Pointers

Pointers are often thought to be the most difficult aspect of C. In C when we define a pointer variable by preceding its name with an *. In C we also give our pointer a type which refers to the type of data stored at the address we will be storing in our pointer. Here is an example

```c
#include <stdio.h>
main()
{
    int *pt; /* Define a pointer.*/
    pt = (int*) calloc (1, sizeof(int)); /* We need to assign an address for pt. */
    *pt=9; /* Assign 9 to the variable pt points to. */
    printf("The value pt points to is %d\n", *pt);
    free(pt); /* free the memory you requested. */
}
```

Here, pt is the name of our variable. The '*' informs the compiler that we want a pointer variable. So the compiler can assign enough space to store an address in memory. The int says that we intend to use our pointer variable to store the address of an integer. Also, if we place the * in front of a pointer, the * operator accesses the value pointed to by that pointer.

Note: You’re response for all the memory you requested.

We will introduce another operator ‘&’ using the example below

```c
#include <stdio.h>
main()
{
    int *pt;
    int i = 9;
    pt = &i; /* Assign the address of variable i to pt. */
    printf("The value pt points to is %d\n", *pt);
    *pt = 11; /* If we change the variable value pt points to to 11. */
    printf("The new value for variable i is %d\n", i);
}
```

What the ‘&’ operator does is retrieve the address of k. And we can also assign the address of i to pt.

Actually, if we create an array a[4], the name of the array ‘a’ can be treat as a pointer which points to the first element of the array. Now, consider the following example,
Note: 1. We can use the form `{1, 2, 3, 4}` to assign values to an array only when we define the array. Check the following example

```c
#include <stdio.h>
main()
{
    int i, a[4] = {1, 2, 3, 4};
    int i, *pt;
    /* We can access the array using the usual way. */
    for(i=0; i<4; i++) {
        printf(" %d", a[i]);
    }
    printf("\n");
    /* We also can access the content using pointer. */
    for(i=0; i<4; i++) {
        printf(" %d", *(a+i));
    }
    printf("\n");
    /* Or we can try this way. */
    pt = &a[0];
    for(i=0; i<4; i++) {
        printf(" %d", *(pt+i));
    }
    printf("\n");
    /* And we can show the address for each element of the array. */
    for(i=0; i<4; i++) {
        printf("%p\n", (a+i));
    }
}
```

2. The computer address is represented in Hexadecimal. On our system, int has 32 bits and the size is 4 bytes.
Actually, a[i] is equivalent to *(array + i). Here is a very interesting example,

```c
#include <stdio.h>
main()
{
    int a[4]={1, 2, 3, 4};
    int i, *pt;
    printf("The original array is:\n");
    for(i=0; i<4; i++) {
        printf(" %d", a[i]);
    }
    printf("\n");
    pt = &a[1];
    printf("The new array is:\n");
    for(i=0; i<3; i++) {
        printf(" %d", pt[i]);
    }
    printf("\n");
}
```

We know that: the elements of a multidimensional array are stored in memory linearly. So actually, we can use one dimensional array to access it.

```c
#include <stdio.h>
main()
{
    int i, j, b[3][3]={1, 2, 3, 4, 5, 6, 7, 8, 9}, *pt;
    printf("The original array is:\n");
    for(i=0; i<3; i++) {
        for(j=0; j<3; j++) {
            printf(" %d", b[i][j]);
        }
        printf("\n");
    }
    printf("\n");
    /* Show the address of each element. */
    for(i=0; i<3; i++) {
        for(j=0; j<3; j++) {
            printf(" %p", &b[i][j]);
        }
        printf("\n");
    }
    printf("\n");
    /*Access the array using pointer.*/
    pt = &b[0][0];
    for(i=0; i<9; i++) {
        printf(" %d", pt[i]);
    }
    printf("\n");
}
```
Double pointer
Since we can have pointers to int, or any type of variable in C, it shouldn't come as too much of a surprise that we can have pointers to other pointers. Just keep in mind the difference between pointer itself and what it points to. We can define the double pointer in this way

```c
Int **pptr;
```

where two asterisks indicate that two levels of pointers are involved.

```c
#include <stdio.h>
main()
{
    /* Here is an example of double pointer. */
    int i=1, j=2, k=3;
    int *ptr1=&i, *ptr2=&j;
    int **pptr=&ptr1; /* Define a pointer which points to the pointer ptr1. */
    printf("pptr=%p\n", pptr);
    printf("&ptr1=%p\n", &ptr1);
    printf("*pptr=%p\n", *pptr);
    printf("&i=%p\n\n", &i);

    /*Three ways to display variable i */
    printf("i=%d", i);
    printf(" i=%d", *ptr1);
    printf(" i=%d\n\n", **pptr);

    *pptr=ptr2;
    printf("*ptr1=%d\n", *ptr1);
}
```
Another interesting example of double pointer is 2-dimensinal array. Here is the C code

```c
#include <stdio.h>
main()
{
    int i, j, b[3][3]={1, 2, 3, 4, 5, 6, 7, 8, 9}, *pt;
    printf("The original array is:\n");
    for(i=0; i<3; i++) {
        for(j=0; j<3; j++) {
            printf(" %d", b[i][j]);
        }
        printf("\n");
    }
    printf("\n");
    printf("The following variables are equivalent:\n");
    printf("*b=%p\n", *b);
    printf("b[0]=%p\n", b[0]);
    printf("&b[0][0]=%p\n", &b[0][0]);
    printf("\n");
    /*Here is the method to access the content using double pointer. */
    for(i=0; i<3; i++) {
        for(j=0; j<3; j++) {
            printf(" %d", *(*(b+i)+j));
        }
        printf("\n");
    }
}
```

And as you may notice, in the example above, b is the name of the 2-dimensional array. It’s also the address of its first element. But differing from one-dimensional array, the first element is an array which contains 3 integers. So the following pairs of variable are equivalent

- ‘b’ and ‘&b[0]’
- ‘(b+1)’ and ‘&b[1]’
- ‘(b+2)’ and ‘&b[2]’
- ‘b[0]’ and ‘&b[0][0]’
- ‘b[1]’ and ‘&b[1][0]’
- ‘b[2]’ and ‘&b[2][0]’

That means: in the 2-dimensional array, the array name b is a double pointer which points to the pointer b[0]. And then the pointer b[0] points to the first element b[0][0].
**Prefix and postfix increment/decrement**

C provides prefix and postfix increment (and decrement) operators. Examples for prefix and postfix increments are i++ and ++i which we often use in for-loop. Usually the variable is an integer type (short, int, or long) but it can be a floating point type (float or double). No character is allowed between the two plus (or minus) signs. Although both i++ and ++i can be treated as i=i+1 in same sense. We will illustrate the difference using the following example

```c
#include <stdio.h>

main()
{
    int i = 0; /* Set the initial value to be 0*/
    int pos_fn, pre_fn;
    printf("The initial value of i is %d\n", i);
    pos_fn = i++;
    printf("The function value of i++ is %d\n", pos_fn);
    printf("The value of i after operation i++ is %d\n", i);
    printf("\n\n");

    i = 0;
    printf("The initial value of i is %d\n", i);
    pre_fn = ++i;
    printf("The function value of ++i is %d\n", pre_fn);
    printf("The value of i after operation ++i is %d\n", i);
}
```

From the example we can see that postfix increment evaluates the value of the expression before applying the plus operator. But prefix increment evaluates the value of the expression after applying the plus operator. An application of Prefix and postfix increment would be

```c
#include <stdio.h>

main()
{
    int a[3] = {1, 2, 3};
    int i;
    for(i = 0; i < 3;) {
        printf("%d\n", a[i++]);
    }

    printf("\n\n");
    /* This function will do the same thing.*/
    for(i = -1; i < 2;) {
        printf("%d\n", a[++i]);
    }
}
```