

LAPORAN PRAKTIKUM
MODUL 5
SINGLE LINKED LIST (BAGIAN KEDUA)



Disusun Oleh :

Farhan Kurniawan (2311104073)

Kelas:

SE-07-2

Dosen :

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

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2024

I. 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

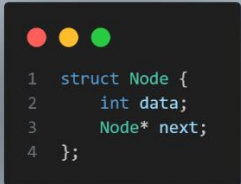
II. LANDASAN TEORI

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 *single linked list*, dan aplikasi pada bahasa pemrograman Cp semua ADT tersebut tersimpan dalam *file *.c* dan *file *.h*.

III. GUIDE

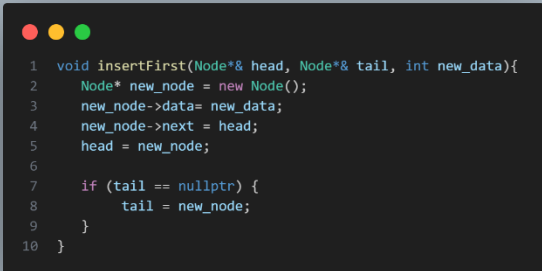
3.1. Struct Node



```
1 struct Node {
2     int data;
3     Node* next;
4 };
```

Struct Node merupakan struktur dasar dari sebuah node dalam linked list. Struktur ini memiliki dua elemen anggota: data dan next. data bertipe int, yang menyimpan nilai atau data aktual dalam node tersebut, sedangkan next adalah pointer ke Node berikutnya dalam linked list. Struktur ini memungkinkan pembentukan rantai node yang saling terhubung.

3.2. Fungsi “insertFirst”



```
1 void insertFirst(Node*& head, Node*& tail, int new_data){
2     Node* new_node = new Node();
3     new_node->data= new_data;
4     new_node->next = head;
5     head = new_node;
6
7     if (tail == nullptr) {
8         tail = new_node;
9     }
10 }
```

- Fungsi *insertFirst* digunakan untuk menambahkan node baru di awal linked list. Fungsi ini menerima tiga parameter: referensi head, tail, dan nilai *new_data* yang ingin ditambahkan. Di dalam fungsi, node baru (*new_node*) dibuat dan diberi data yang diisi dengan *new_data*. Kemudian, next dari *new_node* diatur agar menunjuk ke head (node pertama sebelumnya). Setelah itu, head diperbarui untuk menunjuk ke *new_node*, menjadikannya sebagai node pertama dalam linked list. Jika tail masih kosong (artinya linked list sebelumnya kosong), maka tail juga diatur untuk menunjuk ke *new_node*.

3.3. Fungsi “insertLast”

```
1 void insertLast(Node*& head, Node*& tail, int new_data){
2     Node*new_node = new Node ();
3     new_node->data = new_data;
4     new_node->next = nullptr;
5
6     if (head == nullptr){
7         head = new_node;
8         tail = new_node;
9     } else {
10         tail->next = new_node;
11         tail = new_node;
12     }
13 }
```

Fungsi insertLast berfungsi untuk menambahkan node baru di akhir linked list. Fungsi ini menerima tiga parameter: head, tail, dan nilai new_data yang akan ditambahkan. Pertama, node baru (new_node) dibuat dan diisi dengan new_data, lalu next dari new_node diatur ke nullptr karena node baru ini akan menjadi node terakhir. Jika head kosong, ini berarti linked list masih kosong, sehingga head dan tail diatur untuk menunjuk ke new_node. Namun, jika linked list sudah ada, next dari tail dihubungkan ke new_node, dan tail diperbarui menjadi new_node.

3.4. Fungsi “findElement”

```
1 int findElement(Node* head, int x){
2     Node* current = head;
3     int index = 0;
4
5     while(current != nullptr){
6         if (current->data == x){
7             return index;
8         }
9         current = current->next;
10        index++;
11    }
12    return -1;
13 }
```

Fungsi findElement digunakan untuk mencari nilai tertentu dalam linked list dan mengembalikan indeks pertama dari node yang mengandung nilai tersebut. Fungsi ini menerima dua parameter: head dan nilai x yang akan dicari. Fungsi melakukan iterasi dari head hingga node terakhir, mencocokkan setiap data node dengan x. Jika ditemukan, fungsi akan mengembalikan indeks node tersebut. Jika nilai x tidak ditemukan dalam linked list, fungsi akan mengembalikan -1 sebagai tanda bahwa elemen tersebut tidak ada.

3.5. Fungsi “display”

```
1 void display(Node* node){
2     while (node != nullptr){
3         cout << node->data << " ";
4         node = node->next;
5     }
6     cout << endl;
7 }
```

Fungsi display bertujuan untuk menampilkan semua elemen dalam linked list, dimulai dari head hingga akhir. Fungsi ini menerima satu parameter berupa node, yang menunjuk ke head dari linked list. Selama node tidak bernilai nullptr, fungsi akan mencetak data dari node tersebut dan melanjutkan ke node berikutnya. Setelah mencetak semua elemen, fungsi menambahkan baris baru untuk memisahkan output.

3.6. Fungsi “deleteElement”

```
1 void deleteElement(Node*& head, int x){
2     if (head == nullptr){
3         cout << "Linked List kosong" << endl;
4         return;
5     }
6
7     if (head->data == x){
8         Node* temp = head;
9         head = head->next;
10        delete temp;
11        return;
12    }
13
14    Node* current = head;
15    while(current -> next != nullptr){
16        if(current->next->data == x){
17            Node* temp = current->next;
18            current->next = current->next->next;
19            delete temp;
20            return;
21        }
22        current = current->next;
23    }
24 }
```

Fungsi main adalah titik masuk utama dari program. Di sini, linked list pertama kali dideklarasikan sebagai kosong dengan head dan tail bernilai nullptr. Beberapa elemen kemudian ditambahkan ke linked list menggunakan fungsi insertFirst dan insertLast. Setelah itu, elemen-elemen dalam linked list ditampilkan ke layar dengan fungsi display. Program kemudian meminta input pengguna untuk mencari elemen tertentu dalam linked list, dan hasilnya ditampilkan menggunakan fungsi findElement. Selanjutnya, pengguna juga diminta untuk memasukkan elemen yang ingin dihapus, yang kemudian diproses oleh fungsi deleteElement. Akhirnya, program menampilkan kondisi linked list setelah penghapusan.

3.7. Fungsi “main”

```
1 int main()
2 {
3     Node* head = nullptr;
4     Node* tail = nullptr;
5
6     insertFirst(head, tail, 3);
7     insertFirst(head, tail, 5);
8     insertFirst(head, tail, 7);
9
10    insertLast(head, tail, 11);
11    insertLast(head, tail, 14);
12    insertLast(head, tail, 18);
13
14    cout << "Elemen dalam linked list: ";
15    display(head);
16
17    int x;
18    cout << "Masukkan elemen yang ingin dicari: ";
19    cin >> x;
20
21    int result = findElement(head, x);
22
23    if (result == -1)
24        cout << "Elemen tidak ditemukan dalam linked list" << endl;
25    else
26        cout << "Elemen ditemukan pada indeks " << result << endl;
27
28    cout << "Masukkan elemen yang ingin dihapus: ";
29    cin >> x;
30    deleteElement(head, x);
31
32    cout << "Elemen dalam linked list setelah penghapusan: ";
33    display(head);
34
35    return 0;
36 }
```

Fungsi main adalah titik masuk utama dari program. Di sini, linked list pertama kali dideklarasikan sebagai kosong dengan head dan tail bernilai nullptr. Beberapa elemen kemudian ditambahkan ke linked list menggunakan fungsi insertFirst dan insertLast. Setelah itu, elemen-elemen dalam linked list ditampilkan ke layar dengan fungsi display. Program kemudian meminta input pengguna untuk mencari elemen tertentu dalam linked list, dan hasilnya ditampilkan menggunakan fungsi findElement. Selanjutnya, pengguna juga diminta untuk memasukkan elemen yang ingin dihapus, yang kemudian diproses oleh fungsi deleteElement. Akhirnya, program menampilkan kondisi linked list setelah penghapusan

IV. UNGUIDED

1. Input Program:

1.1. Langkah pertama buatlah file singlelist.h sebagai header file

```
1 #ifndef SINGLELIST_H
2 #define SINGLELIST_H
3
4 #include <iostream>
5 using namespace std;
6
7 typedef int infotype;
8 typedef struct ElmList *address;
9
10 struct ElmList {
11     infotype info;
12     address next;
13 };
14
15 struct List {
16     address First;
17 };
18
19 // Deklarasi fungsi dan prosedur
20 void createlist(List &L);
21 address alokasi(infotype x);
22 void dealokasi(address &P);
23 void printInfo(List L);
24 void insertFirst(List &L, address P);
25
26 #endif
27
```

1.2. Langkah kedua buatlah file singlelist.cpp sebagai implementasi

```
1  #include "singlelist.h"
2
3  void createList(List &L) {
4      L.First = nullptr;
5  }
6
7  address alokasi(infotype x) {
8      address P = new Elmlist;
9      P->info = x;
10     P->next = nullptr;
11     return P;
12 }
13
14 void dealokasi(address &P) {
15     delete P;
16     P = nullptr;
17 }
18
19 void printInfo(List L) {
20     address P = L.First;
21     while (P != nullptr) {
22         cout << P->info << " ";
23         P = P->next;
24     }
25     cout << endl;
26 }
27
28 void insertFirst(List &L, address P) {
29     P->next = L.First;
30     L.First = P;
31 }
32
```

1.3. Langkah ketiga buatlah file main.cpp sebagai main function

```
1  #include "singlelist.h"
2
3  int main() {
4      List L;
5      address P1, P2, P3, P4, P5 = nullptr;
6
7      createList(L);
8
9      P1 = alokasi(2);
10     insertFirst(L, P1);
11
12     P2 = alokasi(0);
13     insertFirst(L, P2);
14
15     P3 = alokasi(8);
16     insertFirst(L, P3);
17
18     P4 = alokasi(12);
19     insertFirst(L, P4);
20
21     P5 = alokasi(9);
22     insertFirst(L, P5);
23
24     printInfo(L);
25
26     return 0;
27 }
28
```

Output Program:

```
Farhan Kurniawan@LAPTOP-DOCOVQRJ MINGW64 /d/Unguided
$ ./program
9 12 8 0 2
```

2. Input Program:

```
1 #include <iostream>
2 #include <vector>
3
4 using namespace std;
5
6 // Asumsikan infotype adalah integer
7 int findElm(const vector<int>& L, int x) {
8     // Menggunakan algoritma pencarian linear
9     for (int i = 0; i < L.size(); ++i) {
10         if (L[i] == x) {
11             return i; // Mengembalikan indeks jika ditemukan
12         }
13     }
14     return -1; // Mengembalikan -1 jika tidak ditemukan
15 }
16
17 int main() {
18     vector<int> myList = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3, 2, 3, 8, 4, 6, 2, 6, 4, 3, 3, 8, 3, 2, 7, 9, 5};
19     int target = 8;
20
21     int result = findElm(myList, target);
22
23     if (result != -1) {
24         cout << target << " ditemukan dalam list" << endl;
25     } else {
26         cout << target << " tidak ditemukan dalam list" << endl;
27     }
28
29     return 0;
30 }
```

Output Program:

```
PS D:\Unguided\Nomor2> cd "d:\Unguided\Nomor2\" ; if ($?){ gcc find_element_list.c -std=c++11 -o find_element_list.exe } ; if ($?) { .\find_element_list.exe }
8 ditemukan dalam list
PS D:\Unguided\Nomor2>
```

3. Input Program:

```
1  #include <iostream>
2  using namespace std;
3
4  typedef int infotype;
5
6  struct Node {
7      infotype data;
8      Node* next;
9  };
10
11 // Fungsi untuk menghitung total nilai semua elemen dalam linked list
12 int sumElements(Node* L) {
13     int sum = 0;
14     Node* current = L;
15     while (current != NULL) {
16         sum += current->data;
17         current = current->next;
18     }
19     return sum;
20 }
21
22 // Fungsi bantu untuk membuat node baru
23 Node* createNode(infotype data) {
24     Node* newNode = new Node;
25     newNode->data = data;
26     newNode->next = NULL;
27     return newNode;
28 }
29
30 int main() {
31     // Contoh pembuatan linked list
32     Node* head = createNode(9);
33     head->next = createNode(12);
34     head->next->next = createNode(8);
35     head->next->next->next = createNode(0);
36     head->next->next->next->next = createNode(2);
37
38     // Menghitung dan mencetak total nilai dari seluruh elemen
39     int total = sumElements(head);
40     cout << "Total info dari kelima elemen adalah " << total << endl;
41
42     // Membebaskan memori
43     Node* temp;
44     while (head != NULL) {
45         temp = head;
46         head = head->next;
47         delete temp;
48     }
49
50     return 0;
51 }
52
```

Output Program:

```
PS D:\Unguided\Nomor2> cd "d:\Unguided\Nomor3\" ; if ($?) {
nerFile } ; if ($?) { .\tempCodeRunnerFile }
Total info dari kelima elemen adalah 31
PS D:\Unguided\Nomor3> 
```


V. KESIMPULAN

Kesimpulannya, operasi **searching** merupakan salah satu operasi dasar pada **single linked list** yang berfungsi untuk mencari node tertentu dengan cara mengunjungi setiap node hingga menemukan node yang diinginkan. Proses ini sangat penting karena memudahkan operasi-operasi lain seperti **insert after**, **delete after**, dan **update**. Seluruh fungsi dasar yang mendukung operasi ini merupakan bagian dari **Abstract Data Type** (ADT) **single linked list**. Dalam implementasinya pada bahasa pemrograman C, ADT tersebut umumnya tersimpan dalam file berekstensi **.c** untuk kode sumber dan **.h** untuk deklarasi fungsi.