

MODUL 10

RECURSIVE LIST



A. Tujuan

1. Pratiikan dapat memahami tentang konsep dari *recursive list* didalam Bahasa Pemrograman C.
2. Pratiikan dapat mengerti cara pemakaian dan pembuatan *syntax-syntax* yang diperlukan oleh *recursive list*.

B. Penjelasan

Recursive list merupakan penggabungan dari modul *linked list* dan *recursive function*. Pada modul ini masih membahas mengenai *linked list*, namun terdapat perbedaan pada penggunaan *recursive* pada setiap perulangan yang dilakukan pada operasi – operasi *linked list*.

1. Struktur

Pada modul sebelumnya, untuk membuat *list*, diperlukan sebuah *struct* untuk menyimpan *pointer first* yang menunjuk ke *node* pertama dari *linked list*. Pada *recursive list* tidak diperlukan *struct* tersebut, melainkan langsung menggunakan sebuah *pointer* dengan nama *List*, yang merupakan *pointer* yang menunjuk ke *Node*.

```
typedef struct node *address;
typedef struct node *List;

typedef struct node{
    int x;
    address next;
}Node;
```

Gambar 1. Struktur *Recursive List*

2. Operasi

➤ Insert First

A screenshot of a code editor window showing the implementation of the insertFirst function. The code is as follows:

```
void insertFirst(List *l, address newNode){  
    newNode->next = (*l);  
    (*l) = newNode;  
}
```

Gambar 2. Insert First pada Recursive List

Langkah:

1. Arahkan *next pointer* milik *node* baru ke *node* pertama pada *linked list* yang ditunjuk oleh *list*: `newNode->next = (*l)`.
2. Arahkan pointer yang menunjuk ke *node* pertama pada *list* ke *node* baru tersebut, sehingga kini `newNode` menjadi *node* pertama pada *linked list*: `(*l) = newNode`.

➤ Insert After

A screenshot of a code editor window showing the implementation of the insertAfter function. The code is as follows:

```
void insertAfter(List *l, address newNode, int prev){  
    address prevNode = findNode(*l, prev);  
  
    if(prevNode!=NULL){  
        newNode->next = prevNode->next;  
        prevNode->next = newNode;  
    }else{  
        printf("\nData Tidak Ada");  
    }  
}
```

Gambar 3. Insert After pada Recursive List

Langkah:

1. Cari alamat dari *node* sebelum data yang akan disimpan, dan simpan hasil pencarian ke dalam sebuah pointer(pada contoh menggunakan `prevNode`).
2. Pastikan *node* tersebut berada dalam *list* (pada contoh `prevNode!=NULL`).
3. Arahkan *next node* dari *node* baru ke *next node* milik `prevNode`.
4. Arahkan *next node* milik `prevNode` ke *node* baru.

➤ Insert Last



```
void insertLast(List *l, address newNode){  
    if(isEmpty(*l)){  
        insertFirst(l, newNode);  
    }else{  
        insertLast(&(*l)->next, newNode);  
    }  
}
```

Gambar 4. Insert Last pada Recursive List

Langkah:

1. Periksa apakah *list* berada dalam kondisi kosong. Jika dalam *list* masih kosong, maka lakukan *insert first*. Ini juga merupakan basis/kondisi pemberhenti yang harus dilakukan untuk menghentikan rekursif.
2. Apabila *list* tidak dalam kondisi kosong, maka lakukan *recursive* dengan memanggil kembali fungsi *insert last* dengan parameter *l* (List) menjadi *next node* dari *list* (*l->next*). Hal ini dilakukan untuk menggeser *next* sampai dengan setelah *node* terakhir dari *list*

➤ Delete First



```
void deleteFirst(List *l){  
    address del = (*l);  
  
    (*l) = (*l)->next;  
    free(del);  
}
```

Gambar 5. Delete First pada Recursive List

Langkah:

1. Buatlah sebuah *pointer* bernama *del*, yang akan menunjuk ke *node* pertama yang akan dihapus.
2. Arahkan *pointer* yang menunjuk ke *node* pertama pada *list*, ke *next node* (*node* berikutnya).
3. Hapus/dealokasi *node* yang ditunjuk *pointer del*.

➤ Delete At

A screenshot of a code editor window showing the implementation of the `deleteAt` function for a recursive list. The code is as follows:

```
void deleteAt(List *l, address delNode)
{
    if((*l)==delNode){
        deleteFirst(l);
    }else{
        deleteAt(&(*l)->next, delNode);
    }
}
```

Gambar 6. Delete At pada Recursive List

Langkah:

1. Periksa apakah *node* pertama pada *list* merupakan *node* yang sesuai dengan `delNode` (apakah sama dengan *node* yang akan kita hapus).
2. Apabila sudah sama, artinya sudah menemukan *node* yang akan dihapus, maka *recursive* berhenti dan lakukan *delete first*.
3. Apabila tidak sama, maka lakukan *recursive* dengan memanggil kembali fungsi *delete at* dengan parameter `l` (List) menjadi *next* dari *node list* (`l->next`). Hal ini dilakukan untuk menggeser *next* sampai dengan ketemu *node* yang dihapus.

➤ Delete Last

A screenshot of a code editor window showing the implementation of the `deleteLast` function for a recursive list. The code is as follows:

```
void deleteLast(List *l){
    if(isOneElmt(*l)){
        deleteFirst(&(*l));
    }else{
        deleteLast(&(*l)->next);
    }
}
```

Gambar 7. Delete Last pada Recursive List

Langkah:

1. Periksa apakah *list* tersebut sudah menunjuk pada *node* terakhir (`!isOneElmt(*l)`).
2. Apabila sudah berada pada *node* paling belakang, maka lakukan *delete first*. Ini juga merupakan basis/kondisi pemberhenti yang harus dilakukan untuk menghentikan rekursif.

3. Apabila belum berada pada *node* terakhir, maka lakukan *recursive* dengan memanggil fungsi *delete last* yang parameter *l* (List) merupakan *next node* dari *list* (*l->next*). Hal ini dilakukan untuk menggeser *next* sampai dengan *node* terakhir.

➤ Find Node



```
address findNode(List l, int x){
    if(isEmpty(l)){
        return NULL;
    }else{
        if(l->x == x){
            return l;
        }
        return findNode(l->next, x);
    }
}
```

Gambar 8. Find Node pada Recursive List

Langkah:

1. Apabila *list* kosong, returnkan NULL, ini merupakan basis/kondisi pemberhenti yang harus dilakukan untuk menghentikan *recursive*. Artinya bahwa data yang dicari tidak ditemukan.
2. Apabila *list* tidak kosong, maka bandingkan data pada *node* pertama, apakah sama dengan data yang dicari. Jika sama, artinya ketemu, maka returnkan alamat dari *node* tersebut, dan *recursive* berhenti.
3. Jika tidak sama, maka lakukan *recursive* dengan memanggil kembali fungsi dengan parameter *l* (List) menjadi *next node* dari *list* (*l->next*). Hal ini dilakukan untuk menggeser *next* sampai *list* sudah habis atau sampai ketemu *node* yang dicari.

➤ Print All



```
void printAll(List l){
    if(!isEmpty(l)){
        printf("%d - ", l->x);
        printAll(l->next);
    }
}
```

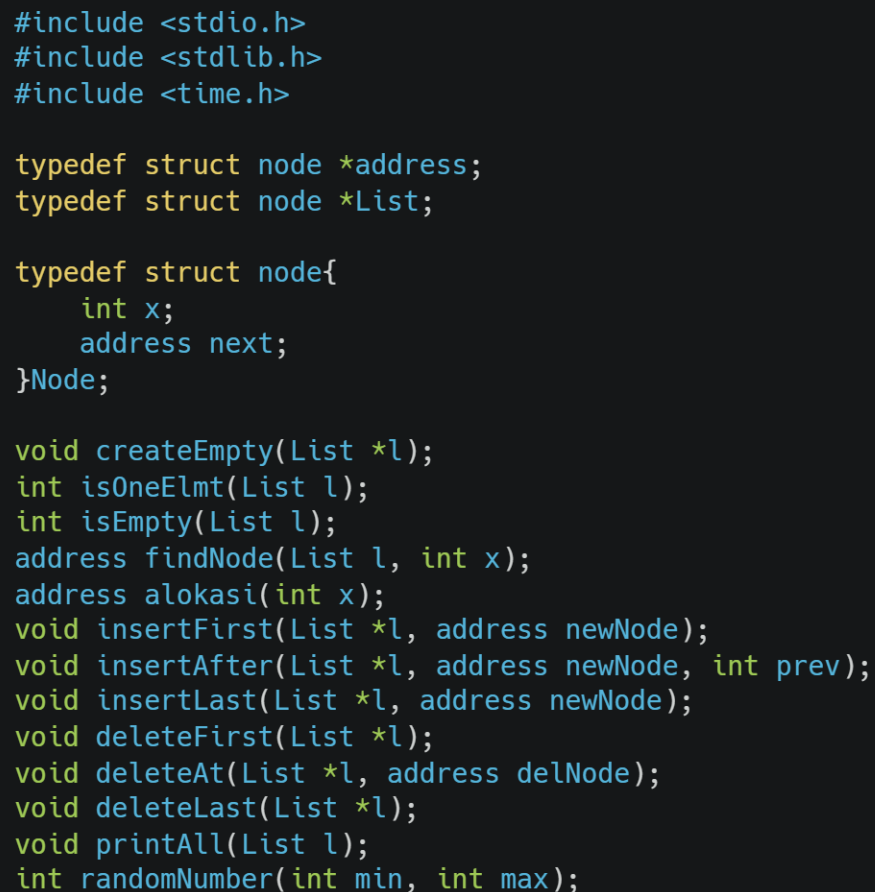
Gambar 9. Print All pada Recursive List

Langkah:

1. Selama *list* tidak dalam keadaan kosong (`isEmpty(l)`), maka tampilkan data saat ini dan lakukan *recursive* dengan memanggil kembali fungsi *print all* yang parameter *l* (List) adalah *next node* dari *list* (`l->next`). Hal ini dilakukan untuk menggeser *next* sampai *list* sudah habis.

C. Guided

1. Header



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

typedef struct node *address;
typedef struct node *List;

typedef struct node{
    int x;
    address next;
}Node;

void createEmpty(List *l);
int isOneElmt(List l);
int isEmpty(List l);
address findNode(List l, int x);
address alokasi(int x);
void insertFirst(List *l, address newNode);
void insertAfter(List *l, address newNode, int prev);
void insertLast(List *l, address newNode);
void deleteFirst(List *l);
void deleteAt(List *l, address delNode);
void deleteLast(List *l);
void printAll(List l);
int randomNumber(int min, int max);
```

Gambar 10. Source code untuk Header

2. Source

```
#include "header.h"

void createEmpty(List *l){
    (*l) = NULL;
}

int isOneElmt(List l){
    return l->next == NULL;
}

int isEmpty(List l){
    return l == NULL;
}

address findNode(List l, int x){
    if(isEmpty(l)){
        return NULL;
    }else{
        if(l->x == x){
            return l;
        }
        return findNode(l->next, x);
    }
}

address alokasi(int x){
    address p;
    p = (Node*) malloc(sizeof(Node));

    p->x = x;
    p->next = NULL;

    return p;
}

void insertFirst(List *l, address newNode){
    newNode->next = (*l);
    (*l) = newNode;
}

void insertAfter(List *l, address newNode, int prev){
    address p = findNode(*l, prev);

    if(p!=NULL){
        newNode->next = p->next;
        p->next = newNode;
        printf("\n[+] Insert After %d", prev);
    }else{
        printf("\nData Tidak Ada");
    }
}
```

Gambar 11.a. Source code untuk Source 1

```
void insertLast(List *l, address newNode){
    if(isEmpty(*l)){
        insertFirst(l, newNode);
        printf("\n[+] Insert Last %d", newNode->x);
    }else{
        insertLast(&(*l)->next, newNode);
    }
}

void deleteFirst(List *l){
    address del = (*l);

    (*l) = (*l)->next;
    free(del);
}

void deleteAt(List *l, address delNode){
    if((*l)==delNode){
        printf("\n[-] Delete At %d", (*l)->x);
        deleteFirst(l);
    }else{
        deleteAt(&(*l)->next, delNode);
    }
}

void deleteLast(List *l){
    if(isOneElmt(*l)){
        printf("\n[-] Delete Last %d", (*l)->x);
        deleteFirst(&(*l));
    }else{
        deleteLast(&(*l)->next);
    }
}

void printAll(List l){
    if(!isEmpty(l)){
        printf("%d - ", l->x);
        printAll(l->next);
    }
}

int randomNumber(int min, int max){
    return rand()%(max-min+1)+min;
}
```

Gambar 11.b. Source code untuk Source 2

3. Main

```

#include "header.h"

int main(int argc, char *argv[]) {
    srand(time(NULL));

    int menu, submenu, x, prev;

    List l;

    address delNode;

    createEmpty(&l);

    do{
        system("cls");
        printf("\nGD Recursive List\n");
        printf("\nData: \n");
        printAll(l);
        printf("\n\n[1] Insert");
        printf("\n[2] Delete");
        printf("\n[3] Find");
        printf("\n[4] Print All\n");
        printf("\n[0] keluar");
        printf("\n>> ");scanf("%d", &menu);
        switch(menu){
            case 1:
                x = randomNumber(0, 10);
                printf("\nInput Data: %d", x);

                printf("\n\n\t[1] Insert First");
                printf("\n\t[2] Insert After");
                printf("\n\t[3] Insert Last");
                printf("\n\t>> ");scanf("%d", &submenu);
                switch(submenu){
                    case 1:
                        insertFirst(&l, alokasi(x));
                        printf("\n[+] Insert First %d", x);
                        break;

                    case 2:
                        printf("\n\tMasukkan Setelah Data : "); scanf("%d", &prev);
                        insertAfter(&l, alokasi(x), prev);
                        break;

                    case 3:
                        insertLast(&l, alokasi(x));
                        break;

                    default:
                        printf("\n\tMenu Invalid");
                        break;
                }
            break;
        }
    }
}

```

Gambar 12.a. Source code untuk Main 1

```

case 2:
    printf("\n\t[1] Delete First");
    printf("\n\t[2] Delete At");
    printf("\n\t[3] Delete Last");
    printf("\n\t>> "); scanf("%d", &submenu);
    switch(submenu){
        case 1:
            if(isEmpty(l))
                break;

            printf("\n[-] Delete First %d", l->x);
            deleteFirst(&l);
            break;

        case 2:
            if(isEmpty(l))
                break;

            printf("\nData yang Ingin Dihapus: "); scanf("%d", &x);
            delNode = findNode(l, x);
            if(delNode!=NULL){
                deleteAt(&l, delNode);
            }else{
                printf("\nData Tidak Ada");
            }
            break;

        case 3:
            if(isEmpty(l))
                break;

            deleteLast(&l);
            break;

        default:
            break;
    }
    break;

```

Gambar 12.b. Source code untuk Main 2

```

case 3:
    printf("\nData yang Ingin Dicari: "); scanf("%d", &x);
    if(findNode(l, x)==NULL){
        printf("\nData Tidak Ada");
    }else{
        printf("\nData Ada");
    }
    break;

case 4:
    printf("\nData : ");
    printAll(l);
    break;

case 0:
    printf("\n[NAMA - NPM - KELAS]"); //ganti dengan identitas masing-masing
    break;

default:
    printf("\nMenu Invalid");
    break;
}getch();
}while(menu!=0);
return 0;
}

```

Gambar 12.c. Source code untuk Main 3

Ketentuan Pengerjaan:

Pastikan ekstensi program file main adalah .c **bukan** .cpp

1. Pastikan semua file dikerjakan dalam satu folder (termasuk file .dev dan .c).
2. Pastikan terdapat 3 file (header, source, dan main) dalam folder tersebut.
3. Guided dikerjakan dalam folder dengan format **GD10_X_YYYYY** lalu di zip.

Note :

X = Kelas

YYYYY = 5 digit terakhir NPM

Kesalahan pengumpulan format penamaan akan dikurangi 10 poin.

Harap diperhatikan!

D. Hint

Saya akan memberikan *hint* buat teman-teman untuk menghadapi UGD *recursive list*, silahkan untuk teman-teman mempelajari bagaimana cara untuk memanipulasi *list* menggunakan cara *recursive*, seperti :

- Memisahkan *list*
- Menggabungkan *list*
- Menampilkan *list*
- Mencari *node* pada *list*
- Menghitung jumlah *node* pada *list*
- Meng-*sorting* *list*
- Menambahkan *node* pada *list*
- Meng-random *node* pada *list*
- Menghapus *node* pada *list*
- Membalikkan *list*

Semoga teman-teman dapat memahami logika *recursive* secara baik, guna untuk mengerjakan UGD maupun TGS.

~ Good luck ~
