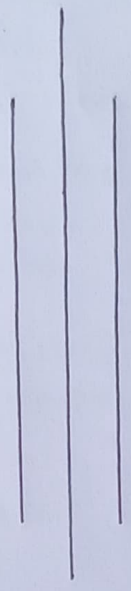
TRIBHUVAN UNIVERSITY

Faculty of Humanities & Social Science

HIMALAYA COLLEGE OF ENGINEERING

Chyasal, Lalitpur



C Programming Lab Report 5 : Arrays

Submitted by: Sujal Gurung

Roll no: 34

BCA 2nd Semester

OBJECTIVES:

- To learn to work with arrays, initialize & access individual elements
- To apply knowledge of 2-dimensional arrays for performing matrix operations.

THEORY:

An array is a homogenous data structure that can hold multiple elements of same data type, under a single identifier.

Initialization: data type identifier[dimension][d_2]..[d_n]

One dimensional arrays can be thought of as lists, while 2D arrays can be thought of as matrixes or an array of multiple 1D arrays.

Array elements are assigned adjacent memory locations & can be accessed using their index value, starting from 0.

For example, the 5th element might be accessed using a[4].

PROGRAMS:

- WAP to find sum of 10 elements of an array Read elements from the user.

```
#include <stdio.h>
void main()
{
    int a[10],i,sum=0;
    printf("Enter elements of array");
    for(i=0;i<10;i++)
    {
        scanf("%d",&a[i]);
        sum = sum + a[i];
    }
    printf("Sum is %d",sum);
}
```

- WAP to add 2 matrices A & B. Display result in matrix format.

```
#include<stdio.h>
void main()
{
    int i,j, a[2][3], b[2][3];
    printf("Enter elements of matrix A");
    for (i=0;i<2;i++)
    {
        for(j=0;j<3;j++)
        {
            scanf("%d", &a[i][j]);
        }
    }
~~for(j=0;j~~
    printf("Enter elements of matrix B");
    for (i=0;i<2;i++)
    {
        for(j=0;j<3;j++)
        {
            scanf("%d", &b[i][j]);
        }
    }
    printf("The added matrix is:\n");
    for (i=0;i<2;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("%d\t", a[i][j]+b[i][j]);
        }
        printf("\n");
    }
}
```

- WAP to sort contents of a 1 dimensional array in ascending order.

```
// Here, we use bubble sort.
#include <stdio.h>
void main()
{
    int i,j,temp,a[5];
    printf("Enter elements of array");
    for(i=0;i<5;i++)
    {
        scanf("%d",&a[i]);
    }
    for(i=0;i<4;i++)
    {
        for(j=0;j<4;j++)
        {
            if(a[j]>a[j+1])
            {
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
    }
    for(i=0;i<5;i++)
    printf("The sorted array is:\t");
    for(i=0;i<5;i++)
    {
        printf("%d\t",a[i]);
    }
}
```

CONCLUSION

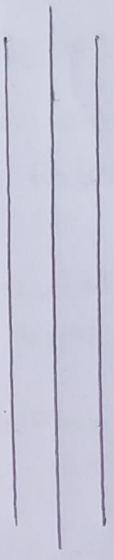
Thus, we were able to practically work with arrays & perform various operations. We also successfully conducted mathematical matrix operations & bubble sort algorithm.

TRIBHUVAN UNIVERSITY

Faculty of Humanities & Social Sciences

HIMALAYA COLLEGE OF ENGINEERING

Chyasal, Lalitpur



C Programming Lab Report 6 : Functions

Submitted by: Sujal Gurung

Roll No: 34

#OBJECTIVES

- To learn how to use user-defined functions
- To learn hao to work with variations of return types & arguement types.

#THEORY:

A function is a module / block of code that gets executed only when it is called. void main() itself is a function that the compiler executes first. printf(), scanf() are also built in functions that have specific uses.

Aside from these, we can define our own function with custom statements/actions. First, we need to declare it like so:

return_type function_identifier(data type 1 arguement 1, ... data type n arguement n);

Later, we need to define it by specifing what commands should be exeated upon calling.

If the function returns a value, we need to specify its data type. Similarly, if we send values or variables to the function, we specify it ~~inside~~ as arguements.

Here, actual arguements are what we send while calling function. Formal arguements are variables inside function for holding values from actual arguements.

#PROGRAMS

① WAP to add 2 numbers using function having return type & arguements.

```
- #include <stdio.h>
  int add(int a, int b);
  void main()
  {
      int x,y;
      printf("Enter 2 numbers");
      scanf("%d %d", &x, &y);
```

```
    printf("The sum is %d", add(x,y));
}
int add(int a, int b)
{
    return a+b;
}
```

(2) WAP to subtract 2 numbers. Use function with arguements but no return type.

```
#include<stdio.h>
void sub(int a, int b);
void main()
{
    int x,y;
    printf("Enter 2 numbers");
    scanf("%d %d", &x, &y);
    sub(x,y);
}
void sub(int a, int b)
{
    printf("Their difference is %d", a-b);
}
```

(3) WAP to multiply 2 numbers. Use function with return type but no arguements.

```
#include <stdio.h>
int multi();
void main()
{
    int m;
    m= multi();
    printf("Their product is %d", m);
}
int multi()
{
    int a,b;
    printf("Enter 2 numbers");
    scanf("%d %d", &a, &b);
    return a*b;
}
```

(4) WAP to divide 2 numbers. Use function with no return type nor arguements.

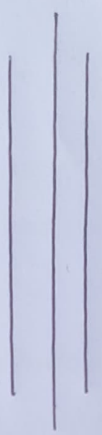
```
#include<stdio.h>
Void div();
void main()
{
   div();
}
void div()
{
   float a,b;
   printf("Enter 2 numbers");
   scanf("%d %d", &a, &b);
   printf("The quotient is %f", a/b);
}
```

CONCLUSION

Thus, we were able to learn how to use functions as well as how to write programs based on function's return type & arguements.

TRIBHUVAN UNIVERSITY
Faculty of Humanities & Social sciences

HIMALAYA COLLEGE OF ENGINEERING
Chyasal, Lalitpur



C Programming Lab Report 8: Structures

Submitted by: Sajal Gurung
Roll no: 34
BCA 2nd Semester

OBJECTIVES:

- To learn to work with structures
- To learn how to access & sort ~~structu~~ array of structures.

THEORY:

Structures are heterogenous data structures where multiple elements of varying data types can be held under a single identifier.

Syntax:

```
struct <identifier>
{
    datatype 1  member1;
    data type n  membern;
} variable;
```

Here, the members are what hold specified data types. The structure variable is used to access these members using the dot operator.

Example: s.name
where s is a struct variable & name is a member.

PROGRAMS

(1) Create a structure named student that has name, roll & marks as members. Assume approximate types and size of members. Use this structure to read & display records of 5 students.

```
#include <stdio.h>
struct student
{
    int roll, marks;
    char name[20];
} s[5];
void main()
{
    int i;
    for (i=0 ; i<5; i++)
    {
        printf ("Enter name, roll & marks of student %d", i+1);
        scanf ("%[^\n]", s[i].name];
```

```
        scanf("%d %d", &s[i].roll, &s[i].marks);
    }
    printf("\n Name: \t Roll no. \t Marks \n");
    for (i=0; i<5; i++)
    {
        printf("%s \t %d \t \t %d \n", s[i].name,
                s[i].roll, s[i].marks);
    }
}
```

② From above program, display record only of student having highest marks.

```
#include <stdio.h>
struct student
{
    int roll, marks;
    char name[20];
} s[5];
void main()
{
    int i, max;
    printf
    for (i=0; i<5; i++)
    {
        printf("Enter name, roll no. & marks of student
                %d", i+1);
        scanf("%[^\n]", s[i].name);
        scanf("%d %d", &s[i].roll, &s[i].marks);
    }
    max=0;
    for (i=0; i<5; i++)
    {
        if (s[i].marks > s[max].marks)  max=i;
    }
    printf("%s \t %d \t %d", s[max].name, s[max].roll, s[max]
            .marks");
}
```

② For the program in question 1, sort the records in ascending order based on name.

```
#include <stdio.h>
#include <string.h>
struct student
{
    int roll, marks;
    char name[20];
} s, temp;
void main()
{
    int i, j;
    for(i=0; i<5; i++)
    {
        printf("Enter name, roll & marks of student %d", i+1);
        scanf("%[^\n]", s[i].name);
        scanf("%d %d", &s[i].roll, &s[i].marks);
    }
    for (i=0; i<4; i++)   // using bubble sort
    {
        for(j=0; j<4; j++)
        {
            if (strcmp(s[j].name, s[j+1].name)>0)
            {
                temp = s[j];
                s[j] = s[j+1];
                s[j+1] = temp;
            }
        }
    }
    printf("Ascending order:\n");
    for(i=0; i<5; i++)
    {
        printf("%s\t%d\t%d\n", s[i].name, s[i].roll, s[i].marks);
    }
}
```

#CONCLUSION:

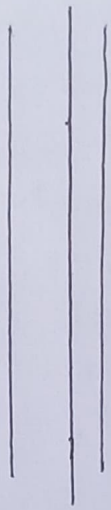
Thus we learned to work with structures & implement old knowledge of conditional statements, sorting here.

TRIBHUVAN UNIVERSITY

Faculty of Humanities & Social Sciences

HIMALAYA COLLEGE OF ENGINEERING

Chyasal, Lalitpur



C Programming Lab Report 9: File Handling

Submitted by: Sujal Gurung

Roll no: 34

OBJECTIVES:

* To learn to read/write data to/from files.
* To learn differences between & when to use text & binary files

THEORY:

Till now, all data from our programs is stored in memory only temporarily. This is referred to as console I/O, where data is lost after program execution ends.

We can store data for future use with files in secondary memory. We have 2 options/file formats to choose from: text files & binary files. Text files are simple, human readable files & we can easily modify their data without having to write a program. Binary files have special uses like storing data from structures but aren't easily readable by us.

We have some file handling functions similar to the ones we have been using for console I/O such as:

* Unformatted:
  - fgets()
    - ↳ takes input/gets string from file & stores to a variable
    - ↳ syntax: fgets(write variable, limit, file ptr);
  - fputs()
    - ↳ gives output/puts string to file, from specified string or variable
    - ↳ syntax: fputs(read variable, file ptr);
* Formatted:
  - These allow us to customize how output looks or specify what type of input to take.

- fscanf
  - ↳ take input/scan from file & store to specified variable
  - ↳ syntax: fscanf (file ptr, "control variable", &variable)

- fprintf
  - ↳ stores output/prints to file using given data or variable
  - ↳ syntax: fprintf (file ptr, "control variable, read variable);

In all the above functions, we make use of a file pointer. It indicates current position of text cursor & we can read/write from that position. It can be declared & initialized as follows:

FILE *fp = fopen("filename", "mode");

Here, mode dictates what operations we can perform. Some common ones are:-

- .r ⇒ readonly. File needs to exist beforehand
- .w ⇒ write-only. File is created if it doesn't exist. data is overwritten when we perform write operation.
- .a ⇒ append. Adds to pre-existing data.
- .r+, w+ ⇒ both read/write operation.

* After opening a file, we need to close it like so:
  fclose(fp);

Another useful function is rewind(), which we can use to reset file pointer to starting position.

PROGRAMS

① WAP to write "welcome to BCA program" to file BCA.txt

```
#include <stdio.h>
void main()
{
    FILE *fp=fopen("BCA.txt","w");
    fputs("welcome to BCA program",fp);
    fclose(fp);
}
```

② WAP to take a string input from user. write it to a file, then read & display in console.

```
#include <stdio.h>
void main()
{
    FILE *fp=fopen("string.txt","w+");
    char s[20], fs[20];
    printf("Enter string");
    gets(s);
    fputs(s,fp);
    rewind(fp);
    fgets(fs,fp); /*fs is separate variable for storing string from file*/
    puts(fs);
    fclose(fp);
}
```

③ WAP to input student name, roll no. & write to file. Read & display in console. Use formatted file handling.

```
#include<stdio.h>
void main()
{
    FILE *fp=fopen("student.txt","wt");
    int r,fr;
    char n[20],fn[20];
    printf("Enter name & roll no.");
    scanf("%[^\n] %d",n,&r);
    fprintf(fp,"%s \n %d",n,r);
    rewind(fp);
    fscanf(fp,"%[^\n] %d",fn,&fr);
    printf("Name: %s \n Roll no: %d",fn,fr);
    fclose(fp);
}
```