SIES: BASIC C PROGRAMMING

L #13: C POINTERS

Seung Beop Lee
School of International Engineering and Science
CHONBUK NATIONAL UNIVERSITY

Outline

- What are pointers?
- How to use pointers?
- Pointers in Detail

Pointer

Pointer

- What are pointers?
  - Before considering the pointer, **every variable** is a **memory location** and **every memory location** has **its address defined** which can be accessed using **ampersand (&)** operator, which denotes an address in memory.
  - Consider the following example, which will print the address of the variables defined:

```c
#include <stdio.h>
int main()
{
    // declaration the variable var1
    int var1;
    // declaration the array var2
    char var2[10];

    printf("Address of var1 variable: %x\n", &var1);  // ampersand (&) operator denotes an address of the variable var 1
    printf("Address of var2 variable: %x\n", &var2);  // ampersand (&) operator denotes an address of the array variable var 2

    return 0;
}

/* when the condition is true,
        the following statements will be exe
 // Increment operator(++), j increases the
```

Microsoft Visual Studio Debug Console

```
Address of var1 variable: 3bf710
Address of var2 variable: 3bf6fc

C:₩Users₩SBLEE₩source₩repos₩Project1₩Debug₩Project1.exe (process 3408) exit
To automatically close the console when debugging stops, enable Tools->Opti
le when debugging stops.
```

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**.
  - Like any variable or constant, you must declare a pointer before you can use it to store any variable address.
- Syntax
  - The general form of a pointer variable declaration is:

```
type *var-name;
```

  - Here, type is the pointer's base type; it must be a valid C data type and var-name is the name of the pointer variable.
  - The asterisk * you used to declare a pointer is the same asterisk that you use for multiplication.
  - However, in this statement the asterisk is being used to designate a variable as a pointer. Following are the valid pointer declaration:

```
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch     /* pointer to a character */
```

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**.
  - Like any variable or constant, you must declare a pointer before you can use it to store any variable address.
- Syntax
  - The general form of a pointer variable declaration is:

```
type *var-name;
```

  - Here, type is the pointer's base type; it must be a valid C data type and var-name is the name of the pointer variable.
  - The asterisk * you used to declare a pointer is the same asterisk that you use for multiplication.
  - However, in this statement the asterisk is being used to designate a variable as a pointer. Following are the valid pointer declaration:
  - The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

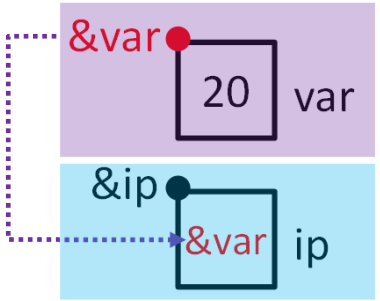
How to use pointers?

- How to use pointers?
  - There are few important operations, which we will do with the help of pointers very frequently. **(a)** we define a pointer variable **(b)** assign the address of a variable to a pointer and **(c)** finally access the value at the address available in the pointer variable. This is done by using unary operator * that <u>returns the value of the variable located at the address specified by its operand</u>.
  - Following example makes use of these operations:

```c
#include <stdio.h>
int main()
{
    int var = 20; /* actual variable declaration */
    int *ip; /* pointer variable declaration */
    ip = &var; /* store address of var ip pointer variable*/

    printf("Address of var variable: %x\n", &var);
    /* address stored in pointer variable */
    printf("Address stored in ip variable: %x\n", ip);
    /* access the value using the pointer */
    printf("Value of *ip variable: %d\n", *ip);

    return 0;
}

/* when the co
```

C:\Users\SBLEE\source\repos\Project1\Debug\Project1.exe

```
Address of var variable: 93fbe0
Address stored in ip variable: 93fbe0
Value of *ip variable: 20
```



Null pointers in C

- Null pointer
  - It is always a good practice to assign a **NULL** value to a pointer variable in case you **do not have exact address** to be assigned.
  - This is done at the time of pointer variable declaration. A pointer that is assigned **NULL** is called a **null** pointer.
  - The **NULL** pointer is a constant with a value of **zero** defined in several standard libraries. Consider the following program:

```c
#include <stdio.h>
int main()
{
    int *ptr = NULL;

    printf("The value of ptr is : %x\n", ptr);

    return 0;
}

/* whe
```

C:\Users\SBLEE\source\repos\Project1\Debug\Project1.exe

```
The value of ptr is : 0
```

Pointers in Detail

- Five important concepts related to pointers
  - Pointers have many but easy concepts and they are very important to C programming.
  - The following important pointer concepts should be clear to any C programmer:

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 pointers 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 pointer 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

- Pointer Arithmetic
  - As explained in the previous contents, C pointer is an address, which is a **numeric value** (**long hexadecimal number**). Therefore, you can perform arithmetic operations on a pointer just as you can a numeric value.
  - There are four arithmetic operators that can be used on pointers: **++, --, +, and -** .

Pointer Arithmetic

- Incrementing a pointer
  - Unlike the array name which cannot be incremented because it is a constant pointer, you prefer using a pointer in your program instead of an array because the variable pointer can be incremented,
  - The following program increments the variable pointer to access each succeeding element of the array:

```c
#include <stdio.h>

const int MAX = 3;

int main()
{
    int var[] = { 10, 100, 200 };
    int i, *ptr;

    /* let us have array address in pointer */
    ptr = var;

    for (i = 0; i < MAX; i++)
    {
        printf("Address of var[%d] = %x\n", i, ptr);
        printf("Value of var[%d] = %d\n", i, *ptr);

        /* move to the next location */
        ptr++;
    }
    return 0;
}
/* when the condi
        the follow
// Increment oper
```

C:\Users\SBLEE\source\repos\Project1\Deb

```
Address of var[0] = c6f7c4
Value of var[0] = 10
Address of var[1] = c6f7c8
Value of var[1] = 100
Address of var[2] = c6f7cc
Value of var[2] = 200
```

Pointer Arithmetic

- Decrementing a pointer
  - The same considerations apply to decrementing a pointer, which decreases its value by the number of bytes of its data type as shown below:

```c
#include <stdio.h>
const int MAX = 3;

int main()
{
    int var[] = { 10, 100, 200 };
    int i, *ptr;

    /* let us have array address in pointer */
    ptr = &var[MAX - 1];

    for (i = MAX; i > 0; i--)
    {
        printf("Address of var[%d] = %x\n", i, ptr);
        printf("Value of var[%d] = %d\n", i, *ptr);

        /* move to the previous location */
        ptr--;
    }
    return 0;
}
/* when the con
        the foll
// Increment op
```

```
C:\Users\SBLEE\source\repos\Project1\Debu
Address of var[3] = 6ff83c
Value of var[3] = 200
Address of var[2] = 6ff838
Value of var[2] = 100
Address of var[1] = 6ff834
Value of var[1] = 10
```

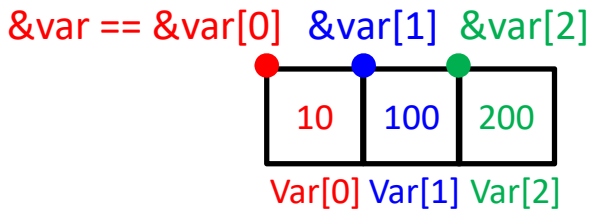
Pointer Arithmetic

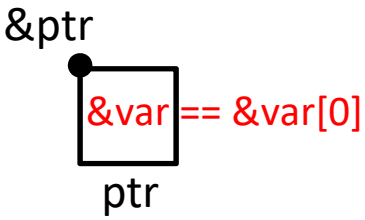
- Pointer comparisons
  - Pointers may be compared by using relational operators, such as **==, <, and >**. If p1 and p2 point to variables that are related to each other, such as elements of the same array, then p1 and p2 can be **meaningfully** compared.
  - The following program modifies the previous example one by incrementing the variable pointer so long as the address to which it points is either less than or equal to the address of the last element of the array, which is &var[MAX - 1]:

```c
#include <stdio.h>
const int MAX = 3;
int main()
{
    int var[] = { 10, 100, 200 };
    int i, *ptr;
    /* let us have address of the first element in pointer */
    ptr = var;
    i = 0;

    while (ptr <= &var[MAX - 1])
    {
        printf("Address of var[%d] = %x\n", i, ptr);
        printf("Value of var[%d] = %d\n", i, *ptr);

        /* point to the previous location */
        ptr++;
        i++;
    }
    return 0;
}
/* when the condi
        the follow
// Increment oper
```

C:\Users\SBLEE\source\repos\Project1\Debug\Project1.exe

```
Address of var[0] = 77fca8
Value of var[0] = 10
Address of var[1] = 77fcac
Value of var[1] = 100
Address of var[2] = 77fcb0
Value of var[2] = 200
```





Array of pointers

- Array
  - Before we understand the concept of **arrays of pointers**, let us consider the following example, which makes use of an array of 3 integers:

```c
#include <stdio.h>
const int MAX = 3;
int main()
{
    int var[] = { 10, 100, 200 };
    int i;
    for (i = 0; i < MAX; i++)
    {
        printf("Value of var[%d] = %d\n", i, var[i]);
    }
    return 0;
}
/* when the cor
        the foll
// Increment op
```

Select C:\Users\SBLEE\source\repos\Project1\

```
Value of var[0] = 10
Value of var[1] = 100
Value of var[2] = 200
```

Array of pointers

- Array of pointers
  - The array of pointers can be used a situation when we want to maintain an array, which can store pointers to an int or char or any other data type available.
  - Following is the declaration of an array of pointers to an integer:

```
int *ptr[MAX];
```

  - This declares ptr as an array of MAX integer pointers. Thus, <u>each element in ptr, now holds a pointer to an int value</u>.
  - Following example makes use of three integers, which will be stored in an array of pointers as follows:

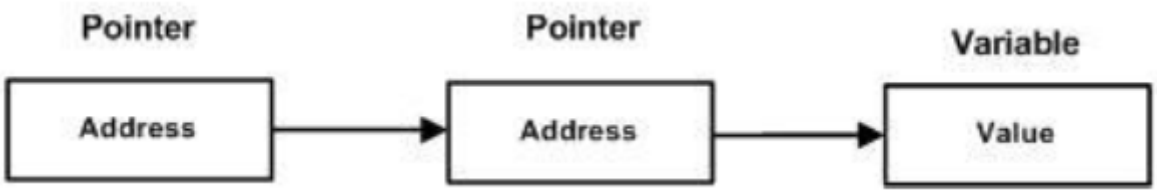
```
#include <stdio.h>

int main()
{
    int var[] = { 10, 100, 200 };
    int i, *ptr[3];
    int MAX = 3;

    for (i = 0; i < MAX; i++)
    {
        ptr[i] = &var[i]; /* assign the address of integer. */
    }

    for (i = 0; i < MAX; i++)
    {
        printf("Value of var[%d] = %d\n", i, *ptr[i]);
    }

    return 0;
}
/* when the con
    the foll
```

C:\Users\SBLEE\source\repos\Project1\Debug\Project1.exe

```
Value of var[0] = 10
Value of var[1] = 100
Value of var[2] = 200
```

Pointer to pointer

- Pointer to pointer
  - A pointer to a pointer is a form of **multiple indirection**, or a chain of pointers. Normally, a pointer contains the address of a variable.
  - When we define **a pointer to a pointer**, the **first pointer contains the address of the second pointer**, which points to the location that contains the actual value as shown below.



- Syntax
  - A variable that is a pointer to a pointer must be declared as such. This is done by placing an additional asterisk in front of its name.
  - For example, following is the declaration to declare a pointer to a pointer of type int:

```
int **var;
```

Pointer to pointer

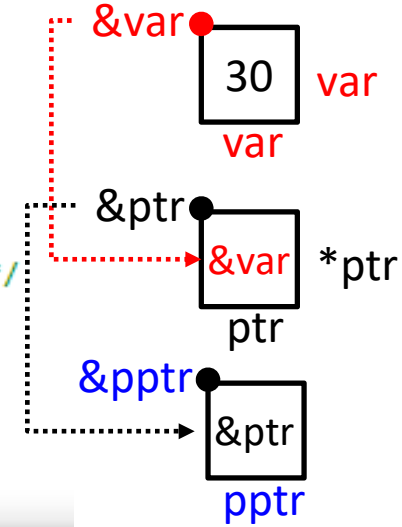
- Pointer to pointer
  - When a target value is **indirectly** pointed to by <u>a pointer to a pointer</u>, accessing that value requires that the asterisk operator be applied **twice**, as is shown below in the example:

```c
#include <stdio.h>

int main()
{
    int var;
    int *ptr;
    int **pptr;
    var = 30;

    /* take the address of var */
    ptr = &var;
    /* take the address of ptr using address of operator & */
    pptr = &ptr;

    /* take the value using pptr */
    printf("Value of var = %d\n", var);
    printf("Value available at *ptr = %d\n", *ptr);
    printf("Value available at **pptr = %d\n", **pptr);

    return 0;
}
```

*(*pptr) == *(*&prt)==*prt ==*(&var)

*ptr == &var

*&var == 30



C:\Users\SBLEE\source\repos\Project1\Debug\Project1.exe

```
Value of var = 30
Value available at *ptr = 30
Value available at **pptr = 30
```

*pptr == &var

*&var == 30

**pptr == 30

Passing pointers to functions

- Passing pointers to functions
  - C programming language allows you to pass a pointer to a function. To do so, simply declare the function parameter as a pointer type.
  - Following a simple example where we pass an integer pointer to a function and change the value inside the function which reflects back in the calling function:

```c
#include <stdio.h>

// function declaration using the pointes variable as a paramenter for getAverage functioin
double getAverage(int *arr, int size);

int main()
{
    /* an int array with 5 elements */
    int balance[5] = { 1000, 2, 3, 17, 50 };
    double avg;

    /* pass pointer to the array as an argument */
    avg = getAverage(balance, 5);

    /* output the returned value */
    printf("Average value is: %f\n", avg);
    return 0;
}

//definition of the getSecond function
double getAverage(int *arr, int size)
{
    int i, sum = 0;
    double avg;

    for (i = 0; i < size; ++i)
    {
        sum += arr[i];
    }
    avg = (double)sum / size;
    return avg;
}
```

```
C:\Users\SBLEE\source\repos\Project1\Debug\Project1.exe
Average value is: 214.400000
```

Return pointer from functions

- Return pointer from functions
  - As we have seen in last chapter, how C programming language allows to return an array from a function, similar way C allows you to **return a pointer** from a function.
  - Note that it is not good idea to return the address of a local variable to outside of the function so you would have to define the local variable as **static variable**.

- Syntax
  - To do so, you would have to declare a function returning a pointer as in the following example:

```
int * myFunction()
{
.
.
.
}
```

Return pointer from functions

- Return pointer from functions
  - Now, consider the following function, which will generate 10 random numbers and returns them using an array name which represents a pointer, i.e., address of first array element.

```c
#include <stdio.h>
//#include <time.h>

/* function to generate and retrun random numbers. */
int * getRandom()
{
    static int r[10];
    int i;
    /* set the seed */
    //srand((unsigned)time(NULL));
    for (i = 0; i < 10; ++i)
    {
        r[i] = rand();
        printf("%d\n", r[i]);
    }
    return r;
}

/* main function to call above defined function */
int main()
{
    /* a pointer to an int */
    int *p;
    int i;
    p = getRandom();
    for (i = 0; i < 10; i++)
    {
        printf("*(p + [%d]) : %d\n", i, *(p + i));
    }
    return 0;
}
```

C:\Users\SBLEE\source\r

```
41
18467
6334
26500
19169
15724
11478
29358
26962
24464
*(p + [0]) : 41
*(p + [1]) : 18467
*(p + [2]) : 6334
*(p + [3]) : 26500
*(p + [4]) : 19169
*(p + [5]) : 15724
*(p + [6]) : 11478
*(p + [7]) : 29358
*(p + [8]) : 26962
*(p + [9]) : 24464
```

Summary

Summary

- ✓ We considered the **definition of the pointers and pointers in detail**.

Thank You