United International University

Dept. of Electrical and Electronic Engineering (EEE)

Course No. : **EEE 122**

Course Title: **Structured Programming Laboratory**

Lab Sheet 7
Pointer in C

Outcomes

After finishing this lab students should be able to ...

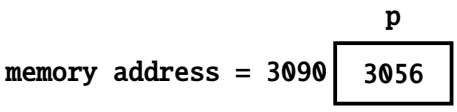
1. understand the C pointer types, the operations which may be performed with pointers, and how dynamic allocation/deallocation of memory is performed.
2. understand the relationship between pointers and arrays
3. learn how to access dynamically allocated arrays using pointers
4. understand and use pointers to functi

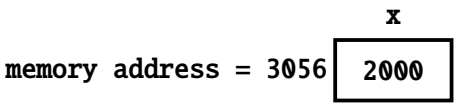
Contents

1 C - Pointer

1.1 The pointer type

A pointer is a variable whose values are addresses. If pointer **p** has as a value the memory address of variable **x**, one says that **p** points to **x**.





A pointer is bound to a type. If **x** is of type **int**, then pointer **p** is bound to type **int**.
Declaring a pointer is similar to declaring any variable, except for the fact that the pointer name is preceded by the character *:

```
type *name;
```

Example:

```
int *p;
```

The address (i.e. a pointer to) a variable is obtained using the unary operator **&**, called a **referencing operator**.
Example. Consider the declarations:

```
int x;
int *p;
```

Then, **p=&x;** has as an effect the assignment of the address of variable **x** to **p**. In the above sketch, the variable **x** is located at address **3056**, and thus the value of **p** will be **3056**.
To get the value stored in a memory area whose address is contained in a pointer, **p**, use the unary operator ***, called a **dereferencing operator**
Example:

```
In following examples, p must be bound to the type of x and y like this:
int x, y;
int *p;

a)Statement x=y is equivalent to one of the following sequences:
p=&x; or p=&y;
*p=y;    x=*p;

b) Statement x++ is equivalent to the sequence:
p=&x;
(*p)++;
```

1.2 The relationship between pointers and arrays

A name of an array has for a value the address of its first element. In consequence, one says that the name of an array is a constant pointer, which cannot be changed at run time.
Example:

```
int arr[100];
int *p;
int x;
...
p=arr; /* p is assigned the address of the element arr[0] */
...
```

In these above example, the assignment **x=arr[0]** is equivalent to **x=*p**;

Thus, it comes out that if an effective (actual) parameter is a single-dimensional array, then the corresponding formal parameter can be declared in two ways:

1. As an array: **type formal_parameter_name[];**
2. As a pointer: **type *formal_parameter_name;**

The example below is intended for finding the minimum and maximum element of a sequence.

```c
/* Program L7Ex1.c */
/* Shows a use of an array as a formal parameter */
#include <stdio.h>
void Max_min2(int n, int *a, int *max, int *min);
void Max_min1(int n, int a[], int *max, int* min);
int main(void){
  int i, n, maximum, minimum;
  int x[100];
  /* Data input */
  printf("\nMaximum and minimum element of integer array x.\nArray size is:");
  scanf("%d", &n);
  for (i=0; i<n; i++){
    printf("\nx[%d]=", i);
    scanf("%d", &x[i]);
  }
  /* Call of the first procedure */
  Max_min1(n, x, &maximum, &minimum);
  printf("Max_min1: maximum=%d and minimum=%d\n", maximum, minimum);
  /* Call of the second procedure */
  Max_min2(n, x, &maximum, &minimum);
  printf("Max_min2: maximum=%d and minimum=%d\n", maximum, minimum);
  return 0;
}
void Max_min1(int n, int a[], int *max, int* min){
  int i;
  *max=a[0];
  *min=a[0];
  for (i=1; i<n; i++){
    if (a[i]>*max) *max=a[i];
    else if (a[i]< *min) *min=a[i];
  }
}
void Max_min2(int n, int *a, int *max, int *min){
  int i;
  *max=a[0];
  *min=a[0];
  for (i=1; i<n; i++){
    if (a[i]>*max) *max=a[i];
    else if (a[i]< *min) *min=a[i];
  }
}
```

1.3 Mathematical operations with pointers

The following operations are applicable to pointers:

1. **Increment/decrement by 1.** In this case the value of the pointer is incremented/decremented with the number of bytes needed to store a data item of the type to which the pointer is bound to. The operators used are ++ and −−.
   Example:

```c
int arr[100];
int *p;
...
```

```
    p=&arr[10];
    p++; /* Value of p is incremented by sizeof(int),
        p has now the address of arr[11] */
```

2. **Addition/subtraction of an integer to/from a pointer.** The operation $p \pm n$ has as an effect the increase (p+n) or the decrease (pn) of the value of pointer **p** with **n*number of bytes** used to store a data item of the type to which the pointer is bound to. For the example above, if x is of type int, then
x=arr[i];
is equivalent to::
x=*(arr+i);

3. **The difference of two pointers.** If two pointers, **p** and **q** point to elements **i** and **j** of the same array, i.e. p=&arr[i] and q=&arr[j], and j > i, then **q-p = (j i)*number of bytes** used to store a data item of the base-type of that array.

4. **Pointer comparison.** Two pointers which both point to the elements of the same array can be compared using the relation, and the equality operators:

```
  < <= > >= == !=
```

Below is the program previously shown, but this time rewritten to use pointer operations.

```
/* Program L7Ex2.c */
/* Shows operations with pointers */
#include <stdio.h>
void Max_min1(int n, int a[], int *max, int* min);
void Max_min2(int n, int *a, int *max, int *min);
int main(void){
  int i, n, maximum, minimum, x[100];
  /* Data input */
  printf("\nMaximum and minimum element of integer array x.\nArray size is:");
  scanf("%d", &n);
  for (i=0; i<n; i++){
    printf("\nx[%d]=", i);
    scanf("%d", &x[i]);
  }
  /* Call of the first procedure */
  Max_min1(n, x, &maximum, &minimum);
  printf("Max_min1: maximum=%d and minimum=%d\n", maximum, minimum);
  /* Call of the second procedure */
  Max_min2(n, x, &maximum, &minimum);
  printf("Max_min2: maximum=%d and minimum=%d\n", maximum, minimum);
  return 0;
}

void Max_min1(int n, int a[], int *max, int* min){
  int i;
  *max=a[0];
  *min=a[0];
  for (i=1; i<n; i++){
    if (a[i] >*max) *max=a[i];
    else if (a[i] < *min) *min=a[i];
  }
}
```

```
void Max_min2(int n, int *a, int *max, int *min){
  int i;
  *max=*a;
  *min=*a;
  for (i=1; i<n; i++){
    if (*(a+i) >*max) *max=*(a+i);
    else if (*(a+i) < *min) *min=*(a+i);
  }
}
```

2 Programming Examples

Example: 1

Description: C Basic pointer function

Source Code

```
# include <stdio.h>
void fun(int *ptr);
int main(void){
  int y = 20;
  fun(&y);
  printf("%d", y);
  return 0;
}
void fun(int *ptr){
  *ptr = 30;
}
```

Output

```
30
```

Example: 2

Description:Program to dereference pointer variables.

Source Code

```
# include <stdio.h>
int main(void){
  int *ptr;
  int x;

  ptr = &x;
  *ptr = 0;
  printf(" x = %d\n", x);
  printf(" *ptr = %d\n", *ptr);

  *ptr += 5;
  printf(" x  = %d\n", x);
  printf(" *ptr = %d\n", *ptr);

  (*ptr)++;
  printf(" x = %d\n", x);
  printf(" *ptr = %d\n", *ptr);
  return 0;
}
```

Output

```
x = 0
*ptr = 0
x  = 5
*ptr = 5
x = 6
*ptr = 6
```

Example: 3

Description: Size of array and pointer type variables.

Source Code

```
# include <stdio.h>
int main(void){
  char arrc[] = {'a','b','c'};
  char *ptrc = arrc;

  int arri[] = {1, 2 ,3};
  int *ptri = arri;

  float arrf[] = {2.4,3.6,8.9};
  float *ptrf = arrf;

  double arrd[] = {3.0, 4.5, 7.6};
  double *ptrd = arrd;

  printf("sizeof arrc[] = %d \n", sizeof(arrc));
  printf("sizeof ptrc = %d \n", sizeof(ptrc));

  printf("sizeof arri[] = %d \n", sizeof(arri));
  printf("sizeof ptri = %d \n", sizeof(ptri));

  printf("sizeof arrf[] = %d \n", sizeof(arrf));
  printf("sizeof ptrf = %d \n", sizeof(ptrf));

  printf("sizeof arrd[] = %d \n", sizeof(arrd));
  printf("sizeof ptrd = %d \n", sizeof(ptrd));

  return 0;
}
```

Output

```
sizeof arrc[] = 3
sizeof ptrc = 4
sizeof arri[] = 12
sizeof ptri = 4
sizeof arrf[] = 12
sizeof ptrf = 4
sizeof arrd[] = 24
sizeof ptrd = 4
```

Example: 4

Description:Pointer arithmetics

Source Code

```
# include <stdio.h>
int main(void){
  float arr[5] = {12.5, 10.0, 13.5, 90.5, 0.5};
  float *ptr1 = &arr[0];
  float *ptr2 = ptr1 + 3;
  printf("%.2f \n", *ptr2);
  printf("%d \n\n", ptr2 - ptr1);

  ptr1++;
  ptr2--;
  printf("%.2f \n", *ptr1);
  printf("%.2f \n\n", *ptr2);
  return 0;
}
```

Output

```
90.50
3

10.00
13.50
```

3 Practice session

Sl	Source Code

Practice 1	`# include <stdio.h>`
	`int main(void){`
	`  int a;`
	`  char *x;`
	`  x = (char *) &a;`
	`  a = 512;`
	`  x[0] = 1;`
	`  x[1] = 2;`
	`  printf("%d\n",a);`
	`  return 0;`
	`}`

Practice 2	`# include <stdio.h>`
	`#define SIZE 4`
	`void fun(int *arrPtr, int size);`
	`int main(void){`
	`  int i;`
	`  int arr[SIZE] = {10, 20 ,30, 40};`
	`  fun(arr,SIZE);`
	`  return 0;`
	`}`
	`void fun(int *arrPtr, int size){`
	`  int i;`
	`  for (i = 0; i < size; i++)`
	`  printf("%d ", *(arrPtr+i));`
	`}`

Practice 3	`# include <stdio.h>`
	`void mystery(int *ptra, int *ptrb);`
	`int main(void){`
	`  int a=2016, b=0, c=4, d=42;`
	`  mystery(&a, &b);`
	`  if (a < c)`
	`  mystery(&c, &a);`
	`  mystery(&a, &d);`
	`  printf("%d\n", a);`
	`  return 0;`
	`}`
	`void mystery(int *ptra, int *ptrb) {`
	`  int *temp;`
	`  temp = ptrb;`
	`  ptrb = ptra;`
	`  ptra = temp;`
	`}`

Practice 4	`#include <stdio.h>`
	`int main(void){`
	`  int a = 300;`
	`  int *ptr = &a;`
	`  printf(" %d vs %d \n",a,*ptr);`
	`  printf(" %d vs %d \n",&a,ptr);`
	`  printf(" %d vs %d \n",&ptr,&*ptr);`
	`  printf(" %d vs %d \n",&a,*&ptr);`
	`  return 0;`
	`}`

4 Lab Assignments

1. What is pointer? What are the advantages of using a pointer?
2. Declare the following variables or functions in C
   (a) p is a pointer to a 10-element integer array.
   (b) p is a pointer to a function that accepts an argument which is a pointer to character array and returns a pointer to an integer quantity.
3. What are the difference between call by value and call by reference? Explain with a suitable example.
4. Write a C program which takes the radius of a circle from user and call a void function by reference which calculates the area and perimeter of the circle

---

Acknowledgment

First OBE version, prepared by:
B.K.M. Mizanur Rahman,
Assistant Professor,
Department of EEE, UIU

Second Update , prepared by:
Nazmul Alam,
Part Time Faculty,
Department of EEE, UIU