Lecture 3

What we will study

1. Definition of Pointers
2. Operation on Pointers
3. Arrays and Pointers
4. Structures and Pointers

If you want to be proficient(익숙한, 숙달한, 능숙한, 능란한) in writing code in C language, you need to be very much comfortable(기분 좋은, 편안한) with pointers. Actually pointers are both an advantage and curse(저주하다, 재앙을 빌다) for C language. We will learn about advantages of pointers in C language and you will learn disadvantages through experience. At the end of the lecture, I will list some disadvantages about pointers.

Before starting with pointers, it is important to get a deep ((아래로) 깊은) insight(통찰, 간파;통찰력,) about variables. So, we will first start with variables, moving on to pointers, and then we will learn how pointers are related to array. After this, we will learn about how pointers are used with structures and functions.

What is a variable in C? A variable is something(섬싱) with a name and it can hold (갖고 있다, 붙들다) a value. We can change () the value stored in a variable. For example

```
int x;
```

x is the name of the variable, and it can hold integer values. When we declare an integer variable, C compiler allocates (배분하다) 4 bytes of memory. This 4 bytes of memory has certain address and this is the address where the value is stored. This address is called as address of variable. And at this address, value is stored.

We can have an analogy (유사, 비슷함) with a big vacant piece of land in Suwon. Mr. Huh wants a pie (파이, 파이 모양의 것) out of that land for building a house. So Mr Huh visit the office of Municipal Authority of Suwon and request for a pie out of that land. After some days, Municipal Authority of Suwon randomly (닥치는 대로의, 되는 대로의) allocates a pie of land to Mr. Huh. Mr Huh is very happy that now he can build a house of his own on that land.

Piece of Land is like memory, municipal authority is like compiler, and pie of land allocated for building house is like part of memory allocated to the variable. The allocated land to Mr huh is used for building a house, and allocated memory to the variable is used for holding value.

Now let us get deeper into variables, for that take an example

```
int x = 4;
```

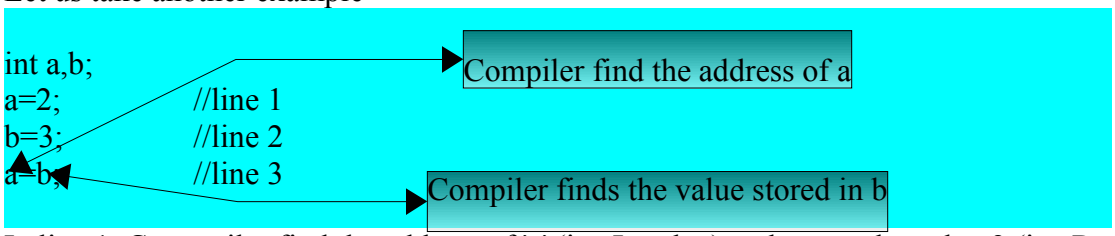
there are 2 values associated (상기시키다, 관련시켜 생각하다) with variable 'x'. One is the address of x and second is the value stored in that address. Compiler

randomly choose an addresses for a variable.
There is a defined nomenclature (학명, 술어) for the 2 values associated with a variable rvalue (r-value) and lvalue (l-value).

l-value is the address of x, which is the place where something can be stored. In the above example l-value is on the left side
r-value is the value which is stored in x. In the above example r-value is on the right side.

2=x is invalid i.e. R-value = L-value is invalid

Let us take another example



In line 1, C compiler find the address of 'a' (i.e. L-value) and stores the value 2 (i.e. R-value) in that address of memory.

In line2, C compiler find the address of 'b' (i.e. L-value) and stores the value 3 (i.e. R-value) in that address of memory.

In line3, C compiler find the address of 'a' (i.e L-value)and stores the value of b (i.e. R-value)

Can we change the value of a? Yes
Can we change the address of a? No,
Can I know, What is the address of a? Yes
Can I store the address of 'a' somewhere? Yes
How can I store the address of 'a'?

Let us assume that address of 'a' is 420. To store the address of a, C provided as 'pointers'. I can store the address of 'a' in an integer pointer.

For example:

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

'ptr' is a pointer variable of type integer. That means, it can only store addresses of integer variables.
We already know that the address of a variable can be obtained by &. Let us write a program to store and display the address.

```
1. void main( )
2. {
3.        int x; // x contains some junk value
```

```
4.      int *ptr; //ptr contains NULL
5.
6.      x=2;
7.      ptr=&x;
8.
9.      printf(“Address of x is = %u”, &x);
10.     pritnf(“Value of x is =%d  ”, x);
11.     printf(“value of ptr is = %u”, ptr);
12.     printf(“Address of ptr is = %u”, &ptr);
13.     printf(“value stored at address %u is %d”, ptr,*ptr);
14.
15.     *ptr = 10;
16.    print(“value of x =  %d”, x);
17.    printf(“value of x is = %d”,*ptr);
14.}
```

Line no.3 declares an integer variable x, that means, compiler allocates 4 bytes of memory for x. Initially x contains garbage value.
Line no.4 declares an integer pointer variable ptr, compiler allocates 2 bytes of memory for a pointer. Initially ptr contains NULL.

Line no. 6 assigns 2 to x.
Line no.7 assigns the address of x to ptr.

Line no.9, prints the address of x (assume 420)
Line no.10, prints 2, the value of x
Line no.11, prints 420, the value of ptr
Line no.12, prints the address of ptr (assume 800)
Line no.13, prints 420 (address of x) and 2 (get the value stored at address pointed by ptr)

Line no. 15: We need to understand this one. **It means store 10 at the address contained in ptr**. Address contained in ptr is 200. So store 10 at the memory location 200. '*' is called as dereferencing operator, that means, **get the value from the address pointed by ptr**.

```
#include <stdio.h>

int j, k;
int *ptr;

int main(void)
{
    j = 1;
    k = 2;
    ptr = &k;
    printf("\n");
    printf("j has the value %d and is stored at %p\n", j, &j);
    printf("k has the value %d and is stored at %p\n", k, &k);
    printf("ptr has the value %p and is stored at %p\n", ptr, &ptr);
    printf("The value of the integer pointed to by ptr is %d\n",
*ptr);

}
```

What is the output of above program?

Arrays and Pointers

If we have an array,

```
int x[5] = {1,6,2,9,4}; and base address is 400.
```

What is the output of

```
printf("%u",x);
printf("%u",&x[0]);
```

We know that, **name of the address contains the base address (start address of the array or address of first element)**.

x gives the address of 1st element in the array &x[0]
x+1 gives the address of 2nd element i.e. &x[1]
x+2 gives the address of 3rd element i.e. &x[2]
.
.
.
x+i gives the address of (i+1)th element i.e. &x[i]

```
for(i=0;i<5;i++)
    printf("%u",(x+i));
```

Above loop will print the addresses of all the elements of the array.
Now, What is the output of

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

In *(x) , '*' is a dereferencing operator here, this means, get the value stored at the address pointed by x. i.e. 1

*(x) prints the value of 1st element in the array i.e. x[0]
*(x+1) prints the value of 2nd element in the array i.e. x[1]
*(x+2) prints the value of 3rd element in the array i.e. X[2]
.
.
.*(x+i) prints the value of (i+1)th element in the array i.e. x[i]

Program for printing the value of all the elements of the array using pointers is

```
for(i=0;i<5;i++)
    printf("%d",*(x+i))
```

Pointers and Strings

First start with the definition of String, A String is defined as an array of characters terminated by a null character ('**\0**').

For example

```
char my_string[10];

my_string[0] = 'Y';
my_string[1] = 'O';
my_string[2] = 'U';
my_string[3] = 'N';
my_string[4] = 'G';
my_string[5] = 'I';
my_string[6] = 'M';
my_string[7] = '\0';
```

OR

```
char my_string[10] = “YOUNGIM”;
```

When the double quotes are used, instead of the single quotes as was done in the previous examples, the null character (**'\0'**) is automatically appended to the end of the string

OR

```
char my_string[10] = {'Y', 'O', 'U', 'N', 'G', 'I', 'M', '\0'};
```

Program to find the length of a string

```
void main()
{
        char my_string[50];
        int i=0;
        int length=0;
        printf(“Enter the string”);
        scanf(“%s”,my_string);

        printf(“String enetered by user is %s”, my_string);

        While(my_string[i] != '/0') // my_string [i] is not equal to null
        {
                length = length + 1;
                i++;
        }

        printf(“length of the string %s  is %d”str, length);
}
```

In the above program, while loop does the job of finding the length of the array. Step
Step 1: Initially length is 0 and i is also 0.
Step 2: While my_string[i] is not equal to null,
while loop continues to iterate increasing i by 1 in each iteration.
As my_string[i] becomes equal to null, while loop terminates.

Step3: After while loop finishes, length variable contains the length of the array.

```
Program to find the length of the array using pointers

void main()
{
        char my_string[50];
        int i=0;
        int length=0;
        printf("Enter the string");
        scanf("%s",my_string);

        printf("String enetered by user is %s", my_string);

        While (*(my_string+i) != '/0') // my_string [i] is not equal to null
        {
                length = length + 1;
                i++;
        }

        printf("length of the string %s  is %d"str, length);
}
```

In the above program, we only replaced my_string[i] with *(my_string + i)

Now, let us write a function for finding the length of the string.

```
Program to find the length of the array using function
void main()
{
        char my_string[50];
        int length=0;
        printf("Enter the string");
        scanf("%s",my_string);

        printf("String enetered by user is %s", my_string);

        length = findlength(my_string); // pass the address of first element of the array

        printf("length of the string %s  is %d"str, length);
}

int findlength(char *s)
{
        int l=0;
        int i=0;
        While (*(str+i) != '\0')
        {
                length = length +1;
```

```
        }

        return length;
}
```

Write a Program to reverse a string.

```
Program to reverse  a string.

void main()
{
        char my_string[40]=”Shinwa”;
        char *rev_str;
        int i,j;

        int length = strlen(my_string);
        rev_str = (char *) malloc(sizeof(char) * length);

        for(i=legth – 1, j=0; i>=0 ; i--,j++)
                rev_str[j] = my_string[i];

        rev_str[j] = '\0';

        printf(“Original String is %s”, my_string);
        printf(“Reverse String is %s”, rev_str);

}
```

Write the above program using pointers

```
Program to reverse  a string using Pointers

void main()
{
        char my_string[40]=”Shinwa”;
        char *rev_str;
        int i,j;

        int length = strlen(my_string);
        rev_str = (char *) malloc(sizeof(char) * length);

        for(i=legth – 1, j=0; i>=0 ; i--,j++)
                *(rev_str + j) = *(my_string + i);

        *(rev_str + j) = '\0';

        printf(“Original String is %s”, my_string);
        printf(“Reverse String is %s”, rev_str);
```

```
}
```

In the above program we only replaced rev_str[j] with *(rev_str + j) and my_string[i] with *(my_string + i).

Write the program to reverse a string using function.

```
void main()
{
        char my_string[40]=”Shinwa”;
        char *rev_str;
        int i,j;

        int length = strlen(my_string);
        rev_str = (char *) malloc(sizeof(char) * length);

        reverse(my_string,rev_str,length);

        printf(“Original String is %s”, my_string);
        printf(“Reverse String is %s”, rev_str);
}

void reverse(char *s, char *d, int len)
{
        int i;
        int j;
        for(i=len -1, j=0; i >= 0 ; i++ )
                *(d + j) = *(s + i);
        *(d+ j) = '\0';
}
```