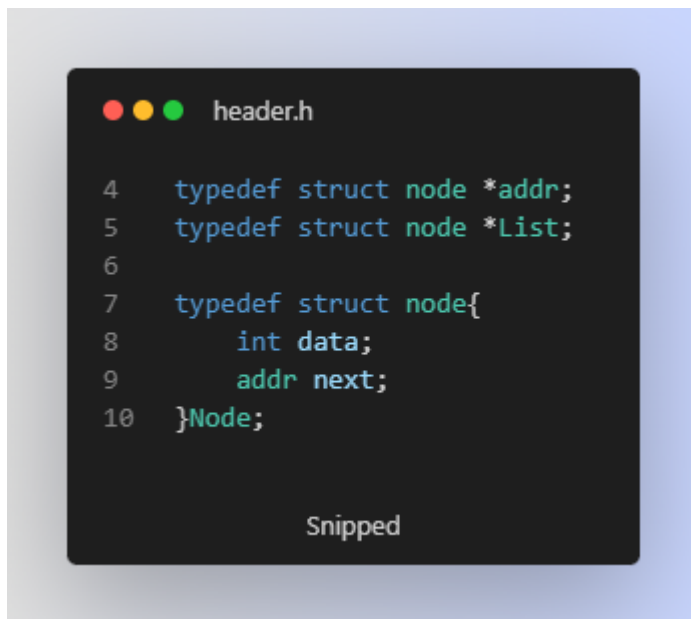


## MODUL 10 - RECURSIVE LIST

### I. Dasar Teori

Recursive List pada dasarnya merupakan gabungan dari modul linked list dan recursive function. Pada modul ini kita masih sama membahas mengenai linked list, akan tetapi perbedaannya terletak pada penggunaan recursive untuk setiap perulangan pada operasi-operasi linked list.



Gambar diatas merupakan stuktur untuk list yang akan kita gunakan dalam modul recursive list. Pada modul sebelumnya kita diharuskan membuat struct yang berisi node pertama dari list. Untuk recursive list kita tidak memerlukan lagi struct yang mengakses node pertama melainkan langsung menggunakan list sebagai node pertama karena sekarang list merupakan pointer.

## II. Operasi pada Recursive List

Code pada operasi-operasi recursive list umumnya lebih pendek daripada menggunakan perulangan biasa. Tetapi, walaupun code yang dibutuhkan lebih sedikit pemakaian recursive list lebih melibatkan memori yang lebih besar. Karena untuk setiap iterasi, seluruh algoritma pada fungsi akan dieksekusi lagi.

- Insert First



Langkah – langkah :

1. Arahkan next pointer node baru ke node list
2. Ubah node list dengan node baru

- Insert After



Langkah -langkah :

1. Cari node sebelumnya dari node data yang akan diinputkan setelahnya dan simpan ke dalam sebuah variabel seperti contoh code memakai variabel prevNode
2. Pastikan node tersebut ada di dalam list
3. Ubah next node dari node baru dengan next node prevNode
4. Ubah next node prevNode node baru

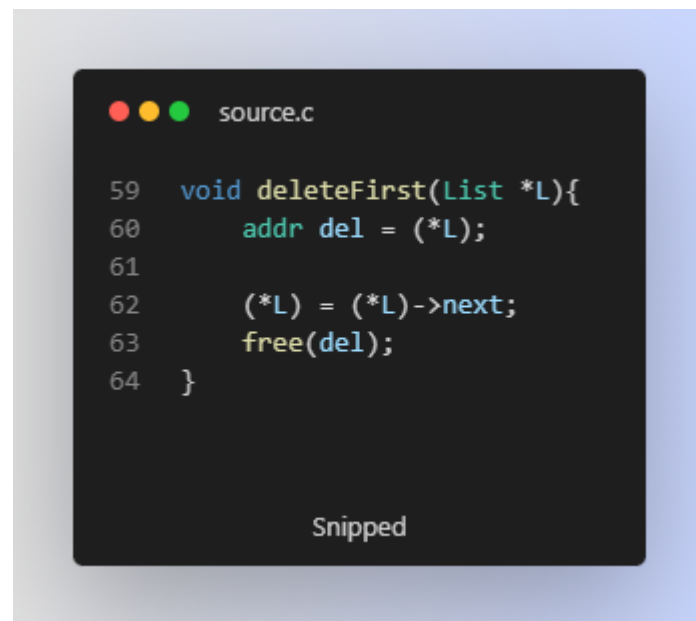
- Insert Last



Langkah -langkah :

1. Periksa apakah list masih dalam keadaan kosong/NULL/merupakan node terakhir dari list. Jika list masih kosong lakukan insert first
2. Lakukan recursive dengan memanggil kembali fungsi yang parameter pertamanya adalah next node dari list untuk menuju ke node terakhir dari list.

- Delete First



Langkah -langkah :

1. Simpan list ke dalam variabel temp
2. Ubah list menjadi next nodenya
3. Hapus node temp

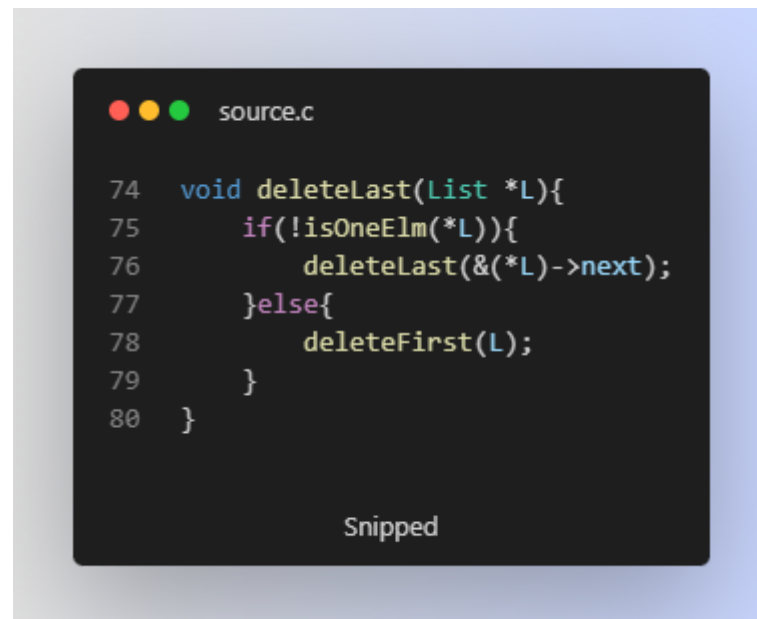
- Delete At



Langkah – langkah :

1. Lakukan recursive apabila node pertama list tidak sesuai dengan node yang akan dihapus. Recursive dilakukan dengan memanggil kembali fungsi yang parameter pertamanya adalah next node dari list
2. Recursive berhenti saat node list sesuai dengan node yang akan dihapus
3. Lakukan delete first

- Delete Last

A screenshot of a code editor window titled 'source.c' with a dark background. It shows a C function 'deleteLast' that takes a 'List \*L' as an argument. The function uses a recursive approach: if the list has more than one element, it calls 'deleteLast' on the next node; otherwise, it calls 'deleteFirst' on the current list. The code is as follows:

```
74 void deleteLast(List *L){
75     if(!isOneElm(*L)){
76         deleteLast(&(*L)->next);
77     }else{
78         deleteFirst(L);
79     }
80 }
```

The word 'Snipped' is visible at the bottom of the code block.

Langkah – langkah :

1. Lakukan recursive apabila list tersebut memiliki node lebih dari 1. Recursive dilakukan dengan memanggil kembali fungsi yang parameter listnya adalah next node dari list
2. Recursive akan berhenti apabila next node list adalah NULL
3. Lakukan delete first

- Find Node

A screenshot of a code editor window titled 'source.c' with a dark background. It shows a C function 'findNode' that takes a 'List L' and an 'int data' as arguments. The function checks if the list is empty; if so, it returns NULL. If not, it checks if the current node's data matches the target data. If it does, it returns the current node; otherwise, it recursively calls 'findNode' on the next node. The code is as follows:

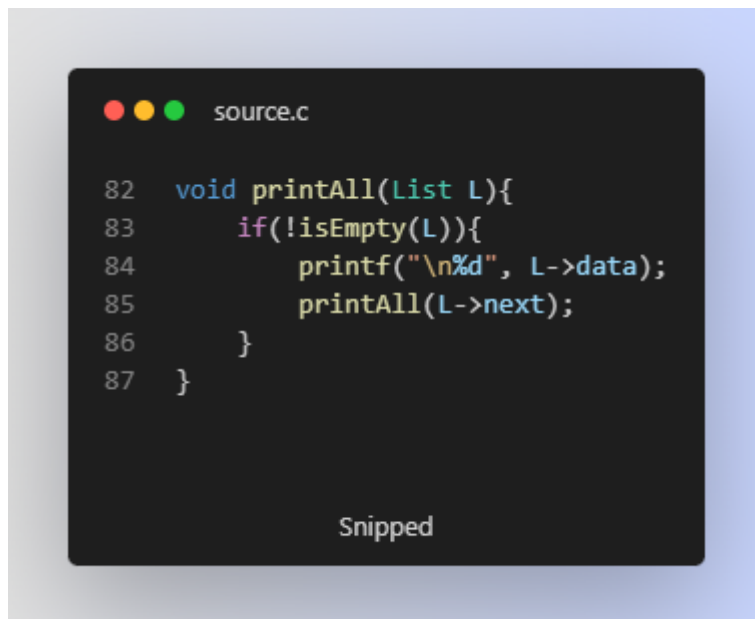
```
15 addr findNode(List L, int data){
16     if(isEmpty(L)){
17         return NULL;
18     }else{
19         if(L->data==data){
20             return findNode(L->next, data);
21         }
22         return L;
23     }
24 }
```

The word 'Snipped' is visible at the bottom of the code block.

Langkah – langkah :

1. Lakukan recursive dengan memanggil kembali fungsi yang parameter pertamanya adalah next node dari list
2. Recursive berhenti jika data node list sudah sesuai dengan data yang akan dicari dan meretrunkan node tersebut

- Print Data List



Langkah – langkah :

1. Tampilkan data dari node list jika keadaan awal list bukan kosong
2. Lakukan recursive dengan memanggil kembali fungsi yang parameter listnya adalah next node dari list
3. Recursive akan berhenti apabila node list telah NULL

## GUIDED 10 – RECURSIVE LIST

Silahkan kerjakan guided di bawah ini sebelum mengikuti praktikum. Format pengumpulan guided adalah **GD10\_Y\_XXXXX.zip** dengan **XXXXX** adalah **5-digit terakhir NPM** dan **Y** adalah kelas.

### header.h

```
1. #include <stdio.h>
2. #include <stdlib.h>
3.
4. typedef struct node *addr;
5. typedef struct node *List;
6.
7. typedef struct node{
8.     int data;
9.     addr next;
10. }Node;
11.
12. void createEmpty(List *L);
13. int isOneElm(List L);
14. int isEmpty(List L);
15. addr findNode(List L, int data);
16. addr alokasi(int data);
17. void insertFirst(List *L, addr newNode);
18. void insertAfter(List *L, addr newNode, int prevData);
19. void insertLast(List *L, addr newNode);
20. void deleteFirst(List *L);
21. void deleteAt(List *L, addr delNode);
22. void deleteLast(List *L);
23. void printAll(List L);
```

### source.c

```
1. #include "header.h"
2.
3. void createEmpty(List *L){
4.     (*L) = NULL;
5. }
6.
7. int isOneElm(List L){
8.     return L->next == NULL;
9. }
10.
11. int isEmpty(List L){
12.     return L == NULL;
13. }
14.
15. addr findNode(List L, int data){
16.     if(isEmpty(L)){
17.         return NULL;
18.     }else{
19.         if(L->data!=data){
20.             return findNode(L->next, data);
21.         }
22.         return L;
23.     }
24. }
25.
26. addr alokasi(int data){
27.     addr N = (Node*) malloc(sizeof(Node));
```

```

28.
29.     N->next = NULL;
30.     N->data = data;
31.
32.     return N;
33. }
34.
35. void insertFirst(List *L, addr newNode){
36.     newNode->next = (*L);
37.     (*L) = newNode;
38. }
39.
40. void insertAfter(List *L, addr newNode, int prevData){
41.     addr prevNode = findNode(*L, prevData);
42.
43.     if(prevNode!=NULL){
44.         newNode->next = prevNode->next;
45.         prevNode->next = newNode;
46.     }else{
47.         printf("\ndata tidak ada");
48.     }
49. }
50.
51. void insertLast(List *L, addr newNode){
52.     if(isEmpty(*L)){
53.         insertFirst(L, newNode);
54.     }else{
55.         insertLast(&(*L)->next, newNode);
56.     }
57. }
58.
59. void deleteFirst(List *L){
60.     addr del = (*L);
61.
62.     (*L) = (*L)->next;
63.     free(del);
64. }
65.
66. void deleteAt(List *L, addr delNode){
67.     if((*L)!=delNode){
68.         deleteAt(&(*L)->next, delNode);
69.     }else{
70.         deleteFirst(L);
71.     }
72. }
73.
74. void deleteLast(List *L){
75.     if(!isOneElm(*L)){
76.         deleteLast(&(*L)->next);
77.     }else{
78.         deleteFirst(L);
79.     }
80. }
81.
82. void printAll(List L){
83.     if(!isEmpty(L)){
84.         printf("\n%d", L->data);
85.         printAll(L->next);
86.     }
87. }

```



## main.c

```
1. #include "header.h"
2.
3. int main(int argc, char *argv[]) {
4.     int menu=-1, data, prevData;
5.
6.     List L;
7.     addr delNode;
8.
9.     createEmpty(&L);
10.
11.     while(menu!=0){
12.         system("cls");
13.
14.         printf("1. Insert first");
15.         printf("\n2. Insert after");
16.         printf("\n3. Insert last");
17.         printf("\n4. Delete first");
18.         printf("\n5. Delete at");
19.         printf("\n6. Delete last");
20.         printf("\n7. Find data");
21.         printf("\n8. Print all");
22.         printf("\n0. Keluar");
23.
24.         printf("\n>> "); scanf("%d", &menu);
25.
26.         switch(menu){
27.             case 1:
28.                 printf("\ndata: "); scanf("%d", &data);
29.
30.                 insertFirst(&L, alokasi(data));
31.                 break;
32.
33.             case 2:
34.                 printf("\ndata: "); scanf("%d", &data);
35.
36.                 printf("masukkan setelah data: ");
37.                 scanf("%d", &prevData);
38.
39.                 insertAfter(&L, alokasi(data), prevData);
40.                 break;
41.
42.             case 3:
43.                 printf("\ndata: "); scanf("%d", &data);
44.
45.                 insertLast(&L, alokasi(data));
46.                 break;
47.
48.             case 4:
49.                 deleteFirst(&L);
50.                 break;
51.
52.             case 5:
53.                 printf("\ndata yang ingin dihapus: "); scanf("%d", &data);
54.
55.                 delNode = findNode(L, data);
56.
57.                 if(delNode != NULL)
58.                     deleteAt(&L, delNode);
59.                 else
60.                     printf("data tidak ada");
61.                 break;
62.
63.             case 6:
64.                 deleteLast(&L);
65.                 break;
66.         }
```

```
67.         case 7:
68.             printf("\ndata yang ingin dicari: "); scanf("%d", &data);
69.
70.             if(findNode(L, data) == NULL)
71.                 printf("\ndata tidak ada");
72.             else
73.                 printf("\ndata ada");
74.         break;
75.
76.         case 8:
77.             printAll(L);
78.         break;
79.
80.         case 0:
81.             break;
82.
83.         default:
84.             printf("\ninvalid");
85.     }
86.
87.     getch();
88. }
89.
90. return 0;
91. }
92.
```