Short-term Hands-on Supplementary Course on C programming

Session 11:
Structures

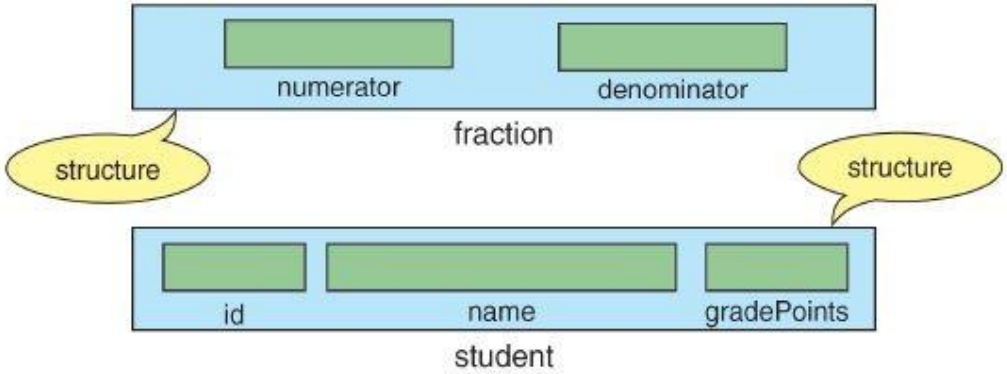
Nandakishor. V
Nitheesh Kumar. N
Sanjay Rojar U

Agenda

1. Administrative Instructions
2. What are Structures in C?
3. Declaring Structures in C
4. Structures in Memory
5. Initializing Structures
6. Accessing data in Structures
7. Array of Structures
8. Nested Structures
9. Functions and Structures
10. Tutorial
11. Next Session: More Structures

What are structures in C?

A **structure** in C is a user-defined data type. It is used to bind the two or more similar or different data types or data structures together into a single type. Structure is created using struct keyword and a structure variable is created using struct keyword and the structure tag name.a single name.



Declaring structures in C

```
struct structure_name
{
    Data_member_type data_member_defination;
    Data_member_type data_member_defination;
    Data_member_type data_member_defination;
    ...
    ...
}(structure_variables);


struct Student
{
    char name[50];
    int class;
    int roll_no;
} student1;
```

1

```
// First way to typedef
typedef struct strucutre_name new_name;

-- -
// Second way to typedef
typedef struct strucutre_name
{
    // body of structure
}new_name;
```

3

```
struct structure_name {
    // body of structure
} variables;

struct Student {
    char name[50];
    int class;
    int roll_no;
} student1; // here 'student1' is a structure variable
```

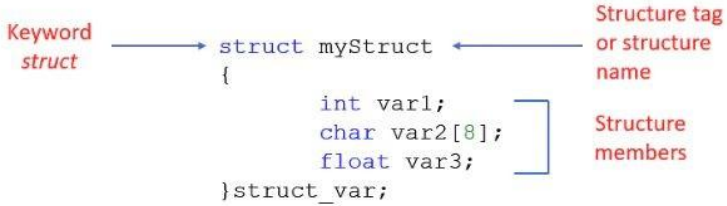
2

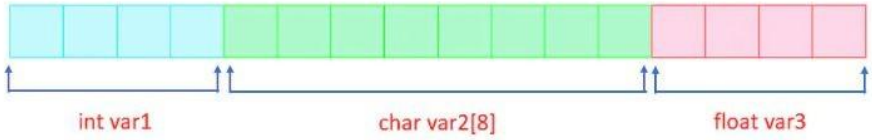
```
struct Student
{
    char name[50];
    int class;
    int roll_no;
};

int main()
{
    //struct structure_name variable_name;

    struct Student a; // here a is the variable of type Student
    return 0;
}
```

4

Structures in Memory

If we create an object of some structure, then the compiler allocates **contiguous memory** for the **data members** of the structure. The size of allocated memory is ***at least*** *the* ***sum of sizes of all data members***. The compiler can use **padding** and in that case there will be **unused space** created between two data members.

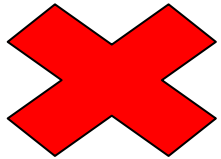




Memory representation for struct_var

Initializing data for Structures

```
struct Student
{
    char name[50] = {"Student1"};
    int class = 1;
    int roll_no = 5;
};
```



1. Using dot '.' operator
2. Using curly braces '{}'
3. Designated initializers

```
#include <stdio.h>

// creating a structure
struct Student
{
     char name[50];
     int class;
     char section;
};

int main ()
{
    // creating a structure variable and initialzing some of its members
    struct Student student1 = {.section = 'B', .class = 6};

    // printing values
    printf("Student1 Class is: %d\n",student1.class);
    printf("Student1 Section is: %c",student1.section);
}
```

```
struct structure_name variable_name;

variable_name.member = value;
```

```
struct stucture_name v1 = {value, value, value, ..};
```

Accessing data in Structures

Just like initialization, we use the dot (.) operator

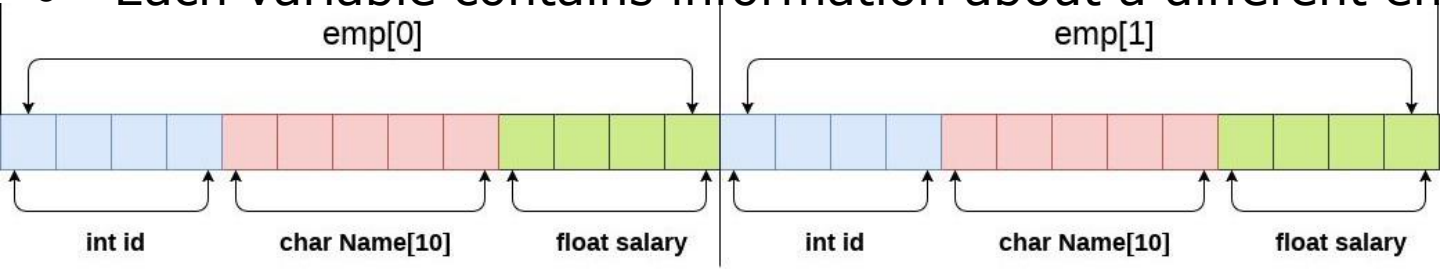
```
structure_variable.structure_member;
```

```
// creating structure
struct Complex
{
    // defining its members
   int real;
   int imaginary;
};
```

```
// declaring structure variable
struct Complex var;

// accessing class variables and assigning them value
var.real = 5;
var.imaginary = 7;
```

Array of Structures

- Collection of multiple structure variables.
- Each variable contains information about a different entity.




```
struct employee
{
    int id;
    char name[5];
    float salary;
};
struct employee emp[2];
```

sizeof (emp) = 4 + 5 + 4 = 13 bytes

sizeof (emp[2]) = 26 bytes

Functions and Structures

- Structure instances can be passed around as arguments, like any other data type.
- They can also be returned by any function.

```c
struct address{
    char door_num[5];
    char street[20];
    char pin[10];

};

typedef struct address Address;

void display_address(Address address){
    // Note that the 'Address' type already has access to the members of the structure
    printf("\nDoor No.%s, %s. PIN: %s\n", address.door_num, address.street, address.pin);
}
```

Nested Structures

```
struct Parent{
  //.....
  struct NestedStructure{
    //.....
  }
}
```

- **Embedded Structure**
  Define one structure in the definition of another structure.

- **Separate Structure**
  The dependent structure is used inside the Main/Parent structure by taking a member of the dependent structure type in the definition of the parent structure.

Tutorial

1. Create a data type ‘fraction’ with numerator and denominator. Perform fraction addition, subtraction, multiplication and division. Note: Find LCM for addition and subtraction problem. Make it a menu driven program.
2. Create a datatype ‘Student’ with name, roll number, marks for 5 subjects as an array. Maintain an array of student type and sort them based on their total marks in descending order.

Any Queries !?

Thank You for attending!

Contact us regarding any questions through email

nandakishor2010608@ssn.edu.in
nitheesh2010343@ssn.edu.in
sanjayrojar2010085@ssn.edu.in