Praktikum Struktur Data
Teknik Informatika | Universitas Negeri Padang
TIM DOSEN ©2023
Page. 1 – XX

JOBSHEET 05

Double Linked List & Circular Linked List

Fakultas	Proram Studi	Kode MK	Waktu
Teknik	Teknik Informatika	INF1.62.2014	2 x 50 Menit

TUJUAN PRAKTIKUM

1. Mahasiswa memahami konsep double linked list dan circular linked list dalam pemrograman C.
2. Mahasiswa mampu mengimplementasikan double linked list dan circular linked list dalam pemrograman C.
3. Mahasiswa mampu menyelesaikan suatu permasalahan sederhana menggunakan double linked list dan circular linked list dalam pemrograman C.

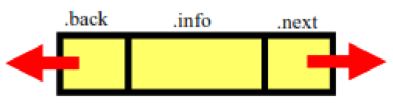
HARDWARE & SOFTWARE

1. Personal Computer
2. DevC++ IDE

TEORI SINGKAT

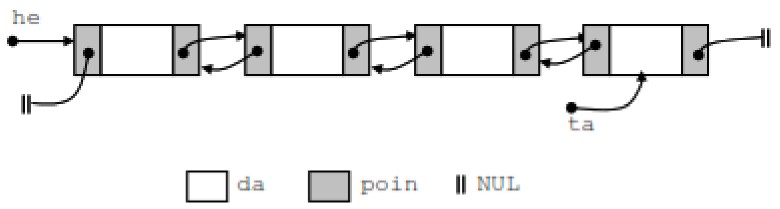
A. Double Linked List

Double linked list Elemen-elemen dihubungkan dengan dua pointer dalam satu node. Struktur ini menyebabkan list melintas baik ke depan (next) maupun ke belakang (prev) atau (back).



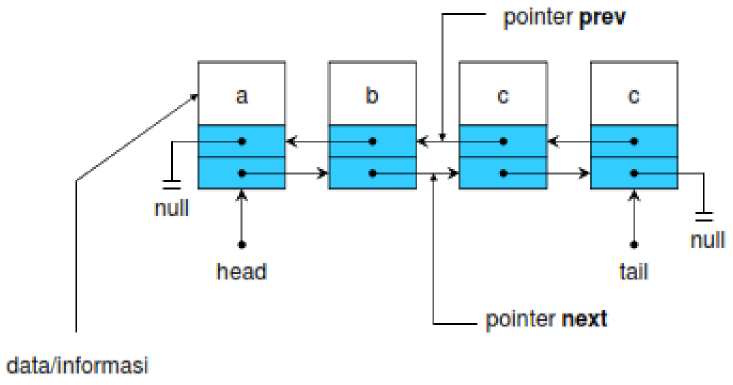
Double Linked List

Masing-masing elemen pada double linked list terdiri dari tiga bagian, disamping data (info) dan pointer next, masing-masing elemen dilengkapi dengan pointer prev atau back yang menunjuk ke elemen sebelumnya. Untuk menunjukkan head dari double linked list, maka pointer prev dari elemen pertama menunjuk NULL. Untuk menunjukkan tail dari double linked list tersebut, maka pointer next dari elemen terakhir menunjuk NULL.



Elemen yang dihubungkan dengan double linked list

Untuk melintas kembali melalui double linked list, dapat digunakan pointer prev dari elemen yang berurutan pada arah tail ke head. Double linked list mempunyai fleksibilitas yang lebih tinggi daripada single linked list dalam perpindahan pada list.

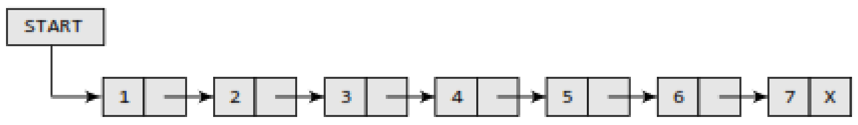


Double Linked List

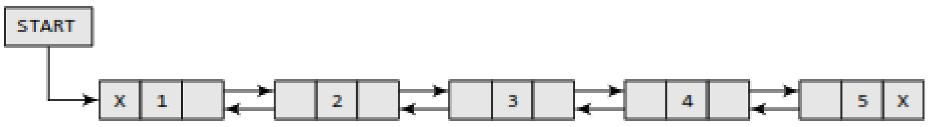
B. Circular Linked List

Circular list adalah bentuk lain dari linked list yang memberikan fleksibilitas dalam melewatkan elemen. Circular list bisa berupa single linked list atau double linked list, tetapi tidak mempunyai tail. Pada circular list, pointer next dari elemen terakhir menunjuk ke elemen pertama dan bukan menunjuk NULL. Pada double linked circular list, pointer prev dari elemen pertama menunjuk ke elemen terakhir.

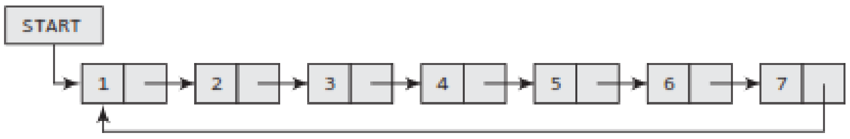
Single Linked List



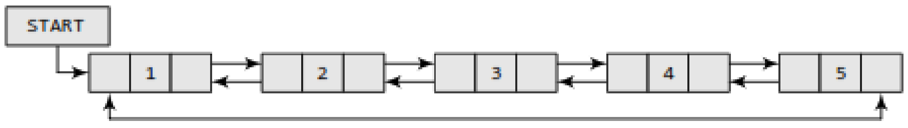
Double Linked List



Circular Linked List



Circular Double Linked List



PERCOBAAN

1. Deklarasi Double Linked List

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

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

struct node *head = NULL;
struct node *tail = NULL;
struct node *current;

int main () {
    current = (struct node *) malloc (sizeof (struct node));
    current -> data = 1;
    current -> next = NULL;
    current -> back = NULL;

    head = tail = current;
    return 0;
}
```

2. Double Linked List

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

struct doublelinkedlist{
    int data;
    doublelinkedlist *next, *prev;
```

```
}*head, *tail, *current;

void insertlast(int data){
    current = (struct doublelinkedlist
*)malloc(sizeof(doublelinkedlist));
    current->data = data;
    current->prev = current->next = NULL;

    if(head==NULL){
          head=tail=current;
    }else{
          current->prev = tail;
          tail->next = current;
          tail = current;
    }
}

void insertfirst(int data){
    current = (struct doublelinkedlist
*)malloc(sizeof(doublelinkedlist));
    current->data=data;
    current->next = current->prev=NULL;

    if(head==NULL){
          head=tail=current;
    }else{
          head->prev=current;
          current->next=head;
          head=current;
    }
}

void insertmid(int data){
    if(head==NULL){
          insertfirst(data);
    }else if(data < head->data){
          insertfirst(data);
    }else if(data > tail->data){
          insertlast(data);
    }else{
          current = (struct doublelinkedlist
*)malloc(sizeof(doublelinkedlist));
          current->data = data;
          current->next = current->prev = NULL;

          struct doublelinkedlist *temp=head;
          while(temp!=NULL && temp->data < current->data){
                temp=temp->next;
          }
          current->prev=temp->prev;
          current->next=temp;

          temp->prev->next=current;
          temp->prev=current;
    }
}

void deletefirst(){
    if(head==NULL){
          printf("No Data\n");
    }else if(head==tail){
          current=head;
          head=tail=NULL;
          free(current);
    }else{
          current=head;
          head=head->next;
          head->prev=NULL;
          free(current);
    }
}

void deletelast(){
    if(head==NULL){
          printf("No Data\n");
    }else if(head==tail){
          current=tail;
          head=tail=NULL;
          free(current);
    }else{
          current=tail;
          tail=tail->prev;
          tail->next=NULL;
          free(current);
    }
}

void deletemid(int data){
    int temu=0;
    if(head==NULL){
          printf("No Data\n");
    }else{
          current=head;
          while(current!=NULL){
                if(current->data==data){
                      temu=1;
                      break;
                }
                current=current->next;
          }
          if(temu==1){
                if(current==head){
                deletefirst();
                }else if(current==tail){
                      deletelast();
                }else{
                      current->prev->next=current->next;
                      current->next->prev=current->prev;
                      free(current);
                }
          }else{
                printf("Data Not Found\n");
          }
    }
}

void deleteall(){
    while(head!=NULL){
            deletefirst();
        }
}

void print(){
    current=head;
    if(current!=NULL){
          while(current!=NULL){
                printf("Data %d\n", current->data);
                current=current->next;
          }
    }else{
          printf("No Data\n");
    }
}

int main(){
    insertfirst(2);
    insertfirst(3);
    insertfirst(4);
    insertfirst(5);
    insertlast(6);
    insertmid(7);
    deletemid(7);
    //popMid(6);
    //popAll();
    print();
    getchar();
}
```

3. **Membuat Program Single Linked List**

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <malloc.h>

struct node
{
    int data;
    struct node *next;
};
    struct node *start = NULL;
    struct node *create_linked_list(struct node *);
    struct node *display(struct node *);
    struct node *insert_beg(struct node *);
    struct node *insert_end(struct node *);
    struct node *insert_before(struct node *);
    struct node *insert_after(struct node *);
    struct node *delete_beg(struct node *);
    struct node *delete_end(struct node *);
    struct node *delete_node(struct node *);
    struct node *delete_after(struct node *);
    struct node *delete_list(struct node *);
    struct node *sort_list(struct node *);

int main(int argc, char *argv[]) {

int option;
```

```
do
{
  printf("\n\n ======= SINGLE LINKED LIST =======\n");
  printf("\n 1:  Buat Single Linked List");
  printf("\n 2:  Tampilkan semua node Single Linked List");
  printf("\n 3:  Tambah node di awal Single Linked List");
  printf("\n 4:  Tambah node di akhir Single Linked List");
  printf("\n 5:  Tambah node sebelum node yang diberikan");
  printf("\n 6:  Tambah node setelah node yang diberikan");
  printf("\n 7:  Hapus node di awal Single Linked List");
  printf("\n 8:  Hapus node di akhir Single Linked List");
  printf("\n 9:  Hapus sebuah node tertentu");
  printf("\n 10: Hapus node setelah node yang diberikan");
  printf("\n 11: Hapus semua node Single Linked List");
  printf("\n 12: Urutkan data Single Linked List");
  printf("\n 13: Keluar dari Program");
  printf("\n\n Masukan pilihan menu : "); scanf("%d", &option);

  switch(option)
  {
        case 1:
        system("cls");
        start = create_linked_list(start);
        printf("\n LINKED LIST CREATED");
        break;

        case 2:
        system("cls");
        start = display(start);
        break;

        case 3:
        system("cls");
        start = insert_beg(start);
        break;

        case 4:
        system("cls");
        start = insert_end(start);
        break;

        case 5:
        system("cls");
        start = insert_before(start);
        break;

        case 6:
        system("cls");
        start = insert_after(start);
        break;

        case 7:
        system("cls");
        start = delete_beg(start);
        break;

        case 8:
        system("cls");
        start = delete_end(start);
```

```
            break;

            case 9:
            system("cls");
            start = delete_node(start);
            break;

            case 10:
            system("cls");
            start = delete_after(start);
            break;

            case 11: start = delete_list(start);
            printf("\n LINKED LIST DELETED");
            break;

            case 12: start = sort_list(start);
            break;
      }
}
    while(option !=13);
    getch();
    return 0;
}

struct node *create_linked_list(struct node *start)
{
    system("cls");
    struct node *new_node, *ptr;
    int num;
    printf("\n Tekan 0 untuk berhenti input data");
    printf("\n Masukan data : ");scanf("%d", &num);
    while(num!=0)
           {
            new_node = (struct node*)malloc(sizeof(struct node));
            new_node -> data=num;
            if(start==NULL)
                  {
                    new_node -> next = NULL;
                    start = new_node;
                  }
                        else
                               {
                                      ptr=start;
                                      while(ptr->next!=NULL)
                                      ptr=ptr->next;
                                      ptr->next = new_node;
                                      new_node->next=NULL;
                               }
                  printf(" Masukan data berikutnya : ");scanf("%d",
&num);
                  }
    return start;
}


struct node *display(struct node *start)
{
    struct node *ptr;
    ptr = start;
    printf(" DATA SINGLE LINKED LIST \n\n");
    while(ptr != NULL)
    {
          printf("\t %d", ptr -> data);
          ptr = ptr -> next;
    }
return start;
}


struct node *insert_beg(struct node *start)
{
    struct node *new_node;
    int num;
    printf(" TAMBAH DATA DI AWAL \n\n");
    printf("\n Masukan data : ");
    scanf("%d", &num);

    new_node = (struct node *)malloc(sizeof(struct node));
    new_node -> data = num;
    new_node -> next = start;
    start = new_node;
    printf("\n DATA BERHASIL DITAMBAHKAN! \n");
return start;
}

struct node *insert_end(struct node *start)
{
    struct node *ptr, *new_node;
    int num;
    printf(" TAMBAH DATA DI AKHIR \n\n");
    printf("\n Masukan data : ");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node -> data = num;
    ptr=start;
    while(ptr -> next != NULL)
    ptr = ptr -> next;
    ptr -> next = new_node;
    new_node -> next = NULL;

    printf("\n DATA BERHASIL DITAMBAHKAN! \n");
return start;
}


struct node *insert_before(struct node *start)
{
    struct node *new_node, *ptr, *preptr;
    int num, val;
    printf(" TAMBAH DATA SEBELUM NODE TERTENTU \n\n");
    printf("\n Masukan data : ");
    scanf("%d", &num);

    printf("\n Data baru ditambahkan sebelum node dengan data :
");scanf("%d", &val);

    new_node = (struct node *)malloc(sizeof(struct node));
    new_node -> data = num;
    ptr = start;

    while(ptr -> data != val)
    {
          preptr = ptr;
          ptr = ptr -> next;
    }

preptr -> next = new_node;
new_node -> next = ptr;
printf("\n DATA BERHASIL DITAMBAHKAN! \n");
return start;
}


struct node *insert_after(struct node *start)
{
    struct node *new_node, *ptr, *preptr;
    int num, val;
    printf(" TAMBAH DATA SETELAH NODE TERTENTU \n\n");
    printf("\n Masukan data : ");
    scanf("%d", &num);
    printf("\n Data baru ditambahkan setelah node dengan data :
");scanf("%d", &val);

    new_node = (struct node *)malloc(sizeof(struct node));
    new_node -> data = num;
    ptr = start;
    preptr = ptr;

    while(preptr -> data != val)
    {
          preptr = ptr;
          ptr = ptr -> next;
    }

preptr -> next=new_node;
new_node -> next = ptr;
printf("\n DATA BERHASIL DITAMBAHKAN! \n");
return start;
}


struct node *delete_beg(struct node *start)
{
    struct node *ptr;
    ptr = start;

    start = start -> next;
    free(ptr);
    printf("\n DATA AWAL BERHASIL DIHAPUS! \n");
    return start;
}


struct node *delete_end(struct node *start)
{
    struct node *ptr, *preptr;
    ptr = start;

    while(ptr -> next != NULL)
    {
          preptr = ptr;
          ptr = ptr -> next;
    }
preptr -> next = NULL;
free(ptr);
printf("\n DATA AKHIR BERHASIL DIHAPUS! \n");
return start;
}


struct node *delete_node(struct node *start)
{
    struct node *ptr, *preptr;
    int val;
    printf("\n HAPUS DATA TERTENTU SINGLE LINKED LIST \n");
    printf("\n Masukan data node yang ingin dihapus : ");
    scanf("%d", &val);

    ptr = start;

    if(ptr -> data == val)
    {
          start = delete_beg(start);
          return start;
    } else
          {
                while(ptr -> data != val)
                {
                      preptr = ptr;
                      ptr = ptr -> next;
                }
    preptr -> next = ptr -> next;
    free(ptr);
    printf("\n DATA BERHASIL DIHAPUS! \n");
    return start;
    }
}


struct node *delete_after(struct node *start)
{
    struct node *ptr, *preptr;
    int val;
    printf("\n HAPUS DATA AWAL SINGLE LINKED LIST \n");
    printf("\n Masukan data setelah node yang akan dihapus : ");
    scanf("%d", &val);
    ptr = start;

    while(ptr -> data != val)

    ptr = ptr->next;
    preptr = ptr -> next;
    ptr -> next = preptr -> next;
    preptr -> next  = ptr;
    free(preptr);
    printf("\n DATA BERHASIL DIHAPUS! \n");
    return start;
}


struct node *delete_list(struct node *start)
{
    struct node *ptr;
    if(start!=NULL)
     {
     ptr=start;
     while(ptr != NULL)
          {
            printf("\n %d is to be deleted next", ptr -> data);
            start = delete_beg(ptr);
            ptr = start;
          }
    }
return start;
}


struct node *sort_list(struct node *start)
{
    printf("\n DATA SINGLE LINKED LIST SUDAH TERURUT DARI K > B!
\n");
    struct node *ptr1, *ptr2;
    int temp;
    ptr1 = start;

    while(ptr1 -> next != NULL)
    {
    ptr2 = ptr1 -> next;
          while(ptr2 != NULL)
          {
                if(ptr1 -> data > ptr2 -> data)
                {
                   temp = ptr1 -> data;
                   ptr1 -> data = ptr2 -> data;
                   ptr2 -> data = temp;
                }
          ptr2 = ptr2 -> next;
          }
    ptr1 = ptr1 -> next;
    }
return start;
}
```

4. Double Linked List

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <stdlib.h>

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

    struct node *start = NULL;
    struct node *create_ll(struct node *);
    struct node *display(struct node *);
    struct node *insert_beg(struct node *);
    struct node *insert_end(struct node *);
    struct node *insert_before(struct node *);
    struct node *insert_after(struct node *);
    struct node *delete_beg(struct node *);
    struct node *delete_end(struct node *);
    struct node *delete_before(struct node *);
    struct node *delete_after(struct node *);
    struct node *delete_list(struct node *);


int main()
{
    int option;
    do
    {
    printf("\n\n ======= DOUBLE LINKED LIST =======\n");
    printf("\n 1:  Buat Double Linked List");
    printf("\n 2:  Tampilkan semua node Double Linked List");
    printf("\n 3:  Tambah node di awal Double Linked List");
    printf("\n 4:  Tambah node di akhir Double Linked List");
    printf("\n 5:  Tambah node sebelum node yang diberikan");
    printf("\n 6:  Tambah node setelah node yang diberikan");
    printf("\n 7:  Hapus node di awal Double Linked List");
    printf("\n 8:  Hapus node di akhir Double Linked List");
    printf("\n 9:  Hapus semua node Double Linked List");
    printf("\n 10: EXIT");
    printf("\n\n Enter your option : ");
    scanf("%d", &option);
    switch(option)
  {

case 1:
          system("cls");
          start = create_ll(start);
          printf("\n DOUBLE LINKED LIST CREATED");
          break;

          case 2:
          system("cls");
          start = display(start);
          break;

          case 3:
          system("cls");
          start = insert_beg(start);
          break;

          case 4:
          system("cls");
          start = insert_end(start);
          break;
```

```
            case 5:
            system("cls");
            start = insert_before(start);
            break;

            case 6:
            system("cls");
            start = insert_after(start);
            break;

            case 7:
            system("cls");
            start = delete_beg(start);
            break;

            case 8:
            system("cls");
            start = delete_end(start);
            break;

            case 9: start = delete_list(start);
            printf("\n DOUBLE LINKED LIST DELETED");
            break;

      }
}
      while(option !=10);
      getch();
      return 0;
}

struct node *create_ll(struct node *start)
{
      system("cls");
      struct node *new_node, *ptr;
      int num;
      printf("\n Tekan 0 untuk berhenti input data");
      printf("\n Masukan data : ");scanf("%d", &num);
      while(num!=0)
            {
             new_node = (struct node*)malloc(sizeof(struct node));
             new_node -> data=num;
             if(start==NULL)
                   {
                     new_node -> next = NULL;
                     new_node -> prev = NULL;
                     start = new_node;
                   }
                         else
                               {
                                     ptr=start;
                                     while(ptr->next!=NULL)
                                     ptr=ptr->next;
                                     ptr->next = new_node;
                                     new_node->next=NULL;
                                     new_node->prev=NULL;
                               }
                   printf(" Masukan data berikutnya : ");scanf("%d",
&num);
```

```
                }
    return start;
}

struct node *display(struct node *start)
{
    struct node *ptr;
    ptr = start;
    printf(" DATA DOUBLE LINKED LIST \n\n");
    while(ptr != NULL)
    {
          printf("\t %d", ptr -> data);
          ptr = ptr -> next;
    }
return start;
}


struct node *insert_beg(struct node *start)
{
    struct node *new_node;
    int num;
    printf(" TAMBAH DATA DI AWAL \n\n");
    printf("\n Masukan data : ");
    scanf("%d", &num);

    new_node = (struct node *)malloc(sizeof(struct node));
    new_node -> data = num;
    start->prev = new_node;
    new_node -> next = start;
    new_node-> prev = NULL;
    start = new_node;
    printf("\n DATA BERHASIL DITAMBAHKAN! \n");
    return start;
}

struct node *insert_end(struct node *start)
{
    struct node *ptr, *new_node;
    int num;
    printf(" TAMBAH DATA DI AKHIR \n\n");
    printf("\n Masukan data : ");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node -> data = num;
    ptr=start;
    while(ptr -> next != NULL)
    ptr = ptr -> next;
    ptr -> next = new_node;
    new_node -> prev = ptr;
    new_node -> next = NULL;


    printf("\n DATA BERHASIL DITAMBAHKAN! \n");
    return start;
}

struct node *insert_before(struct node *start)
{
    struct node *new_node, *ptr;
    int num, val;
    printf(" TAMBAH DATA SEBELUM NODE TERTENTU \n\n");
    printf("\n Masukan data : ");
    scanf("%d", &num);

    printf("\n Data baru ditambahkan sebelum node dengan data :
");scanf("%d", &val);

    new_node = (struct node *)malloc(sizeof(struct node));
    new_node -> data = num;
    ptr = start;
    while(ptr -> data != val){
    ptr = ptr -> next;
    new_node -> next = ptr;
    new_node -> prev = ptr-> prev;
    ptr -> prev -> next = new_node;
    ptr -> prev = new_node;
    }

return start;
}


struct node *insert_after(struct node *start)
{
    struct node *new_node, *ptr;
    int num, val;
    printf(" TAMBAH DATA SETELAH NODE TERTENTU \n\n");
    printf("\n Masukan data : ");
    scanf("%d", &num);
    printf("\n Data baru ditambahkan setelah node dengan data :
");scanf("%d", &val);

    new_node = (struct node *)malloc(sizeof(struct node));
    new_node -> data = num;

    ptr = start;
    while(ptr -> data != val){
          ptr = ptr -> next;
          new_node -> prev = ptr;
          new_node -> next = ptr -> next;
          ptr -> next -> prev = new_node;
          ptr -> next = new_node;
     }
    printf("\n DATA BERHASIL DITAMBAHKAN! \n");
    return start;
}

struct node *delete_beg(struct node *start)
{
    struct node *ptr;
    ptr = start;
    start = start -> next;
    start -> prev = NULL;
    free(ptr);
    printf("\n DATA AWAL BERHASIL DIHAPUS! \n");
    return start;
}
```

```
struct node *delete_end(struct node *start)
{
    struct node *ptr, *preptr;
    ptr = start;
    while(ptr -> next != NULL)
    {
          preptr = ptr;
          ptr = ptr -> next;
    }
    preptr -> next = NULL;
    free(ptr);

    printf("\n DATA AKHIR BERHASIL DIHAPUS! \n");
    return start;
}

struct node *delete_list(struct node *start)
{
    struct node *ptr;
    if(start!=NULL)
     {
     ptr=start;
     while(ptr != NULL)
          {
            printf("\n %d is to be deleted next", ptr -> data);
            start = delete_beg(ptr);
            ptr = start;
          }
    }
return start;
}
```

5. Circular Singly Linked List

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

struct node {
    int data;
    struct node *next;
};
struct node *head;
void tambahAwal ();
void tambahAkhir ();
void hapusAwal();
void hapusAkhir();
void tampilkan();
void cari();

int main() {
    int choice =0;
    while(choice != 7) {
        printf("\n*********Main Menu*********\n");
        printf("\nSilakan pilih salah satu dari menu berikut
...\n");

printf("\n===============================================\n");
```

```
        printf("\n1.Masukan data di awal\n2.Masukan data di
akhir\n3.Hapus dari awal\n4.Hapus dari akhir\n5.Cari sebuah
elemen\n6.Tampilkan\n7.Exit\n");
        printf("\nMasukan pilihanmu? \n");
        scanf("\n%d",&choice);
        switch(choice) {
            case 1:
            tambahAwal();
            break;
            case 2:
            tambahAkhir();
            break;
            case 3:
            hapusAwal();
            break;
            case 4:
            hapusAkhir();
            break;
            case 5:
            cari();
            break;
            case 6:
            tampilkan();
            break;
            case 7:
            exit(0);
            break;
            default:
            printf("Mohon masukan pilihan yang sesuai menu..");
        }
    }
}
void tambahAwal(){
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL){
        printf("\nOVERFLOW");
    }
    else {
        printf("\nMasukan 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 telah dimasukan\n");
    }
}
void tambahAkhir() {
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL) {
        printf("\nOVERFLOW\n");
    }
    else {
        printf("\nMasukan 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 hapusAwal() {
    struct node *ptr;
    if(head == NULL) {
        printf("\nUNDERFLOW");
    }
    else if(head->next == head) {
        head = NULL;
        free(head);
        printf("\nnode dihapus\n");
    }

    else {   ptr = head;
        while(ptr -> next != head)
            ptr = ptr -> next;
        ptr->next = head->next;
        free(head);
        head = ptr->next;
        printf("\nnode dihapus\n");
    }
}
void hapusAkhir() {
    struct node *ptr, *preptr;
    if(head==NULL) {
        printf("\nUNDERFLOW");
    }
    else if (head ->next == head) {
        head = NULL;
        free(head);
        printf("\nnode dihapus\n");

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

void cari() {
    struct node *ptr;
    int item,i=0,flag=1;
    ptr = head;
    if(ptr == NULL) {
        printf("\nList Kosong\n");
    }
    else {
        printf("\nMasukan item yang ingin dicari?\n");
        scanf("%d",&item);
        if(head ->data == item) {
           printf("item ditemukan pada lokasi %d",i+1);
           flag=0;
        }
        else {
        while (ptr->next != head) {
            if(ptr->data == item) {
                printf("item ditemukan pada lokasi %d ",i+1);
                flag=0;
                break;
            }
            else {
                flag=1;
            }
            i++;
            ptr = ptr -> next;
        }
        }
        if(flag != 0) {
            printf("Item tidak ditemukan\n");
        }
    }
}

void tampilkan() {
    struct node *ptr;
    ptr=head;
    if(head == NULL) {
        printf("\ntidak ada sesuatu untuk ditampilkan");
    }
    else {
        printf("\n Menampilkan nilai... \n");
        while(ptr -> next != head)  {
            printf("%d\n", ptr -> data);
            ptr = ptr -> next;
        }
        printf("%d\n", ptr -> data);
    }
}
```

6. Circular Doubly Linked List

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

struct node {
    struct node *prev;
    struct node *next;
    int data;
};
struct node *head;
void tambahAwal();
void tambahAkhir();
void hapusAwal();
void hapusAkhir();
void tampilkan();
void cari();

int main () {
int choice =0;
    while(choice != 9){
        printf("\n*********Main Menu*********\n");
        printf("\nSilakan pilih salah satu dari menu berikut
...\n");

printf("\n===============================================\n");
        printf("\n1.Masukan data di awal\n2.Masukan data di
akhir\n3.Hapus dari awal\n4.Hapus dari akhir\n5.Cari sebuah
elemen\n6.Tampilkan\n7.Exit\n");
        printf("\nMasukan pilihanmu? \n");
        scanf("\n%d",&choice);
        switch(choice){
            case 1:
                tambahAwal();
                break;
            case 2:
                tambahAkhir();
                break;
            case 3:
                hapusAwal();
                break;
            case 4:
                hapusAkhir();
                break;
            case 5:
                cari();
                break;
            case 6:
                tampilkan();
                break;
            case 7:
                exit(0);
                break;
            default:
                printf("Mohon masukan pilihan yang sesuai menu..");
        }
    }
}
void tambahAwal() {
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL) {
        printf("\nOVERFLOW");
    }
    else {
     printf("\nMasukan nilai item");
     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 tambahAkhir() {
    struct node *ptr,*temp;
    int item;
    ptr = (struct node *) malloc(sizeof(struct node));
    if(ptr == NULL) {
        printf("\nOVERFLOW");
    }
    else {
        printf("\nMasukan nilai");
        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 ditambahkan\n");
```

```
}

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

void tampilkan() {
    struct node *ptr;
    ptr=head;
    if(head == NULL){
        printf("\nTidak ada item yang bisa ditampilkan");
    }
    else {
        printf("\n Menampilkan nilai ... \n");
        while(ptr -> next != head) {
            printf("%d\n", ptr -> data);
            ptr = ptr -> next;
        }
        printf("%d\n", ptr -> data);
    }
```

```
}

void cari()
{
    struct node *ptr;
    int item,i=0,flag=1;
    ptr = head;
    if(ptr == NULL) {
        printf("\nList kosong\n");
    }
    else {
        printf("\nMasukan item yang ingin kamu cari\n");
        scanf("%d",&item);
        if(head ->data == item) {
           printf("item ditemukan pada lokasi %d",i+1);
           flag=0;
        }
        else {
        while (ptr->next != head) {
            if(ptr->data == item) {
                printf("item ditemukan pada lokasi %d ",i+1);
                flag=0;
                break;
            }
            else {
                flag=1;
            }
            i++;
            ptr = ptr -> next;
        }
        }
        if(flag != 0){
            printf("Item tidak ditemukan\n");
        }
    }
}
```

LATIHAN

1. Latihan 1

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
  int no_pol;
  struct ---- *next;
};
void display(struct node* head)
{
      struct node *temp = head;
      printf("\n\nurutan parkiran : - \n");
      while(temp != NULL)
      {
   printf("%d --->",temp->no_pol);
   temp = temp->next;
      }
}

void insertAtFront(struct node** headRef, int value) {
    struct node* head = *headRef;

    struct node *newNode;
    newNode = (node*)malloc(sizeof(struct node));
    newNode->no_pol = value;
    newNode->next = head;
    head = newNode;

    *headRef = head;
}
int main()
{
      struct node *head;
  struct ---- *avanza = NULL;
  struct ---- *hilux = NULL;
  struct ---- *kijangF1 = NULL;

  avanza = (node*)malloc(sizeof(struct node));
  hilux = (node*)malloc(sizeof(struct node));
  kijangF1 = (node*)malloc(sizeof(struct node));

  avanza-> ---- = 1234;
  hilux-> ---- = 2345;
  kijangF1-> ---- = 3456;
  /* Connect nodes */
  avanza->next = hilux;
  hilux->next = kijangF1;
  kijangF1->next = NULL;
  /* Save address of first node in head */
  head = avanza;
  insertAtFront(&head, 46543);
  display(head);
}
```

2. Latihan 2

```
------
------
------

struct gudang {
            char nama_barang[50];
    int harga;
    struct gudang *next;
    struct gudang *prev;
};
void cetak(struct gudang* head)
{
      struct gudang *temp = head;
      printf("\n\nList elements are - \n");
      while(temp != NULL)
      {
   printf("%s = %d --->",temp->nama_barang,temp->harga);
   temp = temp->next;
      }
}

int main()
{
	struct ---- *head;
	struct ---- *one = NULL;
	struct gudang *two = NULL;
	struct gudang *three = NULL;

	/* Allocate memory */
	one = ---- malloc(sizeof(struct gudang));
	---- = ---- malloc(sizeof(struct gudang));
	three = ---- malloc(sizeof(struct gudang));

	/* Assign data values */
	one->---- = 1000;
	strcpy(one->----, "obat nyamuk");
	two->---- = 5000;
	strcpy(two->----, "Gula 1 kilo");
	three->---- = 50000;
	strcpy(three->nama_barang, "Minyak Goreng");

	/* Connect nodes */
	one->next = two;
	one->prev = NULL;
	two->next = three;
	two->prev = one;
	three->next = NULL;
	three->prev = two;

	/* Save address of first node in head */
	head = one;
	cetak(----);

}
```

3. Latihan 3

```
#include <stdio.h>
#include <stdlib.h>
// Structure of the node
struct Node
{
    int data;
    struct Node *----; // Pointer to next node
    struct Node *----; // Pointer to previous node
};

void insertEnd(struct Node** start, int value)
{
    // If the list is empty, create a single node
    // circular and doubly list
    if (*start == NULL)
    {
        struct ----* new_node = new Node;
        new_node->data = value;
        new_node->next = new_node->prev = new_node;
        *start = new_node;
        return;
    }
  // If list is not empty
   /* Find last node */
    ---- *last = (*start)->prev;
      // Create Node dynamically
    struct Node* new_node = new Node;
    new_node->data = value;
  // Start is going to be next of new_node
    new_node->next = *start;
   // Make new node previous of start
    (*start)->prev = new_node;
   // Make last preivous of new node
    new_node->prev = last;
  // Make new node next of old last
    last->next = new_node;
}
void display(struct ----* start)
{
    struct Node *temp = start;
    printf("\nTraversal in forward direction \n");
    while (temp->next != start)
    {
        ----("%d ", temp->data);
        temp = temp->next;
    }
    printf("%d ", temp->data);
      printf("\nTraversal in reverse direction \n");
    Node *last = start->prev;
    temp = last;
    while (temp->prev != last)
    {
        ----("%d ", temp->data);
        temp = temp->prev;
    }
    printf("%d ", temp->data);
}
int main()
{
    /* Start with the empty list */
    struct Node* start = NULL;
    // Insert 5. So linked list becomes 5->NULL
    insertEnd(&start, 5);
    insertEnd(&start, 8);
    insertEnd(&start, 9);
    insertEnd(&start, 3);
    printf("Created circular doubly linked list is: ");
    display(----);
return 0;
}
```

TUGAS

1. Implementasikan Double Linked List, dimana data yang dipakai adalah data buku yang ada dalam sebuah perpustakaan (judul, nama pengarang, tahun). Program juga mengimplementasikan penambahan dan pengurangan simpul pada Linked List berdasarkan judul buku.

2. Implementasikan Double Linked List ini juga untuk menggambarkan antrian mobil yang ada pada sebuah perparkiran. Data yang digunakan adalah: no plat, merk mobil nama pemilik. Program juga mengimplementasikan penambahan dan pengurangan simpul pada Linked List berdasarkan no plat.

DAFTAR PUSTAKA

1. Kernighan, Brian W, & Ritchie, Dennis M. 1988. The Ansi C Programming Language Second Edition, Prentice-Hall.
2. Cipta Ramadhani. 2015. Dasar Algoritma & Struktur Data. Yogyakarta: ANDI.