

Operating Systems

LAB 4

Introduction to C Programming

Objective:

To gain experience with:

1. Writing simple C programs with more than one function
2. Basic concepts of Pointers in C
3. Using Arrays in C
4. Using Structures in C
5. Dynamic Memory Allocation
6. Use of Linked List

Purpose:

To gain experience about C programming

1. A Simple C program with more than one function (Parameters passed by value)

Read:

```
int scanf ( const char * format, ... );
```

This function reads data from stdin and stores them according to the parameter format into the locations pointed by the additional arguments.

Display:

```
int printf ( const char * format, ... );
```

This function writes the C string pointed by format to the standard output (stdout). If format includes format specifiers (subsequences beginning with %), the additional arguments following format are formatted and inserted in the resulting string replacing their respective specifiers.

<i>specifier</i>	Output	Example
d or i	Signed decimal integer	392
u	Unsigned decimal integer	7235
o	Unsigned octal	610
x	Unsigned hexadecimal integer	7fa
f	Decimal floating point	392.65
e	Scientific notation (mantissa/exponent)	3.9265e+2
a	Hexadecimal floating point, lowercase	-0xc.90fep-2
c	Character	a
s	String of characters	sample
p	Pointer address	b8000000

Task # 1: Write a program reads a number from user and finds its factorial using function. Pass the argument to function by value.

2. Basic concepts of Pointers in C

Every variable in C has a name (variable name), a memory location (to store the data), and an address.

For the variable declaration

```
int num;
```

num -----> variable name

memory

8152 -----> address

A variable used to store the address value is called as the Pointer. It can be defined as

```
int *ptr;
```

Task 2: The following program demonstrates about the pointer variable, * and & operators. Run and observe the output.

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

    printf("Enter an Integer: ");
    scanf("%d", &a);

    p=&a;

    printf("The value of the variable a is %d\n", a);
    printf("The address of the variable a is %x\n", &a);
    printf("The value of variable p is %x\n", p);
    printf("The value pointed by p is *P = %d\n", *p);
    printf("The address of p is %x\n", &p);
    return(0);
}
```

Task 3: Redo task number 1. The result should be passed by pointer.

3. Using Arrays in C

Arrays are important to C and should need a lot more attention. The following important concepts related to array should be clear to a C programmer.

S.N.	Concept & Description
1	Multi-dimensional arrays C supports multidimensional arrays. The simplest form of the multidimensional array is the two-dimensional array.
2	Passing arrays to functions You can pass to the function a pointer to an array by specifying the array's name without an index.
3	Return array from a function C allows a function to return an array.
4	Pointer to an array You can generate a pointer to the first element of an array by simply specifying the array name, without any index.

Task 4: Write a function that calculates the dot product of two dimensional array. Call this function from main() function and display the product.

4. Using Structures in C

Arrays allow defining type of variables that can hold several data items of the same kind. Similarly structure is a collection/group of different/same variables.

Task 5: Run the following program and observe the output.

Program

```
#include<stdio.h>
main(){
    struct student {
        char name[20];
        int id;
    };

    struct student s1, s2, s3;

    printf("Please enter the student name, and id\n");
    scanf("%s %d", &s1.name, &s1.id);
    scanf("%s %d", &s2.name, &s2.id);
    scanf("%s %d", &s3.name, &s3.id);

    printf("\nThe student details");
    printf("\n%s \t\t%d", s1.name, s1.id);
    printf("\n%s \t\t%d", s2.name, s2.id);
    printf("\n%s \t\t%d", s3.name, s3.id);

}
```

Sample output

```
colonel$ ./structure
Please enter the student name, and id
Ahamed 9876
Ali 9979
Yahya 9988

The student details
Ahamed          9876
Ali             9979
Yahya           9988
```

Task 6: Write a C code to declare “Time” structure that contains hour, minute and seconds as its data members. Write a function that adds two time instances and return the resultant time to the main function.

5. Dynamic Memory Allocation

Using malloc to obtain memory at run-time:

- Memory can be allocated dynamically (at run-time) using the function **malloc()** – accessible through **<stdlib.h>**
- The allocation is made from a special memory area called the **heap**.
- The function, **malloc()** returns a pointer (address) to the allocated storage.
- However, **malloc()** does not associate any type to the pointer it returns – it is said to be **void**.
- For the pointer to be useful, it must be associated with a type using casting .e.g.,

```
int *int_ptr;  
int_ptr=(int *) malloc(4);  
*int_ptr =17;
```
- The above statements reserve four bytes and return the address of first byte, cast it to **int** and assign it to integer pointer **int_ptr**.
- Since the bytes allocated to **int** is system-dependent, it is safer to use the function **sizeof ()** to get the actual number of bytes associated with the particular type being considered.
- **sizeof()** is system-independent and can be used even with user-defined types.
- Thus, the above statements are better represented as follows:

```
int *int_ptr;  
int_ptr=(int *) malloc(sizeof(int));  
*int_ptr =17;
```
- Note that there is no name associated with the memory obtained by **malloc**. It can only be accessed as ***int_ptr**. It is sometimes called **anonymous** variable.
- Thus, should **int_ptr** be given another address, the location (returned by **malloc**) will be lost. It can neither be accessed by the program nor by the system. It is said to be a **lost** object.
- When we no longer need a dynamic variable, we can return the storage it occupies using the **free()** function.
e.g. **free(int_ptr);**



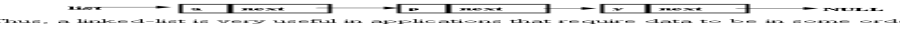
Task 7: Write a program that takes the size of the array as input from the user, create the array and then take the elements of array as input and sort in ascending order.

6. Use of Linked List in C

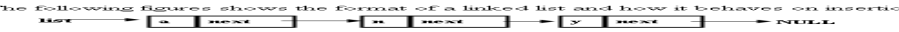


A linked list is a sequence of data structures, which are connected together via links. Linked list, like stack and queue is a homogeneous linear list consisting of nodes in which each node is linked to the next.

However, unlike stack and queue, an item can be deleted at any location in the list, and can be added (inserted) at any location provided the order of the items in the list is maintained.

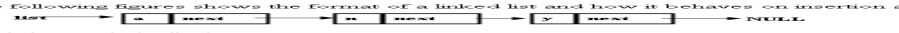
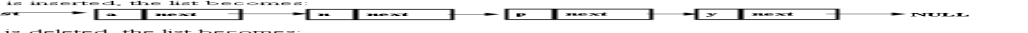

The following figure shows the format of a linked list and how it behaves on insertion and deletion:

- > The following figures shows the format of a linked list and how it behaves on insertion and deletion.

- > If p is inserted, the list becomes:

- > If n is deleted, the list becomes:

- > Thus, a linked-list is very useful in applications that require data to be in some order at all times.

If p is inserted, the list becomes:

- > The following figures shows the format of a linked list and how it behaves on insertion and deletion.

- > If p is inserted, the list becomes:

- > If n is deleted, the list becomes:

- > Thus, a linked-list is very useful in applications that require data to be in some order at all times.

If n is deleted, the list becomes:

- > The following figures shows the format of a linked list and how it behaves on insertion and deletion.

- > If p is inserted, the list becomes:

- > If n is deleted, the list becomes:

- > Thus, a linked-list is very useful in applications that require data to be in some order at all times.

Thus a linked-list is very useful in applications that require data to be in some order at all times.

Task 8: Write a complete menu driven program to do the following:

- Build a linked list to save a list of names. Name will not exceed 50 characters.
- Write a function `add` to append a new name to the list. The function prototype is given as

```
void add (list *head, char *newname);
```

- Write a function `search` to look for a given name in the list. If that name is found in list then the function should return true, otherwise, return false.
- Write a main method to test your two functions.

In C language, the boolean type and the boolean literals (true, false) are not defined. We can define these in our program as follow:

```
typedef enum {false = 0, true} boolean;
```

The skeleton of your program should look like the following:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct list {
    ...
    ...
} list;

typedef enum {false=0, true} boolean;
void add (list *, char *);
boolean search (list *, char *);

int main()
{
    ...
    ...
}

void add (list *head, char *newname)
{
    ...
}
boolean search (list *head, char *name)
{
    ...
}
```