8. Structure

Dr. Ami Tusharkant Choksi
Associate Professor,
Computer Engineering Department,
C.K.Pithawala College of Engineering & Technology, Surat.
2019
ami.choksi@ckpcet.ac.in

8. Structure

Basics of structure, structure members, accessing structure members, nested structures, array of structures, structure and functions, structures and pointers

Total Hours: 04

Module Weightage: 9%

Table of contents

- Introduction-Basics of structure
- Structure Members
- Accessing structure members
- Nested structures
- Array of structures
- Structure and functions
- Structures and pointers

Introduction-Basics of structure

- A structure is a user defined data type in C.
- A structure creates a data type that can be used to group items of possibly different types into a single type.
- 'struct' keyword is used to create a structure.
- syntax :

```
struct structname{
        data_type var1name;
        data_type var2name;
      };
```

Structure members

- ```
  struct student{
  int rollno;
  char name[25];
  double percentrage;
  };
  ```
- rollno, name, percentage are called the members of structures.
  ```

Structure Variable

- Structure variables: A structure variable can either be declared
  - with structure declaration or
  - as a separate declaration like basic types.

Structure variable: with structure declaration

```
struct Point
{
    int x, y;
} p1;  // The variable p1 is
declared with 'Point'
```

Structure variable: as a separate declaration like basic types

```
struct Point {
    int x, y;
};
int main()
{
    struct Point p1;  // The
variable p1 is declared like a
normal variable
}
```

Strucutre members initialization

- Structure members cannot be initialized with declaration. For example the following C program fails in compilation.

```
struct Point {
    int x = 0;  // COMPILER ERROR:
cannot initialize members here
    int y = 0;  // COMPILER ERROR:
cannot initialize members here
};
```

Strucutre members initialization

- Structure members can be initialized using curly braces '{}'. For example, following is a valid initialization.

```
int main() {
   // A valid initialization.
member x gets value 0 and y
   // gets value 1.  The order of
declaration is followed.
   struct Point p1 = {0, 1};  }
```

Accessing structure members

- Structure members can be accessed using '.' operator
- syntax:
- p1.x = 20;
- p1.y = 10;

Structure Accessing Program

```
#include<stdio.h>
struct Point  {
int x, y; };
int main() {
struct Point p1 = {0, 1};
// Accesing members of point p1
p1.x = 20;
printf ("x = %d, y = %d", p1.x,
p1.y);
```

Structure Accessing Program...

```
return 0;
}
```

Output:

x = 20, y = 1

Designated Initialization

- Designated Initialization allows structure members to be initialized in any order.

```
#include<stdio.h>
struct Point {
int x, y, z; };
int main(){
// Examples of initializtion
using designated initialization
```

Designated Initialization

```
struct Point p1 = {.y = 0, .z =
1, .x = 2};
struct Point p2 = {.x = 20};
printf ("x = %d, y = %d, z =
%d\n", p1.x, p1.y, p1.z);
printf ("x = %d", p2.x);
return 0; }
```

//Output: x = 2, y = 0, z = 1

//x = 20

Nested Structures

- Nesting one structure within another structure to create complex data.

```
struct Point {
int x, y;
};
struct circle{
struct Point center;
float radius;};
```

Nested Structures

```
struct address   {
    char city[20];
    int pin;
    char phone[14];
};
struct employee{
    char name[20];
    struct address add;
 };
```

Nested structure Program

```
#include <stdio.h>
struct Point {
    int x,y;
};
struct circle {
   struct Point center;
   float radius;
};
int main(){
    struct circle c;
    c.center.x =10;
    c.center.y =10;
    c.radius=20.5;
    printf("%d:%d:%f",
c.center.x, c.center.y,
c.radius);    return 0;}
//Output: 10:10:20.50000
```

Array of structures

- Like other primitive data types, we can create an array of structures.
- syntax: `struct Point arr[10];`

Array of structures: Example

```
#include<stdio.h>
struct Point {
int x, y;
};
int main() {
//Create an array of structures
struct Point arr[10];
```

Array of structures: Example

```
// Access array members
arr[0].x = 10;
arr[0].y = 20;

printf("%d %d", arr[0].x, arr[0].y);
return 0;
}
//Output:10 20
```

Structure and functions

- Just like other variables, a structure can also be passed to a function.
- We may pass the structure members into the function or pass the structure variable at once.

Structure and functions:Example

```
#include <stdio.h>
struct Point {
        int x,y;
};
int main(){
void display(struct Point p);
struct Point p = {10,10};
display(p);    return 0; }
```

```
void display(struct Point c){
   printf("%d:%d",c.x, c.y);
}
//Output: 10:10
```

Structures and pointers

- Like primitive types, we can have pointer to a structure. If we have a pointer to structure, members are accessed using arrow (->) operator.
- `struct Point *p2 = &p1;`
- Accessing structure members using structure pointer
- `printf("%d %d", p2->x, p2->y);`

Structures and pointers:Program

```
#include<stdio.h>
struct Point {
int x, y;
};
int main(){
struct Point p1 = {1, 2};
```

Structures and pointers:Program

```
// p2 is a pointer to structure
p1
struct Point *p2 = &p1;
// Accessing structure members
using structure pointer
printf("%d %d", p2->x, p2->y);
return 0;
}
```

References

1. Structure in C, https://www.geeksforgeeks.org/structures-c/
2. Nested structure in C, https://www.javatpoint.com/nested-structure-in-c
3. Contents here are also taken from the Assignments I solved while attending NPTEL course titled “Problem solving through Programming In C”, 2019.
4. Structure, https://www.includehelp.com/c-programs/c-structure-and-union-program-to-demonstrate-example-of-nested-structure.aspx