**CHAPTER 7 : STRUCTURES AND UNIONS**

**Q. Create a structure named Employee with structure members name, eid, address and gender. Structure need to read information for 50 employees. Write all content into the file info.dat and while retrieving display only the information of those employee whose address is “Kathmandu”. [2019-Fall]**

 Solution:

#include<stdio.h> #include<string.h> struct Employee

{

char name[256],address[256],gender; int eid;

};

int main()

{

struct Employee e[50],temp; int i;

FILE \*fptr;

printf("Enter details of 50 employees :\n"); for(i=0;i<50;i++)

{

printf("Employee %d :\n",i+1); printf("Name : ");

gets(e[i].name);

printf("eid : "); scanf("%d",&e[i].eid); printf("Address : ");

gets(e[i].address);

printf("Gender :"); scanf("%c",&e[i].gender);

}

fptr=fopen("info.dat","w"); //opening in write mode for(i=0;i<50;i++)

{

fprintf(fptr,"\t%s %s %c %d\n"

,e[i].name,e[i].address,e[i].gender,e[i].eid);

}

fclose(fptr);

fptr=fopen("info.dat","r"); //opening in read mode printf("Employee details who are from Kathmandu :\n"); char c;

while(c=(fgetc(fptr))!=EOF) //read until end of file

{

fscanf(fptr,"\t%s %s %c

%d",&temp.name,&temp.address,&temp.gender,&temp.eid); if(strcmp("Kathmandu",temp.address)==0)

{

printf("Name : %s \t Eid : %d \t Gender: %c \n", temp.name,temp.eid,temp.gender);

}

}

fclose(fptr); //closing the file

return 0;

}

### Q. What is the significance of file pointer in file handling. Consider a following structure

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Roll No | Name | Address | Faculty | Date of Birth | | | | |
|  |  |  |  |  | mm | dd | yy |  |
|  |  |  |  |

**Write a program to create “student.txt” file to store the above records for 100 students. Also display these records of students who are not from Kathmandu.[2018-Spring]**

Solution:

#include<stdio.h> #include<string.h> struct DateofBirth

{

int mm,dd,yy;

};

struct Student

{

char name[256],address[256],faculty[256]; int roll;

struct DateofBirth dob;

};

int main()

{

struct Student std[100],temp; int i;

FILE \*fptr;

printf("Enter details of 100 students :\n"); for(i=0;i<100;i++)

{

printf("Student %d :\n",i+1); printf("Name : ");

gets(std[i].name);

printf("Roll no : "); scanf("%d",&std[i].roll); printf("Address : "); gets(std[i].address); printf("Faculty :"); gets(std[i].faculty);

printf("Date of birth (mm/dd/yy) :"); scanf("%d%d%d",&std[i].dob.mm,&std[i].dob.dd,&std[i].dob.yy);

}

fptr=fopen("student.txt","w"); //opening in write mode for(i=0;i<100;i++)

{

fprintf(fptr,"\t%s %d %s %s %d-%d-%d\n" ,std[i].name,std[i].roll, std[i].address,std[i].faculty,std[i].dob.mm,std[i].dob.dd,std[i].dob. yy);

}

fclose(fptr);

fptr=fopen("student.txt","r"); //opening in read mode

printf("Student records who are not from Kathmandu :\n"); char c;

while(c=(fgetc(fptr))!=EOF) //read until end of file

{

fscanf(fptr,"\t%s %d %s %s %d-%d-%d\n" ,&temp.name,&temp.roll, &temp.address,&temp.faculty,&temp.dob.mm,&temp.dob.dd,&temp.dob.yy); if(strcmp("Kathmandu",temp.address)!=0)

{

printf("Name : %s \t Roll : %d \t Address: %s \t Faculty: %s \t Dateofbirth: %d-%d-%d

\n",temp.name,temp.roll,temp.address,temp.faculty, temp.dob.mm,temp.dob.dd,temp.dob.yy);

}

}

fclose(fptr); //closing the file return 0;

}

### Q. Write a program to create structure for the following data for cricket game. (Country name, Player name, Playing type(e.g. bating, balling or both), Number of matches played by player and salary). Save the information in a file named “cricket.txt” and display the information of those players who had played more than 10 matches. [2018-Fall]

 Solution:

#include<stdio.h> struct Cricket

{

char cName[256],pName[256],pType[100]; int matches;

float salary;

};

int main()

{

int n,i;

printf("How many player records to enter ?\n"); scanf("%d",&n);

struct Cricket player[n],temp; FILE \*fptr;

printf("Enter details of %d students :\n",n); for(i=0;i<n;i++)

{

printf("Player %d :\n",i+1); printf("Player name : "); gets(player[i].pName); printf("Country name : "); gets(player[i].cName); printf("Playing type : "); gets(player[i].pType); printf("No.of matches :"); scanf("%d",&player[i].matches); printf("Salary :"); scanf("%f",&player[i].salary);

}

fptr=fopen("cricket.txt","w"); //opening in write mode for(i=0;i<n;i++)

{

fprintf(fptr,"\t%s %s %s %d %f\n",player[i].pName,player[i].cName, player[i].pType,player[i].matches,player[i].salary);

}

fclose(fptr);

fptr=fopen("cricket.txt","r"); //opening in read mode printf("Players who have played more than 10 matches :\n"); char c;

while(c=(fgetc(fptr))!=EOF) //read until end of file

{

fscanf(fptr,"\t%s %s %s %d %f\n",&temp.pName,&temp.cName, &temp.pType,&temp.matches,&temp.salary); if(temp.matches>10)

{

printf("Name : %s \t Country : %s \t Playing-type: %s \t Matches:

%d \t Salary: %.2f \n",temp.pName,temp.cName,temp.pType, temp.matches,temp.salary);

}

}

fclose(fptr); //closing the file return 0;

}

### Q. Differentiate structure and union. How the members of structure are accessed? [2017-Spring]



|  |  |
| --- | --- |
| Structure | Union |
| The keyword **struct** is used to define a structure. | The keyword **union** is used to define a union. |
| Each member of a structure is assigned its own unique storage. It takes more memory spaces than that of union | All members of a union share the memory space. It takes less memory spaces than that of structure. |
| The amount of memory required to store a structure is the sum of the memory spaces required by all its members. | The amount of memory required to store a union is same as memory size occupied by a member which requires the largest memory space among the members. |
| All the members of structure can be accessed at  any point of the time. | Only one member of union can be accessed at any  given time. |
| Structure is declared as struct book  {  char name[100],author[100]; int bookid;  float price;  }; | Union is declared as union book  {  char name[100],author[100]; int bookid;  float price;  }; |

A structure member can be accessed using period or dot (i.e. “.”) operator. The synatax for accessing of a structure variable is as follows:

*structure\_name.member*

where **structure\_name** refers to the name of the structure-type variable and **member** refers the name of a member within the structure. Notice the period (.) that separates the structure variable name from the member name. For nested structure, the member is accessed as:

*structure\_name.member\_structure.member*

An example to illustrate the processing of a structure:

#include<stdio.h> struct book

{

char name[256]; char author[256]; int bookId;

};

int main()

{

struct book b = {"C:Questions And Answers","PU Tech Club",101};

//structure can be initialized in same way as array printf("Book Details:\n");

printf("Name: %s\n",b.name); //accessing a structure member printf("Author: %s\n",b.author);

printf("Book Id: %d\n",b.bookId); return 0;

}

### Q. How do you declare and initialize array of structure variables? How is structure different from Union. Give example codes [2017-Fall]



A structure can be declared using keyword **struct.** A structure is declared as:

struct structure\_name

{

//body

}

A structure variable can be initialized in a similar manner as that of an array. The members to be initialized must appear in order as in the definition of structure within braces and separated by commas. However, C does not allow the initialization of individual structure members within its definition.

Example:

struct student

{

char name[256]; char address[256]; int roll;

};

struct student std = {"Ram","Hemja",1}; //intialization of structure

It is same as :

struct student std; strcpy(std.name,"Ram"); strcpy(std.address,"Hemja"); std.roll=1;

### Q. Self-referential structure [Short note:2017-Fall,2015-Spring]

 A structure which has at least one member of type pointer to the same structure is known as self- referential structure. In other words, structures pointing to the same type of structures are self- referential in nature.

Example:

struct node {

int data1; char data2;

struct node\* link;

};

int main()

{

struct node ob; return 0;

}

Here, ‘link’ is a pointer to a structure of type ‘node’. Hence, the structure ‘node’ is a self-referential structure with ‘link’ as the referencing pointer.

Self-referential structures are very useful in creation of other complex data structures like linked lists, stacks, queues, trees, etc.

### Q. Write a program to create structure for the following data for student (RN, Name, phone, address and semester). Read the 10 students by user and write only those students whose semester is 1 in file “student.txt”. [2016-Fall]

 Similar to 2018-Fall question

### Q. Differentiate structure and union. How the members of Nested structure are accessed? Show it with example. [2016-Fall]

 For difference, please look at fore page.

A structure which is defined as a member to another structure is known as nested structure. A nested structure can be accessed in same manner as any other member of structure.

### Q. What do you mean by nested structure? Write a program to explain nested structure. [2015- Fall]

 One structure may be nested within another structure’s definition. In other words, a structure may be defined as a member of another structure. The outer structure is known as nesting structure and the inner structure (member structure) is nested structure. The nested structure can be accessed using dot (.) operator just like any other member of structure.

A nested may be created as follows:

struct nesting\_structure { data-type variable;

..

struct nested\_structure name;

};

Here, **nested\_strucutre** is an already existing structure. A nested structure may be accessed as follows: *nesting\_strucutre.nested\_structure.member* An example to explain nested structure; #include<stdio.h>

struct date

{

int mm,dd,yy;

};

struct student //nesting structure

{

char name[256]; int roll;

struct date dob; //nested structure

};

int main()

{

struct student st ={ "Laxman",

15,

{11,11,1999}

}; //initializing structure printf("Student details :\n"); printf("Name : %s\n",st.name); printf("Roll : %d\n",st.roll);

printf("Date of birth : %d-%d-%d\n",st.dob.mm,st.dob.dd,st.dob.yy); return 0;

}

### Q. Define a structure called ‘football’ that will describe the following information: player name, country name, number of goal scored

**Using football, declare an array player with 50 elements and write a program to read the information about all the 50 players and print a country-wise list containing names of players with their number of goals scored. [2014-Fall]**

 Solution:

#include<stdio.h> #include<string.h> struct football

{

char pName[256],cName[256]; int goals;

};

int main()

{

struct football player[50],temp; int arr[50],i,j;

printf("Enter details of 50 students :\n"); for(i=0;i<50;i++)

{

printf("Player %d :\n",i+1); printf("Player name : "); gets(player[i].pName); printf("Country name : "); gets(player[i].cName); printf("No.of goals scored :"); scanf("%d",&player[i].goals);

}

printf("List of player according to country:\n"); for(i=0;i<50;i++)

{

if(arr[i]!=0)

{

printf("Country : %s\n",player[i].cName);

printf("Player name: %s \t Goals scored : %d \n",player[i].pName, player[i].goals);

arr[i]=0; //0 for printed for(j=0;j<50;j++)

{

if(arr[j]!=0 && strcmp(player[i].cName,player[j].cName)==0)

{

printf("Player name: %s \t Goals scored : %d

\n",player[j].pName, player[j].goals);

arr[j]=0;

}

}

}

}

return 0;

}

### Extra questions:

1. **How to return multiple values from a function in C ?**

 By default, a non-void function returns a single value. But programmers often need to return multiple values from a function. Luckily, there are several workarounds in C to return multiple values.

1. Using pointers : We can use pointers in C to return more than one value from the function by passing pointers as function parameters and use them to set multiple values, which will then have visibility in the caller function.

Example:

#include<stdio.h>

void setValue(int \*x, int \*y, char \*z)

{

\*x=1, \*y=1, \*z='A';

}

int main()

{

int a,b; char c;

setValue(&a,&b,&c);

printf("a = %d, b = %d, c = %c\n",a,b,c); return 0;

}

### OUTPUT:

a = 1, b = 1, c = A

1. Using structure : We can also use structures in C to return more than one value from the function. We know that a structure is user defined datatype in C that can hold several data types of the same or different kind. The idea is to create a struct containing all required data types as its members and return that struct from our function. Then we can retrieve the desired values from the struct inside our caller function.

Example:

#include<stdio.h> struct items { int x,y;

char z;

};

struct items setValue() //function to return multiple values

{

struct items temp = {1,1,'A'};

return temp; //returning a struture from function

}

int main()

{

int a,b; char c;

struct items test = setValue(); a = test.x;

b = test.y; c = test.z;

printf("a = %d, b = %d, c = %c\n",a,b,c); return 0;

}

### OUTPUT:

a = 1, b = 1, c = A

1. Using array: We have seen how to return values of different data types from the function using pointers and structure. Now if all values are of same datatype, we can encapsulate the values in an array and return that array as

shown below: Example: #include<stdio.h>

void setValue(int arr[])

{

arr[0]=0;

arr[1]=1;

arr[2]=2;

}

int main()

{

int a,b,c; int temp[3];

setValue(temp); a= temp[0];

b= temp[1]; c=temp[2];

printf("a = %d, b = %d, c = %d\n",a,b,c); return 0;

}

### OUTPUT:

a = 0, b = 1, c = 2