Problem Solving With C

UE15CS151

Structures-2

This Notes can be used only for reference and kindly do not solely depend on it. Only those topics which need more explanation are included here. Please Note "The prescribed Text book has to be referred for the examination"

Array of Structures

When We can have array of integers where int is a predefined data type, cant we have an array for structured type ??

yes, we can

```
struct  struct1   a;   // Single structure variable  declaration
```

Here is the syntax

Declaration 1:

```
struct  tag
{
        members;
} arr_name[Size];
```

Example:

```
struct  struct1
{
        int a;
        float b;
} arr[5];                       //Array of structure declaration whose size is 5
```

Declaration 2:

```
struct  tag  arr_name[Size];

struct  struct1 arr[5];       //Array of structure declaration whose size is 5

struct struct1 arr[] ;        // Error – Must mention the array size
```

Assigning values to the members of array of structure

Consider this simple structure

```
struct  struct1
{
        int a;
        float b;
}s1 ;

s1 = { 100,12.5};                    //Method 1

s1.a = 100;                  // Method 2 – using DOT operator
s1.b = 12.5;
```

Now Consider an Array of structure of Size 3

```
struct struct1 arr[3]
```

inthis array of structure even index of the array is a structure

Example :
arr[0] is a structure
arr[1] is a structure
arr[2] is a structure

Hence we using DOT operator of similar syntax to assign values

Example:

```
arr[0].a = 100;                          //Method 1
arr[0].b = 12.5;

arr[1].a = 200;
arr[1].b = 5.6;

arr[2].a = 50;
arr[2].b = 8.9;
```

We can also use the array type of initialization for the array of structure

Example :

```
struct  struct1 arr[3] = {{100,12.5},{200,5.6},{50,8.9}};
```

Accessing of the members in Array of structure

We use DOT operator for this

let us display the contents of array of structure using a loop

Example :

```
struct  struct1 arr[3] = {{100,12.5},{200,5.6},{50,8.9}};

int i;
for(i=0;i<3;i++)
        printf("%d index structure values are  :   %d  &   %f\n"i,arr[i].a,arr[i].b);
```

Output :

```
        Structure values at 0 index are : 100 & 12.5
        Structure values at 1 index are : 200 & 5.6
        Structure values at 2 index are : 50 & 8.9
```

Else,

Indivitual members can also be accesed using DOT operator

Example :

```
struct  struct1 arr[3] = {{100,12.5},{200,5.6},{50,8.9}};
printf("   %d  &    %f   ", arr[0].a,arr[2].b);
```

output :   100 & 8.9

typedef Keyword:

The C programming language provides a keyword called **typedef**, which you can use to give a type, a new name

Example :

```
typedef unsigned int new;
            new a=100,b;
            new arr[10];
```

After this type definition, the identifier “new” can be used as an abbreviation for the type **unsigned int**

You can use **typedef** to give a name to your user defined data types as well.

Example :

```
typedef struct my_struct
{
        int a;
        float b;
}new;
new s1[10];   //Array of structure
new s2,s3;    //Structure variables
new *s;               //Pointer of structure type
```

Bit Fields:

Consider this example

Example 1:

```
        struct  student

        {

                        unsigned int age;

                        unsigned int marks;

       }s1={18,75};

        printf(" Structure size is %d\n",sizeof(s1));

        printf("Age is %d     &   marks is %d\n",s1.age,s1.marks");
```

Considering int to be 4 bytes

```
output:                         8

                                Age is 18     &    marks is 75
```

age for a 1st year student won't exceed 20 and marks cannot exceed the value 100. we know that the given data is stored in the form of bits and each bit carries a binary value. the binary representation of 100 is "1100100". The number of bits that it will take to store this data is 7bits and age can be store in 5 bits.

7+5 = 12 bits is enough to store the values for age and marks but in the above example the memory that will be allocated will be 8bytes and hence the memory will be wasted

so prevent such wastage we will be using bit fields

```
struct  student

        {

                        unsigned int age:7;

                        unsigned int marks:7;

       }s1;

    printf(" Structure size is %d\n",sizeof(s1));
printf("Age is %d     &   marks is %d\n",s1.age,s1.marks");
```

```
out put      :         4

Age is 18     &    marks is 75
```

The minimum size allocated for a structure variable using bit field is of one integer size . The above example shows that am sussessfully able to retain the original data.

Example 2:

```
struct  student -
{
        unsigned int age;
        unsigned int marks;
        unsigned int MorF;      // 0 – Female    &  1 – Male
        unsigned int roll_no;       //values btw 1-70
        unsigned int


}s1;
printf(" Structure size is %d\n",sizeof(s1));
```

```
output:          20             // considering int to be of size 20
```

```
struct  student -
{
        unsigned int age:5;
        unsigned int marks:7;
        unsigned int MorF:1;            // 0 – Female    &  1 – Male
        unsigned int roll_no:7;     //values btw 1-70
        unsigned int

}s1;
printf(“ Structure size is %d\n”,sizeof(s1));
```

output: 4 // considering int to be of size 20

size variations of the structure variable with respect to the bits usage:

Example 1:

```
struct new
{
        unsigned int a:25;
        unsigned int b:6:
}new1;
printf(“ Structure size is %d\n”,sizeof(new1));
```

output : 4 //total number of bits used is 25+6 = 31

Example 2:

```
struct new
{
        unsigned int a:25;
        unsigned int b:7:
}new1;
 printf(“ Structure size is %d\n”,sizeof(new1));
```

output : 4 //total number of bits used is 25+7 = 32

Example 1:

```
struct new
{
        unsigned int a:27;
        unsigned int b:6:
}new1;
 printf(“ Structure size is %d\n”,sizeof(s1));
```

output :

```
8      //total number of bits used is 25+6 = 33
```

here the size is 8 bytes

this is basically because the number of bits used is 33, which is greater than 4 bytes(32 bits), hence one more interger size ll be added to the size of the structure with bit fields.

Pointer to a structure:

Syntax : struct tag *pointer_name;

Example :

```
struct new
{
        int a,b,c;
}n1;
int main()
{
        struct new *p;
        p->a = 200;  //      Assigning the values using pointer
        p->b = 150;  //      -> is the arrow operator
        p->c = 123;
        printf("a is %d\n b is %d\n c is %d\n\n",p->a,p->b,p->c);
}
```

```
output                  :       a is 200
                                b is 150
                                c is 123
```

Array of Pointer to an array of structure:

Example:

```
struct new
{
int a; int b;
};
struct new n[3];            //    Array of structure
struct new  *p[3];          //    Array of pointer

int main()
{
int i;
for (i = 0;i<3;i++)
{
```

```
p[i]=&n[i];//Assigning the address of the array of structure to the array of pointers
}
for (i = 0;i<3;i++)
{
scanf("%d%d",&(p[i]->a),&(p[i]->b));
    }
for (i = 0;i<3;i++)
{
printf("a is %d    b is %d",p[i]->a,p[i]->b);
   }
}
```

Example 2:

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

struct employee
{
        char name[30];
        int emp_id;
        double salary;
};

int main()
{
        int i,size;
        printf("Enter the size\n");
        scanf("%d",&size);
        struct employee emp[size];



        for(i = 0; i<size ;i++)
        {
                printf("Enter the name\n");
                scanf("%s", emp[i].name);
                printf("Enter the id\n");
                scanf("%d", &emp[i].emp_id);
                printf("Enter the salary\n");
                scanf("%lf", &emp[i].salary);
```

```
	}

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

		printf("%s\t%d\t%lf\n", emp[i].name,emp[i].emp_id,emp[i].salary);
	}

	struct employee *p[size];

	for(i = 0; i<size ;i++)
	{
		p[i] = &emp[i];
		printf("%s\t%d\t%lf\n", p[i]->name,p[i]->emp_id,p[i]->salary);

	}
}
```

Array of pointer to a structure [using Dynamic memory allocation)

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

struct student
{
	char name[30];
	int age;
};
int main()
{
	struct student *p[3];

	for(int i = 0 ;i<3 ; i++)
	{
		p[i] = (struct student *)malloc(sizeof(struct student));
		printf("Enter name and age\n");
		scanf("%s%d", p[i] -> name, &p[i] -> age);
	}
	for(int i = 0 ;i<3 ; i++)
	{
		printf("%s %d\n", p[i] -> name, p[i] -> age);		}	}
```

Parameter Passing:

PASSING INDIVIDUAL MEMBERS OF A STRCTURE TO FUNCTION

[Pass by value]

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

struct A
{
        int a;
        double b;
}x;

double fun(int, double);
int main()
{
        double z;
        z = fun(x.a, x.b);
        printf("%lf\n",z);

}

double fun(int p, double q)
{
        p = 100;
        q = 10.5;
        q = p+q;
        return q;
}
```

PASSING INDIVIDUAL MEMBERS OF A STRCTURE TO FUNCTION USING & OPERATOR

[Pass by Reference]

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

struct A
{
        int a;
        double b;
}x;

double fun(int *, double *);
int main()
{
        double z;
        z = fun(&x.a, &x.b);
        printf("%lf\n",z);

}

double fun(int *p, double *q)
{
        *p = 100;
        *q = 10.5;
        *q = *p+*q;
        return *q;
}
```

PASSING THE ENTIRE STRCTURE TO FUNCTION

[Pass by value]

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

struct A
{
        int a;
        double b;
}x;

fun(struct A);
int main()
{
        double z;
        z = fun(x);
        printf("%lf\n",z);

}

fun(struct A p)
```

```
{
        p.a = 100;
        p.b = 10.5;
        double c = 100+10.5;
        return c;
}
```

PASSING THE ENTIRE STRCTURE TO FUNCTION USING & OPERATOR

[Pass by reference]

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

struct A
{
        int a;
        double b;
}x;

fun(struct A *);
int main()
{
        double z;
        z = fun(&x);
        printf("%lf\n",z);

}

fun(struct A *p)
{
        p->a = 100;
        p->b = 10.5;
        double c = p->a + p->b;
        return c;
}
```

FUNCTION RETURNING A STRUCTURE

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

struct A
{
	int a;
	double b;
}x;

struct A fun(struct A);
int main()
{
	struct A z;
	z = fun(x);
	printf("%d %lf\n",z.a,z.b);

}

struct A fun(struct A p)
{
	p.a = 100;
	p.b = 10.5;
	return p;
}
```

FUNCTION RETURNING A STRUCTURE (Using & operator)

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

struct A
{
	int a;
	double b;
}x;

struct A fun(struct A *);
int main()
```

```
{
	struct A z;
	z = fun(&x);
	printf("%d %lf\n",z.a,z.b);

}

struct A fun(struct A *p)
{
	p->a = 100;
	p->b = 10.5;
	return *p;
}
```