Review of Arrays

```
int mark[5] = {19, 10, 8, 17, 9};

int mark[5] = {19, 10, 8, 17, 9}
mark[2] = -1; // make the value of the third element to -1
mark[4] = 0; // make the value of the fifth element to 0


scanf("%d", &mark[2]); // take input and store it in the 3rd element

scanf("%d", &mark[i]); // take input and store it in the ith element
```

```c
// Program to take 5 values from the user and store them in an array
// Print the elements stored in the array
#include <stdio.h>

int main() {
  int values[5];

  printf("Enter 5 integers: ");

  // taking input and storing it in an array
  for(int i = 0; i < 5; ++i) {
     scanf("%d", &values[i]);
  }

  printf("Displaying integers: ");

  // printing elements of an array
  for(int i = 0; i < 5; ++i) {
     printf("%d\n", values[i]);
  }
  return 0;
}
```

```
//Program to find the average of n numbers
using arrays
int main()
{
 int marks[10], i, n, sum = 0, average;

 printf("Enter number of elements: ");
 scanf("%d", &n);

  for(i=0; i<n; ++i)      {
    printf("Enter number %d: ",i+1);
     scanf("%d", &marks[i]);
// adding integers entered by the
user to the sum variable
          sum += marks[i];      }
     average = sum/n;
    printf("Average = %d", average);
     return 0;   }
```

```
// C program to store temperature of two cities of a week and display it.
#include <stdio.h>
const int CITY = 2;
const int WEEK = 7;
int main()
{
  int temperature[CITY][WEEK];

  // Using nested loop to store values in a 2d array
  for (int i = 0; i < CITY; ++i)
  {
    for (int j = 0; j < WEEK; ++j)
    {
      printf("City %d, Day %d: ", i + 1, j + 1);
      scanf("%d", &temperature[i][j]);
    }
  }

  printf("\nDisplaying values:
\n\n");

  // Using nested loop to display
vlues of a 2d array
  for (int i = 0; i < CITY; ++i)
  {
    for (int j = 0; j < WEEK; ++j)
    {
      printf("City %d, Day %d =
%d\n", i + 1, j + 1,
temperature[i][j]);
    }
  }
  return 0;
}
```

Output

City 1, Day 1: 33
City 1, Day 2: 34
City 1, Day 3: 35
City 1, Day 4: 33
City 1, Day 5: 32
City 1, Day 6: 31
City 1, Day 7: 30
City 2, Day 1: 23
City 2, Day 2: 22
City 2, Day 3: 21
City 2, Day 4: 24
City 2, Day 5: 22
City 2, Day 6: 25
City 2, Day 7: 26

```
// C Program to store and print 12 values entered by the user
#include <stdio.h>
int main()
{
  int test[2][3][2];
  printf("Enter 12 values: \n");
  for (int i = 0; i < 2; ++i)
  {
    for (int j = 0; j < 3; ++j)
    {
      for (int k = 0; k < 2; ++k)
      {
        scanf("%d", &test[i][j][k]);
      }    }   }
```

```
// Printing values with proper index.

  printf("\nDisplaying values:\n");
  for (int i = 0; i < 2; ++i)
  {
    for (int j = 0; j < 3; ++j)
    {
      for (int k = 0; k < 2; ++k)
      {
        printf("test[%d][%d][%d] = %d\n",
            i, j, k, test[i][j][k]);
      }
    }
  } return 0; }
```

Structure

A struct (or structure) is a collection of variables (can be of different types) under a single name.

How to define structures?

Before you can create structure variables, you need to define its data type.
To define a struct, the struct keyword is used.

```
struct structureName
{
    dataType member1;
    dataType member2;
    ...
};
```

```
struct Person
{
    char name[50];
    int citNo;
    float salary;
};
```

```
struct Point
{
   int x = 0;
   int y = 0;
};
```

Create struct variables

When a struct type is declared, no storage or memory is allocated. To allocate memory of a given structure type and work with it, we need to create variables.

```
struct Person
{
    char name[50];
    int citNo;
    float salary;
};
int main()
{
    struct Person person1, person2, p[20];
    return 0;
}
```

```
struct Person
{
    char name[50];
    int citNo;
    float salary;
} person1, person2, p[20];
```

```
struct Point
{
   char  x[10];
int  y;
};
```

```
int main()
{
      struct Point p1 = {0, 1};
       struct point p1;
       ”;
       P1.y=20;
}
```

Access members of a structure
There are two types of operators used for accessing members of a structure.
. - Member operator
-> - Structure pointer operator

```
struct Point
{
   int x, y;
};

int main()
{
   struct Point p1 = {1, 2};

   // p2 is a pointer to structure p1
   struct Point *p2 = &p1;

// Accessing structure members using structure pointer
   printf("%d %d", p2->x, p2->y);
   return 0;
}
```

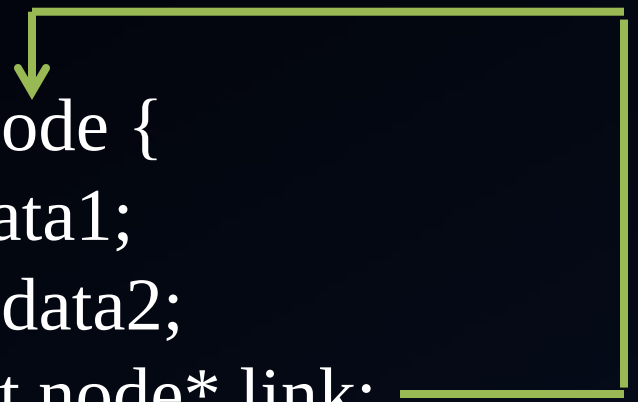
```c
#include <stdio.h>
struct student {
    char name[50];
    int roll;
    float marks;
} s;
int main() {
    printf("Enter information:\n");
    printf("Enter name: ");
    fgets(s.name, sizeof(s.name), stdin);
    printf("Enter roll number: ");
    scanf("%d", &s.roll);
    printf("Enter marks: ");
    scanf("%f", &s.marks);
    printf("Displaying Information:\n");
    printf("Name: ");
    printf("%s", s.name);
    printf("Roll number: %d\n", s.roll);
    printf("Marks: %.1f\n", s.marks);

    return 0;
}
```

Demonstrate the Dynamic Memory Allocation for Structure

```
#include <stdio.h>
#include <stdlib.h>
struct course {
    int marks;
    char subject[30];
};

int main() {
    struct course *ptr;
    int i, noOfRecords;
    printf("Enter the number of
records: ");
    scanf("%d", &noOfRecords);
 // Memory allocation for noOfRecords structures
    ptr = (struct course *)malloc(noOfRecords *
sizeof(struct course));
    for (i = 0; i < noOfRecords; ++i) {
        printf("Enter the name of the subject and marks
respectively:\n");
        scanf("%s %d", (ptr + i)->subject, &(ptr + i)->marks
    }

    printf("Displaying Information:\n");
    for (i = 0; i < noOfRecords; ++i)
     printf("%s\t%d\n", (ptr + i)->subject, (ptr + i)->marks)

    return 0;
}
```

Self Referential structures

Self Referential structures are those structures that have one or more pointers which point to the same type of structure, as their member.



```
struct node {
    int data1;
    char data2;
    struct node* link;
};
```

```
int main()
{
    struct node ob;
    return 0;
}
```

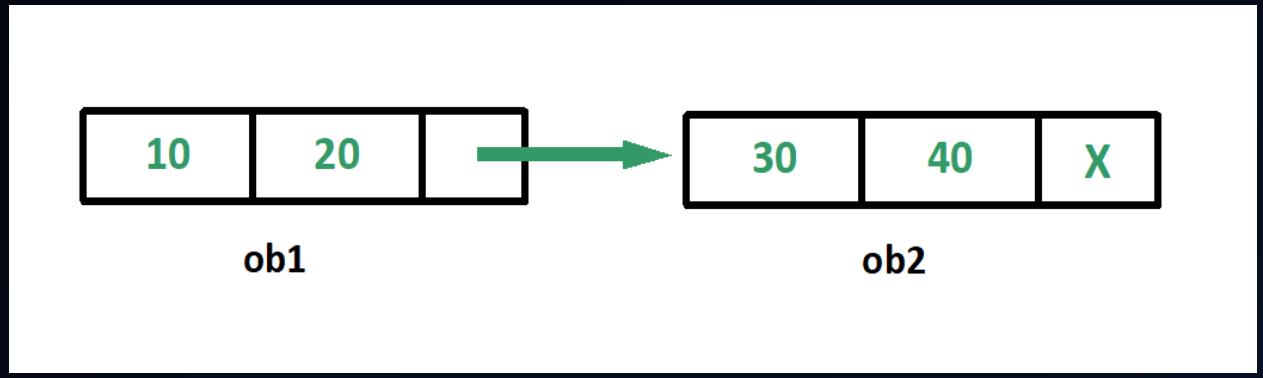
```
struct SELF_REF {
        int      a;
        struct  SELF_REF  b;
        int      c;
} ;
```

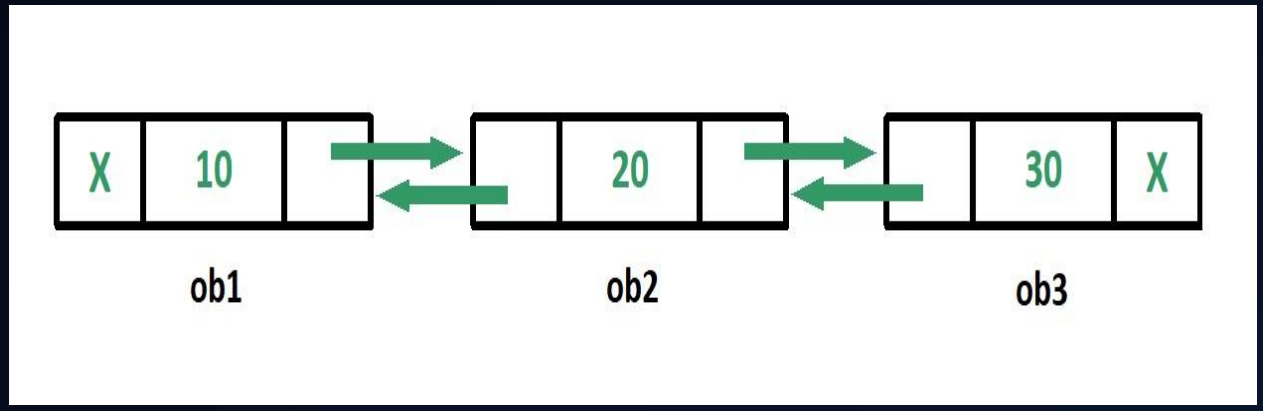
```
typedef struct {
        int      a;
        struct  SELF_REF  *b;
        int      c;
}  SELF_REF ;
```

```
#include <stdio.h>

struct node {
    int data1;
    char data2;
    struct node* link;
};
  int main()
{
    struct node ob1; // Node1
      // Initialization
    ob1.link = NULL;
    ob1.data1 = 10;
    ob1.data2 = 20;
 struct node ob2; // Node2

    // Initialization
    ob2.link = NULL;
    ob2.data1 = 30;
    ob2.data2 = 40;

// Linking ob1 and ob2
    ob1.link = &ob2;
// Accessing data members of  ob2 using
ob1
    printf("%d", ob1.link->data1);
    printf("\n%d", ob1.link->data2);
    return 0;
 }
```

Self referential structure with multiple link

```
struct node {
    int data;
    struct node* prev_link;
    struct node* next_link;
};
```





Applications:

Self referential structures are very useful in creation of other complex data structures like:

- Linked Lists
- Stacks
- Queues
- Trees
- Graphs