Lecture 09: Structures Miscellaneous

Today’s Lecture

- Passing Structure Members as arguments to Function
- Passing Structure Variables as Parameters
- Returning Structure from Function
- Pointers to structure variables
- Passing Structure Pointers as Argument to a Function
- Returning a Structure Pointer from Function
- Passing Array of structures
- Dynamic Memory Allocation (DMA) of Structure Type Variables

Example : Passing a Structure Member

We can pass individual members to a function just like ordinary variables.

```
struct student
{
    char name[20];
    int roll_no;
    int marks;
};
void print_struct(char name[], int roll_no, int marks);   //prototype of function
int main()
{
    struct student stu = {“Rashid", 1, 78};       //Note: Initializing the members of the structure;
    print_struct(stu.name, stu.roll_no, stu.marks);
    return 0;
}
void print_struct(char name[], int roll_no, int marks)
{
    cout<<"Name: "<< name;
   cout<<"Roll no: "<< roll_no;
   cout<<"Marks: "<< marks;
}
```

Passing Structure Variables as Parameters

- An individual structure member may be passed as a parameter to a function as shown in the last example.
- However, passing individual structure members, if there are many, is not only tiresome but also error-prone.
- So in such cases instead of passing members individually, we can pass structure variable itself.

Example : Passing a Structure as argument

- We can pass structure variable to a function just like ordinary variables.

```
struct student
{     char name[20];
      int roll_no, marks;
};
void print_struct(struct student stu);  //prototype of function
int main() {
    struct student stu = {“Rashid", 1, 78};
    print_struct(stu);
    return 0;
}
void print_struct(struct student stu) {
    cout<<"Name: "<< stu.name;
   cout<<"Roll no: "<< stu.roll_no;
   cout<<"Marks: "<< stu.marks;
}
```

Returning Structure from Function

- Just as we can return fundamental types and arrays, we can also return a structure from a function.
- To return a structure from a function we must specify the appropriate return type in the function definition and declaration. For example:

```
struct student change(struct student);
```

This function accepts an argument of type struct student and returns an argument of type struct student.

Example : Returning Structure from Function

We can pass structures to a function just like ordinary variables and return structure variables.

```
struct student {
    char name[20];
    int roll_no, marks;
};
void print_struct(struct student stu);     //prototype of function
struct student change(struct student); //prototype of function
int main() {
    struct student stu = {“Sohail", 1, 55};
    print_struct(stu);
   stu = change(stu);
   cout<<"After calling change() function"<<endl;
   print_struct(stu);
    return 0;
}
void print_struct(struct student stu) {
    cout<<"Name: "<< stu.name<<endl<<"Roll no: “<<endl<< stu.roll_no<<"Marks: "<< stu.marks;
}
struct student change(student csstu) {
csstu.roll_no  = 21;
 return csstu;   }
```

```
Name: Sohail
Roll no: 1
Marks: 55

After calling change() function
Name: Sohail
Roll no: 21
Marks: 55
```

Passing Structure Variables as Parameters

- Unlike arrays, the name of structure variable is not a pointer.
- So when we pass a structure variable to a function, the formal argument of print_struct() is assigned a copy of the original structure.
- Both structures reside in different memory locations and hence they are completely independent of each other.
- Any changes made by function print_struct() doesn't affect the original structure variable in the main() function.

POINTERS TO STRUCTURE VARIABLES

Pointers to structure variables

- Pointers of structure variables can be declared like pointers to any basic data type

```
Student csstudent, *ptrstudent;

ptrstudent = &csstudent;
```

- Members of a pointer structure type variable can be accessed using arrow (->) operator

```
ptrstudent-> roll_number=20; //alternate: (*ptrstudent).roll_number;

strcpy( ptrstudent->Name, “Ali”);
```

- Members of the structure variables can be accessed directly or indirectly via pointer.

```
cout<<"Name: "<< csstudent.name<<endl<<ptrstudent->rollnumber;
cout<<"Roll no: "<< csstudent.roll_no<<endl<<ptrstudent->Name;;
```

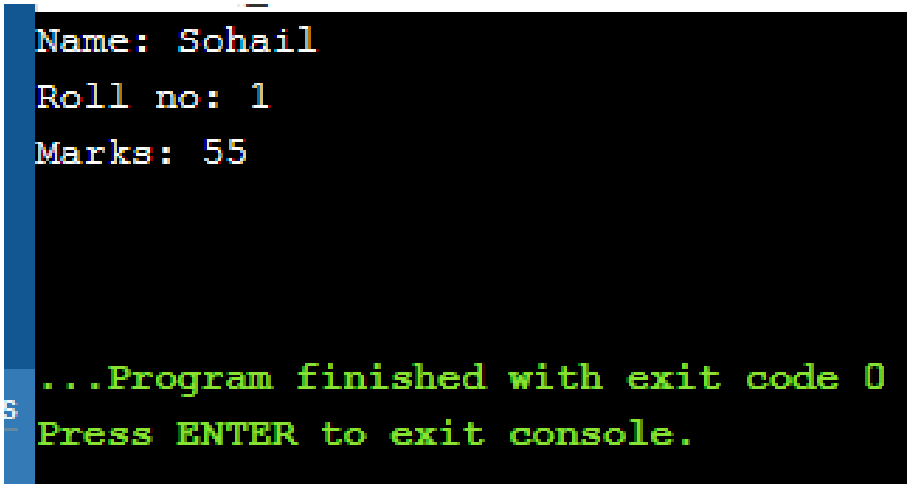
PASSING STRUCTURE POINTERS AS ARGUMENT TO A FUNCTION

Passing structure variables as pointers

- When we pass structure variables by value, a copy of the structure is passed to the formal argument.
- If the structure is large it can take quite a bit of time which make the program inefficient.
- Additional memory is required to save every copy of the structure.
- It is better to pass structure pointers as arguments to a function

Passing structure variables as pointers

```
struct student
{
    char name[20];
    int roll_no;
    int marks;
};
void print_struct(student *);          //prototype of function
int main()
{
    struct student stu = {“Sohail", 1, 55};
    print_struct(& stu);
    return 0;
}
void print_struct(student * csstu)
{
    cout<<"Name: "<< csstu->name<<endl;        //alternate. cout<< (*csstu).name
   cout<<"Roll no: "<< csstu->roll_no<<endl;   //alternate. cout<< (*csstu).roll_no
   cout<<"Marks: "<< csstu->marks<<endl;       //alternate. cout<< (*csstu).marks
}
```



Passing structure variables as pointers

- Passing structure pointer to a variable lets the function modify the values of the members of the structure, which will also be reflected in the main.

RETURNING A STRUCTURE POINTER FROM FUNCTION

Returning a Structure Pointer from Function

- A function can also return a pointer to structure variable.
- Appropriate return type shall be specified in the function definition and function declaration. Such as

```
struct student * change(struct student *);
```

Example : Returning a Structure Pointer from Function

```
struct student {
    char name[20];
    int roll_no, marks;
};
void print_struct(struct student stu *);  //prototype of function
struct student * change(struct student *);        //prototype of function
int main() {
    struct student stu = {“Sohail", 1, 55};
struct student *ptr_stu1 = &stu, *ptr_stu2;
    print_struct(ptr_stu1);
   ptr_stu2 = change(ptr_stu1);
   cout<<"After calling change() function"<<endl;
   print_struct(ptr_stu2);
    return 0;
}
void print_struct(struct student * stu) {
cout<<"Name: "<< stu->name<<endl<<"Roll no: “<<endl<< stu->roll_no<<"Marks: "<< stu->marks;
}
struct student * change(student *csstu) {
csstu->marks  = 50;
 return csstu;   }
```

```
Name: Sohail
Roll no: 1
Marks: 55

After calling change() function
Name: Sohail
Roll no: 1
Marks: 50
```

PASSING ARRAY OF STRUCTURE TO FUNCTION

Passing Array of Structures to Function

- Array of structures can be passed to functions the same way as array of integers is passed to function.
- For example, to pass an array of struct student, the prototype of function print_struct() is declared which accepts an argument of type array of structures.

```
void print_struct (struct student str_arr[]);
```

- In main(), an array of structures of type student can be declared as:

```
struct student stu[3] = { {“John", 1, 55 },  {“Tim", 2, 65 }, {“Green", 3,75}};
```

- Similarly, in main() print_strct() can be called as:

```
print_struct(stu);
```

- Note that the name of the array is a constant pointer to the 0th element of the array, the formal argument of print_struct() is assigned the address of variable students.

Passing Array of Structures to Function

- Print_struct() function can then iterate through the elements of the array that is passed to it as argument. For example:

```
void print_struct(struct student stu[ ]) {

        int i;

            for(i= 0; i<3; i++)
        {
        cout<<"Name: "<< stu[i].name;
        cout<<"Roll no: "<< stu[i].roll_no;
        cout<<"Marks: "<< stu[i].marks;
        }
}
```

Dynamic Memory Allocation (DMA) of Structure Type Variables

- We can also dynamically allocate the memory of any structure type variable using new operator.
- For example.

```
Student *ptrstudent;
ptrstudent = new ptrstudent;
```

- Syntax is similar to dynamically allocating memory for variables of other normal data types such as int, float etc.

```
float *PtrTofloat;
PtrTofloat = new float;
```

- We can delete memory allocated at execution time using delete

```
delete ptrstudent;
```