LAPORAN PRAKTIKUM

Modul 5

SINGLE LINKED LIST (BAGIAN KEDUA)

Disusun Oleh:

RifqiMRamdani 2311104044

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

Linked list adalah struktur data yang terdiri dari node-node yang saling terhubung dengan pointer. Setiap node dalam linked list terdiri dari dua bagian: data dan pointer yang menunjuk ke node berikutnya. Berbeda dengan array, linked list dapat berubah ukurannya secara dinamis, yang membuatnya lebih efisien dalam hal memori ketika data terus berubah. Ada beberapa operasi dasar yang dapat dilakukan pada linked list seperti insert, delete, dan search.

Searching adalah proses pencarian node dengan nilai tertentu dalam linked list. Proses ini dilakukan dengan mengunjungi setiap node satu per satu hingga ditemukan nilai yang sesuai.

Operasi-operasi ini diimplementasikan dalam bentuk Abstract Data Type (ADT), dan digunakan dalam berbagai bahasa pemrograman termasuk C/C++.

3. Guided

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.

```
1   /*file : list .h*/
2   /* contoh ADT list berkait dengan representasi fisik pointer*/
3   /* representasi address    dengan pointer*/
4   /* info tipe adalah integer */
5   #ifndef list_H
6   #define list_H
7   #include "boolean.h"
8   #include <stdio.h>
9   #define Nil NULL
10  #define info(P) (P)->info
11  #define next(P) (P)->next
12  #define first(L) ((L).first)
13
14  /*deklarasi record dan struktur data list*/
15  typedef int infotype;
16  typedef struct elmlist *address;
17  struct elmlist{
18      infotype info;
19      address next;
20  };
21
22  /* definisi list : */
23  /* list kosong jika First(L)=Nil */
24  /* setiap elemen address P dapat diacu info(P) atau next(P) */
25  struct list {
26      address first;
27  };
28  /********* pengecekan apakah list kosong ***********/
29  boolean ListEmpty(list L);
30  /*mengembalikan nilai true jika list kosong*/
31
32  /********* pembuatan list kosong *********/
33  void CreateList(list &L);
34  /* I.S. sembarang
35     F.S. terbentuk list kosong*/
36
37  /********* manajemen memori *********/
38  void dealokasi(address P);
39  /* I.S. P terdefinisi
40     F.S. memori yang digunakan P dikembalikan ke sistem */
41
42
43  /********* pencarian sebuah elemen list *********/
44  address findElm(list L, infotype X);
45  /* mencari apakah ada elemen list dengan info(P) = X
```

```
46      jika ada, mengembalikan address elemen tab tsb, dan Nil jika sebaliknya
47  */
48
49  boolean fFindElm(list L, address P);
50  /* mencari apakah ada elemen list dengan alamat P
51     mengembalikan true jika ada dan false jika tidak ada */
52
53  address findBefore(list L, address P);
54  /* mengembalikan address elemen sebelum P
55     jika prec berada pada awal list, maka mengembalikan nilai Nil */
56
57  /********* penambahan elemen **********/
58  void insertFirst(list &L, address P);
59  /* I.S. sembarang, P sudah dialokasikan
60     F.S. menempatkan elemen beralamat P pada awal list */
61
62  void insertAfter(list &L, address P, address Prec);
63  /* I.S. sembarang, P dan Prec alamt salah satu elemen list
64     F.S. menempatkan elemen beralamat P sesudah elemen beralamat Prec */
65
66  void insertLast(list &L, address P);
67  /* I.S. sembarang, P sudah dialokasikan
68     F.S. menempatkan elemen beralamat P pada akhir list */
69
70  /********* penghapusan sebuah elemen *********/
71  void delFirst(list &L, adress &P);
72  /* I.S. list tidak kosong
73     F.S. adalah alamat dari alamat elemen pertama list
74     sebelum elemen pertama list dihapus
75     elemen pertama list hilang dan list mungkin menjadi kosong
76     first elemen yang baru adalah successor first elemen yang lama */
77
78  void delLast(list &L, adress &P);
79  /* I.S. list tidak kosong
80     F.S. adalah alamat dari alamat elemen terakhir list
81     sebelum elemen terakhir list dihapus
82     elemen terakhir list hilang dan list mungkin menjadi kosong
83     last elemen yang baru adalah successor last elemen yang lama */
```

```
84
85  void delAfter(list &L, address &P, address Prec);
86  /* I.S. list tidak kosng, Prec alamat salah satu elemen list
87     F.S. P adalah alamatdari next(Prec), menghapus next(Prec) dari list */
88
89  void delP (list &L, infotype X);
90  /* I.S. sembarang
91     F.S. jika ada elemen list dengan alamat P, dimana info(P)=X, maka P
92     dihapus
93     dan P di-dealokasi, jika tidak ada maka list tetap
94     list mungkin akan menjadi kosong karena penghapusan */
95
96  /********** proses semau elemen list *********/
97  void printInfo(list L);
98  /* I.S. list mungkin kosong
99     F.S. jika list tidak kosong menampilkan semua info yang ada pada list */
100
101 int nbList(list L);
102 /* mengembalikan jumlah elemen pada list */
103
104 /********** proses terhadap list **********/
105 void delAll(list &L);
106 /* menghapus semua elemen list dan semua elemen di-dealokasi */
107
108 void invertList(list &L);
109 /* I.S. sembarang
110    F.S. elemen - elemen list dibalik */
111 void copyList(list L1, list &L2)
112 /* I.S. L1 sembarang
```

```
113     F.S. L1 = L2, L1 dan L2 menunjuk pada elemen yang sama */
114
115 list fCopyList(list L);
116 /* mengembalikan list yang merupakan salinan dari L */
117 #endif
```

Latihan Praktikum Di Kelas

Kode Program:

main.cpp | main.cpp | *main.cpp

```cpp
#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;
```

main.cpp | main.cpp | *main.cpp

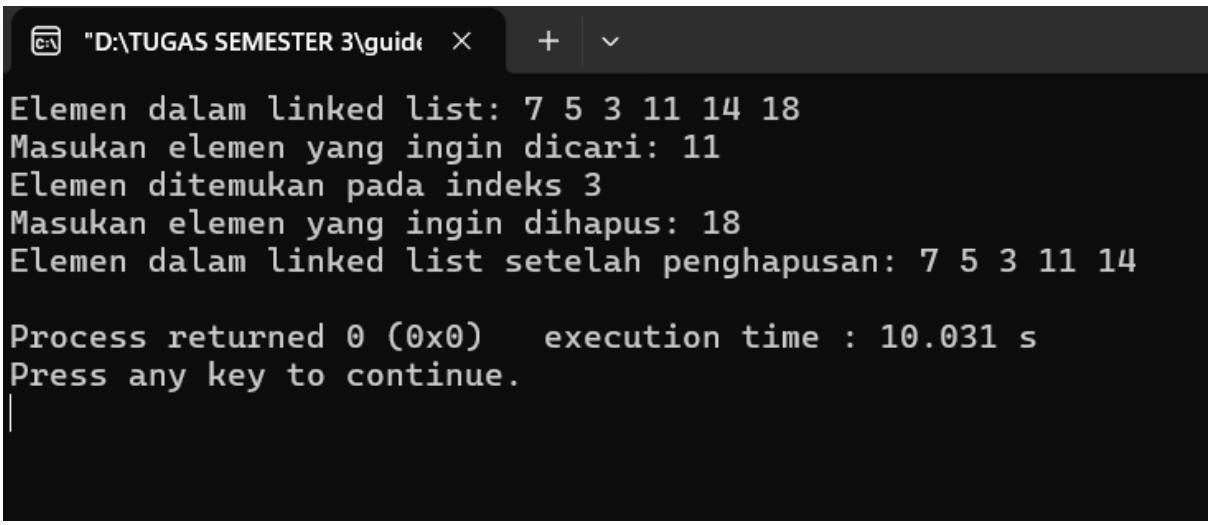
```cpp
    Node* current = head;
    int index = 0;

    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) {
```

main.cpp | main.cpp | *main.cpp

```cpp
            Node* temp = current->next;
            current->next = current->next->next;
            delete temp;
            return;
        }
        current = current->next;
    }

    cout << "Elemen tidak ditemukan dalam linked list" << endl;
}

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
```

```
109         else
110             cout << "Elemen ditemukan pada indeks " << result << endl;
111
112         cout << "Masukan elemen yang ingin dihapus: ";
113         cin >> x;
114         deleteElement(head, x);
115
116         cout << "Elemen dalam linked list setelah penghapusan: ";
117         display(head);
118
119         return 0;
120     }
121
```

Maka Akan Menghasilkan Output:

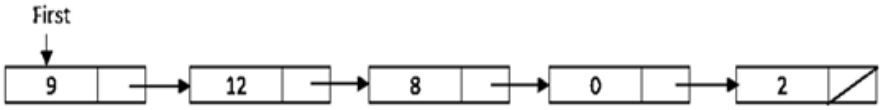


4. Unguided

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



Gambar 5-1 Ilustrasi elemen

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;
}
```

Jawab

Buat Kelas Singlelist.cpp

```cpp
#include <iostream>
#include "singlelist.h"

using namespace std;

void createList(List &L) {
    L.First = NULL;
}

address alokasi(infotype x) {
    address P = new ElmList;
    if (P != NULL) {
        P->info = x;
        P->next = NULL;
    }
    return P;
}

void dealokasi(address &P) {
    delete P;
    P = NULL;
}

void printInfo(const List &L) {
    address P = L.First;
    while (P != NULL) {
        cout << P->info << " ";
        P = P->next;
    }
    cout << endl;
}

void insertFirst(List &L, address P) {
    P->next = L.First;
    L.First = P;
}
```

Lalu Buat Kelas Singlelist.h

```cpp
#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
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;
    }
}

// Fungsi untuk menambahkan elemen baru di akhir linked list
void insertLast(Node*& head, Node*& tail, int new_data) {
    Node* new_node = new Node();
    new_node->data = new_data;
    new_node->next = nullptr;  // New node will be the last, so it points to nullptr

    if (head == nullptr) {
        // If the list is empty, the new node becomes both the head and the tail
        head = new_node;
        tail = new_node;
    } else {
        // If the list is not empty, append the new node to the tail
        tail->next = new_node;
        tail = new_node;
    }
}

// Fungsi untuk mencari elemen dalam linked list
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;
}

// Fungsi untuk menampilkan elemen dalam linked list
void display(Node* node) {
    while (node != nullptr) {
        cout << node->data << " ";
        node = node->next;
    }
    cout << endl;
}

// Fungsi untuk menghapus elemen dari linked list
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;

    // Insert elements at the start of the linked list
    insertFirst(head, tail, 3);
    insertFirst(head, tail, 5);
    insertFirst(head, tail, 7);

    // Insert elements at the end of the linked list
    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 << "Elemen ditemukan pada 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;
}

```

Lalu Membuat Kelas Main

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

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;
}
```

Maka Akan Menghasilkan Output

```
[Running] cd "c:\Users\ACER\OneDrive\Documents\HAHAHA\" && g++ main.cpp -o main && "c:\Users\ACER\OneDrive\Documents\HAHAHA\"main
9 12 8 0 2

[Done] exited with code=0 in 1.963 seconds
```

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.
```

Jawab

Kode Program:

*main.cpp

```c
#include <stdio.h>

typedef struct tElmtList {
    int info;
    struct tElmtList *next;
} ElmtList;

typedef ElmtList* address;
typedef int infotype;

address findElm(ElmtList *L, infotype x) {
    while (L != NULL) {
        if (L->info == x) {
            return L;
        }
        L = L->next;
    }
    return NULL;
}

int main() {

    ElmtList e1, e2, e3;
    e1.info = 5; e1.next = &e2;
    e2.info = 8; e2.next = &e3;
    e3.info = 10; e3.next = NULL;

    address result = findElm(&e1, 8);

    if (result != NULL) {
        printf("%d ditemukan dalam list\n", result->info);
    } else {
        printf("Elemen tidak ditemukan\n");
    }

    return 0;
}
```

Maka Akan Menghasilkan Output

```
"D:\TUGAS SEMESTER 3\findE
8 ditemukan dalam list

Process returned 0 (0x0)   execution time : 0.130 s
Press any key to continue.
```

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.
```

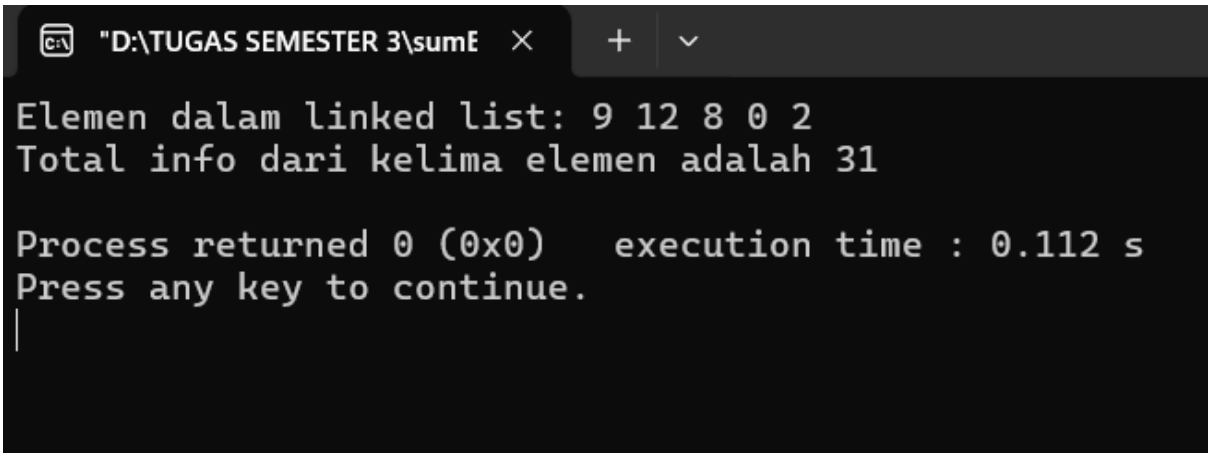
Jawab

main.cpp *main.cpp

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

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

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

    if (head == nullptr) {

        head = new_node;
        tail = new_node;
    } else {

        tail->next = new_node;
        tail = new_node;
    }
}

int sumElements(Node* head) {
    int total = 0;
    Node* current = head;
    while (current != nullptr) {
        total += current->data;
        current = current->next;
    }
    return total;
}

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

```
37          cout << node->data << " ";
38          node = node->next;
39      }
40      cout << endl;
41  }
42
43  int main() {
44      Node* head = nullptr;
45      Node* tail = nullptr;
46
47      insertLast(head, tail, 9);
48      insertLast(head, tail, 12);
49      insertLast(head, tail, 8);
50      insertLast(head, tail, 0);
51      insertLast(head, tail, 2);
52
53      cout << "Elemen dalam linked list: ";
54      display(head);
55
56      int total = sumElements(head);
57      cout << "Total info dari kelima elemen adalah " << total << endl;
58
59      return 0;
60  }
61
```

Maka Akan Menghasilkan Output



5. Kesimpulan

Dalam modul ini, kita telah mempelajari dan mempraktikkan implementasi Single Linked List dengan operasi dasar seperti insert, delete, dan searching. ADT (Abstract Data Type) untuk linked list memungkinkan pengelolaan memori secara dinamis serta fleksibilitas dalam menambah atau menghapus elemen tanpa harus memindahkan elemen lain, seperti pada array.