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

Lecture Notes
Problem Solving With C
UE24CS151B

Lecture #11
Structures and Functions

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: Structures and Functions

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

Structures are used to group different types of data into a single logical unit, while **functions** allow breaking the program into smaller and reusable blocks of code. When we combine these two, we get a powerful way to **pass, manipulate, and return complex data** efficiently. Also, we can build modular, organized, and reusable code using structures and Functions together.

.

Passing a structure to a function

Functions typically receive data through arguments. When dealing with multiple related data items, structures offer a clean way to group them. To operate on such grouped data, we can **pass entire structure or it's address to functions.** Depending on the use case, structures can be passed to a function which **copies the entire structure or it's address only**. Also, function can return a structure to the calling function.

Consider the Player structure.

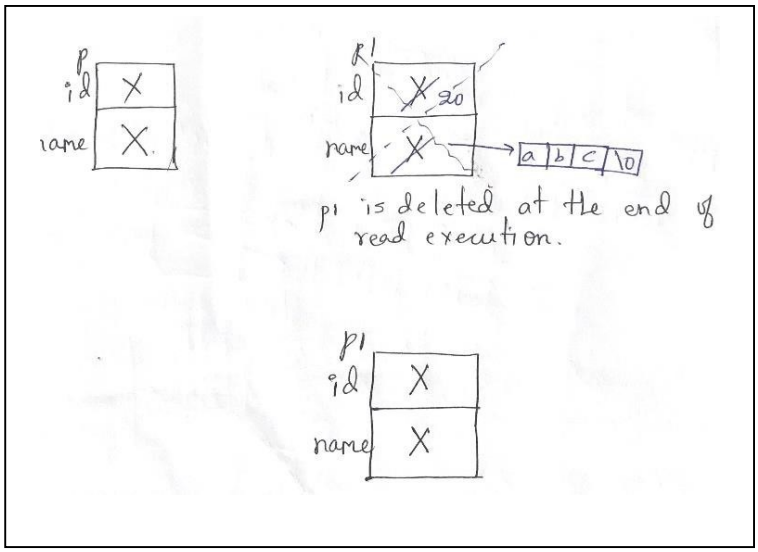
```
struct Player {
        int id;
        char name[20];
};
```

Coding Example_1: Function to read and display the player details. Does this code work accordingly? Think!!

```
int main(){
        struct Player p;
        printf("Enter id and name\n");
        read(p);        disp(p);
        return 0;
}
void read(struct Player p1){
        scanf("%d ", &p1.id);           // user enters 20  and abc
        scanf("%[^\n]s", p1.name);
}
```

```
void disp(struct Player p1){
        printf("%d\n",p1.id);          // undefined values get printed
        printf("%s",p1.name);
}
```

Parameter passing is always by value in C. The argument p is passed to the parameter p1 in read function. This is pass by value and as the structure is involved, it is member-wise copy. Whatever the user enters, it is stored in p1, not p. p1 is a parameter and it is lost once after the completion of read( ) execution. The same p is passed to display function. Again pass by value and member-wise copy. So the output is undefined values only. **Pictorial representation is as below.**



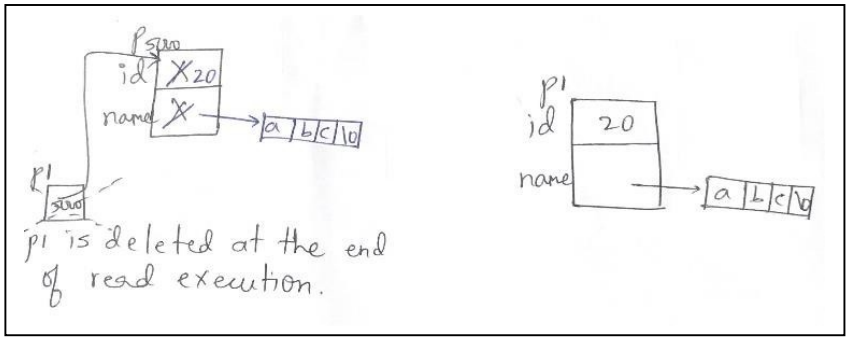
If you want to modify the structure inside a function, pass a pointer to a structure(l-value) as an argument.

Coding Example_2:

```
int main( ){
        struct Player p;
        printf("Enter id and name\n");
        read(&p);      disp(p);/ / Thin    k, can we also pass &p to disp(). Is there any harm?
        return 0;
}
```

```
void read(struct Player *p1) {
        scanf("%d ", &(p1->id));       // or &((*p1).id)        // user enters 20
        scanf("%[^\n]s", p1->name);            // abc entered
}
void disp(struct Player p1) {
        printf("%d ",p1.id);    // actual values will be printed.
        printf("%s",p1.name);
}
```

Parameter passing is always by value in C. The argument &p is passed to the parameter p1(pointer to structure) in read function. The parameter p1 is pointing the p structure. Whatever the user enters, it is stored in p, through p1. p1 is a parameter and it is lost once after the completion of read( ) execution. The **updated p is passed to display function. Again pass by value and member-wise copy of p to p1 in the display( ).** So the output is updated values in p. **Pictorial representation is as below.**

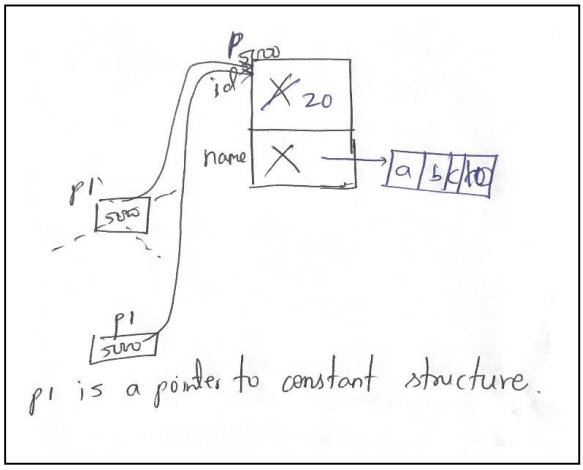


Assume the struct Player contains many data members. Can we just use structure variable in the parameter of disp()? If we do so, this is considered as a bad practice. As every member of argument is copied to every member of parameter(member-wise copy), it requires more space and more time. Also, when we are very sure that we do not want to make any changes to the argument, **parameter of disp() can be a pointer to constant structure.**

Coding Example_3:

```
int main( ){
        struct Player p;          printf("Enter id and name\n");
        read(&p);      disp(&p);      return 0;
}
```

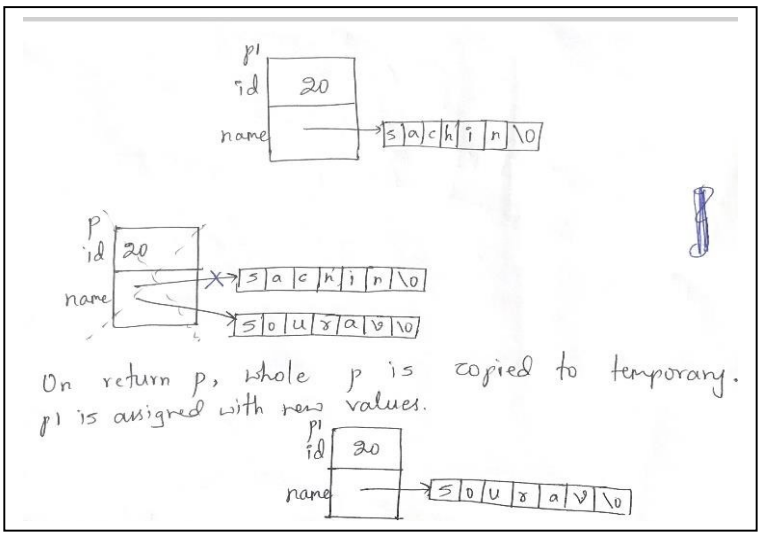
```
// read() remains the same as previous
void disp(const struct Player* p1)
{
        // p1.id = 34;                  // throws error as p1 is a pointer to constant structure
        printf("%d ",p1->id);           // actual values will be printed.
        printf("%s",p1->name);
}
```



Coding Example_4: Function returning the structure to the calling function.

```
struct Player modify(struct Player);
int main()
{
        struct Player p1={20,"sachin"};
        printf("before change %s",p1.name);
        p1=modify(p1);
        printf("after change %s",p1.name);
        return 0;
}
struct Player modify(struct Player p)
{
        strcpy(p.name, ,"Sourav");
        return p;
}
```

When the function modify() returns a changed parameter, it is copied to a temporary. Then in the client code, the temporary is assigned to p1.



Think about it!!

- Can we make const struct Player p as the parameter for read?
- Can we say struct player *p as the parameter without changing the client code?
- If we change the client code, can we use struct player *p as the parameter?
- Can we return pointer to structure?
- Can we use the const qualifier with the return type of the read( ) ?

Keep exploring Structures and Functions together!!