Experiment No.: 1

Title: Demonstrate the use of arrays, array of structure and pointers using C.

Batch: B1 **Roll Nos.:** ***16010420133*** **and** ***16010420141*** **Experiment No.: 1**
Names: ***Soumen Samanta*** **and** ***Omkar Karbhari***

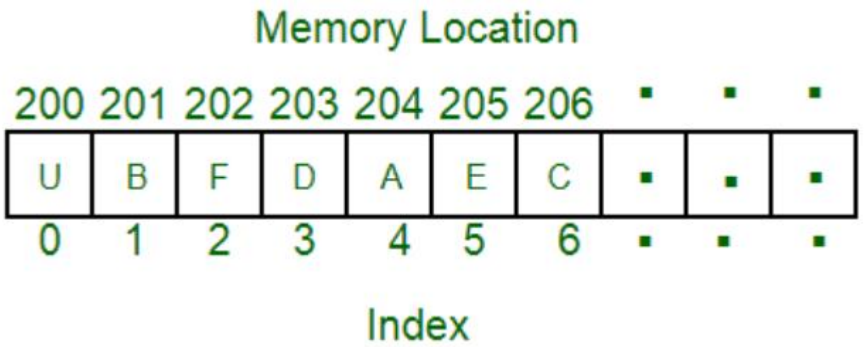
Aim: Implement and Demonstrate the use of arrays, array of structure and pointers using C.

Resources needed: Turbo C/C++ editor and C compiler (Online/Offline)

Theory

1) Arrays

An array is a collection of items stored at contiguous memory locations. The idea is to store multiple items of the same type together. This makes it easier to calculate the position of each element by simply adding an offset to a base value, i.e., the memory location of the first element of the array (generally denoted by the name of the array).



The above image can be looked as a top-level view of a staircase where you are at the base of the staircase. Each element can be uniquely identified by their index in the array (in a similar way as you could identify your friends by the step on which they were on in the above example).

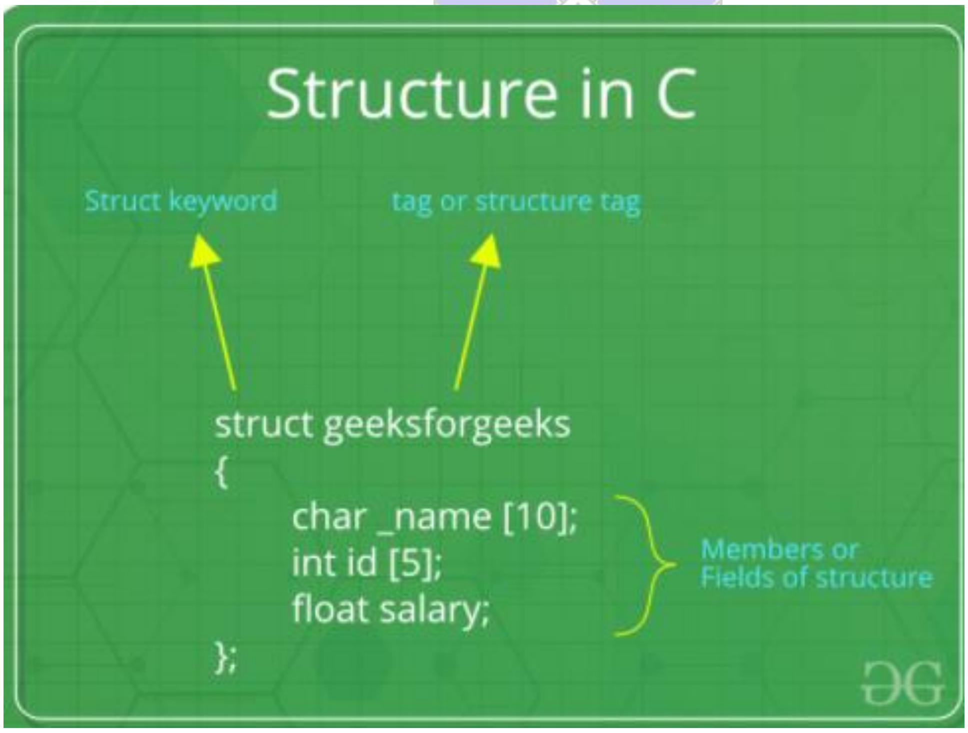
Examples

```
// Program to take 5 values from the user and store them in an array// Print
the elements stored in the array
```

```
#include <stdio.h>
int main() {
 int values[5];
 printf("Enter 5 integers: ");
 // taking input and storing it in an array
 for(int i = 0; i < 5; ++i) {     scanf("%d", &values[i]);  }
printf("Displaying integers: ");
 // printing elements of an array  for(int i = 0; i < 5; ++i) {
printf("%d\n", values[i]);  }  return 0;}
```

2) Structures

A structure is a user defined data type in C/C++. A structure creates a data type that can be used to group items of possibly different types into a single type. 'struct' keyword is used to create a structure.



```
struct geeksforgeeks
{
        char _name [10];
        int id [5];
        float salary;
};
```

A structure variable can either be declared with structure declaration or as a separate declaration like basic types.

Examples

```
// Program to add two distances (feet-inch)
#include <stdio.h>
struct Distance
```

```
{
        int feet;
        float inch;
} dist1, dist2, sum;

int main()
 {
      printf("1st distance\n");
      printf("Enter feet: ");
      scanf("%d", &dist1.feet);

      printf("Enter inch: ");
      scanf("%f", &dist1.inch);
      printf("2nd distance\n");

      printf("Enter feet: ");
      scanf("%d", &dist2.feet);

      printf("Enter inch: ");
      scanf("%f", &dist2.inch);

      // adding feet
      sum.feet = dist1.feet + dist2.feet;
      // adding inches
      sum.inch = dist1.inch + dist2.inch;

      // changing to feet if inch is greater than 12
      while (sum.inch >= 12)
      {
          ++sum.feet;
          sum.inch = sum.inch - 12;
      }

      printf("Sum of distances = %d\'-%.1f\"", sum.feet, sum.inch);
      return 0;
}
```

3) Array of Structure

An array of structures is simply an array in which each element is a structure of the same type. The referencing and subscripting of these arrays (also called structure arrays) follow the same rules as simple arrays.

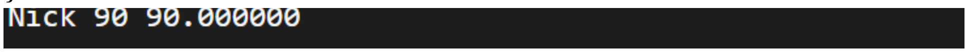
Creating an Array of Structures

The easiest way to create an array of structures is to use the REPLICATE function. The first parameter to REPLICATE is a reference to the structure of each element.

Examples

#include<stdio.h>

```
struct student
{
   char name[20];
   int id;
   float marks;
};
void main()
 {
   struct student s1,s2,s3;
   int dummy;
   printf("Enter the name, id, and marks of student 1 ");
   scanf("%s %d %f",s1.name,&s1.id,&s1.marks);
   scanf("%c",&dummy);
   printf("Enter the name, id, and marks of student 2 ");
   scanf("%s %d %f",s2.name,&s2.id,&s2.marks);
   scanf("%c",&dummy);
   printf("Enter the name, id, and marks of student 3 ");
   scanf("%s %d %f",s3.name,&s3.id,&s3.marks);
   scanf("%c",&dummy);
   printf("Printing the details....\n");
   printf("%s %d %f\n",s1.name,s1.id,s1.marks);
   printf("%s %d %f\n",s2.name,s2.id,s2.marks);
   printf("%s %d %f\n",s3.name,s3.id,s3.marks);
}
```



```
Nick 90 90.000000
```

4) Pointers and Pointers to Structures

Pointers

The pointer in C language is a variable which stores the address of another variable. This variable can be of type int, char, array, function, or any other pointer. The size of the pointer depends on the architecture. However, in 32-bit architecture the size of a pointer is 2 byte.
Consider the following example to define a pointer which stores the address of an integer.

```
int n = 10;
int* p = &n; // Variable p of type pointer is pointing to the address of the variable n of type integer.
```

The pointer in c language can be declared using * (asterisk symbol). It is also known as indirection pointer used to dereference a pointer.

```
int *a;//pointer to int
char *c;//pointer to char
```

Pointer structure

Pointer to structure holds the add of the entire structure.

It is used to create complex data structures such as linked lists, trees, graphs and so on.

The members of the structure can be accessed using a special operator called as an arrow operator (->).

Declaration

Following is the declaration for pointers to structures in C programming −

```
struct tagname *ptr;
```

For example − struct student *s −

Accessing

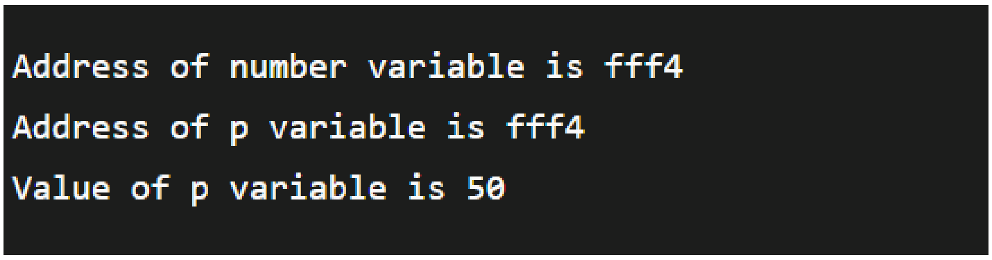
It is explained below how to access the pointers to structures.

```
Ptr-> membername;
```

For example − s->sno, s->sname, s->marks;

Examples

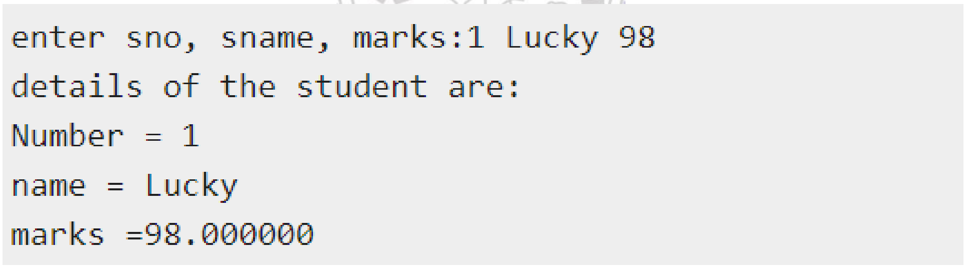
```
//Pointer Example
#include<stdio.h>
int main(){
int number=50;
int *p;
p=&number;
printf("Address of p variable is %x \n",p);
printf("Value of p variable is %d \n",*p);
return 0;
}
```



```
Address of number variable is fff4
Address of p variable is fff4
Value of p variable is 50
```

//The following program shows the usage of pointers to structures

```
#include<stdio.h>
struct                                                  student{
   int                                                      sno;
   char                                                sname[30];
   float                                                  marks;
};
main                        (                                ){
   struct                     student                         s;
   struct                    student                        *st;
   printf("enter        sno,        sname,        marks:");
   scanf    ("%d%s%f",    &    s.sno,    s.sname,    &s.    marks);
   st                          =                             &s;
   printf      ("details      of      the      student      are");
   printf      ("Number      =      %d\n",      st      ->sno);
   printf        ("name        =        %s\n",        st->sname);
   printf        ("marks        =%f\n",        st        ->marks);
   getch                        (                                );
}
```



```
enter sno, sname, marks:1 Lucky 98
details of the student are:
Number = 1
name = Lucky
marks =98.000000
```

5) Functions and Function signature

Function signatures are the "declaration" of the functions in a program. Declaration of a function instructs a compiler on how to call a function. Function declarations comprise of the following:

Name of the function

Return type : type of the value that will be returned to the program when function is executed

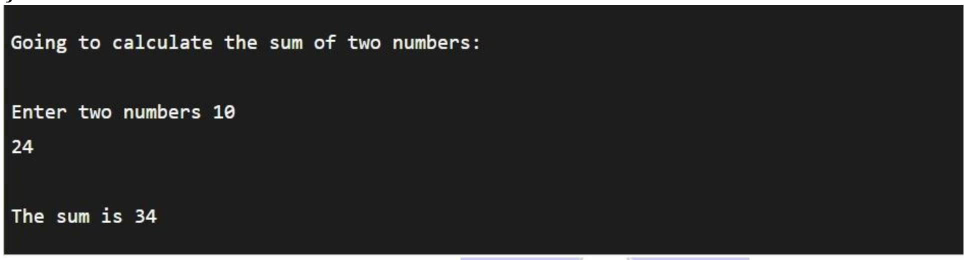
Parameter(s) : the type of values that you pass to the function while calling it

In c, we can divide a large program into the basic building blocks known as function. The function contains the set of programming statements enclosed by {}. A function can be called multiple times to provide reusability and modularity to the C program. In other words, we can say that the collection of functions creates a program. The function is also known as procedure or subroutine in other programming languages.

Examples

___________#include<stdio.h>

```
void sum();
void main()
{
   printf("\nGoing to calculate the sum of two numbers:");
   sum();
}
void sum()
{
   int a,b;
   printf("\nEnter two numbers");
   scanf("%d %d",&a,&b);
   printf("The sum is %d",a+b);
}
```



```
Going to calculate the sum of two numbers:


Enter two numbers 10
24


The sum is 34
```

Activity :

Program should demonstrate the use of concepts of arrays, pointers, structures, array of structure, pointers to structure. Students are required to choose a proper example and show the use of above concept in the implementation of the example. Consider implementing a modular programming technique by making use of user defined functions.

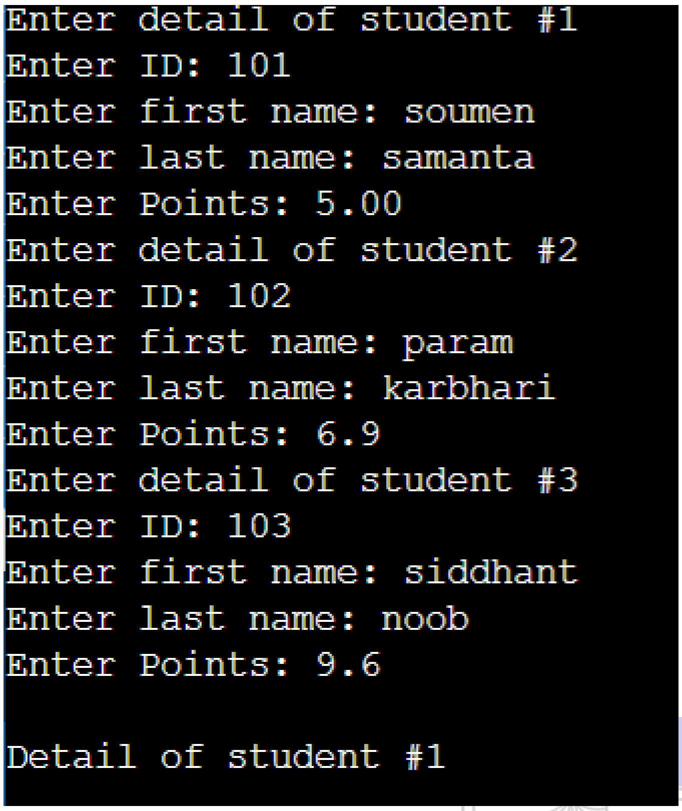
```
#include <stdio.h>

int main(void) {

   // student structure

  struct student {

    char id[15];

    char firstname[64];

    char lastname[64];

    float points;

  };

  // student structure variable

  struct student std[3];

  // student structure pointer variable
```

```
struct student *ptr = NULL;

// other variables

int i;

// assign std to ptr

ptr = std;


// get detail for user

for (i = 0; i < 3; i++) {

  printf("Enter detail of student #%d\n", (i + 1));

  printf("Enter ID: ");

  scanf("%s", ptr->id);

  printf("Enter first name: ");

  scanf("%s", ptr->firstname);

  printf("Enter last name: ");

  scanf("%s", ptr->lastname);

  printf("Enter Points: ");

  scanf("%f", &ptr->points);


  // update pointer to point at next element

  // of the array std

  ptr++;

}

// reset pointer back to the starting

// address of std array

ptr = std;

for (i = 0; i < 3; i++) {

  printf("\nDetail of student #%d\n", (i + 1));

   // display result via std variable
```

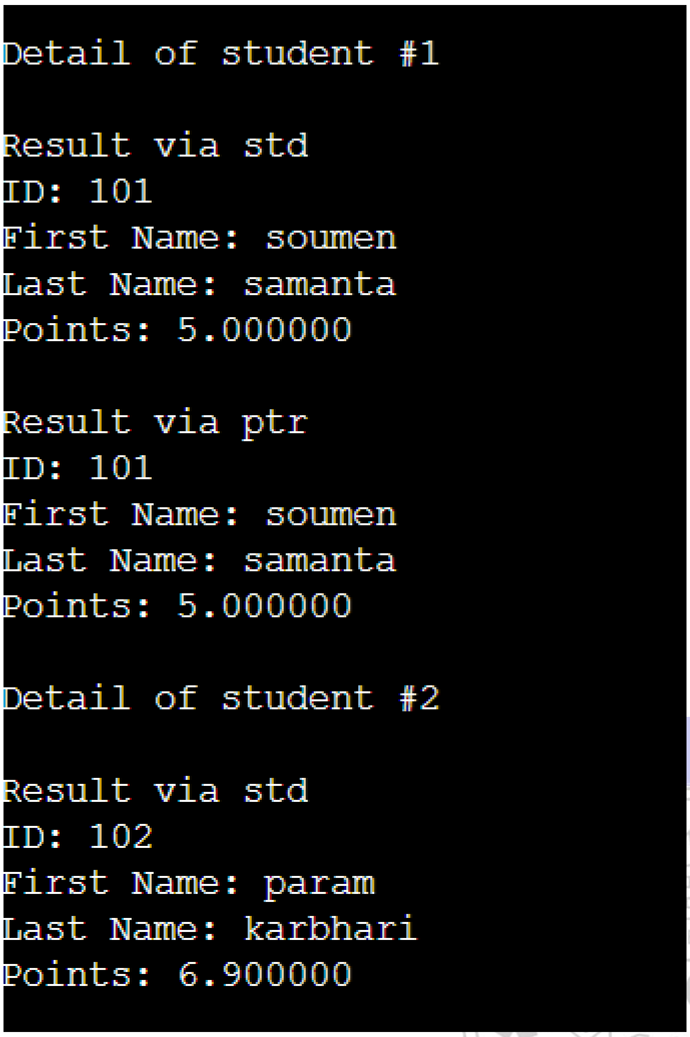
```
    printf("\nResult via std\n");

    printf("ID: %s\n", std[i].id);

    printf("First Name: %s\n", std[i].firstname);

    printf("Last Name: %s\n", std[i].lastname);

    printf("Points: %f\n", std[i].points);


    // display result via ptr variable

    printf("\nResult via ptr\n");

    printf("ID: %s\n", ptr->id);

    printf("First Name: %s\n", ptr->firstname);

    printf("Last Name: %s\n", ptr->lastname);

    printf("Points: %f\n", ptr->points);


    // update pointer to point at next element

    // of the array std

    ptr++;

  }


  return 0;

}
```

Results: A C program depicting the correct behaviour of mentioned concept and capable of handling all possible exceptional conditions/inputs and the same is reflecting clearly in the output.

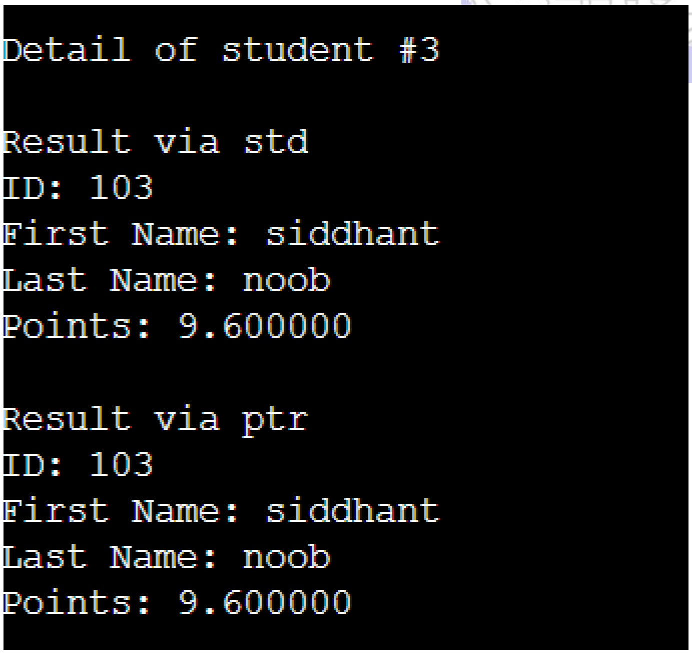
Outcomes:



```
Enter detail of student #1
Enter ID: 101
Enter first name: soumen
Enter last name: samanta
Enter Points: 5.00
Enter detail of student #2
Enter ID: 102
Enter first name: param
Enter last name: karbhari
Enter Points: 6.9
Enter detail of student #3
Enter ID: 103
Enter first name: siddhant
Enter last name: noob
Enter Points: 9.6

Detail of student #1
```



```
Detail of student #1

Result via std
ID: 101
First Name: soumen
Last Name: samanta
Points: 5.000000

Result via ptr
ID: 101
First Name: soumen
Last Name: samanta
Points: 5.000000

Detail of student #2

Result via std
ID: 102
First Name: param
Last Name: karbhari
Points: 6.900000
```



```
Detail of student #3

Result via std
ID: 103
First Name: siddhant
Last Name: noob
Points: 9.600000

Result via ptr
ID: 103
First Name: siddhant
Last Name: noob
Points: 9.600000
```

Conclusion:

Using C programming

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

- Y. Langsam, M. Augenstin and A. Tannenbaum, "**Data Structures using C**", Pearson Education Asia, 1st Edition, 2002

- **Data Structures A Psedocode Approach with C**, Richard F. Gilberg&Behrouz A. Forouzan, secondedition, CENGAGE Learning