Problem Solving With C - UE24CS151B

Structures in C

Prof. Sindhu R Pai
PSWC Theory Anchor, Feb-May, 2025
Department of Computer Science and Engineering

Structures in C

- Introduction
- Characteristics
- Declaration
- Accessing members
- Initialization
- Memory allocation
- Comparison

Introduction

- A user-defined data type that allows us to combine data of different types together.
- Helps to construct a complex data type which is more meaningful.
- Provides a single name to refer to a collection of related items of different types.
- Provides a way of storing many different values in variables of potentially different types under the same name.
- Generally useful whenever a lot of data needs to be grouped together.
- Creating a new type decides the binary layout of the type

Characteristics/Properties

- Contains one or more components(homogeneous or heterogeneous) – Generally known as data members. These are named ones.

- **Order of fields and the total size of a variable** of that type is decided when the new type is created

- Size of a structure depends on implementation. Memory allocation would be **at least equal to the sum of the sizes of all the data members** in a structure. Offset is decided at compile time.

- Compatible structures may be assigned to each other.

Syntax :

- Keyword **struct** is used for creating a structure.
- The format for declaring a structure is as below:

```
struct <structure_name>
{
                data_type member1;
                data_type member2;
                .....
                data_type memebern;
};                  // semicolon compulsory
```

Example: User defined type Student entity is created.

```
struct Student
{
                 int roll_no;
                char name[20];
                int marks;
};
```

Note: No memory allocation for declaration/description of the structure.

Declaration

- Members of a structure can be accessed only when instance variables are created

- If struct Student is the type, the instance variable can be created as:

    struct student s1;   // s1 is the instance variable of type struct Student

    struct student* s2;   // s2 is the instance variable of type struct student*.
                                        // s2 is pointer to structure

- Declaration (global) can also be done just after structure body but before semicolon.

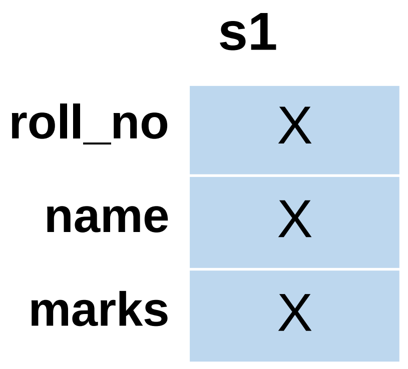

Fig. 1. After declaration, only undefined entries (X)

Initialization

- Structure members can be initialized using curly braces ‘{}’ and separated by comma.

- Data provided during initialization is mapped to its corresponding members by the compiler automatically.

- Further extension of initializations can be:

  1. **Partial initialization**: Few values are provided.
     Remaining are mapped to zero. For strings, ‘\0’.
  2. **Designated initialization**:
     - Allows structure members to be initialized in any order.
       - This feature has been added in C99 standard.
     - Specify the name of a field to initialize with **‘.member_name =’ OR ‘member_name:’** before the element value. Others are initialized to default value.

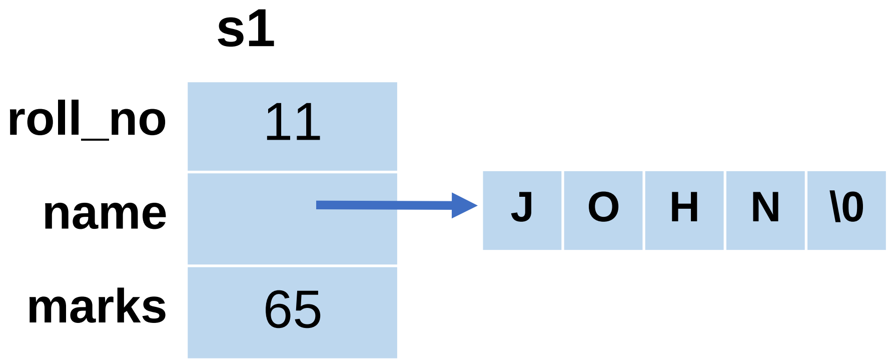

Fig. 2. After initialization, entries are mapped

Accessing data members

- Operators used for accessing the members of a structure.
  1. Dot operator (.)
  2. Arrow operator (->)
- Any member of a structure can be accessed using the structure variable as:

structure_variable_name.member_name

Example: s1.roll_no

//where s1 is the structure variable name and roll_no member is data member of s1.

- Any member of a structure can be accessed using the pointer to a structure as:

pointer_variable->member_name

Example: s2->roll_no

// where s2 is the pointer to structure variable and we want to access roll_no member of s2.

Memory allocation

- **At least equal to the sum of the sizes of all the data members.**
- Size of data members is implementation specific.
- Coding Examples

Comparison of structures

- Comparing structures in C is not permitted to check or compare directly with logical and relational operators.
- Only structure members can be compared with relational operator.
- Coding examples

THANK YOU

Department of Computer Science and Engineering

Dr. Shylaja S S, Director, CCBD & CDSAML, PESU
Prof. Sindhu R Pai - sindhurpai@pes.edu
Dr. Shruti Jadon, CSE, PESU

Ack: Teaching Assistant - U Shivakumar