CSE101-Lec#36-38

Nested Structure

Union

Created By:
Amanpreet Kaur &
Sanjeev Kumar
SME (CSE) LPU

Outline

- Nested Structure
- Union

Nested Structure

- Nested structures are structures as member of another structure.
- We can also take objects of one structure as member in another structure.
- Thus, a structure within a structure can be used to create complex data application.
- Dot operator is used twice because we are accessing first structure through the object of second structure.

Nested Structure

Two ways of declaring structure within structure or Nested structure:

- Declare two separate structures
- Embedded structures

Example

```
struct Date
{
        int dd;
        int mm;
        int yy;
};
struct Student
{
        char name[20];
        int rollno;
        int marks;
        struct Date dob;
};
```

Here structure Student is nesting structure and structure date is nested structure

Example: embedded structures

```
struct Student
{
        char name[20];
        int rollno;
        struct date
        {
                int dd;
                int mm;
                int yy;
        } dob;
};
```

```
#include<stdio.h>
void main()
{
 struct time{
  int second;
  int minute;
  int hour;
 };
 struct car
 {
  int carno;
  struct time st;
 };

 struct car myCar;
 printf("\n car no. starting time reaching
time:");
 scanf("%d", &myCar.carno);
 scanf("%d %d %d", & myCar.st.hour,
&myCar.st.minute, &myCar.st.second);
 printf("\n%d", myCar.carno);
 printf("\t %d:%d:%d \t" myCar.st.hour,
myCar.st.minute, myCar.st.second);
}
```

WAP to read and display the car number, starting time and reaching time using structure within structure.

Unions

- `union`
  - Memory that contains a variety of objects over time
  - Only contains one data member at a time
  - Members of a `union` share space
  - Conserves storage
  - Only the last data member defined can be accessed
- `union` definitions
  - Same as struct

```
union Number {
  int x;
  float y;
};
union Number value;
```

Union

- Union is similar as structure. The major distinction between them is in terms of storage.
- In structure each member has its own storage location whereas all the members of union uses the same location.
- The union may contain many members of different data type but it can handle only one member at a time union can be declared using the keyword union.

Example

- A class is a very good example of structure and union in this example students are sitting in contiguous memory allocation as they are treated as a structure individually. And if we are taking the place of teacher then in a class only one teacher can teach. After leaving the first teacher then another teacher can enter.

Union Declaration

```
union item
{
 int m;
 float x;
 char c;
}code;
```

This declare a variable code of type union item

Unions

- Valid `union` operations
  - Assignment to `union` of same type: `=`
  - Taking address: `&`
  - Accessing union members: `.`
  - Accessing members using pointers: `->`

Program using union.

```
#include <stdio.h>
union job{
   char name[32];
   float salary;
   int worker_no;
}u;
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);
}
```

```
Enter name Hillary
Enter salary 1234.23
Displaying
 Name: f%Bary
Salary: 1234.2
```

Initially, *Hillary* will be stored in u.name and other members of union will contain garbage value. But when user enters value of salary, 1234.23 will be stored in u.salary and other members will contain garbage value. Thus in output, salary is printed accurately but, name displays some random string.

Enumeration Constants

- Enumeration
  - Set of integer constants represented by identifiers
  - Enumeration constants are like symbolic constants whose values are automatically set
    - Values start at `0` and are incremented by `1`
    - Values can be set explicitly with `=`
    - Need unique constant names
  - Example:

```
enum Months { JAN = 1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT,
  NOV, DEC};
```

    - Creates a new type `enum` Months in which the identifiers are set to the integers `1` to `12`
  - Enumeration variables can only assume their enumeration constant values (not the integer representations)

```
  #include <stdio.h>
   /* enumeration constants represent months of the
year */

   enum months { JAN = 1, FEB, MAR, APR, MAY, JUN,
                 JUL, AUG, SEP, OCT, NOV, DEC };
void main()
{
 enum months month; /* can contain any of the 12
months */
 /* initialize array of pointers */

 const char *monthName[] = { "", "January",
"February", "March", "April", "May", "June",
"July", "August", "September", "October",
"November", "December" };

for ( month = JAN; month <= DEC; month++ ) {
 printf( "%2d%11s\n", month, monthName[ month ] );
} /* end for */
}
```

```
 1    January
 2   February
 3      March
 4      April
 5        May
 6       June
 7       July
 8     August
 9  September
10    October
11   November
12   December
```

Next Class: File Handling

cse101@lpu.co.in