LAPORAN PRAKTIKUM
Modul 05
"SINGLE LINKED LIST (BAGIAN KEDUA)"

Disusun Oleh:
Ganesha Rahman Gibran -2211104058
Kelas S1SE-07-02

Dosen :
Wahyu Andi Saputra, S.Pd., M.Eng.

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO

2024

Tujuan

1. Memahami penggunaan linked list dengan pointer operator- operator dalam program.
2. Memahami operasi-operasi dasar dalam linked list.
3. Membuat program dengan menggunakan linked list dengan prototype yang ada

Landasan Teori

Searching

Searching atau pencarian adalah salah satu operasi dasar dalam single linked list, di mana pencarian dilakukan dengan mengunjungi setiap node satu per satu hingga elemen yang dicari ditemukan. Operasi ini berjalan secara linear, dimulai dari node pertama (head) hingga elemen yang diinginkan ditemukan atau akhir list tercapai.

Fungsi Pencarian dalam Single Linked List

1. **Pencarian Elemen Berdasarkan Nilai (findElm)**

   Fungsi findElm(list L, infotype X) digunakan untuk mencari node dengan informasi yang sesuai dengan nilai X. Proses pencarian ini berjalan dari elemen pertama hingga akhir list. Jika elemen ditemukan, fungsi ini akan mengembalikan alamat (address) node tersebut; jika tidak, fungsi mengembalikan nilai Nil

2. **Pencarian Elemen Berdasarkan Alamat (fFindElm)**

   Fungsi fFindElm(list L, address P) digunakan untuk memeriksa apakah ada elemen dalam list yang beralamat sama dengan P. Jika ada, fungsi akan mengembalikan nilai true, jika tidak ada, mengembalikan false.

3. **Pencarian Elemen Sebelum Elemen Tertentu (findBefore)**

   Fungsi findBefore(list L, address P) digunakan untuk mencari elemen yang berada sebelum elemen dengan alamat P. Jika elemen P adalah elemen pertama, maka fungsi mengembalikan Nil.

Proses Pencarian

- Proses pencarian dalam single linked list dilakukan secara **sekuensial**. Dimulai dari elemen pertama, setiap node diperiksa satu per satu, dan traversal berlanjut hingga elemen yang dicari ditemukan atau list berakhir.
- **Efisiensi Pencarian:** Karena sifat dasar single linked list yang hanya memungkinkan traversal satu arah, pencarian dalam single linked list memiliki kompleksitas waktu **O(n)**, di mana n adalah jumlah elemen dalam list.

Operasi pencarian pada *Single Linked List* dapat dilakukan dengan menggunakan metode *sequential search*

```
/* contoh sintaks pencarian elemen */

#include <stdio.h>

#include <stdlib.h>

// Definisi struktur node

typedef int infotype;

typedef struct elmlist *address;

struct elmlist {

    infotype info;

    address next;

};

// Definisi list

struct list {

    address first;

};

// Fungsi untuk membuat list kosong

void CreateList(struct list *L) {

    L->first = NULL;

}

// Fungsi untuk mengalokasikan node baru

address alokasi(infotype X) {

    address P = (address) malloc(sizeof(struct elmlist));

    if (P != NULL) {

        P->info = X;

        P->next = NULL;

    }

    return P;

}
```

```
// Fungsi untuk menambahkan elemen di awal list

void insertFirst(struct list *L, address P) {

    P->next = L->first;

    L->first = P;

}

// Fungsi untuk mencari elemen dengan nilai X

address findElm(struct list L, infotype X) {

    address P = L.first;

    while (P != NULL) {

        if (P->info == X) {

            return P;  // Mengembalikan alamat elemen yang ditemukan

        }

        P = P->next;

    }

    return NULL;  // Mengembalikan NULL jika tidak ditemukan

}

// Fungsi untuk menampilkan semua elemen dalam list

void printInfo(struct list L) {

    address P = L.first;

    while (P != NULL) {

        printf("%d -> ", P->info);

        P = P->next;

    }

    printf("NULL\n");

}

int main() {

    struct list L;
```

```
    CreateList(&L);

    // Menambahkan beberapa elemen ke dalam list

    insertFirst(&L, alokasi(10));

    insertFirst(&L, alokasi(20));

    insertFirst(&L, alokasi(30));

    // Menampilkan list

    printf("Isi list: ");

    printInfo(L);

    // Mencari elemen

    int X = 20;

    address found = findElm(L, X);

    if (found != NULL) {

        printf("Elemen dengan nilai %d ditemukan.\n", X);

    } else {

        printf("Elemen dengan nilai %d tidak ditemukan.\n", X);

    }

    return 0;

}
```

Penjelasan Kode :

- CreateList: Membuat list kosong dengan first diinisialisasi menjadi NULL.
- alokasi: Mengalokasikan node baru yang berisi nilai X.
- insertFirst: Menambahkan elemen baru di awal list.
- findElm: Mencari elemen dalam list yang memiliki nilai X. Fungsi ini mengembalikan alamat node jika ditemukan, dan NULL jika tidak ada elemen dengan nilai yang dicari.
- printInfo: Menampilkan semua elemen dalam list

Guided

Input :

```
#include <iostream>
using namespace std;


struct Node{
    int data;
    Node* next;
};

void insertFirst(Node*& head, Node*& tail, int new_data){
    Node* new_node = new Node();
    new_node -> data = new_data;
    new_node -> next = head;
    head = new_node;

    if (tail == nullptr){
        tail = new_node;
    }
}
void insertLast(Node*& head, Node*& tail, int new_data){
    Node* new_node = new Node();
    new_node -> data = new_data;
    new_node -> next = nullptr;

    if (head == nullptr){
        head = new_node;
        tail = new_node;
    } else {
        tail ->next = new_node;
        tail = new_node;
    }
}
```

```
int findElement(Node*head, int x) {
    Node* current = head;
    int index = 0;

    while (current != nullptr){
        if (current->data==x){
            return index;
        }
        current = current->next;
        index++;
    }
    return -1;
}

void display(Node* node){
    while (node != nullptr) {
        cout << node -> data << " ";
        node = node->next;
    }
    cout << endl;
}
```

Telkom University Purwokerto

```
void deleteElement(Node*& head, int x){
    if (head == nullptr){
        cout << "Linked List Kosong" << endl;
        return;
    }

    if (head->data==x){
        Node* temp = head;
        head = head->next;
        delete temp;
        return;
    }

    Node* current = head;
    while (current ->next != nullptr) {
        if (current->next->data == x) {
            Node* temp = current ->next;
            current->next = current->next->next;
            delete temp;
            return;
        }
        current = current->next;
    }
}
```

```
int main(){
    Node* head = nullptr;
    Node* tail = nullptr;

    insertFirst(head,tail,3);
    insertFirst(head,tail,5);
    insertFirst(head,tail,7);

    insertLast(head,tail, 11);
    insertLast(head,tail, 14);
    insertLast(head,tail, 18);

    cout << "Element dalam Linked List :";
    display(head);

    int x;
    cout << "Masukkan Element yang ingin dicari : ";
    cin >> x;

    int result = findElement (head, x);

    if (result == -1)
        cout << "Element tidak ditemukan dalam Linked List" << endl;
    else
        cout << "Element ditemukan pada indeks" << result << endl;

    cout << "Masukkan Element yang ingin dihapus : ";
    cin >> x;
    deleteElement(head,x);

    cout << "Element dalam Linked List setelah penghapusan : ";
    display(head);
    return 0;
}
```

Output :

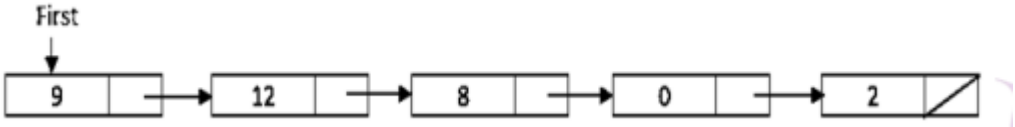
```
Element dalam Linked List :7 5 3 11 14 18
Masukkan Element yang ingin dicari : 3
Element ditemukan pada indeks2
Masukkan Element yang ingin dihapus : 11
Element dalam Linked List setelah penghapusan : 7 5 3 14 18
PS E:\Struktur Data\2211104058_Ganesha_Rahman_Gibran_SE-06-02>
```

Unguided

1. Buatlah ADT Single Linked list sebagai berikut di dalam file "singlelist.h":

```
Type infotype : int
Type address : pointer to ElmList
Type ElmList <
   info : infotype
   next : address >
Type List : < First : address >
prosedur CreateList( in/out L : List )
fungsi alokasi( x : infotype ) : address
prosedur dealokasi( in/out P : address )
prosedur printInfo( in L : List )
prosedur insertFirst( in/out L : List, in P : address )
```

Kemudian buat implementasi ADT Single Linked list pada file "singlelist.cpp". Adapun isi data



Cobalah hasil implementasi ADT pada file "main.cpp"

```
int main()
{
List L;
address P1, P2, P3, P4, P5 = NULL;
createList(L);
P1 = alokasi(2);
insertFirst(L,P1);

P2 = alokasi(0);
insertFirst(L,P2);

P3 = alokasi(8);
insertFirst(L,P3);

P4 = alokasi(12);
insertFirst(L,P4);

P5 = alokasi(9);
insertFirst(L,P5);
printInfo(L)
return 0; }
```

Input :

```
#include <iostream>
```

Telkom University Purwokerto

```
using namespace std;

struct List {
    int info;
    List* next;
};

typedef List* address;

// membuat list kosong
void CreateList(List* &L) {
    L = NULL;
}

// alokasi node baru
address alokasi(int x) {
    address P = new List;
    P->info = x;
    P->next = NULL;
    return P;
}

// dealokasi node
void dealokasi(address &P) {
    delete P;
}

// menambah node di awal list
void insertFirst(List* &L, address P) {
    if (L == NULL) {
        L = P;
    } else {
        P->next = L;
        L = P;
    }
}

// mencetak isi list
void printInfo(List* L) {
    address P = L;
    while (P != NULL) {
        cout << P->info;
        if (P->next != NULL) {
            cout << " -> ";
        }
        P = P->next;
    }
    cout << endl;
}

int main() {
    List* L;
    address P1, P2, P3, P4, P5 = NULL;

    CreateList(L);
```

```
    // membuat node lalu menyisipkan sesuai urutan
    P1 = alokasi(2);
    insertFirst(L,P1);

    P2 = alokasi(0);
    insertFirst(L,P2);

    P3 = alokasi(8);
    insertFirst(L,P3);

    P4 = alokasi(12);
    insertFirst(L,P4);

    P5 = alokasi(9);
    insertFirst(L,P5);

    // mencetak output
    cout << "Hasil akhir linked list: ";
    printInfo(L);

    return 0;
}
```

Output :

```
' '--interpreter=mi'
Hasil akhir linked list: 9 -> 12 -> 8 -> 0 -> 2
PS E:\Struktur Data\2211104058_Ganesha_Rahman_Gibran_SE-06-02>
```

2. Carilah elemen dengan info 8 dengan membuat fungsi baru. fungsi findElm(L : List, x : infotype) : address

```
8 ditemukan dalam list
Process returned 0 (0x0)   execution time : 0.020 s
Press any key to continue.
```

Input :

```
#include <iostream>
using namespace std;

struct List {
    int info;
    List* next;
};

typedef List* address;

// fungsi list
void CreateList(List* &L) {
    L = NULL;
}

address alokasi(int x) {
    address P = new List;
    P->info = x;
    P->next = NULL;
```

```
    return P;
}

void insertFirst(List* &L, address P) {
    if (L == NULL) {
        L = P;
    } else {
        P->next = L;
        L = P;
    }
}

// fungsi pencarian
address findElm(List* L, int x) {
    address P = L;
    while (P != NULL) {
        if (P->info == x) {
            return P;
        }
        P = P->next;
    }
    return NULL;
}

int main() {
    List* L;
    address P1, P2, P3, P4, P5 = NULL;

    CreateList(L);

    // membuat linked list
    P1 = alokasi(2);
    insertFirst(L, P1);
    P2 = alokasi(0);
    insertFirst(L, P2);
    P3 = alokasi(8);
    insertFirst(L, P3);
    P4 = alokasi(12);
    insertFirst(L, P4);
    P5 = alokasi(9);
    insertFirst(L, P5);

    // input
    int searchValue;
    cout << "Masukkan nilai yang ingin dicari: ";
    cin >> searchValue;

    // mencari elemen dengan nilai input
    cout << "Mencari elemen dengan info " << searchValue << ":" << endl;
    address found = findElm(L, searchValue);
    if (found != NULL) {
        cout << searchValue << " ditemukan dalam list" << endl;
    } else {
        cout << "Elemen dengan info " << searchValue << " tidak ditemukan"
    << endl;
```

```
    }

    return 0;
}
```

Output :

```
Masukkan nilai yang ingin dicari: 8
Mencari elemen dengan info 8:
8 ditemukan dalam list
PS E:\Struktur Data\2211104058_Ganesha_Rahman_Gibran_SE-06-02>
```

3. Hitunglah jumlah total info seluruh elemen (9+12+8+0+2=31).

```
Total info dari kelima elemen adalah 31
Process returned 0 (0x0)   execution time : 0.019 s
Press any key to continue.
```

Input :

```
#include <iostream>
using namespace std;

struct List {
    int info;
    List* next;
};

typedef List* address;

// fungsi buat list
void CreateList(List* &L) {
    L = NULL;
}

address alokasi(int x) {
    address P = new List;
    P->info = x;
    P->next = NULL;
    return P;
}

void insertFirst(List* &L, address P) {
    if (L == NULL) {
        L = P;
    } else {
        P->next = L;
        L = P;
    }
}

// fungsi hitung total
int sumInfo(List* L) {
    address P = L;
    int total = 0;
    while (P != NULL) {
        total += P->info;
        P = P->next;
    }
```

```
    return total;
}

int main() {
    List* L;
    address P1, P2, P3, P4, P5 = NULL;

    CreateList(L);

    //linked list
    P1 = alokasi(2);
    insertFirst(L, P1);
    P2 = alokasi(0);
    insertFirst(L, P2);
    P3 = alokasi(8);
    insertFirst(L, P3);
    P4 = alokasi(12);
    insertFirst(L, P4);
    P5 = alokasi(9);
    insertFirst(L, P5);

    // print
    cout << "Total info dari kelima elemen adalah " << sumInfo(L) << endl;

    return 0;
}
```

Output :

```
' '--interpreter=mi'
Total info dari kelima elemen adalah 31
PS E:\Struktur Data\2211104058_Ganesha_Rahman_Gibran_SE-06-02>
```

Kesimpulan

Searching pada single linked list merupakan proses dasar yang dilakukan secara sekuensial dengan menelusuri setiap node dari awal hingga elemen yang dicari ditemukan atau akhir list tercapai. Dengan kompleksitas waktu O(n), pencarian pada single linked list tidak efisien untuk jumlah elemen yang besar, namun tetap efektif untuk struktur data dinamis yang memungkinkan penambahan dan penghapusan elemen secara fleksibel. Implementasi fungsi pencarian seperti findElm memudahkan operasi lainnya, seperti penghapusan atau modifikasi elemen dalam list.