Structures

Structures

Structure is a method of packing data of different types, a convenient method of handling a group of related data items of different data types.
A structure type is defined by

```
struct struct-name
{
        type field-name;
        type field-name; ...
}
```

Example:

```
struct lib_books
{
        char title[20];
        char author[15];
        int pages;
        float price;
};
```

Another example:

```
struct student_type
{
        char name[20];
        int ID;
}
```

Structures

the keyword struct declares a structure to holds the details of four fields namely title, author pages and price.

Each member may belong to different or same data type.

The structure we just declared is not a variable by itself but a template for the structure (a type)

We can declare variables of this structure by

struct lib_books book1,book2,book3;

declares book1,book2,book3 as variables of type struct lib_books each declaration has four elements of the structure lib_books..

You can make the the structure into a type:

typedef struct lib_books

Accessing Structure members Directly

to declare a variable of the Structure and access a member within the class, use the operator '.' Which is known as dot operator or period operator.

```
lib_books book1;

book1. price = 15;
strcpy(book1.title,"basic");
strcpy(book1.author,"Balagurusamy");
book1.pages=250;
```

****Lesson-12.1**

Accessing Structure members Indirectly

To declare a pointer variable of the Structure and access a member within the class, use the operator ‘->’.

```
lib_books  book1,*ptr_book1;

ptr_book1-> ptr_book1-> price = 15;
strcpy(ptr_book1-> title,”basic”);
strcpy(ptr_book1-> author,”Balagurusamy”);
ptr_book1-> pages=250;
```

****Lesson-12.2**

****Create an Employee managing program**

Dynamic Memory Allocation of Structures

```
/* allocating a struct with malloc() */
struct my_struct *s = NULL;
s = (struct my_struct *)malloc(sizeof(*s));  /* NOT sizeof(s)!! */
if (s == NULL) {
  printf(stderr, “no memory!”);
  exit(1);
}

memset(s, 0, sizeof(*s));

/* another way to initialize an alloc’d structure: */
struct my_struct init = {
  counter: 1,
  average: 2.5,
  in_use: 1
};


/* when you are done with it, free it! */
free(s);
s = NULL;
```

malloc() allocates n bytes

Why?

Always check for NULL.. Even if you just exit(1).

malloc() does not zero the memory, so you should memset() it to 0.

Use pointers as implied in-use flags!