01

Recursive function, pointer, structure and self-referential structure

DATA STRUCTURE LAB

SESSION - 01

CSE135 | DAFFODIL INTERNATIONAL UNIVERSITY

Recursion:

Some computer programming languages allow a module or function to call itself. This technique is known as recursion. **Example**:

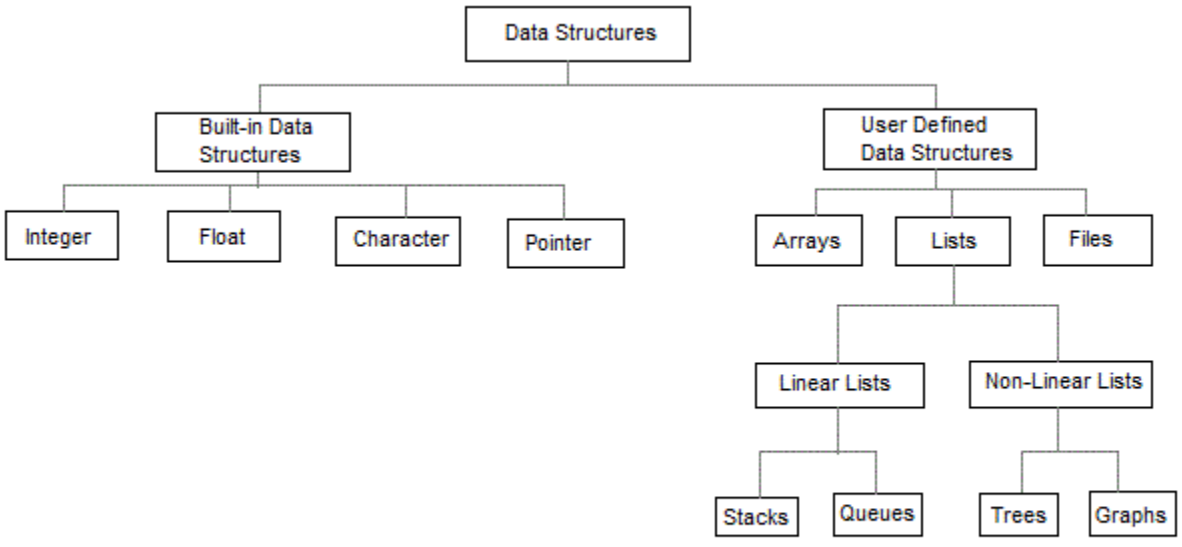
```
int function(int value) {
   if(value < 1)
      return;
   function(value - 1);
   printf("%d ",value);
}
```

Exercises:

1. Write a recursive function that computes the sum of all numbers from 1 to n, where n is given as parameter.

2. Write a recursive function that finds and returns the minimum element in an array, where the array and its size are given as parameters.

Basic Structure:



INTRODUCTION TO DATA STRUCTURES

Pointer:

When setting up data structures like lists, queues and trees, it is necessary to have pointers to help manage how the structure is implemented and controlled. The basic syntax to define a pointer is:

```
int *ptr;
```

In any case, once a pointer has been declared, the next logical step is for it to point at something:

```
int *ptr = NULL;
ptr = &a;
int a = 5;
```

Self-Referential Structure Using Pointer:

If a structure contains one or more pointers to itself (a structure of same type) as its members, then the structure is said to be a self-referential structure, that is, a structure that contains a reference to its own structure type.

In brief, One of the member of the structure is pointer of the same type.

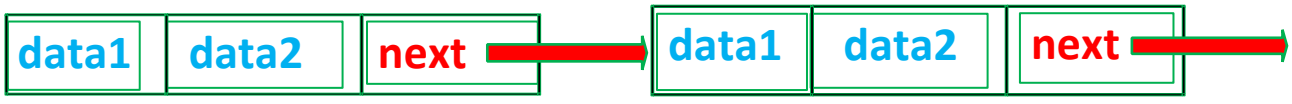
For example:

```
struct node        //{here marked portion is “data type}
{
 int data1;
 int data2;
 struct node *next; //{here red marked portion is “self referential member”}
};
```

Here, the structure '*node*' contains a pointer named '*next*', which is of the same type (struct node) as the structure it contains in (struct node).

The above illustrated structure prototype describes one node that comprises of two logical segments. One of them stores data/information and the other one is a pointer indicating where the next component can be found. .Several such inter-connected nodes create a chain of structures.

The following figure depicts the composition of such a node.



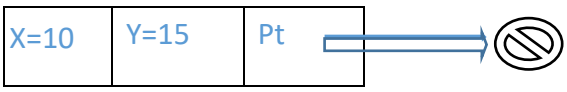
TYEPEDEF:

typedef is used to give a new name to a type. Lets define a new type for the given *struct node,*

***typedef** struct node point;* //*green* marked is old data type and *blue* marked is new data type

Now structure is declared as "point" instead of node.

typedef really does simply declare a new *name* for a type. It does not create a new type. #simple program on creating nodes and assigning data:




```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
    int x,y;
    struct Node *next;
};
typedef struct Node  node;
int main()
{
    node *head;
    head=(node*)malloc(sizeof(node));
    head->x=10;
    head->y=15;
    head->pt=NULL;
    printf("%d %d\n",head->x,head->y);
    return 0;
}
```

Structures in C:

What is Structure?

- A structure is a user defined data type in C/C++. A structure creates a data type that can be used to group items of possibly different types into a single type

How to create Structure?

- 'struct' keyword is used to create a structure. Following is an example.

```
struct address
{
   char name[50];
   char street[100];
   char city[50];
   char state[20]
   int pin;
};
```

Here is an **example** program:

```
struct database {
  int id_number;
  int age;
  float salary;
};

int main()
{
  struct database employee;  /* There is now an employee variable that has
                                modifiable variables inside it.*/
  employee.age = 22;
  employee.id_number = 1;
  employee.salary = 12000.21;
}
```

Accessing Structure Members

To access any member of a structure, we use the **member access operator (.)**. The member access operator is coded as a period between the structure variable name and the structure member that we wish to access. You would use the keyword **struct** to define variables of structure type. The following example shows how to use a structure in a program –

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

```c
struct Books {
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
};

int main( ) {

   struct Books Book1;        /* Declare Book1 of type Book */
   struct Books Book2;        /* Declare Book2 of type Book */

   /* book 1 specification */
   strcpy( Book1.title, "C Programming");
   strcpy( Book1.author, "Nuha Ali");
   strcpy( Book1.subject, "C Programming Tutorial");
   Book1.book_id = 6495407;

   /* book 2 specification */
   strcpy( Book2.title, "Telecom Billing");
   strcpy( Book2.author, "Zara Ali");
   strcpy( Book2.subject, "Telecom Billing Tutorial");
   Book2.book_id = 6495700;

   /* print Book1 info */
   printf( "Book 1 title : %s\n", Book1.title);
   printf( "Book 1 author : %s\n", Book1.author);
   printf( "Book 1 subject : %s\n", Book1.subject);
   printf( "Book 1 book_id : %d\n", Book1.book_id);

   /* print Book2 info */
   printf( "Book 2 title : %s\n", Book2.title);
```

```
   printf( "Book 2 author : %s\n", Book2.author);

   printf( "Book 2 subject : %s\n", Book2.subject);

   printf( "Book 2 book_id : %d\n", Book2.book_id);


   return 0;
}
```

When the above code is compiled and executed, it produces the following result –

```
Book 1 title : C Programming
Book 1 author : Nuha Ali
Book 1 subject : C Programming Tutorial
Book 1 book_id : 6495407
Book 2 title : Telecom Billing
Book 2 author : Zara Ali
Book 2 subject : Telecom Billing Tutorial
Book 2 book_id : 6495700
```

Another quick **example**:

```
#include <stdio.h>

struct xampl {
  int x;
};

int main()
{
    struct xampl structure;
    struct xampl *ptr;

    structure.x = 12;
    ptr = &structure; /* Yes, you need the & when dealing with
                         structures and using pointers to them*/
    printf( "%d\n", ptr->x );  /* The -> acts somewhat like the * when
                                  does when it is used with pointers
                                   It says, get whatever is at that memory
                                  address Not "get what that memory address
                                  is"*/
    getchar();
}
```