MAKALAH
SINGLE LINKED LIST

Disusun oleh :

Nama	: L Hafid Alkhair
NIM	: 2023903430060
Kelas	: TRKJ 1.C
Jurusan	: Teknologi Informasi dan Komputer
Program Studi	: Teknologi Rekayasa Komputer Jaringan
Dosen Pengajar	: Indrawati, SST. MT

JURUSAN TEKNOLOGI INFORMASI KOMPUTER

PRODI TEKNOLOGI REKAYASA KOMPUTER JARINGAN

POLITEKNIK NEGERI LHOKSEUMAWE

TAHUN AJARAN 2023/2024

`

DAFTAR ISI

BAB I 2

PEMBAHASAN 2

A. Pengertian Linked List 2

B. Definisi Linked List 6

C. Representasi Simpul (Node). 7

D. Alokasi Simpul 9

E. Membuat Node Di Depan 13

F. Membuat Node Di Belakang. 16

G. Mebuat Node Di Tengah 19

H. Menampilkan/ Membaca isi Linked List 22

I. Beberapa Metode Single Linked list 24

J. Jenis-jenis Single Linked List 26

BAB II 29

PENUTUP 29

Kesimpulan 29

DAFTAR ISI 31

`

BAB I
PEMBAHASAN

A. Pengertian Linked List

Salah satu bentuk struktur data yang berisi kumpulan data yang tersusun secara sekuensial, saling bersambungan, dinamis dan terbatas adalah senarai berkait (linked list). Suatu senarai berkait (linked list) adalah suatu simpul (node) yang dikaitkan dengan simpul yang lain dalam suatu urutan tertentu. Suatu simpul dapat berbentuk suatu struktur atau class. Simpul harus mempunyai satu atau lebih elemen struktur atau class yang berisi data.

Secara teori, linked list adalah sejumlah node yang dihubungkan secara linier dengan bantuan pointer. Dikatakan singly (single) linked apabila hanya ada satu pointer yang menghubungkan setiap node. Singly artinya field pointer-nya hanya satu buah saja dan satu arah.

Senarai berkait adalah struktur data yang paling dasar. Senarai berkait terdiri atas sejumlah unsur-unsur dikelompokkan, atau terhubung, bersama-sama di suatu deret yang spesifik. Senarai berkait bermanfaat di dalam memelihara koleksi-koleksi data, yang serupa dengan array/larik yang sering digunakan. Bagaimanapun juga, senarai berkait memberikan keuntungan-keuntungan penting yang melebihi array/larik dalam banyak hal. Secara rinci, senarai berkait lebih efisien di dalam melaksanakan penyisipan-penyisipan dan penghapusan-penghapusan. Senarai berkait juga menggunakan alokasi penyimpanan secara dinamis, yang merupakan penyimpanan yang dialokasikan pada runtime. Karena di dalam banyak aplikasi, ukuran dari data itu tidak diketahui pada saat kompile, hal ini bisa merupakan suatu atribut yang baik juga.

Setiap node akan berbentuk struct dan memiliki satu buah field bertipe struct yang sama, yang berfungsi sebagai pointer. Dalam menghubungkan setiap node, kita dapat menggunakan cara first-create-first-access ataupun first-create-last-access. Yang berbeda dengan deklarasi struct sebelumnya adalah satu field bernama next,

`

yang bertipe struct tnode. Hal ini sekilas dapat membingungkan. Namun, satu hal yang jelas, variabel next ini akan menghubungkan kita dengan node di sebelah kita, yang juga bertipe struct tnode. Hal inilah yang menyebabkan next harus bertipe struct tnode.

Linked List atau juga biasa dalam Bahasa Indonesia disebut "Senarai Berantai" adalah struktur data yang terdiri dari urutan record data dimana setiap record memiliki field yang menyimpan alamat atau referensi dari record selanjutnya sesuai dengan urutan. Node merupakan suatu sebutan dari elemen data yang dihubungkan dengan link pada linked list. Biasanya linked list menggunakan pointer.

Linked List biasa juga digunakan dalam permasalahan secara *Real Time* maksudnya adalah permasalahan yang tidak bisa diprediksi seperti "Jika ada orang yang ingin memasukan data dengan sejumlah angka tapi kita tidak tahu seberapa banyak angka yang akan Ia masukan". Dari kasus inilah biasanya digunakan Linked List dan bukan Array. Karena kalau menggunakan Array kita harus mengetahui jumlah data dan urutannya secara pasti yang akan dimasukkan atau bersifat "Statis". Sedangkan Linked List bersifat "Dinamis" yaitu, angka yang ingin dimasukkan ke dalam data dan urutannya tidak terbatas/bisa kita tidak ketahui.

Dalam pembelajaran struktur data, kita akan lebih sering mengenal dengan istilah:

1. **Push untuk menambah data.**
    - PushHead – Menambah data ke barisan paling awal
    - PushTail – Menambah data ke barisan paling akhir
    - PushMid – Menambah data ke barisan di tengah (sorting**)**
2. **Pop untuk menghapus data.**
    - PopHead – Menghapus data paling awal
    - PopTail – Menghapus data paling akhir
    - PopMid – Menghapus data ditengah (sesuai parameter value)

`

Contoh pembuatan single linked list untuk menyimpan nama seseorang beserta umurnya.

1. Deklarasi Struct

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct human{
    //menampung integer umur
    int age;
    //menampung nama manusia
    char name[30];
    //menampung alamat dari data selanjutnya
    human *next;
}*head, *tail, *current;
```

Gambar 1.1 Deklarasi Struct

2. Push Head

```c
void pushHead(int age, char name[]){
    //alokasi memory untuk data baru
    current = (human*)malloc(sizeof(struct human));
    //assign data ke dalam struct
    current->age = age;
    strcpy(current->name, name);

    //apabila linked list kosong/tidak ada data
    if(head == NULL){
        head = tail = current;
    }
    //kondisi tidak kosong
    else{
        current->next = head;
        head = current;
    }
}
```

Gambar 1.2 Push head

`

3. Push Tail

```
void pushTail(int age, char name[]){
    //alokasi memory untuk data baru
    current = (human*)malloc(sizeof(struct human));
    //assign data ke dalam struct
    current->age = age;
    strcpy(current->name, name);

    //apabila linked list kosong/tidak ada data
    if(head == NULL){
        head = tail = current;
    }
    //kondisi tidak kosong
    else{
        tail->next = current;
        tail = current;
    }
    tail->next = NULL;
}
```

Gambar 1.3 Push Tail

4. Push Mid

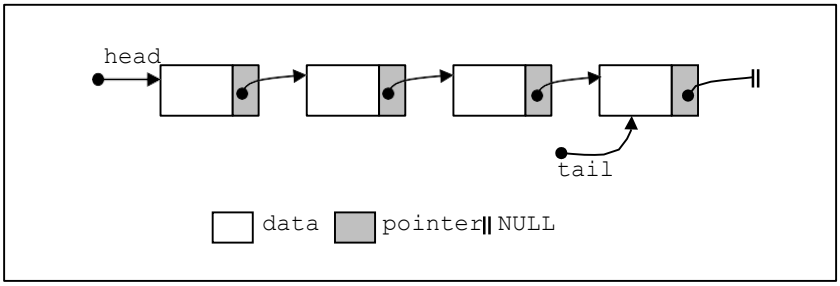
```
void pushMid(int age, char name[]){
    //alokasi memory untuk data baru
    current = (human*)malloc(sizeof(struct human));
    //assign data ke dalam struct
    current->age = age;
    strcpy(current->name, name);

    //apabila linked list kosong/tidak ada data
    if(head == NULL){
        head = tail = current;
    }
    //jika umur data yang barusan dimasukkan lebih kecil dari umur data pertama (head)
    else if(current->age < head->age){
        pushHead(age, name);
    }
    //jika umur data yang barusan dimasukkan lebih besar dari umur data terakhir (tail)
    else if(current->age > tail->age){
        pushTail(age, name);
    }
    //push ditengah-tengah
    else{
        human *temp = head;
        //mencari posisi data yang sesuai untuk diselipkan
        while(temp->next->age < current->age){
            temp = temp->next;
        }
        current->next = temp->next;
        //mengarahkan penunjuk ke alamat data baru
        temp->next = current;
    }
}
```

Gambar 1.4 Push Mid

`

B. Definisi Linked List

Single linked list atau biasa disebut linked list terdiri dari elemen-elemen individu, dimana masing-masing dihubungkan dengan pointer tunggal. Masing-masing elemen terdiri dari dua bagian, yaitu sebuah data dan sebuah pointer yang disebut dengan pointer next. Dengan menggunakan struktur two-member seperti ini, linked list dibentuk dengan cara menunjuk pointer next suatu elemen ke elemen yang mengikutinya seperti gambar 1.5. Pointer next pada elemen terakhir merupakan NULL, yang menunjukkan akhir dari suatu list. Elemen pada awal suatu list disebut head, dan elemen terakhir dari suatu list disebut tail.

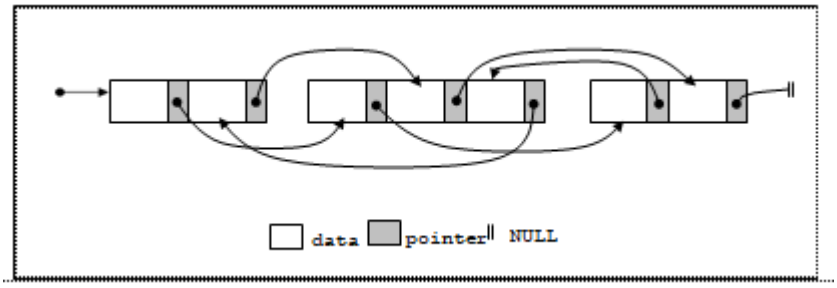


Gambar 1.5 Elemen yang Dihubungkan Bersama dalam bentuk Linked List

Untuk mengakses elemen dalam linked list, dimulai dari head dan menggunakan pointer next dari elemen selanjutnya untuk berpindah dari elemen ke elemen berikutnya sampai elemen yang diminta dicapai. Dengan single linked list, list dapat dilintasi hanya satu arah dari head ke tail karena masing-masing elemen tidak terdapat link dengan elemen sebelumnya. Sehingga, apabila kita mulai dari head dan berpindah ke beberapa elemen dan berharap dapat mengakses elemen sebelumnya, kita harus mulai dari head.

Secara konseptual, linked list merupakan deretan elemen yang berdampingan. Akan tetapi, karena elemen-elemen tersebut dialokasikan secara dinamis (menggunakan malloc), sangat penting untuk diingat bahwa kenyataannya, linked list akan terpencar- pencar di memori seperti Gambar 1.6. Pointer dari elemen ke elemen berarti sebagai penjamin bahwa semua elemen dapat diakses.

`



Gambar 1.6 Elemen pada linked list dihubungkan secara terpancar pada Alamat memori

C. Representasi Simpul (Node).

Dalam pemrograman, representasi simpul atau node adalah konsep yang sering digunakan dalam struktur data seperti graf, pohon, dan banyak struktur data lainnya. Simpul adalah elemen dasar dalam struktur data ini, dan mereka memiliki informasi atau nilai yang terkait dengan mereka, serta referensi ke simpul-simpul lain dalam struktur data. Mari kita bahas lebih lanjut dengan contoh dan program dalam bahasa C.

Simpul adalah entitas dalam struktur data yang memiliki dua komponen utama:

1. Nilai atau informasi yang dapat disimpan dalam simpul.
2. Referensi atau pointer ke simpul-simpul lain dalam struktur data.

Contoh Simpul pada Struktur Data Linked List:

Linked list adalah salah satu struktur data yang menggunakan simpul. Setiap simpul dalam linked list menyimpan data (misalnya, integer) dan pointer yang menunjuk ke simpul berikutnya dalam linked list. Berikut contoh representasi simpul dalam linked list:

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

Gambar 1.7 Simpul dalam linked list

`

Dalam contoh 1.87, struct Node adalah representasi simpul. Simpul ini memiliki dua anggota:

1. data: Menyimpan nilai atau informasi yang akan disimpan dalam simpul.
2. next: Pointer yang menunjuk ke simpul berikutnya dalam linked list. Ini membuat koneksi antara simpul-simpul.

Program Contoh Simpul pada Linked List:

Berikut ini adalah program sederhana dalam bahasa C untuk membuat simpul pada linked list dan menambahkan simpul ke linked list:

```c
struct Node {   // Definisi struktur Node
    int data;
    struct Node* next;
};

int main() {
    // Membuat simpul pertama
    struct Node* head = NULL; // Menunjuk ke simpul pertama dalam linked list
    struct Node* second = NULL; // Menunjuk ke simpul kedua dalam linked list
    struct Node* third = NULL; // Menunjuk ke simpul ketiga dalam linked list

    // Mengalokasikan memori untuk simpul-simpulnya
    head = (struct Node*)malloc(sizeof(struct Node));
    second = (struct Node*)malloc(sizeof(struct Node));
    third = (struct Node*)malloc(sizeof(struct Node));

    // Menyimpan data dan mengatur pointer next
    head->data = 1; // Menyimpan data dalam simpul pertama
    head->next = second; // Mengatur pointer next simpul pertama ke simpul kedua

    second->data = 2; // Menyimpan data dalam simpul kedua
    second->next = third; // Mengatur pointer next simpul kedua ke simpul ketiga

    third->data = 3; // Menyimpan data dalam simpul ketiga
    third->next = NULL; // Mengatur pointer next simpul ketiga ke NULL, menandakan akhir dari linked list

    // Traversing dan mencetak data dari linked list
    struct Node* current = head; // Mulai dari simpul pertama
    while (current != NULL) {
        printf("%d -> ", current->data); // Cetak data dari simpul saat ini
        current = current->next; // Pindah ke simpul berikutnya
    }
    printf("NULL\n");

    // Menghapus simpul-simpul (dealokasi memori)
    free(head);
    free(second);
    free(third);

    return 0;
}
```

Gambar 1.8 Menambah simpul ke linked list

`

Penjelasan Program 1.8 :

- Program ini membuat tiga simpul dalam linked list dengan nilai-nilai 1, 2, dan 3.
- Setiap simpul dialokasikan menggunakan fungsi malloc untuk mengalokasikan memori yang cukup untuk menyimpan sebuah simpul.
- Pointer next dalam setiap simpul diatur untuk menunjuk ke simpul berikutnya.
- Program kemudian mencetak nilai dari setiap simpul dengan melakukan traversing melalui linked list.
- Akhirnya, program membebaskan memori yang ditempati oleh setiap simpul menggunakan fungsi free.

D. Alokasi Simpul

Alokasi simpul (node allocation) adalah proses mengalokasikan ruang memori untuk menyimpan data dan referensi ke simpul lainnya dalam struktur data seperti linked list. Dalam bahasa pemrograman C, alokasi memori dapat dilakukan menggunakan fungsi malloc atau calloc. Alokasi memori ini memungkinkan kita untuk membuat simpul-simpul baru secara dinamis selama program berjalan.

Penjelasan Proses Alokasi Simpul:

Deklarasi Struktur Node:

Pertama-tama, kita mendeklarasikan struktur Node yang akan digunakan dalam linked list. Struktur ini memiliki dua bagian: data untuk menyimpan nilai dan next untuk menunjuk ke simpul berikutnya.

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

Gambar1.9 Mendeklarasi Struktur Node

`

Penggunaan Fungsi malloc:

Fungsi malloc digunakan untuk mengalokasikan memori dinamis sejumlah byte tertentu dan mengembalikan alamat memori pertama dari blok tersebut. Dalam kasus alokasi simpul, malloc digunakan untuk mengalokasikan memori untuk menyimpan simpul baru.

```
1   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
```

Gambar 1.10 Penggunaan Fungso malloc

Dalam contoh di atas, sizeof(struct Node) mengembalikan ukuran memori yang diperlukan untuk menyimpan satu simpul dalam linked list.

Pemeriksaan Keberhasilan Alokasi:

Penting untuk memeriksa apakah alokasi memori berhasil atau tidak. Jika alokasi gagal (karena kekurangan memori, misalnya), malloc akan mengembalikan NULL. Oleh karena itu, perlu dilakukan pemeriksaan:

```
1  if (newNode == NULL) {
2      printf("Error: Unable to allocate memory for new node.\n");
3      exit(-1);
4  }
```

Gambar 1.11 Pemeriksaan Keberhasilan Alokasi

Jika alokasi gagal, program akan mencetak pesan kesalahan dan keluar dari program dengan kode kesalahan -1.

`

Contoh Program Alokasi Simpul pada Linked List:

Berikut adalah contoh lengkap program dalam bahasa C yang menggunakan alokasi simpul untuk membuat dan menampilkan linked list yang berisi tiga simpul.

```
4   // Struktur simpul
5   struct Node {
6       int data;
7       struct Node* next;
8   };
```

Gambar 1.12 Pendefinisi Struktur 'Node'

struct Node adalah sebuah struktur yang memiliki dua anggota: data (untuk menyimpan nilai dari simpul) dan next (untuk menyimpan alamat dari simpul berikutnya dalam linked list).

```
9   // Fungsi untuk membuat simpul baru
10  struct Node* createNode(int data) {
11      struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
12      if (newNode == NULL) {
13          printf("Alokasi memori gagal\n");
14          exit(1);
15      }
16      newNode->data = data;
17      newNode->next = NULL;
18      return newNode;
19  }
```

Gambar 1.13 Fungsi createNode(int data)

createNode() adalah fungsi untuk membuat simpul baru.Fungsi ini juga mengalokasikan memori untuk simpul baru menggunakan fungsi malloc(). Jika alokasi memori gagal, program akan menampilkan pesan kesalahan dan keluar menggunakan exit(1). Dan nilai data dimasukkan ke dalam simpul baru, dan next diatur menjadi NULL. Fungsi mengembalikan alamat dari simpul baru yang telah dibuat.

`

```c
// Fungsi untuk menampilkan linked list
void printLinkedList(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}
```

Gambar 1.14 Fungsi printLinkedList(struct Node* head)

printLinkedList() adalah fungsi untuk menampilkan linked list dari simpul pertama hingga terakhir.Fungsi ini menggunakan pointer current untuk melacak simpul saat ini.Selama current bukan NULL, nilai dari simpul saat ini dicetak, dan current diubah menjadi simpul berikutnya dalam linked list.Fungsi menampilkan NULL ketika mencapai akhir linked list.

```c
int main() {
    // Membuat tiga simpul
    struct Node* firstNode = createNode(1);
    struct Node* secondNode = createNode(2);
    struct Node* thirdNode = createNode(3);
    // Menghubungkan simpul-simpulnya untuk membentuk linked list
    firstNode->next = secondNode;
    secondNode->next = thirdNode;
    // Menampilkan linked list
    printf("Linked List: ");
    printLinkedList(firstNode);
    // Membebaskan memori yang dialokasikan untuk simpul-simpul
    free(firstNode);
    free(secondNode);
    free(thirdNode);
    return 0;
}
```

Gambar 1.15 Fungsi main()

Di dalam fungsi main(), tiga simpul (firstNode, secondNode, dan thirdNode) dibuat menggunakan fungsi createNode() untuk membentuk linked list.Simpul-simpul inikemudian dihubungkan satu sama lain dengan mengatur next dari setiap simpul.Fungsi printLinkedList() digunakan untuk menampilkan linked list ke layar.

Setelah selesai menggunakan linked list, memori yang dialokasikan untuk simpul-simpul tersebut dibebaskan menggunakan fungsi free() untuk mencegah kebocoran memori.

`

```
Linked List: 1 -> 2 -> 3 -> NULL

--------------------------------
```

Gambar 1.16 Hasil Ouput pada program Linked list

E. Membuat Node Di Depan

Penambahan node baru akan dikaitan di node paling depan, namun pada saat pertama kali (data masih kosong), maka penambahan data dilakukan dengan cara: node head ditunjukkan ke node baru tersebut. Prinsipnya adalah mengkaitkan node baru dengan head, kemudian head akan menunjuk pada data baru tersebut sehingga head akan tetap selalu menjadi data terdepan.

Contoh Program :

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

Gambar 1.17 Pendefinisian Struktur Node

struct Node adalah sebuah struktur yang memiliki dua anggota: data (untuk menyimpan nilai dari simpul) dan next (untuk menyimpan alamat dari simpul berikutnya dalam linked list).

```c
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Alokasi memori gagal\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
```

Gambar 1.18 Fungsi createNode(int data)

createNode() adalah fungsi untuk membuat simpul baru.Fungsi ini mengalokasikan memori untuk simpul baru menggunakan malloc().Jika alokasi memori gagal, program akan menampilkan pesan kesalahan dan keluar menggunakan exit(1).Nilai

`

data dimasukkan ke dalam simpul baru, dan next diatur menjadi NULL.Fungsi mengembalikan alamat dari simpul baru yang telah dibuat.

```
15  void insertAtBeginning(struct Node** head, int data) {
16      struct Node* newNode = createNode(data);
17      newNode->next = *head;
18      *head = newNode;
19  }
```

Gambar 1.19 Fungsi insertAtBeginning(struct Node** head, int data)

insertAtBeginning() adalah fungsi untuk menyisipkan simpul baru di depan (awal) dari linked list.Fungsi ini mengambil alamat dari pointer ke kepala linked list (struct Node** head) dan data untuk simpul baru.Fungsi menggunakan createNode() untuk membuat simpul baru dengan data yang diberikan.Simpul baru diatur untuk menunjuk ke simpul yang saat ini adalah kepala linked list (newNode->next = *head).Pointer kepala linked list diperbarui untuk menunjuk ke simpul baru (*head= newNode).

```
20  void printLinkedList(struct Node* head) {
21      struct Node* current = head;
22      while (current != NULL) {
23          printf("%d -> ", current->data);
24          current = current->next;
25      }
26      printf("NULL\n");
27  }
```

Gambar 1.20 Fungsi printLinkedList(struct Node* head)

printLinkedList() adalah fungsi untuk menampilkan linked list dari simpul pertama hingga terakhir.Fungsi ini menggunakan pointer current untuk melacak simpul saat ini. Selama current bukan NULL, nilai dari simpul saat ini dicetak, dan current diubah menjadi simpul berikutnya dalam linked list. Fungsi menampilkan NULL ketika mencapai akhir linked list.

`

```c
int main() {
    struct Node* head = NULL;

    // Menyisipkan simpul baru di awal linked list
    insertAtBeginning(&head, 3);
    insertAtBeginning(&head, 2);
    insertAtBeginning(&head, 1);

    // Menampilkan linked list
    printf("Linked List: ");
    printLinkedList(head);

    // Membebaskan memori yang dialokasikan untuk simpul-simpul
    struct Node* current = head;
    while (current != NULL) {
        struct Node* temp = current;
        current = current->next;
        free(temp);
    }

    return 0;
}
```

Gambar 1.21 Fungsi main()

Di dalam fungsi main(), sebuah pointer head dibuat untuk merepresentasikan kepala dari linked list.Fungsi insertAtBeginning() dipanggil tiga kali untuk menyisipkan tiga simpul (dengan nilai 1, 2, dan 3) di awal linked list.Linked list kemudian ditampilkan menggunakan fungsi printLinkedList().Setelah selesai menggunakan linked list, memori yang dialokasikan untuk simpul-simpul tersebut dibebaskan menggunakan loop while pada akhir program.

```
Linked List: 1 -> 2 -> 3 -> NULL

--------------------------------
```

Gambar 1.22 Hasil Ouput

`

F. Membuat Node Di Belakang.

Penambahan data dilakukan di belakang, namun pada saat pertama kali, node langsung ditunjuk oleh head. Penambahan di belakang membutuhkan pointer bantu untuk mengetahui node terbelakang. Kemudian, dikaitkan dengan node baru.

Contoh Program :

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

Gambar 1.23 Pendefinisian Struktur Node

struct Node adalah struktur data yang digunakan untuk merepresentasikan simpul dalam linked list.Anggota data menyimpan nilai dari simpul.Anggota next adalah pointer yang menunjuk ke simpul berikutnya dalam linked list.

```
5  struct Node* createNode(int data) {
6      struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
7      if (newNode == NULL) {
8          printf("Alokasi memori gagal\n");
9          exit(1);
10     }
11     newNode->data = data;
12     newNode->next = NULL;
13     return newNode;
14 }
```

Gambar 1.24 Fungsi createNode(int data)

createNode() adalah fungsi untuk membuat simpul baru.Fungsi ini mengalokasikan memori untuk simpul baru menggunakan malloc().Jika alokasi memori gagal, program menampilkan pesan kesalahan dan keluar menggunakan exit(1).Nilai data dimasukkan ke dalam simpul baru, dan next diatur menjadi NULL.Fungsi mengembalikan alamat dari simpul baru yang telah dibuat.

`

```
15  void insertAtEnd(struct Node** head, int data) {
16      struct Node* newNode = createNode(data);
17      if (*head == NULL) {
18          *head = newNode;
19      } else {
20          struct Node* current = *head;
21          while (current->next != NULL) {
22              current = current->next;
23          }
24          current->next = newNode;
25      }
26  }
```

Gambar 1.25 Fungsi insertAtEnd(struct Node** head, int data)

insertAtEnd() adalah fungsi untuk menyisipkan simpul baru di belakang (akhir) linked list.Fungsi ini mengambil alamat dari pointer ke kepala linked list (struct Node** head) dan data untuk simpul baru.Jika linked list masih kosong (*head == NULL), simpul baru diatur sebagai kepala langsung.Jika tidak, fungsi mencari simpul terakhir dalam linked list dengan menggunakan loop while dan kemudian menambahkan simpul baru di belakangnya.

```
27  void printLinkedList(struct Node* head) {
28      struct Node* current = head;
29      while (current != NULL) {
30          printf("%d -> ", current->data);
31          current = current->next;
32      }
33      printf("NULL\n");
34  }
```

Gambar 1.26 Fungsi printLinkedList(struct Node* head)

printLinkedList() adalah fungsi untuk menampilkan linked list dari simpul pertama hingga terakhir.Fungsi ini menggunakan pointer current untuk melacak simpul saat ini.Selama current bukan NULL, nilai dari simpul saat ini dicetak, dan current diubah menjadi simpul berikutnya dalam linked list.Fungsi menampilkan NULL ketika mencapai akhir linked list.

`

```c
int main() {
    struct Node* head = NULL;

    // Menyisipkan simpul baru di belakang linked list
    insertAtEnd(&head, 1);
    insertAtEnd(&head, 2);
    insertAtEnd(&head, 3);

    // Menampilkan linked list
    printf("Linked List: ");
    printLinkedList(head);

    // Membebaskan memori yang dialokasikan untuk simpul-simpul
    struct Node* current = head;
    while (current != NULL) {
        struct Node* temp = current;
        current = current->next;
        free(temp);
    }

    return 0;
}
```

Gambar 1.27 Fungsi main()

Di dalam fungsi main(), sebuah pointer head digunakan untuk merepresentasikan kepala dari linked list. Fungsi insertAtEnd() dipanggil tiga kali untuk menyisipkan tiga simpul (dengan nilai 1, 2, dan 3) di belakang linked list. Linked list kemudian ditampilkan menggunakan fungsi printLinkedList(). Setelah selesai menggunakan linked list, memori yang dialokasikan untuk simpul-simpul tersebut dibebaskan menggunakan loop while pada akhir program.

```
Linked List: 1 -> 2 -> 3 -> NULL

--------------------------------
```

Gambar 1.28 Hasil Ouput.

`

G. Mebuat Node Di Tengah

Untuk menyisipkan simpul baru di tengah (atau di antara simpul-simpul yang sudah ada) dalam linked list, Anda memerlukan posisi tempat Anda ingin menyisipkan simpul baru. Dalam contoh berikut, saya akan menunjukkan cara menyisipkan simpul baru setelah simpul tertentu dalam linked list:

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

Gambar 1.29 Pendefinisian Struktur Node

struct Node adalah struktur data yang merepresentasikan simpul dalam linked list.Anggota data menyimpan nilai dari simpul.Anggota next adalah pointer yang menunjuk ke simpul berikutnya dalam linked list.

```
5  struct Node* createNode(int data) {
6      struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
7      if (newNode == NULL) {
8          printf("Alokasi memori gagal\n");
9          exit(1);
10     }
11     newNode->data = data;
12     newNode->next = NULL;
13     return newNode;
14 }
```

Gambar 1.30 Fungsi createNode(int data)

createNode() adalah fungsi untuk membuat simpul baru.Fungsi ini mengalokasikan memori untuk simpul baru menggunakan malloc().Jika alokasi memori gagal, program menampilkan pesan kesalahan dan keluar menggunakan exit(1).Nilai data dimasukkan ke dalam simpul baru, dan next diatur menjadi NULL.Fungsi mengembalikan alamat dari simpul baru yang telah dibuat.

`

```
15 void insertAfter(struct Node* prevNode, int data) {
16     if (prevNode == NULL) {
17         printf("Simpul sebelumnya tidak boleh NULL\n");
18         return;
19     }
20
21     struct Node* newNode = createNode(data);
22     newNode->next = prevNode->next;
23     prevNode->next = newNode;
24 }
```

Gambar 1.31 Fungsi insertAfter(struct Node* prevNode, int data)

insertAfter() adalah fungsi untuk menyisipkan simpul baru setelah simpul tertentu (prevNode) dalam linked list.Fungsi ini memeriksa apakah simpul sebelumnya (prevNode) tidak boleh NULL.Simpul baru (newNode) dibuat dan diatur untuk menunjuk ke simpul setelah prevNode.Pointer next dari prevNode diubah untuk menunjuk ke simpul baru.

```
25 void printLinkedList(struct Node* head) {
26     struct Node* current = head;
27     while (current != NULL) {
28         printf("%d -> ", current->data);
29         current = current->next;
30     }
31     printf("NULL\n");
32 }
```

Gambar 1.32 Fungsi printLinkedList(struct Node* head)

printLinkedList() adalah fungsi untuk menampilkan linked list dari simpul pertama hingga terakhir.Fungsi ini menggunakan pointer current untuk melacak simpul saat ini.Selama current bukan NULL, nilai dari simpul saat ini dicetak, dan current diubah menjadi simpul berikutnya dalam linked list.Fungsi menampilkan NULL ketika mencapai akhir linked list.

`

```c
int main() {
    struct Node* head = createNode(1);
    struct Node* secondNode = createNode(3);
    head->next = secondNode;

    // Menyisipkan simpul baru di antara simpul pertama dan kedua
    insertAfter(head, 2);

    // Menampilkan linked list
    printf("Linked List: ");
    printLinkedList(head);

    // Membebaskan memori yang dialokasikan untuk simpul-simpul
    struct Node* current = head;
    while (current != NULL) {
        struct Node* temp = current;
        current = current->next;
        free(temp);
    }

    return 0;
}
```

Gambar 1.33 Fungsi main()

Di dalam fungsi main(), dua simpul pertama dibuat (head dan secondNode) dan simpul baru dengan nilai 2 disisipkan di antara simpul pertama dan kedua menggunakan insertAfter().Program kemudian menampilkan linked list dengan menggunakan printLinkedList().Setelah selesai menggunakan linked list, memori yang dialokasikan untuk simpul-simpul tersebut dibebaskan menggunakan loop while pada akhir program.

```
Linked List: 1 -> 2 -> 3 -> NULL

--------------------------------
```

Gambar 1.34 Hasil Ouput

`

H. Menampilkan/ Membaca isi Linked List

Linked list ditelusuri satu-persatu dari awal sampai akhir node. Penelusuran dilakukan dengan menggunakan pointer bantu, karena pointer head yang menjadi tanda awal list tidak boleh berubah/berganti posisi. Penelusuran dilakukan terus sampai ditemukan node terakhir yang menunjuk ke nilai NULL. Jika tidak NULL, maka node bantu akan berpindah ke node selanjutnya dan membaca isi datanya dengan menggunakan field next sehingga dapat saling berkait.

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

Gambar 1.35 Mendefinisikan Struktur Node Linked List

Di sini, sebuah struktur bernama Node didefinisikan. Setiap node dalam linked list memiliki dua bagian: data untuk menyimpan nilai, dan next yang merupakan pointer ke node berikutnya dalam linked list.

```
5  void append(struct Node** head, int data) {
6      struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
7      newNode->data = data;
8      newNode->next = NULL;
9
10     if (*head == NULL) {
11         *head = newNode;
12     } else {
13         struct Node* temp = *head;
14         while (temp->next != NULL) {
15             temp = temp->next;
16         }
17         temp->next = newNode;
18     }
19 }
```

Gambar 1.36 Fungsi append untuk Menambahkan Node Baru ke Linked List

Fungsi append menerima dua parameter: pointer ke pointer kepala (head) dan nilai data untuk node baru. Fungsi ini mengalokasikan memori untuk node baru, mengisi nilai dan mengatur pointer next. Jika linked list kosong (*head == NULL), node baru langsung dijadikan sebagai kepala. Jika tidak, fungsi ini melintasi linked list hingga menemukan node terakhir dan menambahkan node baru di akhir.

`

```c
void printLinkedList(struct Node* head) {
    struct Node* temp = head;
    printf("Linked List: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
```

Gambar 1.37 Fungsi printLinkedList untuk Menampilkan Isi Linked List

Fungsi printLinkedList menerima pointer kepala dan menggunakan loop while untuk melintasi linked list dari awal hingga akhir. Selama melintasi, fungsi ini mencetak nilai setiap node dan kemudian berpindah ke node berikutnya menggunakan pointer next. Ketika mencapai akhir linked list (temp == NULL), fungsi mencetak "NULL" untuk menandakan akhir dari linked list.

```c
int main() {
    struct Node* head = NULL; // Inisialisasi linked list kosong

    // Menambahkan beberapa node ke linked list
    append(&head, 1);
    append(&head, 2);
    append(&head, 3);
    append(&head, 4);

    // Menampilkan isi linked list
    printLinkedList(head);

    // Membebaskan memori yang digunakan oleh linked list
    struct Node* temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }

    return 0;
}
```

Gambar 1.38 Fungsi main untuk Menjalankan Program

Dalam fungsi main, sebuah linked list kosong diinisialisasi dengan head = NULL. Kemudian, menggunakan fungsi append, beberapa node ditambahkan ke linked list.

`

Setelah itu, isi linked list ditampilkan menggunakan fungsi printLinkedList. Terakhir, memori yang dialokasikan untuk setiap node dalam linked list dibebaskan menggunakan loop while, sehingga mencegah kebocoran memori. Program ini menunjukkan cara membuat, menampilkan, dan membebaskan memori linked list dalam bahasa C.

I. Beberapa Metode Single Linked list

Single linked list adalah struktur data yang terdiri dari sejumlah simpul (node) yang terhubung satu sama lain melalui pointer. Berikut adalah beberapa metode yang umum digunakan dalam pengelolaan single linked list:

Penambahan Node

Tambahkan di Awal: Tambahkan node baru di awal linked list. Ubah pointer `next` node baru untuk menunjuk ke node pertama sebelumnya.

Tambahkan di Akhir Tambahkan node baru di akhir linked list. Jika linked list kosong, node baru menjadi node pertama. Jika tidak, cari node terakhir dan ubah pointer `next`-nya untuk menunjuk ke node baru.

Tambahkan di Tengah Tambahkan node baru di antara dua node yang sudah ada. Ubah pointer `next` node sebelumnya untuk menunjuk ke node baru, dan pointer `next` node baru untuk menunjuk ke node setelahnya.

Penghapusan Node

Hapus di Awal Hapus node pertama. Ubah pointer `head` untuk menunjuk ke node kedua.

Hapus di Akhir Hapus node terakhir. Cari node kedua terakhir, dan ubah pointer `next`-nya untuk menunjuk ke NULL.

Hapus Node Tertentu: Hapus node dengan nilai atau posisi tertentu. Cari node sebelumnya dari node yang akan dihapus, dan ubah pointer `next` node sebelumnya untuk menghindari node yang akan dihapus.

`

Pencarian Node

Berdasarkan Nilai Cari node dengan nilai tertentu dalam linked list.

Berdasarkan Posisi Cari node pada posisi tertentu dalam linked list.

Menampilkan Linked List

Iteratif: Gunakan loop untuk melintasi linked list dari awal hingga akhir dan tampilkan nilai setiap node.

Rekursif: Gunakan rekursi untuk mencetak nilai setiap node dalam linked list.

Menghitung Jumlah Node

Gunakan loop atau rekursi untuk menghitung jumlah node dalam linked list.

Membalik Linked List

Gunakan teknik iteratif atau rekursif untuk membalikkan urutan node dalam linked list.

Penggabungan Linked List

Gabungkan dua linked list menjadi satu dengan mengubah pointer `next` node terakhir dari linked list pertama untuk menunjuk ke node pertama dari linked list kedua.

Deteksi dan Penghapusan Loop

Gunakan algoritma Floyd's cycle detection atau Hashing untuk mendeteksi loop dalam linked list.

Untuk menghapus loop, temukan node dalam loop menggunakan Floyd's algorithm, lalu atur pointer `next` node terakhir dalam loop untuk menunjuk ke NULL.

Penanganan Operasi Khusus

Operasi seperti menggabungkan dua linked list secara terurut, membagi linked list menjadi dua, dan banyak operasi khusus lainnya yang tergantung pada kebutuhan spesifik.

`

Setiap metode ini dapat diimplementasikan dalam bahasa pemrograman C atau bahasa pemrograman lainnya untuk memanipulasi dan mengelola linked list sesuai kebutuhan aplikasi Anda.

J. Jenis-jenis Single Linked List

Single Linked List Biasa

Ini adalah jenis linked list paling sederhana, di mana setiap node hanya memiliki satu pointer yang menunjuk ke node berikutnya dalam urutan.

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

Gambar 1.39 Single Linked List Biasa

Circular Linked List

Circular linked list adalah linked list di mana node terakhir menunjuk kembali ke node pertama, menciptakan lingkaran dalam struktur data. Dalam implementasi C, ini dapat dicapai dengan mengatur pointer next dari node terakhir untuk menunjuk kembali ke node pertama.

```
1  struct Node {
2      int data;
3      struct Node* next;
4  };
5
6  // Inisialisasi circular linked list
7  struct Node* head = NULL; // Atau, node terakhir menunjuk ke node pertama
```

Gambar 1.40 Circular Linked List.

`

Doubly Linked List

Doubly linked list adalah linked list di mana setiap node memiliki dua pointer: satu menunjuk ke node sebelumnya dan satu lagi ke node berikutnya dalam urutan. Ini memungkinkan navigasi maju dan mundur melalui linked list.

```
1  struct Node {
2      int data;
3      struct Node* prev;
4      struct Node* next;
5  };
```

Gambar 1.41 Doubly Linked List

Singly Linked List dengan Tail Pointer

Ini adalah jenis linked list di mana ada pointer tambahan yang menunjuk ke node terakhir. Dengan adanya pointer ini, operasi penambahan di akhir linked list menjadi lebih efisien karena tidak perlu melintasi seluruh linked list setiap kali sebuah node ditambahkan di akhir.

```
1  struct Node {
2      int data;
3      struct Node* next;
4  };
5
6  struct LinkedList {
7      struct Node* head;
8      struct Node* tail;
9  };
```

Gambar 1.42 Singly Linked List dengan Tail Pointer.

Sparse Linked List

Sparse linked list digunakan ketika sebagian besar elemen dalam linked list memiliki nilai nol atau nilai default lainnya. Dalam implementasi ini, hanya node dengan nilai non-nol yang disimpan, menghemat ruang penyimpanan.

`

```
struct Node {
    int index;
// Indeks elemen non-nol dalam array atau struktur data
    int data;
    struct Node* next;
};
```

Gambar 1.43 Sparse Linked List

Self-Adjusting Linked List

Dalam jenis linked list ini, node yang sering diakses dipindahkan ke depan, meningkatkan efisiensi pencarian dan akses data. Ini dilakukan dengan memindahkan node yang diakses ke depan setiap kali diakses.

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

Gambar 1.44 Self-Adjusting Linked List

Setiap jenis linked list memiliki kegunaan dan keuntungan masing-masing tergantung pada kebutuhan aplikasi. Implementasi dan penggunaan linked list jenis tertentu akan tergantung pada tugas atau masalah yang perlu dipecahkan dalam konteks program yang sedang Anda bangun.

`

BAB II
PENUTUP

Kesimpulan

Kesimpulan tentang Single Linked List

Single linked list adalah salah satu struktur data yang paling dasar dan penting dalam ilmu komputer. Dengan menggunakan konsep node yang terhubung melalui pointer, linked list memberikan fleksibilitas dalam memanipulasi dan mengelola data secara dinamis. Berikut adalah beberapa kesimpulan penting tentang single linked list:

Single linked list memungkinkan penyimpanan data secara dinamis, yang berarti ukuran linked list dapat berubah selama program berjalan. Ini sangat berbeda dengan array yang memiliki ukuran tetap setelah alokasi awal.

Menyisipkan atau menghapus elemen di awal linked list memiliki kompleksitas waktu O(1) karena hanya memerlukan penyesuaian pointer. Namun, menyisipkan atau menghapus elemen di tengah atau akhir linked list juga memerlukan perpindahan pointer yang berarti, namun tetap lebih efisien daripada array dalam beberapa situasi.

Akses elemen dalam single linked list memerlukan pencarian dari awal hingga elemen yang diinginkan. Oleh karena itu, waktu akses adalah O(n), di mana n adalah jumlah elemen dalam linked list.

Implementasi single linked list relatif sederhana dan dapat disesuaikan dengan kebutuhan spesifik aplikasi. Masing-masing node hanya menyimpan data dan satu pointer, membuatnya mudah dimengerti dan diimplementasikan.

Single linked list menggunakan memori lebih efisien daripada array untuk menyimpan data yang bersifat dinamis. Namun, setiap node memerlukan ruang tambahan untuk menyimpan pointer ke node berikutnya.

`

Selain single linked list biasa, ada variasi lain seperti circular linked list (lingkaran tertutup), doubly linked list (dengan dua pointer, maju dan mundur), dan jenis-jenis khusus lainnya yang digunakan dalam aplikasi khusus.

Linked list digunakan dalam berbagai aplikasi termasuk implementasi antrian, tumpukan, dan sebagai bagian dari struktur data kompleks seperti graf dan tabel hash.

Dalam pengembangan perangkat lunak, pemahaman tentang single linked list adalah keterampilan yang penting. Kemampuan untuk mengimplementasikan, memanipulasi, dan memahami kompleksitas dari struktur data ini sangat berharga dalam penyelesaian berbagai masalah komputasi.

`

DAFTAR ISI

https://www.markijar.com/2015/05/linked-list-dalam-bahasa-c.html

https://www.belajarstatistik.com/blog/2022/01/29/contoh-program-single-linked-list-dalam-bahasa-c/

https://www.belajarstatistik.com/blog/2022/01/29/contoh-program-single-linked-list-dalam-bahasa-c/

https://www.belajarstatistik.com/blog/2022/01/29/contoh-program-single-linked-list-dalam-bahasa-c/