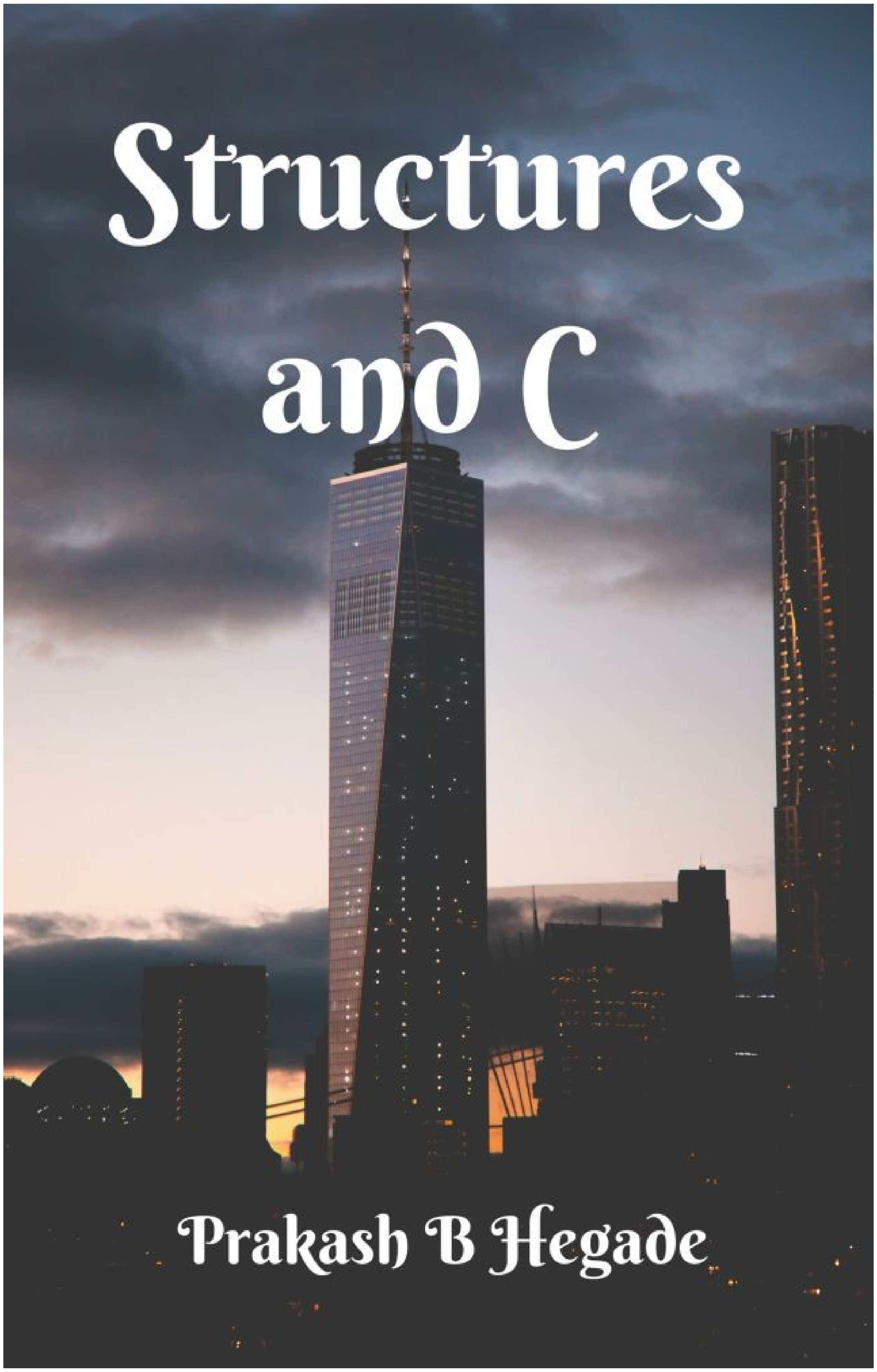


# Structures and C

Prakash B Hegade

~*~*~*~*~*~*~*~

**Structures and C**

By Prakash Hegade

**Smashwords Edition**

~*~*~*~*~*~*~*~

~*~*~*~*~*~*~*~

For any query or questions kindly mail: **prakash.hegade@gmail.com**

~*~*~*~*~*~*~*~

# Structures and C

**Prakash B. Hegade**

**First Edition**

The programs used in the book are made available online. They can be downloaded and used from the following Github link:

**https://github.com/prakashbh/structures-and-c-ebook-programs**

## The Engine

## Before Getting Started

### Introduction

‘Does understanding structures make us a better programmer?’ ask this question to a programming expert and one might think and give a detailed answer. If you put the question this way: ‘Does understanding structures clearly and in dept make us a better programmer?’, then there would be a quick ‘yes’ with no second thoughts.

Understanding ‘structures’ has given stunning buildings and architectural wonders. Understanding ‘structures’ has explained us how to visualize the universe and the planetary system. Understanding ‘structures’ has given out patterns and designs. Structures have made the study easy and involving. Understanding ‘structures’ will also make you a better programmer. Though we are referring to different structures in the context, they stand important in their respective domains.

This book assumes that the reader has all the required C pre-requisites before an attempt is made to read. Pre-requisite covers the concepts: data types, operators, statements, functions, array, pointers and needless to mention, the related concepts to the said ones.

Sit with an IDE of your comfort and code where ever possible. Working with the code and experimenting is the best way to learn programming. The concepts are built incrementally in a very slow pace so that it will be easy to understand and realize the concepts in a right direction. We have just ignited the engine and the journey through several bogies will take you through the structures and its aspects. And yeah, it's a passenger train!

The book covers the concepts with programs where ever necessary. There are ample amount of examples and exercise questions to make the concept clear. There is also an exercise provided at the end of the book. For any other questions or discussion you can write to me at my email address provided in the page above.

- **Prakash B. Hegade**

## Bogie Number 01

## Ladies and Gentlemen, Please Welcome,

## Structures

Let me tell you my story. It’s not a romantic or a thriller. It’s rather an inspirational one. Let’s go back to time where people had only arrays to work with. I mean, okay! Homogeneous collection of data! All int’s together, all float together, all double together. LOL. It was obvious to feel the insufficiency in what we had.

Let us take an example. You get admission to a college after clearing the entrance and you walk with several set of documents. The college admission department maintains several procedures and collects some certificates to return back at the time of graduation. Let us say there are like ten kinds of documents to be maintained for every student. Institute takes a decision that all documents of same kind will be put together in a single file. All transfer certificates one file, all fee receipts one file etc and so on. Then comes the days where students wanted to collect back their certificates. Every student had to visit nearly ten different files. Wow! Now that was weird and students started cursing the management. Management understood the problem and took a decision that all documents relating to one student will be in the same file.  Access was easy. The method was found to be beneficial and everyone was happy.

In the scenario explained above the first method was an array. The second was me AKA structures. I combine the heterogeneous data and put together in the same block. You can see that real time data has components of all kinds. Array does not suffice to be an ideal solution. A better management of heterogeneous data is to use the structures.

The motivation and take away is that whenever we need to deal with heterogeneous data, I am the solution! Let us get into the technical aspects. We need to understand how to work with me in a way machine can understand. There is a difference between the way you see and the way machines see. The next bogie will deal with definition and syntactical aspects of structures and so on.

Enjoy the reads. Learn more about me! Here I sign off! - Structures!

## Bogie Number 02

## The Get Together Party

### Definition

Let us see the process by which the get-together of various data types is going to be celebrated. Firstly, we are going to define a new data type. We need a keyword to represent this data type. The way we say int, float, char, double etc we are going to call this as ‘struct’. ‘struct’ from structures, makes sense right? Every data type of this kind will start with the keyword ‘struct’.

Do you realize how the grouping happens in C language? Like, when you want to combine several things into one, we use the flower braces. For example we can see that statements inside the functions are grouped together using { and }. In the same way we are going to combine the various data types into one single structure using flower braces.

What we have so far is the keyword ‘struct’ and we are going to combine the different data types using flower braces. Let us see how it looks.

```
struct
{
      data_type member-1;
      data_type member-2;
      …
      data_type member-n;
}
```

Let's play the guess game. What do you think is missing?  Look closely. Does something look odd and fishy? Let's compare it with a basic data type definition example like

**int number;**

As every structure is going to be different in its construction and we would require a name to associate to it. The structure we have defined so far needs to be identified with a unique name. We also need a mechanism to define variables for the defined structure and a semi-colon to mark the end. Let us look at the refined complete syntax to define the structure.

```
struct structure_name
{
    data_type member-1;
    data_type member-2;
    ...
    data_type member-n;
} structure_variables;
```

Now we have the complete syntax for the structure. Let us create some examples. For now let us skip the part of creating variables. Let us only put down the relevant structures for the given scenarios.

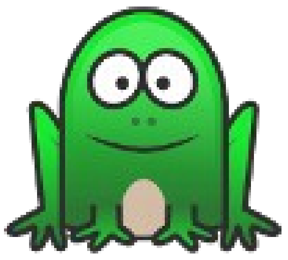
**Example 01:** How about a structure to capture the students who bunk the class? Let us have the details of semester, number of students and reason for the bunk. Let us have a component to write down the punishment as well.

```
struct bunk_class
{
        int sem;
        int no_of_students;
        char reason[20];
        char punishment[20];
} ;
```

**Example 02:** Let us write one more structure to capture the details of a person for unique identity along with his basic details. Let us say the identity is confirmed by any government valid identify proof. It could be driving license, voter ID, etc. Let us have a field for contact where it can be a mobile number, landline number, email id or any of the social networking handles which the person is active on and could be easily contacted.

```
struct person_details
{
        char name[30];
        char DOB[20];
        char sex[8];
        int age;
        float height;
        float weight;
        char unique_id[30];
        char permanent_address[50];
        char contact_id[30];
};
```

**Example 03:** Can you write the structure for the image given below? If we keep the image and the structure side by side, the structure has to be a textual representation of the image. Some fields are done for you. Complete the remaining.



```
struct frog
{
        char color[10];
        int number_of_legs;
        …
        [fill up the remaining]
};
```

**Example 04:** Pick up a favorite scene from the movie of your choice. Can you write a structure to capture the details of that scene in a structure? You can give a try! Few things are done for you already!

```
struct movie_scene
{
        char movie_name[30];
        …
        [fill up the remaining]
};
```

**Example 05:** Write a structure to capture the details of your browser. Some data members are done for you already.

```
struct browser
{
        char name[20];
        float version;
        char owner[20];
        …
        [fill up the remaining]
};
```

We have now learnt how to create user defined data types and they are not difficult!

## Bogie Number 03

## Making Structures Usable

### Declaring Variables

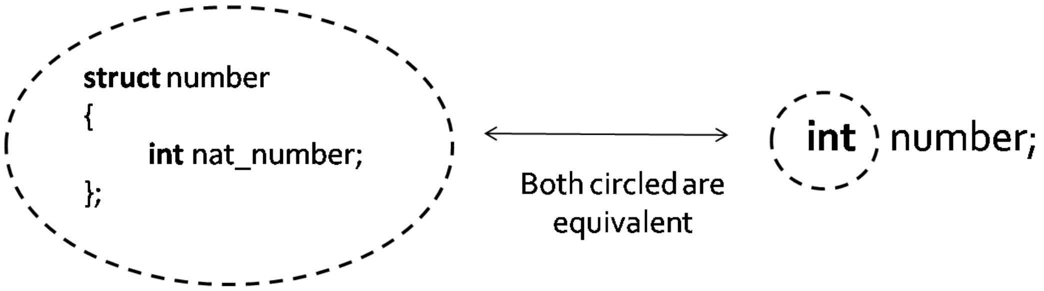
Well, let us see how to declare variables to the structures we create. We have already had the insights of it in bogie number 02. Here we shall see some examples. To make a comparison with

" **int number;** "

what we have so far is equivalent to the **int** part. We now need to create the variables. Not clear? Look at this example. Let us say we got addicted so much to structures that even to define a number we have created a structure.

```
struct number
{
        int nat_number;
};
```

Now let us make a comparison.



You get it right? We only have the data type so far. We don't have the variables yet. Consider the structure we created earlier:

```
struct bunk_class
{
        int sem;
        int no_of_students;
        char reason[20];
        char punishment[20];
} ;
```

What do you think is memory allocated to this? ehh? Did start counting? What is the memory allocated if we just say 'int'? Zero right? We don't have an allocated memory until we create variables. So is for the structures. Memory is allocated only when we declare the variables to it.

Variables can be created in two ways. We can create along with the structure definition or separately later. Both the ways will work.

**One way:**

```
struct bunk_class
{
        int sem;
        int no_of_students;
        char reason[20];
        char punishment[20];
} day1, day2;
```

**Another way:**

**struct** bunk_class day3;

If students had bunked for several days and if you need an array variable, the syntax is the way exactly we did for array variables.

**struct** bunk_class days[10]; → now we can capture the details of 10 bunked classes. Well if all the ten are populated, class deserves a good punishment. What say? Or maybe the professor isn't good! :-P

Let us take one more example. Let us create a structure for iPod and create 2 variables one of each way.

```
struct iPod
{
        char type[20];
        float cost;
        float memory;
        int id;
} iPod1;
struct iPod iPod2;
```

## Bogie Number 04

## Give It a Short Name I say!

### Type Defining the Structures

Say you run a company and you are creating a structure for every department in your company.

```
struct my_company_department_1  {
        char department_name[30];
        int id;
        char manager[20];
        int employees;
};
```

Likewise you create a structure for each department. Okay. Fine! Let us now say under each department there are ‘n’ numbers of divisions and we need a structure for each. Here is how it goes:

```
struct my_company_department_1_division_1 {
        ….
};
```

Okay, that structure name is already irritating. Adding to all this now you are also supposed to create a structure for each team under division! What??

```
struct my_company_department_1_division_1_team_1 {
        ….
};
```

Do you see how tedious it would be to refer to this structure with such a huge name? So why, we have the typedef's. We can type define a smaller name to the structure using a '**typedef**'. We can again achieve this in two ways. Let us look at same bunk_class example.

```
struct bunk_class
{
        int sem;
        int no_of_students;
        char reason[20];
        char punishment[20];
};
typedef struct bunk_class BS;
```

We can now refer to this structure using the short name BS. The standard convention is to use all uppercases for the type defined names. To create a variable we can now only say:

```
BS day1;
```

**OR** yet another way is that we need not name the structure and directly typedef it using the following syntax:

```
typedef struct
{
        int sem;
        int no_of_students;
        char reason[20];
        char punishment[20];
}BS;
```

However, the suggestion would be to know the second approach and use the first one. In that way we can give a long meaningful name to the structure and use a short form of it.

**Case study:**

Have you worked with files in C? Here is something to know. Let us look at the structure of the file pointer. It is defined in the library in the following way:

```
typedef struct
{
        short level ;
        short token ;
        short bsize ;
        char fd ;
        unsigned flags ;
        unsigned char hold ;
        unsigned char *buffer ;
        unsigned char * curp ;
        unsigned istemp;
} FILE ;
```

Ahh! Now, do you understand why we create a file pointer variable as **FILE *fp;** ? Yes. The structure is type defined to the name **FILE**.

## Bogie Number 05

## Let’s give some Value!

### Initialization

To initialize an integer number we do the following:

```
int number = 31;
        OR
int number;
number = 31;
```

This notion holds well with structures too. We can initialize in both the ways. Let us learn it through the running example of bunk_class.

```
struct bunk_class
{
          int sem;
          int no_of_students;
          char reason[20];
          char punishment[20];
} day1 = {3, 70, “Tired”, “Severe”};
```

**OR**

```
struct bunk_class day2 = {3, 70, “Bored”, “Severe”};
```

Here are the rules to remember while initializing the structures. They are pretty straight forward and logically make sense.

**Rule 01:** The members of the structure cannot be initialized in the structure definition.

```
struct bunk_class
{
        int sem = 3; → INVALID
        int no_of_students;
        char reason[20];
        char punishment[20];
};
```

How odd is that? Just like we don’t say **int = 3 number;** LOL.

**Rule 02:** The initial values are assigned to members of the structure on one-to-one basis. Yes, like an injective function.

**Rule 03:** During partial initialization the values are assigned in increasing member order and the remaining is initialized with default values.

```
struct bunk_class
{
        int sem;
        int no_of_students;
        char reason[20];
        char punishment[20];
};
```

```
struct bunk_class day2 = {3, 70, “Bored”};
```

This is valid. Punishment can be populated later after having a meet with the principle and discussing an appropriate one!

**Rule 04:** During initialization the number of initializers should not exceed the number of members. True to the point!

**Rule 05:** There is no way to initialize in the middle of a structure without initializing the previous members. Let us say the professor is strict and for any kind of bunk he gives a severe punishment. He can do the following:

```
struct bunk_class
{
        int sem;
        int no_of_students;
        char reason[20];
        char punishment[20];
};
struct bunk_class day2 = {0, 0, “NULL”, “Severe”};
```

**Example:** Can you define and initialize a variable for the below structure:

```
struct my_company_department_one {
        char department_name[30];
        int id;
        char manager[20];
        int employees;
};
```

## Bogie Number 06

## Let's Talk to Members

### Accessing Members

It is very important that we have to know how to access the members of the structures. There will be times where we will have to use one or two members of the structures and operate only on them. Look at the below structure for example:

```
struct student
{
        char name[20];
        int roll;
        float exam1_marks;
        float exam2_marks;
        float exam3_marks;
        float avg_marks;
};
```

From the defined structure when we want to print only the student name and average marks, we will have to access only those members. The process is simple.

**Question:** Who knows the details of the structure members?

**Answer:** Structure Variable.

And so why we use a structure variable to access the structure members and operate on them. We need a separator between the two. If the variable is 'value' variable we use a dot (period) operator to separate a variable name and member name and an arrow operator (->) in case of ‘pointer’ variable.

**Example:**

```
struct bunk_class
{
        int sem;
        int no_of_students;
        char reason[20];
        char punishment[20];
} day1, day2;
```

For the above structure we can access the members as follows:

- day1.sem
- day1.reason
- day2.punishment

## Bogie Number 07

## Valid and Invalids

### Operations on Structures

Following are the things to keep in mind while operating on structures:

**Comparison**

1. Two structure variables cannot be compared. Whether they are of same type of dissimilar, they can't be.
2. Members of two structure variables of same type can be compared using relational operators.

**Copying 2 structure variables**

1. One structure variable can be assigned to another structure variable if and only if both of them have the same data type.
2. It can be done using the assignment operator (=).

**Arithmetic operations on structures**

1. Members of structures are similar to any other variables.
2. Any operation that can be performed on a variable, the same operation can be performed on structure members as well.

## Bogie Number 08

## Homogeneous inside the Heterogeneous

### Array within Structures

Can we have arrays as structure members? Well, we need not ask that question. Yes. We can, as arrays are like any other variables. Let us see this through an example. In that way we can also put all the concepts learnt so far together.

**Example:** Write a structure for browser and a program to take user input for three browsers and output the captured details.

```
#include <stdio.h>
#include <stdlib.h>

struct browser
{
        char name[20];
        char owner[20];
        float version;
        char add_ons[3][20];
};

int main()
{
  int i;
```

*Note that the members: name, owner and add_ons are the arrays within the structure*

```
  for(i = 0; i< 3; i++)
  {
    printf("Enter the Browser name, owner, version, and 3 add-ons available\n");
    scanf("%s", b[i].name);
    scanf("%s", b[i].owner);
    scanf("%f", &b[i].version);
    scanf("%s", b[i].add_ons[0]);
    scanf("%s", b[i].add_ons[1]);
    scanf("%s", b[i].add_ons[2]);
  }

  printf("The entered data is .. \n");
  for(i = 0; i< 3; i++)
  {
    printf("Browser Name: %s\n", b[i].name);
    printf("Owner : %s\n", b[i].owner);
    printf("Version : %f\n", b[i].version);
   printf("Addons : %s\t%s\t%s\n\n", b[i].add_ons[0], b[i].add_ons[1],
                       b[i].add_ons[2]);
  }
  return 0;
}
```

Let us also learn how to initialize such structures statically. Below is a provided example.

```
struct browser
{
        char name[20];
        char owner[20];
        float version;
        char add_ons[3][20];
};

struct browser b[3] = {
    {"firefox", "Mozilla", 43.0, {"Reader", "startHQ", "Greasemonkey"} },
    {"chrome", "Google", 47.0, {"Honey", "Room", "WhatFont"} },
    {"opera", "Opera", 34.0, {“Guru", "Translate", "Buffer"} }
};
```

Notice how the add-ons are initialized.

## Bogie Number 09

## One inside Other inside Another…

### Structures within Structures

With growing need of more structures for a situation, having structures within structures AKA nested structures become inevitable. Either we can write structures inside a structure literally or we can have variable of one structure as a member of another structure. Let us take a simple example to understand both the cases.

**The first case:**

```
struct outer {
        int member_one;
        struct inner {
                int member_two;
        }in;
};
```

**The second case:**

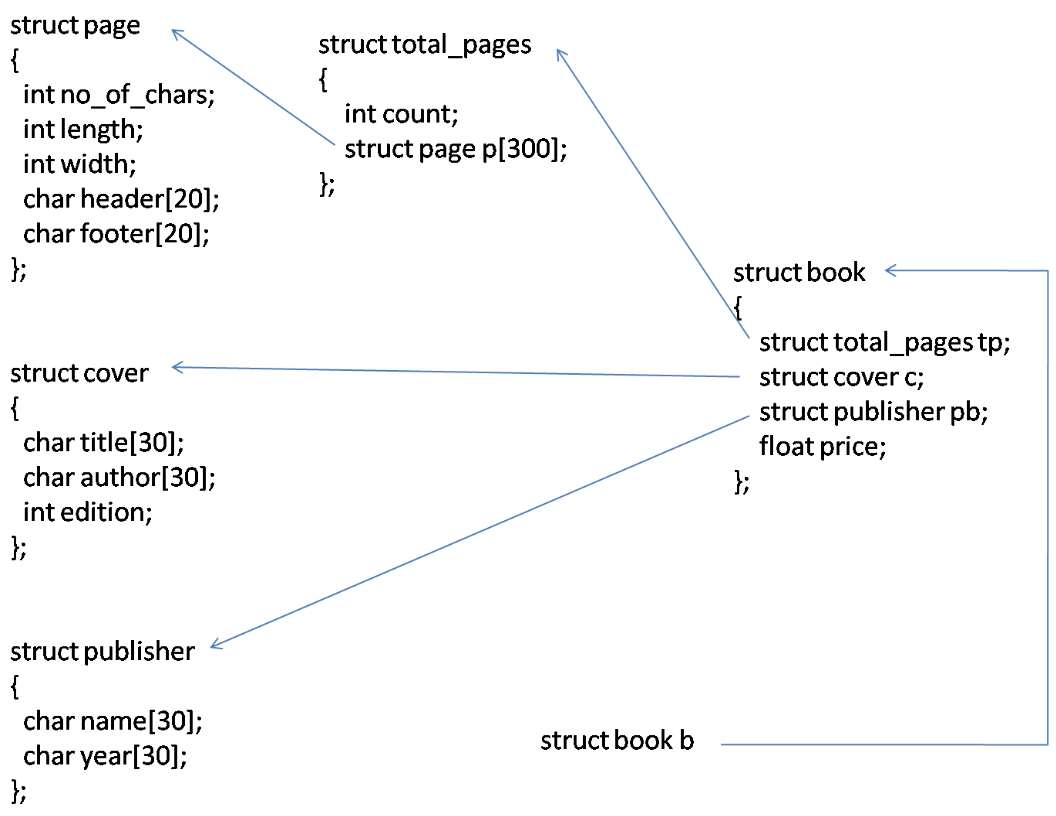
```
struct inner {
        int member_two;
};
struct outer {
        int member_one;
        struct inner in;
};
```

We shall stick to the second case. You are free to experiment and explore the limitations and advantages of the first case. Let us consider one more example to capture the date and time. We can do it systematically in the following way:

```
struct time
{
        int hh;
        int mm;
        int ss;
};

struct date
{
        int dd;
        int mm;
        int yy;
};

struct dt
{
        struct time t;
        struct date d;
};
```

Did you notice the way ‘struct dt’ members were populated? Just like any other variables we did write the data type followed by the variable name. ‘struct time’ is the data type and ‘t’ is the variable name.

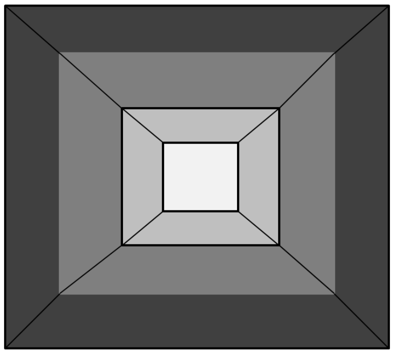
Let us understand in more detail. Consider the example below of a book structure:



```
struct page
{
  int no_of_chars;
  int length;
  int width;
  char header[20];
  char footer[20];
};

struct total_pages
{
   int count;
   struct page p[300];
};

struct book
{
  struct total_pages tp;
  struct cover c;
  struct publisher pb;
  float price;
};

struct cover
{
  char title[30];
  char author[30];
  int edition;
};

struct publisher
{
  char name[30];
  char year[30];
};

struct book b
```

At a very higher level we have ‘page’, ‘cover’ and ‘publisher’. Then we have ‘total_pages’. The structure ‘book’ finally encloses all. We have created a variable ‘b’ for the outermost structure. The question is how we can access the members in such cases.

Look at the figure below:



To reach the innermost step we need to start our walk from the outermost one. Similarly, the general rule is:

**structure_variable.outer_member.inner_member**

With the given rules, this is how we can access the said values:

```
// To access price of the book:
b.price

// To access publisher name:
b.pb.name

// To access book title
b.c.title

// To access total page count
b.tp.count

// To access number of characters in 10th page
// Page number starts from 1
```

```
  b.tp.p[10].no_of_chars
```

Write down that example structure for book clearly in your book and trace how each of the example access is done. Considering you have no doubts so far, let us look at a program example.

**Example:** Browser program to demonstrate structure within structures

```
#include <stdio.h>
struct addons
{
 char add_on1[20];
 char add_on2[20];
 char add_on3[20];
};
struct browser
{
 char name[20];
 char owner[20];
 float version;
 struct addons aon;
};

int main()
{
  struct browser b[2];
  int i;
  for(i = 0; i< 2; i++)
  {
    printf("Enter the Browser name, owner, version, and 3 add ons available\n");
    scanf("%s", b[i].name);
    scanf("%s", b[i].owner);
    scanf("%f", &b[i].version);
    scanf("%s", b[i].aon.add_on1);
    scanf("%s", b[i].aon.add_on2);
    scanf("%s", b[i].aon.add_on3);
  }
  printf("The entered data is .. \n");
  for(i = 0; i< 2; i++)
  {
    printf("Browser Name: %s\n", b[i].name);
    printf("Owner : %s\n", b[i].owner);
    printf("Version : %f\n", b[i].version);
    printf("Addons : %s\t%s\t%s\n\n", b[i].a_on.add_on1, b[i].aon.addon2,
                  b[i].aon.add_on3);
  }
  return 0;
}
```

## Bogie Number 10

## Structures Parameterize as Arguments

### Structures and Functions

Structures can be passed as arguments to the functions in the following ways:

1. Passing the individual structure members to the function by call by value or call by reference
2. Passing the structure variables to functions using pass by value
3. Passing the structure variables to functions using pass by reference

We shall understand all of the above techniques using the same example. Consider an example where a structure captures student name, effort put and marks. If there is effort and if student has scored below threshold, then the bonus marks are credited.

**Example 1:** Let us first see an example where we pass structure members as arguments.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct student
{
    char name[20];
     int effort;
    float marks;
};

void update_marks(int effort, float *marks)
{
   if(effort == 1) {
     if(*marks <=40)
         *marks =  *marks + *marks * 0.25;
  }
}

int main()
{
  struct student s1;
  strcpy(s1.name, "Vishwa");
  s1.marks = 30;
  s1.effort = 1;

  // Notice that one member is passed as call by value
  // (effort is only to check status)
  // and another by reference (as marks need to be updated)
  update_marks(s1.effort, &s1.marks);

  printf("Student Details:\n");
  printf("Name: %s\nMarks: %f\n", s1.name, s1.marks);
  return 0;
}
```

**Example 2:** Passing structures to functions using pass by value

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct student
{
 char name[20];
 int effort;
 float marks;
};

struct student update_marks(struct student s1)
{
  if(s1.effort == 1)
  {
    if(s1.marks <=40)
      s1.marks =  s1.marks + s1.marks * 0.25;
  }
  return s1;
}

int main()
{
  struct student s1;
  strcpy(s1.name, "Vishwa");
  s1.marks = 30;
  s1.effort = 1;

   // Structure is passed by call by value.
   // Hence we have the return type to hold the updated result
  s1 = update_marks(s1);
  printf("Student Details:\n");
  printf("Name: %s\nMarks: %f\n", s1.name, s1.marks);
  return 0;
}
```

**Example 3:** Passing structures to functions using pass by reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct student
{
 char name[20];
 int effort;
 float marks;
};

void update_marks(struct student *s1)
{
  if(s1->effort == 1)
  {
      if(s1->marks <=40)
          s1->marks =  s1->marks + s1->marks * 0.25;
  }
}

int main()
{
  struct student s1;
  strcpy(s1.name, "Vishwa");
  s1.marks = 30;
  s1.effort = 1;

    // Pass the address of the structure.
   // Notice the change in function call and function definition.
  update_marks(&s1);
  printf("Student Details:\n");
  printf("Name: %s\nMarks: %f\n", s1.name, s1.marks);
  return 0;
}
```

## Bogie Number 11

## The Dynamics of Freeness!

### Dynamic Memory Allocation and Structures

This is not a concept of its own. Let us just see the notational changes when we dynamically allocate memory for the structure variables. Consider the program for bunk_class structure.

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

struct bunk_class
{
   int sem;
   int no_of_students;
   char reason[50];
   char punishment[20];
};

int main()
{
   struct bunk_class *day1;
   day1 = (struct bunk_class *) malloc(sizeof(struct bunk_class));
   if(day1== NULL){
       printf("Memory Allocation Failed\n");
       return -1;
   }

   day1->sem = 3;
   day1->no_of_students = 50;
   strcpy(day1->reason, "Too Tired From Sports");
   strcpy(day1->punishment, "NO");

   printf("Here is what happened on day one:\n");
   printf("%d students of %d sem Bunked class for %s reason and
           got %s punishment!\n",  day1->no_of_students, day1->sem,
           day1->reason, day1->punishment);
   return 0;
}
```

**Note:**
If you want to understand about how malloc( ) works, read the Quora answer here:
**https://www.quora.com/In-a-C-program-with-the-function-int-*-malloc-sizeof-int-what-does-int-*-mean/answer/Prakash-Hegade-1**

As we can see, the only notational difference is that we don't use the dot operator to access the members. Instead we use the arrow operator. Rest of all the conventions remains the same.

## Bogie Number 12

## Do You See the Recursion?

### Self Referential Structures

Self-Referential Structures allow you to create data structures that contain references to data of the same type as themselves.

Let’s decode it!

Self – Itself

Reference - Address

Structure – We know!

So, it’s a structure which has something referring to itself. Yes, in a very much way there is a recursion.

**Example:**

```
struct list
{
        int data;
        struct list *next;
};
```

In the example we can notice that the variable ‘next’ is a pointer of data type ‘struct list’. We have used the data type before we could complete the definition of ‘struct list’ and compiler will not throw any errors or warnings for this. Once when you start learning about ‘linked list’ data structures you will appreciate the self referential structures.

## Bogie Number 13

## Who is Friends with Structures?

### Unions

We are talking about unions because they are just like structures. We will not deal with examples. We shall only see a little bit of characteristics and a simple example.

Unions,

- The size of memory allocated by the compiler is the size of the member that occupies largest space
- Memory allocated is shared by individual members of union
- The address is same for all the members of a union
- Only one member can be accessed at a time
- Only the first member of a union can be initialized

Consider the union example as below:

```
union bunk_class
{
        int sem;
        int no_of_students;
        char reason[20];
        char punishment[20];
} day1;
```

Memory allocated for the variable ‘day1’ is 20 bytes. It allocates the memory for the member with highest size. This clearly means that at any point of time there can only be one member carrying a valid value. All other operations and syntax remain same as that of structures.

## Bogie Number 14

## Let us Put Them All Together

### Examples, Programs and Exercises

In this bogie we shall see a few examples and programs to put everything together and master the concepts. There are some exercises as well.

**Q-1-size-of-structure**

**Question: Write a program to identify the size of the given structure.**

```
struct bunk_class
{
        int sem;
        int no_of_students;
        char reason[20];
        char punishment[20];
} ;
```

**Solution:**

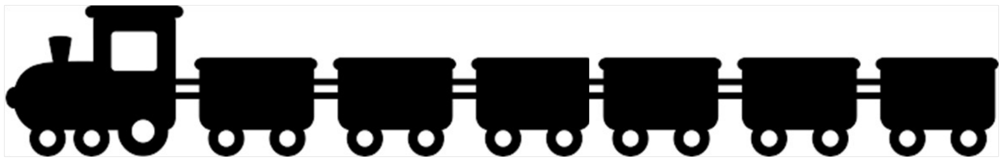
To identify the size of the structure we first need to create a variable. Then we can use the sizeof() operator and find the size. Following program does the work.

```
#include<stdio.h>
struct bunk_class
{
        int sem;
        int no_of_students;
        char reason[20];
        char punishment[20];
};
typedef struct bunk_class BC;

int main()
{
  BC day1;
  int size = 0;
  size = sizeof(day1);
  printf("Size of the structure is %d\n", size);
  return 0;
}
```

**Q-2-structure-for-train**

**Question: Design a Structure**

Design a structure which captures the details of this train. Very similar to what we did for the frog example. If we keep the structure and image side by side, one should be visual and other textual representation of same data.



**Q-3-pack-n-parcel**

**Question: Pack-N-Parcel Automation**

Pack N Parcel is a delivery agency. The different kinds of delivery made are home delivery, through mail and voice mails. The program below explains the process. This is not exactly a question. It’s a program itself to get well versed with concepts.

```
#include <stdio.h>
#include <stdlib.h>
#include<string.h>

// Function Declaration
void get_transactions();
void business();

// Structure to define home delivery
struct address
{
   int house_number;
   char house_name[20];
   char street[20];
   int pincode;
};

// Structure to define mail delivery
struct web
{
   char mail_id[30];
};
```

```
// Structure to define voice delivery
struct voice_mail
{
   int code;
   int number_part1;
   int number_part2;
};

// A structure combining all 3 delivery modes.
struct delivery_mode
{
   struct address a;
   struct web w;
   struct voice_mail v;
};

// Structure for Pack N Parcel
struct packnparcel
{
   char name[20];
   int delivery_type;
   struct delivery_mode d;
   char month[15];
   int year;
   float charges;
} p[5];
```

```
// main() calls the two functions
int main()
{
   printf("Enter the input\n");
   get_transactions();
   business();
   return 0;
}

void get_transactions()
{
   int i = 0;
   int temp_type;

    // Get 5 inputs from the user
   for(i = 0; i < 5; i++)
   {
      printf("Enter name of customer, month name, and year\n");
      scanf("%s%s%d", p[i].name, p[i].month, &p[i].year);

      // Ask for the delivery type
      printf("Enter Delivery Type: 1 - Home delivery\n2 - Web\n3-Voice Mail\n");
      scanf("%d", &p[i].delivery_type);

       // Depending on type input by user,
       // get the values for appropriate structure members
      // Make a note of how various structure members are accessed
      if(p[i].delivery_type ==  1)
      {
        printf("Enter input for House No, Name, street, Pincode\n");
        scanf("%d%s%s%d",&p[i].d.a.house_number, p[i].d.a.house_name,
                 p[i].d.a.street, &p[i].d.a.pincode);
        p[i].charges = 250.00;
      }

      if(p[i].delivery_type == 2)
      {
        printf("Enter mail Id\n");
        scanf("%s", p[i].d.w.mail_id);
        p[i].charges = 100.00;
      }

      if(p[i].delivery_type == 3)
      {
        printf("Enter code, Part1, part2\n");
        scanf("%d%d%d", &p[i].d.v.code, &p[i].d.v.number_part1,
                 &p[i].d.v.number_part2);
        p[i].charges = 200.00;
      }
   }

}
```

```
// Find out the transactions happened over the day
void business()
{
   int i = 0;
   float home_delivery_charges = 0, web_charges = 0, voice_charges = 0;
   float total_charges = 0;
   for(i = 0; i < 5; i++)
   {
      if(p[i].delivery_type == 1)
         home_delivery_charges += p[i].charges;
      else if (p[i].delivery_type == 2)
         web_charges += p[i].charges;
      else
         voice_charges += p[i].charges;
   }

    // Print the business over different modes
   printf("The Businness done over each type are..\n");
   printf("Home delivery Business = %f\n", home_delivery_charges);
   printf("Web mail Business = %f\n", web_charges);
   printf("Voice Businness = %f\n", voice_charges);

    // Print the total business
   printf("Total Business = %f\n", home_delivery_charges + web_charges
                                      + voice_charges);
}
```

**Q-4-cricket-match**

**The Cricket Match**

Consider the 2 structures as mentioned below:

```
struct information
{
   char name[20];
   int age;
   int total_score;
   char match[10];
};

struct cricket
{
   struct information info;
   int number_of_matches;
   int score[3];
   int match_type ;
}c[6];
```

Write a C program with following 2 functions:

initialize_input() → will get the input for player name, age, number of matches played, scores of 3 matches and the type of match : ODI/ Test from the user. 1 represents an ODI match and 2 for Test. Compute the total score of 3 matches and initialize the value total_score. Based on 'match_type' initialize the value for member 'match'.

Compute_top_players() → this function has to compute top 2 players of each ODI and test matches. If there are only two inputs for any case, then only the top player has to be displayed. If there is only one input to any case no result should be displayed.

Handle all conditions appropriately.

**Q-5-local-train-station**

**Question: The Local Train Station**

The local train station provides the daily train departure details. It has a registry maintained having the details of train name, number, source, destination, departure time and reach time. Provide software solution for the local train station system. Use the below mentioned structures:

```
struct time
{
  int hh;
  int mm;
  int ss;
};

struct train_details
{
  char train_name[20];
  int train_number;
  char source[20];
  char destination[20];
  struct time departure_time;
```

```
   struct time reach_time;
};
```

Support your program with following options:

-Add a new train details

-Provide all train details to passenger

-Search based on source and destination. This search should also give the total journey time, if there is a match.

**Q-6-retail-shop**

**Question:** The Retail Shop

A retail shopper is not able to keep track of his profit and loss over sold products. So he is thinking of getting a software solution. He demands, the solution should calculate the profit/loss encountered over each product he owns in his shop. Implement this task. Assume the structures mentioned below:

```
struct date
{
   int dd;
   int mm;
   int yy;
};

struct shopee
{
  char good_name[20];
  char company[20];
  float cost_price;
  float selling_price;
  struct date expiry_date;
} s[11];
```

Write functions to get input of 10 products and a function to compute a loss/profit over a product. The task is to group all the similar products with respect to name and compute the combined profit/loss. Handle all conditions appropriately.

**Partial Solution**

A partial solution is provided to you. It does not handle the case of similar products. Add the part to the program as an exercise.

```
#include <stdio.h>
#include <stdlib.h>
#include<string.h>
void get_input();
void compute();

struct shopee
{
  char good_name[20];
  float cost_price;
  float selling_price;
} s[11];

int main()
{
   printf("Enter the input\n");
   get_input();
   compute();
   return 0;
}

void get_input()
{
   int  i = 0;

   printf("Enter the following details for 10 items..\n");
   printf("Goodname\tCost price\tSelling price\n");
   for(i = 1; i <= 10; i++)
   {
      printf("Data %d\n", i);
      scanf("%s%f%f", s[i].good_name, &s[i].cost_price, &s[i].selling_price);
   }
}

void compute()
{
   int i = 0;
   float business = 0;

   for(i = 1; i <= 10; i++) {
      business =  business + (s[i].selling_price - s[i].cost_price);
      if(business < 0)
         printf("Good %s got a loss of %f\n\n", s[i].good_name, business);
```

```
    else
      printf("Good %s got a profit of %f\n\n", s[i].good_name, business);

    business = 0;
  }
}
```

**Q-7-matches-schedules**

**Question: Matches and Schedule**

Your seniors have challenged you to be ready for the first match of the season. Their team is already ready and they are waiting for you to form a team and fix the fixtures. One of your batch mates also plans to automate everything to keep the match smooth. Here is what you are supposed to do to help out:

First finalize the match details. Following structures can be used. The input for below structures has to be taken from the user clearly indicating the purpose.

```
struct date
{
  int dd;
  int mm;
  int yy;
};

struct time
{
  int hh;
```

```
   int mm;
   int ss;
};

struct stadium
{
   char name[20];
   char city[20];
   int total_seats;
};

struct match_schedule
{
   struct date d;
   struct time t;
   struct stadium st;
};
```

Next are the details of umpire, team and the players. Statically initialize the data for all the 03 following structures. There has to be 03 umpire data, 01 team data and 40 player's data.

```
struct umpire
{
   char name[20];
   int age;
   char years_of_experience;
};

struct team
{
   char team_name[30];
   char coach_name[20];
   int team_id;
   int total_players;
};

struct player
{
  char name[20];
  int age;
  char city[20];
  int matches_played;
  int total_score;
};
```

Now there is a final structure which has to be output to the challenged senior team.

```
struct match
{
   struct match_schedule ms;
   struct umpire u;
   struct team t;
   struct player list[13];
};
```

While building the structure ‘match’, consider the following:

-The first structure member ‘ms’ is given by the user

-Selecting one best umpire out of 3 based on highest years of experience

-Select top 13 players based on average score (Average score = total_score / matches_played)

-Don’t select any player aged below 17 or above 27

-Finally output all the details neatly and clearly to the user.

**Q-8-olympics**

**Question:** Olympics 2016 – Rio de Janerio

Rio de Janerio, Brazil is getting all geared up for hosting Olympics 2016. Preparations are all in progress. The Olympiad committee is also thinking of redesigning and automating a few things. Let us look at the first step from Olympics team of designing the structures in the same regard.

The first structure captures the event details.

```
struct event
{
  char name[20];
  int duration;
  int age_limit;
  float weight_limit;
  float height_limit;
};
```

Statically initialize the above designed structure for event with atleast 05 event variables like shooting, weightlifting, archery etc.

The Next two structures talk about Olympic park created to host all the events and the volunteer details.

```
struct olympic_park
{
   char city[20];
   char country[20];
   int no_of_stadiums;
   int total_capcity;
};

struct voulnteer
{
   char name[20];
   char country[20];
   char designation[20];
   int id;
};
```

Collect the input from the user for structure olympic_park. We shall use the volunteer structure later.

The opening ceremony is already planned! The details are captured in below structures.

```
struct date
{
   int dd;
   int mm;
   int yy;
};

struct time
{
   int hh;
   int mm;
   int ss;
};

struct opening_ceremony
{
   struct date d;
   struct time t;
   char venue[20];
   int no_of_audience;
};
```

Collect the relevant input from the user for all above.

And we in India are not lagging. Preparation has started rigorously so that we will win more than 06 medals in the next game.

```
struct biodata
{
   char name[20];
   char state[20];
   float height;
   float weight;
   int age;
};

struct participant
{
   struct biodata bd;
   char sport[20];
   int medals_won;
};
```

Initialize atleast 40 different participant variables statically with atleast 5 different sports for above structure.

Now there is a final structure which captures our country details.

```
struct olympics_india
{
   struct participant p[10];
   struct voulnteer v[4];
};
```

Now output the following:

-The event details given by Olympics committee

-The Olympic park details

-The opening ceremony details

-Out of 40 players from India select at-most 10 best players where there are at max two players for each described event and output them: selection has to happen based on number of medals won within the given age, height and weight limit

-Select 04 volunteers where 02 for organizers and 02 for helpers and display them.

**Q-9-college-hometown**

**Question:** The College at your Hometown

Construction of a college building at your home town is going on at a very fast pace. The architecture of the building is eye catching. Let us try to capture a few details through structures.

Let us research some accountings over the construction. Statically initialize the values for below two structures:

```
struct resources_per_mixture
{
   int bricks;
   int cement;
   int sand;
   int jelly_stones;
};

struct resource_cost
{
   float brick_price;
   float cement_price;
   float sand_price;
   float jelly_stones_price;
};
```

Bricks should be quantified with numbers and others measured per KG. Similarly initialize the cost for unit values. Next task is to find out the cost for the mixture. Use the below structure.

```
struct mixture
{
   struct resources_per_mixture rpm;
   struct resource_cost rc;
   float cost_per_mixture;
   int mixtures_required;
   float total_cost;
};
```

Calculate the value for cost_per_mixture and total_cost. Get the input for mixtures_required from the user.

{ Note: total_cost can be calculated as mixtures_required * cost_per_mixture }

Structures Below record the room details in the building.

```
struct class_room
{
   char name[20];
   int number;
   int no_of_benches;
   int capacity;
};

struct staff_room
{
   char name[20];
   int number;
};

struct other_rooms
{
   char name[20];
   int number;
};

struct rooms
{
   struct class_room cr[15];
   struct staff_room sr[5];
   struct other_rooms cr[5];
};
```

Statically initialize the above designed structures as specified in structure room. 15 classrooms, 5 staff rooms and 5 other rooms.

For the structure given below, collect the input from the user.

```
struct college_building
{
   char engineer_name[20];
   int contract_period;
   float budget;
};
```

Your program should output the following:

-All the data gathered in first 03 structures

-Name of the engineer, contract period and budget

-Print the other expenditures excluding total cost on mixtures

-Print the total number of benches and total capacity of the building

-Print the other rooms available in the building

-Print if there is an auditorium in the building

-Print the 2 class room names with highest capacity along with all the details

**Q-10-structure-union**

**A discussion between Union and Structure**

Union: "I almost got you killed in programmers market!"

Structure: "Dude! Most of the budding programmers clearly don't even know you!" check out what this ISE coder writes when I ask,

Differentiate between union and structure with suitable tiny code snippets.

## Bogie Number 15

## Pick The Right One!

### Multiple Choice Questions

**1. Consider the following program:**

```
#include<stdio.h>
struct table
{
   char meterial_used[20];
   char color[20];
   int no_of Legs;
};

int main()
{
   struct table t;
   return 0;
}
```

**What is the correct way to initialize the color of the table to be "brown"?**

**a.** t.table.color = "brown";

**b.** t.color = "brown"

**c.** strcpy( t.table.color , "brown");

**d.** strcpy( t.color, "brown");

**2. Consider the following anonymous structure (structure with no name). What would be the output of the code?**

```
#include <stdio.h>
struct
{
   int i;
   float u;
}v;

int main()
{
   v.i = 10;
   printf("%d\n", v.i);
}
```

**a.** Run time error

**b.** 10

**c.** Compiler time error

**d.** Undefined behavior

**3. We can type cast a structure variable to required data type.**

**a**. Yes

**b.** No

**4. What is the output of below program?**

```
#include <stdio.h>

struct data
```

```
{
   int i;
   float f;
};

int main()
{
  struct data d1, d2;
  d1.i = 10;
  d1.i = d2.i;
  printf("%d\n", d1.i);
  return 0;
}
```

**a.** 10

**b.** Garbage Value

**c.** Compile error

**d.** System Hangs

**5. What would be the output of the following program?**

```
#include <stdio.h>

struct data
{
   char *string;
};

int main()
```

```
{
   struct data d;
   d.string = "abc";
   printf("%s\n", d.string);
   return 0;
}
```

**a.** No need to run the code. There is error. No memory allocated for ‘string’.

**b.** Runtime error

**c.** abc

**d.** Compile time error

**6. What will be the output of the following program?**

```
#include <stdio.h>

struct dog
{
    char name[20];
    char color[20];
    int age;
};

int main()
{
   struct dog d = {"Tommy",6};
   printf("%s %s %d", d.name, d.color, d.age);
   return 0;
}
```

**a**. Tommy NULL 6

**b.** Tommy Garbage 0

**c.** Tommy 6 Garbage

**d.** Compile Error

**7. Which of the following statement is false?**

**a.** S1 = S2 copies all the member values of structure variable S2 to S1, provided they both are variables of same structure.

**b.** Structure members are accessed using dot/arrow operator for value/pointer variables respectively.

**c.** We can use strcmp() to compare the two structure string members.

**d.** All structures must have a name compulsorily.

**8. What is the output of the following program?**

```
#include <stdio.h>

struct data
{
    char *string;
    int *number;
};

int main()
{
   struct data d;
   printf("%d\n", sizeof(d));
   return 0;
```

}

**a.** 5

**b.** 6

**c.** 8

**d.** Error

**9. You miss to give a semicolon at the end of the structure definition.**

**a.** You get a compile time error

**b.** Program works perfectly

**c.** Program crashes at run time

**d.** Any of the above can happen depending on the compiler

**10. A structure can have a member whose data type is same as the defining structure.**

**a.** True

**b.** False

## The Station

## It Must Have Been a Happy Journey

### Concluding Thoughts

We have gained sufficient knowledge on structures conceptually and as well with respect to code and implementation. This will make the data structures and algorithms journey peaceful. If you have any anything to share, let me know. I would be happy to receive.

Very soon my book '**C the Data Structures'** will be out which covers the basic data structures in C. If you have liked this ebook, you might get comfortable to read that as well. Do look out!

Regards,

**Prakash B. Hegade**

**prakash.hegade@gmail.com**