**Experiment No. : 1**

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

**Batch: A1 Roll No.: 16010420026 Experiment No.: 1**

**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 structure that holds similar type of data in consecutive memory location. The data can be accessed by the index of that data and hence this data structure is ordered, we can even store duplicate elements in a array. The data is accessed by using the index in unit time hence it is easy for accessing the given data.

**Examples**

**#include<stdio.h>**

**int main(void)**

**{**

**int a[5] = {10,25,34,67,81};**

**printf("%d\n",a[1]);**

**printf("%d\n",a[4]);**

**char s[4] = {'d','a','t','a'};**

**for( int i=0; i<4; i++ )**

**printf("%c ",s[i]);**

**return 0;**

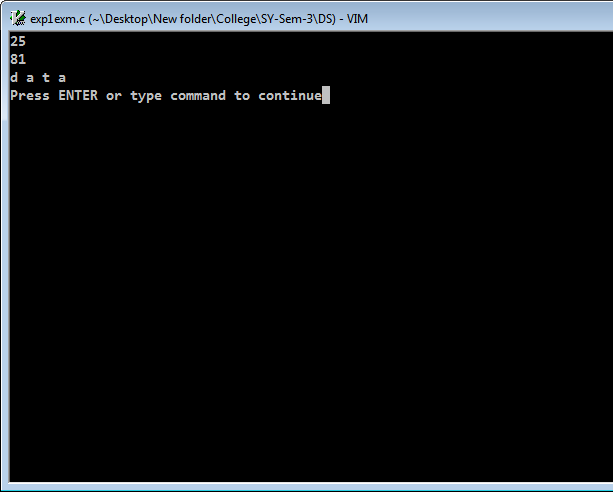
**}**

**Output:**

**25**

**81**

**d a t a**



**In the above example, we have created an array of 5 elements of type int and we were easily able to access it using its index which is zero-based indexing. We also have create an array of char type and printed the elements one by one in a for loop.**

1. **Structures**

Structures are used to create a data that can be used to group multiple types of data into a single type. They are used to create a custom data types which can hold multiple primitive or non-primitive data types in a programming language. Using structures, we can create more complex and appropriate for certain use cases data types in a program or an application.

**Examples**

**#include<stdio.h>**

**#include<stdbool.h>**

**struct blog{**

**char post[64];**

**bool is\_open\_source;**

**};**

**int main(void)**

**{**

**struct blog b1;**

**printf("Enter the post name: ");**

**scanf("%s", b1.post);**

**printf("Is the Blog Open Source? (true or false) ");**

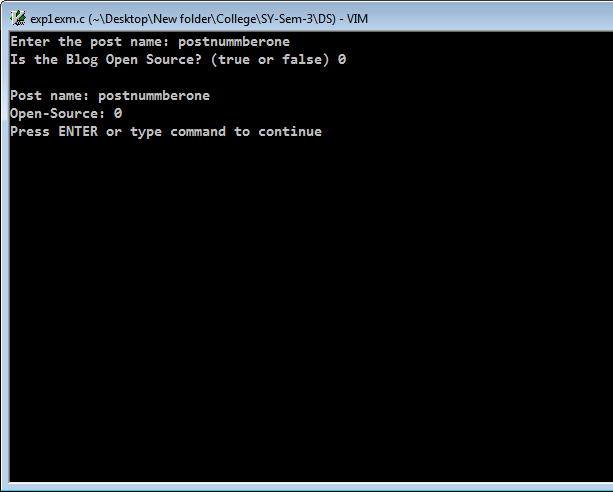
**scanf("%s",&b1.is\_open\_source);**

**printf("\nPost name: %s",b1.post);**

**printf("\nOpen-Source: %s",&b1.is\_open\_source);**

**return 0;**

**}**



From the above code example, we were able to create a simple structure and have some basic data types defined inside of it like char and bool, we can have int, float and others as well. We have contained those data in a structure named blog and have used for performing some input output operation.

1. **Array of Structure**

Array of structure is a group of a single structure. A structure is an entity that holds multiple kinds of data. Thus we can create multiple copies of a structure for holding data for multiple entries. We can access the structures using indexing and hence it becomes quite easy and efficient to access the particular data in the multiple structures of the same type.

**Examples**

#include<stdio.h>

#include<stdbool.h>

struct blog{

int id;

char post[64];

bool is\_open\_source;

};

int main(void)

{

int n=0;

printf("Enter the number of post: ");

scanf("%d",&n);

struct blog b[n];

struct blog \*p;

for(int i=0; i<n; i++ ){

b[i].id=i;

printf("Enter the post name: ");

scanf("%s",b[i].post);

int temp;

printf("Is the Blog Open Source? (0 for no and 1 for yes) ");

scanf("%d",&temp);

b[i].is\_open\_source=temp;

}

for(int i=0; i<n; i++ ){

p = &b[i];

printf("\nPost name: %s",p->post);

if(p->is\_open\_source==false)

printf("\nOpen-Source: No\n");

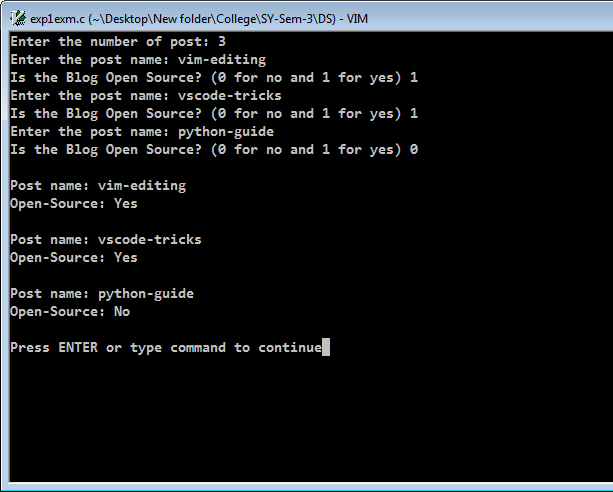
else

printf("\nOpen-Source: Yes\n");

}

return 0;

}



From the above example, we were able to create an group or array of structures and able to input multiple fields of the data, we were using pointers to refer the data stored in the array of those structures. We were able to create a list of the same structure using very minimal code.

1. **Pointers**

Pointers are just memory locations of the variables/data structures used in the program. It is basically used for efficient storing and retrieval of data as well as used for referencing the value of the variable in the memory address.

**Examples**

**#include<stdio.h>**

**int main(void)**

**{**

**int n=4;**

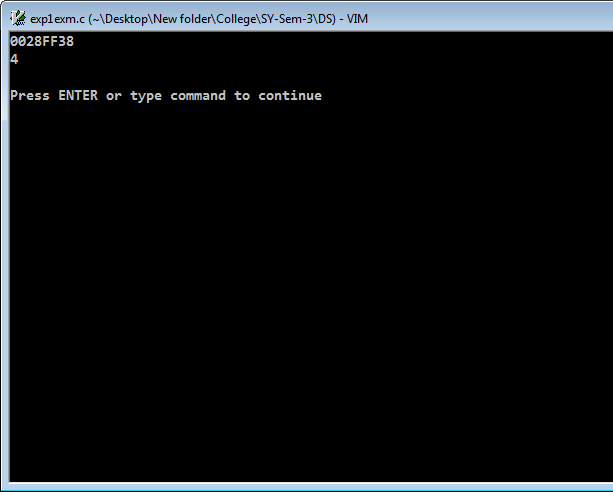
**int \*p=&n;**

**printf("%p\n",p);**

**printf("%d\n",\*p);**

**return 0;**

**}**



We are able to access the memory address of the variable n using the pointer p and also able to access the content of its memory using the \* operator which gives the value stored in the memory address of a pointer.

1. **Functions and Function signature**

Functions are used to do a particular task They are used for avoiding repetition and making code much more easy and cleaner to read as well as for scaling the code. They are a chunk of code defined in the program that can be called from within the program but by using stack memory.

Function signature defines the function i.e. the return type of the function, what parameters will it accept. The return type is the type of data that the function will return to the main program once the execution is completed and the parameters are the variables or data that we need to work with inside of the function which are taken from the main flow of the program.

**Examples**

**#include<stdio.h>**

**int exponential(int , int);**

**int main(void)**

**{**

**int base,power,ans;**

**printf("Enter the base number: ");**

**scanf("%d",&base);**

**printf("\nEnter the power: ");**

**scanf("%d",&power);**

**ans=exponential(base,power);**

**printf("\n%d ^ %d = %d\n",base,power,ans);**

**return 0;**

**}**

**int exponential(int x, int y){**

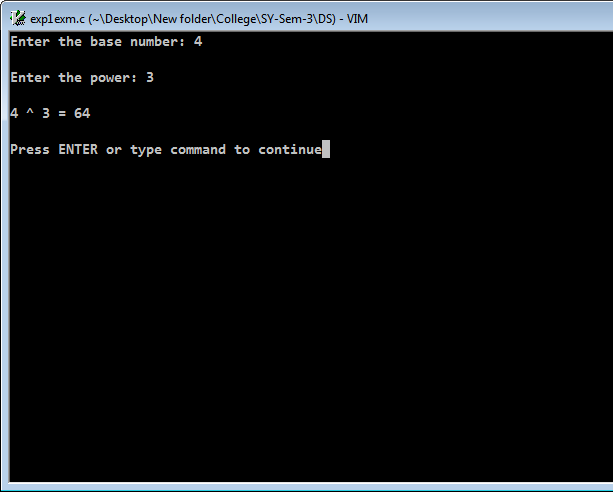
**int s=1;**

**for(int i=0; i<y; i++)**

**s\*=x;**

**return s;**

**}**



Using functions, we are able to use the code from outside of the main program or the logic and do the processing independent of the stack of the current logic in the main program. We can us the function multiple times and hence use the same piece of code multiple times. This can also be used for recursion and other concepts as well. We have first defined the function as a prototype which takes the return type (here it is int), we are passing the parameters as two integers to work with. We have assigned the variable ans by calling the function and hence storing the statement in it.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**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.

Code:

#include <stdio.h>

#include <string.h>

struct FootballTeam {

  char name[16];

  char player[11][16];

};

void printTeam(struct FootballTeam f) {

  for (int x = 0; x < 11; x++)

    printf("Player %d: %s\n", x + 1, f.player[x]);

}

// function to delete a player from a particular team

void delplayer(struct FootballTeam \*fteam, char \*p) {

  int n;

  for (int j = 0; j < 11; j++) {

    if (strcmp(p, fteam->player[j]) == 0)

      n = j;

  }

  for (int i = n; i < 11; i++) {

    strcpy(fteam->player[i], fteam->player[i + 1]);

    if (i == 10)

      strcpy(fteam->player[i], "");

  }

}

// function to add a player to a particular team

void addPlayer(struct FootballTeam \*f, char \*p) {

  int size = sizeof(f->player) / sizeof(f->player[0]);

  for (int i = size - 1; i >= 0; i--) {

    if (strcmp(f->player[i], "") != 0) {

      strcpy(f->player[i + 1], p);

      break;

    }

    if (i == 0)

      printf("Team already has 11 players\n");

  }

}

// function to replace a player from a particular team

void replacePlayer(struct FootballTeam \*f, char \*p, char \*c) {

  for (int i = 0; i < 11; i++) {

    if (strcmp(f->player[i], p) == 0)

      strcpy(f->player[i], c);

  }

}

int main(void) {

  int tnum;

  printf("Enter the number of teams: ");

  scanf("%d", &tnum);

  printf("%d", tnum);

  struct FootballTeam fteams[tnum];

  struct FootballTeam \*p;

  for (int i = 0; i < tnum; i++) {

    printf("Enter the name of the team : ");

    scanf(" %s", fteams[i].name);

    for (int j = 0; j < 11; j++) {

      printf("Enter the name of the player %d : ", j + 1);

      scanf(" %s", fteams[i].player[j]);

    }

  }

  for (int i = 0; i < tnum; i++) {

    p = &fteams[i];

    printf("The team name is %s\n", p->name);

    printf("The name of the players are :");

    for (int j = 0; j < 11; j++) {

      printf("%s ,", p->player[j]);

    }

    printf("\n");

  }

// Choice to remove/add/replace player within a team

  int choice = 0;

  while (choice != 4) {

    printf("What would you like to do?\n");

    printf("1: Remove a player from a team\n");

    printf("2: Add a player to a team\n");

    printf("3: Replace a player from a team\n");

    printf("4: Exit (do nothing)\n");

    printf("\n Enter either of 1/2/3 or 4 \n");

    scanf("%d", &choice);

    switch (choice) {

    case 1: {

      char tname[16];

      printf("Enter the exact name of the team: ");

      scanf("%s", tname);

      for (int i = 0; i < tnum; i++) {

        if (strcmp(fteams[i].name, tname) == 0) {

          char pname[16];

          printf("Enter the exact name of the player: ");

          scanf("%s", pname);

          delplayer(&fteams[i], pname);

          printTeam(fteams[i]);

        }

      }

      break;

    }

    case 2: {

      char tname[16];

      printf("Enter the exact name of the team: ");

      scanf("%s", tname);

      for (int i = 0; i < tnum; i++) {

        if (strcmp(fteams[i].name, tname) == 0) {

          char pname[16];

          printf("Enter the exact name of the player: ");

          scanf("%s", pname);

          addPlayer(&fteams[i], pname);

          printTeam(fteams[i]);

        }

        break;

      }

      break;

    }

    case 3: {

      char tname[16];

      printf("Enter the exact name of the team: ");

      scanf("%s", tname);

      for (int i = 0; i < tnum; i++) {

        if (strcmp(fteams[i].name, tname) == 0) {

          char pname[16], newply[16];

          printf("Enter the exact name of the player: ");

          scanf("%s", pname);

          printf("Enter the name of the new player: ");

          scanf("%s", newply);

          replacePlayer(&fteams[i], pname, newply);

          printTeam(fteams[i]);

        }

        break;

      }

    }

    case 4: {

      break;

    }

    default: {

      printf("Entet a valid Input choice\n");

      break;

    }

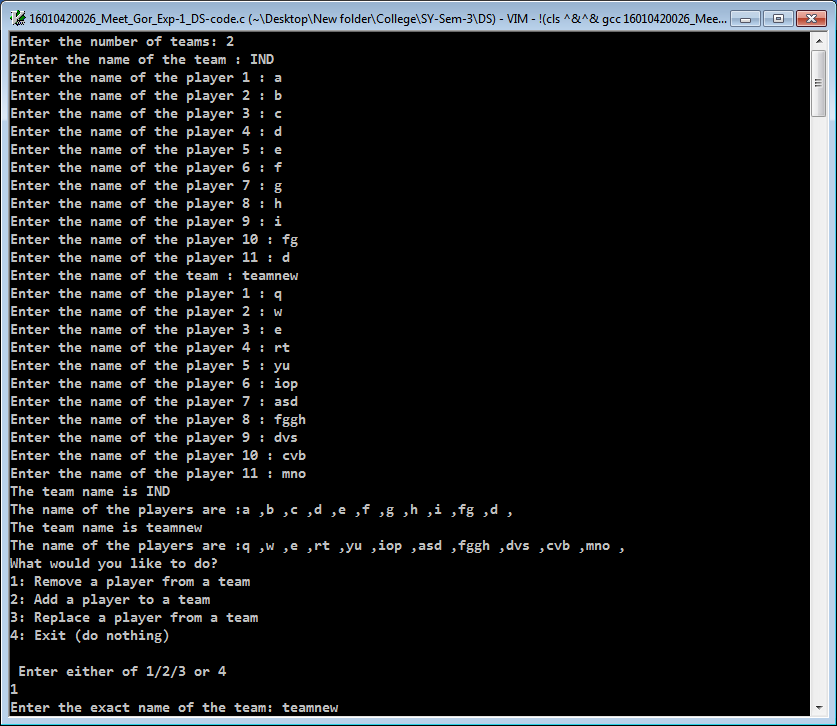
    }

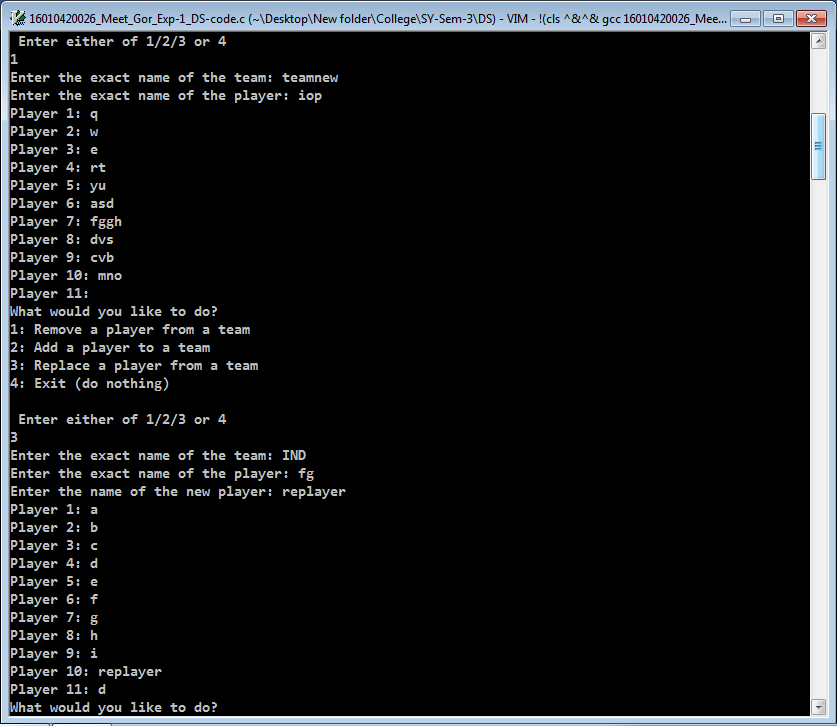
  }

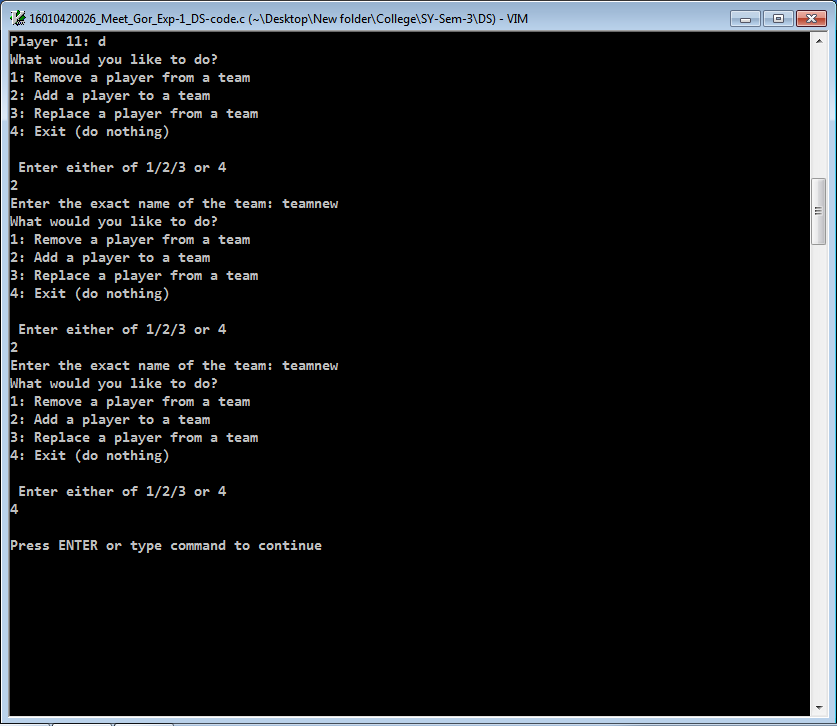
  return 0;

}

**Output Screenshot:**







**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:**

Explain the different data structures used in problem solving

**Conclusion:**

From the following example, we were able to understand and use the data structures like pointers, arrays, structures and functions in a single program to demonstrate the use cases and its functionality. We were also able to make an application program to use and apply those concepts to use in the features of it.

**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