LAPORAN PRAKTIKUM
Modul 5
SINGLE LINKED LIST BAGIAN 2

Disusun Oleh:
Zhafir Zaidan Avail
S1-SE-07-2

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

1. 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

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.

```
/*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) */
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

Code:

```
#include <iostream>

using namespace std;

// Struktur untuk node dalam linked list

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

// Fungsi untuk menambahkan elemen baru ke awal linked list 11
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;
}

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);

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

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

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

    int result = findElement(head, x);

    if (result == -1)
        cout << "Elemen tidak ditemukan dalam linked list" << endl;
    else
        cout << "Elemen ditemukan pada indeks " << result << endl;

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

    cout << "Elemen dalam linked list setelah penghapusan: ";
    display(head);

    return 0;
}
```

Output:

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

Process returned 0 (0x0)   execution time : 9.061 s
Press any key to continue.
```

4. Unguided

1. Unguided 1

File singlelist.h

```
#ifndef SINGLELIST_H
#define SINGLELIST_H

typedef int infotype;
typedef struct ElmtList* address;

struct ElmtList {
    infotype info;
    address next;
};

struct List {
    address first;
};

// Function prototypes
void createList(List& L);
address alokasi(infotype x);
void dealokasi(address& P);
void insertFirst(List& L, address P);
void printInfo(const List& L);

#endif
```

File singlelist.cpp

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

// Initialize an empty list
void createList(List& L) {
    L.first = nullptr;
}

// Allocate a new node with the given value
address alokasi(infotype x) {
    address P = new ElmtList;
    if (P != nullptr) {
        P->info = x;
        P->next = nullptr;
    }
    return P;
}

// Deallocate memory of a node
void dealokasi(address& P) {
    delete P;
    P = nullptr;
}

// Insert a new node at the beginning of the list
void insertFirst(List& L, address P) {
```

```
    if (P != nullptr) {
        P->next = L.first;
        L.first = P;
    }
}

// Print all elements in the list
void printInfo(const List& L) {
    address P = L.first;
    while (P != nullptr) {
        cout << P->info << " ";
        P = P->next;
    }
    cout << endl;
}
```

File main.cpp

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

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

    createList(L);

    // Allocate and insert elements into the 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 the list
    printInfo(L);

    return 0;
}
```

Output:

```
9 12 8 0 2

Process returned 0 (0x0)   execution time : 0.067 s
Press any key to continue.
```

2. Unguided 2

File singlelist.h

```
#ifndef SINGLELIST_H
#define SINGLELIST_H

typedef int infotype;
typedef struct ElmtList* address;
```

```
struct ElmtList {
    infotype info;
    address next;
};

struct List {
    address first;
};

// Function prototypes
void createList(List& L);
address alokasi(infotype x);
void dealokasi(address& P);
void insertFirst(List& L, address P);
void printInfo(const List& L);
address findElm(const List& L, infotype x); // Fungsi baru untuk
pencarian elemen

#endif
```

File singlelist.cpp

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

// Initialize an empty list
void createList(List& L) {
    L.first = nullptr;
}

// Allocate a new node with the given value
address alokasi(infotype x) {
    address P = new ElmtList;
    if (P != nullptr) {
        P->info = x;
        P->next = nullptr;
    }
    return P;
}

// Deallocate memory of a node
void dealokasi(address& P) {
    delete P;
    P = nullptr;
}

// Insert a new node at the beginning of the list
void insertFirst(List& L, address P) {
    if (P != nullptr) {
        P->next = L.first;
        L.first = P;
    }
}

// Print all elements in the list
void printInfo(const List& L) {
    address P = L.first;
    while (P != nullptr) {
        cout << P->info << " ";
        P = P->next;
    }
    cout << endl;
}

// Find an element with the given value
address findElm(const List& L, infotype x) {
```

```
    address P = L.first;
    while (P != nullptr) {
        if (P->info == x) {
            return P; // Elemen ditemukan
        }
        P = P->next;
    }
    return nullptr; // Elemen tidak ditemukan
}
```

File main.cpp

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

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

    createList(L);

    // Allocate and insert elements into the 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 the list
    printInfo(L);

    // Find element with info 8
    address found = findElm(L, 8);
    if (found != nullptr) {
        cout << "Elemen dengan info 8 ditemukan: " << found->info <<
endl;
    } else {
        cout << "Elemen dengan info 8 tidak ditemukan." << endl;
    }

    return 0;
}
```

Output:

```
9 12 8 0 2
Elemen dengan info 8 ditemukan: 8

Process returned 0 (0x0)   execution time : 0.070 s
Press any key to continue.
```

3. Unguided 3

File singlelist.h

```
#ifndef SINGLELIST_H
#define SINGLELIST_H
```

```
typedef int infotype;
typedef struct ElmtList* address;

struct ElmtList {
    infotype info;
    address next;
};

struct List {
    address first;
};

// Function prototypes
void createList(List& L);
address alokasi(infotype x);
void dealokasi(address& P);
void insertFirst(List& L, address P);
void printInfo(const List& L);
address findElm(const List& L, infotype x);
int sumInfo(const List& L);  // Fungsi baru untuk menghitung jumlah
total info

#endif
```

File singlelist.cpp

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

// Initialize an empty list
void createList(List& L) {
    L.first = nullptr;
}

// Allocate a new node with the given value
address alokasi(infotype x) {
    address P = new ElmtList;
    if (P != nullptr) {
        P->info = x;
        P->next = nullptr;
    }
    return P;
}

// Deallocate memory of a node
void dealokasi(address& P) {
    delete P;
    P = nullptr;
}

// Insert a new node at the beginning of the list
void insertFirst(List& L, address P) {
    if (P != nullptr) {
        P->next = L.first;
        L.first = P;
    }
}

// Print all elements in the list
void printInfo(const List& L) {
    address P = L.first;
    while (P != nullptr) {
        cout << P->info << " ";
        P = P->next;
    }
```

```
    cout << endl;
}

// Find an element with the given value
address findElm(const List& L, infotype x) {
    address P = L.first;
    while (P != nullptr) {
        if (P->info == x) {
            return P;  // Elemen ditemukan
        }
        P = P->next;
    }
    return nullptr;  // Elemen tidak ditemukan
}

// Calculate the sum of all elements' info
int sumInfo(const List& L) {
    int sum = 0;
    address P = L.first;
    while (P != nullptr) {
        sum += P->info;
        P = P->next;
    }
    return sum;
}
```

File main.cpp

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

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

    createList(L);

    // Allocate and insert elements into the 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 the list
    printInfo(L);

    // Find element with info 8
    address found = findElm(L, 8);
    if (found != nullptr) {
        cout << "Elemen dengan info 8 ditemukan: " << found->info <<
endl;
    } else {
        cout << "Elemen dengan info 8 tidak ditemukan." << endl;
    }

    // Calculate and print the sum of all elements' info
```

```
    int totalSum = sumInfo(L);
    cout << "Jumlah total info seluruh elemen: " << totalSum << endl;

    return 0;
}
```

Output:

```
9 12 8 0 2
Elemen dengan info 8 ditemukan: 8
Jumlah total info seluruh elemen: 31

Process returned 0 (0x0)   execution time : 0.063 s
Press any key to continue.
```

5. Kesimpulan

Pada kode di atas, implementasi ADT (Abstract Data Type) single linked list mencakup berbagai fungsi dasar yang penting untuk manajemen dan manipulasi data dalam list berkait. Single linked list disusun dengan node yang memiliki dua bagian: nilai atau info dan pointer next yang menunjuk ke elemen berikutnya. Dengan struktur ini, list dapat dengan mudah dimodifikasi dengan menambahkan, menghapus, atau mengupdate elemen. ADT ini dipisahkan menjadi dua file, yaitu list.h untuk deklarasi dan header, serta list.c untuk implementasi fungsi, yang memberikan modularitas dan kemudahan dalam pengembangan program.

Proses searching atau pencarian merupakan salah satu operasi utama pada linked list. Fungsi seperti findElm dan fFindElm mempermudah pencarian nilai tertentu atau node berdasarkan alamatnya. Searching ini sangat penting karena digunakan sebagai dasar untuk operasi lainnya, seperti insertAfter, deleteAfter, dan update. Dengan melakukan pencarian terlebih dahulu, kita dapat menentukan posisi node target atau elemen sebelum target, sehingga proses modifikasi pada linked list menjadi lebih efisien dan aman. Jika elemen yang dicari tidak ditemukan, fungsi akan mengembalikan Nil atau false, menandakan bahwa elemen tersebut tidak ada dalam list.

Selain fungsi pencarian, ADT ini juga mencakup berbagai operasi penting seperti penambahan elemen (misalnya insertFirst, insertAfter, dan insertLast), penghapusan elemen (delFirst, delLast, dan delAfter), serta proses seluruh elemen list (printInfo dan nbList). Dengan adanya fungsi seperti delAll dan invertList, pengguna juga dapat menghapus semua elemen sekaligus atau membalik urutan list dengan mudah. Fleksibilitas dan kemudahan pemakaian ADT linked list ini membuatnya cocok untuk berbagai aplikasi di bidang informatika dan ilmu komputer, seperti manajemen data dinamis dan implementasi struktur data lainnya.