



Struktur Data
Pendidikan Teknik Informatika
Universitas Negeri Padang
TIM DOSEN ©2022

JOBSHEET (JS-03)

Linked List

| Fakultas | Proram Studi | Kode MK | Waktu |
|----------|-------------------------------|--------------|--------------|
| Teknik | Pendidikan Teknik Informatika | TIK1.61.2317 | 4 x 50 Menit |

TUJUAN PRAKTIKUM

-
1. Mahasiswa mampu mengaplikasikan konsep linked list dalam menyelesaikan kasus.

HARDWARE & SOFTWARE

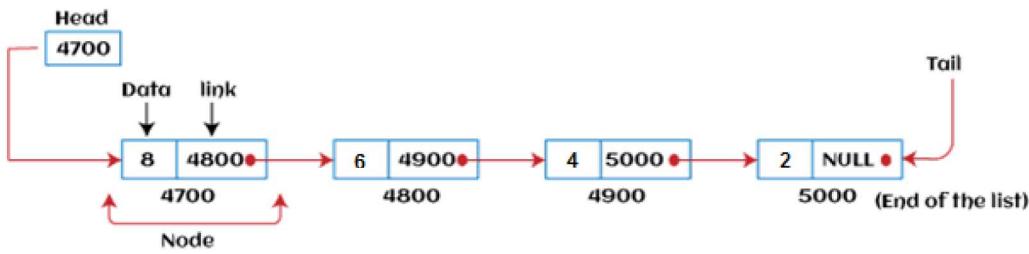
1. Personal Computer
 2. IDE: Dev C++
-

TEORI SINGKAT

Linked list adalah struktur data linier yang mencakup serangkaian simpul yang terhubung. Linked list dapat didefinisikan sebagai node yang disimpan secara acak dalam memori. Sebuah node dalam linked list berisi dua bagian, yaitu, pertama adalah bagian data dan kedua adalah bagian alamat. Node terakhir dari daftar berisi pointer ke nol. Setelah array, linked list adalah struktur data kedua yang paling banyak digunakan. Dalam linked list, setiap tautan berisi koneksi ke tautan lain.

A. Representasi dari linked list

Linked list dapat direpresentasikan sebagai koneksi node di mana setiap node menunjuk ke node berikutnya dari daftar. Representasi dari linked list ditunjukkan di bawah ini :



Sampai sekarang, kita telah menggunakan struktur data array untuk mengatur kelompok elemen yang akan disimpan satu per satu dalam memori. Namun, Array memiliki beberapa kelebihan dan kekurangan yang harus diketahui untuk menentukan struktur data yang akan digunakan di seluruh program.

B. Mengapa menggunakan linked list daripada array?

Linked list adalah struktur data yang mengatasi keterbatasan array. Mari kita lihat beberapa batasan array :

1. Ukuran array harus diketahui terlebih dahulu sebelum digunakan dalam program.
2. Meningkatkan ukuran array adalah proses yang memakan waktu. Hampir tidak mungkin untuk memperluas ukuran array saat dijalankan.
3. Semua elemen dalam array harus disimpan secara berurutan dalam memori. Menyisipkan elemen dalam array membutuhkan pergeseran dari semua pendahulunya.

Linked list berguna karena :

1. Ini mengalokasikan memori secara dinamis. Semua node dari linked list tidak disimpan secara berurutan dalam memori dan dihubungkan bersama dengan bantuan pointer.
2. Dalam linked list, ukuran tidak lagi menjadi masalah karena kita tidak perlu mendefinisikan ukurannya pada saat deklarasi. Daftar bertambah sesuai permintaan program dan terbatas pada ruang memori yang tersedia.

C. Deklarasikan linked list

Sangat mudah untuk mendeklarasikan array, karena bertipe tunggal, sedangkan deklarasi linked list sedikit lebih khas daripada array. Linked list berisi dua bagian, dan keduanya memiliki tipe yang berbeda, yaitu, satu adalah variabel sederhana, sedangkan yang lain adalah variabel penunjuk. Kita dapat mendeklarasikan linked list dengan menggunakan **struktur tipe data yang ditentukan pengguna**.

Deklarasi linked list diberikan sebagai berikut -

```
struct node
{
    int data;
    struct node *next;
}
```

Dalam deklarasi di atas, kita telah mendefinisikan sebuah struktur bernama **node** yang berisi dua variabel, satu adalah **data** yang bertipe integer, dan yang lainnya **adalah** pointer yang berisi alamat node berikutnya.

D. Jenis Linked list

Linked list diklasifikasikan ke dalam jenis berikut :

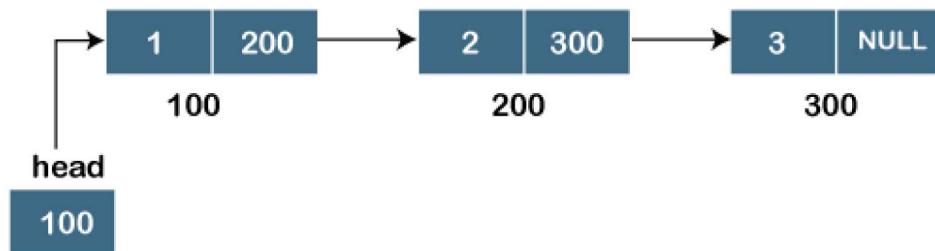
1. Linked list tunggal (Singly-linked list)

Linked list tunggal dapat didefinisikan sebagai kumpulan elemen yang diurutkan. Sebuah node dalam linked list tunggal terdiri dari dua bagian: bagian data dan bagian tautan. Bagian data dari node menyimpan informasi aktual yang akan diwakili oleh node, sedangkan bagian link dari node menyimpan alamat penerus langsungnya.

Linked list tunggal adalah linked list yang umum digunakan dalam program. Jika kita berbicara tentang linked list, itu berarti linked list tunggal. Linked list tunggal adalah struktur data yang berisi dua bagian, yaitu, satu adalah bagian data, dan yang lainnya adalah bagian alamat, yang berisi alamat

node berikutnya atau penerusnya. Bagian alamat dalam sebuah node juga dikenal sebagai pointer .

Misalkan kita memiliki tiga node, dan alamat dari ketiga node ini masing-masing adalah 100, 200 dan 300. Representasi dari tiga node sebagai linked list ditunjukkan pada gambar di bawah ini:



Kita dapat mengamati pada gambar di atas bahwa ada tiga node berbeda yang memiliki alamat masing-masing 100, 200 dan 300. Node pertama berisi alamat node berikutnya, yaitu 200, node kedua berisi alamat node terakhir, yaitu 300, dan node ketiga berisi nilai NULL di bagian alamatnya karena tidak menunjuk ke node manapun. Pointer yang menyimpan alamat node awal dikenal sebagai pointer kepala.

Linked list, yang ditunjukkan pada diagram di atas, dikenal sebagai linked list tunggal karena hanya berisi satu tautan. Dalam daftar ini, hanya traversal maju yang dimungkinkan; kita tidak dapat melintasi arah mundur karena hanya memiliki satu tautan dalam daftar.

Representasi node dalam linked list tunggal :

```
struct node
{
    int data;
    struct node *next;
}
```

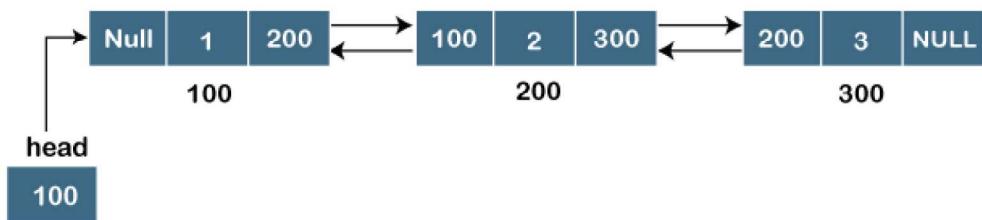
Dalam representasi di atas, kami telah mendefinisikan struktur yang ditentukan pengguna bernama node yang berisi dua anggota, yang pertama adalah data tipe integer, dan yang lainnya adalah pointer (next) dari tipe node.

2. Linked list ganda (Doubly linked list)

Linked list ganda adalah jenis linked list yang kompleks di mana sebuah simpul berisi penunjuk ke simpul sebelumnya serta simpul berikutnya dalam urutan. Oleh karena itu, dalam linked list ganda, sebuah simpul terdiri dari tiga bagian: data simpul, penunjuk ke simpul berikutnya secara berurutan (penunjuk berikutnya), dan penunjuk ke simpul sebelumnya (penunjuk sebelumnya).

Seperti namanya, linked list ganda berisi dua petunjuk. Kita dapat mendefinisikan linked list ganda sebagai struktur data linier dengan tiga bagian: bagian data dan dua bagian alamat lainnya. Dengan kata lain, linked list ganda adalah daftar yang memiliki tiga bagian dalam satu simpul, termasuk satu bagian data, penunjuk ke simpul sebelumnya, dan penunjuk ke simpul berikutnya.

Misalkan kita memiliki tiga node, dan alamat node ini masing-masing adalah 100, 200 dan 300. Representasi dari node-node ini dalam linked list ganda ditunjukkan di bawah ini:



Seperti yang dapat kita amati pada gambar di atas, simpul dalam linked list ganda memiliki dua bagian alamat; satu bagian menyimpan alamat berikutnya sementara bagian lain dari node menyimpan alamat node sebelumnya . Node awal dalam linked list ganda memiliki nilai NULL di bagian alamat, yang menyediakan alamat node sebelumnya.

Representasi node dalam linked list ganda :

```
struct node
{
    int data;
    struct node *next;
    struct node *prev;
}
```

Dalam representasi di atas, kami telah mendefinisikan struktur yang ditentukan pengguna bernama node dengan tiga anggota, satu adalah data tipe integer, dan dua lainnya adalah pointer, yaitu, next dan prev dari tipe node. Variabel pointer berikutnya menyimpan alamat node berikutnya, dan pointer sebelumnya menyimpan alamat node sebelumnya. Tipe dari kedua pointer, yaitu next dan prev adalah struct node karena kedua pointer tersebut menyimpan alamat node dari tipe node struct .

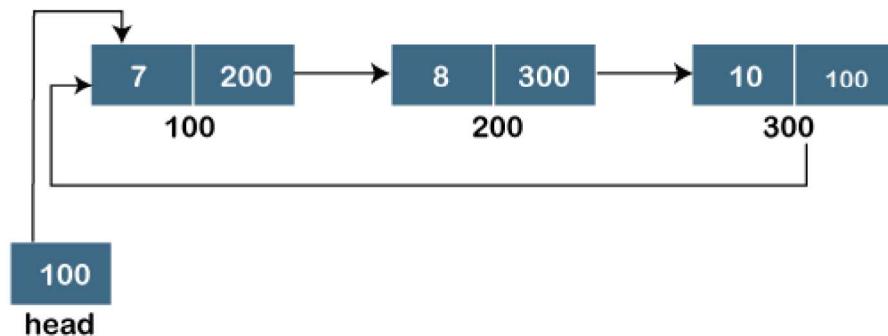
3. Linked list tunggal melingkar (Circular singly linked list)

Dalam linked list tunggal melingkar, simpul terakhir dari daftar berisi penunjuk ke simpul pertama dari daftar. Kita dapat memiliki linked list tunggal melingkar serta linked list ganda melingkar.

Linked list melingkar adalah variasi dari linked list tunggal. Satu-satunya perbedaan antara linked list tunggal dan linked list melingkar adalah bahwa simpul terakhir tidak menunjuk ke simpul mana pun dalam linked list tunggal, sehingga bagian tautannya berisi nilai NULL. Di sisi lain, linked list melingkar adalah daftar di mana simpul terakhir terhubung ke simpul pertama, sehingga bagian tautan dari simpul terakhir menyimpan alamat simpul pertama. Linked list melingkar tidak memiliki simpul awal dan akhir. Kita bisa melintasi ke segala arah, yakni mundur atau maju. Representasi diagram dari linked list melingkar ditunjukkan di bawah ini:

```
struct node
{
    int data;
    struct node *next;
}
```

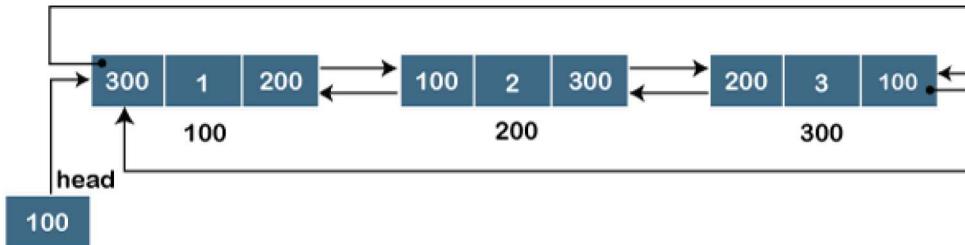
Linked list melingkar adalah urutan elemen di mana setiap simpul memiliki tautan ke simpul berikutnya, dan simpul terakhir memiliki tautan ke simpul pertama. Representasi dari linked list melingkar akan mirip dengan linked list tunggal, seperti yang ditunjukkan di bawah ini:



4. Linked list ganda melingkar (Circular doubly linked list)

Linked list ganda melingkar adalah jenis struktur data yang lebih kompleks di mana sebuah simpul berisi penunjuk ke simpul sebelumnya serta simpul berikutnya. Linked list ganda melingkar tidak mengandung NULL di salah satu node. Node terakhir dari daftar berisi alamat node pertama dari daftar. Node pertama dari daftar juga berisi alamat node terakhir di pointer sebelumnya.

Linked list melingkar ganda memiliki fitur linked list melingkar dan linked list ganda .



Gambar di atas menunjukkan representasi dari linked list melingkar ganda di mana simpul terakhir dilampirkan ke simpul pertama dan dengan demikian menciptakan sebuah lingkaran. Ini adalah linked list ganda juga karena setiap node juga menyimpan alamat node sebelumnya. Perbedaan utama antara linked list ganda dan linked list melingkar ganda adalah bahwa linked list ganda tidak mengandung nilai NULL di bidang node sebelumnya. Karena tautan melingkar ganda berisi tiga bagian, yaitu, dua bagian alamat dan satu bagian data, maka representasinya mirip dengan linked list ganda.

```
struct node
{
    int data;
    struct node *next;
    struct node *prev;
}
```

E. Keuntungan dari linked list

Keuntungan menggunakan linked list adalah sebagai berikut :

1. Struktur data dinamis

Ukuran linked list dapat bervariasi sesuai dengan persyaratan. Linked list tidak memiliki ukuran tetap.

2. Penyisipan dan penghapusan

Tidak seperti array, penyisipan, dan penghapusan dalam linked list lebih mudah. Elemen array disimpan di lokasi yang berurutan, sedangkan elemen dalam

linked list disimpan di lokasi acak. Untuk menyisipkan atau menghapus elemen dalam array, kita harus menggeser elemen untuk membuat spasi. Sedangkan pada linked list, alih-alih menggeser, kita hanya perlu mengupdate alamat pointer dari node tersebut.

3. **Hemat memori**

Ukuran linked list dapat bertambah atau berkurang sesuai dengan kebutuhan, sehingga konsumsi memori dalam linked list menjadi efisien.

4. **Implementasi**

Dapat mengimplementasikan tumpukan dan antrian menggunakan linked list.

F. Kekurangan Linked list

Batasan menggunakan linked list adalah sebagai berikut :

1. **Penggunaan memori**

Dalam linked list, node menempati lebih banyak memori daripada array. Setiap node dari linked list menempati dua jenis variabel, yaitu, satu adalah variabel sederhana, dan satu lagi adalah variabel pointer.

2. **Traversal**

Traversal tidak mudah dalam linked list. Jika kita harus mengakses sebuah elemen dalam linked list, kita tidak dapat mengaksesnya secara acak, sedangkan dalam kasus array kita dapat mengaksesnya secara acak dengan indeks. Misalnya, jika kita ingin mengakses node ke-3, maka kita harus melintasi semua node sebelumnya. Jadi, waktu yang dibutuhkan untuk mengakses node tertentu besar.

3. **Reverse traversing**

Backtracking atau reverse traversing sulit dilakukan dalam linked list. Dalam linked list ganda, lebih mudah tetapi membutuhkan lebih banyak memori untuk menyimpan penunjuk belakang.

G. Operasi yang dapat dilakukan pada Linked list

Operasi dasar yang didukung oleh linked list adalah sebagai berikut -

1. **Penyisipan** - Operasi ini dilakukan untuk menambahkan elemen ke dalam daftar.
2. **Penghapusan** - Ini dilakukan untuk menghapus operasi dari daftar.
3. **Tampilan** - Ini dilakukan untuk menampilkan elemen daftar.
4. **Pencarian** - Ini dilakukan untuk mencari elemen dari daftar menggunakan kunci yang diberikan.

LATIHAN

1. Singly Linked list

```
/* Nama File : Singly Linked list
Pembuat      : tuliskan nama dan NIM anda
Tgl pembuatan : tuliskan tanggal hari ini*/

#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *head;

void begininsert ();
void lastinsert ();
void randominsert();
void begin_delete();
void last_delete();
void random_delete();
void display();
void search();
int main ()
{
    int choice =0;
    while(choice != 9)
    {
        printf("\n*****Singly Linked
list*****\n");
    }
}
```

```

        printf("\n*****Main
Menu*****\n");
        printf("\nChoose one option from the following
list ...\\n");

printf("\n=====
n");
        printf("\n1.Insert in begining\\n2.Insert at
last\\n3.Insert at any random location\\n4.Delete from
Beginning\\n5.Delete from last\\n6.Delete node after
specified location\\n7.Search for an
element\\n8.Show\\n9.Exit\\n");
        printf("\nEnter your choice?\n");
        scanf("\n%d", &choice);
        switch(choice)
        {
            case 1:
                begininsert();
                break;
            case 2:
                lastinsert();
                break;
            case 3:
                randominsert();
                break;
            case 4:
                begin_delete();
                break;
            case 5:
                last_delete();
                break;
            case 6:
                random_delete();
                break;
            case 7:
                search();
                break;
            case 8:
                display();
                break;
            case 9:
                exit(0);
                break;
            default:

```

```
        printf("Please enter valid choice..");
    }
}
}

void begininsert()
{
    struct node *ptr;
    int item;
    ptr = (struct node *) malloc(sizeof(struct node *));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter value\n");
        scanf("%d",&item);
        ptr->data = item;
        ptr->next = head;
        head = ptr;
        printf("\nNode inserted");
    }
}

void lastinsert()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node*)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter value?\n");
        scanf("%d",&item);
        ptr->data = item;
        if(head == NULL)
        {
            ptr -> next = NULL;
            head = ptr;
            printf("\nNode inserted");
        }
    }
}
```

```

        else
        {
            temp = head;
            while (temp -> next != NULL)
            {
                temp = temp -> next;
            }
            temp->next = ptr;
            ptr->next = NULL;
            printf("\nNode inserted");
        }
    }
void randominsert()
{
    int i,loc,item;
    struct node *ptr, *temp;
    ptr = (struct node *) malloc (sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter element value");
        scanf("%d",&item);
        ptr->data = item;
        printf("\nEnter the location after which you want
to insert ");
        scanf("\n%d",&loc);
        temp=head;
        for(i=0;i<loc;i++)
        {
            temp = temp->next;
            if(temp == NULL)
            {
                printf("\ncan't insert\n");
                return;
            }
        }
        ptr ->next = temp ->next;
        temp ->next = ptr;
    }
}

```

```
        printf("\nNode inserted");
    }
}
void begin_delete()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\nList is empty\n");
    }
    else
    {
        ptr = head;
        head = ptr->next;
        free(ptr);
        printf("\nNode deleted from the begining ... \n");
    }
}
void last_delete()
{
    struct node *ptr, *ptr1;
    if(head == NULL)
    {
        printf("\nlist is empty");
    }
    else if(head -> next == NULL)
    {
        head = NULL;
        free(head);
        printf("\nOnly node of the list deleted ... \n");
    }
    else
    {
        ptr = head;
        while(ptr->next != NULL)
        {
            ptr1 = ptr;
            ptr = ptr ->next;
        }
        ptr1->next = NULL;
        free(ptr);
        printf("\nDeleted Node from the last ... \n");
    }
}
```

```

}

void random_delete()
{
    struct node *ptr,*ptr1;
    int loc,i;
    printf("\n Enter the location of the node after which
you want to perform deletion \n");
    scanf("%d",&loc);
    ptr=head;
    for(i=0;i<loc;i++)
    {
        ptr1 = ptr;
        ptr = ptr->next;

        if(ptr == NULL)
        {
            printf("\nCan't delete");
            return;
        }
        ptr1 ->next = ptr ->next;
        free(ptr);
        printf("\nDeleted node %d ",loc+1);
    }
}

void search()
{
    struct node *ptr;
    int item,i=0,flag;
    ptr = head;
    if(ptr == NULL)
    {
        printf("\nEmpty List\n");
    }
    else
    {
        printf("\nEnter item which you want to
search?\n");
        scanf("%d",&item);
        while (ptr!=NULL)
        {
            if(ptr->data == item)
            {
                printf("item found at location %d ",i+1);
                flag=0;
            }
        }
    }
}

```

```
        }
        else
        {
            flag=1;
        }
        i++;
        ptr = ptr -> next;
    }
    if(flag==1)
    {
        printf("Item not found\n");
    }
}

void display()
{
    struct node *ptr;
    ptr = head;
    if(ptr == NULL)
    {
        printf("Nothing to print");
    }
    else
    {
        printf("\nprinting values . . . .\n");
        while (ptr!=NULL)
        {
            printf("\n%d",ptr->data);
            ptr = ptr -> next;
        }
    }
}
```

2. Doubly Linked list

```
/* Nama File : Doubly Linked list
Pembuat      : tuliskan nama dan NIM anda
Tgl pembuatan : tuliskan tanggal hari ini*/  
  
#include<stdio.h>
#include<stdlib.h>
```

```

struct node
{
    struct node *prev;
    struct node *next;
    int data;
};

struct node *head;
void insertion_beginning();
void insertion_last();
void insertion_specified();
void deletion_beginning();
void deletion_last();
void deletion_specified();
void display();
void search();
int main ()
{
int choice =0;
    while(choice != 9)
    {
        printf("\n*****Doubly Linked
list*****\n");
        printf("\n*****Main
Menu*****\n");
        printf("\nChoose one option from the following
list ...\\n");

printf("\n=====\
n");
        printf("\n1.Insert in begining\\n2.Insert at
last\\n3.Insert at any random location\\n4.Delete from
Beginning\\n5.Delete from last\\n6.Delete the node after the
given data\\n7.Search\\n8.Show\\n9.Exit\\n");
        printf("\nEnter your choice?\\n");
        scanf("\\n%d", &choice);
        switch(choice)
        {
            case 1:
                insertion_beginning();
                break;
            case 2:
                insertion_last();
                break;
            case 3:
                insertion_specified();
                break;
            case 4:

```

```
        deletion_beginning();
        break;
    case 5:
        deletion_last();
        break;
    case 6:
        deletion_specified();
        break;
    case 7:
        search();
        break;
    case 8:
        display();
        break;
    case 9:
        exit(0);
        break;
    default:
        printf("Please enter valid choice..");
    }
}
}
void insertion_beginning()
{
    struct node *ptr;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter Item value");
        scanf("%d",&item);

        if(head==NULL)
        {
            ptr->next = NULL;
            ptr->prev=NULL;
            ptr->data=item;
            head=ptr;
        }
        else
        {
            ptr->data=item;
            ptr->prev=NULL;
```

```

        ptr->next = head;
        head->prev=ptr;
        head=ptr;
    }
    printf("\nNode inserted\n");
}

}

void insertion_last()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *) malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter value");
        scanf("%d",&item);
        ptr->data=item;
        if(head == NULL)
        {
            ptr->next = NULL;
            ptr->prev = NULL;
            head = ptr;
        }
        else
        {
            temp = head;
            while(temp->next!=NULL)
            {
                temp = temp->next;
            }
            temp->next = ptr;
            ptr ->prev=temp;
            ptr->next = NULL;
        }
    }
    printf("\nnode inserted\n");
}
void insertion_specified()
{
    struct node *ptr,*temp;
    int item,loc,i;

```

```
ptr = (struct node *)malloc(sizeof(struct node));
if(ptr == NULL)
{
    printf("\n OVERFLOW");
}
else
{
    temp=head;
    printf("Enter the location");
    scanf("%d",&loc);
    for(i=0;i<loc;i++)
    {
        temp = temp->next;
        if(temp == NULL)
        {
            printf("\n There are less than %d
elements", loc);
            return;
        }
    }
    printf("Enter value");
    scanf("%d",&item);
    ptr->data = item;
    ptr->next = temp->next;
    ptr -> prev = temp;
    temp->next = ptr;
    temp->next->prev=ptr;
    printf("\nnode inserted\n");
}
}
void deletion_beginning()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\n UNDERFLOW");
    }
    else if(head->next == NULL)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }
    else
    {
        ptr = head;
        head = head -> next;
    }
}
```

```

        head -> prev = NULL;
        free(ptr);
        printf("\nnode deleted\n");
    }

}

void deletion_last()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\n UNDERFLOW");
    }
    else if(head->next == NULL)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }
    else
    {
        ptr = head;
        if(ptr->next != NULL)
        {
            ptr = ptr -> next;
        }
        ptr -> prev -> next = NULL;
        free(ptr);
        printf("\nnode deleted\n");
    }
}

void deletion_specified()
{
    struct node *ptr, *temp;
    int val;
    printf("\n Enter the data after which the node is to
be deleted : ");
    scanf("%d", &val);
    ptr = head;
    while(ptr -> data != val)
    ptr = ptr -> next;
    if(ptr -> next == NULL)
    {
        printf("\nCan't delete\n");
    }
    else if(ptr -> next -> next == NULL)
    {

```

```
        ptr ->next = NULL;
    }
else
{
    temp = ptr -> next;
    ptr -> next = temp -> next;
    temp -> next -> prev = ptr;
    free(temp);
    printf("\nnode deleted\n");
}
}
void display()
{
    struct node *ptr;
    printf("\n printing values...\n");
    ptr = head;
    while(ptr != NULL)
    {
        printf("%d\n",ptr->data);
        ptr=ptr->next;
    }
}
void search()
{
    struct node *ptr;
    int item,i=0,flag;
    ptr = head;
    if(ptr == NULL)
    {
        printf("\nEmpty List\n");
    }
    else
    {
        printf("\nEnter item which you want to
search?\n");
        scanf("%d",&item);
        while (ptr!=NULL)
        {
            if(ptr->data == item)
            {
                printf("\nitem found at location %d
",i+1);
                flag=0;
                break;
            }
            else
            {

```

```

        flag=1;
    }
    i++;
    ptr = ptr -> next;
}
if(flag==1)
{
    printf("\nItem not found\n");
}
}
}

```

3. Circular Singly Linked list

```

/* Nama File : Circular Singly Linked list
Pembuat      : tuliskan nama dan NIM anda
Tgl pembuatan : tuliskan tanggal hari ini*/

#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *head;

void begininsert ();
void lastinsert ();
void randominsert();
void begin_delete();
void last_delete();
void random_delete();
void display();
void search();
int main ()
{
    int choice =0;
    while(choice != 7)
    {
        printf("\n*****Circular Singly Linked
list*****\n");
        printf("\n*****Main
Menu*****\n");
        printf("\nChoose one option from the following
list ... \n");

```

```
printf("\n=====\
n");
    printf("\n1.Insert in begining\n2.Insert at
last\n3.Delete from Beginning\n4.Delete from
last\n5.Search for an element\n6.Show\n7.Exit\n");
    printf("\nEnter your choice?\n");
    scanf("\n%d",&choice);
    switch(choice)
    {
        case 1:
        begininsert();
        break;
        case 2:
        lastinsert();
        break;
        case 3:
        begin_delete();
        break;
        case 4:
        last_delete();
        break;
        case 5:
        scarch();
        break;
        case 6:
        display();
        break;
        case 7:
        exit(0);
        break;
        default:
        printf("Please enter valid choice..");
    }
}
}

void begininsert()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
}
```

```

    }
else
{
    printf("\nEnter the node data?");
    scanf("%d",&item);
    ptr -> data = item;
    if(head == NULL)
    {
        head = ptr;
        ptr -> next = head;
    }
    else
    {
        temp = head;
        while(temp->next != head)
            temp = temp->next;
        ptr->next = head;
        temp -> next = ptr;
        head = ptr;
    }
    printf("\nnode inserted\n");
}

}
void lastinsert()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW\n");
    }
    else
    {
        printf("\nEnter Data?");
        scanf("%d",&item);
        ptr->data = item;
        if(head == NULL)
        {
            head = ptr;
            ptr -> next = head;
        }
        else

```

```
{  
    temp = head;  
    while(temp -> next != head)  
    {  
        temp = temp -> next;  
    }  
    temp -> next = ptr;  
    ptr -> next = head;  
}  
  
printf("\nnode inserted\n");  
}  
}  
  
void begin_delete()  
{  
    struct node *ptr;  
    if(head == NULL)  
    {  
        printf("\nUNDERFLOW");  
    }  
    else if(head->next == head)  
    {  
        head = NULL;  
        free(head);  
        printf("\nnode deleted\n");  
    }  
  
    else  
    {  
        ptr = head;  
        while(ptr -> next != head)  
            ptr = ptr -> next;  
        ptr->next = head->next;  
        free(head);  
        head = ptr->next;  
        printf("\nnode deleted\n");  
    }  
}  
void last_delete()  
{  
    struct node *ptr, *preptr;  
    if(head==NULL)  
    {  
        printf("\nUNDERFLOW");  
    }
```

```

    }
    else if (head ->next == head)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }
    else
    {
        ptr = head;
        while(ptr ->next != head)
        {
            preptr=ptr;
            ptr = ptr->next;
        }
        preptr->next = ptr -> next;
        free(ptr);
        printf("\nnode deleted\n");
    }
}

void search()
{
    struct node *ptr;
    int item,i=0,flag=1;
    ptr = head;
    if(ptr == NULL)
    {
        printf("\nEmpty List\n");
    }
    else
    {
        printf("\nEnter item which you want to
search?\n");
        scanf("%d",&item);
        if(head ->data == item)
        {
            printf("item found at location %d",i+1);
            flag=0;
        }
        else
        {
            while (ptr->next != head)
            {

```

```
        if(ptr->data == item)
        {
            printf("item found at location %d ",i+1);
            flag=0;
            break;
        }
        else
        {
            flag=1;
        }
        i++;
        ptr = ptr -> next;
    }
}
if(flag != 0)
{
    printf("Item not found\n");
}
}

void display()
{
    struct node *ptr;
    ptr=head;
    if(head == NULL)
    {
        printf("\nnothing to print");
    }
    else
    {
        printf("\n printing values ... \n");

        while(ptr -> next != head)
        {

            printf("%d\n", ptr -> data);
            ptr = ptr -> next;
        }
        printf("%d\n", ptr -> data);
    }
}
```

4. Circular Doubly Linked list

```
/* Nama File : Circular Doubly Linked list
Pembuat      : tuliskan nama dan NIM anda
Tgl pembuatan : tuliskan tanggal hari ini*/

#include<stdio.h>
#include<stdlib.h>
struct node
{
    struct node *prev;
    struct node *next;
    int data;
};
struct node *head;
void insertion_beginning();
void insertion_last();
void deletion_beginning();
void deletion_last();
void display();
void search();
int main ()
{
int choice =0;
while(choice != 9)
{
    printf("\n*****Circular Doubly Linked
list*****\n");
    printf("\n*****Main
Menu*****\n");
    printf("\nChoose one option from the following
list ... \n");

printf("\n=====\
n");
    printf("\n1.Insert in Beginning\n2.Insert at
last\n3.Delete from Beginning\n4.Delete from
last\n5.Search\n6.Show\n7.Exit\n");
    printf("\nEnter your choice?\n");
    scanf("\n%d", &choice);
    switch(choice)
    {
        case 1:
            insertion_beginning();
            break;
```

```
        case 2:
            insertion_last();
        break;
        case 3:
            deletion_beginning();
        break;
        case 4:
            deletion_last();
        break;
        case 5:
            search();
        break;
        case 6:
            display();
        break;
        case 7:
            exit(0);
        break;
        default:
            printf("Please enter valid choice..");
    }
}
}

void insertion_beginning()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter Item value");
        scanf("%d",&item);
        ptr->data=item;
        if(head==NULL)
        {
            head = ptr;
            ptr -> next = head;
            ptr -> prev = head;
        }
        else
```

```

{
    temp = head;
    while(temp -> next != head)
    {
        temp = temp -> next;
    }
    temp -> next = ptr;
    ptr -> prev = temp;
    head -> prev = ptr;
    ptr -> next = head;
    head = ptr;
}
printf("\nNode inserted\n");
}

}

void insertion_last()
{
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *) malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter value");
        scanf("%d",&item);
        ptr->data=item;
        if(head == NULL)
        {
            head = ptr;
            ptr -> next = head;
            ptr -> prev = head;
        }
        else
        {
            temp = head;
            while(temp->next != head)
            {
                temp = temp->next;
            }
            temp->next = ptr;
        }
    }
}

```

```
        ptr ->prev=temp;
        head -> prev = ptr;
        ptr -> next = head;
    }
}

printf("\nnode inserted\n");
}

void deletion_beginning()
{
    struct node *temp;
    if(head == NULL)
    {
        printf("\n UNDERFLOW");
    }
    else if(head->next == head)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }
    else
    {
        temp = head;
        while(temp -> next != head)
        {
            temp = temp -> next;
        }
        temp -> next = head -> next;
        head -> next -> prev = temp;
        free(head);
        head = temp -> next;
    }
}

void deletion_last()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\n UNDERFLOW");
    }
    else if(head->next == head)
    {
```

```

        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }
else
{
    ptr = head;
    if(ptr->next != head)
    {
        ptr = ptr -> next;
    }
    ptr -> prev -> next = head;
    head -> prev = ptr -> prev;
    free(ptr);
    printf("\nnode deleted\n");
}
}

void display()
{
    struct node *ptr;
    ptr=head;
    if(head == NULL)
    {
        printf("\nnothing to print");
    }
    else
    {
        printf("\n printing values ... \n");

        while(ptr -> next != head)
        {

            printf("%d\n", ptr -> data);
            ptr = ptr -> next;
        }
        printf("%d\n", ptr -> data);
    }
}

void search()
{
    struct node *ptr;

```

```
int item,i=0,flag=1;
ptr = head;
if(ptr == NULL)
{
    printf("\nEmpty List\n");
}
else
{
    printf("\nEnter item which you want to
search?\n");
    scanf("%d",&item);
    if(head ->data == item)
    {
        printf("item found at location %d",i+1);
        flag=0;
    }
    else
    {
        while (ptr->next != head)
        {
            if(ptr->data == item)
            {
                printf("item found at location %d ",i+1);
                flag=0;
                break;
            }
            else
            {
                flag=1;
            }
            i++;
            ptr = ptr -> next;
        }
    }
    if(flag != 0)
    {
        printf("Item not found\n");
    }
}
```

TUGAS

Buatlah Menu untuk program simulasi dengan **struct album**, yakni:

1. cetak album
 - jika kosong >> cetak “Album masih kosong”
 - jika tidak kosong >> cetak no + title + time
2. tambah lagu di awal album
 - jika kosong >> lagu baru menjadi lagu pertama
 - jika tidak kosong >> lagu ditambahkan di bagian awal album
3. tambah lagu di akhir album
 - jika kosong >> lagu baru menjadi lagu pertama
 - jika tidak kosong >> lagu ditambahkan di bagian akhir album
4. hapus lagu di awal album
 - jika kosong >> cetak “Album masih kosong”
 - jika tidak kosong >> hapus lagu yang paling awal
5. hapus lagu di akhir album
 - jika kosong >> cetak “Album masih kosong”
 - jika tidak kosong >> hapus lagu yang paling akhir
6. exit
 - keluar dari program simulasi

(disarankan untuk membuat function untuk setiap menunya)

DAFTAR PUSTAKA

Kelley, Al and Pohl, Ira. 2003. *C by Dissection: The Essentials of C Programming*. Addison-Weasley.

Khannedy, Eko Kurniawan. 2007. *Diktat Pemrograman C*. Bandung : Unikom.

Liem, Ingriani. 2003. *Catatan Singkat Bahasa C*. Jurusan Teknik Informatika. Institut Teknologi Bandung.

Reema Thareja. 2014. *Data Structures Using C*. India : Oxford University Press.

Solichin, Achmad. 2003. *Pemrograman Bahasa C dengan Turbo C*. Kuliah Berseri ilmukomputer.com