LAB 4

Experiment No. 1

TITLE:

To demonstrate different components of a function.

OBJECTIVE:

- To learn about function in C programming.
- To demonstrate program using function.

THEORY:

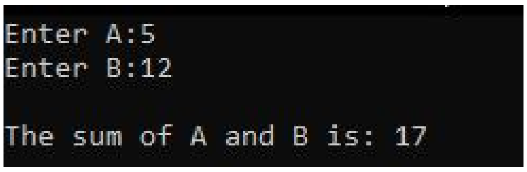
Functions in C are the basic building blocks of a C program. A function is a set of statements enclosed within curly brackets ({}) that take inputs, do the computation, and provide the resultant output. You can call a function multiple times, thereby allowing re-usability and modularity in C programming.

To demonstrate different components of C, here the simple example for sum of two digits.

PROGRAM:

```
#include <stdio.h>

int sum(int, int); //function decleration...
void main() // main funcrion...
{
   int a,b,s; // variable decleration...

   printf("Enter A:");
   scanf("%d",&a);
   printf("Enter B:");
   scanf("%d",&b);

   s = sum(a,b); //fuction call...
   printf("\nThe sum of A and B is: %d",s);
}

int sum(int x,int y) //function definition....
{
   return (x+y);
}
```

Output:



```
Enter A:5
Enter B:12

The sum of A and B is: 17
```

RESULTS AND DISCUSSION:

The experiment was successful to demonstrate function in C program. The two integer sum program served as a basic introduction to function C programming language.

CONCLUSION:

This laboratory exercise provided a hands-on experience in use of function in C program. Students gained practical knowledge of the basic of function in C programming and are now better equipped to undertake more complex programming tasks in the future.

Experiment No. 2

TITLE: To demonstrate different categories of a function.

OBJECTIVE:

- To explain and demonstrate different categories of a function in C programming.
- To understand and implement functions categories.

THEORY:

Basically there are two types of function in C programming.
1. In-Build / Pre-defined function
2. User defined function.

In-Build / Pre-defined function : The functions which are already defined in the library and we can use it anywhere within our porgram by calling it. It id defined in the preprocessor directory file which is saved with .h extension.

User defined function : The type of function which is defined by the programmer in the same program where we want to use it. The syntax to define the function is given below:

Syntax:

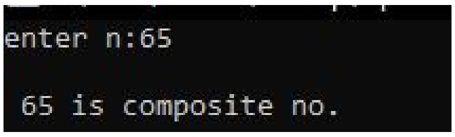
```
data_type function_name(list of parameter){
        // block of statement…
}
```

PROGRAM:

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

void prime_compoite(int n) // function decleation and definition…
{
        int i;
        for(i=2;i<=n/2;i++)
        {
                if(n%i==0){
                        printf("\n %d is composite no.",n);
                        exit(0);
                }
        }
        printf("\n %d is Prime No.",n);
}
```

```
void main(){
        int n;
        printf("enter n:");
        scanf("%d",&n);
        prime_compoite(n);
}
```

Output:



```
enter n:65

 65 is composite no.
```

RESULTS AND DISCUSSION:

In this above program, the user give the number and the output comes the given number is prime or composite. There are two pre processor directory are imported in which printf() and scanf() function are from stdio.h and exit(0) function from stdlib.h. Here we create the user defined function called prime_composite() having single paramete (i.e. int n) which is responsible to find the given number is prime or composite.

CONCLUSION:

This laboratory exercise provided practical experience of different categories of function in C programming. Now students can perform different categories of function and make the useful programs in C programming.

Experiment No. 3

TITLE: To demonstrate passing array to function.

OBJECTIVE:

- To understand how to pass array to function.
- To learn various operation using passing array to function.

THEORY:

In C, there are various general problems which requires passing more than one variable of the same type to a function. As we know that the array_name contains the address of the first element. Here, we must notice that we need to pass only the name of the array in the function which is intended to accept an array. The array defined as the formal parameter will automatically refer to the array specified by the array name defined as an actual parameter.

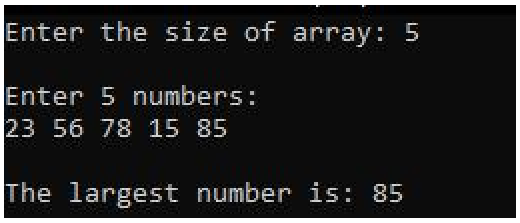
syntax

```
functionname(arrayname);//passing array
```

PROGRAM:

```
#include<stdio.h>

void input(int x[],int n){
        int i;
        // initialization of array elements.....
        printf("\nEnter %d numbers: \n",n);
        for(i=0;i<n;i++)
                scanf("%d",&x[i]);
}

int largest(int x[],int n){
        int i , largest = x[0];
        // Largest value finding operation...
        for(i=0;i<n;i++){
                if(x[i] > largest)
                        largest = x[i];
        }
        return largest;
}
int main(){
        int a[100],n,large;

        printf("Enter the size of array: ");
        scanf("%d",&n);
        input(a,n);
        large = largest(a,n);

        printf("\nThe largest number is: %d",large);
}
```

Output:



```
Enter the size of array: 5

Enter 5 numbers:
23 56 78 15 85

The largest number is: 85
```

RESULTS AND DISCUSSION:

The program prompts the user to input the size of an array and then the array elements. It finds and prints the largest number in the array. The largest number is determined by repeating through the array once and updating a variable whenever a larger number is meet.

CONCLUSION:

In this laboratory experiment, we successfully demonstrated the concept of passing arrays to functions in C programming. By passing the array name as a parameter to a function, we can manipulate the array elements within the function. This method allows for modular and reusable code, which makes it easier to use arrays in complex programs. To develop efficient and maintainable C programs, it is important to understand how arrays are passed from one function to another.

Experiment No. 4

TITLE:

To demonstrate declaration, initialization and accessing pointer variable.

OBJECTIVE:

- To learn how to declare pointer variable in C programming.
- To learn about initializing and accessing pointer variable.

THEORY:

A pointer variable in C stores memory addresses rather than data values. It holds the location of another variable in memory. Declaring a pointer variable involves specifying the data type it points to followed by an asterisk (*).

syntax

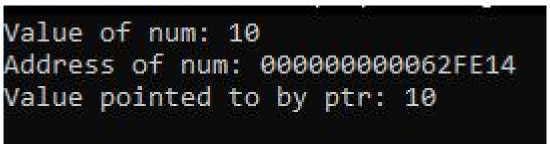
*data_type *pointer_variable;*

Initialization involves assigning the address of another variable to the pointer using the address-of operator (&).
Accessing the value stored at the memory location pointed to by a pointer is done using the dereference operator (*).

PROGRAM:

```
#include <stdio.h>

int main() {
   int num = 10;
   int *ptr; // Declare a pointer to an integer..

   // Initialize the pointer with the address of num
   ptr = &num;

   printf("Value of num: %d\n", num); // Print the value of num
   printf("Address of num: %p\n", &num); // Print the address of num

   // Access and print the value stored at the address pointed to by ptr
   printf("Value pointed to by ptr: %d\n", *ptr);

   return 0;
}
```

Output:



```
Value of num: 10
Address of num: 000000000062FE14
Value pointed to by ptr: 10
```

RESULTS AND DISCUSSION

We created a pointer that stored the memory address of a regular variable. We then used the pointer to access and modify the value of the original variable indirectly. The key points of pointer are:

- Pointers store memory addresses.
- The address-of operator (&) gets the memory address of a variable.
- The dereference operator (*) accesses the value stored at the address a pointer points to.
- Modifying the value through a pointer affects the original variable.

CONCLUSION

This lab successfully demonstrated the declaration, initialization, and accessing of pointer variables in C. We learned how to declare pointers, assign memory addresses, and use the dereference operator to work with data indirectly through pointers.

Experiment No. 5

TITLE: To demonstrate passing pointer to a function.

OBJECTIVE:

- To learn how to pass pointers to functions in C programming.

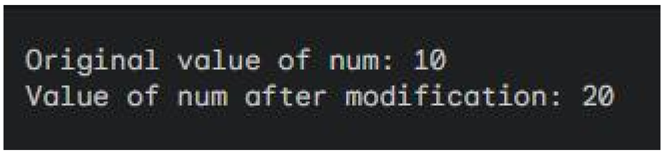
THEORY:

By default, arguments passed to functions in C are passed by value. This means a copy of the variable is passed, and changes made within the function only affect the copy. Passing pointers allows us to pass the memory address of a variable, enabling the function to directly modify the original data.

PROGRAM:

```
#include <stdio.h>

void update(int *ptr) {
   // Modify the value at the memory location pointed to by ptr
   *ptr = *ptr * 2;
}

int main() {
   int num = 10;

   printf("Original value of num: %d\n", num);

   // Pass the address of num to change the pointer value…
   update(&num);

   printf("Value of num after modification: %d\n", num);

   return 0;
}
```

Output:



```
Original value of num: 10
Value of num after modification: 20
```

RESULTS AND DISCUSSION:

- We define a function update() that takes an integer pointer (int *ptr) as a parameter.
- In the main function, we declare an integer num and initialize it with 10.
- We call update(), passing the address of num using the address-of operator (&).
- Inside update, we use the dereference operator (*) to access the value stored at the memory location pointed to by ptr. We then double that value.
- Since ptr points to the original num, modifying the value through the pointer directly affects num in the main function.
- The output demonstrates that the original value of num (10) is doubled after the function call.

CONCLUSION:

This lab successfully demonstrated passing pointers to functions in C. We learned how to modify variables outside the function by passing their addresses and using the dereference operator.

Experiment No. 6

TITLE: To demonstrate relationship of array and pointer.

OBJECTIVE:

- To understand the connection between arrays and pointers in C programming.
- To demonstrate how array names and pointers can be used interchangeably in some scenarios.

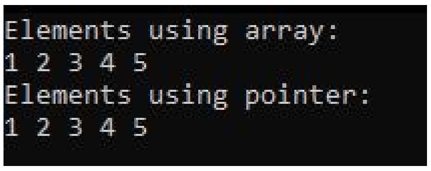
THEORY:

Arrays and pointers are closely related in C. The name of an array without square brackets actually decays to a constant pointer pointing to the first element of the array. This allows us to use array names and pointers in similar ways for certain operations.

PROGRAM:

```
#include <stdio.h>

int main() {
   int numbers[5] = {1, 2, 3, 4, 5};
        int *ptr = numbers; // pointer to points the first element

   // Print elements using array indexing
   printf("Elements using array:\n");
   for (int i = 0; i < 5; i++) {
      printf("%d ", numbers[i]);
   }
   printf("\n");

   // Print elements using pointer arithmetic
   printf("Elements using pointer:\n");
   for (int i = 0; i < 5; i++) {
      printf("%d ", *(ptr + i));
   }
   printf("\n");

   return 0;
}
```

Output:



```
Elements using array:
1 2 3 4 5
Elements using pointer:
1 2 3 4 5
```

RESULTS AND DISCUSSION:

- We declare an integer array numbers with five elements.
- We print the elements using a loop and array indexing (numbers[i]).
- We then declare a pointer ptr and initialize it with numbers. This is because the name of the array numbers decays to a pointer to the first element (similar to &numbers[0]).
- We use a loop with pointer arithmetic to access and print the elements.
  - ptr + i adds an offset to ptr for each iteration, effectively pointing to subsequent elements in the array.
  - *(ptr + i) dereferences the pointer at the current position to access the value stored there.

CONCLUSION:

This lab demonstrates the relationship between arrays and pointers in C. We observed how array names can be used as pointers to the first element and how pointer arithmetic allows us to iterate through an array.

Experiment No. 7

TITLE: To use dynamic memory allocation.

OBJECTIVE:

- To understand the concept of dynamic memory allocation using the in C programming.

THEORY:

Static memory allocation in C allocates memory for variables at compile time. However, sometimes the program may not know the size of the data it needs to store beforehand. Dynamic memory allocation allows allocating memory during program execution using functions like malloc.

malloc takes the size of the memory block (in bytes) as input and returns a void pointer to the allocated memory. It's crucial to remember to free the allocated memory later using free to prevent memory leaks.

syntax

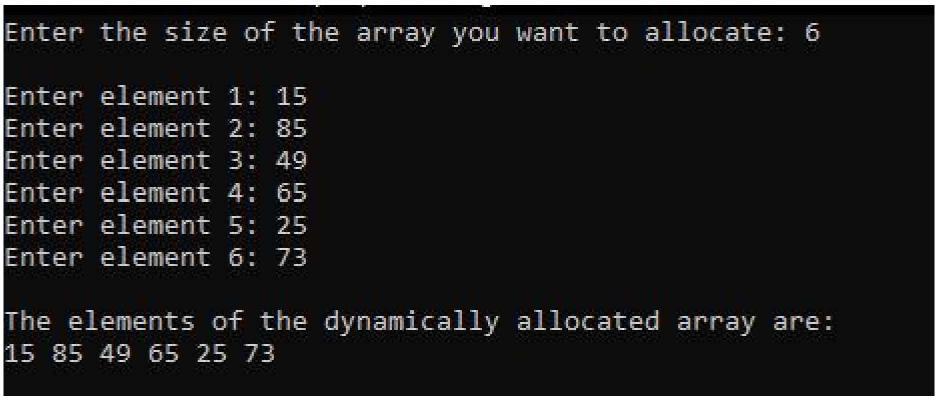
```
data_type *pointer_variable = (data_type*)malloc(size_of_memory);
```

PROGRAM:

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

int main() {
   int *ptr;
   int size;

   printf("Enter the size of the array you want to allocate: ");
   scanf("%d", &size);

   printf("\n");
   // Allocate memory for an integer array of size 'size'
   ptr = (int*)malloc(size * sizeof(int));

   // Check if memory allocation was successful
   if (ptr == NULL) {
      printf("Memory allocation failed!\n");
      return 1;
   }

   // Access and modify elements of the dynamically allocated array
   for (int i = 0; i < size; i++) {
      printf("Enter element %d: ", i + 1);
      scanf("%d", &ptr[i]); // Access elements using pointer arithmetic
   }
```

```
    printf("The elements of the dynamically allocated array are: \n");
    for (int i = 0; i < size; i++) {
        printf("%d ", ptr[i]);
    }
    printf("\n");

    // Free the allocated memory to prevent memory leaks
    free(ptr);

    return 0;
}
```

Output:



```
Enter the size of the array you want to allocate: 6

Enter element 1: 15
Enter element 2: 85
Enter element 3: 49
Enter element 4: 65
Enter element 5: 25
Enter element 6: 73

The elements of the dynamically allocated array are:
15 85 49 65 25 73
```

RESULTS AND DISCUSSION:

The program prompts the user to input the size of an array and then the array elements. In the background, the memory was dynamically allocated for array elements. This program simply print the all elements entered by the user from dynamically allocated memory.

CONCLUSION:

This lab successfully demonstrated dynamic memory allocation in C using malloc. We learned how to allocate memory for data structures like arrays at runtime based on user input. We also emphasized the importance of freeing the allocated memory to prevent memory leaks.