C Programming
Lecture 10-1 : Array & Pointer

Character Array

- String
  - A sequence of characters
  - The last character should be **'\0'** that indicates **"the end of string"**

```
char greeting[] = "hello";
```

greeting

0	1	2	3	4	5
h	e	l	l	o	\0

same

```
char greeting[] = { 'h', 'e', 'l', 'l', 'o', '₩0' };
```

String I/O

```
char myName[8];
gets(myName);
```

myName

0	1	2	3	4	5	6	7
?	?	?	?	?	?	?	?

myName

0	1	2	3	4	5	6	7
L	e	e	\0	?	?	?	?

Lee

If a user gives "Lee" as a keyboard input following gets() function, myName array will be filled with 'L','e','e','₩0' .

If keyboard input string is too big (>= 8 characters), an error will occur.

```
// Character array (string) input/output)
#include <stdio.h>

int main()
{
    int i;
    char string1[20];

    printf("Enter the string \"hello there\": ");
    gets(string1);

    printf("\nstring1 :\n");
    for (i = 0; string1[ i ] != '\0'; i++ )
        printf("%c");
    printf("\nthe length of string1 is %d\n",i);

    return 0;
}
```

```
$ ./a.out
Enter the string "hello there": hello there

string1 :
hello there
the length of string1 is 11
```

Pointer

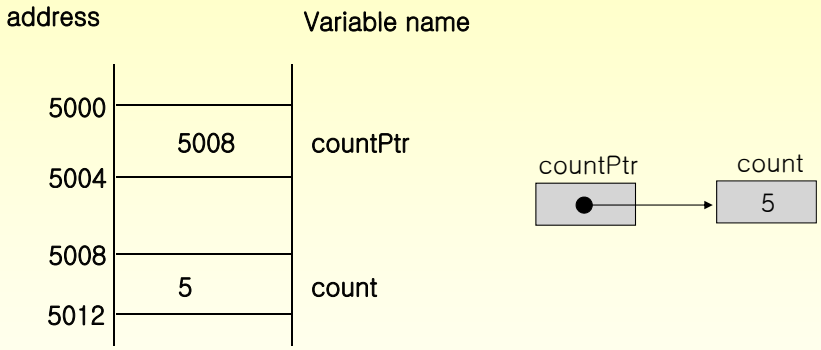
Pointer?

- Pointer
  - Usually means memory address
- Declaration of Pointer-type variable
  - Declare a variable that stores memory address
- Pointer operators
  - * (Dereference operator)
    - means “the value of”
  - & (address-of operator)
    - means “address of”

Example : Pointer Variable and Pointer Operators

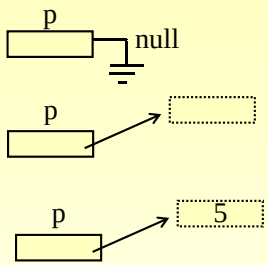
```
int *countPtr, count;    // countPtr is pointer
                         // count is an integer type

int *aPtr, *bPtr;        // aPtr and bPtr are pointer type
aPtr = NULL;             // aPtr is a NULL pointer. NULL = 0

count = 5;               // assignment
countPtr = &count;       // countPtr points to count that means
                         // the value of countPtr is the address of count

*countPtr = 10;          // the value countPtr points to is changed to 10
                         // that means count becomes 10
```

Memory

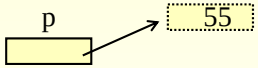


Pointers and Addresses

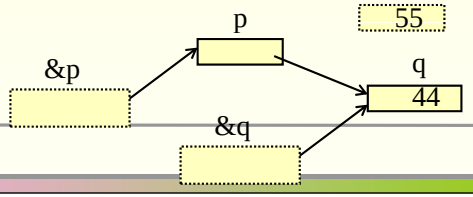
```
int  *p = 0;

P=(int*)malloc(sizeof(int));

*p=5
```




```
*p  = 55;
```



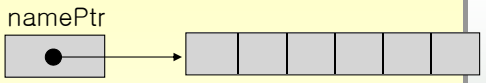

```
int  q = 44;
     p = &q;
```



Dynamic Allocation (동적할당)

```
malloc : memory allocation
free   : memory deallocation
```


```
int* intPtr;
char* nameStr;
```




```
intPtr = (int*)malloc(sizeof(int));      // memory allocation
nameStr = (char*)malloc(sizeof(char)*6);

free(intPtr);         // deallocation
free(nameStr);        // deallocation
```

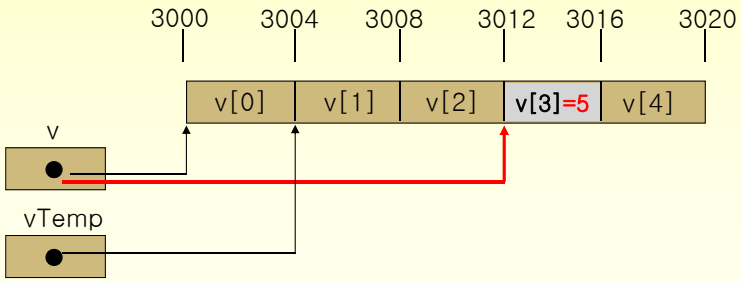
Memory Leak : The loss of available memory space that occurs when data is dynamically allocated but never deallocated.

Inaccessible object : a dynamic variable on the free store without any pointer pointing to it

Dangling pointer : a pointer that points to a variable that has been deallocated.

Pointer Arithmetic

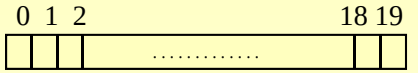
```
int *v, *vTemp;
v = (int*)malloc(sizeof(int)*5);
```




```
vTemp= v+1;
v += 3;
*v=5
```

- **Arrays**

```
char   charray[20];
```



charray is an array variable. charray is also a pointer that points to the address of the first array element (= &charray[0])

***charray is the same as charray[0]**

***(charray+1) is the same as charray[1]**

Arrays & Pointers

- Almost interchangeable.
- You must understand!

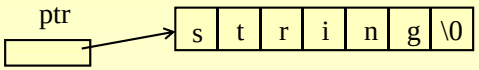
- Difference?

```
int b[5];
int *b;

int* a=b;
int* a=&b[0];

int c = b[1];
int c = *(b+1);
```

- You must always be careful when using pointers with dynamic allocation especially when using in a function (memory leak).

- What is Garbage Collection?

```
char   *ptr =  "string";
```



Pointer **ptr** can be used as a form of an array. For example

```
*ptr == ptr[0],   *(ptr + 1) == ptr[1], . . .
```

sizeof operator

- Returns the size of an array or of any other data type, variable or constant

- The size of pointer type variable is 4byte in 32bit computer.

- Example

```
double Array[20];
char a, b[10], *c *d;
d = (char*)malloc(sizeof(char)*100);

sizeof(Array) // 160
sizeof(a)     // 1
sizeof(b)     // 10
sizeof(c)     // 4
sizeof(d)     // 4
sizeof(int)   // 4
```

String processing functions

- Defined in <string.h>

```
char* strcpy(char *s1, const char *s2);
char* strncpy(char *s1, const char *s2, size_t n);
char* strcat(char *s1, const char *s2);
char* strncat(char *s1, const char *s2, size_t n);
int strcmp(const char *s1, const char* s2);
int strncmp(const char *s1, const char* s2, size_t n);
char* strtok(char *s1, const char *s2);
size_t strlen(const char *s);
```

example1

```
#include <stdio.h>

void printIntVar(char *name, int value)
{
    printf("%s\t = %d\n", name, value);
}

int main()
{
    int one = 1;
    int *to_one;

    to_one = &one;
    printIntVar("one", one);
    *to_one = one + 1;
    printIntVar("one", one);
    *to_one = *to_one + 1;
    printIntVar("one", one);
    (*to_one)++;
    printIntVar("one", one);

    return 0;
}
```

output:
one = 1
one = 2
one = 3
one = 4

Example2 : swap function

```
#include <stdio.h>

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

int main()
{
          int x=3, y=2;
          printf("before: x=%d, y=%d\n",x,y);
          swap(x,y);
          printf("after : x=%d, y=%d\n",x,y);
}
```

Output :

```
#include <stdio.h>

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

int main()
{
          int x=3, y=2;
          printf("before: x=%d, y=%d\n",x,y);
          swap(&x,&y);
          printf("after : x=%d, y=%d\n",x,y);
}
```

Output :

Example 3

```
#include <stdio.h>

int main()
{
    char *quote[] = {
        "To err is human, to forgive divine.",
        "To detect errors is compiler, to correct them is human.",
        "Time flies like an arrow, fruit flies like a banana."
    };
    int i;
    const int num = sizeof quote/sizeof *quote;

    for (i = 0; i < num; ++i)
        printf("%s\n", quote[i]);

    return 0;
}
```

Example 4

```
#include <stdio.h>

#define PRPTR(p) printf("%p\n", p)

int main()
{
    int nums[] = {1,3,2,4,3,5,4,2};
    int *a = nums;
    int *b = a + 4;

    printf("sizeof(int) = %d\n", sizeof(int));
    PRPTR(a + 0);
    PRPTR(a + 1);
    PRPTR(b - 2);
    PRPTR(b - 1);

    return 0;
}
```