LAPORAN PRAKTIKUM
Modul 5
“Single Linked List (Bagian Kedua)”

Disusun Oleh:
Dimastian Aji Wibowo (2311104058)
SE-07-02

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO

2024

1. **Tujuan**
   - Memahami penggunaan *linked list* dengan *pointer* operator – operator dalam program.
   - Memahami operasi – operasi dasar dalam linked list.
   - Membuat program dengan menggunakan linked list dengan *prototype* yang ada.
2. **Landasan Teori**

   Searching

   Searching merupakan operasi dasar *list* dengan melakukan aktivitas pencarian terhadap node tertentu. Proses ini berjalan dengan mengunjungi setiap *node* dan berhenti setelah *node* yang dicari ketemu. Dengan melakukan operasi searching, operasi – operasi seperti *insert after*, *delete after*, dan *update* akan lebih mudah.

   Semua fungsi dasar diatas merupakan bagian dari ADT dari singgle linked list, dan aplikasi pada bahasa pemrograman Cp semua ADT tersebut tersimpan dalam *file* *.c dan *file* *.h.

```c
/*file : list .h*/
/* contoh ADT list berkait dengan representasi fisik pointer*/
/* representasi address dengan pointer*/
/* info tipe adalah integer */
#ifndef list_H
#define list_H
#include "boolean.h"
#include <stdio.h>
#define Nil NULL
#define info(P) (P)->info
#define next(P) (P)->next
#define first(L) ((L).first)
/*deklarasi record dan struktur data list*/
typedef int infotype;
typedef struct elmlist *address;
struct elmlist{
infotype info;
address next;
};
/* definisi list : */
/* list kosong jika First(L)=Nil */
/* setiap elemen address P dapat diacu info(P) atau next(P) */
```

Telkom University Purwokerto

```
struct list {
address first;
};
/********* pengecekan apakah list kosong **********/
boolean ListEmpty(list L);
/*mengembalikan nilai true jika list kosong*/
/********* pembuatan list kosong *********/
void CreateList(list &L);
/* I.S. sembarang
 F.S. terbentuk list kosong*/

/********* manajemen memori *********/
void dealokasi(address P);
/* I.S. P terdefinisi
 F.S. memori yang digunakan P dikembalikan ke sistem */

/********* pencarian sebuah elemen list *********/
address findElm(list L, infotype X);
/* mencari apakah ada elemen list dengan info(P) = X
 jika ada, mengembalikan address elemen tab tsb, dan Nil jika
sebaliknya
*/
boolean fFindElm(list L, address P);
/* mencari apakah ada elemen list dengan alamat P
 mengembalikan true jika ada dan false jika tidak ada */
address findBefore(list L, address P);
/* mengembalikan address elemen sebelum P
 jika prec berada pada awal list, maka mengembalikan nilai Nil
*/

/********* penambahan elemen **********/
void insertFirst(list &L, address P);
/* I.S. sembarang, P sudah dialokasikan
 F.S. menempatkan elemen beralamat P pada awal list */

void insertAfter(list &L, address P, address Prec);
/* I.S. sembarang, P dan Prec alamt salah satu elemen list
 F.S. menempatkan elemen beralamat P sesudah elemen beralamat
Prec */
void insertLast(list &L, address P);
/* I.S. sembarang, P sudah dialokasikan
 F.S. menempatkan elemen beralamat P pada akhir list */

/********* penghapusan sebuah elemen *********/
void delFirst(list &L, adress &P);
/* I.S. list tidak kosong
 F.S. adalah alamat dari alamat elemen pertama list
 sebelum elemen pertama list dihapus
```

```
 elemen pertama list hilang dan list mungkin menjadi kosong
 first elemen yang baru adalah successor first elemen yang
lama */
void delLast(list &L, adress &P);
/* I.S. list tidak kosong
 F.S. adalah alamat dari alamat elemen terakhir list
 sebelum elemen terakhir list dihapus
 elemen terakhir list hilang dan list mungkin menjadi kosong
 last elemen yang baru adalah successor last elemen yang lama
*/
void delAfter(list &L, address &P, address Prec);
/* I.S. list tidak kosng, Prec alamat salah satu elemen list
 F.S. P adalah alamatdari next(Prec), menghapus next(Prec)
dari list */
void delP (list &L, infotype X);
/* I.S. sembarang
 F.S. jika ada elemen list dengan alamat P, dimana info(P)=X,
maka P
 dihapus
 dan P di-dealokasi, jika tidak ada maka list tetap
 list mungkin akan menjadi kosong karena penghapusan */

/********** proses semau elemen list *********/
void printInfo(list L);
/* I.S. list mungkin kosong
 F.S. jika list tidak kosong menampilkan semua info yang ada
pada list */
int nbList(list L);
/* mengembalikan jumlah elemen pada list */
/********** proses terhadap list **********/
void delAll(list &L);
/* menghapus semua elemen list dan semua elemen di-dealokasi
*/
void invertList(list &L);
/* I.S. sembarang
 F.S. elemen - elemen list dibalik */
void copyList(list L1, list &L2)
/* I.S. L1 sembarang
 F.S. L1 = L2, L1 dan L2 menunjuk pada elemen yang sama */
list fCopyList(list L);
/* mengembalikan list yang merupakan salinan dari L */
#endif
```

3. Guided

1. Membuat struktur node dengan dua atribut yaitu data untuk menyimpan nilai integer yang akan disimpan di dalam node dan next

untuk menyimpan pointer yang menunjuk ke node berikutnya dalam linked list.

2. Membuat fungsi InsertFirst untuk menambahkan elemen baru (new_data) di awal linked list dengan membuat new_node yang akan menyimpan new_data, menyambungkan new_node ke node pertama (head) yang ada saat ini, menetapkan head ke new_node, dan jika tail bernilai nullptr, artinya linked list kosong. Jadi, tail diatur ke new_node.
3. Membuat fungsi InsertLast untuk menambahkan elemen baru (new_data) di akhir linked list dengan membuat new_node yang akan menyimpan new_data dan mengarahkan new_node->next ke nullptr, jika linked list kosong (head == nullptr), menetapkan head dan tail ke new_node, dan jika linked list tidak kosong, sambungkan tail->next ke new_node dan pindahkan tail ke new_node.

```
#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;
    }
}
```

4. Membuat fungsi FindElement untuk mencari elemen dengan nilai x dalam linked list dan mengembalikan indeksnya dengan menggunakan perulangan untuk mengecek pada setiap node dari head.
5. Membuat fungsi Display untuk menampilkan semua elemen dalam linked list yang dimulai dari head.

```
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;
}
```

6. Membuat fungsi DeleteElement untuk menghapus elemen dengan nilai x dari linked list dengan jika head kosong maka cetak pesan bahwa list kosong dan keluar dari fungsi, jika elemen head memiliki nilai x maka pindahkan head ke elemen berikutnya dan hapus elemen pertama, jika elemen yang ingin dihapus berada di tengah atau akhir maka cari node yang nilai next-nya sama dengan x, dan setelah ditemukan maka pindahkan current->next ke elemen setelah node yang ingin dihapus, lalu hapus node tersebut.

```
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;
    }
}
```

7. Membuat fungsi main() dengan memanggil fungsi – fungsi untuk menguji fungsi yang telah dibuat.

```cpp
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<<"Elemen dalam linked list: ";
    display(head);

    int x;
    cout<<"Masukan elemen yang ingin dicari: ";
    cin>>x;

    int result= findElement(head, x);
    if(result==-1)
        cout<<"elemen tidak ditemukan dalam linked list"<<endl;
    else
        cout<<"elesmen ditemukan dalam indeks "<<result<<endl;

    cout<<"Masukan elemen yang ingin dihapus: ";
    cin>>x;
    deleteElement(head, x);
    cout<<"elemen dalam linked list setelah penghapusan: ";
    display(head);
    return 0;
}
```

8. Dengan output berikut.

```
Elemen dalam linked list: 7 5 3 11 14 18
Masukan elemen yang ingin dicari: 14
elemen ditemukan dalam indeks 4
Masukan elemen yang ingin dihapus: 14
elemen dalam linked list setelah penghapusan: 7 5 3 11 18
```

4. Unguided

A. Unguided 1

1. Membuat file header diikuti membuat *header guard* yang digunakan untuk mencegah multiple inclusions dari file header yang sama dan membuat pustaka iostream.
2. Mendefinisikan infotype sebagai alias untuk int dan Mendefinisikan address sebagai alias untuk pointer ke tipe data ElmList, yaitu struktur elemen yang berisi data dan pointer ke elemen berikutnya dalam list.
3. Membuat struktur yang merepresentasikan sebuah node dalam linked

list dengan infotype info untuk menyimpan informasi utama dari node sebagai int dan address next; untuk pointer yang menunjuk ke elemen berikutnya dalam list, membentuk tautan antar node.

4. Membuat struktur yang merepresentasikan linked list secara keseluruhan dengan address First; yang menunjuk ke node pertama (head) dalam linked list. Jika First bernilai nullptr, itu berarti list kosong.
5. Mendeklarasikan prosedur untuk membuat linked list kosong, fungsi untuk mengalokasikan elemen baru dalam memori dengan nilai x, prosedur untuk me-*dealokasi* atau membebaskan memori yang digunakan oleh node tertentu, sehingga memori dapat digunakan untuk keperluan lain, prosedur untuk mencetak semua elemen dalam linked list dari awal hingga akhir, dan prosedur untuk menambahkan node di awal list.

```
#ifndef SINGLELIST_H
#define SINGLELIST_H
#include <iostream>

typedef int infotype;
typedef struct ElmList *address;

struct ElmList {
    infotype info;
    address next;
};
struct List {
    address First;
};
void createList(List &L);

address alokasi(infotype x);

void dealokasi(address &P);

void printInfo(const List &L);

void insertFirst(List &L, address P);

#endif
```

6. Membuat file singlelist.cpp diikuti mengikutkan file header dan daftar pustaka.
7. Membuat prosedur ini menginisialisasi linked list kosong dengan mengatur pointer First dari List ke nullptr, menandakan bahwa list tidak memiliki elemen saat pertama kali dibuat.
8. Membuat fungsi untuk ini mengalokasikan node baru di memori dengan tipe ElmList menggunakan new untuk membuat node baru, mengisi info dengan nilai x, dan mengatur next ke nullptr lalu mengembalikan pointer P ke node baru tersebut agar bisa digunakan untuk disisipkan ke dalam list.
9. Menggunakan prosedur untuk membebaskan memori dari node P yang tidak lagi dibutuhkan menggunakan delete.
10. Memanggil fungsi untuk menampilkan semua elemen dalam linked

list.

11. Membuat prosedur untuk menambahkan node baru P di awal list dengan mengatur P->next untuk menunjuk ke elemen pertama dari list yang sebelumnya dan menjadikan P sebagai elemen First baru dari list, sehingga node P sekarang menjadi elemen pertama yang akan dicetak oleh printInfo.

```
#include "singlelist.h"
#include <iostream>
using namespace std;

void createList(List &L) {
    L.First = nullptr;
}
address alokasi(infotype x) {
    address P = new ElmList;
    P->info = x;
    P->next = nullptr;
    return P;
}
void dealokasi(address &P) {
    delete P;
    P = nullptr;
}
void printInfo(const List &L) {
    address P = L.First;
    while (P != nullptr) {
        cout << P->info << " ";
        P = P->next;
    }
    cout << endl;
}
void insertFirst(List &L, address P) {
    P->next = L.First;
    L.First = P;
}
```

12. Membuat file main.cpp dengan mengikutkan file header dan menuliskan daftar pustaka.
13. Membuat main() diikuti mendeklarasikan variabel L bertipe List, yang akan digunakan sebagai linked list dan mendeklarasikan beberapa pointer (P1, P2, P3, P4, P5) bertipe address yang menunjuk ke elemen-elemen linked list yang diinisialisasi dengan nullptr.

14. Memanggil prosedur createList untuk menginisialisasi linked list L sebagai list kosong, di mana L.First diatur ke nullptr.
15. Memanggil alokasi(x) yang digunakan untuk membuat node baru dengan nilai x dan mengembalikan pointer yang menunjuk ke node tersebut dan memanggil insertFirst(L, P) untuk memasukkan node baru P ke awal linked list L yang dilakukan berulang – ulang untuk setiap elemen.
16. Memanggil prosedur printInfo untuk mencetak seluruh elemen di linked list L dari awal hingga akhir.
17. Menuliskan return 0; untuk mengakhiri fungsi main().

```
#include "singlelist.h"
#include <iostream>
using namespace std;

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

    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;
}
```

18. Berikut merupakan output dari program tersebut.

```
9 12 8 0 2
```

B. Unguided 2

1. Mendeklarasikan fungsi findElm pada file header.

```
address findElm(const List &L, infotype x);
```

2. Membuat fungsi findElm yang mencari elemen dengan nilai x dalam linked list L pada file singlelist.cpp dengan parameter const List &L yang merupakan referensi konstan ke linked list L dan infotype x adalah nilainya.
3. address P = L.First; P adalah pointer yang akan digunakan untuk traversal (penelusuran) dalam linked list, mulai dari elemen pertama L.First.
4. Menggunakan perulangan while yang berjalan selama P tidak nullptr, yang berarti kita belum mencapai akhir dari linked list dan di setiap iterasi loop, P akan menunjuk ke elemen berikutnya dalam list hingga mencapai elemen terakhir.

```
address findElm(const List &L, infotype x) {
    address P = L.First;
    while (P != nullptr) {
        if (P->info == x) {
            return P;
        }
        P = P->next;
    }
    return nullptr;
```

5. Memanggil fungsi findElm pada file main.cpp untuk mencari elemen dengan nilai 8 dan menggunakan if else untuk mengetahui hasil.

```
address found = findElm(L, 8);
if (found != nullptr) {
    cout << found->info << " ditemukan dalam list" << endl;
} else {
    cout << "Elemen tidak ditemukan dalam list" << endl;
}
return 0;
```

6. Berikut merupakan output dari program tersebut

```
9 12 8 0 2
8 ditemukan dalam list
```

C. Unguided 3

1. Mendeklarasikan fungsi sumInfo di file header.

```
int sumInfo(const List &L);
```

2. Membuat fungsi sumInfo yang menghitung jumlah info dari semua elemen dalam linked list pada file singlelist.cpp dengan mendeklarasikan variabel P berupa address dan total berupa int.
3. Menggunakan perulangan while yang berjalan selama P tidak nullptr, yang berarti belum mencapai akhir list, lalu menambahkan nilai info dari elemen yang sedang ditunjuk oleh P ke total, mengarahkan P ke elemen berikutnya dalam list untuk melanjutkan penjumlahan, dan setelah semua elemen dijumlahkan fungsi mengembalikan nilai total.

```
int sumInfo(const List &L) {
    address P = L.First;
    int total = 0;
    while (P != nullptr) {
        total += P->info;
        P = P->next;
    }
    return total;
```

4. Memanggil fungsi sumInfo di main.cpp untuk menghitung dan mencetak jumlah info dari semua elemen.

```
int total = sumInfo(L);
cout << "Total info dari kelima elemen adalah " << total << endl;
```

5. Berikut merupakan output dari program tersebut.

```
Total info dari kelima elemen adalah 31
```

5. Kesimpulan

Telah dipelajari implementasi dasar dari struktur data *Single Linked List menggunakan C++*. Contoh dari operasi – operasi utama pada linked list, yaitu *insert first*, yang menambahkan elemen di awal list, memungkinkan penambahan data baru dengan cepat di posisi pertama; *insert last*, yang menambahkan elemen di akhir list, memungkinkan penambahan data secara berurutan di posisi terakhir; serta *delete*, yang menghapus elemen tertentu dari list berdasarkan nilainya, memerlukan penelusuran linked list untuk menemukan dan menghapus elemen yang diinginkan. Selain itu, terdapat juga *searching* untuk mencari elemen dalam linked list dan mengembalikan posisinya jika ditemukan, atau mengindikasikan bahwa elemen tersebut tidak ada dalam list. Melalui operasi – operasi ini, telah dipahami bagaimana *Single Linked List* dapat digunakan untuk mengelola data dinamis secara efisien, terutama ketika penambahan dan penghapusan data sering terjadi di posisi awal dan akhir list.