**DATA STRUCTURES**

**SINGLE LINKED LIST (FORWARD LINKED LIST)**

**SRP** – Self Referential Pointer ---- When a member of a structure is a pointer which is pointing to itself

Row houses --- villas or homes of same types

Struct home

{

Int value;

Struct home \*ptr;

} HOME; // Pointer is pointing to itself

HOME h1; 🡺 Then a box of two parts will be formed which would be allotted for value and pointer

Ex: 1000

10 \*

djhbhd

h1.value = 10;

h1.ptr = NULL; // because it behaves as wild pointer

1. Creating a node
2. Initialization of the node

Once the pointer is grounded(made null) then it can be initialized like

Struct Home \*ptr = NULL;

Ptr = &h1;

Pf(“%d”, h1.value) 🡺 10

Pf(“%d”,ptr -> value); 🡺 10

**Note:** Pointer can hold only one address (one box – it may contain any number of partitions

1. Creating a relationship ---- to navigate to next location

To point to next location first pointer must hold the address of the next home node(box)

h1.ptr = &h2;

h2.ptr = &h3;

This also leads to the creation of list(linked list).

1. Navigation ---

Ptr 1 = h2.ptr;

Ptr 2 = h3.ptr;

1 #include <stdio.h>

2 #include <stdlib.h>

3

4 typedef struct node

5 {

6 int val;

7 struct node \*ptr; //Self Referential Pointer(SELF)

8 }NODE;

9

10 int main()

11 {

12 //Creating nodes and allocated memory -- memory allocation is done by the compiler as it is static

13 NODE h1,h2,h3;

14

15 //ptr is used to navigate/ traverse between nodes

16 NODE \*ptr1 = NULL;

17

18 //Initialize values to each node

19

20 h1.val = 10;

21 h1.ptr = NULL;

22

23 h2.val = 20;

24 h2.ptr = NULL;

25

26 h3.val = 30;

27 h3.ptr = NULL;

28

29 printf("\nAddress of h1=%p",&h1);

30 printf("\nAddress of h2=%p",&h2);

31 printf("\nAddress of h3=%p",&h3);

32 printf("\nAddress of ptr1=%p",&ptr1);

33

34 printf("\nh1.value=%d and ptr=%p",h1.val,h1.ptr);

35 printf("\nh2.value=%d and ptr=%p",h2.val,h2.ptr);

36 printf("\nh3.value=%d and ptr=%p",h3.val,h3.ptr);

37

38 printf("\nContents of ptr1 / ptr1 is pointing to: %p",ptr1);

39

40 //Making relationship(traverse)/creating LList

41 h1.ptr = &h2;

42 h2.ptr = &h3;

43

44 //Traverse

45

46 ptr1 = &h1;

47

48 printf("\n%d",h1.val);

49 printf("\n%d",ptr1 -> val);

50

51 //ptr1 = &h2;

52 ptr1 = h1.ptr; //ptr1 is pointing to h2

53 printf("\n%d",ptr1 -> val);

54

55 //ptr1 = &h3;

56 ptr1 = h2.ptr ; //from earlier line by 57

57 printf("\n%d",ptr1 -> val);

58

59

60 printf("\n\n");

61

62 }

O/P

user50@trainux01:~/practice/try/day16.1$ gcc linkedList.c

user50@trainux01:~/practice/try/day16.1$ ./a.out

Address of h1=0x7ffed321ee20

Address of h2=0x7ffed321ee30

Address of h3=0x7ffed321ee40

Address of ptr1=0x7ffed321ee18

h1.value=10 and ptr=(nil)

h2.value=20 and ptr=(nil)

h3.value=30 and ptr=(nil)

Contents of ptr1 / ptr1 is pointing to: (nil)

10

10

20

30

Inserting a new node between the existing --- n4 is the new node formed to insert between n1 and n2

So n1.ptr = &n4;

And n4.ptr = &n2;

Refer repository for different versions

If we need to insert a new node n6 at the beginning of the list then,

n6.ptr = &n1;

ptr1 = &n6;

So here the base address itself is changed to the address of the new node n6.

**NOTE**: First whenever a new node is created it should be initialized with a value and then the pointer of that node must be subjected to NULL and then can be used in the relation

IF we need to add a node to the list we need to check whether the list is present or not if not then the node inserted will be the first node of the list and the address will be the base address.

Append Node is done…refer it

Only a forward move in linked list --- pointer and the temporary variable at the starting will be pointing to base address and after the movement temporary address will be one step behind the head pointer.

Delete a node with the key value(value of the node)

While deleting the node if it is beginning of the node or end of the list, both must be considered and write the code and can it be freed or not.

Free the node deleted and space memory

* The pointer used here is the forward list pointer

**DOBLE LINKED LIST**

**CIRCULAR LINKED LIST**