CSE101-Lec#30-31

Derived Types

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

Outline

- Declaration of a structure
- Definition and initialization of structures.
- Accessing structures.

Introduction

- Structures are **derived data types—they are constructed using objects of other types.**
- Structures
  - Structure is a group of data items of different data types held together in a single unit.
  - Collections of related variables under **one name**
    - Can contain variables of different data types
  - Commonly used to define records to be stored in files.

Why Use Structures?

- Quite often we deal with entities that are collection of dissimilar data types.
- For example, suppose you want to store data about a **car**. You might want to store its name (a string), its price (a float) and number of seats in it (an int).
- If data about say 3 such cars is to be stored, then we can follow two approaches:
  - Construct individual arrays, one for storing names, another for storing prices and still another for storing number of seats.
  - Use a structure variable.

Structure

- There are three aspects of working with structures:
  - Defining a structure type
  - Declaring variables and constants of newly created type
  - Using and performing operations on the objects of structure type

Structure Definition

Syntax

```
struct  sname {
        type var1;
        type var2;
        type var3;
                .
                .
        type varN;
};
```

struct is a keyword to define a structure.
sname is the name given to the structure/structure tag.
type is a built-in data type.
var1,var2,var3,…..,varN are elements of structure being defined.
; semicolon at the end.

Structure Definitions

- Example:

```
struct car{
  char *name;
  int seats;
  float price
 };
```

  - `struct` **keyword** introduces the definition for structure `car`
  - `car` is the structure name or tag and is used to declare variables of the structure type
  - `car` contains three members of type `char, float, int`
    - These members are `name, price` and `seats`.

➢ No variable has been associated with this structure

➢ No memory is set aside for this structure.

Structure Definitions

- `struct` information
  - A structure definition does not reserve space in memory .
    - Instead creates a new data type used to define structure variables
- Defining variables of structure type
  - Defined like other variables:

```
Car myCar, cars[5], *cPtr;
```

  - Can use a comma separated list along with structure definition:

```
struct car{
    char *name;
    int seats;
    float price;
} myCar, cars[5], *cPtr;
```

At this point, the memory is set aside for the structure variable myCar.

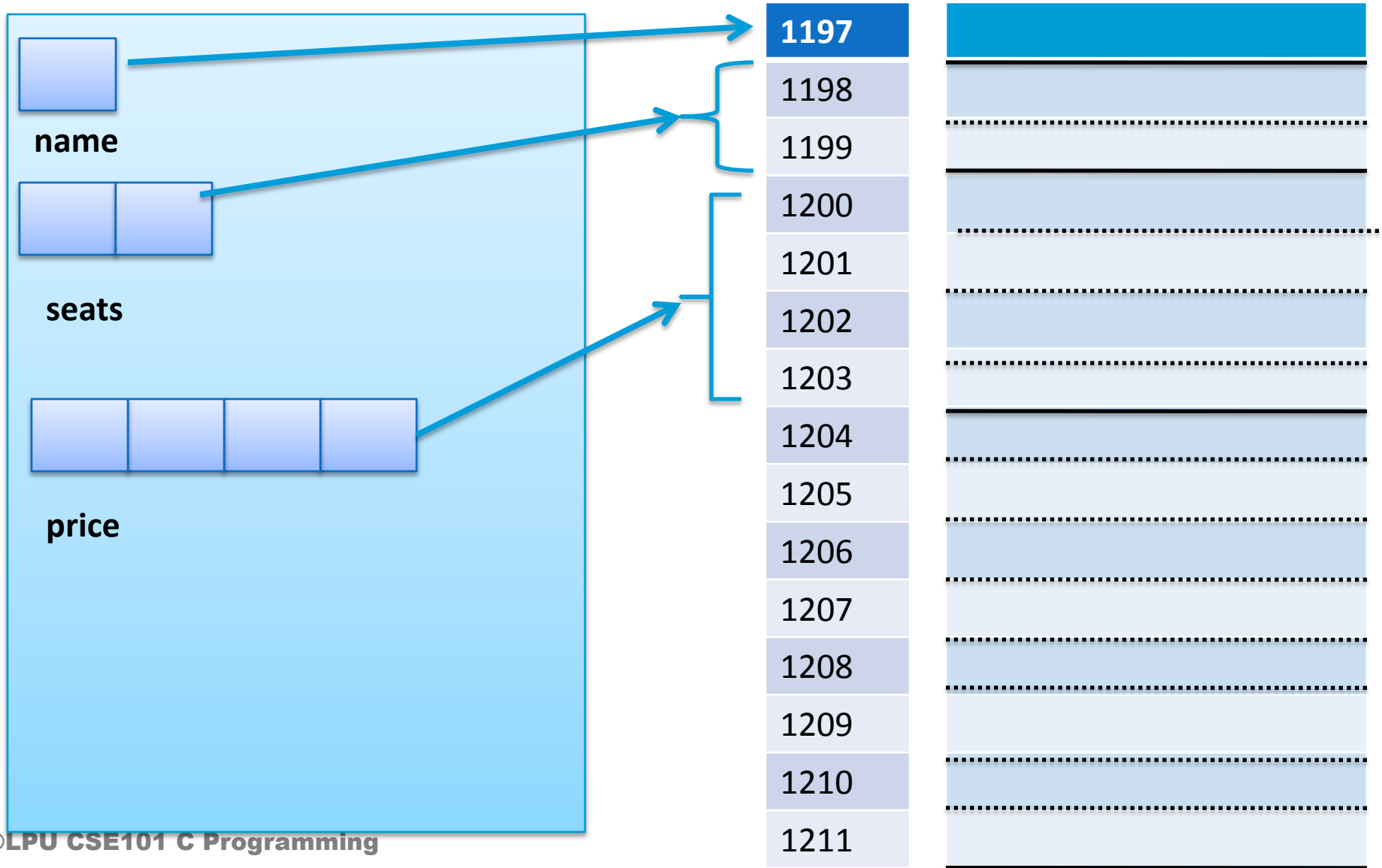
How the members are stored in memory

Consider the declarations to understand how the members of the **structure variables are stored in memory**

```
struct car{
    char *name;
    int seats;
    float price;
}myCar,
```

Note: **all members are stored in contiguous memory location in order in which they are declared.**

How the members of the structure variables are stored in memory



Structure Definitions

- Operations that can be performed on structures
  - Assigning a structure to a structure of the same type
  - Taking the address (`&`) of a structure
  - Accessing the members of a structure
  - Using the `sizeof` operator to determine the size of a structure

Initializing Structures

- Initializer list
  - To initialize a structure similarly like arrays
  - Example:

```
car myCar = { “Renault", 500000, 2 };
```

  - Could also define and initialize `myCar` as follows:

```
Car myCar;
myCar.name = “Renault”;
myCar.price = 500000;
myCar.seats = 2;
```

Accessing Members of Structures

- Two operators are used to access members of Structures:
  - Dot operator (.) used with structure variables

```
car myCar;
Printf("%d", myCar.seats);
```

  - Arrow operator (->) used with pointers to structure variables

```
car *myCarPtr = &myCar;
printf("%d", myCarPtr->seats);
```

  - `myCarPtr->name` is equivalent to

```
(*myCarPtr).seats
```

dot Operator

- Members are accessed using dot operator.
- It provides a powerful and clear way to refer to an individual element.

- **Syntax:** **sname.vname**
- **sname is structure variable name.**
- **vname is name of the element of the structure.**
- **Eg:** the members of the structure variable ***car*** can be accessed as

```
myCar.name, myCar.seats, myCar.price
```

Use of Assignment Statement for Structures

- The main advantage of structure is that it can be treated as single entity.
- The only legal operations that can be performed on structure are copying to it as a single unit using the assignment operator.
- Value of one structure variable can be assigned to another variable of the **same type** using simple assignment statement.
- If `myCar` and `newCar` are structure variable of type `car`, then

```
newCar = myCar;
```

Use of Assignment Statement for Structures

- When we assigns value of structure variable ***myCar*** to ***newCar;*** all values of members of one structure get copied into corresponding members of another structure.
- Or we can copy one member at a time:
  - `newCar.name = myCar.name;`
- ***Simple assignment cannot be used this way for arrays.***
- This is really a big advantage over arrays where in order to copy one array into another of same type, we have copied the contents element by element either using loop or individually.

Program to show how to access structure.

```
#include <stdio.h>
struct car{
        char *name;
        int seats;
        float price;
}; //end structure car

int main()
{
struct car myCar; //define struct variable
myCar.name = "Renault";
myCar.price = 500000;
myCar.seats = 2;
printf("%s %f %d \n", myCar.name,
myCar.price, myCar.seats);
} //end main
```

```
Renault 500000 2
```

Program to enter data into structure.

```
#include <stdio.h>
struct  car{
       char name[50];
        int seats;
        float price;
};
main()
{
   struct car myCar;
   printf("Enter name of car:\n");
     gets(myCar.name);
   printf("Enter number of seats in car:\n");
     scanf("%d", &myCar.seats);
   printf("Enter price of car:\n");
     scanf("%f", &myCar.price);
   printf("\n\nParticulars of car are:\n");
   printf("Car name:%s",myCar.name);
   printf("\nNumber of seats:%d",
myCar.seats);
   printf("\nPrice:%f", myCar.price);
} //end main
```

```
Enter name of car: Micra
Enter number of seats in car: 4
Enter price of car: 600000

Particulars of car are:
Car name: Micra
Number of seats: 4
Price: 600000
```

Array & Structure

Array	Structure
1. It is a collection of data items of same data type.	1. It is a collection of data items of different data types.
2. It has declaration only.	2. It has declaration & definition.
3. There is no keyword.	3. `struct` is the keyword.
4. An array name represents the address of the starting element.	4. A structure name is called tag. It is a short hand notation of the declaration.
5. An array cannot have bit fields.	5. It may contain bit fields.

typedef

- typedef
  - Creates synonyms (aliases) for previously defined data types
  - Use typedef to create shorter type names
  - Example:

```
typedef struct car CAR;
```

  - Defines a new type name CAR as a synonym for type struct car *
  - typedef does not create a new data type
    - Only creates an alias

typedef

- C programmers often use `typedef` to define a structure type, so a structure **tag** is not required.
- For example, the following definition

```
typedef struct {
        char *name;
        int seats;
        flaot price
        }car;
```

creates the structure type car without the need for a separate typedef statement.

- `car myCar; /*we can create variable of car without using struct keyword*/`

Next Class: Structure, functions and pointers

cse101@lpu.co.in