PROBLEM SOLVING THROUGH PROGRAMMING (CS1001-2)

Unit-3

- **Structures and Union**
- **Pointers**
- **File handling.**

Structures

Defining a structure:

Structure is a user-defined datatype in **C language** which allows us to combine data of different types together. **Structure** helps to construct a complex data type which is more meaningful. It is somewhat similar to an Array, but an array holds data of similar type only. In **structure**, data is stored in form of records.

Lets say we need to store the data of students like student name, age, address, id etc.
One way of doing this would be creating a different variable for each attribute, however when you need to store the data of multiple students then in that case, you would need to create these several variables again for each student. This is such a big headache to store data in this way.
We can solve this problem easily by using structure. We can create a structure that has members for name, id, address and age and then we can create the variables of this structure for each student.
struct keyword is used to define a structure. struct defines a new data type which is a collection of primary and derived datatypes.

Syntax of defining structure:

```
struct [structure_tag]
{
        //member variable 1
        //member variable 2
        //member variable 3
        ...
}[structure_variables];
```

As you can see in the syntax above, we start with the struct keyword, then it's optional to provide your structure a name, we suggest you to give it a name, then inside the curly braces, we have to mention all the member variables, which are nothing but normal C language variables of different types like int, float, array etc.
After the closing curly brace, we can specify one or more structure variables, again thisis optional.

Note: The closing curly brace in the structure type declaration must be followed by a semicolon(;).

Example of Structure

```
struct Student
{
        char    name[25];
        int age;
        char branch[10];
        char gender; // F for female and M for male
};
```

Here struct Student declares a structure to hold the details of a student which consists of 4 data fields, namely name, age, branch and gender. These fields are called **structure elements or members**.

Each member can have different datatype, like in this case, name is an array of char type and age is of int type etc. **Student** is the name of the structure and is called as the **structure tag**.

Declaring structure variables:

It is possible to declare variables of a **structure**, either along with structure definition orafter the structure is defined. **Structure** variable declaration is similar to the declaration of any normal variable of any other datatype.

Structure variables can be declared in following two ways:

1) Declaring Structure variables separately

```
struct Student
{
        char name[25];int age;
        char branch[10];
        char gender; //F for female and M for male
};

struct Student S1, S2;      //declaring variables of struct Student
```

2) Declaring Structure variables with structure definition

```
struct Student
{
        char   name[25];
        int age;
        char branch[10];
        char gender; //F for female and M for male
}S1, S2;
```

Here S1and S2 are variables of structure Student. However this approach is not much recommended.

ACCESSING STRUCTURE MEMBERS:

Structure members can be accessed and assigned values in a number of ways. Structure members have no meaning individually without the structure. In order to assign a value to any structure member, the member name must be linked with the **structure** variable using a dot(.) operator also called **period** or **member access** operator.

For example:

```
#include<stdio.h>
#include<string.h>

struct Student
{
        char name[25];
        int age;
        char branch[10];
        char gender; //F for female and M for male
};

int main()
{
        struct Student s1;
                /*
                s1 is a variable of Student type and
                age is amember of Student
        */
        s1.age = 18;
        /*
        using  string  function  to  add
        name*/
        strcpy(s1.name, "Viraaj");

        /*
                displaying the stored values
        */
        printf("Name    of    Student    1:    %s\n",    s1.name);
        printf("Age of Student 1: %d\n", s1.age);
        return 0;
}
```

Output:

```
Name of Student 1:
Viraaj
Age of Student 1:
18
```

Copying and comparing structured variable:

Two variables of the same structure type can be copied the same way as ordinary variables. If person1 and person2 belong to the same structure, then the following statements are valid.

```
person1 = person2;
person2 = person1;
```

C does not permit any logical operators on structure variables In case, we need to compare them, we may do so by comparing members individually.

```
person1 = = person2 (Equality)
person1! = person2 (Not Equal)
```

Statements are not permitted.

Ex: Program for comparison of structure variables

```
struct class
{
   int no;
   char name [20];
   float marks;
};

int main ( )
{
  int x;
  struct class stu1 = {111, “Rao”, 72.50};
  struct class stu2 = {222,“Reddy”,67.80};
  struct class stu3;
  stu3 = stu2;

  x = ( ( stu3.no= = stu2.no) && ( stu3.marks = = stu2.marks))?1:0;
  if ( x==1)
  {
     printf (“ \n student 2 & student 3 are same \n”);
     printf (“%d\t%s\t%f  “ stu3.no, stu3.name, stu3.marks);
  }
  else
     printf ( “\n student 2 and student 3 are different “);
  }
```

Array of structures:

- It is possible to store a structure has an array element. i.e., an array in which each element is a structure.
- Just as arrays of any basic type of variable are allowed, so are arrays of a given type of structure.
- Although a structure contains many different types, the compiler never gets to know this information because it is hidden away inside a sealed structure capsule, so it can believe that all the elements in the array have the same type, even though that type is itself made up of lots of different types.

```
struct struct_name {
 type element 1;
 type element 2;
 ……………..
 type element n;
}array name[size];
```

Example:

```
struct student {
 int rollno;
char name[25];
float totalmark;
} stud[100];
```

In this declaration stud is a 100-element array of structures. Hence, each element of stud is a separate structure of type student. An array of structure can be assigned initial values just as any other array. So the above structure can hold information of 100 students.

Program to demonstrate use of array of structure:

```
 #include <stdio.h>
void main()
{
struct student {
   int rollno;
   char name[25];
int totalmark;
}stud[100];
int n,i;
printf("Enter total number of students\n\n");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("Enter details of %d-th student\n",i+1);
printf("Name:\n");
scanf("%s",&stud[i].name);
printf("Roll number:\n");
scanf("%d",&stud[i].rollno);
printf("Total mark:\n");
```

```
      scanf("%d",&stud[i].totalmark);
      }

   printf("STUDENTS DETAILS:\n");
   for(i=0;i<n;i++)        {
    printf("\nRoll number:%d\n",stud[i].rollno);
 printf("Name:%s\n",stud[i].name);
 printf("Total mark:%d\n",stud[i].totalmark);
 }
 }
```

```
  OUTPUT :
 Enter total number of students: 3
 Enter details of 1-th student
 Name:SUBAHAS
 Roll number:11
 Total mark:589
 Enter details of 2-th student
 Name:POOJA
 Roll number:12
 Total mark:594
 Enter details of 3-th student
 Name:AKASH
 Roll number:13
 Total mark:595

 STUDENTS DETAILS:
 Roll number:11
 Name: SUBAHAS
 Total mark:589
 Roll number:12
 Name: POOJA
 Total mark:594
 Roll number:13
 Name: AKASH
 Total mark:595
```

Nested Structure:

In C, a structure is defined using the struct keyword, followed by the structure name, and then the structure members enclosed within curly braces.

Syntax of Nested Structure in C:

```
 struct outer_struct {
    // outer structure members
    int outer_member1;
    float outer_member2;
```

```
  // nested structure definition
  struct inner_struct {
    // inner structure members
    int inner_member1;
    float inner_member2;
  } inner;
};
```

Example of Nested Structure:

```
struct school{
  int  numberOfStudents;
  int numberOfTeachers;
  struct student{
      char name[50];
      int class;
      int roll_Number;
   } std;
};
```

In the above example of nested structure in C, there are two structures Student (depended structure) and another structure called School(Outer structure). The structure School has the data members like numberOfStudents, numberOfTeachers, etc. and the Student structure is nested inside the structure School and it has the data members like name, class, roll_Number, etc.

To access the members of the school structure, you use dot notation. For example, to access the numberOfStudents member of the school structure, you would use the following syntax:

```
struct school s;
s.numberOfStudents = 100;
```

To access the members of the nested student structure, you would use the following syntax:

```
strcpy(s.std.name, "John Smith");
s.std.class = 10;
s.std.roll_Number = 1;
```

Note that the dot notation is used to access the members of the std structure, which is a member of the school structure.
The nested structure student contains three members: name, class, and roll_Number. This structure can be useful in situations where you need to store information about a student, such as their name, class, and roll number.

Simple Programs on Structures

1. Store Information and Display it Using Structure

```
#include <stdio.h>

struct student
{
        char name[50];
        int roll;
        float marks;
} s;

int main()
{
        printf("Enter information:\n");
        printf("Enter name: ");
        scanf("%s", s.name);
        printf("Enter roll number: ");
        scanf("%d", &s.roll);
        printf("Enter marks: ");
        scanf("%f", &s.marks);

        printf("Displaying Information:\n");
        printf("Name: ");
        puts(s.name);
        printf("Roll number: %d\n",s.roll);
        printf("Marks: %.1f\n", s.marks);
        return 0;
}
```

2. Program to add two distances in inch-feet system

```
#include <stdio.h>
struct Distance
{
        int feet;
        float inch;
} d1, d2, sumOfDistances;

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);

	sumOfDistances.feet = d1.feet+d2.feet;
	sumOfDistances.inch = d1.inch+d2.inch;

	// If inch is greater than 12, changing it to feet.
	if (sumOfDistances.inch>12.0)
	{
		sumOfDistances.inch = sumOfDistances.inch-12.0;
		++sumOfDistances.feet;
	}
	printf("\nSum of distances = %d\'-%.1f\"",sumOfDistances.feet, sumOfDistances.inch);
	return 0;
}
```

Output:

```
Enter 1st distance
Enter feet: 23
Enter inch: 8.6

Enter  2nd  distance
Enter feet: 34
Enter inch: 2.4

Sum of distances = 57'-11.0"
```

Pointers

Introduction:

A pointer is a derived data type in C. It is built from fundamental data types available in C. Pointers contain memory addresses as their values. Since these memory addresses are locations in the computer memory where program instructions are stored, pointers can be used to access and manipulate data stored in memory.

Understanding pointers:

A pointer is a variable that can hold the address of another variable or address of memory location.

The following are some of the benefits of using pointers in C:

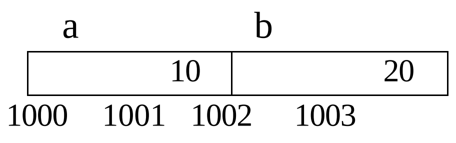
1. Pointers provide direct access to memory
2. Pointers are used to access the value of the variable through its address
3. Pointers provide a way to return more than one value to functions
4. Addresses of objects can be extracted using pointers
5. Reduces the storage space and complexity of program
6. Reduces the execution time of program .Execution is faster

A **pointer** is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, we must declare a pointer before using it to storeany variable address.

Accessing the address of a variable:

In C every variable is a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator or address operator, which denotes an address in memory.

Consider two variables a and b.
The declaration of them is shown below:
int a=10;int b=20;



The value of a and b is 10 and 20.

The compiler allocates two bytes of memory starting at address 1000 and stores the value 10 at that location and gives ‘**a**’ as name of that location. Similarly for second variable ‘**b**’ as shown in above figure.

In order to access address, operator &(ampersand) is used. &a is used to get address of variable a . &b is used to get address of variable b.

The address of variable a is 1000 and variable b has address 1002.

In order to access values using address one must use dereferencing operator or pointer denotedby symbol *. That is we can access the value stored in that address as shown below:

```
printf("\n value of a=%d\n",*&a);
printf("\n value of b=%d\n",*&b);
```

The value of a=10 and the value of b=20.

Consider the following example, which prints the address of the variables defined:

```
#include <stdio.h>
int main ()
{
  int a=100;int
  b=450;
  printf("value of  a=%d\n", a  );
  printf("value of b=%d\n", b );
  printf("Address of  variable a=%d\n", &a);
  printf("Address of variable b= %d\n", &b);
  return 0;
}
```

When the above code is compiled and executed, it produces the following result −value of a=100

```
value of  b=450
Address of variable a= bff5a400
Address of variable= bff5a3f6
```

Declaring pointer variables:

The general form of a pointer variable declaration is

type *var-name;

Here, **type** is the pointer's base type; it must be a valid C data type and **var-name** is the name ofthe pointer variable.

The asterisk * used to declare a pointer is the same asterisk used for multiplication. However, inthis statement the asterisk is being used to designate a variable as a pointer.

Take a look at some of the valid pointer declarations −

```
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch     /* pointer to a character */
```

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

Steps to be followed to use pointers:

• Declare a data variable	Ex: int x;
• Declare a pointer variable	Ex: int *p;
• Initialize a pointer variable	Ex: p=&x;
• Access data using pointer variable	Ex: y=*p

There are a few important operations, which we will do with the help of pointers very frequently.
(a) We define a pointer variable,
(b) assign the address of a variable to a pointer and
(c) finally access the value at the address available in the pointer variable. This is done byusing unary operator ***** that returns the value of the variable located at the address specified by itsoperand.

Simple programs on pointers:

1) This program declares and initializes a variable and prints the address of variable and illustrates the important operations done with the help of pointers, that is **:**

(a) We define a pointer variable,
(b) assign the address of a variable to a pointer and
(c) finally access the value at the address available in the pointer variable.

```
 #include <stdio.h>
int main ()
{
  int  var = 20;  /* actual variable declaration */

  int  *ip;        /* pointer variable declaration */

  ip = &var;      /* store address of var in pointer variable*/

  printf("Address of var variable: %x\n", &var  );

  /* address stored in pointer variable */

  printf("Address stored in ip variable: %x\n", ip );

   /* access the value using the pointer */

  printf("Value of *ip variable: %d\n", *ip );

  return 0;

}
```

Output:

```
Address of var variable: bffd8b3c
Address  stored  in  ip variable: bffd8b3c
Value of  *ip variable: 2
```

2) Write a C program to add two numbers using pointers

```
#include <stdio.h>
int main ()
{
        int a=10,b=20,sum;int *p1, *p2;
        p1=&a;
        p2=&b;
        sum=*p1+*p2;
        printf("\n Sum=%d",sum);
}
```

Output:

```
Sum=30
```

File handling

A **file** represents a sequence of bytes on the disk where a group of related data is stored. File iscreated for permanent storage of data.

In C language, we use a structure **pointer of file type** to declare a file.

FILE ***fp**;

C provides a number of functions that helps to perform basic file operations. Following are the functions:

Function	**Description**
fopen()	create a new file or open a existing file
fclose()	closes a file
getc()	reads a character from a file
putc()	writes a character to a file
fscanf()	reads a set of data from a file
fprintf()	writes a set of data to a file
getw()	reads a integer from a file
putw()	writes a integer to a file
fseek()	set the position to desire point
ftell()	gives current position in the file
rewind()	set the position to the beginning point

OPENING A FILE OR CREATING A FILE:

- The fopen() function is used to create a new file or to open an existing file.
- We must open a file before it can be read, write, or update.

General Syntax :

fp = FILE *fopen(const char ****filename, const char ****mode***);**

The fopen() function accepts two parameters:

- The file name (string).
  Here **filename** is the name of the file to be opened If the file is stored at some specific location, then we must mention the path at which the file is stored. For example, a file name can be like **"c://some_folder/some_file.ext"**.
- The mode in which the file is to be opened. It is a string.

***fp** is the FILE pointer (FILE *fp), which will hold the reference to the opened (or created) file.

Mode can be of following types:

mode	**description**
r	opens a text file in reading mode
w	opens or create a text file in writing mode.
a	opens a text file in append mode
r+	opens a text file in both reading and writing mode
w+	opens a text file in both reading and writing mode
a+	opens a text file in both reading and writing mode
rb	opens a binary file in reading mode
wb	opens or create a binary file in writing mode
ab	opens a binary file in append mode
rb+	opens a binary file in both reading and writing mode
wb+	opens a binary file in both reading and writing mode
ab+	opens a binary file in both reading and writing mode

CLOSING A FILE:

The fclose() function is used to close an already opened file.

General Syntax :

int fclose(FILE **fp*);

- Here fclose() function closes the file and returns zero on success, or EOF if there is an errorin closing the file. This EOF is a constant defined in the header file stdio.h.

READING FROM A FILE:

The file read operations can be performed using functions fscanf or fgets. Both the functions performed the same operations as that of scanf and gets but with an additional parameter, the file pointer. So, it depends on you if you want to read the file line by line or character by character.

The fscanf() function is used to read set of characters from file. It reads a word from the file andreturns EOF at the end of file.

Syntax:

int fscanf(FILE *stream, const char *format [, argument, ...]);

And the code snippet for reading a file is as:

```
 FILE * filePointer;
filePointer = fopen(“fileName.txt”, “r”);
fscanf(filePointer, "%s %s %s %d", str1, str2, str3, &year);
```

WRITING TO A FILE:

The file write operations can be perfomed by the functions fprintf and fputs with similarities toread operations.

The fprintf() function is used to write set of characters into file. It sends formatted output to astream.

Syntax:

int fprintf(FILE *stream, const char *format [, argument, ...]);

The code for writing to a file is as :

```
FILE *filePointer ;
filePointer = fopen(“fileName.txt”, “w”);
fprintf(filePointer, "%s %s %s %d", "We", "are", "in", 2012);
```

INPUT/OUTPUT OPERATION ON FILE:

getc() and putc() are simplest functions used to read and write individual characters toa file.

```
 #include<stdio.h>
 int main( )

   FILE *fp;char ch;
   fp = fopen("one.txt", "w");  .
   printf("Enter data");
   while( (ch = getchar()) != EOF)
    {
         putc(ch,fp);
   }
   fclose(fp);
   fp = fopen("one.txt", "r");
   while( (ch = getc()) != EOF)
          printf("
   %c",ch);
   fclose(fp);
}
```

READING AND WRITING FROM FILE USING fprintf() and fscanf():

```
#include<stdio.h>
struct  emp
{
       char name[10];
       int age;
};

int main()
{
       struct emp e;
       FILE *fp;
       fp=fopen("employee.txt","w");
       printf("\nEnter  Name  and  Age\n");
       scanf("%s %d", e.name, &e.age);
       fprintf(fp,"%s  %d",  e.name,  e.age);
       fclose(fp);
       fp= fopen("employee.txt", "r");
       do
       {
              fscanf(fp,"%s %d", e.name, e.age);
              printf("%s %d", e.name, e.age);
       }while( !feof(fp) );
}
```

Here in the program **fprintf()** function directly writes into the file, while **fscanf()** reads from the file, which can then be printed on console using standard **printf()** function. Here feof function specifies end of file .

The program creates a new empty file called employee.txt with write mode.
Enter name and age
Raju 40

The program writes a data into the file called employee.txt

The output in the file employee.txt will be:
Raju 40
The output on the display screen will be:
Raju 40