

LAPORAN PRAKTIKUM
STRUKTUR DATA 5
"SINGLE LINKED LIST"



Oleh:

NAMA: Ammar Dzaki Nandana

NIM: 2311104071

KELAS: SE 07 B

DOSEN: Wahyu Andi Saputra

PRODI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
2023/2024

I. TUJUAN

- Memahami Struktur Data: Untuk memahami konsep dan implementasi dari struktur data Single Linked List.
- Mengimplementasikan Operasi Dasar: Untuk dapat melakukan operasi dasar pada Single Linked List, seperti penambahan (insert), penghapusan (delete), dan pencarian (search) elemen.
- Menganalisis Kelebihan dan Kekurangan: Untuk menganalisis kelebihan dan kekurangan dari penggunaan Single Linked List dibandingkan dengan struktur data lainnya.

II. DASAR TEORI

Single Linked List adalah salah satu jenis struktur data yang terdiri dari serangkaian elemen yang terhubung secara berurutan, di mana setiap elemen disebut sebagai node. Setiap node terdiri dari dua bagian:

- Data: Menyimpan informasi yang relevan.
- Pointer/Link: Menunjukkan alamat dari node berikutnya dalam urutan.

Karakteristik Single Linked List:

1. Dinamika: Kapasitas Single Linked List dapat berubah secara dinamis. Node dapat ditambahkan atau dihapus tanpa memerlukan alokasi ulang memori untuk seluruh daftar.
2. Akses Terbatas: Akses ke elemen dalam Single Linked List hanya dapat dilakukan secara berurutan, mulai dari node pertama (head) hingga node terakhir (tail).
3. Penggunaan Memori: Menggunakan lebih banyak memori dibandingkan dengan array, karena setiap node memerlukan ruang untuk menyimpan pointer ke node berikutnya.

Operasi Dasar pada Single Linked List:

1. Penambahan (Insert): Menambahkan elemen baru ke dalam list, bisa di awal, akhir, atau di posisi tertentu.
2. Penghapusan (Delete): Menghapus elemen dari list berdasarkan nilai atau posisi.
3. Pencarian (Search): Mencari elemen tertentu dalam list.
4. Traversing: Melalui setiap node dalam list untuk menampilkan atau memproses data.

III. GUIDED

```
1  #include <iostream>
2  using namespace std;
3
4  struct Node {
5      int data;
6      Node* next;
7  };
8
9  // Fungsi untuk menambahkan elemen di awal linked list
10 void insertFirst(Node*& head, Node*& tail, int new_data){
11     Node* new_node = new Node();
12     new_node->data = new_data;
13     new_node->next = head;
14     head = new_node;
15
16     if (tail == nullptr){
17         tail = new_node;
18     }
19 }
20
21 // Fungsi untuk menambahkan elemen di akhir linked list
22 void insertLast(Node*& head, Node*& tail, int new_data){
23     Node* new_node = new Node();
24     new_node->data = new_data;
25     new_node->next = nullptr;
26
27     if (head == nullptr){
28         head = new_node;
29         tail = new_node;
30     } else {
31         tail->next = new_node;
32         tail = new_node;
33     }
34 }
35
36 // Fungsi untuk mencari elemen dalam linked list
37 int findElement(Node* head, int x){
38     Node* current = head;
39     int index = 0;
40
41     while (current != nullptr){
42         if (current->data == x){
43             return index;
44         }
45         current = current->next;
46         index++;
47     }
48     return -1;
49 }
50
51 // Fungsi untuk menampilkan linked list
52 void display(Node* node){
53     while (node != nullptr){
54         cout << node->data << " ";
55         node = node->next;
56     }
57     cout << endl;
58 }
59
```

```

60 // Fungsi untuk menghapus elemen dari linked list
61 void deleteElement(Node*& head, int x){
62     if (head == nullptr){
63         cout << "Linked list kosong" << endl;
64         return;
65     }
66
67     if (head->data == x){
68         Node* temp = head;
69         head = head->next;
70         delete temp;
71         return;
72     }
73
74     Node* current = head;
75     while (current->next != nullptr){
76         if (current->next->data == x){
77             Node* temp = current->next;
78             current->next = current->next->next;
79             delete temp;
80             return;
81         }
82         current = current->next;
83     }
84 }
85
86 int main(){
87     Node* head = nullptr;
88     Node* tail = nullptr;
89
90     // Urutan dimasukkan agar sesuai dengan yang diinginkan
91     insertFirst(head, tail, 5);
92     insertFirst(head, tail, 3);
93     insertFirst(head, tail, 7);
94
95     insertLast(head, tail, 11);
96     insertLast(head, tail, 14);
97     insertLast(head, tail, 18);
98
99     // Menampilkan elemen-elemen dalam linked list
100    cout << "Element dalam linked list: ";
101    display(head);
102
103    int x;
104    cout << "Masukan element yang ingin dicari: ";
105    cin >> x;
106
107    int result = findElement(head, x);
108
109    if (result == -1)
110        cout << "Element tidak ditemukan dalam linked list" << endl;
111    else
112        cout << "Element ditemukan dalam indeks " << result << endl;
113
114    cout << "Masukan element yang ingin dihapus: ";
115    cin >> x;
116    deleteElement(head, x);
117
118    cout << "Element dalam linked list setelah penghapusan: ";
119    display(head);
120
121    return 0;
122 }
123

```

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

- **Node:** Struktur yang merepresentasikan elemen dalam linked list. Setiap node memiliki dua komponen:
 - data: Menyimpan nilai (dalam hal ini bertipe int).
 - next: Pointer yang menunjuk ke node berikutnya dalam linked list.

Fungsi Utama

1. Insert First

```
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 ini menambahkan elemen baru di awal linked list.**
- Mengalokasikan memori untuk node baru (new_node), mengisi data, dan mengatur pointer next agar menunjuk ke node saat ini (head).
- head diperbarui menjadi new_node.
- Jika tail adalah nullptr (linked list kosong), tail diatur ke new_node.

1. Insert Last

```
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;
    }
}
```

- **Fungsi ini menambahkan elemen baru di akhir linked list.**
- Mengalokasikan memori untuk node baru, mengisi data, dan menyetel next ke nullptr.
- Jika head adalah nullptr, ini berarti linked list kosong, sehingga head dan tail diatur ke new_node.
- Jika tidak kosong, node baru ditambahkan di akhir dengan mengatur pointer next dari tail ke new_node, dan tail diperbarui untuk menunjuk ke node baru.

1. Find Element

```
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 ini mencari elemen dengan nilai x dalam linked list.**
- Memeriksa setiap node dalam linked list dengan menggunakan pointer current.
- Jika nilai ditemukan, fungsi mengembalikan indeks node tersebut. Jika tidak ditemukan, mengembalikan -1.

1. Display

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

- **Fungsi ini menampilkan semua elemen dalam linked list.**
- Menggunakan loop untuk mencetak data dari setiap node hingga mencapai akhir linked list (nullptr).

1. Delete Element

```
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;
    }
}

```

- **Fungsi ini menghapus elemen dengan nilai x dari linked list.**
- Jika linked list kosong, mencetak pesan bahwa linked list kosong.
- Jika node pertama (head) adalah yang ingin dihapus, head diperbarui untuk menunjuk ke node berikutnya dan memori untuk node yang dihapus dibebaskan.
- Untuk node lainnya, fungsi mencari node dengan nilai x , dan jika ditemukan, menghapus node tersebut dengan mengatur pointer `next` dari node sebelumnya.

Fungsi **main**

```

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

    // Urutan dimasukkan agar sesuai dengan yang diinginkan
    insertFirst(head, tail, 5);
    insertFirst(head, tail, 3);
    insertFirst(head, tail, 7);

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

    // Menampilkan elemen-elemen dalam linked list
    cout << "Element dalam linked list: ";
}

```

```

display(head);

int x;
cout << "Masukan 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 dalam indeks " << result << endl;

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

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


return 0;
}

```

- Di dalam main, dua pointer (head dan tail) diinisialisasi sebagai nullptr.
- Fungsi insertFirst dan insertLast digunakan untuk menambahkan elemen ke linked list.
- Fungsi display digunakan untuk menampilkan elemen-elemen dalam linked list.
- Pengguna diminta untuk memasukkan elemen yang ingin dicari, dan fungsi findElement digunakan untuk mencarinya.
- Pengguna juga diminta untuk memasukkan elemen yang ingin dihapus, dan fungsi deleteElement digunakan untuk menghapusnya.
- Terakhir, linked list ditampilkan setelah penghapusan.

Program ini adalah implementasi sederhana dari Single Linked List dalam C++, dengan fungsi-fungsi untuk menambah, mencari, menghapus, dan menampilkan elemen dalam linked list. Struktur ini berguna untuk menyimpan data secara dinamis dengan operasi yang efisien.

IV. UNGUIDED



```
1  #ifndef SINGLELIST_H
2  #define SINGLELIST_H
3  #include <iostream>
4
5  typedef int infotype;
6  typedef struct ElmList *address;
7
8  struct ElmList
9  {
10     infotype info;
11     address next;
12 };
13
14 struct List
15 {
16     address First;
17 };
18
19
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
27 address findElm(List L, infotype x);
28 int sumInfo(List L);
29 address findMax(List L);
30 address findMin(List L);
31
32 #endif
```



```
1  #include "singlelist.h"
2  #include <iostream>
3
4  void CreateList(List &L) {
5      L.First = NULL;
6  }
7
8  address alokasi(infotype x) {
9      address P = new ElmList;
10     P->info = x;
11     P->next = NULL;
12     return P;
13 }
14
15 void dealokasi(address &P) {
16     delete P;
17 }
18
19 void printInfo(List L) {
20     address P = L.First;
21     while (P != NULL) {
22         std::cout << P->info << " ";
23         P = P->next;
24     }
25     std::cout << std::endl;
26 }
27
28 void insertFirst(List &L, address P) {
29     P->next = L.First;
30     L.First = P;
31 }
32
33 address findElm(List L, infotype x) {
34     address P = L.First;
35     while (P != NULL) {
36         if (P->info == x) {
37             return P;
38         }
39         P = P->next;
40     }
41     return NULL;
42 }
43
44 int sumInfo(List L) {
45     int total = 0;
46     address P = L.First;
47     while (P != NULL) {
48         total += P->info;
49         P = P->next;
50     }
51     return total;
52 }
```

```

1  #include <iostream>
2  #include "singlelist.h"
3  #include "singlelist.cpp"
4
5  using namespace std;
6
7  int main() {
8      List L;
9      address P1, P2, P3, P4, P5 = NULL;
10
11      CreateList(L);
12
13      P1 = alokasi(2);
14      insertFirst(L, P1);
15
16      P2 = alokasi(0);
17      insertFirst(L, P2);
18
19      P3 = alokasi(8);
20      insertFirst(L, P3);
21
22      P4 = alokasi(12);
23      insertFirst(L, P4);
24
25      P5 = alokasi(9);
26      insertFirst(L, P5);
27
28      printInfo(L);
29
30      return 0;
31 }
32
33 #include <iostream>
34 #include "singlelist.h"
35 #include "singlelist.cpp"
36
37 using namespace std;
38
39 int main() {
40     List L;
41     address P1, P2, P3, P4, P5 = NULL;
42
43     CreateList(L);
44
45     P1 = alokasi(2);
46     insertFirst(L, P1);
47
48     P2 = alokasi(0);
49     insertFirst(L, P2);
50
51     P3 = alokasi(8);
52     insertFirst(L, P3);
53
54     P4 = alokasi(12);
55     insertFirst(L, P4);
56
57     P5 = alokasi(9);
58     insertFirst(L, P5);
59
60     address foundElement = findElm(L, 8);
61     if (foundElement != NULL) {
62         cout << "8 ditemukan dalam list" << endl;
63     } else {
64         cout << "8 tidak ditemukan dalam list" << endl;
65     }
66
67     return 0;
68 }
69
70 #include <iostream>
71 #include "singlelist.h"
72 #include "singlelist.cpp"
73
74 using namespace std;
75
76 int main() {
77     List L;
78     address P1, P2, P3, P4, P5 = NULL;
79
80     CreateList(L);
81
82     P1 = alokasi(2);
83     insertFirst(L, P1);
84
85     P2 = alokasi(0);
86     insertFirst(L, P2);
87
88     P3 = alokasi(8);
89     insertFirst(L, P3);
90
91     P4 = alokasi(12);
92     insertFirst(L, P4);
93
94     P5 = alokasi(9);
95     insertFirst(L, P5);
96
97     int totalInfo = sumInfo(L);
98     cout << "Total info dari kelima elemen adalah " << totalInfo << endl;
99
100    return 0;
101 }

```

V. KESIMPULAN

Praktikum tentang Single Linked List menunjukkan bahwa struktur data ini sangat berguna untuk menyimpan data secara dinamis. Meskipun akses ke elemen tidak secepat struktur data seperti array, kelebihan dalam hal fleksibilitas dan efisiensi penggunaan memori membuatnya menjadi pilihan yang baik untuk banyak aplikasi.

Dalam praktikum ini, peserta dapat memahami cara mengimplementasikan berbagai operasi dasar dan bagaimana cara kerja Single Linked List secara keseluruhan. Selain itu, analisis kelebihan dan kekurangan menunjukkan bahwa pemilihan struktur data yang tepat tergantung pada kebutuhan spesifik dari aplikasi yang sedang dikembangkan.

VI. UNGUIDED

NO. 1

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6      float bil1, bil2;
7
8      // Input dua buah bilangan dari user
9      cout << "Masukkan bilangan pertama: ";
10     cin >> bil1;
11     cout << "Masukkan bilangan kedua: ";
12     cin >> bil2;
13
14     // Tampilkan hasil penjumlahan
15     cout << "Hasil penjumlahan: " << bil1 + bil2 << endl;
16
17     // Tampilkan hasil pengurangan
18     cout << "Hasil pengurangan: " << bil1 - bil2 << endl;
19
20     // Tampilkan hasil perkalian
21     cout << "Hasil perkalian: " << bil1 * bil2 << endl;
22
23     // Tampilkan hasil pembagian
24     // Pengecekan untuk menghindari pembagian dengan nol
25     if (bil2 != 0) {
26         cout << "Hasil pembagian: " << bil1 / bil2 << endl;
27     } else {
28         cout << "Pembagian tidak bisa dilakukan karena bilangan kedua adalah nol." << endl;
29     }
30
31     return 0;
32 }
33
34
35 //2311104071_AMMAR DZAKI NANDANA
36
```

NO. 2

```
1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
6 // Fungsi untuk mengubah angka menjadi teks
7 string angkaKeTeks(int angka)
8 {
9     string satuan[] = {"", "satu", "dua", "tiga", "empat", "lima", "enam", "tujuh", "delapan", "sembilan"};
10    string belasan[] = {"sepuluh", "sebelas", "dua belas", "tiga belas", "empat belas", "lima belas", "enam belas", "tujuh belas", "delapan belas", "sembilan belas"};
11    string puluhan[] = {"", "dua puluh", "tiga puluh", "empat puluh", "lima puluh", "enam puluh", "tujuh puluh", "delapan puluh", "sembilan puluh"};
12
13    // Jika angka adalah 0
14    if (angka == 0)
15    {
16        return "nol";
17    }
18
19    // Jika angka adalah 100
20    if (angka == 100)
21    {
22        return "seratus";
23    }
24
25    // Jika angka di bawah 10
26    if (angka < 10)
27    {
28        return satuan[angka];
29    }
30
31    // Jika angka di antara 10 dan 19 (belasan)
32    if (angka >= 10 && angka < 20)
33    {
34        return belasan[angka - 10];
35    }
36
37    // Jika angka di antara 20 dan 99 (puluhan)
38    if (angka >= 20 && angka < 100)
39    {
40        return puluhan[angka / 10] + (angka % 10 != 0 ? " " + satuan[angka % 10] : "");
41    }
42
43    return "";
44 }
45
46 int main()
47 {
48     int angka;
49
50     // Input dari pengguna
51     cout << "Masukkan angka antara 0 sampai 100: ";
52     cin >> angka;
53
54     // Memastikan input valid
55     if (angka < 0 || angka > 100)
56     {
57         cout << "Masukkan angka yang valid (0 sampai 100)." << endl;
58     }
59     else
60     {
61         // Output hasil dalam bentuk teks
62         cout << angka << ": " << angkaKeTeks(angka) << endl;
63     }
64
65     return 0;
66 }
67 //2311104071_AMMAR DZAKI NANDANA
```


NO. 3

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main() {
6     int n;
7
8     // Input dari pengguna
9     cout << "Masukkan angka: ";
10    cin >> n;
11
12    // Loop untuk mencetak pola
13    for (int i = n; i >= 1; i--) {
14        // Bagian kiri: menurun dari i hingga 1
15        for (int j = i; j >= 1; j--) {
16            cout << j << " ";
17        }
18
19        // Cetak tanda bintang *
20        cout << " ";
21
22        // Bagian kanan: menaik dari 1 hingga i
23        for (int j = 1; j <= i; j++) {
24            cout << j << " ";
25        }
26
27        // Baris baru untuk pola berikutnya
28        cout << endl;
29    }
30
31    // Output terakhir hanya tanda *
32    cout << "*" << endl;
33
34    return 0;
35 }
36 #include <iostream>
37 #include <string>
38
39 using namespace std;
40
41 // Fungsi untuk mengubah angka menjadi teks
42 string angkaKeTeks(int angka)
43 {
44     string satuan[] = {"", "satu", "dua", "tiga", "empat", "lima", "enam", "tujuh", "delapan", "sembilan"};
45     string belasan[] = {"sepuluh", "sebelas", "dua belas", "tiga belas", "empat belas", "lima belas", "enam belas", "tujuh belas", "delapan belas", "sembilan belas"};
46     string puluhan[] = {"", "", "dua puluh", "tiga puluh", "empat puluh", "lima puluh", "enam puluh", "tujuh puluh", "delapan puluh", "sembilan puluh"};
47
48     // Jika angka adalah 0
49     if (angka == 0)
50     {
51         return "nol";
52     }
53
54     // Jika angka adalah 100
55     if (angka == 100)
56     {
57         return "seratus";
58     }
59
60     // Jika angka di bawah 10
61     if (angka < 10)
62     {
63         return satuan[angka];
64     }
65
66     // Jika angka di antara 10 dan 19 (belasan)
67     if (angka >= 10 && angka < 20)
68     {
69         return belasan[angka - 10];
70     }
71
72     // Jika angka di antara 20 dan 99 (puluhan)
73     if (angka >= 20 && angka < 100)
74     {
75         return puluhan[angka / 10] + (angka % 10 != 0 ? " " + satuan[angka % 10] : "");
76     }
77
78     return "";
79 }
80
81 int main()
82 {
83     int angka;
84
85     // Input dari pengguna
86     cout << "Masukkan angka antara 0 sampai 100: ";
87     cin >> angka;
88
89     // Memastikan input valid
90     if (angka < 0 || angka > 100)
91     {
92         cout << "Masukkan angka yang valid (0 sampai 100)." << endl;
93     }
94     else
95     {
96         // Output hasil dalam bentuk teks
97         cout << angka << " : " << angkaKeTeks(angka) << endl;
98     }
99
100    return 0;
101 }
102 //2311104071_AMMAR DZAKI NANDANA
```

VII. KESIMPULAN

Penginstalan Code::Blocks beserta kompiler MinGW (untuk pengguna Windows) telah berhasil dilakukan. IDE ini siap digunakan untuk pengembangan program dalam bahasa C atau C++. Dengan menggunakan Code::Blocks, proses pengembangan menjadi lebih mudah karena tersedianya fitur debugging dan kompilasi otomatis.