Department of Computer Science and Engineering, PES University, Bangalore, India

Lecture Notes
Problem Solving With C
UE24CS151B

Lecture #13
Array of Structures

By,
Prof. Sindhu R Pai,
Theory Anchor, Feb-May, 2025
Assistant Professor
Dept. of CSE, PESU

Many Thanks to
Dr. Shylaja S S (Director, CCBD and CDSAML Research Center, PES University)
Prof. Nitin V Pujari (Dean, Internal Quality Assurance Cell, PES University)

Unit #: 3

Unit Name: Text Processing and User-Defined Types

Topic: Array of Structures

Course objectives: The objective(s) of this course is to make students

- Acquire knowledge on how to solve relevant and logical problems using computing Machine.
- Map algorithmic solutions to relevant features of C programming language constructs.
- Gain knowledge about C constructs and its associated ecosystem.
- Appreciate and gain knowledge about the issues with C Standards and it's respective behaviours.

Course outcomes: At the end of the course, the student will be able to:

- Understand and Apply algorithmic solutions to counting problems using appropriate C Constructs.
- Understand, Analyze and Apply sorting and Searching techniques.
- Understand, Analyze and Apply text processing and string manipulation methods using Arrays, Pointers and functions.
- Understand user defined type creation and implement the same using C structures, unions and other ways by reading and storing the data in secondary systems which are portable.

Sindhu R Pai

Theory Anchor, Feb - May, 2025

Dept. of CSE,

PES University

Introduction

We know that the structures are used to group different types of variables under one name. But what if we want to store multiple records of such structured data? That's where A**rray of structures** comes into play. **An array of structures allows us to maintain a collection of records—each record being a structure.** This is particularly useful in programs dealing with multiple entities of the same type, such as **collection of students, collection of employees, collection of products, collection of books, collection of items, collection songs, collection of movies, collection of images, collection of subjects, collection of files, collection of devices etc.**

An **array of structures** is a collection **of structure variables stored in contiguous memory locations**, allowing you to manage **multiple records of the same type** efficiently.

Consider the Student structure.

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

The new data type called **struct student** is shown as above.

- We create a structure variable s1 as:
  struct Student s1;
- If the user wishes to store details of two students then two variables should be created as:
  struct Student s1, s2;
- To **store the details of 100 students,** we would be required to use 100 different structure variables from s1 to s100, which is definitely not very convenient. A better approach would be to **use an Array of structures.**

Defining an Array of structure variable – Two Approaches

1. **At the time of structure Declaration:** Immediately after defining the structure by appending them to the closing brace.

```
struct student {
   int roll_no;    char name[100];    int marks;
} s[100];  // Structure variable 's' declared as an array of 100 elements
```

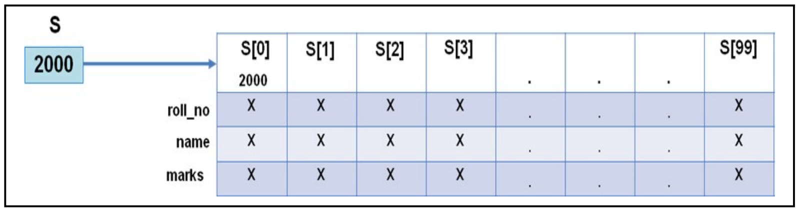
2. **After the Structure Declaration is complete:**
   - Inside the main() function:

```
struct student s[100];
```

   - Globally using the struct keyword: (Less preferred approach)

```
struct student {
        int roll_no;    char name[100];    int marks;
};
struct student s[100];
```

NOTE: In an array of structures, all elements are stored in contiguous memory locations, just like arrays of primitive data types.



Initialization of Array of Structures

Compile-time Initialization: Here, user has to enter the details within the program.

Case 1: `struct student S1[] = { {1, "John", 60}, {2,"Jack", 40}, {3, "Jill", 77} };`

// size is decided by the compiler **based on how many students details are stored**

Case 2: `struct student S2[3] = { {1, "John", 60}, {2,"Jack", 40}, {3, "Jill", 77} };`

// size is specified and initialized values are exactly matching with the size specified.

Case 3: `struct student S3[3] = {1, "John", 60, 2,"Jack", 40, 3, "Jill", 77};`

//initialization can also be done this way

Case 4: struct student S4[5] = { {1, "John", 60} };

//Partial initialization. Default values are stored in remaining memory locations

Runtime Initialization: Runtime is the period of time when a program is running and user is allowed to enter the input values to the variables of the structures.

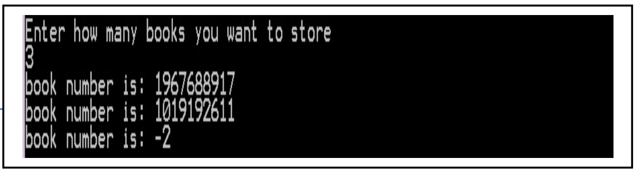
Point to note:

Flexible Array Member(FAM) is a feature introduced in C99, which allows a structure to contain an array without a fixed size as its last member. This enables the structure to accommodate a variable amount of data. We can declare an array without a dimension and whose size is flexible in nature. Such an array inside the structure should **preferably be declared as the last member of structure** and its size is variable meaning can be changed be at runtime. Hence, must use **Dynamic Memory Management functions** to create enough space. The structure must contain at least one more named member in addition to the FAM. Example structure:

```
struct student {
   int roll_no;
   char name[100]
   int marks[];  // Flexible Array Member
};
```

Coding Example_1: Without initializing any value to member of structure i,e Number

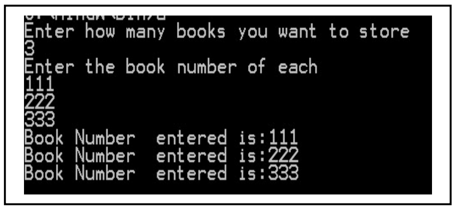
```
#include <stdio.h>
struct book {   int Number;   };
int main(){
        int n; int i;
        struct book  r[10];
        printf("Enter how many books you want to store\n");
        scanf("%d",&n);
        for(i=0;i<n;i++){
                printf ("book number is: %d", r[i].Number);
        }
        return 0;  }
```



```
Enter how many books you want to store
3
book number is: 1967688917
book number is: 1019192611
book number is: -2
```

Coding Example_2: Initializing the structure member

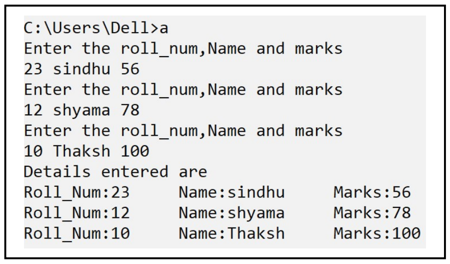
```
#include <stdio.h>
struct book{    int Number;    };
int main(){
        int n; int i;
        struct book r[10];
        printf("Enter how many books you want to store\n");
        scanf("%d",&n); // assumption n < =10
        printf("Enter the book number of each\n");
        for( i=0;i<n;i++){
            fflush(stdin);
            scanf("%d",&r[i].Number);
        }
        for(i=0;i<n;i++)
              printf("Book Number entered is:%d\n",r[i].Number);
        return 0;
}
```



```
Enter how many books you want to store
3
Enter the book number of each
111
222
333
Book Number  entered is:111
Book Number  entered is:222
Book Number  entered is:333
```

Coding Example_3: Read and display the details of n=3 students

```
#include<stdio.h>
struct student  {    int roll_no;  char name[30]; int marks;     };
int main(){
        struct student s[100];
        int i;
        for( i=0;i<3;i++) {
                printf("Enter the roll_num, Name and marks\n");
                scanf("%d%s%d",&s[i].roll_no,s[i].name,&s[i].marks);  //for Name variable no &
         }
        printf("Details entered are\n");
        for(i=0;i<3;i++) {
                printf("Roll_Num:%d\t",s[i].roll_no); pr intf("Name:%s\t",s[i].name);
                printf("Marks:%d\n",s[i].marks);  }
  return 0;   }
```



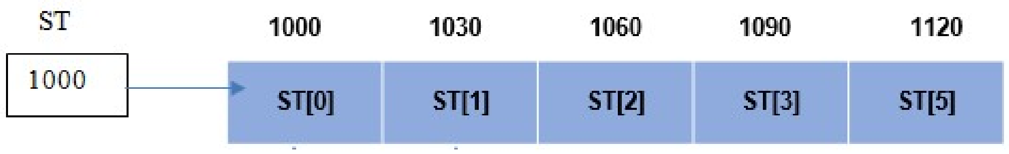
```
C:\Users\Dell>a
Enter the roll_num,Name and marks
23 sindhu 56
Enter the roll_num,Name and marks
12 shyama 78
Enter the roll_num,Name and marks
10 Thaksh 100
Details entered are
Roll_Num:23     Name:sindhu     Marks:56
Roll_Num:12     Name:shyama     Marks:78
Roll_Num:10     Name:Thaksh     Marks:100
```

Pointer to an Array of Structures

Pointer is used to access the array of structure variables efficiently. Suppose we have to store record of 5 students, then using array of structure it can be easily done as:

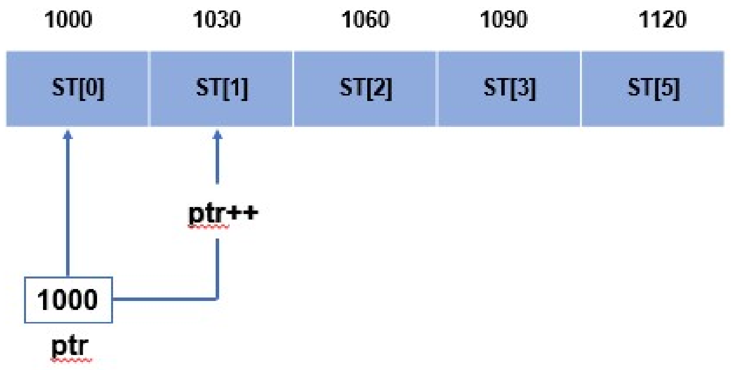
```
struct student
{       int roll_no;
        char name[30];
        int marks;
}ST[5];
```

The sizeof operator when applied on structure student will take at least 30 bytes in memory.(Please note this is implementation specific). Suppose, base address of the array ST is 1000, then pictorial representation of the same in memory will be:



Now, if the pointer ptr initialized to ST as: **struct student *ptr = ST;**

It means that the pointer ptr is of type struct student and is holding the address pointed by the array of structure ST (which is 1000).



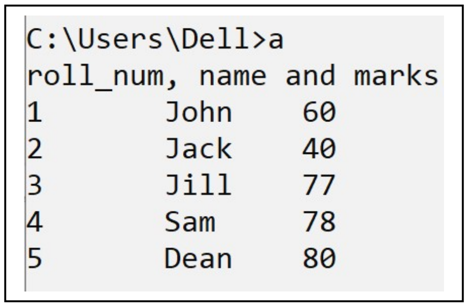
Suppose if we perform ptr++, the ptr gets updated and now it will start pointing to next array record starting from 1030 (ptr++ → ptr+1 → 1000 + 30 = 1030), which is the record of next student.

Coding Example_4:

```
struct Student ST[] = {{1, "John", 60}, {2, "Jack", 40}, {3, "Jill", 77}, {4, "Sam", 78 }, {5, "Dean", 80}};
struct Student *ptr = ST;        //&ST is illogical
printf("\n%d\n", *ptr);          //prints 1
printf("%d \t %s \t %d\n", ptr->Roll_no, (ptr+1)->Name, (ptr+2)->Marks); //prints 1 Jack 77
ptr++;          // ptr now points to ST[1]
printf("\n%d\n", *ptr);          //prints 2
```

Coding Example_5: How we can use the pointer ptr to print all the details of all students.

```
struct student *ptr = ST;
int i;
printf("roll_num, name and marks\n");
for(i = 0; i < (sizeof(ST)/sizeof(ST[0])) ; i++)
{
        printf("%d\t",(ptr+i)->roll_no);   // ptr[i].roll_no
        printf("%s\t", (ptr+i)->name);
        printf("%d\n",(ptr+i)->marks);
}
```



```
C:\Users\Dell>a
roll_num, name and marks
1        John    60
2        Jack    40
3        Jill    77
4        Sam     78
5        Dean    80
```

Passing an Array of structure to a function

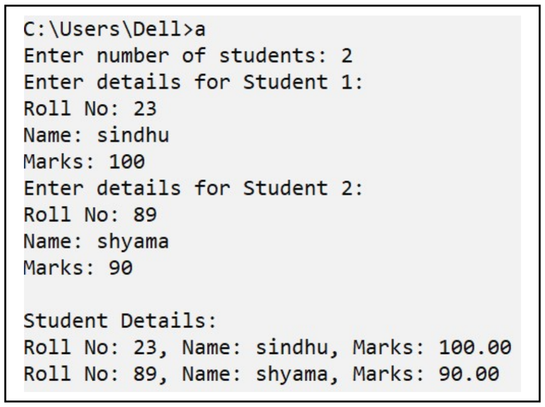
Structures allow grouping of related data under a single name. When multiple records of such structured data (like multiple students, employees, products, etc.) need to be managed, an array of structures is used. To process this data efficiently, we often need to pass the entire array of structures to a function. This approach promotes modularity and avoids repetitive code. **Passing an array of structures to a function allows the function to access and modify the original data directly.**

Consider,

```
// Structure definition
struct Student {   int roll_no;   char name[50];   float marks;  };
```

Coding Example_6: Read and display the details of n=3 students using two functions – read() and display()

```
// Function to read input for all students
void read(struct Student *s, int n) {
    for(int i = 0; i < n; i++) {
        printf("Enter details for Student %d:\n", i + 1);
        printf("Roll No: ");
        scanf("%d", &s[i].roll_no);
        printf("Name: ");
        scanf(" %[^\n]", s[i].name);  // to read full name with spaces
        printf("Marks: ");
        scanf("%f", &s[i].marks);
    }
}

// Function to display all students
void display(struct Student *s, int n) {
    printf("\nStudent Details:\n");
    for(int i = 0; i < n; i++) {
        printf("Roll No: %d, Name: %s, Marks: %.2f\n", s[i].roll_no, s[i].name, s[i].marks);
    }
}

int main() {
    struct Student s[1000];
    int n;
    printf("Enter number of students: ");
    scanf("%d", &n);
    read(s, n);     // Reading data   - This array degenerates to a pointer at runtime
    display(s, n);  // Displaying data – Here too, an rray degenerates to a pointer at runtime
    return 0;
}
```



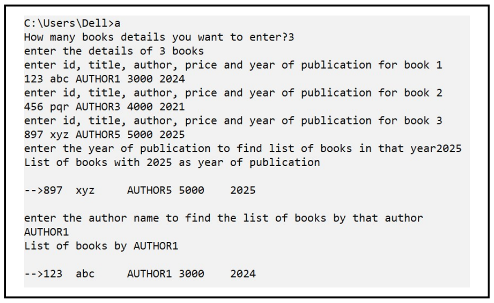
```
C:\Users\Dell>a
Enter number of students: 2
Enter details for Student 1:
Roll No: 23
Name: sindhu
Marks: 100
Enter details for Student 2:
Roll No: 89
Name: shyama
Marks: 90

Student Details:
Roll No: 23, Name: sindhu, Marks: 100.00
Roll No: 89, Name: shyama, Marks: 90.00
```

Coding Example_7: The Book entity contains these data members**: id, title, author, price, year of publication**. Write separate functions to **read details of n books and display it.** Also include functions to do the following.

- **Fetch all the details of books published in the year entered by the user.**
- **Fetch all the details of books whose author name is entered by the user. Display appropriate message if no data found.**

```
typedef struct Book {
        int id;         char title[100];        char author[100];       int price;      int year;
    }book_t;

#include<stdio.h>
#include<string.h>
// Function prototypes
void read_details(book_t *b,int n);
void display_details(book_t *b, int n);
int fetch_books_year(book_t *b,int n, int year, book_t*);
int fetch_books_author(book_t *b,int n, char *author, book_t*);

// Client Code
int main()
{
        book_t b[100];
        book_t b_year[100];
        book_t b_author[100];
        printf("How many books details you want to enter?");
        int n;          scanf("%d",&n);
        printf("enter the details of %d books\n",n);    read_details(b,n);
        int count;
        printf("enter the year of publication to find list of books in that year");
        int year;       scanf("%d",&year);
        count = fetch_books_year(b,n, year,b_year);
        if(count){
                printf("List of books with %d as year of publication\n",year);
                  display_details(b_year, count);
        }
        else
                printf("books published in %d is not available in the dataset\n",year);
        printf("\n");
        printf("enter the author name to find the list of books by that author\n");
        char author[100];       scanf("%s",author);
        count = fetch_books_author(b,n, author,b_author); if(count)
        if (count){
                printf("List of books by %s\n",author);
                display_details(b_author, count);
        }
        else
                printf("books by %s is not available in the dataset\n",author);
        return 0;
}

// Function to read book details
void read_details(book_t *b,int n){
        int i;
        for(i = 0; i< n;i++)
        {
                printf("enter id, title, author, price and year of publication for book %d\n",i+1);
                scanf("%d%s%s%d%d",&b[i].id, b[i].title, b[i].author, &b[i].price, &b[i].year);
                // &((b+i)->id) is also valid in scanf. Using the pointer notation
        }
}
```

```
// Function to diaplay book details
void display_details(book_t *b, int n){
        int i;
        for(i = 0; i< n;i++)
        {       printf("\n-->%d\t%s\t%s\t%d\t%d\n",b[i].id, b[i].title, b[i].author, b[i].price,
b[i].year);  // (b+i)->id is a valid pointer notation to print id
        }
}

// Function to fetch count of books with the given year
int fetch_books_year(book_t *b,int n, int year,book_t *b_year)
{       int i; int count = 0;
        for(i = 0; i< n;i++)
        {       if (b[i].year == year)          {   count++;    b_year[count-1] = b[i];   }
        }
        return count;
}

// Function to fetch count of books by the given Author
 int fetch_books_author(book_t *b,int n, char *author, book_t *b_author) {
        int i;    int count = 0;
        for(i = 0; i< n;i++)
        {       if (!(strcmp(b[i].author, author)))
                {
                        count++;        b_author[count-1] = b[i];
                }
        }
        return count;
}
```



```
C:\Users\Dell>a
How many books details you want to enter?3
enter the details of 3 books
enter id, title, author, price and year of publication for book 1
123 abc AUTHOR1 3000 2024
enter id, title, author, price and year of publication for book 2
456 pqr AUTHOR3 4000 2021
enter id, title, author, price and year of publication for book 3
897 xyz AUTHOR5 5000 2025
enter the year of publication to find list of books in that year2025
List of books with 2025 as year of publication

-->897  xyz     AUTHOR5 5000    2025

enter the author name to find the list of books by that author
AUTHOR1
List of books by AUTHOR1

-->123  abc     AUTHOR1 3000    2024
```

Keep Coding using Array of Structures!!!