DS Assignment 2

Name:-Sumit Sunil Koundanya PRN:-2019BTEIT00023 Class:- SYIT

Q.1) Define Structure and create structure variable with example.

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. Structure is a group of variables of different data types represented by a single name. It is also a contiguous memory segment like array, but it allows data member of different data types. Member of function can be accessed using **(.)** operator and if it is pointer to structure then it can be accessed using (->) operator.

* Syntax to define structure

struck structure\_ name

{

data-type member1\_name;

data-type member2\_name;

}

* Syntax for variable declaration

struct structure\_name variable \_name;

* No memory is allocated when we define a structure.
* Size of structure is equal to total amount of space consumed by each data member.

Following is the example of structure variable creation:-

Take the example of student structure

struct student

{

int PRN;

char nm[10];

int mrk;

};

int main()

{

struct student s;

printf(“\nEnter student No: ”);

scanf(“%d”,&s.PRN);

printf(“\n Enter student Name: ”);

scanf(“%s”,s.nm);

printf(“\nEnter student Marks: ”);

scanf(“%d”,&s.mrk);

printf(“\nStudent Data: ”);

printf(“Name: %s\nPRN: %d\nMarks:%d”,s.nm,s.PRN,s.mrk);

return 0;

}

Here size of structure is 2+ 2+ 10 bytes =14 bytes.

Q.2) Explain with example of Array of structures, passing structure to functions.

An array of structures in C can be defined as the collection of multiple structures variables where each variable contains information about different entities. The array of  structures in C are used to store information about multiple entities of different data types. The array of structures is also known as the collection of structures. An array in which each element is of type structure is called Array Of structure.

Syntax:-

struct class {

int a, b, c;

} students[10];

// C program to demonstrate the usage of an array of structures and passing structure to function

#include <stdio.h>

struct class {

int roll\_no;

char grade;

float marks;

};

void display(struct class class\_record[3])

{

int i, len = 3;

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

printf("Roll number : %d\n",

class\_record[i].roll\_no);

printf("Grade : %c\n",

class\_record[i].grade);

printf("Average marks : %.2f\n",

class\_record[i].marks);

printf("\n");

}

}

int main()

{

struct class class\_record[3]

= { { 1, 'A', 89.5f },

{ 2, 'C', 67.5f },

{ 3, 'B', 70.5f } };

display(class\_record);

return 0;

}

Above is the demonstration of an array of structures. The array holds the details of the students in a class. The details include the *roll number, grade*, *and marks*, which have been grouped under a structure (record). There exists one record for each student. This is how a collection of related variables can be assembled under a single entity for enhancing the clarity of code and increasing its efficiency. In this program, the array of structure is passed to display function by value and called through main function.

A structure can be passed to any function from main function or any sub function. Structure definition will be available within the function only. It won’t be available to other functions unless it is passed to those functions by value or by address (reference). Else we have to declare structure variable as global variable. passing structure to function can be done by 3 ways:-

1. Passing structure to function by value
2. Passing structure to a function by address(reference)
3. Declare structure as global.

Q.3) Explain with example Nested Structures.

Nested structure in C is nothing but structure within structure. One structure can be declared inside other structure as we declare structure members inside a structure. The structure variables can be a normal structure variable or a pointer variable to access the data.

* Structure within structure in C using normal variable
* Structure within structure in C using pointer variable

1. STRUCTURE WITHIN STRUCTURE IN C USING NORMAL VARIABLE:-

* This program explains how to use structure within structure in C using normal variable. “student\_college\_detail’ structure is declared inside “student\_detail” structure in this program. Both structure variables are normal structure variables.
* Please note that members of “student\_college\_detail” structure are accessed by 2 dot(.) operator and members of “student\_detail” structure are accessed by single dot(.) operator.

#include <stdio.h>

#include <string.h>

struct student\_college\_detail

{

int college\_id;

char college\_name[50];

};

struct student\_detail

{

int id;

char name[20];

float percentage;

// structure within structure

struct student\_college\_detail clg\_data;

}stu\_data;

int main()

{

struct student\_detail stu\_data = {1, "Raju", 90.5, 71145,"Anna University"};

printf(" Id is: %d \n", stu\_data.id);

printf(" Name is: %s \n", stu\_data.name);

printf(" Percentage is: %f \n\n", stu\_data.percentage);

printf(" College Id is: %d \n",

stu\_data.clg\_data.college\_id);

printf(" College Name is: %s \n",

stu\_data.clg\_data.college\_name);

return 0;

}

2)STRUCTURE WITHIN STRUCTURE (NESTED STRUCTURE IN C ) USING POINTER VARIABLE:

* This program explains how to use structure within structure in C using pointer variable. “student\_college\_detail’ structure is declared inside “student\_detail” structure in this program. one normal structure variable and one pointer structure variable is used in this program.
* Please note that combination of .(dot) and ->(arrow) operators are used to access the structure member which is declared inside the structure.

#include <stdio.h>

#include <string.h>

struct student\_college\_detail

{

int college\_id;

char college\_name[50];

};

struct student\_detail

{

int id;

char name[20];

float percentage;

// structure within structure

struct student\_college\_detail clg\_data;

}stu\_data, \*stu\_data\_ptr;

int main()

{

struct student\_detail stu\_data = {1, "Raju", 90.5, 71145, "Anna University"};

stu\_data\_ptr = &stu\_data;

printf(" Id is: %d \n", stu\_data\_ptr->id);

printf(" Name is: %s \n", stu\_data\_ptr->name);

printf(" Percentage is: %f \n\n",

stu\_data\_ptr->percentage);

printf(" College Id is: %d \n",

stu\_data\_ptr->clg\_data.college\_id);

printf(" College Name is: %s \n",

stu\_data\_ptr->clg\_data.college\_name);

return 0;

}

Q.4) Define Algorithm, Pseudo-code , ADT , Data Structure, Algorithmic Efficiency, Recursion:

1) Algorithm:- It is a process or set of rules to be followed in calculation or other problem solving operations, especially by a computer. Each algorithm begins with header that names it and describes its parameter and list any pre and post condition. This information is important because often programmers using algorithms sees only header information not complete algorithm.

2) Pseudo-code:- Pseudocode is an English like representation of the algorithm.It is part English and part structured code. The English part provide a relaxed syntax that describes what must be done without showing unnecessary details such as error messages. The code part consists of extended version of basic algorithm constructs – sequence, selection and iteration. One of the most common tools for defining algorithm is pseudo code .

3) Abstract Datatype (ADT) :-It referes to a set of data values and associated operations that are specified accurately independent of any particular implementation. Using the ADT , we can not only read characters but we can also convert them into different data structure such as integer and strings.In the other words, ADT consists of set o definitions that are allow the programmers to use the functions while hiding implementation.

4) Data Structure:- Data Structure is an aggregation of atomic and composite data into a set with defined relationship structure means set off rules that holds data together. Data structure can be nested. We can have data structure that consists of other data structure. Data Structures in C are used to store data in an organised and efficient manner. The C Programming language has many data structures like an array, stack, queue, linked list, tree, etc. A programmer selects an appropriate data structure and uses it according to their convenience.

5) Algorithm Efficiency:- Algorithmic efficiency is a property of an algorithm which relates to the number of computational resources used by the algorithm. An algorithm must be analyzed to determine its resource usage, and the efficiency of an algorithm can be measured based on usage of different resources. If function is linear, that is, if it contain no loop or recursion its efficiency is function of number of iterations it contains. The efficiency depends upon the speed of computer and not on a factor in the overall efficiency .The general format is

F(n) = Efficiency

* Linear loop - f(n) =n
* Logarithmic loop - f(n) =log n
* Linear logarithmic loop – f(n) = nlogn
* Quadratic loop – f(n) =n^2
* Dependent Quadratic loop- f(n)=n\*(n+1)/2

6) Recursion:- Recursion is the process which comes into existence when a function calls a copy of itself to work on a smaller problem. Any function which calls itself is called recursive function, and such function calls are called recursive calls. ... For Example, recursion may be applied to sorting, searching, and traversal problems. Recursion is a repetitive process in which an algorithm call itself. Repetitive algorithm uses recursion whenever the algorithm appears within the definition itself. Every recursive call must solve the part of problem or reduce the size of problem. Ex- Tower of Hanoi, Ackermann formula, Factorial , Fibonacci series etc. are example of recursion.

Q.5) Direct and Indirect recursion with example.

1) Direct Recursion

If a function calls itself, it’s known as **direct recursion**. This results in a one-step recursive call: the function makes a recursive call inside its own function body.

#include<stdio.h>

unsigned long long int factorial(unsigned int i)

{

if(i<= 1)

{

return 1;

}

return i \* factorial(i - 1);

}

int main()

{

int i = 12;

printf("Factorial of %d is %ld\n", i, factorial(i));

return 0;

}

In the above example factorial function is call itself so it is direct recursion example. It will continue till the condition in while loop gets failed i.e. i<=1 .  The recursive program has greater space requirements than iterative program as all functions will remain in the stack until the base case is reached. It also has greater time requirements because of function calls and returns overhead.

2) Indirect Recursion

If the function f1 calls another function f2 and f2 calls f1 then it is **indirect recursion** (or mutual recursion). This is a two-step recursive call: the function calls another function to make a recursive call. For indirect recursion, both the functions need to be declared **before** they are defined.

#include<stdio.h>

#define N 20;

int n = 1;

void fun1()

{

if (n <= N)

{

printf("%d", n);

n++;

fun2();

}

else

return;

}

void fun2()

{

if (n <= N)

{

printf("%d", n);

n++;

fun1();

}

else

return;

}

int main(void)

{

fun1();

return 0;

}

In the above program fun 1 and fun 2 two functions are used to perform indirect recursion. When fun 1 is called through main it print the first value of n i.e 1 and due to increment in n , it become 2 .In fun1 after increment fun2 is called so in fun2 it print the value of n i.e 2. In this way program is executed recursively(indirect) till value of N become 20. So we get the output of numbers from 1 to 20.

**Algorithms-**

**1)** To add two distances (in inch-feet system) using structure

Algorithm:- Adds two distances in feet-inch system

Pre: distances must be greater then 0

Post: compute sum of two distances by using structure member.

Step 1: start

Step 2: Define a structure ‘distance ‘as name with member

1. int feet
2. int inch

Step 3:create structure variable say d1,d2,d3

Step 4: access d1 an d2 using structure

Step 5: Compute:

1. d3.feet=d1.feet+d2.feet
2. d3.inch=d1.inch+d2.inch
3. if(d3.inch >12)
4. d3.inch=d3.inch-12
5. d3.feet++

Step 6: print added distance (d3.feet,d3.inch)

Step 7: stop.

**2)** To add two complex number by passing structure to a function

Algorithm: To add two complex number by passing structure to a function

Pre: two complex no

Post: compute sum

Step 1: Start

Step 2: define structure ‘complex’ with member as

1. int real
2. int img

Step 3:declare structure variables c1 and c2 ,c3.

Step 4: input/Access c1 and c2 using structure

Step 5: Define add function which takes c1 and c2 as argument

1. c3.real=c1+real +c2.real
2. c3.img=c1.img+c2.img

Step 6: return c3 to main function which contains added result

Step 7: stop

**3)** To store data in structures Dynamically

Algorithm:- Dynamic allocation of structure

Pre: number of student(n)

Post: create memory dynamically and free memory

Step 1: start

Step 2: define structure ‘student’ as member

1. int mrk

Step 3:create structure variable as \*ptr

Step 4: input n ; set 0 to i ;

Step 5: Allocate memory dynamically using malloc function

Step 6: loop(i<n)

Display the element of structure (\*(ptr+i))

Step 7: end loop

Step 8: stop

**4)** Recursive program to find gcd of two numbers ,to generate Fibonacci series

* Algorithm: computes gcd of two number using recursion

Pre: number must have data

Post: gcd of two number

Step 1: int gcd\_algorithm(int x, int y)

Step 2: if (y == 0) {

return x;

} else if (x >= y && y > 0) {

return gcd\_algorithm(y, (x % y));

Step 3: int main(void)

Step 4: int num1, num2, **gcd**;

Step 5: compute gcd

Step 6: stop

* Algorithm: To generate Fibonacci series

Pre: Number of terms >0 and take fib as functionv

Post: Get Fibonacci series

Step 1: define recursion for Fibonacci series

Step 2: input n

Step 3: set i to 0

Step 4: in fibo function

1. If(n==0)

Return 0

1. If(n==1)

Return 1

1. Else return fib(n-1) + fib(n-2)

Step 5: print Fibonacci series

Step 6: stop