Unit #: 2

Unit Name: Counting, Sorting and Searching

Topic: Arrays and function

Course objectives:

The objective(s) of this course is to make students

CObj1: Acquire knowledge on how to solve relevant and logical problems using computing machine

CObj2: Map algorithmic solutions to relevant features of C programming language constructs

CObj3: Gain knowledge about C constructs and it's associated eco-system

CObj4: Appreciate and gain knowledge about the issues with C Standards and it's respective behaviors

CObj5: Get insights about testing and debugging C Programs

Course outcomes:

At the end of the course, the student will be able to

CO1: Understand and apply algorithmic solutions to counting problems using appropriate C Constructs

CO2: Understand, analyse and apply text processing and string manipulation methods using C Arrays, Pointers and functions

CO3: Understand prioritized scheduling and implement the same using C structures

CO4: Understand and apply sorting techniques using advanced C contructs

CO5: Understand and evaluate portable programming techniques using preprocessor directives and conditional compilation of C Programs

Team – PSWC,

Jan - May, 2022

Dept. of CSE,

PES University

Array and functions

When array is passed as an argument to a function, arguments are copied to parameters of the function and parameters are always pointers. **Array degenerates to a pointer at runtime**. All the operations that are valid for pointer will be applicable for array too in the body of the function. **Function call happens always at run time.**

Coding Example_1: Using functions read n elements into an array and display it. Also include a function to find the sum of all the elements in the array

Version 1: n is a global variable

```
#include<stdio.h>
void read_array(int[]);
int n;  // global variable
void display_array(int[]);
int find_sum(int[]);
int main()
{
        int a[100];
        printf("how many elements u want to add\n");
        scanf("%d",&n);
        printf("enter %d elements\n",n);
        read_array(a);
        printf("entered elements are\n");
        display_array(a);
        printf("\nsum is %d\n",find_sum(a));
        return 0;
}
void read_array(int a[])
{
        for(int i= 0; i<n;i++)
        {
                scanf("%d",&a[i]);
        }
}
void display_array(int a[])
{
        for(int i= 0; i<n;i++)
        {
                printf("%d\t",a[i]);
        }
}
int find_sum(int a[])
{
        int sum = 0;
        for(int i= 0; i<n;i++)
        {
                sum = sum+a[i];
        }
        return sum;
}
```

Version 2: n is local to a main function. Can other functions access n then? It throws Compile time Error

Version 3: As the array becomes pointer at runtime, finding the size of the passed argument to any function is same as finding the size of the pointer.

```
#include<stdio.h>
void read_array(int[]);
int n;  // global variable
void display_array(int[]);
int find_sum(int[]);
int main()
{
        int a[100];
        printf("how many elements u want to add\n");
        scanf("%d",&n);
        printf("sizeof a is %d\n",sizeof(a));     // in my system, 400 bytes
        printf("enter %d elements\n",n);
        read_array(a);
        …
        return 0;
}
void read_array(int a[])
{
        printf("inside read_array sizeof a is %d\n",sizeof(a));
                                            //4 bytes or constant value for any pointer
        for(int i= 0; i<n;i++)
        {
                scanf("%d",&a[i]);
        }
}
```

Version 4: As n is local to main function, send this to functions as an argument

```
#include<stdio.h>
void read_array(int[],int);
void display_array(int[],int);
int find_sum(int[],int);
void increment(int a[],int n);
int main()
{
        int a[100];
        int n;
        printf("how many elements u want to add\n");
        scanf("%d",&n);
        printf("enter %d elements\n",n);
        printf("sizeof a is %d\n",sizeof(a));
        read_array(a,n);
        printf("entered elements are\n");
        display_array(a,n);
        printf("\nsum is %d\n",find_sum(a,n));
        increment(a,n);
        printf("array with updated elements are\n");
        display_array(a,n);
        return 0;
}
void read_array(int a[],int n)
{
        for(int i= 0; i<n;i++)
        {
                scanf("%d",&a[i]);
        }
}
void display_array(int a[],int n)
{
        for(int i= 0; i<n;i++)
        {
                printf("%d\t",a[i]);
        }
}
int find_sum(int a[],int n)
{
        int sum = 0;
        for(int i= 0; i<n;i++)
        {
                sum = sum+a[i];
        }
        return sum;
}
```

Version 5: Using the pointer in the parameter makes more sense as array become pointer at runtime during function call.

```
#include<stdio.h>
void read_array(int*,int);
void display_array(int*,int);
int find_sum(int*,int);  // observe this
int main()
{       …       }
void read_array(int *a,int n) // this too
{       …       }
void display_array(int *a,int n)
{       …       }
int find_sum(int *a,int n)
{       …       }
```

Version 6: The display and find_sum functions should not be allowed to make any changes to the array sent in the argument. But it is possible now.

```
#include<stdio.h>
void read_array(int*,int);
void display_array(int*,int);
int find_sum(int*,int);
int main()
{       …       }
void read_array(int *a,int n)
{       …      }

void display_array(int *a,int n)
```

```
{       a[4] = 8989;       // allowed     }
int find_sum(int *a,int n)
{       …       }
```