Program to use structure along with functions and accesing data through pointers.

Pranav Tiwari  
Department of Electronics and Communication  
Indian Institute of Information Technology, AllahabadMumbai, Maharastra  
[iec2021048@iiita.ac.in](mailto:iec2021048@iiita.ac.in)

Shriyansh Lohiya  
Department of Electronics and Communication  
Indian Institute of Information Technology, AllahabadJaipur, Rajasthan  
[iec2021047@iiita.ac.in](mailto:iec2021047@iiita.ac.in)

Sreevardhan  
Department of Electronics and Communication  
Indian Institute of Information Technology, AllahabadChittoor, Andra Pradesh [iec2021049@iiita.ac.in](mailto:iec2021049@iiita.ac.in)

Mohd Kaif   
Department of Electronics and Communication  
Indian Institute of Information Technology, AllahabadAmroha, Uttar Pradesh [iec2021046@iiita.ac.in](mailto:iec2021046@iiita.ac.in)

Create a structure to specify data on students as given below: Roll number, Name, Department, Course, and Year of joining. Assume that there are not more than 450 students in the college. (a) Write a function to print the names of all students who joined in the last 3 years. (b) Write a function to print the data of a student whose roll numbers are divisible by

# Introduction

In this project since we want to store multiple variables for a given student, we would make use of data type called structure. After making a struct we initialize a dynamic array which would have this structure as its data type so thata we can store data of 450 students without making the code too lengthy or initializing too many variables. Now since this array is dynamic, we would make use of pointers in order to access different elements in this array.

# Structure

A structure is a user defined data type in C. It is used to store multiple items under a single type. For example, it can be used to store price of all food grains in a shop.

One can initialize a structure by writing the below command: -

struct struct\_name

{

//data\_types\_stored;

};

Here you can have multiple data types stored as mentioned above.

# Do while Loop

A Do while loop is a repetition control structure which allows us to write a loop which runs until the condition in the while statement becomes false. In this kind of loop even when the condition is false the loop would run for one time, where as in While loop the loop would not run at all.

The basic form of declaring a Do while loop is given below:-

do

{

//Statements you want to run in the loop.

} while(condition for the loop to repeat);

# For Loop

A for loop is used to make a loop executed for given number of times. Unlike Do while loop we can use this loop when we have fix conditions of initialization and some increments after each loop. The basic form of making a for loop is given below:-

for(initialization; condition ; increment)

{

//statement you want to execute in the loop.

}

# If Else Statements

If else statements are conditional statements which are used to run statements if they follow a certain condition mentioned by the if else statements. The basic form of writing a if else statement is given below :-

If(condition)

{

//body of the statement.

}

Else if(condition)

{

//body of the statement.

}

Else{

//body of the statement.

}

Here if none of the given conditions are true, instead of just ending the process the program would run the else statement since it doesn’t have any condition. NOTE: You can only write one else statement per if else sereies.

# Nesting Of For Loop

Sometimes we feel the need of adding another loop inside a existing loop in order to have 2 elements which changes on their own conditions. For example if one wants to input elements of 3x3 matrix, he/she cannot just go and make a loop for 9 elements, he/she would first have to make a loop for the rows followed by a loop of column such that each time a row loop is running, the values of columns are the only one changing while row index is remaining constant.

The basic form of declaring a nested loop in C is given below:-

for(initialization; condition ; increment)

{

for(initialization; condition; increment)

{

//the statements that you want to execute.

}

}

# Pointers

Pointers provide direct access to memory. Pointers allow us to perform dynamic memory allocation and deallocation. ... Pointers helps us to build complex data structures like linked list, stack, queues, trees, graphs etc. Pointers allow us to resize the dynamically allocated memory block.

The Pointer in C, is a variable that stores address of another variable. A pointer can also be used to refer to another pointer function. A pointer can be incremented/decremented, i.e., to point to the next/ previous memory location. The purpose of pointer is to save memory space and achieve faster execution time.

After declaring a pointer, we initialize it like standard variables with a variable address. If pointers in C programming are not uninitialized and used in the program, the results are unpredictable and potentially disastrous.

To get the address of a variable, we use the ampersand (&)operator, placed before the name of a variable whose address we need.

Here in this cases we have created a pointer to the structure and had passed it as an argument to the function

# Procedure

In the start of our program, first we have made a structure outside the main function so that we can access it easily in corresponding functions. In this structure we have made variables which would store various information regarding the students like roll no, name, department, course and year of joining. In order to store numbers, we used unsigned short int for roll number, since we know that roll numbers are always positive and their values would be quite small for int data type therefore used short.

Furthermore, storing department name, course name and name of the student we made an array of 25 size of char. The code for this structure is given below: -

struct student\_info

{

// here roll number cannot be negative and can be only between 0 to 450.

unsigned short int roll\_number;

// according to international standards name should not exceed 25 characters.

char name[25], department[25], course[25];

// year of join is small number so use unsigned short int.

unsigned short int year;

};

After making the structure, we have declared two functions with the argument of a struct data type pointer. So that inside the function we can use the dynamic array more freely. Code for that is given below: -

void names\_of\_prev\_3yrs\_students (struct student\_info \*);

void data\_of\_student\_whose\_roll\_is\_divisible\_by\_4(struct student\_info \*);

Now, after making the struct and declaring the function, we started the main function in which we first made use of do while loop to store the number of students the end user want to input, where according the conditions of the do while loop end user would have to enter a number x which would lie in the range of [0,450);

unsigned short int no\_of\_students;

do

{

printf("Enter the number of students studying in the college : ");

scanf("%hu", &no\_of\_students);

} while (no\_of\_students <= 0 || no\_of\_students > 450);

Now our next step is to take input of data of the ‘n’ students and we can do that by using a for loop and inputting each value of the data inside that loop. We also made use of %\*c%[^\n] to take input a whole line, since the name of a student can be multiple words and the same can be said about department and course. There also some intentional limiations established in this part of the code like the roll no of a student can be only in between [0,450) and year of joining cannot be above 2021.

The code for the step is given below: -

for (unsigned short int i = 0; i < no\_of\_students; i++)

{

do

{

printf("Enter the roll number of student %hu : ", i + 1);

scanf("%hu", &student[i].roll\_number);

} while (student[i].roll\_number <= 0 || student[i].roll\_number > 450);

//~ As per the question students are not more than 450 and students can't be 0.

printf("Enter the name of student %hu : ", i + 1);

scanf("%\*c%[^\n]", student[i].name);

//~ As to take space seperated names as input we use "%\*c%[^\n]".

printf("Enter the department of student %hu : ", i + 1);

scanf("%\*c% [^\n]", student[i].department);

//~ As to take space seperated names as input we use "%\*c%[^\n]".

printf("Enter the course of student %hu : ", i + 1);

scanf("%\*c%[^\n]", student[i].course);

//~ As to take space seperated names as input we use "%\*c%[^\n]".

do

{

printf("Enter the joining year of student %hu : ", i + 1);

scanf("%hu", &student[i].year);

} while (student[i].year > 2021);

//~ this is the maximum number of years in which a student can complete his/her b-tech.

}

After finishing the given steps we have a complete structure filled with student datas, so the only thing left is to write 2 functions, One for the names of the students who joined the college in last 3 years and the second is the data of the students whose roll numbers are divisible by 4.

For doing this we made 2 separate functions named respectively and used pointers to access the structural array which we made to store data of the students.

The first one is the names of the students joined in past 3 years, here we have taken a pointer as an argument this pointer points towards an element of the structural array. Now inside the function we use this pointer and dereference it to find the year of joining of a given student and if it is above 2019 it would just print the name of the student and return the nothing since it’s a void function.

void names\_of\_prev\_3yrs\_students(struct student\_info \*pointer)

{

if ((\*pointer).year >= 2019)

{

printf("%s\n", (\*pointer).name);

}

}

The other function is about giving data of students whose roll number are divisible by 4. For this we would again take a pointer to structure student as an argument and use later dereference it to an element of the Student so that we can find the data of that student whose roll number is divisible by 4.

The code for the following is given below:-

void data\_of\_student\_whose\_roll\_is\_divisible\_by\_4(struct student\_info \*pointer)

{

if ((\*pointer).roll\_number % 4 == 0)

{

printf("Roll number : %hu\n", (\*pointer).roll\_number);

printf("Name : %s\n", (\*pointer).name);

printf("Department : %s\n", (\*pointer).department);

printf("Course : %s\n", (\*pointer).course);

printf("Year : %hu\n\n", (\*pointer).year);

}

}

With the above two functions defined we would now use a for loop where a pointer which points at the first element of the Student structure array would also go inside and after each time the loop runs it would change the pointer and make it so that it points towards the next element of the struct student. And after doing this for the first function we would repeat it again for the second function. The code for the following is given below:-

printf("\n\nNames of the students who joined in past 3 years are given below:-\n\n");

for (unsigned short int i = 0; i < no\_of\_students; i++, ptr++)

{

names\_of\_prev\_3yrs\_students(ptr);

//~sending data in the form of pointer to reduce memory allocation.

}

ptr = student;

//~reinitializing the pointer to the structure.

printf("\n\nData of the Students whose Roll No. is divisible by 4 are given below:-\n\n");

for (unsigned short int i = 0; i < no\_of\_students; i++, ptr++)

{

data\_of\_student\_whose\_roll\_is\_divisible\_by\_4(ptr);

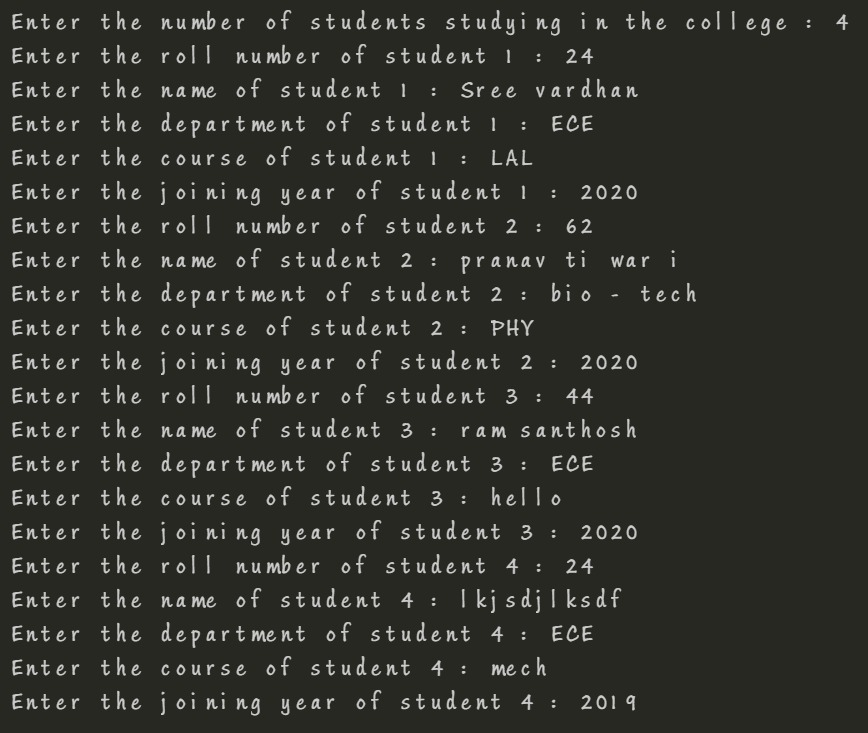
}

# TimeComplexity

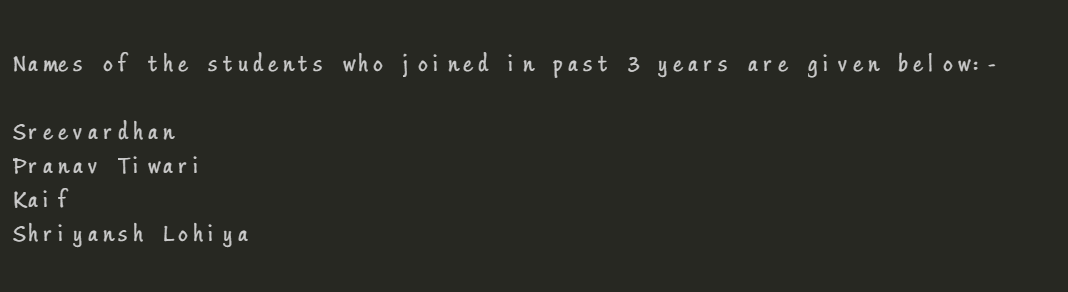
Time complexity is the amount of time taken by an algorithm to run, as a function of the length of the input. It measures the time taken to execute each statement of code in an algorithm.

In, this program the time complexity for loops is O(n) and an algorithm is said to take linear time, or O(n) time, if its time complexity is O(n). Informally, this means that the running time increases at most linearly with the size of the input. More precisely, this means that there is a constant c such that the running time is at most c\*n for every input of size n.

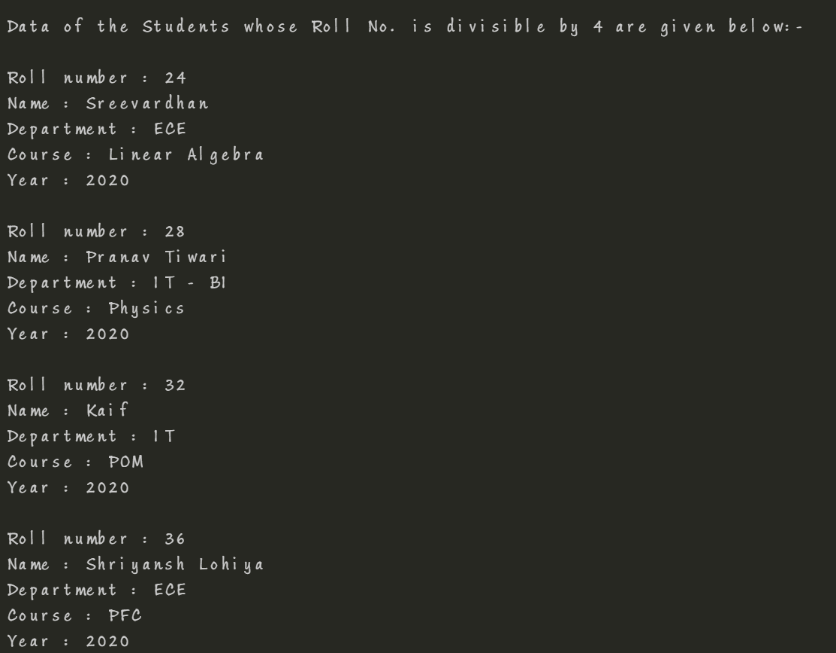
# Inputs



# Output-1



# Output-2



# References

<https://www.quora.com/What-does-scanf-n-c-name-mean> (%[^\n]%\*c)