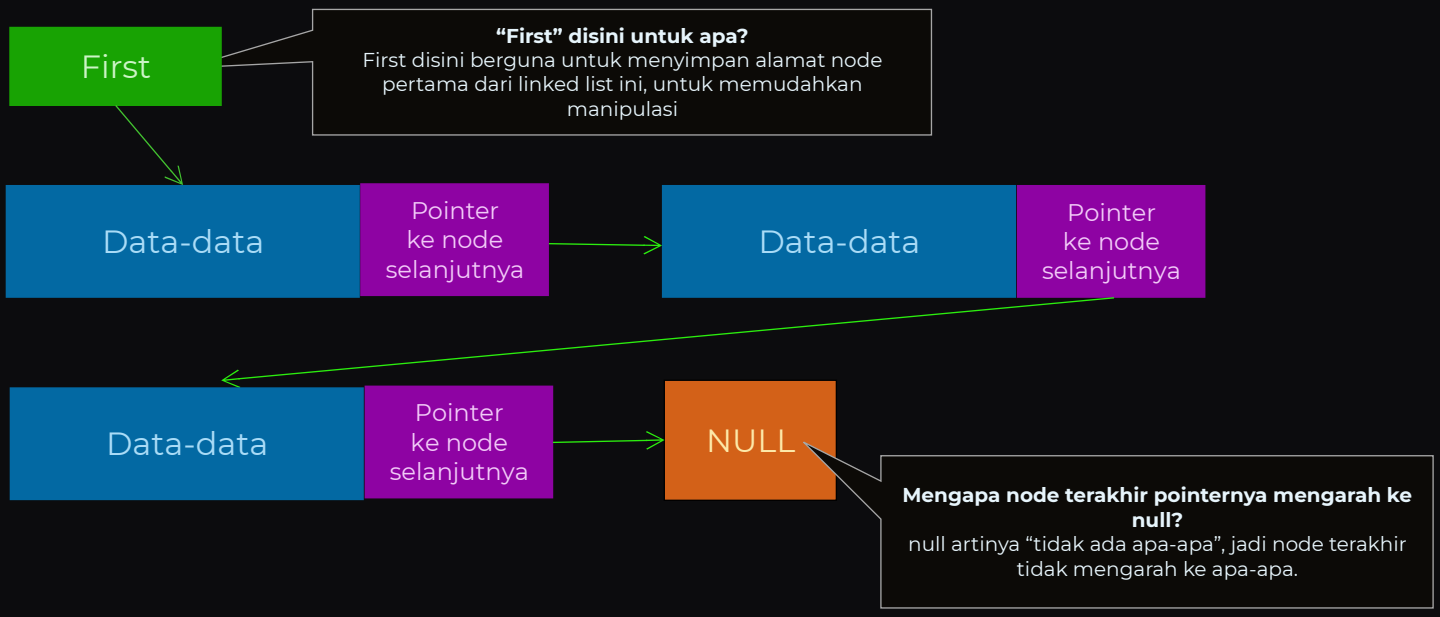
Modul 6 – Linked List 1

Penjelasan Dasar Linked List

Linked list adalah salah satu jenis struktur data yang menggunakan fitur pointer pada materi yang sudah kalian pelajari minggu sebelumnya. Linked list pada dasarnya terdiri dari 2 bagian yaitu ruangan yang menyimpan data-data seperti angka, string, struct, dan lainnya, serta bagian kedua adalah ruangan yang menyimpan alamat memori untuk arah ke node/simpul selanjutnya. Illustrasi dari linked list dapat dilihat seperti dibawah ini...



Dalam guided ini, kita akan belajar beberapa operasi dasar yang bisa dilakukan dengan linked list seperti:

- Inisialisasi/create empty list: prosedur untuk mengatur bahwa tidak ada node apapun pada linked list.
- Alokasi Node Baru / memory allocation: prosedur dimana kita mengalokasikan memori untuk menyimpan data-data yang ada di node linked list.
- Is empty: fungsi yang mengembalikan nilai true jika list kosong, dan false jika list mempunyai node
- Insert First: prosedur untuk memasukkan sebuah data dari awal list.
- Insert Last: kebalikan dari insert first, insert last memasukkan data pada bagian akhir dari list.
- Delete First: prosedur yang menghapus sebuah node dari awal list.
- Delete Last: kebalikan dari delete first, delete last menghapus data yang berada pada bagian akhir list.
- Print All: mencetak semua data di list ke layar.

tanpa berlama-lama lagi, mari kita lanjut ke kodingannya.

Teman-teman diharapkan untuk membaca comment kodingnya, karena berguna untuk mengerti cara linked list bekerja.

Modul 6 – Linked List 1

Guided

Backstory

Sebagai programmer yang ingin memprogram banyak hal, Anda mempunyai ide untuk membuat sistem untuk memudahkan pencatatan data mahasiswa pada satu ruangan. Anda membuat program yang mempunyai fungsi untuk **memasukkan data, menghapus data, dan melihat seluruh data**.
Data-data tersebut tidak terbatas jumlahnya, dan **berisikan kolom nama dan NPM.**

Kodingan

Dalam guided ini, seperti biasa kita menyiapkan 3 file yaitu **main.c, source.c, dan header.h**

Kita mulai dengan **header.h**

Kode - header.h

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

//============================//
//   Deklarasi Tipe Data      //
//============================//
// definisi tipe data string
typedef char str[64];

// definisi tipe data linked list
typedef struct Node *address; // ini digunakan untuk tipe data address dibawah

typedef struct Node {
  str nama;
  int npm;
  address next;
} tNode;

typedef struct {
  address first;
} List;

//==================================//
//   Deklarasi Prodsedur dan Fungsi  //
//==================================//
bool is_list_empty(List L);
bool is_only_have_one_data(List L);

void create_empty_list(List *L);
address memory_allocation(str nama, int npm);

void insert_first(List *L, str nama, int npm);
void insert_last(List *L, str nama, int npm);

void delete_first(List *L);
void delete_last(List *L);

void print_list(List L);

```

Kemudian kita bergerak ke **source.c**

*Note: Karena font yang digunakan, mungkin ada simbol yang belum pernah kalian temui... Simbol panah dapat diketik dengan “->”, simbol tidak sama dengan (simbol sama dengan dicoret) itu “!=”, simbol sama dengan double seperti biasa pakai “==” :D

Kode - source.c

```c
#include "header.h"

bool is_list_empty(List L) {
  // mengembalikan true jika list kosong (alamat pertama list adalah NULL/tidak ada)
  // dan false jika list tidak kosong (alamat pertama list tidak NULL)
  return (L.first == NULL);

  // di atas adalah versi singkatan
  // dibawah adalah versi yang lebih jelasnya
  //
  // bool result;
  // if (L.first == NULL) {
  //   result = true;
  // } else {
  //   result = false;
  // }
  // return result;
}

bool is_only_have_one_data(List L) {
  // mengembalikan true jika list hanya mempunyai satu data
  // dan false jika list mempunyai lebih dari satu data
  return (L.first->next == NULL);
}

void create_empty_list(List *L) {
  // menginisialisasi list dengan membuat list kosong
  // caranya adalah membuat bagian pertama dari list menuju ke null
  L->first = NULL;
}

address memory_allocation(str nama, int npm) {
  /*
    berfungsi mengalokasikan memori untuk data baru dan meng-set value kedalam memori langsung...
    karena di bahasa C untuk alokasi memori secara dinamis (seperti linked list) idak ada cara yang otomatis,
    memori harus dialokasikan secara eksplisit dengan menggunakan fungsi malloc
  */

  // kita membuat alamat temporary untuk menampung data baru
  address temp;

  // kemudian mengalokasikan memori dengan malloc,
  // malloc mempunyai parameter berupa ukuran memori yang ingin dialokasikan,
  // disini kita ingin mengalokasikan memori dengan ukuran sebanyak tNode (yang isinya nama(string), npm(integer), dan next)
  temp = malloc(sizeof(tNode));

  // setelah itu, kita mengisi data ke tempat temporary tersebut
  strcpy(temp->nama, nama);
  temp->npm = npm;
  temp->next = NULL;

  // dan akhirnya mengembalikan alamat ke pemanggil fungsi ini
  return temp;
}

void insert_first(List *L, str nama, int npm) {
  /*             ^------  ^-------  ^------
                 |        | Data yang mau dimasukkan ke list, ada nama(string), dan npm(integer)
                 | List yang ingin dimasukkan data tersebut

    Cara kerja fungsi ini dapat dilihat di bawah...
  */

  // Step 1: mengalokasikan memori untuk data baru
  address temp = memory_allocation(nama, npm);
  //             ^----------------------- Dapat diperhatikan bahwa kita memanggil fungsi alokasiMemori yang
  //                                      telah dibuat di bagian sebelumnya.

  // Step 2: Meng-set alamat next dari data baru tersebut ke alamat data pertama (First) dari list
  temp->next = L->first;

  // Step 3: Meng-set alamat data pertama dari list (First) ke alamat data baru
  L->first = temp;

  // Selesai
  printf("\nData berhasil dimasukkan dari awal list");
}

```

Kode - source.c

```c
void insert_last(List *L, str nama, int npm) {
  /*            ^------  ^-------  ^------
                |        | Data yang mau dimasukkan ke list, ada  nama(string), dan npm(integer)
                | List yang ingin dimasukkan data tersebut

   Cara kerja fungsi ini dapat dilihat di bawah...
  */

  // Step 1: Membuat variabel alamat
  address temp, last; // temp digunakan untuk data baru
  //                     last digunakan untuk menyimpan alamat terakhir dari list

  // Step 2: Cek apakah list kosong atau tidak, jika kosong maka
  //         kita langsung memasukkan node dengan insert_first
  if (is_list_empty(*L)) {
    // Step 2.A: Langsung memasukkan dengan insert_first jika list kosong.
    insert_first(L, nama, npm);
  } else {
    // Step 2.B: Memasukkan dengan cara, jika list sudah mempunyai isi.

    // Step 2.B.1: Mengalokasikan memori untuk data baru
    temp = memory_allocation(nama, npm);

    // Step 2.B.2: Mengisi alamat terakhir dari list ke variabel last
    // Cara kerjanya adalah dengan pertama mengset last ke alamat awal list,
    // kemudian dengan menggunakan looping mengset last ke alamat selanjutnya
    // sampai ditemukan NULL
    last = L->first;
    while (last->next != NULL) {
      last = last->next;
    }

    // Step 2.B.3: Meng-set alamat next dari data terakhir (variabel last) ke alamat data baru
    last->next = temp;

    // selesai
    printf("\nData berhasil dimasukkan dari akhir list");
  }
}

void delete_first(List *L) {
  //              ^------ List yang datanya mau dihapus

  // Step 1: Membuat variabel alamat untuk menyimpan alamat/node yang akan dihapus
  address temp;

  // Step 2: Cek apakah list kosong atau tidak, jika kosong maka
  //         kita tidak melakukan apa-apa
  if (is_list_empty(*L)) {
    printf("\nList kosong, tidak ada data yang dihapus");
  } else {
    // Step 3: Mengisi variabel temp dengan alamat data pertama dari list
    temp = L->first;

    // Step 4: Meng-set alamat data pertama dari list ke alamat data kedua dari list
    L->first = temp->next;

    // Step 5: Menghapus data pertama dari list
    free(temp); // free adalah lawan malloc, jika malloc mengalokasikan memori, free akan membebaskan memori.

    // selesai
    printf("\nData berhasil dihapus dari awal list");

    // FUN FACT: Apa yang terjadi jika memori tidak di free()?
    // Jawabannya adalah, memori akan terlihat masih terpakai dan tidak dapat digunakan oleh proses lain,
    // sehingga penggunaan memori aplikasi akan sama meskipun data sudah dihapus.
    // Kondisi ini dinamai dengan memory leak. Jika dibiarkan terlalu lama dan aplikasinya kompleks, maka RAM dapat penuh.
  }
}

```

Kode - source.c

```c

void delete_last(List *L) {
  //             ^------ List yang datanya mau dihapus

  // Step 1: Membuat variabel alamat untuk menyimpan alamat/node yang akan dihapus
  address temp, hapus; // temp digunakan untuk menyimpan alamat node sebelum node yang akan dihapus
  //                      hapus digunakan untuk menyimpan alamat node yang akan dihapus

  // Step 2: Cek apakah list kosong atau tidak, jika kosong maka kita tidak melakukan apa-apa
  if (is_list_empty(*L)) {
    printf("\nList kosong, tidak ada data yang dihapus");
  } else {
    // Step 3: Cek apakah list mempunyai satu data atau lebih?
    //         Jika hanya satu data, maka kita langsung menggunakan delete_first
    if (is_only_have_one_data(*L)) {
      delete_first(L);
    } else {
      // Step 4: Jika list mempunyai lebih dari satu data, maka kita menggunakan cara dibawah...

      // Step 4.1: Mengisi variabel temp dengan alamat data pertama dari list
      temp = L->first;

      // Step 4.2: Menggunakan looping untuk mengisi variabel temp dengan alamat node sebelum node terakhir
      while (temp->next->next != NULL) {
        temp = temp->next;
        //temp adalah node sebelum node terakhir
      }

      // Step 4.3: Mengisi variabel hapus dengan alamat node terakhir
      hapus = temp->next;

      // Step 4.4: Meng-set alamat next dari node sebelum node terakhir ke NULL
      temp->next = NULL;

      // Step 4.5: Meng-free memori dari node terakhir
      free(hapus);

      // selesai
      printf("\nData di akhir list berhasil dihapus");
    }
  }
}

void print_list(List L) {
  //            ^----- List yang datanya mau dihapus

  // Step 1: Membuat variabel alamat untuk menyimpan alamat/node yang akan dihapus
  address temp;

  // Step 2: Cek apakah list kosong atau tidak, jika kosong maka kita tidak melakukan apa-apa
  if (is_list_empty(L)) {
    printf("\nList kosong, tidak ada data yang ditampilkan\n");
  } else {
    // Step 3: Mengisi variabel temp dengan alamat data pertama dari list
    temp = L.first;

    // Step 4: Menggunakan looping untuk menampilkan data dari list
    while (temp != NULL) {
      printf("\nNama: %s", temp->nama);
      printf("\nNPM: %d", temp->npm);
      printf("\n------------------------\n");

      // ini akan mengubah alamat temp ke alamat node selanjutnya
      temp = temp->next;
    }
  }
}
```

Terakhir adalah **main.c**

```c
#include "header.h"

int main() {
  int menu;
  List a; // "a" adalah nama list yang akan kita pakai

  //variabel temp buat inputan
  str temp_nama;
  int temp_npm;

  create_empty_list(&a); // membuat list kosong / inisialisasi list
  do {
    system("cls");
    printf("Guided Linked List 1\n");
    printf("Dibuat oleh NAMA - KELAS - NPM\n\n"); //jangan lupa diganti identitas kalian
    printf("==== Insert Data ====\n");
    printf("[1] Insert First\n");
    printf("[2] Insert Last\n");
    printf("\n");
    printf("==== Delete Data====\n");
    printf("[3] Delete First\n");
    printf("[4] Delete Last\n");
    printf("\n");
    printf("[5] Print Data\n");
    printf("[0] Exit\n");
    printf("\n");
    printf("Pilih menu: "); scanf("%d", &menu);
    switch (menu) {
    case 1:
      printf("Masukkan nama: "); fflush(stdin); gets(temp_nama);
      printf("Masukkan npm: "); scanf("%d", &temp_npm);
      insert_first(&a, temp_nama, temp_npm);
      break;

    case 2:
      printf("Masukkan nama: "); fflush(stdin); gets(temp_nama);
      printf("Masukkan npm: "); scanf("%d", &temp_npm);
      insert_last(&a, temp_nama, temp_npm);
      break;

    case 3:
      delete_first(&a);
      break;

    case 4:
      delete_last(&a);
      break;

    case 5:
      print_list(a);
      break;

    case 0:
      printf("Mengeluarkan program...");
      break;
```

```
    default:
      printf("\nMenu tidak tersedia!");
      break;
    }

    if (menu != 0) getch();
  } while (menu != 0);

  return 0;
}
```

Aturan pengumpulan guided:
- comment tidak perlu ditulis
- format nama folder dan archive: **GD6_X_YYYYY** (X adalah kelas, YYYYY adalah 5 digit terakhir NPM), format archive zip/rar/tar/7z bebas.

“First, solve the problem. Then, write the code.”

John Johnson