Lab # 08

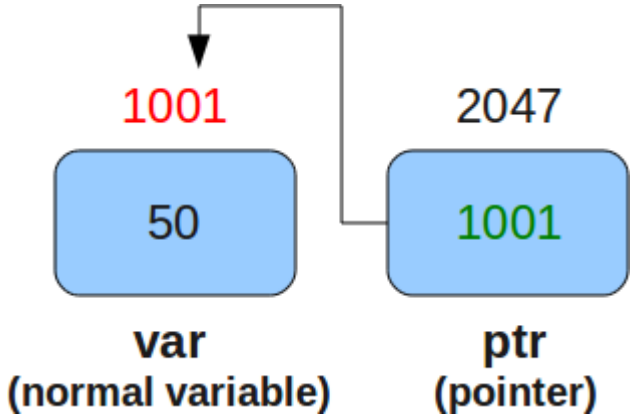
Introduction to Pointers

Objective: Objective of this lab is to understand

- What are Pointers?
- Functions and Array
- Pointers and functions
- Call by value
- Call by reference
- Array with pointers

What are Pointers?

Different from other normal variables which can store values, pointers are special variables that can hold the address of a variable. Since they store memory address of a variable, the pointers are very commonly said to "point to variables". Lets try to understand the concept.



As shown in the above diagram:

- A normal variable 'var' has a memory address of 1001 and holds a value 50.
- A **pointer variable** has its own address 2047 but stores 1001, which is the address of the variable 'var'

An example of a pointer declaration can be :

```
char *chptr;
```

the above declaration, 'char' signifies the pointer type, chptr is the name of the pointer while the asterisk '*' signifies that 'chptr' is a pointer variable.

For example :

```
char ch = 'c';
char *chptr = &ch; //initialize
OR char ch = 'c';
char *chptr;
chptr = &ch //initialize
```

In the code above, we declared a character variable ch which stores the value 'c'. Now, we declared a character pointer 'chptr' and initialized it with the address of variable 'ch'.

Note that the '&' operator is used to access the address of any type of variable.

Consider the following code :

Example 1:

```
#include <stdio.h>
int main(void) {
    char ch = 'c';
    char *chptr = &ch;
    int i = 20;
    int *intptr = &i;
    float f = 1.20000;
    float *fptr = &f;
    printf("\n [%c], [%d], [%f]\n", *chptr, *intptr, *fptr,);
    return 0;
}
```

OUTPUT:

```
[c], [20], [1.200000]
```

Example 2:

```
int main(void){
int i = 3 ;
int *j ;
j = &i ;
printf ( "\nAddress of i = %u", &i ) ;
printf ( "\nAddress of i = %u", j ) ;
printf ( "\nAddress of j = %u", &j ) ;
printf ( "\nValue of j = %u", j ) ;
printf ( "\nValue of i = %d", i ) ;
printf ( "\nValue of i = %d", *( &i ) ) ;
printf ( "\nValue of i = %d", *j ) ;
}
```

The output of the above program would be:

```
Address of i = 65524
Address of i = 65524
Address of j = 65522
Value of j = 65524
Value of i = 3
Value of i = 3
Value of i = 3
```

When we pass argument to functions there are two methods

Call by value

Call by reference

For call by reference we need to use pointers

Call by value:

Example 3:

```
void modify(int,int);
main()
{
int a,b;
printf("enter two numbers");
scanf("%d %d",&a,&b);
modify(a,b);
printf(" a and b in the main %d and %d",a,b);
}
void modify(int a,int b)
{
a=a*3;
b=b*3;
printf(" a and b in function %d and %d",a,b);
}
output:
enter two numbers
3
2
a and b in function 9 and 6
a and b in the main 3 and 2.
```

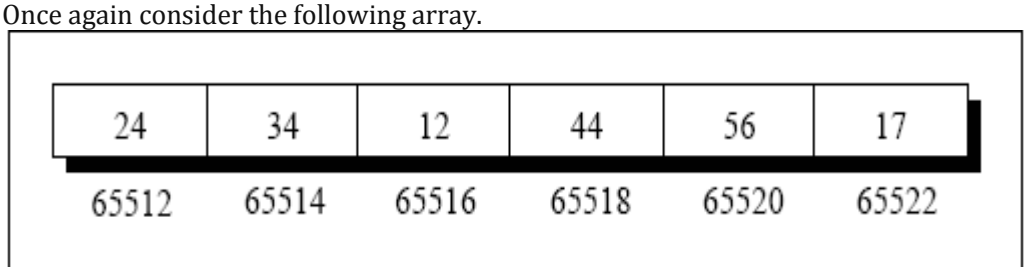
Call by reference:

Example 4:

```
void modify(int *,int *);
main()
{
int a,b;
printf("enter two numbers");
scanf("%d %d",&a,&b);
modify(&a,&b);
printf(" a and b in the main %d and %d",a,b);
}
void modify(int *a,int *b)
{
*a=*a * 3;
*b=*b * 3;
printf(" a and b in function %d and %d",a,b);
}
output:
enter two numbers
3
2
a and b in function 9 and 6
a and b in the main 9 and 6.
```

Array with pointers:

Once again consider the following array.



This is how we would declare the above array in C,
int num[ ] = { 24, 34, 12, 44, 56, 17 } ;
We also know that on mentioning the name of the array we get its base address. Thus, by saying *num we would be able to refer to the zeroth element of the array, that is, 24. One can easily see that
*num and *(num + 0) both refer to 24.
Similarly, by saying *(num + 1) we can refer the first element of the array, that is, 34. In fact, this is what the C compiler does internally. When we say, num[i], the C compiler internally
converts it to *(num + i). This means that all the following notations are same:

num[i]
*(num + i)
*(i + num)
i[num]

Example 5:

```
/* Accessing array elements in different ways */
main( )
{
int num[ ] = { 24, 34, 12, 44, 56, 17 } ;
int i ;
for ( i = 0 ; i <= 5 ; i++ )
{
printf ( "\naddress = %u ", &num[i] ) ;
printf ( "element = %d %d ", num[i], *( num + i ) ) ;
printf ( "%d %d", *( i + num ), i[num] ) ;
}
}
The output of this program would be:
address = 65512 element = 24 24 24 24
address = 65514 element = 34 34 34 34
address = 65516 element = 12 12 12 12
address = 65518 element = 44 44 44 44
address = 65520 element = 56 56 56 56
address = 65522 element = 17 17 17 17
```

strings Library Functions

Following are the functions defined in the header string.h:

S.N.	Function & Description
1	**void *memchr(const void *str, int c, size_t n)** Searches for the first occurrence of the character c (an unsigned char) in the first n bytes of the string pointed to, by the argument *str*.
2	**int memcmp(const void *str1, const void *str2, size_t n)** Compares the first n bytes of *str1* and *str2*.
3	**void *memcpy(void *dest, const void *src, size_t n)** Copies n characters from src to *dest*.
4	**void *memmove(void *dest, const void *src, size_t n)** Another function to copy n characters from *str2* to *str1*.
5	**void *memset(void *str, int c, size_t n)** Copies the character c (an unsigned char) to the first n characters of the string pointed to, by the argument *str*.
6	**char *strcat(char *dest, const char *src)** Appends the string pointed to, by *src* to the end of the string pointed to by *dest*.
7	**char *strncat(char *dest, const char *src, size_t n)** Appends the string pointed to, by *src* to the end of the string pointed to, by *dest*up to n characters long.
8	**char *strchr(const char *str, int c)** Searches for the first occurrence of the character c (an unsigned char) in the string pointed to, by the argument *str*.
9	**int strcmp(const char *str1, const char *str2)** Compares the string pointed to, by *str1* to the string pointed to by *str2*.
10	**int strncmp(const char *str1, const char *str2, size_t n)** Compares at most the first n bytes of *str1* and *str2*.
11	**int strcoll(const char *str1, const char *str2)** Compares string *str1* to *str2*. The result is dependent on the LC_COLLATE setting of the location.
12	**char *strcpy(char *dest, const char *src)** Copies the string pointed to, by *src* to *dest*.
13	**char *strncpy(char *dest, const char *src, size_t n)** Copies up to n characters from the string pointed to, by *src* to *dest*.
14	**size_t strcspn(const char *str1, const char *str2)** Calculates the length of the initial segment of str1 which consists entirely of characters not in str2.
15	**char *strerror(int errnum)** Searches an internal array for the error number errnum and returns a pointer to an error message string.

16	**size_t strlen(const char *str)** Computes the length of the string str up to but not including the terminating null character.
17	**char *strpbrk(const char *str1, const char *str2)** Finds the first character in the string *str1* that matches any character specified in*str2*.
18	**char *strrchr(const char *str, int c)** Searches for the last occurrence of the character c (an unsigned char) in the string pointed to by the argument *str*.
19	**size_t strspn(const char *str1, const char *str2)** Calculates the length of the initial segment of *str1* which consists entirely of characters in *str2*.
20	**char *strstr(const char *haystack, const char *needle)** Finds the first occurrence of the entire string *needle* (not including the terminating null character) which appears in the string *haystack*.
21	**char *strtok(char *str, const char *delim)** Breaks string *str* into a series of tokens separated by *delim*.
22	**size_t strxfrm(char *dest, const char *src, size_t n)** Transforms the first **n** characters of the string **src** into corrent locale and places them in the string **dest**.

Exercise:

1. Execute all examples given in the manual.
2. Write a program to swap two variables by passing the reference of these variables into a function declared as void swap(int *, int *).
3. Write a program to take numbers in array and access them through pointers. Pass data to a function and then function will sort the data of array using bubble sort.
4. Write a program which takes radius of a circle input from the user and pass it to a function called **AreaPerimeter**(). This function calculates the area and perimeter of a circle and return both, area and perimeter, to **main**().
5. An array of five elements passed through a function **sum**(). Calculate the sum and returns sum to main to print in the **main**().
6. Write a single function that receives an array of 5 integers and returns the sum, average and standard deviation of these numbers **without using return statement**. Call this function from **main**( ) and print the results in **main**( ).

$$\sigma = \sqrt{\frac{\sum(x-\mu)^2}{N}}$$

Where x represents each value in the population, μ is the mean value of the population, Σ is the summation (or total), and N is the number of values in the population.

7. Write a modular program that accepts at least 10 integer test scores from the user and stores them in an array. Then main should display how many perfect scores were entered (i.e., scores of 100), using a value-returning **countPerfect** function to help it.
8. A local zoo wants to keep track of how many pounds of food each of its three monkeys eats each day during a typical week. Write a program that stores this information in a two dimensional 3×7 array, where each row represents a different monkey and each column represents a different day of the week. The program should first have the user input the data for each monkey. Then it should create a report that includes the following information:
   • Average amount of food eaten per day by the whole family of monkeys.
   • The least amount of food eaten during the week by any one monkey.
   • The greatest amount of food eaten during the week by any one monkey.
9. Write a program that can be used to gather statistical data about the number of movies college students see in a month. The program should ask the user how many students were surveyed and dynamically allocate an array of that size. The program should then allow the user to enter the number of movies each student has seen. The program should then calculate the average, median, and mode of the values entered.
10. Write a program to sort a set of names stored in an array in alphabetical order.
11. Write a program to reverse the strings.