Experiment No.3

Arrays, Strings, Functions, System Calls

1. Arrays in C

Arrays are collection of similar elements placed at contiguous memory locations.

Arrays are beneficial if you have to store many similar elements. For example if you have to store scores of 100 students either you can create 100 variables to store them or create an array to store 100 elements. It would be much easier to access and manage individual scores in case of array.

Advantages of using array

We would need very less code to access the data. We can either use a loop to traverse through the array or access any element randomly.

Disadvantage of using array

An array has a fixed size. Whatever size we define during declaration that remains fixed for the array and we can't exceed that.

Declaration of an array

Syntax

```
data_type array_name[array_size];
```

Example

```
int marks[5];
```

Here marks is an integer array of size 5.

Initialization of an Array

After declaration if we don't initialize an array then its elements can have any unknown garbage values. To prevent uncertainty its better to initialize an array.

Example 1

C program to declare initialize and print elements of an array

```
#include<stdio.h>
int main(){
      int i=0;
      int mark[5];//declaration of array
```

```
      mark[0]=80;//initialization of array
      mark[1]=60;
      mark[2]=70;
      mark[3]=85;
      mark[4]=75;

//traversal of array
      for(i=0;i<5;i++){
      printf("%d \n",mark[i]);
      }
return 0;
}
```

Example 2

C program to merge initialization and declaration and print elements of an array

```
#include<stdio.h>
int main(){
      int i=0;
      int mark[5]={20,30,40,50,60};    //declaration and
initialization of array
         //int mark[]={20,30,40,50,60};
      /*Since elements are being supplied
        there is no need to define size.Comment out the
        previous declaration and uncomment the above one to verify
*/

      //traversal of array
      for(i=0;i<5;i++){
      printf("%d \n",mark[i]);
```

```
    }
return 0;
}
```

Example 3

C program to update 2nd element of an array by obtaining a user input for the same

```
#include<stdio.h>
int main(){

    int mark[]={20,30,40,50,60};     //declaration and
initialization of array

    printf("Supply new integer second element:");
    scanf("%d", &mark[2]);

    printf("Updated second element %d \n",mark[2]);

return 0;
}
```

Example 4

C program to find average of n numbers (n<10) using arrays

```
#include <stdio.h>
int main()
{
    int marks[10], i, n, sum = 0, average;
    printf("Enter n: ");
    scanf("%d", &n);
    for(i=0; i<n; ++i)
    {
```

```
            printf("Enter number%d: ",i+1);
            scanf("%d", &marks[i]);
            sum += marks[i];
        }
        average = sum/n;

        printf("Average = %d", average);

        return 0;
}
```

Few things to note about arrays

1. Arrays have 0 as their first index and not 1 so first element is at `marks[0]` for above examples.

2. For an array of size n the last element is at index n-1

Two Dimensional Array in C

Two dimensional arrays in C is represented in form of rows and columns sometimes known as matrix. It is also called Array of arrays or list of arrays.

Declaration of two dimensional Array in C

Syntax

```
data_type array_name[size1][size2];
```

For example

```
int arr[4][3];
```

Here `arr` is an array of 3 element integer array of size 4. Or equivalently `arr` is a matrix of 4 rows and 3 columns.

Example 5

C program to declare a 2D array and print its elements

```
#include<stdio.h>
int main(){
      int i=0,j=0;
      int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
      //int arr[4][3]={1,2,3,2,3,4,3,4,5,4,5,6}; //even this would
work compiler would deduce the 2d array


      //traversing 2D array
      for(i=0;i<4;i++){
            for(j=0;j<3;j++){
                  printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);
                  }//end of j
            }//end of i
return 0;
}
```

Example 6

C program to find sum of two 2X2 order matrices using 2D array

```
#include <stdio.h>
int main()
{
   float a[2][2], b[2][2], c[2][2];
   int i, j;

   // Taking input using nested for loop
   printf("Enter elements of 1st matrix\n");
   for(i=0; i<2; ++i)
    for(j=0; j<2; ++j)
    {
       printf("Enter a%d%d: ", i+1, j+1);
```

```
      scanf("%f", &a[i][j]);
   }

// Taking input using nested for loop
 printf("Enter elements of 2nd matrix\n");
 for(i=0; i<2; ++i)
  for(j=0; j<2; ++j)
  {
      printf("Enter b%d%d: ", i+1, j+1);
      scanf("%f", &b[i][j]);
  }

 // adding corresponding elements of two arrays
 for(i=0; i<2; ++i)
  for(j=0; j<2; ++j)
  {
      c[i][j] = a[i][j] + b[i][j];
  }

 // Displaying the sum
 printf("\nSum Of Matrix:\n");

 for(i=0; i<2; ++i)
  for(j=0; j<2; ++j)
  {
      printf("%f\t", c[i][j]);

      if(j==1)
         printf("\n");
  }
```

```
return 0;
}
```

2. Strings in C

String in C is an array of characters that is terminated by a '\0' null character. There is no string type in C. We have to always use character arrays.

Declaration of Strings

Syntax

```
char ch[6]={'O','S','L','A','B', '\0'}; //character array

char ch[]={'O','S','L','A','B', '\0'}; //character array

char ch[]="OSLAB" //String literal //Compiler appends with '\0'
```

Example 7

```
#include<stdio.h>
#include <string.h>
int main(){
          char ch1[6]={'O','S','L','A','B', '\0'};
          char ch2[5]="OSLAB" ;
          printf("Char Array Value is: %s\n", ch1);
          printf("String Literal Value is: %s\n", ch2);

          //Printing String by acceessing charater Array
          int i = 0;
               while(ch2[i]!='\0'){
                    printf("\%c",ch2[i]);
                    i++;
               }
```

```
                //checking if compiler automatically added null
                if(ch2[5]=='\0')
                printf("\nyes compiler added null%c", ch1[5]);


return 0;
}
```

`gets()` and `puts()` funtions

The `gets()` reads the string from user and `puts()` prints the string. Both these functions are defined in `stdio.h` header file.

Example 8

```
#include<stdio.h>
int main(){
char name[50];
    printf("Enter your name: ");
    gets(name); //reads string from user
    printf("Your name is: ");
    puts(name);  //displays string
 return 0;
}
```

C String Functions

There are many functions that are uses to do various manipulations with string. These functions are defined in `string.h` header file

Below is a list of string manipulation functions that could be useful. Try to check their parameters and return types and values to be able to use them efficiently

Function	Use
strlen	Finds length of a string
strlwr	Converts a string to lowercase
strupr	Converts a string to uppercase
strcat	Appends one string at the end of another
strncat	Appends first n characters of a string at the end of another
strcpy	Copies a string into another
strncpy	Copies first n characters of one string into another
strcmp	Compares two strings
strncmp	Compares first n characters of two strings
strcmpi	Compares two strings without regard to case ("i" denotes that this function ignores case)
stricmp	Compares two strings without regard to case (identical to strcmpi)
strnicmp	Compares first n characters of two strings without regard to case
strdup	Duplicates a string
strchr	Finds first occurrence of a given character in a string
strrchr	Finds last occurrence of a given character in a string
strstr	Finds first occurrence of a given string in another string
strset	Sets all characters of string to a given character
strnset	Sets first n characters of a string to a given character
strrev	Reverses string

Example 9

C program to show utility of `strcpy, strcat` and `strlen` functions

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

int main () {

   char str1[12] = "Hello";
```

```
   char str2[12] = "World";
   char str3[12];
   int  len ;

   /* copy str1 into str3 */
   strcpy(str3, str1);
   printf("strcpy( str3, str1) :  %s\n", str3 );

   /* concatenates str1 and str2 */
   strcat( str1, str2);
   printf("strcat( str1, str2):   %s\n", str1 );

   /* total lenghth of str1 after concatenation */
   len = strlen(str1);
   printf("strlen(str1) :  %d\n", len );

   return 0;
}
```

2 Dimensional array of characters

Example 10

C program to check names against a list and if name matches the list entry will be given into palace.

```
#include<stdio.h>
#define FOUND 1
#define NOTFOUND 0
int main( )
{
      char masterlist[6][10] = {
                              "akshay",
```

```
                        "parag",
                        "raman",
                        "srinivas",
                        "gopal",
                        "rajesh"
                    } ;
    int i, flag, a ;
    char yourname[10] ;

    printf ( "\nEnter your name " ) ;
    scanf ( "%s", yourname ) ;

    flag = NOTFOUND ;
    for ( i = 0 ; i <= 5 ; i++ )
    {
        a = strcmp ( &masterlist[i][0], yourname ) ;
        if ( a == 0 )
        {
            printf ( "Welcome, you can enter the palace" ) ;
            flag = FOUND ;
            break ;
        }
    }
    if ( flag == NOTFOUND )
    printf ( "Sorry, you are a trespasser" ) ;


return 0;
}
```

3. Functions in C

A function in C is self-contained block of statements that performs a coherent task of some kind. It provides modularity and code reusability. We can define the task that has to be performed many times in a program in a function and instead of writing code for doing that task again and again we can simply call the function as many times as required. So it prevents us from writing redundant code.

Types of functions in C

1. Library Functions: These are the functions declared in C header files like scanf(), printf() etc

2. User defined functions: These are the functions created by the programmer so that he can call them many times. It reduces the complexity of the code.

Declaration of a function

Syntax

```
return_type function_name(data_type parameter1,data_type parameter2,.....,data_type parametern){

//code to be executed

}
```

Return value

A C function may or may not return a value from a function. If a function doesn't return any value the retun_type is void

Input parameters

A function can have zero or many input parameters

Example 11

C program to print hello world using function

```
#include<stdio.h>
int main(){

      function1();//calling a function

 return 0;
}

void function1()   // Function definition
{
      printf("Hello World");

}
```

Example 12

C program to find cube of a number using function

```
#include<stdio.h>

//defining function
int cube(int n){
return n*n*n;
}

int main(){
           int result1=0,result2=0;

           result1=cube(2);//calling function
           result2=cube(3);
           printf("%d \n",result1);
```

```
            printf("%d \n",result2);
return 0;
}
```

Example 13

C program to add 2 numbers using functions

```
#include<stdio.h>

//defining function
float add(float a, float b){
return a+b;
}

int main(){
            float result;

            result=add(2,4.5);//calling function


            printf("%f \n",result);

return 0;
}
```

4. Recursion In C

When a function is called within the same function, it is called recursion.The function which calls itself is known as recursive function.

Example 14

C program to find factorial of a number using recursion.

```
#include<stdio.h>
int factorial (int n)
{
    if ( n < 0){
       printf("enter +ve integer starting 0");
       return -1; /*Wrong value*/
    }
    if (n == 0)
        return 1; /*Terminating condition*/
    return (n * factorial (n-1));
}

int main(){
     int fact=0;
     int n = 0;
     printf("enter +ve integer starting 0 to find factorial:");
     scanf("%d",&n);
     fact=factorial(n);
     printf("\n factorial of %d is %d",n,fact);
return 0;
}
```

5. System call

A system call provides an interface to the services made available by operating system. It is programmatic way by which a computer program requests a service from the kernel of the operating system.

We would today be discussing about fork(), getpid() and getppid() system calls. There are many system calls but they are out of scope for the current experiment. We will discuss other system calls after pointers.

`fork()` system call

Synopsis

```
#include <unistd.h>


pid_t fork(void);
```

`fork()` is used to create a new process which becomes child process of the caller. After a new child process is created, both processes will execute the next instruction following the fork() system call. Therefore, we have to distinguish the parent from the child. This can be done by testing the returned value of `fork()`.

1. If `fork()` returns a negative value, the creation of a child process was unsuccessful.
2. `fork()` returns a zero to the newly created child process.
3. `fork()` returns a positive value, the process ID of the child process, to the parent. The returned process ID is of type pid_t defined in sys/types.h. Normally, the process ID is an integer.

`getpid()` and `getppid()` system call

Synopsis

```
#include <sys/types.h>
#include <unistd.h>


pid_t getpid(void);
pid_t getppid(void);
```

`getpid()` returns the process ID of the calling process. The returned process ID is of type `pid_t` defined in `sys/types.h`. Normally, the process ID is an integer.

`getppid()` returns the process ID of the parent of the calling process.The returned process ID is of type `pid_t` defined in `sys/types.h`. Normally, the process ID is an integer.

Example 15

C program to show the usage of `fork()`, `getpid()`, `getppid()` syscall

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
int main() {
  pid_t  pid;
  pid_t  ppid;
  // make two process which run same
  // program after this instruction
  fork();
  fork();
  fork();

  pid = getpid();
  ppid = getppid();

  printf("\nHi this is Process with pid:%d and ppid:
  %d\n",pid,ppid);
  return 0;

}
```

Note : In above program you would observe that parent process id may not seem correct. What's happening is that the parent is terminating before the child runs. This leaves the child as an orphan and it gets adopted by the init process. If you are logged in as root

user then PID of init process is 1 otherwise if you are logged in as any user then PID of the init process of the user may be different.

To match the PID of the init process of the user you are logged in as to the ppid of most of the processes use `ps -ef` command and check for the process yourself.

If you put a delay or read data from stdin rather than letting the parent terminate you'll see the result you have expected.

Example 16

C program to show the usage of `fork(),getpid(),getppid()` syscall with delay timer

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
int main() {
      pid_t  pid;
 pid_t  ppid;
 // make two process which run same
 // program after this instruction
 fork();
 fork();
 fork();

 pid = getpid();
 ppid = getppid();
 printf("\nHi this is Process with pid:%d and ppid:
%d",pid,ppid);
 sleep(10);
 //sleep(5) //use this if it takes too long to finish
 return 0;
}
```

Example 17

C program to show the usage of `fork()`, `getpid()`, `getppid()` syscall

```
#include <stdio.h>
#include <sys/types.h>
#include<unistd.h>

int main(){
    int id ;
    id = fork() ;
    printf("id value : %d\n",id);
    if ( id == 0 )
    {
        printf ( "Child : Hello I am the child process\n");
        printf ( "Child : Child's PID: %d\n", getpid());
        printf ( "Child : Parent's PID: %d\n", getppid());
    }
    else
    {
        printf ( "Parent : Hello I am the parent process\n" ) ;
        printf ( "Parent : Parent's PID: %d\n", getpid());
        printf ( "Parent : Child's PID: %d\n", id);
    }

return 0;
}
```

Exercise

Q1. Write a C program to do the following.
It accepts a sequence of positive integers between M and N both inclusive from the keyboard. Value of M must be less than the value of N. The program will stop accepting input once an integer outside the range is entered. The program will finish by printing the total number multiples of 3 and total number of even integers entered.

Q2. WAP in C to take n number of integers from the user and store them into an array. Take another integer (x) from the user and check whether the value (x) is present in the array.

Q3. WAP in C to check whether a given string is Palindrome.

Q4. WAP in C to compute GCD of two given integers using recursion.