C Programming Structure and Pointer

Pointers can be accessed along with structures. A pointer variable of structure can be created as below:

```
struct name {

    member1;

    member2;

    .

    .

};

-------- Inside function -------

struct name *ptr;
```

Here, the pointer variable of type **struct name** is created.

Structure's member through pointer can be used in two ways:

1. Referencing pointer to another address to access memory
2. Using dynamic memory allocation

Consider an example to access structure's member through pointer.

```
#include <stdio.h>
struct name{
  int a;
  float b;
};
int main(){
   struct name *ptr,p;
   ptr=&p;          /* Referencing pointer to memory address of p */
   printf("Enter integer: ");
   scanf("%d",&(*ptr).a);
```

```
    printf("Enter number: ");
    scanf("%f",&(*ptr).b);
    printf("Displaying: ");
    printf("%d%f",(*ptr).a,(*ptr).b);
    return 0;
}
```

In this example, the pointer variable of type **`struct name`** is referenced to the address of*p*. Then, only the structure member through pointer can can accessed.

Structure pointer member can also be accessed using `->` operator.

```
(*ptr).a is same as ptr->a

(*ptr).b is same as ptr->b
```

Accessing structure member through pointer using dynamic memory allocation

To access structure member using pointers, memory can be allocated dynamically usingmalloc() function defined under "`stdlib.h`" header file.

Syntax to use malloc()

```
ptr=(cast-type*)malloc(byte-size)
```

Example to use structure's member through pointer using malloc() function.

```
#include <stdio.h>
#include<stdlib.h>
struct name {
   int a;
   float b;
   char c[30];
};
int main(){
   struct name *ptr;
   int i,n;
   printf("Enter n: ");
   scanf("%d",&n);
   ptr=(struct name*)malloc(n*sizeof(struct name));
```

```
/* Above statement allocates the memory for n structures with pointer ptr pointing to base address */
  for(i=0;i<n;++i){
     printf("Enter string, integer and floating number  respectively:\n");
     scanf("%s%d%f",&(ptr+i)->c,&(ptr+i)->a,&(ptr+i)->b);
  }
  printf("Displaying Infromation:\n");
  for(i=0;i<n;++i)
     printf("%s\t%d\t%.2f\n",(ptr+i)->c,(ptr+i)->a,(ptr+i)->b);
  return 0;
}
```

Output

```
Enter n: 2

Enter string, integer and floating number  respectively:

Programming

2

3.2

Enter string, integer and floating number  respectively:

Structure

6

2.3

Displaying Information

Programming      2      3.20

Structure      6      2.30
```

C Programming Structure and Function

In C, structure can be passed to functions by two methods:

1. Passing by value (passing actual value as argument)
2. Passing by reference (passing address of an argument)

Passing structure by value

A structure variable can be passed to the function as an argument as normal variable. If structure is passed by value, change made in structure variable in function definition does not reflect in original structure variable in calling function.

Write a C program to create a structure student, containing name and roll. Ask user the name and roll of a student in main function. Pass this structure to a function and display the information in that function.

```
#include <stdio.h>
struct student{
   char name[50];
   int roll;
};
void Display(struct student stu);
/* function prototype should be below to the structure declaration otherwise compiler shows
error */
int main(){
   struct student s1;
   printf("Enter student's name: ");
   scanf("%s",&s1.name);
   printf("Enter roll number:");
   scanf("%d",&s1.roll);
   Display(s1);   // passing structure variable s1 as argument
   return 0;
}
void Display(struct student stu){
  printf("Output\nName: %s",stu.name);
  printf("\nRoll: %d",stu.roll);
```

```
}
```

Output

```
Enter student's name: Kevin Amla

Enter roll number: 149

Output

Name: Kevin Amla

Roll: 149
```

Passing structure by reference

The address location of structure variable is passed to function while passing it by reference. If structure is passed by reference, change made in structure variable in function definition reflects in original structure variable in the calling function.

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"
```

Explanation

In this program, structure variables *dist1* and *dist2* are passed by value (because value of *dist1* and *dist2* does not need to be displayed in main function) and *dist3* is passed by reference ,i.e, address of *dist3* (&dist3) is passed as an argument. Thus, the structure pointer variable *d3* points to the address of *dist3*. If any change is made in *d3*variable, effect of it is seed in *dist3* variable in main function.

C Programming Unions

Unions are quite similar to the structures in C. Union is also a derived type as structure. Union can be defined in same manner as structures just the keyword used in defining union in **union** where keyword used in defining structure was **struct.**

```
union car{

  char name[50];

  int price;

};
```

Union variables can be created in similar manner as structure variable.

```
union car{

  char name[50];

  int price;

}c1, c2, *c3;



OR;



union car{
```

```
  char name[50];

  int price;

};

-------Inside Function-----------

union car c1, c2, *c3;
```

In both cases, union variables *c1*, *c2* and union pointer variable *c3* of type **union car** is created.

Accessing members of an union

The member of unions can be accessed in similar manner as that structure. Suppose, we you want to access price for union variable *c1* in above example, it can be accessed as`c1.price`. If you want to access price for union pointer variable *c3*, it can be accessed as `(*c3).price` or as `c3->price`.

Difference between union and structure

Though unions are similar to structure in so many ways, the difference between them is crucial to understand. This can be demonstrated by this example:

```
#include <stdio.h>
union job {        //defining a union
  char name[32];
  float salary;
  int worker_no;
}u;
struct job1 {
  char name[32];
  float salary;
  int worker_no;
}s;
int main(){
  printf("size of union = %d",sizeof(u));
  printf("\nsize of structure = %d", sizeof(s));
  return 0;
```

```
}
```

Output

```
size of union = 32

size of structure = 40
```

There is difference in memory allocation between union and structure as suggested in above example. The amount of memory required to store a structure variables is the sum of memory size of all members.

Fig: Memory allocation in case of structure

But, the memory required to store a union variable is the memory required for largest element of an union.

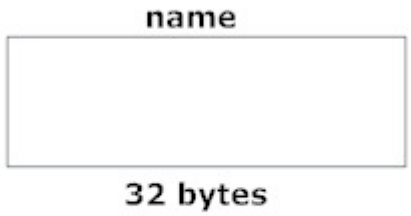

Fig: Memory allocation in case of union

What difference does it make between structure and union?

As you know, all members of structure can be accessed at any time. But, only one member of union can be accessed at a time in case of union and other members will contain garbage value.

```
#include <stdio.h>

union job {

   char name[32];

   float salary;

   int worker_no;
```

```
}u;

int main(){

   printf("Enter name:\n");

   scanf("%s",&u.name);

   printf("Enter salary: \n");

   scanf("%f",&u.salary);

   printf("Displaying\nName :%s\n",u.name);

   printf("Salary: %.1f",u.salary);

   return 0;

}
```

Output

```
Enter name

Hillary

Enter salary

1234.23

Displaying

Name: f%Bary

Salary: 1234.2
```

C Programming Structure Examples

This page contains examples and source code on structures in C programming language. To understand all examples in this page, you should have knowledge of following structure topics.

1. Structure Introduction
2. Structure and Pointers
3. Passing Structure to Function

C Programming Files

In C programming, file is a place on disk where a group of related data is stored.

Why files are needed?

When the program is terminated, the entire data is lost in C programming. If you want to keep large volume of data, it is time consuming to enter the entire data. But, if file is created, these information can be accessed using few commands.

There are large numbers of functions to handle file I/O in C language. In this tutorial, you will learn to handle standard I/O(High level file I/O functions) in C.

High level file I/O functions can be categorized as:

1. Text file
2. Binary file

File Operations

1. Creating a new file
2. Opening an existing file
3. Reading from and writing information to a file
4. Closing a file

Working with file

While working with file, you need to declare a pointer of type file. This declaration is needed for communication between file and program.

```
FILE *ptr;
```

Opening a file

Opening a file is performed using library function fopen(). The syntax for opening a file in standard I/O is:

```
ptr=fopen("fileopen","mode")


For Example:

fopen("E:\\cprogram\program.txt","w");


/* ---------------------------------------------------------- */
 E:\\cprogram\program.txt is the location to create file.
 "w" represents the mode for writing.
/* ---------------------------------------------------------- */
```

Here, the program.txt file is opened for writing mode.

Opening Modes in Standard I/O		
File Mode	Meaning of Mode	During Inexistence of file
r	Open for reading.	If the file does not exist, fopen() returns NULL.
w	Open for writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a	Open for append. i.e, Data is added to end of file.	If the file does not exists, it will be created.
r+	Open for both reading and writing.	If the file does not exist, fopen() returns NULL.

Opening Modes in Standard I/O		
File Mode	Meaning of Mode	During Inexistence of file
w+	Open for both reading and writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
a+	Open for both reading and appending.	If the file does not exists, it will be created.

Closing a File

The file should be closed after reading/writing of a file. Closing a file is performed using library function fclose().

```
fclose(ptr); //ptr is the file pointer associated with file to be closed.
```

The Functions fprintf() and fscanf() functions.

The functions `fprintf()` and `fscanf()` are the file version of `printf()` and `fscanf()`. The only difference while using `fprintf()` and `fscanf()` is that, the first argument is a pointer to the structure FILE

Writing to a file

```
#include <stdio.h>
int main()
{
  int n;
  FILE *fptr;
  fptr=fopen("C:\\program.txt","w");
  if(fptr==NULL){
    printf("Error!");
    exit(1);
```

```
   }
   printf("Enter n: ");
   scanf("%d",&n);
   fprintf(fptr,"%d",n);
   fclose(fptr);
   return 0;
}
```

This program takes the number from user and stores in file. After you compile and run this program, you can see a text file program.txt created in C drive of your computer. When you open that file, you can see the integer you entered.

Similarly, `fscanf()` can be used to read data from file.

Reading from file

```
#include <stdio.h>
int main()
{
   int n;
   FILE *fptr;
   if ((fptr=fopen("C:\\program.txt","r"))==NULL){
       printf("Error! opening file");
       exit(1);         /* Program exits if file pointer returns NULL. */
   }
   fscanf(fptr,"%d",&n);
   printf("Value of n=%d",n);
   fclose(fptr);
   return 0;
}
```

If you have run program above to write in file successfully, you can get the integer back entered in that program using this program.

Other functions like `fgetchar()`, `fputc()` etc. can be used in similar way.

Binary Files

Depending upon the way file is opened for processing, a file is classified into text file and binary file.

If a large amount of numerical data it to be stored, text mode will be insufficient. In such case binary file is used.

Working of binary files is similar to text files with few differences in opening modes, reading from file and writing to file.

Opening modes of binary files

Opening modes of binary files are `rb`, `rb+`, `wb`, `wb+`,`ab` and `ab+`. The only difference between opening modes of text and binary files is that, b is appended to indicate that, it is binary file.

Reading and writing of a binary file.

Functions `fread()` and `fwrite()` are used for reading from and writing to a file on the disk respectively in case of binary files.

Function fwrite() takes four arguments, address of data to be written in disk, size of data to be written in disk, number of such type of data and pointer to the file where you want to write.

```
fwrite(address_data,size_data,numbers_data,pointer_to_file);
```

Function `fread()` also take 4 arguments similar to `fwrite()` function as above.

File Examples

Examples of files in C Programming
C Program to read name and marks of students and store it in file
C Program to read name and marks of students and store it in file. If file already exists, add information to it.
C Program to write members of arrays to a file using fwrite()

Write a C program to read name and marks of n number of students from user and store them in a file

```
#include <stdio.h>

int main(){

   char name[50];
```

```
   int marks,i,n;
   printf("Enter number of students: ");
   scanf("%d",&n);
   FILE *fptr;
   fptr=(fopen("C:\\student.txt","w"));
   if(fptr==NULL){
       printf("Error!");
       exit(1);
   }
   for(i=0;i<n;++i)
   {
      printf("For student%d\nEnter name: ",i+1);
      scanf("%s",name);
      printf("Enter marks: ");
      scanf("%d",&marks);
      fprintf(fptr,"\nName: %s \nMarks=%d \n",name,marks);
   }
   fclose(fptr);
   return 0;
}
```

Write a C program to read name and marks of n number of students from user and store them in a file. If the file previously exits, add the information of n students.

```
#include <stdio.h>

int main(){

   char name[50];

   int marks,i,n;

   printf("Enter number of students: ");

   scanf("%d",&n);

   FILE *fptr;

   fptr=(fopen("C:\\student.txt","a"));

   if(fptr==NULL){

       printf("Error!");

       exit(1);

   }

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

   {

      printf("For student%d\nEnter name: ",i+1);

      scanf("%s",name);

      printf("Enter marks: ");

      scanf("%d",&marks);
```

```
      fprintf(fptr,"\nName: %s \nMarks=%d \n",name,marks);

   }

   fclose(fptr);

   return 0;

}
```

Write a C program to write all the members of an array of strcures to a file using fwrite(). Read the array from the file and display on the screen.

```
#include <stdio.h>

struct s

{

char name[50];

int height;

};

int main(){

    struct s a[5],b[5];

    FILE *fptr;

    int i;

    fptr=fopen("file.txt","wb");

    for(i=0;i<5;++i)

    {
```

```
        fflush(stdin);

        printf("Enter name: ");

        gets(a[i].name);

        printf("Enter height: ");

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

    }

    fwrite(a,sizeof(a),1,fptr);

    fclose(fptr);

    fptr=fopen("file.txt","rb");

    fread(b,sizeof(b),1,fptr);

    for(i=0;i<5;++i)

    {

        printf("Name: %s\nHeight: %d",b[i].name,b[i].height);

    }

    fclose(fptr);

}
```