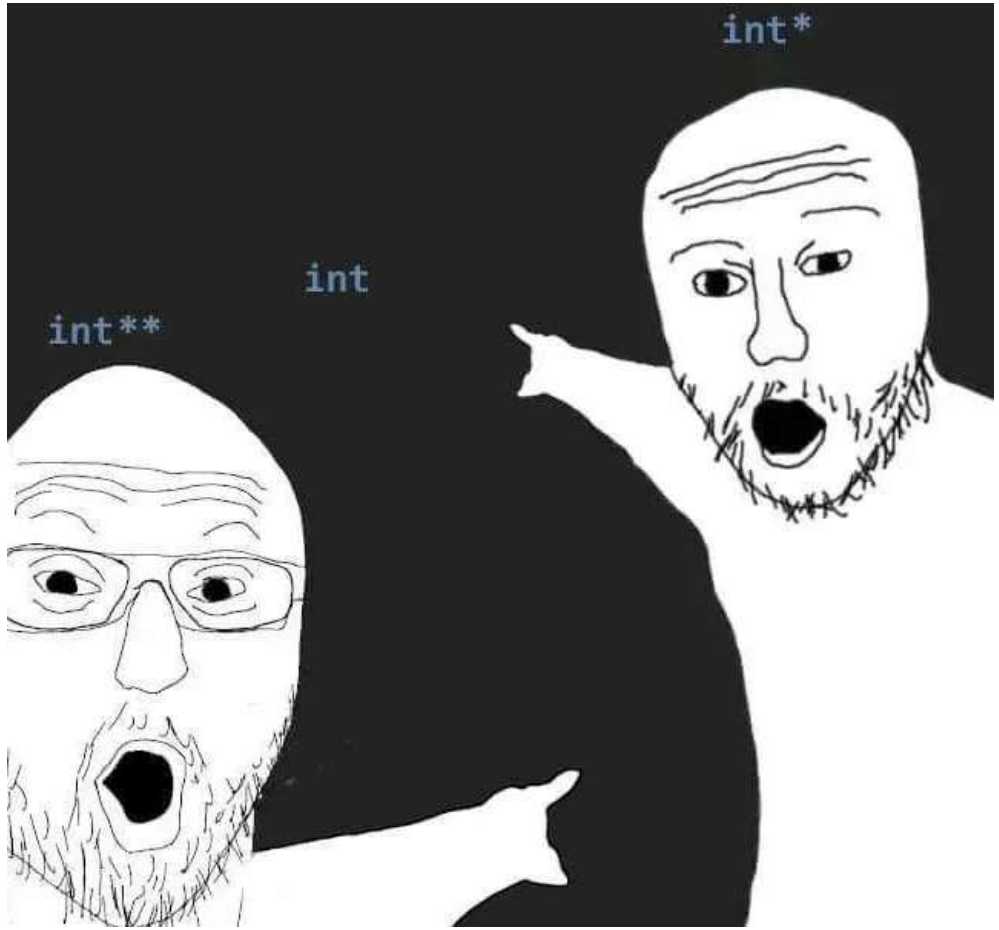
Add to word list

Ljupco/iStock/Getty Images Plus/GettyImages

something that is used for pointing at things, such as a long, thin stick that you hold to direct attention to a place on a map or words on a board, or a cursor

Pointer





Pointer

<Local Variables>

```
int x = ???; // 4 Byte
char c = '?'; // 1 Byte
int y = ???; // 4 Byte
```

1 Byte

Address	Memory
0x10	01111010
0x11	11100010
0x12	00000111
0x13	01011101
0x14	00000101
0x15	
0x16	
0x17	
0x18	11111110
0x19	01100010
0x1A	11100111
0x1B	01011101
0x1C	
0x1D	
0x1E	
0x1F	
0x20	
0x21	
0x22	

Local variables are typically stored in stack memory.

Each local variable's address is managed internally.
(Each local variable's address is determined at runtime using the stack pointer(ESP/RSP) and base pointer(EBP/RBP))

To check where local variable is stored, we use pointer to get its memory address

Pointer

<Local Variables>

```
int x = ???;
char c = '?';
int y = ???;
```

Address	Memory
0x10	01111010
0x11	11100010
0x12	00000111
0x13	01011101
0x14	00000101
0x15	
0x16	
0x17	
0x18	11111110
0x19	01100010
0x1A	11100111
0x1B	01011101
0x1C	
0x1D	
0x1E	
0x1F	
0x20	
0x21	
0x22	

1 Byte

```
type * pointerName;

int * p_i;

char * p_c;
```

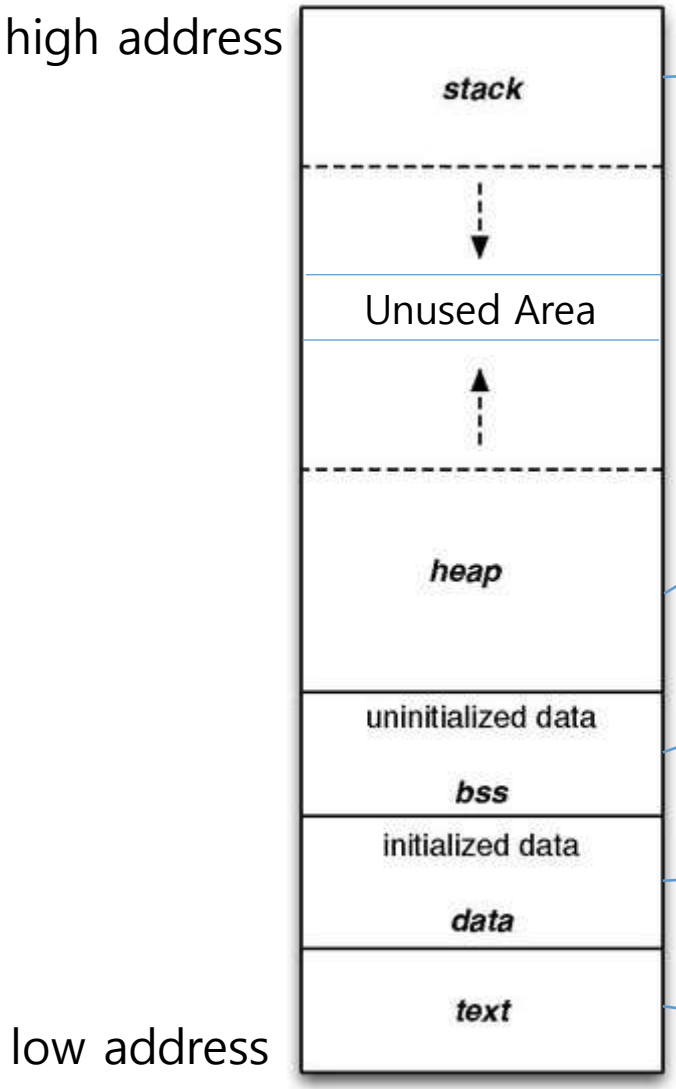
Why do we need pointer?

```
int MultiplyXY(int X, int Y)
{
    int result_func = X * Y;
    return result_func;
}

int main(void)
{
    int a = 9, b = 6;
    int result_main = MultiplyXY(a, b);
    //print
    return 0;
}
```

Address	Memory
0x10	00001001
0x11	00000000
0x12	00000000
0x13	00000000
0x14	
0x15	
0x16	
0x17	
0x18	00001001
0x19	00000000
0x1A	00000000
0x1B	00000000
0x1C	
0x1D	
0x1E	
0x1F	
0x20	
0x21	
0x22	

Why do we need pointer?



Stores Local Variable, Parameter
Grows downwards ↓ (high address -> low address)

Stores **Dynamically allocated memory**(malloc(), new)
Grows upwards ↑ (low address -> high address)

Stores **[Uninitialized]** Global Variables, Static Variables
Block **S**tarted by **S**ymbol => Uninitialized Data Segment

Stores **[Initialized]** Global Variables, Static Variables
Initialized Data Segment

Executable Code(function, loop...etc)

Pointer

&(Ampersand) : Used in front of the variable, returns address of the variable
*(Asterisk) : Can be used for both pointer declaration and pointer dereference

```
int x = 9;
int *p_x = &x;
*p_x = 11936;
printf("%d\n", *p_x);
```

Pointer declaration

Returns address of the variable

Pointer dereference

Pointer

```
int x = 9;
int *p_x = &x;
*p_x = 11936;
printf("%d\n", *p_x);
printf("%d\n", x);
```

Address	Memory
0x10	00001001
0x11	00000000
0x12	00000000
0x13	00000000
0x14	
0x15	
0x16	
0x17	
0x18	
0x19	
0x1A	
0x1B	
0x1C	
0x1D	
0x1E	
0x1F	
0x20	
0x21	
0x22	

&(Ampersand) : Used in front of the variable, returns address of the variable
*(Asterisk) : Can be used for both pointer declaration and pointer dereference

Pointer

```
int x = 9;
int *p_x = &x;
*p_x = 11936;
printf("%d\n", *p_x);
printf("%d\n", x);
```

Address	Memory
0x10	00001001
0x11	00000000
0x12	00000000
0x13	00000000
0x14	
0x15	
0x16	
0x17	
0x18	
0x19	
0x1A	
0x1B	
0x1C	
0x1D	
0x1E	
0x1F	
0x20	
0x21	
0x22	

&(Ampersand) : Used in front of the variable, returns address of the variable
*(Asterisk) : Can be used for both pointer declaration and pointer dereference

Pointer

```
int x = 9;
int *p_x = &x;
*p_x = 11936;
printf("%d\n", *p_x);
printf("%d\n", x);
```

Address	Memory
0x10	00001001
0x11	00000000
0x12	00000000
0x13	00000000
0x14	
0x15	
0x16	
0x17	
0x18	00010000
0x19	00000000
0x1A	00000000
0x1B	00000000
0x1C	00000000
0x1D	00000000
0x1E	00000000
0x1F	00000000
0x20	
0x21	
0x22	

0x10

&(Ampersand) : Used in front of the variable, returns address of the variable
*(Asterisk) : Can be used for both pointer declaration and pointer dereference

Pointer

```
int x = 9;
int *p_x = &x;
*p_x = 11936; // same as x=11936
printf("%d\n", *p_x);
printf("%d\n", x);
```

Address	Memory
0x10	00001001
0x11	00000000
0x12	00000000
0x13	00000000
0x14	
0x15	
0x16	
0x17	
0x18	00010000
0x19	00000000
0x1A	00000000
0x1B	00000000
0x1C	00000000
0x1D	00000000
0x1E	00000000
0x1F	00000000
0x20	
0x21	
0x22	

0x10

&(Ampersand) : Used in front of the variable, returns address of the variable
***(Asterisk) : Can be used for both pointer declaration and pointer dereference

Pointer

```
int x = 9;
int *p_x = &x;
*p_x = 11936; // same as x=11936
printf("%d\n", *p_x);
printf("%d\n", x);
```

dereferencing

Address	Memory
0x10	00001001
0x11	00000000
0x12	00000000
0x13	00000000
0x14	
0x15	
0x16	
0x17	
0x18	00010000
0x19	00000000
0x1A	00000000
0x1B	00000000
0x1C	00000000
0x1D	00000000
0x1E	00000000
0x1F	00000000
0x20	
0x21	
0x22	

0x10

&(Ampersand) : Used in front of the variable, returns address of the variable
*(Asterisk) : Can be used for both pointer declaration and pointer dereference

Pointer

```
int x = 9;
int *p_x = &x;
*p_x = 11936; // same as x=11936
printf("%d\n", *p_x);
printf("%d\n", x);
```

dereferencing

Address	Memory
0x10	10100000
0x11	00101110
0x12	00000000
0x13	00000000
0x14	
0x15	
0x16	
0x17	
0x18	00010000
0x19	00000000
0x1A	00000000
0x1B	00000000
0x1C	00000000
0x1D	00000000
0x1E	00000000
0x1F	00000000
0x20	
0x21	
0x22	

0x10

&(Ampersand) : Used in front of the variable, returns address of the variable
*(Asterisk) : Can be used for both pointer declaration and pointer dereference

Pointer - scanf

```
int x = 0;
scanf("%d", &x);

printf("%d", x);
```

&(Ampersand) : Used in front of the variable, returns address of the variable
***(Asterisk) : Can be used for both pointer declaration and pointer dereference

LAB – PointerIntro

- Create a file named ‘PointerIntro_YourName.c’
- Declare three different types of variables (ex – int, float, char)
- Declare pointers for each of these variables
- Assign values to the variables using pointers
- Print the values using pointers

DoublePointer

```
int x = 9;
int *p_x = &x;
int **pp_x = &p_x;
**pp_x = 11936;
printf("%d\n", x);
```

Address	Memory
0x10	00001001
0x11	00000000
0x12	00000000
0x13	00000000
0x14	
0x15	
0x16	
0x17	
0x18	
0x19	
0x1A	
0x1B	
0x1C	
0x1D	
0x1E	
0x1F	
0x20	
0x21	
0x22	

&(Ampersand) : Used in front of the variable, returns address of the variable
***(Asterisk) : Can be used for both pointer declaration and pointer dereference

DoublePointer

```
int x = 9;
int *p_x = &x;
int **pp_x = &p_x;
**pp_x = 11936;
printf("%d\n", x);
```

Address	Memory
0x10	00001001
0x11	00000000
0x12	00000000
0x13	00000000
0x14	00010000
0x15	00000000
0x16	00000000
0x17	00000000
0x18	00000000
0x19	00000000
0x1A	00000000
0x1B	00000000
0x1C	
0x1D	
0x1E	
0x1F	
0x20	
0x21	
0x22	

&(Ampersand) : Used in front of the variable, returns address of the variable
*(Asterisk) : Can be used for both pointer declaration and pointer dereference

DoublePointer

```
int x = 9;
int *p_x = &x;
int **pp_x = &p_x;
**pp_x = 11936;
printf("%d\n", x);
```

Address	Memory
0x10	00001001
0x11	00000000
0x12	00000000
0x13	00000000
0x14	00010000
0x15	00000000
0x16	00000000
0x17	00000000
0x18	00000000
0x19	00000000
0x1A	00000000
0x1B	00000000
0x1C	
0x1D	
0x1E	
0x1F	
0x20	
0x21	
0x22	

0x10

&(Ampersand) : Used in front of the variable, returns address of the variable
*(Asterisk) : Can be used for both pointer declaration and pointer dereference

DoublePointer

```
int x = 9;
int *p_x = &x;
int **pp_x = &p_x;
**pp_x = 11936;
printf("%d\n", x);
```

Address	Memory
0x10	00001001
0x11	00000000
0x12	00000000
0x13	00000000
0x14	00010000
0x15	00000000
0x16	00000000
0x17	00000000
0x18	00000000
0x19	00000000
0x1A	00000000
0x1B	00000000
0x1C	00011100
0x1D	00000000
0x1E	00000000
0x1F	00000000
0x20	00000000
0x21	00000000
0x22	00000000

0x10

0x14

&(Ampersand) : Used in front of the variable, returns address of the variable
***(Asterisk) : Can be used for both pointer declaration and pointer dereference

DoublePointer

```
int x = 9;
int *p_x = &x;
int **pp_x = &p_x;
**pp_x = 11936;
printf("%d\n", x);
```

Address	Memory
0x10	00001001
0x11	00000000
0x12	00000000
0x13	00000000
0x14	00010000
0x15	00000000
0x16	00000000
0x17	00000000
0x18	00000000
0x19	00000000
0x1A	00000000
0x1B	00000000
0x1C	00011100
0x1D	00000000
0x1E	00000000
0x1F	00000000
0x20	00000000
0x21	00000000
0x22	00000000

0x10

0x14

&(Ampersand) : Used in front of the variable, returns address of the variable
***(Asterisk) : Can be used for both pointer declaration and pointer dereference

DoublePointer

```
int x = 9;
int *p_x = &x;
int **pp_x = &p_x;
**pp_x = 11936;
printf("%d\n", x);
```

dereferencing

Address	Memory
0x10	00001001
0x11	00000000
0x12	00000000
0x13	00000000
0x14	00010000
0x15	00000000
0x16	00000000
0x17	00000000
0x18	00000000
0x19	00000000
0x1A	00000000
0x1B	00000000
0x1C	00011100
0x1D	00000000
0x1E	00000000
0x1F	00000000
0x20	00000000
0x21	00000000
0x22	00000000

0x10

0x14

&(Ampersand) : Used in front of the variable, returns address of the variable
***(Asterisk) : Can be used for both pointer declaration and pointer dereference

DoublePointer

```
int x = 9;
int *p_x = &x;
int **pp_x = &p_x;
**pp_x = 11936;
printf("%d\n", x);
```

dereferencing

dereferencing

Address	Memory
0x10	00001001
0x11	00000000
0x12	00000000
0x13	00000000
0x14	00010000
0x15	00000000
0x16	00000000
0x17	00000000
0x18	00000000
0x19	00000000
0x1A	00000000
0x1B	00000000
0x1C	00011100
0x1D	00000000
0x1E	00000000
0x1F	00000000
0x20	00000000
0x21	00000000
0x22	00000000

0x10

0x14

&(Ampersand) : Used in front of the variable, returns address of the variable
***(Asterisk) : Can be used for both pointer declaration and pointer dereference

DoublePointer

```
int x = 9;
int *p_x = &x;
int **pp_x = &p_x;
**pp_x = 11936;
printf("%d\n", x);
```

dereferencing

dereferencing

Address	Memory
0x10	10100000
0x11	00101110
0x12	00000000
0x13	00000000
0x14	00010000
0x15	00000000
0x16	00000000
0x17	00000000
0x18	00000000
0x19	00000000
0x1A	00000000
0x1B	00000000
0x1C	00011100
0x1D	00000000
0x1E	00000000
0x1F	00000000
0x20	00000000
0x21	00000000
0x22	00000000

0x10

0x14

&(Ampersand) : Used in front of the variable, returns address of the variable

***(Asterisk) : Can be used for both pointer declaration and pointer dereference

LAB – DoublePointer

- Create a file named ‘DoublePointer_YourName.c’
- Declare an int type variable and a pointer to that variable
- Declare a double pointer that stores the address of the first pointer
- Assign values to the int type variable using both single and double pointers
- Print the int type variable’s value using the
  - Variable
  - single pointer
  - double pointer

Pointer - Array

```
int arr[3] = {2,0,25};
printf("%p", (void*)arr);
printf("%d", *arr);
printf("%p", (void*)(arr+1));
printf("%d", *(arr+1));
printf("%d", *(arr)+1);
printf("%d", *(arr+1)+1);
```

Address	Memory
0x10	00000010
0x11	00000000
0x12	00000000
0x13	00000000
0x14	00000000
0x15	00000000
0x16	00000000
0x17	00000000
0x18	00011001
0x19	00000000
0x1A	00000000
0x1B	00000000
0x1C	
0x1D	
0x1E	
0x1F	
0x20	
0x21	
0x22	

Pointer - Array

```
int arr[3] = {2,0,25};
printf("%p", (void*)arr);
printf("%d", *arr);
printf("%p", (void*)(arr+1));
printf("%d", *(arr+1));
printf("%d", *(arr)+1);
printf("%d", *(arr+1)+1);
```

Address	Memory
0x10	00000010
0x11	00000000
0x12	00000000
0x13	00000000
0x14	00000000
0x15	00000000
0x16	00000000
0x17	00000000
0x18	00011001
0x19	00000000
0x1A	00000000
0x1B	00000000
0x1C	
0x1D	
0x1E	
0x1F	
0x20	
0x21	
0x22	

Pointer - Array

```
int arr[3] = {2,0,25};
printf("%p", (void*)arr);
printf("%d", *arr);
printf("%p", (void*)(arr+1));
printf("%d", *(arr+1));
printf("%d", *(arr)+1);
printf("%d", *(arr+1)+1);
```

Address	Memory
0x10	00000010
0x11	00000000
0x12	00000000
0x13	00000000
0x14	00000000
0x15	00000000
0x16	00000000
0x17	00000000
0x18	00011001
0x19	00000000
0x1A	00000000
0x1B	00000000
0x1C	
0x1D	
0x1E	
0x1F	
0x20	
0x21	
0x22	

Pointer - Array

```
int arr[3] = {2,0,25};
printf("%p", (void*)arr);
printf("%d", *arr);
printf("%p", (void*)(arr+1));
printf("%d", *(arr+1));
printf("%d", *(arr)+1);
printf("%d", *(arr+1)+1);
```

4 Byte

Address	Memory
0x10	00000010
0x11	00000000
0x12	00000000
0x13	00000000
0x14	00000000
0x15	00000000
0x16	00000000
0x17	00000000
0x18	00011001
0x19	00000000
0x1A	00000000
0x1B	00000000
0x1C	
0x1D	
0x1E	
0x1F	
0x20	
0x21	
0x22	

+1 to a pointer type :
Add size of the variable type

In this case, arr type is int (4 Byte)

Pointer - Array

```
int arr[3] = {2,0,25};
printf("%p", (void*)arr);
printf("%d", *arr);
printf("%p", (void*)(arr+1));
printf("%d", *(arr+1));
printf("%d", *(arr)+1);
printf("%d", *(arr+1)+1);
```

4 Byte

Address	Memory
0x10	00000010
0x11	00000000
0x12	00000000
0x13	00000000
0x14	00000000
0x15	00000000
0x16	00000000
0x17	00000000
0x18	00011001
0x19	00000000
0x1A	00000000
0x1B	00000000
0x1C	
0x1D	
0x1E	
0x1F	
0x20	
0x21	
0x22	

+1 to a pointer type :
Add size of the variable type

In this case, arr type is int (4 Byte)

Pointer - Array

```
int arr[3] = {2,0,25};
printf("%p", (void*)arr);
printf("%d", *arr);
printf("%p", (void*)(arr+1));
printf("%d", *(arr+1));
printf("%d", *(arr)+1);
printf("%d", *(arr+1)+1);
```

4 Byte

Address	Memory
0x10	00000010
0x11	00000000
0x12	00000000
0x13	00000000
0x14	00000000
0x15	00000000
0x16	00000000
0x17	00000000
0x18	00011001
0x19	00000000
0x1A	00000000
0x1B	00000000
0x1C	
0x1D	
0x1E	
0x1F	
0x20	
0x21	
0x22	

+1 to a pointer type :
Add size of the variable type

In this case, arr type is int (4 Byte)

Pointer - Array

```
int arr[3] = {2,0,25};
printf("%p", (void*)arr);
printf("%d", *arr);
printf("%p", (void*)(arr+1));
printf("%d", *(arr+1));
printf("%d", *(arr)+1);
printf("%d", *(arr+1)+1);
```

4 Byte

Address	Memory
0x10	00000010
0x11	00000000
0x12	00000000
0x13	00000000
0x14	00000000
0x15	00000000
0x16	00000000
0x17	00000000
0x18	00011001
0x19	00000000
0x1A	00000000
0x1B	00000000
0x1C	
0x1D	
0x1E	
0x1F	
0x20	
0x21	
0x22	

Pointer - Array

```
int arr[3] = {2,0,25};
printf("%p", (void*)arr);
printf("%d", *arr);
printf("%p", (void*)(arr+1));
printf("%d", *(arr+1));
printf("%d", *(arr)+1);
printf("%d", *(arr+1)+1);
```

4 Byte

Address	Memory
0x10	00000010
0x11	00000000
0x12	00000000
0x13	00000000
0x14	00000000
0x15	00000000
0x16	00000000
0x17	00000000
0x18	00011001
0x19	00000000
0x1A	00000000
0x1B	00000000
0x1C	
0x1D	
0x1E	
0x1F	
0x20	
0x21	
0x22	

Pointer - Array

```
int arr[3] = {2,0,25};
printf("%p", (void*)arr);
printf("%d", *arr);
printf("%p", (void*)(arr+1));
printf("%d", *(arr+1));
printf("%d", *(arr)+1);
printf("%d", *(arr+1)+1);
```

4 Byte

Address	Memory
0x10	00000010
0x11	00000000
0x12	00000000
0x13	00000000
0x14	00000000
0x15	00000000
0x16	00000000
0x17	00000000
0x18	00011001
0x19	00000000
0x1A	00000000
0x1B	00000000
0x1C	
0x1D	
0x1E	
0x1F	
0x20	
0x21	
0x22	

Pointer - Array

```
int arr[3] = {2,0,25};
printf("%p", (void*)arr);
printf("%d", *arr);
printf("%p", (void*)(arr+1));
printf("%d", *(arr+1));
printf("%d", *(arr)+1);
printf("%d", *(arr+1)+1);
```

4 Byte

Address	Memory
0x10	00000010
0x11	00000000
0x12	00000000
0x13	00000000
0x14	00000000
0x15	00000000
0x16	00000000
0x17	00000000
0x18	00011001
0x19	00000000
0x1A	00000000
0x1B	00000000
0x1C	
0x1D	
0x1E	
0x1F	
0x20	
0x21	
0x22	

Pointer – Array

```
int i = 2;
```

i = 2

```
int arr_1D[3] = {2,0,25};
```

Pointer – Array

```
int i = 2;
```

type of arr_1D = int * -> int = 4 Byte

```
int arr_1D[3] = {2,0,25};
```

arr_1D = address of arr_1D[0] => 2
arr_1D+1 = address of arr_1D[1] => 0

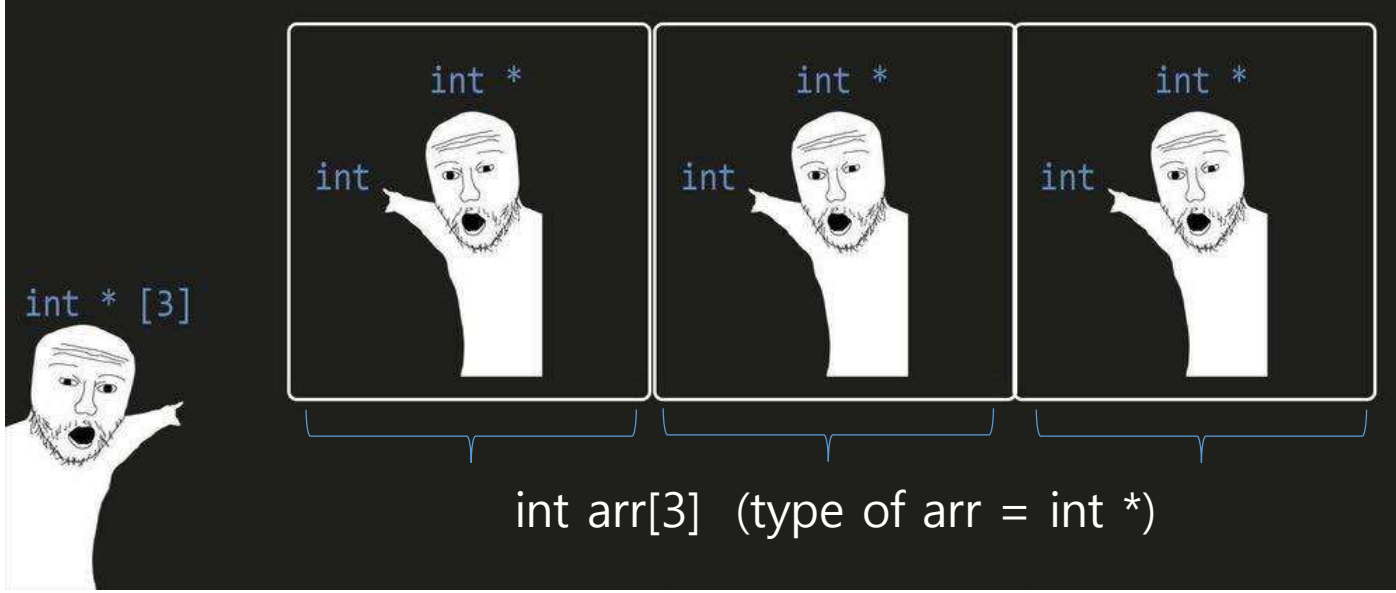
Pointer – 2DArray

```
int i = 2;

int arr_1D[3] = {2,0,25};
```

*(arr_1D) = value of arr_1D[0] => 2
*(arr_1D)+1 = value of arr_1D[0]+1 => 3

Pointer – 2DArray




```
int i = 2;
```

```
int arr_1D[3] = {2,0,25};
```

type of arr_2D = int * [3] -> int[3] = 12 Byte

```
int arr_2D[2][3] = {{2,0,25},{1,3,5}};
```

arr_2D = address of arr_2D[0][0] => 2
arr_2D+1 = address of arr_2D[1][0] => 1

Pointer – 2DArray

```
int i = 2;
```

```
int arr_1D[3] = {2,0,25};
```

type of *(arr_2D) = int * -> int = 4 Byte

```
int arr_2D[2][3] = {{2,0,25},{1,3,5}};
```

*(arr_2D) = address of arr_2D[0][0] => 2
*(arr_2D)+1 = address of arr_2D[0][1] => 0

Pointer – 2DArray

```
int i = 2;

int arr_1D[3] = {2,0,25};

int arr_2D[2][3] = {{2,0,25},{1,3,5}};
```

**(arr_2D) = value of arr_2D[0][0] => 2
**(arr_2D)+1 = value of arr_2D[0][0]+1 => 3

Pointer – 3DArray

```
int i = 2;

int arr_1D[3] = {2,0,25};

int arr_2D[2][3] = {{2,0,25},{1,3,5}};
```

type of arr_3D = int * [2][3] -> int[2][3] = 24 Byte

```
int arr_3D[2][2][3] = {{{2,0,25},{1,3,5}}, {{2,0,24},{9,8,7}}};
```

arr_3D = address of arr_3D[0][0] => 2
arr_3D+1 = address of arr_3D[?][?]+1 => ???

LAB – Pointer2DArray

- Create a file named ‘Pointer2DArray_YourName.c’
- Declare a 2D integer array of size 10 x 5 (int arr[5][10])
- Declare a pointer to the first element of the 2D array
- Use only pointer arithmetic to access and print all elements in row-major order
- Print the values in a formatted matrix style