**Rajarata University of Sri Lanka**

**Faculty of Applied Sciences**

**COM 1407 – Computer Programming**

**Practical – Pointers in C**

---

**Outline**

- Define the C Pointers
- Application of C Pointers
- Pointers Types
- Writing programs with Pointers

---

- As you know, 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.

```c
#include <stdio.h>

int main () {

   int  var1;
   char var2[10];

   printf("Address of var1 variable: %x\n", &var1  );
   printf("Address of var2 variable: %x\n", &var2  );

   return 0;
}
```

## 1 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 using it to store any variable address.
- **Syntax**

```
type *variable-name;
```

- **type** is the pointer's base type; it must be a valid C data type and variable-name is the name of the pointer variable. The asterisk * used to declare a pointer.

## 2 How to Use Pointers?

- There are a few steps to follow
  I. Define a pointer variable
  II. Assign the address of a variable to a pointer
  III. Access the value at the address available in the pointer

```c
#include <stdio.h>

int main () {

   int  var = 20;   /* actual variable declaration */
   int  *ip;        /* pointer variable declaration */

   ip = &var;  /* store address of var in 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;
}
```

## 3 NULL Pointers

- It is always a good practice to assign a NULL value to a pointer variable in case you do not have an exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a null pointer.
- Try below example

```c
#include <stdio.h>

int main () {

   int  *ptr = NULL;

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

   return 0;}
```

# 4 Pointer Arithmetic

- Pointer in c is an address, which is a numeric value. Therefore, you can perform arithmetic operations on a pointer just as you can on a numeric value.

## 4.1 Incrementing a Pointer

- We prefer using a pointer in our program instead of an array because the variable pointer can be incremented, unlike the array name which cannot be incremented because it is a constant pointer.

```
#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;
}
```

## 4.2 Decrementing a Pointer

- The same considerations apply to decrementing a pointer, which decreases its value by the number of bytes of its data type.

```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-1, ptr );
      printf("Value of var[%d] = %d\n", i-1, *ptr );

      /* move to the previous location */
      ptr--;
   }

   return 0;
}
```

### 4.3 Pointer Comparisons

- Pointers may be compared by using relational operators, such as ==, <, and >.

```c
#include <stdio.h>

const int MAX = 3;

int main () {

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

   return 0;}
```

## 5 Arrays of Pointers

- There may be a situation when we want to maintain an array, which can store pointers to an int or char or any other data type available.

```
#include <stdio.h>

const int MAX = 3;

int main () {

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

   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;
}
```

## 6 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.
- **Syntax**

```
int **var;
```

- Try below example

```c
#include <stdio.h>

int main () {

   int  var;
   int  *ptr;
   int  **pptr;

   var = 3000;

   /* 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;
}
```

## 7 Passing Pointers to a function

- C programming allows passing a pointer to a function. To do so, simply declare the function parameter as a pointer type.
- Try below example.

```c
#include <stdio.h>
void salaryhike(int  *var, int b)
{
    *var = *var+b;
}
int main()
{
    int salary=0, bonus=0;
    printf("Enter the employee current salary:");
    scanf("%d", &salary);
    printf("Enter bonus:");
    scanf("%d", &bonus);
    salaryhike(&salary, bonus);
    printf("Final salary: %d", salary);
    return 0;
}
```