**STRUCTURES AND UNIONS**

**Introduction**

Structure is a collection of logically related data items of different data types. For example, it can be used to represent a set of attributes, such as student \_ name, roll \_ number, course and marks. The concept of a structure is analogous to that of a ‘record’ in many other languages. More examples of such structures are:

book : author, title, price, year

city : name, country, population

time : seconds, minutes, hours

date : day, month, year

**Structure Definition**

Unlike arrays, structure must be defined first for their format that may be used later to declare structure variables. Consider a student\_rec database consisting of roll\_no, name, course and fees. A structure definition to hold this information is as follows:

struct student\_rec

{

int roll\_no;

char name[20];

char course[15];

float fees;

};

The keyword struct declares a structure to hold the details of four data fields, namely roll\_no, name, course and fees. These fields are called structure elements or members. 'student\_rec’ is the name of the structure and is called the structure tag. The tag name may be used subsequently to declare variables that have the tag’s structure.

Another example: Structure for book can be defined as follows:

struct book

{

char title[20];

char author[20];

int no\_pages;

float price;

int year\_pub;

};

The general format of a structure definition is as follows:

struct tag \_ name

{

data \_ type member1;

data \_ type member2;

---------

---------

};

Note that the structure declaration, however, does not allocate memory or consume storage space. It just gives the template that conveys to the C compiler how the structure is laid out in the memory and gives details of the member names. Memory is allocated for the structure when variable of the structure is declared.

The general format of a structure variable declaration is as follows:

struct tag \_ name

{

data \_ type member1;

data \_ type member2;

---------

---------

};

struct tag\_name struct\_var;

A variable of student\_rec can be declared by using the following statement after structure definition:

struct student\_rec stud1;

Another way of declaring structure is using the following syntax:

struct student\_rec

{

int roll\_no;

char name[20];

char course[15];

float fees;

} stud1, stud2;

**Type-Defined structures**

The keyword typedef can be used to define a structure as follows:

typedef struct

{

type member1;

type member2;

-------

--------

} type\_name;

The type\_name represents structure definition associated with it and therefore can be used to declare structure variables as shown below:

type\_name var1, var2, . . . . ;

In defining a structure, the following syntax is to be noted:

* The template is terminated with a semicolon.
* While the entire definition is considered as a statement, each member is declared independently for its name and type in a separate statement inside the template.
* The tag name such as student\_rec can be used to declare structure variables of its type, later in the program.
* When structure variables are declared, memory is allocated separately for each variable

**Array Vs Structure**

* An array is a collection of related data elements of same type. Structure can have elements of different types.
* An array is derived data type whereas structure is a programmer-defined one.
* Any array behaves like a built- in data type, wherein array variable can be declared and used. But in the case of a structure, there is a need to design and declare a data structure before the variables of that type are declared and used.

**Giving Values To Members**

The members of a structure can be accessed and assigned values in a number of ways. The members themselves are not variables. They should be linked to the structure variables in order to make them meaningful members. For example, the word title has no meaning, whereas the phrase ‘title of book’ has a meaning. The link between a member and a variable is established using the member operator ’.’,which is also known as ‘dot operator’ or ‘period operator’. For example,

book1.price

is the variable representing the price of the book1 and can be treated like any other ordinary variable. Here is how values to the member of book1 can be assigned:

strcpy(book1.title, “COBOL”); strcpy(book1.author, “M.K.ROY”); book1.pages = 350;

book1. price =140.50;

Also scanf can be used to give the values through the keyboard:

scanf(“%s”, book1.title); scanf(“%d”, &book1.pages);

**Example 1:**

Define a structure type, struct personal, that would contain person name, date of joining and salary. Using this structure, write a program to read this information for one person from the keyboard and print the same on the screen.

Structure definition along with the program is shown below. The scanf and printf functions illustrate how the member operator ‘.’ is used to link the structure members to the structure variables. The variable name with a period and the member name is used like an ordinary variable.

*Program*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* DEFINING AND ASSIGNING VALUES TO STRUCTURE MEMBERS \*/ /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

struct personal

{

char name[20];

int day;

char month[10]; int year;

float salary;

};

main()

{

struct personal person;

printf(“Input values\n”);

scanf(“%s %d %s %d %f”, person .name, &person. day,

person.month, &person.year, person.salary);

printf(“%s %d %s %d %.2f\n”,person .name, person. day, person.month,

person.year, person.salary);

}

## Example 2: Accessing Structure Members

#include <stdio.h>

#include <string.h>

struct Books

{

char title[50];

char author[50];

char subject[100];

int book\_id;

};

int main( )

{

struct Books Book1; /\* Declare Book1 of type Book \*/

struct Books Book2; /\* Declare Book2 of type Book \*/

/\* book 1 specification \*/

strcpy( Book1.title, "C Programming");

strcpy( Book1.author, "Nuha Ali");

strcpy( Book1.subject, "C Programming Tutorial");

Book1.book\_id = 6495407;

/\* book 2 specification \*/

strcpy( Book2.title, "Telecom Billing");

strcpy( Book2.author, "Zara Ali");

strcpy( Book2.subject, "Telecom Billing Tutorial");

Book2.book\_id = 6495700;

/\* print Book1 info \*/

printf( "Book 1 title : %s\n", Book1.title);

printf( "Book 1 author : %s\n", Book1.author);

printf( "Book 1 subject : %s\n", Book1.subject);

printf( "Book 1 book\_id : %d\n", Book1.book\_id);

/\* print Book2 info \*/

printf( "Book 2 title : %s\n", Book2.title);

printf( "Book 2 author : %s\n", Book2.author);

printf( "Book 2 subject : %s\n", Book2.subject);

printf( "Book 2 book\_id : %d\n", Book2.book\_id);

return 0;

}

When the above code is compiled and executed, it produces the following result:

Book 1 title : C Programming

Book 1 author : Nuha Ali

Book 1 subject : C Programming Tutorial

Book 1 book\_id : 6495407

Book 2 title : Telecom Billing

Book 2 author : Zara Ali

Book 2 subject : Telecom Billing Tutorial

Book 2 book\_id : 6495700

**Structure Initialization**

A structure variable can be initialized at compile time.

main( )

{

struct

{

int roll\_no;

char name[20];

}

student ={10, “Asha”};

………

………

}

This assigns the value 10 to student. roll\_no and Asha to student. name. There is a one-to-one correspondence between the members and their initializing values.

A lot of variation is possible in initializing a structure. The following statements initialize two structure variables. Here, it is essential to use a tag name.

main()

{

struct st \_ record

{

int roll\_no;

char name[20];

};

struct st\_record student1 ={10, “Asha”}; struct st\_record student2 ={73, “Usha”};

………

………

}

C language does not permit the initialization of individual structure member within the template. The initialization must be done only in the declaration of the actual variables.

**Comparison Of Structure Variables**

Two variables of the same structure type cannot be compared the same way as ordinary variables. If student1 and student2 belong to the same structure, then the following operations are invalid:

student 1 = = student 2;

student 1 != student 2;

C does not permit any logical operations on structure variables. In such cases, individual member can be compared using logical operators.

However, two variables of the same structure type can be copied the same way as ordinary variables.

student 1 = student 2; // Assign person2 to person1.

student 2 = student 1; // Assign person1 to person 2.

**Arrays of Structure**

Structure is used to describe the format of a number of related variables. For example, in analyzing the marks obtained by a class of students, we may use a template to describe student name and marks obtained in various subjects and then declare all the students as structure variables. In such cases, we may declare an array of structure, each elements of the array representing a structure variable. Array of structures is nothing but collection of structures. This is also called as structure array in C.

The general syntax for declaring an array of a structure

struct tag \_ name

{

data \_ type member1;

data \_ type member2;

---------

---------

};

struct tag\_name struct\_var[index];

For example,

struct class student[50];

It defines an array called student, that consists of 50 elements. Each elements is defined to be of the type struct class. Consider the following declaration:

struct marks

{

int subject1;

int subject2;

int subject3;

};

main()

{

struct marks student[3] =

{ {79, 68, 81}, {75, 53, 69}, {59,36,74}};

This declares the student as an array of three elements students[0], student[1], and student[2] and initializes their members as follows:

student[0].subject1=79;

student[0].subject2=68;

………….

………….

student[2].subject3=74;

An array of structures is stored inside the memory in the same way as a multi- dimensional array.

**Example program for array of structures in C:**

This program is used to store and access “id, name and percentage” for 3 students. Structure array is used in this program to store and display records for many students.

#include <stdio.h>

#include <string.h>

struct student

{

int id;

char name[30];

float percentage;

};

int main()

{

int i;

struct student record[2];

// 1st student's record

record[0].id=1;

strcpy(record[0].name, "Raju");

record[0].percentage = 86.5;

// 2nd student's record

record[1].id=2;

strcpy(record[1].name, "Surendren");

record[1].percentage = 90.5;

// 3rd student's record

record[2].id=3;

strcpy(record[2].name, "Thiyagu");

record[2].percentage = 81.5;

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

{

printf(" Records of STUDENT : %d \n", i+1);

printf(" Id is: %d \n", record[i].id);

printf(" Name is: %s \n", record[i].name);

printf(" Percentage is: %f\n\n",record[i].percentage);

}

return 0;

}

**Output:**

|  |
| --- |
| **Records of STUDENT : 1** Id is: 1 Name is: Raju Percentage is: 86.500000  **Records of STUDENT : 2** Id is: 2 Name is: Surendren Percentage is: 90.500000  **Records of STUDENT : 3** Id is: 3 Name is: Thiyagu Percentage is: 81.500000 |

**Arrays within Structures**

C permits the use of arrays as structure members. For example, the following structure declaration is valid:

struct marks

{

int number;

float subject[3];

} student[2];

Here, the member subject contains three elements, subject [0], subject [1] and subject [2]. These elements can be accessed using appropriate subscripts. For example,

student[1].subject[2];

would refer to the marks obtained in the third subject by the second student.

**Structure within Structures**

Nesting of structures is permitted in C. Let us consider the following structure defined to store information about the salary of employees.

struct salary

{

char name[20];

char department[10];

int basic \_ pay;

int dearness\_ allowance;

int house \_ rent \_ allowance;

int city\_ allowance;

} employee;

This structure defines name, department , basic pay and three kinds of allowances. All items related to allowance together can be grouped and declare them under a substructure as shown below:

struct salary

{

char name[20];

char department[10]; struct

{

int dearness;

int house\_rent; int city;

}

allowance;

}employee;

An inner structure can have more than one variable. The following form of declaration is legal:

struct salary

{

……

struct

{

int dearness; …….

}

allowance,

arrears;

}

employee[100];

It is also possible to nest more than one type of structures.

struct personal\_record

{

struct name\_part name; struct addr\_part address; struct date date \_ of \_ birth ……..

……..

};

struct personal\_record person1;

The first member of this structure is name which is of the type struct name\_part. Similarly, other members have their structure types.

**Example for Nested structure in C**

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;

}

**Output:**

|  |
| --- |
| Id is: 1 Name is: Raju Percentage is: 90.500000  College Id is: 71145 College Name is: Anna University |

**Structures and Function**

C supports the passing of structure values as arguments to functions. There are three methods by which the values of a structure can be transferred from one function to another.

* The first method is to pass each member of the structure as an actual argument of the function call.
* The second method involves passing of a copy of the entire structure to the called function.
* The third approach employs a concept called pointers to pass the structure as an argument.

The general format of sending a copy of a structure to the called function is:

*function \_name(structure\_ variable\_ name);*

The called function takes the following form:

data\_type function\_name(struct\_ type st\_name)

{

……….

………..

return (expression);

}

The following points are important to note:

1. The called function must be declared for its type, appropriate to the data type it is expected to return. For example, if it is returning a copy of the entire structure, then it must be declared as struct with an appropriate tag name.
2. The structure variable used as the actual argument and the corresponding formal argument in the called function must be of the same struct type.
3. The return statement is necessary only when the function is returning some data. The expression may be any simple variable or structure variable or an expression using simple variables.
4. When a function returns a structure, it must be assigned to a structure of identical type in the calling function.
5. The called function must be declared in the calling function for its type, if it is placed after the calling function.

**Example program – passing structure to function in C by value:**

In this program, the whole structure is passed to another function by value. It means the whole structure is passed to another function with all members and their values. So, this structure can be accessed from called function. This concept is very useful while writing very big programs in C.

#include <stdio.h>

#include <string.h>

struct student

{

int id;

char name[20];

float percentage;

};

void func(struct student record);

int main()

{

struct student record;

record.id=1;

strcpy(record.name, "Raju");

record.percentage = 86.5;

func(record);

return 0;

}

void func(struct student record)

{ printf(" Id is: %d \n", record.id);

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

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

}

**Output:**

|  |
| --- |
| Id is: 1  Name is: Raju  Percentage is: 86.500000 |

**Example program – Passing structure to function in C by address:**

In this program, the whole structure is passed to another function by address. It means only the address of the structure is passed to another function. The whole structure is not passed to another function with all members and their values. So, this structure can be accessed from called function by its address.

#include <stdio.h>

#include <string.h>

struct student

{

int id;

char name[20];

float percentage;

};

void func(struct student \*record);

int main()

{

struct student record;

record.id=1;

strcpy(record.name, "Raju");

record.percentage = 86.5;

func(&record);

return 0;

}

void func(struct student \*record)

{

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

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

printf(" Percentage is: %f \n", record->percentage);

}

**Output:**

|  |
| --- |
| Id is: 1  Name is: Raju  Percentage is: 86.500000 |

**Example program to declare a structure variable as global in C:**

Structure variables also can be declared as global variables as we declare other variables in C. So, When a structure variable is declared as global, then it is visible to all the functions in a program. In this scenario, we don’t need to pass the structure to any function separately.

#include <stdio.h>

#include <string.h>

struct student

{

int id;

char name[20];

float percentage;

};

struct student record; // Global declaration of structure

void structure\_demo();

int main()

{

record.id=1;

strcpy(record.name, "Raju");

record.percentage = 86.5;

structure\_demo();

return 0;

}

void structure\_demo()

{

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

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

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

}

**Output:**

|  |
| --- |
| Id is: 1  Name is: Raju  Percentage is: 86.500000 |

**Unions**

C Union is also like structure, i.e. collection of different data types which are grouped together. Each element in a union is called member.

Like structures, a union can be declared using the keyword union as follows:

union item

{

int m;

float x;

char c;

} code;

This declares a variable code of type union item.

The compiler allocates a piece of storage that is large enough to hold the largest variable type in the union.

To access a union member, the same syntax that used for accessing structure members can be used. Example :

code.m

code.x

code.c

are all valid member variables.

During accessing, make sure that we are accessing the member whose value is currently stored. For example, the statement such as

code.m = 379; code.x=7859.36; printf(“%d”, code.m);

would produce erroneous output.

In effect, a union creates a storage location that can be used by any one of its members at a time. When a different member is assigned a new value, the new value supercedes the previous member’s value.

**Example program for C union:**

#include <stdio.h>

#include <string.h>

union student

{

char name[20];

char subject[20];

float percentage;

};

int main()

{

union student record1;

union student record2;

// assigning values to record1 union variable

strcpy(record1.name, "Raju");

strcpy(record1.subject, "Maths");

record1.percentage = 86.50;

printf("Union record1 values example\n");

printf(" Name : %s \n", record1.name);

printf(" Subject : %s \n", record1.subject);

printf(" Percentage : %f \n\n", record1.percentage);

// assigning values to record2 union variable

printf("Union record2 values example\n");

strcpy(record2.name, "Mani");

printf(" Name : %s \n", record2.name);

strcpy(record2.subject, "Physics");

printf(" Subject : %s \n", record2.subject);

record2.percentage = 99.50;

printf(" Percentage : %f \n", record2.percentage);

return 0;

}

**Output:**

|  |
| --- |
| Union record1 values example Name : Subject : Percentage : 86.500000; Union record2 values example Name : Mani Subject : Physics Percentage : 99.500000 |

**Explanation for above C union program:**

There are 2 union variables declared in this program to understand the difference in accessing values of union members.

**Record1 union variable:**

* “Raju” is assigned to union member “record1.name” . The memory location name is “record1.name” and the value stored in this location is “Raju”.
* Then, “Maths” is assigned to union member “record1.subject”. Now, memory location name is changed to “record1.subject” with the value “Maths” (Union can hold only one member at a time).
* Then, “86.50” is assigned to union member “record1.percentage”. Now, memory location name is changed to “record1.percentage” with value “86.50”.
* Like this, name and value of union member is replaced every time on the common storage space.
* So, we can always access only one union member for which value is assigned at last. We can’t access other member values.
* So, only “record1.percentage” value is displayed in output. “record1.name” and “record1.percentage” are empty.

**Record2 union variable:**

* If we want to access all member values using union, we have to access the member before assigning values to other members as shown in record2 union variable in this program.
* Each union members are accessed in record2 example immediately after assigning values to them.
* If we don’t access them before assigning values to other member, member name and value will be over written by other member as all members are using same memory.
* We can’t access all members in union at same time but structure can do that.

**Difference between structure and union in C:**

|  |  |  |
| --- | --- | --- |
| **S.no** | **C Structure** | **C Union** |
| 1 | Structure allocates storage space for all its members separately. | Union allocates one common storage space for all its members. Union finds that which of its member needs high storage space over other members and allocates that much space |
| 2 | Structure occupies higher memory space. | Union occupies lower memory space over structure. |
| 3 | We can access all members of structure at a time. | We can access only one member of union at a time. |
| 4 | Structure example: struct student { int mark; char name[6]; double average; }; | Union example: union student { int mark; char name[6]; double average; }; |
| 5 | For above structure, memory allocation will be like below. int mark – 2B char name[6] – 6B double average – 8B  Total memory allocation = 2+6+8 = 16 Bytes | For above union, only 8 bytes of memory will be allocated since double data type will occupy maximum space of memory over other data types.  Total memory allocation = 8 Bytes |

**Tutorial Questions and Solutions**

1. **Create a structure for the following records :**
2. **Information about particular date**

struct date

{

int day;

int month;

int year;

};

1. **A particular book**

struct book

{

char title[20];

char author[20];

int no\_pages;

float price;

int year\_pub;

};

1. **Write a C program to create a structure called student with the following members id, name and percentage and display the contents of the record.**

#include <stdio.h>

#include <string.h>

struct student

{

int id;

char name[20];

float percentage;

} record;

int main()

{ record.id=1;

strcpy(record.name, "Raju");

record.percentage = 86.5;

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

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

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

return 0;

}

**Output:**

|  |
| --- |
| Id is: 1 Name is: Raju Percentage is: 86.500000 |

1. **Write a C program for illustrating the declaration of many structure variables**

In this program, two structure variables “record1″ and “record2″ are declared for same structure and different values are assigned for both structure variables. Separate memory is allocated for both structure variables to store the data.

#include <stdio.h>

#include <string.h>

struct student

{

int id;

char name[30];

float percentage;

};

int main()

{

int i;

struct student record1 = {1, "Raju", 90.5};

struct student record2 = {2, "Mani", 93.5};

printf("Records of STUDENT1: \n");

printf(" Id is: %d \n", record1.id);

printf(" Name is: %s \n", record1.name);

printf(" Percentage is: %f \n\n", record1.percentage);

printf("Records of STUDENT2: \n");

printf(" Id is: %d \n", record2.id);

printf(" Name is: %s \n", record2.name);

printf(" Percentage is: %f \n\n", record2.percentage);

return 0;

}

**Output:**

|  |
| --- |
| **Records of STUDENT1:** Id is: 1 Name is: Raju Percentage is: 90.500000  **Records of STUDENT2:** Id is: 2 Name is: Mani Percentage is: 93.500000 |

#### **Write a C program for illustrating C typedef keyword :**

// Structure using typedef:

#include <stdio.h>

#include <string.h>

typedef struct student

{

int id;

char name[20];

float percentage;

} status;

int main()

{

status record;

record.id=1;

strcpy(record.name, "Raju");

record.percentage = 86.5;

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

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

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

return 0;

}

#### **Output:**

|  |
| --- |
| Id is: 1 Name is: Raju Percentage is: 86.500000 |

## Write a C program to Add Two distances using structures.

#include <stdio.h>

struct Distance

{

int feet;

float inch;

} d1,d2,sum;

int main()

{

printf("Enter information for 1st distance\n");

printf("Enter feet: ");

scanf("%d",&d1.feet);

printf("Enter inch: ");

scanf("%f",&d1.inch);

printf("\nEnter information for 2nd distance\n");

printf("Enter feet: ");

scanf("%d",&d2.feet);

printf("Enter inch: ");

scanf("%f",&d2.inch);

sum.feet=d1.feet+d2.feet;

sum.inch=d1.inch+d2.inch;

/\* If inch is greater than 12, changing it to feet. \*/

if (sum.inch>12.0)

{

sum.inch=sum.inch-12.0;

++sum.feet;

}

printf("\nSum of distances=%d\'-%.1f\"",sum.feet,sum.inch);

return 0;

}

## Write a C program to show how structures can be passed as Function Arguments

#include <stdio.h>

#include <string.h>

struct Books

{

char title[50];

char author[50];

char subject[100];

int book\_id;

};

/\* function declaration \*/

void printBook( struct Books book );

int main( )

{

struct Books Book1; /\* Declare Book1 of type Book \*/

struct Books Book2; /\* Declare Book2 of type Book \*/

/\* book 1 specification \*/

strcpy( Book1.title, "C Programming");

strcpy( Book1.author, "Nuha Ali");

strcpy( Book1.subject, "C Programming Tutorial");

Book1.book\_id = 6495407;

/\* book 2 specification \*/

strcpy( Book2.title, "Telecom Billing");

strcpy( Book2.author, "Zara Ali");

strcpy( Book2.subject, "Telecom Billing Tutorial");

Book2.book\_id = 6495700;

/\* print Book1 info \*/

printBook( Book1 );

/\* Print Book2 info \*/

printBook( Book2 );

return 0;

}

void printBook( struct Books book )

{

printf( "Book title : %s\n", book.title);

printf( "Book author : %s\n", book.author);

printf( "Book subject : %s\n", book.subject);

printf( "Book book\_id : %d\n", book.book\_id);

}

When the above code is compiled and executed, it produces the following result:

Book title : C Programming

Book author : Nuha Ali

Book subject : C Programming Tutorial

Book book\_id : 6495407

Book title : Telecom Billing

Book author : Zara Ali

Book subject : Telecom Billing Tutorial

Book book\_id : 6495700

## Write a C program to Add Two Complex Numbers using the concept of passing structure as argument to function

#include <stdio.h>

typedef struct complex{

float real;

float imag;

}complex;

complex add(complex n1,complex n2);

int main()

{

complex n1,n2,temp;

printf("For 1st complex number \n");

printf("Enter real and imaginary respectively:\n");

scanf("%f%f",&n1.real,&n1.imag);

printf("\nFor 2nd complex number \n");

printf("Enter real and imaginary respectively:\n");

scanf("%f%f",&n2.real,&n2.imag);

temp=add(n1,n2);

printf("Sum=%.1f+%.1fi",temp.real,temp.imag);

return 0;

}

complex add(complex n1,complex n2){

complex temp;

temp.real=n1.real+n2.real;

temp.imag=n1.imag+n2.imag;

return(temp);

1. **Create a structure called student with the following members student name, roll-no, marks in three tests. Write a C program to create N records and**

**Search on roll-no and display all the records**

**Average marks in each test**

**Highest in each test.**

#include<stdio.h>

#include<string.h>

**//Maintains student name, rollno and marks in 3 tests**

struct student

{

char name[30];

char rollno[20];

int marks[3];

};

typedef struct student stud;

**//Calls the main function to be performed**

void main()

{

stud a[20];

int i,n,ch,high1,high2,high3;

float avg1,avg2,avg3;

char rno[20];

printf(" \*\*\*\*\*\* STUDENTS RECORDS \*\*\*\*\*\* ");

printf("\n\n");

printf("\n Enter the number of students: ");

scanf("%d",&n);

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

{

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

printf("\n Enter name: ");

scanf("%s",&a[i].name);

printf("\n Enter rollno: ");

scanf("%s",&a[i].rollno);

printf("\n Enter marks1: ");

scanf("%d",&a[i].marks[0]);

printf("\n Enter marks2: ");

scanf("%d",&a[i].marks[1]);

printf("\n Enter marks3: ");

scanf("%d",&a[i].marks[2]);

}

printf("\n\n");

printf("\n 1-SEARCH\n");

printf("\n 2-AVERAGE\n");

printf("\n 3-HIGHEST\n");

printf("\n Enter your choice: ");

scanf("%d",&ch);

switch(ch)

{

case 1: **//Function to search rollno & display all records**

printf("\n Enter rollno to be searched: ");

scanf("%s",&rno);

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

if(strcmp(a[i].rollno,rno)==0)

{

printf("\n SEARCH SUCCESSFUL: \n");

printf("\n Details of student is: \n");

printf("\nNAME ROLLNO MARKS1 MARKS2 MARKS3: \n");

printf("\n%s\t%s\t%d\t%d\t%d\t\n",a[i].name,a[i].rollno,a[i].marks[0],a[i].marks[1],

a[i].marks[2]);

}

printf("\n SEARCH FAILED! \n");

break;

case 2: **//Function to find average marks in each test**

avg1=avg2=avg3=0;

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

{

avg1=avg1+a[i].marks[0];

avg2=avg2+a[i].marks[1];

avg3=avg3+a[i].marks[2];

}

printf("\n Average of test1= %f\n",avg1/n);

printf("\n Average of test2= %f\n",avg2/n);

printf("\n Average of test3= %f\n",avg3/n);

break;

case 3: **//Function to find highest marks in each test**

high1=a[0].marks[0];

high2=a[0].marks[1];

high3=a[0].marks[2];

for(i=1;i<n;i++)

{

if(a[i].marks[0]>high1)

high1=a[i].marks[0];

if(a[i].marks[1]>high2)

high1=a[i].marks[1];

if(a[i].marks[2]>high3)

high1=a[i].marks[2];

}

printf("\n Highest marks in test1= %d\n", high1);

printf("\n Highest marks in test2= %d\n", high2);

printf("\n Highest marks in test3= %d\n", high3);

break;

default: **//erroneous input**

printf("\n Invalid choice! \n");

break;

}

}

1. **Create a structure called student with the following members’ student name, roll- no, marks in three tests. Write a C program to create N records and sort them using bubble sort and display sorted records.**

***Program***

#include<stdio.h>

#include<string.h>

void main()

{

**// Maintains student name, rollno and marks in 3 tests**

struct student

{

char name[30];

char rollno[16];

int marks[3];

};

typedef struct student stud;

stud a[20],temp;

int i,j,n;

printf("\n \*\*\*\*\*\* STUDENTS RECORDS \*\*\*\*\*\* \n");

printf("\n Enter the number of students: ");

scanf("%d",&n);

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

{

printf("\n\n Details of %d Students: \n",i+1);

printf("\n Enter name: ");

scanf("%s",&a[i].name);

printf("\n Enter rollno: ");

scanf("%s",&a[i].rollno);

printf("\n Enter marks1: ");

scanf("%d",&a[i].marks[0]);

printf("\n Enter marks2: ");

scanf("%d",&a[i].marks[1]);

printf("\n Enter marks3: ");

scanf("%d",&a[i].marks[2]);

}

**// Function to sort according to rollno**

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

{

for(j=i+1; j<n; j++)

{

if(strcmp(a[i].rollno, a[j].rollno)>0)

{

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}

}

printf("\n\nSorted list according to rollno is: \n");

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

{

printf("\nNAME ROLLNO MARKS1 MARKS2 MARKS3: \n");

printf("\n%s\t%s\t%d\t%d\t%d\t\n",a[i].name,a[i].rollno,a[i].marks[0],a[i].marks[1],

a[i].marks[2]);

}

}

1. **Write a C program where, user is asked to enter two time periods and these two periods are stored in structure variables. This program calculates the difference between these two time period. To perform this task, a function is created which calculates the difference and the result is displayed in main() function without returning it (Using call by reference technique)**

#include <stdio.h>

struct TIME{

int seconds;

int minutes;

int hours;

};

void Difference(struct TIME t1, struct TIME t2, struct TIME \*diff);

int main()

{

struct TIME t1,t2,diff;

printf("Enter start time: \n");

printf("Enter hours, minutes and seconds respectively: ");

scanf("%d%d%d",&t1.hours,&t1.minutes,&t1.seconds);

printf("Enter stop time: \n");

printf("Enter hours, minutes and seconds respectively: ");

scanf("%d%d%d",&t2.hours,&t2.minutes,&t2.seconds);

Difference(t1,t2,&diff);

printf("\nTIME DIFFERENCE: %d:%d:%d - ",t1.hours,t1.minutes,t1.seconds);

printf("%d:%d:%d ",t2.hours,t2.minutes,t2.seconds);

printf("= %d:%d:%d\n",diff.hours,diff.minutes,diff.seconds);

return 0;

}

void Difference(struct TIME t1, struct TIME t2, struct TIME \*differ)

{

if(t2.seconds>t1.seconds){

--t1.minutes;

t1.seconds+=60;

}

differ->seconds=t1.seconds-t2.seconds;

if(t2.minutes>t1.minutes){

--t1.hours;

t1.minutes+=60;

}

differ->minutes=t1.minutes-t2.minutes;

differ->hours=t1.hours-t2.hours;

}

1. **Write a C program to add two distances(feet-inch system) entered by user. To solve this program, make a structure. Pass two structure variable (containing distance in feet and inch) to add function by reference and display the result in main function without returning it.**

#include <stdio.h>

struct distance{

int feet;

float inch;

};

void Add(struct distance d1,struct distance d2, struct distance \*d3);

int main()

{

struct distance dist1, dist2, dist3;

printf("First distance\n");

printf("Enter feet: ");

scanf("%d",&dist1.feet);

printf("Enter inch: ");

scanf("%f",&dist1.inch);

printf("Second distance\n");

printf("Enter feet: ");

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

printf("Enter inch: ");

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

Add(dist1, dist2, &dist3);

/\*passing structure variables dist1 and dist2 by value whereas passing structure variable dist3 by reference \*/

printf("\nSum of distances = %d\'-%.1f\"",dist3.feet, dist3.inch);

return 0;

}

void Add(struct distance d1,struct distance d2, struct distance \*d3)

{

/\* Adding distances d1 and d2 and storing it in d3 \*/

d3->feet=d1.feet+d2.feet;

d3->inch=d1.inch+d2.inch;

if (d3->inch>=12) { /\* if inch is greater or equal to 12, converting it to feet. \*/

d3->inch-=12;

++d3->feet;

}

}

**Output**

First distance

Enter feet: 12

Enter inch: 6.8

Second distance

Enter feet: 5

Enter inch: 7.5

Sum of distances = 18'-2.3"

1. **Write a C program to display total memory size occupied by the union declared.**

#include <stdio.h>

#include <string.h>

union Data

{

int i;

float f;

char str[20];

};

int main( )

{

union Data data;

printf( "Memory size occupied by data : %d\n", sizeof(data));

return 0;

}

When the above code is compiled and executed, it produces the following result:

Memory size occupied by data : 20

1. **Write a C code for real time Bank application to perform all the below operations:**
2. Creating new account – To create a new account
3. Cash Deposit – To Deposit some amount in newly created account
4. Cash withdrawal - To Withdraw some amount from your account
5. Display Account information – It will display all informations of the existing accounts
6. Log out
7. Clearing the output screen and display available options

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

// Structure declaration

struct acc\_type

{

char bank\_name[20];

char bank\_branch[20];

char acc\_holder\_name[30];

int acc\_number;

char acc\_holder\_address[100];

float available\_balance;

};

struct acc\_type account[20];

/\*

printf("The above structure can be declared using

typedef like below");

typedef struct acc\_type

{

char bank\_name[20];

char bank\_branch[20];

char acc\_holder\_name[30];

int acc\_number;

char acc\_holder\_address[100];

float available\_balance;

}Acc\_detail;

Acc\_detail account[20];

\*/

int num\_acc;

void Create\_new\_account();

void Cash\_Deposit();

void Cash\_withdrawl();

void Account\_information();

void Log\_out();

void display\_options();

/\* main program \*/

int main()

{

char option;.

char f2f[50] = "http://global.com/";

num\_acc=0;

while(1)

{

printf("\n\*\*\*\*\* Welcome to Bank Application \*\*\*\*\*\n");

printf("\nThis demo program is brought you by %s",f2f);

display\_options();

printf("Please enter any options (1/2/3/4/5/6) ");

printf("to continue : ");

option = getch();

printf("%c \n", option);

switch(option)

{

case '1': Create\_new\_account();

break;

case '2': Cash\_Deposit();

break;

case '3': Cash\_withdrawl();

break;

case '4': Account\_information();

break;

case '5': return 0;

case '6': system("cls");

break;

default : system("cls");

printf("Please enter one of the options");

printf("(1/2/3/4/5/6) to continue \n ");

break;

}

}

return 0;

}

/\*Function to display available options in this application\*/

void display\_options()

{

printf("\n1. Create new account \n");

printf("2. Cash Deposit \n");

printf("3. Cash withdrawl \n");

printf("4. Account information \n");

printf("5. Log out \n");

printf("6. Clear the screen and display available ");

printf("options \n\n");

}

/\* Function to create new account \*/

void Create\_new\_account()

{

char bank\_name[20];

char bank\_branch[20];

char acc\_holder\_name[30];

int acc\_number;

char acc\_holder\_address[100];

float available\_balance = 0;

fflush(stdin);

printf("\nEnter the bank name : ");

scanf("%s", &bank\_name);

printf("\nEnter the bank branch : ");

scanf("%s", &bank\_branch);

printf("\nEnter the account holder name : ");

scanf("%s", &acc\_holder\_name);

printf("\nEnter the account number(1 to 10): ");

scanf("%d", &acc\_number);

printf("\nEnter the account holder address : ");

scanf("%s", &acc\_holder\_address);

strcpy(account[acc\_number-1].bank\_name,bank\_name);

strcpy(account[acc\_number-1].bank\_branch,bank\_branch);

strcpy(account[acc\_number-1].acc\_holder\_name,

acc\_holder\_name);

account[acc\_number-1].acc\_number=acc\_number;

strcpy(account[acc\_number-1].acc\_holder\_address,

acc\_holder\_address);

account[acc\_number-1].available\_balance=available\_balance;

printf("\nAccount has been created successfully \n\n");

printf("Bank name : %s \n" ,

account[acc\_number-1].bank\_name);

printf("Bank branch : %s \n" ,

account[acc\_number-1].bank\_branch);

printf("Account holder name : %s \n" ,

account[acc\_number-1].acc\_holder\_name);

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

account[acc\_number-1].acc\_number);

printf("Account holder address : %s \n" ,

account[acc\_number-1].acc\_holder\_address);

printf("Available balance : %f \n" ,

account[acc\_number-1].available\_balance);

//num\_acc++;

}

// Displaying account informations

void Account\_information()

{

register int num\_acc = 0;

//if (!strcmp(customer,account[count].name))

while(strlen(account[num\_acc].bank\_name)>0)

{

printf("\nBank name : %s \n" ,

account[num\_acc].bank\_name);

printf("Bank branch : %s \n" ,

account[num\_acc].bank\_branch);

printf("Account holder name : %s \n" ,

account[num\_acc].acc\_holder\_name);

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

account[num\_acc].acc\_number);

printf("Account holder address : %s \n" ,

account[num\_acc].acc\_holder\_address);

printf("Available balance : %f \n\n" ,

account[num\_acc].available\_balance);

num\_acc++;

}

}

// Function to deposit amount in an account

void Cash\_Deposit()

{

auto int acc\_no;

float add\_money;

printf("Enter account number you want to deposit money:");

scanf("%d",&acc\_no);

printf("\nThe current balance for account %d is %f \n",

acc\_no, account[acc\_no-1].available\_balance);

printf("\nEnter money you want to deposit : ");

scanf("%f",&add\_money);

while (acc\_no=account[acc\_no-1].acc\_number)

{

account[acc\_no-1].available\_balance=

account[acc\_no-1].available\_balance+add\_money;

printf("\nThe New balance for account %d is %f \n",

acc\_no, account[acc\_no-1].available\_balance);

break;

}acc\_no++;

}

// Function to withdraw amount from an account

void Cash\_withdrawl()

{

auto int acc\_no;

float withdraw\_money;

printf("Enter account number you want to withdraw money:");

scanf("%d",&acc\_no);

printf("\nThe current balance for account %d is %f \n",

acc\_no, account[acc\_no-1].available\_balance);

printf("\nEnter money you want to withdraw from account ");

scanf("%f",&withdraw\_money);

while (acc\_no=account[acc\_no-1].acc\_number)

{

account[acc\_no-1].available\_balance=

account[acc\_no-1].available\_balance-withdraw\_money;

printf("\nThe New balance for account %d is %f \n",

acc\_no, account[acc\_no-1].available\_balance);

break;

}acc\_no++;

}

**Sample Questions**

1. What is a structure? How does a structure differ from an array?
2. Give the general format of structure definition and explain with example.
3. What is a member? What is the relationship between a member and a structure?
4. How can structure variables be defined? How do structure variable declarations differ from structure type declaration?
5. How is a structure member accessed? How can a structure member be processed?
6. How are the members of structure variable assigned initial values?
7. How will you compare structure variables? Explain.
8. Explain arrays of structures with an example C code.
9. Can be nest structure within a structure? Explain with C code.
10. What is a union? How does it differ from structure?
11. For what kinds of applications are unions useful?
12. How a member of a union variable is assigned an initial value? In what way does the initialization of a union variable differ from the initialization of a structure variable?
13. Give the difference between structures and unions.