

**LAPORAN PRAKTIKUM**  
**Modul 4**  
**“Single Linked List (Bagian Pertama)”**



**Disusun Oleh:**

Ahmad Al - Farizi - 2311104054

**Kelas :**

S1SE-07-02

**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**

### **2.1. Linked List dengan Pointer**

Linked list, atau yang sering disebut sebagai list, adalah struktur data yang terdiri dari serangkaian elemen yang saling terhubung dan fleksibel, memungkinkan penambahan atau pengurangan elemen sesuai kebutuhan. Elemen dalam linked list dapat berupa data tunggal, seperti nama, atau data majemuk, seperti informasi mahasiswa yang mencakup nama, NIM, dan alamat. Linked list dapat diimplementasikan menggunakan array atau pointer, namun pointer lebih disukai karena sifatnya yang dinamis dan kemudahan dalam mengelola data yang saling terhubung. Ada beberapa model dari Abstract Data Type (ADT) linked list, antara lain Single Linked List, Double Linked List, Circular Linked List, Multi Linked List, Stack, Queue, Tree, dan Graph, masing-masing dengan karakteristik dan kegunaan tersendiri. Operasi dasar yang umum dilakukan pada linked list meliputi penciptaan dan inisialisasi list, penyisipan, penghapusan, penelusuran, pencarian, dan pengubahan isi elemen.

### **2.2. Single Linked List**

Single Linked List adalah model dari Abstract Data Type (ADT) linked list yang memiliki satu arah pointer. Dalam struktur ini, setiap elemen terdiri dari beberapa komponen, yaitu data yang menyimpan informasi utama, dan suksesor yang berfungsi sebagai penghubung antar elemen. Beberapa sifat dari Single Linked List meliputi penggunaan satu pointer saja, di mana node terakhir menunjuk ke nilai kosong (Nil), kecuali dalam kasus list circular. Selain itu, Single Linked List hanya memungkinkan pembacaan maju dan pencarian data dilakukan secara sequensial jika data tidak terurut. Keunggulan dari struktur ini adalah kemudahan dalam melakukan penyisipan atau penghapusan elemen di tengah list. Istilah-istilah penting dalam Single Linked List antara lain: first atau head, yang merupakan pointer yang menunjuk ke elemen pertama, next, yang menunjukkan alamat elemen berikutnya, serta Null atau Nil, yang berarti tidak mengacu ke mana pun. Node, atau simpul, adalah tempat penyimpanan data di memori.

### 1. Pembentukan Komponen – Komponen List

- A. Pembentuk List : Proses membuat list baru menggunakan fungsi create list, yang mengatur (first list) dan last (list) ke Nil.
- B. Pengalokasian Memori : Proses untuk mengalokasikan memori bagi elemen list dengan fungsi alokasi().
- C. Dealokasi : Menghapus memori yang dialokasikan. Dalam C, gunakan free(p);, dan dalam C++ gunakan delete;.
- D. Pengecekan List : Fungsi isEmpty() untuk memeriksa apakah list kosong. Mengembalikan true jika kosong, false jika tidak.

### 2. Insert

- A. Insert First : Merupakan metode memasukkan elemen data ke dalam list yang diletakkan pada awal list.
- B. Insert Last : Merupakan metode memasukkan elemen data ke dalam list yang diletakkan pada akhir list.
- C. Insert After : Merupakan metode memasukkan data ke dalam list yang diletakkan setelah node tertentu yang ditunjuk oleh user.

### 3. View

Merupakan operasi dasar pada list yang menampilkan isi node/simpul dengan suatu penelusuran list. Mengunjungi setiap node kemudian menampilkan data yang tersimpan pada node tersebut.

## 2.3. Delete

### 1. Delete First

Adalah pengambilan atau penghapusan sebuah elemen pada awal list.

### 2. Delete Last

Merupakan pengambilan atau penghapusan suatu elemen dari akhir list.

### 3. Delete After

Merupakan pengambilan atau penghapusan node setelah node tertentu.

### 4. Delete Elemen

Operasi yang digunakan untuk menghapus dan membebaskan memori yang digunakan oleh elemen dalam list disebut dealokasi. Terdapat dua fungsi utama yang sering digunakan dalam proses ini, yaitu fungsi dealokasi(P), yang berfungsi untuk membebaskan memori yang digunakan oleh elemen P, dan fungsi delAll(L), yang membebaskan semua memori yang digunakan oleh elemen-elemen dalam list L, sehingga list L menjadi kosong.

## 2.4. Update

Merupakan operasi dasar pada list yang digunakan untuk mengupdate data yang ada di dalam list. Dengan operasi update ini kita dapat meng-update data-data node yang ada di dalam list. Proses update biasanya diawali dengan

proses pencarian terhadap data yang akan di-update.

### 3. Guided

#### 3.1. Linked List

Kode ini memiliki beberapa fungsi untuk mengelola linked list, seperti menambahkan node, menghapus node, menampilkan isi linked list, dan menghapus seluruh node. Kode ini juga memiliki sebuah struct mahasiswa untuk merepresentasikan data mahasiswa. Dalam fungsi main(), kode ini melakukan beberapa operasi seperti menambahkan node, menampilkan isi linked list, menghapus node, dan menghapus seluruh node. Dengan demikian, kode ini dapat digunakan untuk mengelola data mahasiswa dalam sebuah linked list.

Kode program :

```
1  #include <iostream>
2  #include <cstring>
3  using namespace std;
4
5  // Deklarasi Struct untuk mahasiswa
6  struct mahasiswa {
7      char nama[30];
8      char nim[10];
9  };
10
11 // Deklarasi Struct Node
12 struct Node {
13     mahasiswa data;
14     Node *next;
15 };
16
17 Node *head;
18 Node *tail;
19
20 // Inisialisasi List
21 void init() {
22     head = nullptr;
23     tail = nullptr;
24 }
25
26 // Pengecekan apakah list kosong
27 bool isEmpty() {
28     return head == nullptr;
29 }
```

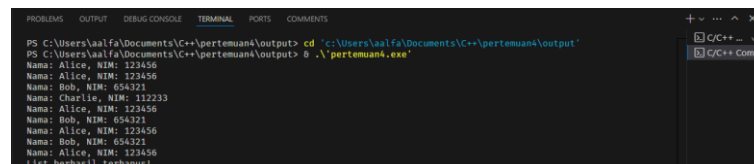
```
1 // Tambah Depan
2 void insertDepan(const mahasiswa &data) {
3     Node *baru = new Node;
4     baru->data = data;
5     baru->next = nullptr;
6     if (isEmpty()) {
7         head = tail = baru;
8     } else {
9         baru->next = head;
10        head = baru;
11    }
12 }
13
14 // Tambah Belakang
15 void insertBelakang(const mahasiswa &data) {
16     Node *baru = new Node;
17     baru->data = data;
18     baru->next = nullptr;
19     if (isEmpty()) {
20         head = tail = baru;
21     } else {
22         tail->next = baru;
23         tail = baru;
24     }
25 }
26
27 // Hitung Jumlah List
28 int hitungList() {
29     Node *current = head;
30     int jumlah = 0;
31     while (current != nullptr) {
32         jumlah++;
33         current = current->next;
34     }
35     return jumlah;
36 }
37
38 // Hapus Depan
39 void hapusDepan() {
40     if (!isEmpty()) {
41         Node *hapus = head;
42         head = head->next;
43         delete hapus;
44         if (head == nullptr) {
45             tail = nullptr; // Jika list menjadi kosong
46         }
47     } else {
48         cout << "List kosong!" << endl;
49     }
50 }
51
52 // Hapus Belakang
53 void hapusBelakang() {
54     if (!isEmpty()) {
55         if (head == tail) {
56             delete head;
57             head = tail = nullptr; // List menjadi kosong
58         } else {
59             Node *bantu = head;
60             while (bantu->next != tail) {
61                 bantu = bantu->next;
62             }
63             delete tail;
64             tail = bantu;
65             tail->next = nullptr;
66         }
67     } else {
68         cout << "List kosong!" << endl;
69     }
70 }
```

```

1 // Tampilkan List
2 void tampil() {
3     Node *current = head;
4     if (!isEmpty()) {
5         while (current != nullptr) {
6             cout << "Nama: " << current->data.nama << ", NIM: " << current->data.nim << endl;
7             current = current->next;
8         }
9     } else {
10        cout << "List masih kosong!" << endl;
11    }
12 }
13
14 // Hapus List
15 void clearList() {
16     Node *current = head;
17     while (current != nullptr) {
18         Node *hapus = current;
19         current = current->next;
20         delete hapus;
21     }
22     head = tail = nullptr;
23     cout << "List berhasil terhapus!" << endl;
24 }
25
26 // Main function
27 int main() {
28     init();
29
30     // Contoh data mahasiswa
31     mahasiswa m1 = {"Alice", "123456"};
32     mahasiswa m2 = {"Bob", "654321"};
33     mahasiswa m3 = {"Charlie", "112233"};
34
35     // Menambahkan mahasiswa ke dalam list
36     insertDepan(m1);
37     tampil();
38     insertBelakang(m2);
39     tampil();
40     insertDepan(m3);
41     tampil();
42
43     // Menghapus elemen dari list
44     hapusDepan();
45     tampil();
46     hapusBelakang();
47     tampil();
48
49     // Menghapus seluruh list
50     clearList();
51
52     return 0;
53 }

```

Output program :



```

PS C:\Users\aa1fa\Documents\C++\pertemuan4\output> cd 'C:\Users\aa1fa\Documents\C++\pertemuan4\output'
PS C:\Users\aa1fa\Documents\C++\pertemuan4\output> g++ .\pertemuan4.exe
Nama: Alice, NIM: 123456
Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321
Nama: Charlie, NIM: 112233
Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321
Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321
Nama: Alice, NIM: 123456
List berhasil terhapus!

```

### 3.2. Single Linked List


Kode ini memiliki beberapa fungsi untuk mengelola linked list, seperti menambahkan node, menampilkan isi linked list, menghitung jumlah elemen, dan menghapus semua elemen dalam list. Kode ini juga memiliki fungsi untuk mengalokasikan dan dealokasikan memori untuk node baru. Dalam fungsi main, kode ini melakukan beberapa operasi untuk mengelola data dalam sebuah linked list.

## Kode Program :

```
1 #include <iostream>
2 using namespace std;
3
4 // Definisi struktur untuk elemen list
5 struct Node {
6     int data; // Menyimpan nilai elemen
7     Node* next; // Pointer ke elemen berikutnya
8 };
9
10 // Fungsi untuk mengalokasikan memori untuk node baru
11 Node* alokasi(int value) {
12     Node* newNode = new Node; // Alokasi memori untuk elemen baru
13     if (newNode != nullptr) { // Jika alokasi berhasil
14         newNode->data = value; // Mengisi data node
15         newNode->next = nullptr; // Set next ke nullptr
16     }
17     return newNode; // Mengembalikan pointer node baru
18 }
19
20 // Fungsi untuk dealokasi memori node
21 void dealokasi(Node* node) {
22     delete node; // Mengembalikan memori yang digunakan oleh node
23 }
24
25 // Pengecekan apakah list kosong
26 bool isListEmpty(Node* head) {
27     return head == nullptr; // List kosong jika head adalah nullptr
28 }
29
30 // Menambahkan elemen di awal list
31 void insertFirst(Node* &head, int value) {
32     Node* newNode = alokasi(value); // Alokasi memori untuk elemen baru
33     if (newNode != nullptr) {
34         newNode->next = head; // Menghubungkan elemen baru ke elemen pertama
35         head = newNode; // Menetapkan elemen baru sebagai elemen pertama
36     }
37 }
38
39 // Menambahkan elemen di akhir list
40 void insertLast(Node* &head, int value) {
41     Node* newNode = alokasi(value); // Alokasi memori untuk elemen baru
42     if (newNode != nullptr) {
43         if (isListEmpty(head)) { // Jika list kosong
44             head = newNode; // Elemen baru menjadi elemen pertama
45         } else {
46             Node* temp = head;
47             while (temp->next != nullptr) { // Mencari elemen terakhir
48                 temp = temp->next;
49             }
50             temp->next = newNode; // Menambahkan elemen baru di akhir list
51         }
52     }
53 }
54
55 // Menampilkan semua elemen dalam list
56 void printList(Node* head) {
57     if (isListEmpty(head)) {
58         cout << "List kosong!" << endl;
59     } else {
60         Node* temp = head;
61         while (temp != nullptr) { // Selama belum mencapai akhir list
62             cout << temp->data << " "; // Menampilkan data elemen
63             temp = temp->next; // Melanjutkan ke elemen berikutnya
64         }
65         cout << endl;
66     }
67 }
68
69 // Menghitung jumlah elemen dalam list
70 int countElements(Node* head) {
71     int count = 0;
72     Node* temp = head;
73     while (temp != nullptr) {
74         count++; // Menambah jumlah elemen
75         temp = temp->next; // Melanjutkan ke elemen berikutnya
76     }
77     return count; // Mengembalikan jumlah elemen
78 }
```

```
1 // Menghapus semua elemen dalam list dan dealokasi memori
2 void clearList(Node* &head) {
3     while (head != nullptr) {
4         Node* temp = head; // Simpan pointer ke node saat ini
5         head = head->next; // Pindahkan ke node berikutnya
6         dealokasi(temp); // Dealokasi node
7     }
8 }
9
10 int main() {
11     Node* head = nullptr; // Membuat list kosong
12
13     // Menambahkan elemen ke dalam list
14     insertFirst(head, 10); // Menambahkan elemen 10 di awal list
15     insertLast(head, 20); // Menambahkan elemen 20 di akhir list
16     insertLast(head, 30); // Menambahkan elemen 30 di akhir list
17
18     // Menampilkan isi list
19     cout << "Isi List: ";
20     printList(head);
21
22     // Menampilkan jumlah elemen
23     cout << "Jumlah elemen: " << countElements(head) << endl;
24
25     // Menghapus semua elemen dalam list
26     clearList(head);
27
28     // Menampilkan isi list setelah penghapusan
29     cout << "Isi List setelah penghapusan: ";
30     printList(head);
31
32     return 0;
33 }
```

### Output Program :



```
PS C:\Users\aa1fa\Documents\C++> cd 'c:\Users\aa1fa\Documents\C++\pertemuan4\output'
PS C:\Users\aa1fa\Documents\C++\pertemuan4\output> 6 .\pertemuan4juga.exe
Isi List: 10 20 30
Jumlah elemen: 3
Isi List setelah penghapusan: List kosong!
PS C:\Users\aa1fa\Documents\C++\pertemuan4\output>
```

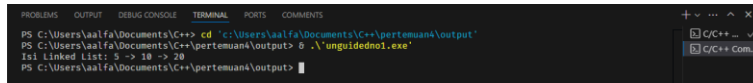


#### 4. Unguided

##### 4.1. Kode program :

```
1  #include <iostream>
2  using namespace std;
3
4  // Struktur node dari linked list
5  struct Node {
6      int data;
7      Node* next;
8  };
9
10 // Fungsi untuk menambah node di depan
11 void insertFront(Node** head, int newData) {
12     // Buat node baru
13     Node* newNode = new Node();
14     newNode->data = newData;
15
16     // Hubungkan node baru ke head lama
17     newNode->next = *head;
18
19     // Jadikan node baru sebagai head
20     *head = newNode;
21 }
22
23 // Fungsi untuk menambah node di belakang
24 void insertBack(Node** head, int newData) {
25     // Buat node baru
26     Node* newNode = new Node();
27     newNode->data = newData;
28     newNode->next = nullptr;
29
30     // Jika linked list kosong, jadikan node baru sebagai head
31     if (*head == nullptr) {
32         *head = newNode;
33         return;
34     }
35
36     // Jika tidak, cari node terakhir
37     Node* last = *head;
38     while (last->next != nullptr) {
39         last = last->next;
40     }
41
42     // Hubungkan node terakhir dengan node baru
43     last->next = newNode;
44 }
45
46 // Fungsi untuk mencetak seluruh isi linked list
47 void printList(Node* head) {
48     while (head != nullptr) {
49         cout << head->data;
50         if (head->next != nullptr) {
51             cout << " → ";
52         }
53         head = head->next;
54     }
55     cout << endl;
56 }
57
58 int main() {
59     Node* head = nullptr; // Inisialisasi linked list kosong
60
61     // Contoh input
62     insertFront(&head, 10); // Tambah node di depan (10)
63     insertBack(&head, 20);  // Tambah node di belakang (20)
64     insertFront(&head, 5);  // Tambah node di depan (5)
65
66     // Cetak linked list
67     cout << "Isi Linked List: ";
68     printList(head); // Output: 5 → 10 → 20
69
70     return 0;
71 }
72
```

Output dari kode program :



```

PS C:\Users\aaifa\Documents\C++> cd 'C:\Users\aaifa\Documents\C++\pertemuan4\output'
PS C:\Users\aaifa\Documents\C++\pertemuan4\output> .\unguidednol.exe
Isi Linked List: 5 -> 18 -> 20
PS C:\Users\aaifa\Documents\C++\pertemuan4\output>
  
```

#### 4.2. Kode program :

```

1  #include <iostream>
2  using namespace std;
3
4  // Struktur node dari linked list
5  struct Node {
6      int data;
7      Node* next;
8  };
9
10 // Fungsi untuk menambah node di depan
11 void insertFront(Node** head, int newData) {
12     Node* newNode = new Node();
13     newNode->data = newData;
14     newNode->next = *head;
15     *head = newNode;
16 }
17
18 // Fungsi untuk menambah node di belakang
19 void insertBack(Node** head, int newData) {
20     Node* newNode = new Node();
21     newNode->data = newData;
22     newNode->next = nullptr;
23
24     if (*head == nullptr) {
25         *head = newNode;
26         return;
27     }
28
29     Node* last = *head;
30     while (last->next != nullptr) {
31         last = last->next;
32     }
33     last->next = newNode;
34 }
35
36 // Fungsi untuk menghapus node dengan nilai tertentu
37 void deleteNode(Node** head, int key) {
38     Node* temp = *head;
39     Node* prev = nullptr;
40
41     // Jika node head yang akan dihapus
42     if (temp != nullptr && temp->data == key) {
43         *head = temp->next; // Ubah head
44         delete temp; // Hapus node
45         return;
46     }
47
48     // Cari node yang akan dihapus, simpan node sebelumnya
49     while (temp != nullptr && temp->data != key) {
50         prev = temp;
51         temp = temp->next;
52     }
53
54     // Jika node dengan nilai tersebut tidak ditemukan
55     if (temp == nullptr) {
56         cout << "Nilai " << key << " tidak ditemukan dalam linked list.\n";
57         return;
58     }
59
60     // Hapus node
61     prev->next = temp->next;
62     delete temp;
63 }
64
65 // Fungsi untuk mencetak seluruh isi linked list
66 void printList(Node* head) {
67     while (head != nullptr) {
68         cout << head->data;
69         if (head->next != nullptr) {
70             cout << " -> ";
71         }
72         head = head->next;
73     }
74     cout << endl;
75 }
  
```

```
1  int main() {
2      Node* head = nullptr; // Inisialisasi linked list kosong
3
4      // Contoh input
5      insertFront(&head, 10); // Tambah node di depan (10)
6      insertBack(&head, 20);  // Tambah node di belakang (20)
7      insertFront(&head, 5);  // Tambah node di depan (5)
8
9      // Cetak linked list sebelum penghapusan
10     cout << "Linked List sebelum penghapusan: ";
11     printList(head); // Output: 5 → 10 → 20
12
13     // Hapus node dengan nilai 10
14     deleteNode(&head, 10);
15
16     // Cetak linked list setelah penghapusan
17     cout << "Linked List setelah penghapusan: ";
18     printList(head); // Output: 5 → 20
19
20     return 0;
21 }
22
```

Output dari kode program :

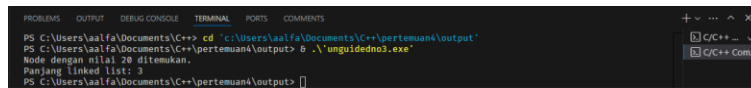


```
PS C:\Users\aa1fa\Documents\C++> cd 'C:\Users\aa1fa\Documents\C++\pertemuan4\output'
PS C:\Users\aa1fa\Documents\C++\pertemuan4\output> g++ 'unguldedno2.exe'
PS C:\Users\aa1fa\Documents\C++\pertemuan4\output> .\unguldedno2.exe
Linked List sebelum penghapusan: 5 -> 10 -> 20
Linked List setelah penghapusan: 5 -> 20
PS C:\Users\aa1fa\Documents\C++\pertemuan4\output>
```

#### 4.3. Kode program :

```
1 #include <iostream>
2 using namespace std;
3
4 // Struktur node dari linked list
5 struct Node {
6     int data;
7     Node* next;
8 };
9
10 // Fungsi untuk menambah node di depan
11 void insertFront(Node** head, int newData) {
12     Node* newNode = new Node();
13     newNode->data = newData;
14     newNode->next = *head;
15     *head = newNode;
16 }
17
18 // Fungsi untuk menambah node di belakang
19 void insertBack(Node** head, int newData) {
20     Node* newNode = new Node();
21     newNode->data = newData;
22     newNode->next = nullptr;
23
24     if (*head == nullptr) {
25         *head = newNode;
26         return;
27     }
28
29     Node* last = *head;
30     while (last->next != nullptr) {
31         last = last->next;
32     }
33     last->next = newNode;
34 }
35
36 // Fungsi untuk mencari node dengan nilai tertentu
37 bool searchNode(Node* head, int key) {
38     Node* current = head;
39     while (current != nullptr) {
40         if (current->data == key) {
41             return true; // Node ditemukan
42         }
43         current = current->next;
44     }
45     return false; // Node tidak ditemukan
46 }
47
48 // Fungsi untuk menghitung panjang linked list
49 int countNodes(Node* head) {
50     int count = 0;
51     Node* current = head;
52     while (current != nullptr) {
53         count++;
54         current = current->next;
55     }
56     return count;
57 }
58
59 // Fungsi untuk mencetak seluruh isi linked list
60 void printList(Node* head) {
61     while (head != nullptr) {
62         cout << head->data;
63         if (head->next != nullptr) {
64             cout << " -> ";
65         }
66         head = head->next;
67     }
68     cout << endl;
69 }
70
71 int main() {
72     Node* head = nullptr; // Inisialisasi linked list kosong
73
74     // Contoh input
75     insertFront(&head, 10); // Tambah node di depan (10)
76     insertBack(&head, 20); // Tambah node di belakang (20)
77     insertFront(&head, 5); // Tambah node di depan (5)
78
79     // Cari node dengan nilai 20
80     int searchValue = 20;
81     if (searchNode(head, searchValue)) {
82         cout << "Node dengan nilai " << searchValue << " ditemukan." << endl;
83     } else {
84         cout << "Node dengan nilai " << searchValue << " tidak ditemukan." << endl;
85     }
86
87     // Hitung panjang linked list
88     int length = countNodes(head);
89     cout << "Panjang linked list: " << length << endl;
90
91     return 0;
92 }
93
```

Output dari kode program :



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
PS C:\Users\aaifa\Documents\C++> cd 'c:\Users\aaifa\Documents\C++\pertemuan4\output'
PS C:\Users\aaifa\Documents\C++\pertemuan4\output> B .\\"unguidedno3.exe\"
Node dengan nilai 20 ditemukan.
Panjang linked list: 3
PS C:\Users\aaifa\Documents\C++\pertemuan4\output> |
```

## 5. Kesimpulan

Linked list adalah struktur data dinamis yang fleksibel, memungkinkan penambahan dan penghapusan elemen dengan efisien menggunakan pointer. Salah satu modelnya adalah single linked list, terdiri dari node yang saling terhubung satu arah, di mana setiap node memiliki data dan pointer yang menunjuk ke node berikutnya, sementara node terakhir menunjuk ke nilai null. Operasi dasar pada Single Linked List meliputi penambahan elemen di awal, akhir, atau setelah node tertentu (insert), serta penghapusan elemen pertama, terakhir, atau setelah node tertentu (delete). Selain itu, linked list mendukung pengecekan apakah list kosong, pencarian elemen, dan pembaruan data (update). Kelebihan Single Linked List adalah kemudahan dalam penambahan dan penghapusan elemen, meskipun pencarian harus dilakukan secara sekuensial.

