Review of C-Programming: Pointer, 1D Array

Young-Keun Kim

mod .2022.08.04

Smart Sensor Systems Lab

Pointer

Pointer

What are Pointers?

- A **pointer** is a variable whose value is the address of another variable
- i.e., direct address of the memory location.
- Pointers are the basis for data structures

(a) Define a pointer variable

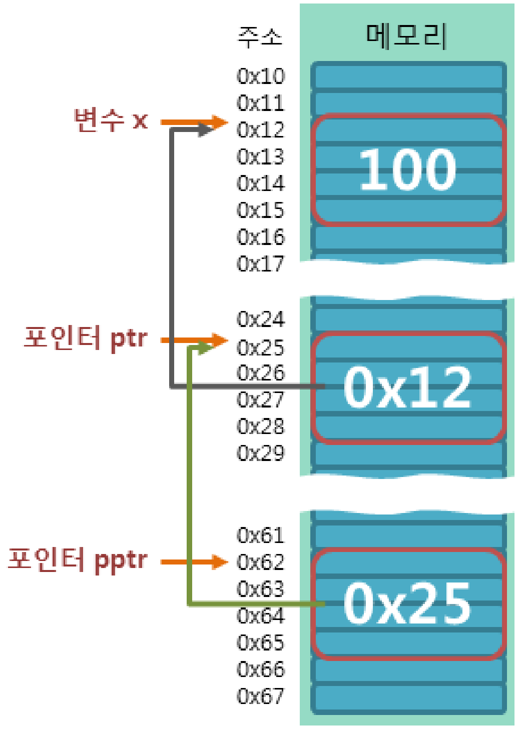
(b) Assign the address of a variable to a pointer and

(c) Access the value at the address available in the pointer variable

```
int  *ptr;
ptr = &x;
int value = *ptr
```

```
int x =100;        // 변수의 선언
int *ptr = &x;     // 포인터의 선언
int **pptr = &ptr; // 포인터의 참조
```

```
int main() {
      int  x = 100;

      // Pointer Define
      int  *ptr;

      // Pointer Assignment
      ptr = &x;// store address of var in pointer variable

      // Pointer Access
      printf("Value of *ptr: %d", *ptr);

      return 0;
}
```



[Image from TCPschool]

Pointer

What to remember in Pointer!!!

1. Define

Use * to **create** “pointer variable“

ex)

```
int    a       =  54;
int  *point_a  =  &a;
```

2.Usage

Use * to **access value** at the address of which “pointer variable“ is pointing

ex)

```
a   =54            point_a  = 0x1F00
*a  (error!)      *point_a  = 54
```

Use **&** to **get** or **assign** the **address** of “variable”

ex)

```
    a          =  54
   &a          =  0x1F00
 & *point_a    =  0x1F00
*& *point_a    =  54
                 …
```

Pointer

What to remember in Pointer!!!

ex) Passing variable's address to a function

```
int main() {
     int val1 = 10;
     int val2 = 20;

     swap(&val1, &val2);
     ...
}
```

Assign

(&val1): to assign and pass address of (val1)

Define

(int *a) : declare 'a' as a pointer

```
void swap( int *a, int *b) {
     int temp = *a;
     *a = *b;
     *b = temp;
}
```

Access

(*a) : to access value of (val1)

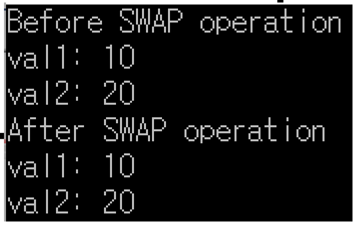
the value at the address of (a==&val1)

Pointer

Call by Value

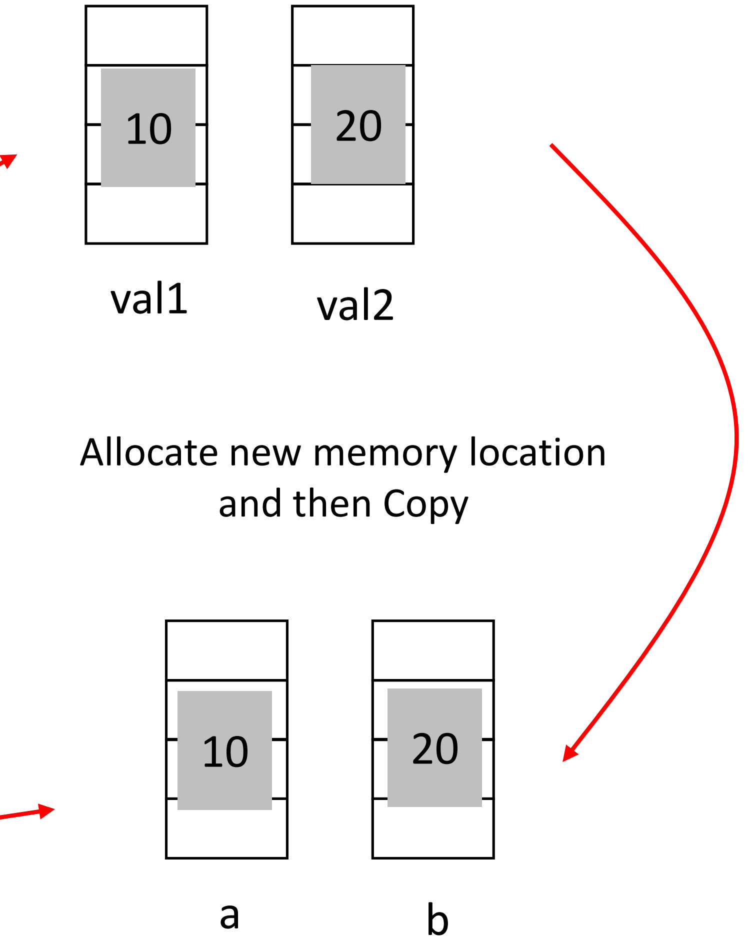
```c
#include <stdio.h>
#include <stdlib.h>

void swap(int a, int b);

int main() {
	int val1 = 10;
	int val2 = 20;

	printf("Before SWAP operation \n");
	printf("val1: %d \n", val1);
	printf("val2: %d \n", val2);

	swap(val1, val2);

	printf("After SWAP operation \n");
	printf("val1: %d \n", val1);
	printf("val2: %d \n", val2);

	system("pause");
	return 0;
}

void swap(int a, int b) {
	int temp = a;
	a = b;
	b = temp;
}
```

Pass values



```
Before SWAP operation
val1: 10
val2: 20
After SWAP operation
val1: 10
val2: 20
```

Values NOT swapped

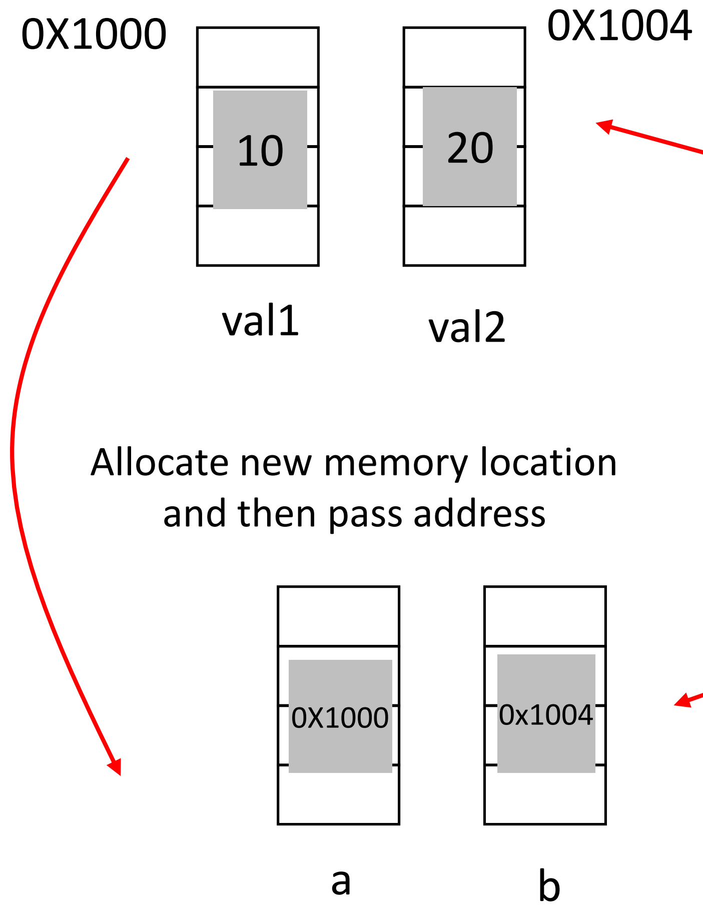


Allocate new memory location and then Copy

Therefore the result here does NOT affect original val1 and val2

Pointer

Call by Reference



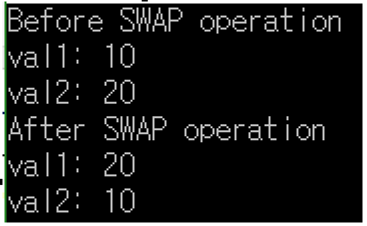
Allocate new memory location
and then pass address

It accesses the memory location
of val1 and val2 using passed addresses.
Therefore values are changed

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

void swap(int *a, int *b);

int main() {
        int val1 = 10;
        int val2 = 20;

        printf("Before SWAP operation ₩n");
        printf("val1: %d ₩n", val1);
        printf("val2: %d ₩n", val2);

        swap(&val1, &val2);

        printf("After SWAP operation ₩n");
        printf("val1: %d ₩n", val1);
        printf("val2: %d ₩n", val2);

        system("pause");
        return 0;
}

void swap(int *a, int *b) {
        int temp = *a;
        *a = *b;
        *b = temp;
}
```

Pass addresses

*b : value at address of which *b is pointing.
(therefore, value of b)



```
Before SWAP operation
val1: 10
val2: 20
After SWAP operation
val1: 20
val2: 10
```

Values swapped

Pointer

Compare the result of following cases:

Case .01

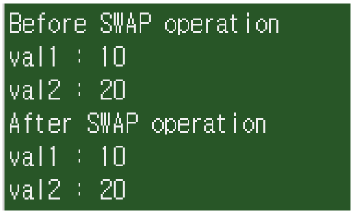
```
void swap( int a, int b);

int main() {
        int val1 = 10;
        int val2 = 20;

        swap(val1, val2);

        return 0;
}

void swap( int a, int b) {
        int temp = a;
        a = b;
        b = temp;
}
```

Case .02

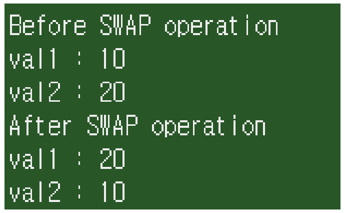
```
void swap( int *a, int *b);

int main() {
        int val1 = 10;
        int val2 = 20;

        swap(&val1, &val2);

        return 0;
}

void swap( int *a, int *b) {
        int temp = *a;
        *a = *b;
        *b = temp;
}
```

Case .03

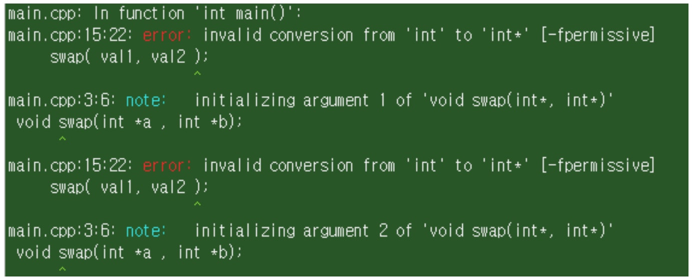
```
void swap( int *a, int *b);

int main() {
        int val1 = 10;
        int val2 = 20;

        swap(val1, val2);

        return 0;
}

void swap( int *a, int *b) {
        int temp = *a;
        *a = *b;
        *b = temp;
}
```



```
Before SWAP operation
val1 : 10
val2 : 20
After SWAP operation
val1 : 10
val2 : 20
```



```
Before SWAP operation
val1 : 10
val2 : 20
After SWAP operation
val1 : 20
val2 : 10
```



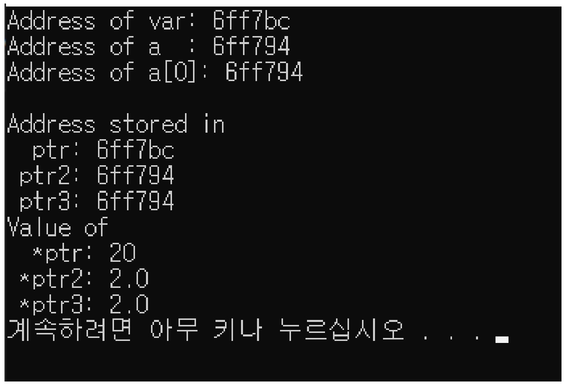
```
main.cpp: In function 'int main()':
main.cpp:15:22: error: invalid conversion from 'int' to 'int*' [-fpermissive]
     swap( val1, val2 );
                      ^
main.cpp:3:6: note:   initializing argument 1 of 'void swap(int*, int*)'
 void swap(int *a , int *b);
      ^
main.cpp:15:22: error: invalid conversion from 'int' to 'int*' [-fpermissive]
     swap( val1, val2 );
                      ^
main.cpp:3:6: note:   initializing argument 2 of 'void swap(int*, int*)'
 void swap(int *a , int *b);
      ^
```

C_pointer_example1.c

Pointer

Example Code

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

int main() {
      int  var = 20;
      double a[4] = { 2, 2, 3, 4 };

      // Pointer Declaration
      int  *ptr;
      double *ptr2 = NULL;// good practice for not Addr. assgined pointer
      double *ptr3 = NULL;

      // Pointer Assignment
      ptr = &var;// store address of var in pointer variable
      ptr2 = a;
      ptr3 = &a[0];

      printf("Address of var: %x\n", &var);
      printf("Address of a  : %x\n", &a);
      printf("Address of a[0]: %x\n", &a[0]);

      // Using Pointer - Access the values pointed by Ptr
      printf("\nAddress stored in \n  ptr: %x \n ptr2: %x  \n ptr3: %x\n", ptr, ptr2, ptr3);
      printf("Value of \n  *ptr: %d \n *ptr2: %.1f  \n *ptr3: %.1f\n", *ptr, *ptr2, *ptr3);

      system("pause");
      return 0;
}
```



```
Address of var: 6ff7bc
Address of a  : 6ff794
Address of a[0]: 6ff794

Address stored in
  ptr: 6ff7bc
 ptr2: 6ff794
 ptr3: 6ff794
Value of
  *ptr: 20
 *ptr2: 2.0
 *ptr3: 2.0
계속하려면 아무 키나 누르십시오 . . .
```

C_pointer_example2.c

Pointer : Exercise

Exercise 1

C_pointer_exercise1.c

- Print the address of variable ‘x’
- Print the address of variable ‘y’

- Print the value of pointer ‘ptrX ‘
- Print the address of pointer ‘ptrX ‘
- Print the size of pointer ‘ptrX ‘

- Print the value of pointer ‘ptrY ‘
- Print the address of pointer ‘ptrY ‘
- Print the size of pointer ‘ptrY ‘

```
int x =10;
double y=2.5;
int *ptrX = &x;
int *ptrY = &y;
```

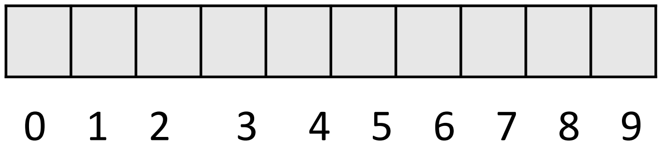
- Check solution: C_pointer_exercise1 _solution.c

1D Array

Array

- **1-D Array**

Declare : uint32_t number[10];



Count starts from zero(0)

Initialize : uint32_t number[10] = {10 , 9 , 8 , 7 , 6 , 5 , 4 , 3 , 2 , 1 };

= {10 , 9 , 8 , 7 , 6 };

= {0 };

Use : uint32_t number[10] = {10 , 9 , 8 , 7 , 6 , 5 , 4 , 3 , 2 , 1 };

number[0] = 10 // number = 0x00001000

number[1] = 9

Address of number[0] constant.

(Quiz)

number[10] = ? number+1 = ?

Array

Example:

C_array1d_example.c

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

void printVec(double *_vec, int _row);

int main()
{
	// Static Matrix Allocation  1-D array
	// fixed array size and initial constant values
	double a[4] = { 1, 2, 3, 4 };
	double b[] = { 2, 3, 4, 5 };
	double c[4] = { 0 };

	// Print 1-D array element
	printVec(a, 4);

	system("pause");
	return 0;
}

void printVec(double *_vec, int _row)
{
	for (int i = 0; i<_row; i++)
	  printf("Vector[%d] = %f \n", i, _vec[i]);
	printf("\n");
}
```

//Declaring & Assigning Array

//Passing to a function:
a is address(i.e no need of '&')

//Passing to a function

Array 1D: Exercise

Exercise1 :

Create functions of

C_array1D_exercise.c

```
addVec(X, Y, Out, dim);

Where
X,Y,Out: 1D array float type
dim: dimension of vector, integer type
```

- *The size of vector need to be fixed in variable declaration in main()*

- Check solution: C_array1D_solution.c

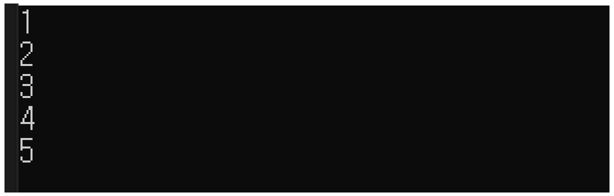
Array 1D: Exercise 2

Exercise2: 1-D Array & pointer

- Assign array address to pointer ptr.
- Print each element of 1D array by using pointer.

C_array1d_exercise2.c

```
#include <stdio.h>

int main(){

    int st[5] = { 1,2,3,4,5 };
    int* ptr;

    ptr = _________________;
    for (int i = 0; i < 5; i++) {
        //  print each element by using pointer  e.g.  (ptr)

    }
}
```



```
1
2
3
4
5
```

- Check solution: C_array1d_exercise2_solution.c