**PROJEK AKHIR**

**Praktikum Struktur Data**

**“Manajemen Tugas Berdasarkan Tenggat Pengumpulannya”**

**Dosen Pengampu :**

**Randi Proska Sandra, S.Pd, M.Sc**



**Disusun oleh**

**Nama : Carel Habsian Osagi**

**Nim : 23343061**

**PROGRAM STUDI INFORMATIKA**

**FAKULTAS TEKNIK**

**UNIVERSITAS NEGERI PADANG**

**2024**

1. **Latar Belakang**

Sebagai seorang mahasiswa, saya sering dihadapkan pada banyak tugas akademik dengan tenggat waktu yang berbeda-beda. Mengendalikan semua tugas ini bisa menjadi tantangan yang besar, terutama ketika ada lebih banyak tugas yang harus diselesaikan. Akibatnya, saya merasa perlu untuk membuat sistem yang akan membantu saya mengatur pekerjaan saya dengan lebih baik.

Saya membuat keputusan untuk menggunakan bahasa C untuk program manajemen pengumpulan tugas ini karena ini adalah satu-satunya bahasa pemrograman yang saya pahami dengan baik saat ini. Selain itu, bahasa C adalah bahasa yang sering digunakan dalam praktikum struktur data, yang membuatnya pilihan yang tepat untuk proyek akhir saya.

Program ini dirancang untuk memudahkan saya dalam menambah, mengedit, menghapus, dan menampilkan tugas-tugas saya. Dengan fitur pengurutan tugas berdasarkan tenggat waktu, baik secara ascending (menaik) maupun descending (menurun), saya dapat dengan mudah melihat tugas mana yang harus diprioritaskan. Selain itu, fitur pencarian tugas berdasarkan nama juga sangat membantu ketika saya ingin menemukan tugas tertentu dengan cepat.

Proyek ini tidak hanya bertujuan untuk membantu saya dalam mengelola tugas-tugas saya, tetapi juga sebagai bagian dari laporan proyek akhir untuk mata kuliah praktikum struktur data. Dengan menyelesaikan proyek ini, saya berharap dapat meningkatkan pemahaman saya tentang struktur data, khususnya dalam penerapan singly linked list dan algoritma pengurutan, serta memperkuat keterampilan pemrograman saya dalam bahasa C.

1. **Penjelasan Program Secara Umum**

Program ini adalah sebuah aplikasi manajemen tugas yang menggunakan struktur data singly linked list tunggal (singly linked list), algoritma pengurutan merge sort, dan algoritma pencarian binary search. Berikut adalah penjelasan dari baris-baris kode dan bagaimana program ini menerapkan struktur data dan algoritma yang disebutkan:

**Struktur Data**

* **Struct Tugas**: Menyimpan informasi tugas, termasuk nama dan tenggat pengumpulan.
* **Struct Node**: Digunakan untuk elemen dalam singly linked list, menyimpan tugas dan pointer ke node berikutnya.

**Fungsi-fungsi Utama**

1. **Menambah Tugas**:
   * Fungsi tam
   * bahTugas menambahkan tugas baru ke dalam singly linked list.
2. **Mengedit Tugas**:
   * Fungsi editTugas mengubah informasi tugas yang sudah ada berdasarkan nama tugas.
3. **Menghapus Tugas**:
   * Fungsi hapusTugas menghapus tugas dari singly linked list berdasarkan nama tugas.
4. **Menampilkan Tugas**:
   * Fungsi tampilkanTugas menampilkan semua tugas yang ada dalam singly linked list.

**Fungsi Pengurutan**

* **Merge Sort**:
  + Fungsi mergeSort mengurutkan tugas berdasarkan tenggat pengumpulan dengan algoritma merge sort, yang dapat mengurutkan secara ascending atau descending.
  + Fungsi merge dan mergeDesc menggabungkan dua sub-list yang sudah diurutkan.
  + Fungsi bagi membagi singly linked list menjadi dua bagian untuk keperluan merge sort.

**Fungsi Pencarian**

* **Binary Search**:
  + Fungsi binarySearch mencari tugas berdasarkan nama menggunakan algoritma binary search setelah singly linked list diurutkan.

**Antarmuka Pengguna**

* **Menu**:
  + Fungsi menu menampilkan pilihan-pilihan operasi yang dapat dilakukan pengguna, seperti menambah, mengedit, menghapus, menampilkan, mengurutkan, dan mencari tugas.
  + Fungsi ambilInput mengambil input dari pengguna untuk berbagai operasi.

**Fungsi Utama (main)**

* Fungsi main adalah titik masuk utama program yang mengelola alur kerja berdasarkan pilihan pengguna. Program menampilkan identitas pengguna, kemudian menampilkan menu dan memproses input pengguna untuk menjalankan fungsi-fungsi yang sesuai.

**Alur Kerja Program**

1. Menampilkan identitas pengguna.
2. Menampilkan menu dan meminta pengguna memilih opsi.
3. Berdasarkan pilihan pengguna:
   * Menambah tugas dengan memasukkan nama dan tenggat.
   * Mengedit tugas dengan memasukkan nama tugas yang akan diubah dan data baru.
   * Menghapus tugas berdasarkan nama.
   * Menampilkan semua tugas.
   * Mengurutkan tugas berdasarkan tenggat secara ascending atau descending.
   * Mencari tugas berdasarkan nama menggunakan binary search.
4. Terus mengulang proses ini hingga pengguna memilih untuk keluar.

Program ini membantu dalam manajemen tugas dengan menyediakan berbagai fungsi untuk mengelola daftar tugas dengan mudah dan efisien.

1. **Penjelasan Baris Program**

* **Library C**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <windows.h>

Mengimpor pustaka standar stdio.h (input/output), stdlib.h (fungsi umum termasuk manajemen memori), string.h (manipulasi string), dan windows.h (fungsi khusus Windows seperti system("cls") untuk membersihkan layar konsol).

* **Struct Tugas**

typedef struct Tugas {

char nama[100];

char tenggat[20];

} Tugas;

Mendefinisikan struktur Tugas yang berisi dua atribut: nama (maksimal 100 karakter) dan tenggat (maksimal 20 karakter).

* **Struct Node**

typedef struct Node {

Tugas tugas;

struct Node \*next;

} Node;

Mendefinisikan struktur Node untuk singly linked list yang berisi tugas (objek Tugas) dan pointer next ke node berikutnya.

* Node \*head = NULL;

Mendeklarasikan pointer head yang menunjuk ke awal singly linked list. Diinisialisasi ke NULL.

* **Fungsi tambahTugas()**

void tambahTugas(char \*nama, char \*tenggat) {

Node \*nodeBaru = (Node \*)malloc(sizeof(Node));

strcpy(nodeBaru->tugas.nama, nama);

strcpy(nodeBaru->tugas.tenggat, tenggat);

nodeBaru->next = NULL;

if (head == NULL) {

head = nodeBaru;

} else {

Node \*temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = nodeBaru;

}

}

Fungsi tambahTugas melakukan beberapa langkah:

1. Allokasi memori untuk node baru dengan malloc(sizeof(Node)). Ini memberikan ruang memori untuk menyimpan data tugas baru.
2. Salin nilai nama dan tenggat ke dalam node baru menggunakan strcpy. Hal ini memastikan data tugas baru disimpan dengan benar di dalam node.
3. Setel next dari node baru menjadi NULL. Ini menunjukkan bahwa node baru adalah node terakhir di dalam singly linked list.
4. Jika singly linked list kosong (head == NULL), maka node baru menjadi simpul pertama. Ini adalah kasus ketika singly linked list masih kosong sehingga node baru langsung ditetapkan sebagai simpul pertama.
5. Jika singly linked list tidak kosong, tambahkan node baru di akhir singly linked list. Dalam kasus ini, fungsi mencari node terakhir dalam singly linked list dengan melintasi semua node yang ada dan menetapkan next node terakhir ke node baru.

Dengan langkah-langkah tersebut, fungsi tambahTugas dapat mengelola penambahan tugas baru ke dalam singly linked list, baik itu di awal atau di akhir singly linked list, tergantung pada kondisi saat penambahan dilakukan.

* **Fungsi editTugas()**

void editTugas(char \*nama, char \*namaBaru, char \*tenggatBaru) {

Node \*node = head;

while (node != NULL) {

if (strcmp(node->tugas.nama, nama) == 0) {

strcpy(node->tugas.nama, namaBaru);

strcpy(node->tugas.tenggat, tenggatBaru);

return;

}

node = node->next;

}

printf("Tugas tidak ditemukan.\n");

}

Fungsi editTugas mencari tugas berdasarkan nama dan mengubahnya dengan nama dan tenggat baru. Jika tugas ditemukan, mengubahnya dan keluar dari fungsi. Jika tidak ditemukan, menampilkan pesan kesalahan.

1. Allokasi memori untuk node baru menggunakan malloc(sizeof(Node)).
2. Salin nilai nama dan tenggat ke dalam node baru menggunakan strcpy.
3. Atur next dari node baru menjadi NULL.
4. Jika singly linked list kosong (head == NULL), maka node baru menjadi simpul pertama.
5. Jika singly linked list tidak kosong, tambahkan node baru di akhir singly linked list.

Dengan langkah-langkah tersebut, fungsi ini berhasil menambahkan tugas baru ke dalam singly linked list, baik di awal maupun di akhir singly linked list sesuai kondisi.

* **Fungsi hapusTugas()**

void hapusTugas(char \*nama) {

Node \*node = head, \*prev = NULL;

while (node != NULL && strcmp(node->tugas.nama, nama) != 0) {

prev = node;

node = node->next;

}

if (node == NULL) {

printf("Tugas tidak ditemukan.\n");

return;

}

if (prev == NULL) {

head = node->next;

} else {

prev->next = node->next;

}

free(node);

}

Fungsi hapusTugas memiliki langkah-langkah berikut:

1. **Inisialisasi Variabel**: Membuat pointer node dan prev yang menunjuk ke head dan NULL, secara berturut-turut.
2. **Perulangan Pencarian**: Melakukan iterasi melalui singly linked list untuk mencari tugas dengan nama yang sesuai.
   * Jika tugas ditemukan (nama sama), node tersebut dihapus dari singly linked list.
   * Jika tidak ditemukan, menampilkan pesan bahwa tugas tidak ditemukan dan menghentikan proses.
3. **Penanganan Node**: Jika node adalah simpul pertama (prev == NULL), mengubah head menjadi node setelahnya.
   * Jika node bukan simpul pertama, menghubungkan simpul sebelumnya dengan simpul setelahnya, mengabaikan node yang dihapus.
4. **Bebas Memori**: Membebaskan memori yang dialokasikan untuk node yang dihapus menggunakan free(node).

Dengan langkah-langkah ini, fungsi hapusTugas memastikan penghapusan tugas dari singly linked list berjalan dengan benar, dengan penanganan khusus jika tugas yang dihapus adalah simpul pertama.

* **Fungsi tampilkanTugas()**

void tampilkanTugas() {

Node \*node = head;

int nomor = 1;

while (node != NULL) {

printf("%d. Nama: %s, Tenggat: %s\n", nomor++, node->tugas.nama, node->tugas.tenggat);

node = node->next;

}

}

Fungsi tampilkanTugas memiliki langkah-langkah sebagai berikut:

1. **Inisialisasi Variabel**: Membuat pointer node yang menunjuk ke head untuk memulai iterasi dari simpul pertama.
2. **Nomor Urut**: Menggunakan variabel nomor untuk memberikan nomor urut pada setiap tugas yang ditampilkan.
3. **Iterasi Melalui Singly linked list**: Melakukan perulangan dengan while sampai node tidak lagi menunjuk ke NULL, artinya telah mencapai akhir singly linked list.
4. **Menampilkan Informasi Tugas**: Pada setiap iterasi, mencetak nomor urut, nama tugas, dan tenggat tugas menggunakan printf.
5. **Pembaruan Pointer Node**: Memperbarui node untuk menunjuk ke node selanjutnya dalam singly linked list dengan node = node->next.

Dengan langkah-langkah ini, fungsi tampilkanTugas dapat menampilkan semua tugas dalam singly linked list secara berurutan dengan informasi yang relevan seperti nomor urut, nama, dan tenggat tugas.

* **Fungsi merge()**

Node\* merge(Node\* kiri, Node\* kanan) {

Node\* hasil = NULL;

if (!kiri) return kanan;

if (!kanan) return kiri;

if (strcmp(kiri->tugas.tenggat, kanan->tugas.tenggat) <= 0) {

hasil = kiri;

hasil->next = merge(kiri->next, kanan);

} else {

hasil = kanan;

hasil->next = merge(kiri, kanan->next);

}

return hasil;

}

Fungsi merge adalah bagian penting dari algoritma merge sort untuk menggabungkan dua sub-singly linked list yang sudah diurutkan secara ascending berdasarkan tenggat tugas. Berikut adalah langkah-langkahnya:

1. **Pengecekan NULL**: Fungsi melakukan pengecekan untuk memastikan bahwa sub-singly linked list kiri dan kanan tidak kosong. Jika salah satu dari keduanya kosong, fungsi mengembalikan sub-singly linked list yang tidak kosong.
2. **Pembandingan Tenggat**: Fungsi membandingkan tenggat tugas dari simpul pertama di sub-singly linked list kiri dan kanan. Jika tenggat tugas di sub-singly linked list kiri lebih kecil atau sama dengan tenggat tugas di sub-singly linked list kanan, maka simpul pertama dari sub-singly linked list kiri akan menjadi simpul pertama dari sub-singly linked list hasil. Jika tidak, simpul pertama dari sub-singly linked list kanan yang menjadi simpul pertama dari sub-singly linked list hasil.
3. **Rekursi**: Setelah menentukan simpul pertama dari sub-singly linked list hasil, fungsi memanggil dirinya sendiri secara rekursif untuk menggabungkan sisa sub-singly linked list yang belum diproses.
4. **Pengembalian Hasil**: Fungsi mengembalikan sub-singly linked list hasil setelah proses penggabungan selesai.

Dengan langkah-langkah di atas, fungsi merge dapat menggabungkan dua sub-singly linked list secara ascending berdasarkan tenggat tugas, sesuai dengan aturan merge sort.

* **Fungsi mergeDesc()**

Node\* mergeDesc(Node\* kiri, Node\* kanan) {

Node\* hasil = NULL;

if (!kiri) return kanan;

if (!kanan) return kiri;

if (strcmp(kiri->tugas.tenggat, kanan->tugas.tenggat) >= 0) {

hasil = kiri;

hasil->next = mergeDesc(kiri->next, kanan);

} else {

hasil = kanan;

hasil->next = mergeDesc(kiri, kanan->next);

}

return hasil;

}

Fungsi mergeDesc adalah bagian dari algoritma merge sort yang menggabungkan dua sub-singly linked list yang telah diurutkan secara descending berdasarkan tenggat tugas. Berikut adalah langkah-langkah yang dilakukan oleh fungsi ini:

1. **Pengecekan NULL**: Fungsi melakukan pengecekan untuk memastikan bahwa sub-singly linked list kiri dan kanan tidak kosong. Jika salah satu dari keduanya kosong, fungsi mengembalikan sub-singly linked list yang tidak kosong.
2. **Pembandingan Tenggat**: Fungsi membandingkan tenggat tugas dari simpul pertama di sub-singly linked list kiri dan kanan. Jika tenggat tugas di sub-singly linked list kiri lebih besar atau sama dengan tenggat tugas di sub-singly linked list kanan, maka simpul pertama dari sub-singly linked list kiri akan menjadi simpul pertama dari sub-singly linked list hasil. Jika tidak, simpul pertama dari sub-singly linked list kanan yang menjadi simpul pertama dari sub-singly linked list hasil.
3. **Rekursi**: Setelah menentukan simpul pertama dari sub-singly linked list hasil, fungsi memanggil dirinya sendiri secara rekursif untuk menggabungkan sisa sub-singly linked list yang belum diproses.
4. **Pengembalian Hasil**: Fungsi mengembalikan sub-singly linked list hasil setelah proses penggabungan selesai.

Dengan langkah-langkah di atas, fungsi mergeDesc dapat menggabungkan dua sub-singly linked list secara descending berdasarkan tenggat tugas, sesuai dengan aturan merge sort.

* **Fungsi bagi()**

void bagi(Node\* sumber, Node\*\* referensiDepan, Node\*\* referensiBelakang) {

Node\* cepat;

Node\* lambat;

if (sumber == NULL || sumber->next == NULL) {

\*referensiDepan = sumber;

\*referensiBelakang = NULL;

} else {

lambat = sumber;

cepat = sumber->next;

while (cepat != NULL) {

cepat = cepat->next;

if (cepat != NULL) {

lambat = lambat->next;

cepat = cepat->next;

}

}

\*referensiDepan = sumber;

\*referensiBelakang = lambat->next;

lambat->next = NULL;

}

}

Fungsi bagi adalah bagian dari algoritma merge sort yang membagi singly linked list menjadi dua bagian menggunakan pendekatan two-pointer, yaitu cepat (fast) dan lambat (slow). Berikut adalah langkah-langkah yang dilakukan oleh fungsi ini:

1. **Pengecekan Kasus Basis**: Fungsi melakukan pengecekan apakah singly linked list kosong atau hanya memiliki satu node. Jika iya, maka kedua pointer referensi referensiDepan dan referensiBelakang diatur sesuai dengan kondisi tersebut.
2. **Pendekatan Two-Pointer**: Jika singly linked list memiliki lebih dari satu node, maka fungsi menggunakan dua pointer, yaitu cepat dan lambat, untuk menelusuri singly linked list. Pointer cepat melompat dua kali lipat dalam setiap iterasi, sedangkan pointer lambat melompat satu kali lipat. Hal ini memungkinkan pointer lambat berada di tengah singly linked list ketika pointer cepat mencapai akhir.
3. **Pembagian Singly linked list**: Setelah pointer cepat mencapai akhir singly linked list, pointer lambat berada di tengah. Fungsi membagi singly linked list menjadi dua bagian dengan mengatur pointer referensiDepan ke awal singly linked list dan pointer referensiBelakang ke tengah singly linked list, serta memutuskan hubungan next dari node terakhir di bagian pertama dengan node pertama di bagian kedua.

Dengan langkah-langkah di atas, fungsi bagi dapat membagi singly linked list menjadi dua bagian yang siap digunakan dalam proses pengurutan dengan algoritma merge sort.

* **Fungsi mergeSort()**

void mergeSort(Node\*\* headRef, int ascending) {

Node\* head = \*headRef;

Node\* kiri;

Node\* kanan;

if ((head == NULL) || (head->next == NULL)) {

return;

}

bagi(head, &kiri, &kanan);

mergeSort(&kiri, ascending);

mergeSort(&kanan, ascending);

if (ascending) {

\*headRef = merge(kiri, kanan);

} else {

\*headRef = mergeDesc(kiri, kanan);

}

}

Fungsi mergeSort merupakan implementasi dari algoritma merge sort yang digunakan untuk mengurutkan tugas dalam singly linked list berdasarkan tenggat. Berikut adalah penjelasan lebih detail tentang langkah-langkah yang dilakukan oleh fungsi ini:

1. **Inisialisasi Variabel**: Fungsi menginisialisasi variabel lokal head, kiri, dan kanan dengan nilai dari pointer referensi headRef.
2. **Pengecekan Kasus Basis**: Fungsi melakukan pengecekan apakah singly linked list kosong atau hanya memiliki satu node. Jika iya, maka fungsi langsung kembali tanpa melakukan pengurutan.
3. **Pemisahan Singly linked list**: Fungsi memanggil fungsi bagi untuk membagi singly linked list menjadi dua bagian, yaitu kiri dan kanan.
4. **Panggilan Rekursif**: Fungsi kemudian memanggil dirinya sendiri secara rekursif untuk mengurutkan kedua bagian singly linked list, kiri dan kanan, dengan urutan yang sama (ascending atau descending).
5. **Penggabungan Kembali**: Setelah kedua bagian singly linked list terurut, fungsi menggunakan fungsi merge atau mergeDesc (sesuai dengan kondisi ascending) untuk menggabungkan kedua bagian tersebut kembali ke dalam singly linked list utama. Hasil penggabungan ini diarahkan ke headRef, sehingga menghasilkan singly linked list yang terurut berdasarkan tenggat dengan urutan yang diinginkan (ascending atau descending).

Dengan langkah-langkah di atas, fungsi mergeSort berhasil mengimplementasikan algoritma merge sort secara rekursif untuk mengurutkan tugas dalam singly linked list sesuai dengan urutan yang diinginkan.

* **Fungsi binarySearch()**

Node \*binarySearch(char \*nama) {

Node \*kiri = head, \*kanan = NULL;

mergeSort(&kiri, 1);

Node\* tengah;

Node\* tengahSebelumnya = NULL;

while (kiri != kanan) {

tengah = kiri;

tengahSebelumnya = NULL;

int length = 0;

Node\* temp = kiri;

while (temp != kanan) {

length++;

temp = temp->next;

}

int tengahPos = length / 2;

for (int i = 0; i < tengahPos; i++) {

tengahSebelumnya = tengah;

tengah = tengah->next;

}

if (strcmp(tengah->tugas.nama, nama) == 0)

return tengah;

else if (strcmp(tengah->tugas.nama, nama) > 0)

kanan = tengah;

else

kiri = tengah->next;

}

return NULL;

}

Fungsi binarySearch adalah implementasi algoritma binary search untuk mencari tugas dalam singly linked list yang diurutkan berdasarkan nama. Berikut adalah penjelasan lebih detail tentang langkah-langkah yang dilakukan oleh fungsi ini:

1. **Inisialisasi Variabel**: Fungsi menginisialisasi variabel lokal kiri dan kanan dengan pointer head dan NULL, serta melakukan pengurutan ascending pada singly linked list menggunakan fungsi mergeSort.
2. **Inisialisasi Variabel Tambahan**: Fungsi juga menginisialisasi variabel lokal tengah, tengahSebelumnya, length, dan tengahPos yang diperlukan untuk melakukan binary search.
3. **Binary Search**: Fungsi menggunakan pendekatan binary search dengan melakukan iterasi dan perbandingan di antara elemen-elemen di tengah singly linked list (tengah) dan menyesuaikan kiri dan kanan berdasarkan perbandingan.

a. Jika tengah->tugas.nama sama dengan nama, maka fungsi mengembalikan node tengah yang sesuai.

b. Jika tengah->tugas.nama lebih besar dari nama, maka kanan diatur menjadi tengah untuk mencari di bagian kiri dari tengah.

c. Jika tengah->tugas.nama lebih kecil dari nama, maka kiri diatur menjadi tengah->next untuk mencari di bagian kanan dari tengah.

1. **Iterasi Binary Search**: Proses ini berlanjut hingga kiri dan kanan berhimpit, menunjukkan bahwa pencarian telah selesai.
2. **Pengembalian Hasil**: Jika tugas dengan nama yang dicari ditemukan, fungsi mengembalikan node yang sesuai. Jika tidak, fungsi mengembalikan NULL untuk menandakan bahwa tugas tidak ditemukan.

Dengan langkah-langkah di atas, fungsi binarySearch dapat secara efisien mencari tugas dalam singly linked list yang telah diurutkan secara ascending berdasarkan nama, menggunakan algoritma binary search.

* **Fungsi menu()**

void menu() {

printf("Manajemen Pengumpulan Tugas\n");

printf("1. Tambah Tugas\n");

printf("2. Edit Tugas\n");

printf("3. Hapus Tugas\n");

printf("4. Tampilkan Semua Tugas\n");

printf("5. Urutkan Tugas Berdasarkan Tenggat Pengumpulan (Naik)\n");

printf("6. Urutkan Tugas Berdasarkan Tenggat Pengumpulan (Turun)\n");

printf("7. Cari Tugas Berdasarkan Nama\n");

printf("8. Keluar\n");

}

Fungsi menu menampilkan opsi-opsi yang tersedia dalam aplikasi manajemen pengumpulan tugas.

* **Fungsi ambilInput()**

void ambilInput(char \*prompt, char \*input, int panjang) {

printf("%s", prompt);

fgets(input, panjang, stdin);

input[strcspn(input, "\n")] = 0; // menghapus karakter newline

}

Fungsi ambilInput digunakan untuk mengambil input dari pengguna dengan beberapa langkah sebagai berikut:

1. **Menampilkan Prompt**: Fungsi ini mulai dengan menampilkan pesan prompt kepada pengguna menggunakan printf("%s", prompt);.
2. **Membaca Input**: Setelah pesan prompt ditampilkan, fungsi menggunakan fgets(input, panjang, stdin); untuk membaca input dari pengguna. Input tersebut disimpan dalam variabel input.
3. **Menghapus Newline**: Kemudian, fungsi menggunakan input[strcspn(input, "\n")] = 0; untuk menghapus karakter newline (\n) dari akhir input. Hal ini dilakukan agar input tidak menyisakan newline yang tidak diinginkan di akhir string.

Dengan langkah-langkah di atas, fungsi ambilInput dapat mengambil input dari pengguna dengan aman dan menghilangkan karakter newline untuk keperluan pengolahan selanjutnya.

* **Fungsi main()**

int main() {

system("cls");

printf("Nama : Carel Habsian Osagi\n");

printf("NIM : 23343061\n");

printf("Prodi: S1 Informatika (NK)\n\n");

int pilihan;

char nama[100], tenggat[20], namaBaru[100], tenggatBaru[20];

while (1) {

menu();

printf("Pilih opsi: ");

scanf("%d", &pilihan);

getchar(); // membersihkan buffer setelah input pilihan

switch (pilihan) {

case 1:

do {

ambilInput("Masukkan nama tugas (-1 untuk selesai): ", nama, 100);

if (strcmp(nama, "-1") != 0) {

ambilInput("Masukkan tenggat tugas (YYYY-MM-DD): ", tenggat, 20);

tambahTugas(nama, tenggat);

}

} while (strcmp(nama, "-1") != 0);

break; case 2:

ambilInput("Masukkan nama tugas yang ingin diubah: ", nama, 100);

Node \*nodeDiedit = head;

while (nodeDiedit != NULL && strcmp(nodeDiedit->tugas.nama, nama) != 0) {

nodeDiedit = nodeDiedit->next;

}

if (nodeDiedit == NULL) {

printf("Tugas tidak ditemukan.\n");

} else {

ambilInput("Masukkan nama tugas baru: ", namaBaru, 100);

ambilInput("Masukkan tenggat tugas baru (YYYY-MM-DD): ", tenggatBaru, 20);

editTugas(nama, namaBaru, tenggatBaru);

}

break;

case 3:

ambilInput("Masukkan nama tugas yang ingin dihapus: ", nama, 100);

hapusTugas(nama);

break;

case 4:

printf("Daftar tugas:\n");

tampilkanTugas();

break;

case 5:

mergeSort(&head, 1);

printf("Tugas telah diurutkan berdasarkan tenggat (Naik).\n");

break;

case 6:

mergeSort(&head, 0);

printf("Tugas telah diurutkan berdasarkan tenggat (Turun).\n");

break;

case 7:

ambilInput("Masukkan nama tugas yang dicari: ", nama, 100);

Node \*hasil = binarySearch(nama);

if (hasil != NULL)

printf("Tugas ditemukan: Nama: %s, tenggat: %s\n", hasil->tugas.nama, hasil->tugas.tenggat);

else

printf("Tugas tidak ditemukan.\n");

break;

case 8:

exit(0);

default:

printf("Pilihan tidak valid.\n");

}

}

return 0;

}

Fungsi main merupakan titik masuk utama program yang menjalankan berbagai fitur dalam manajemen tugas. Berikut penjelasan lebih rinci namun ringkasnya:

1. **Pengaturan Awal**: Fungsi main dimulai dengan membersihkan layar menggunakan system("cls"); dan menampilkan informasi pengguna seperti nama, NIM, dan program studi.
2. **Loop Utama**: Program berjalan dalam loop utama while (1) yang terus berjalan hingga pengguna memilih untuk keluar.
3. **Menu dan Input**: Setiap iterasi loop, program menampilkan menu dan meminta input pilihan dari pengguna. Input diproses menggunakan fungsi ambilInput untuk mengambil nama tugas, tenggat, atau pilihan lainnya.
4. **Switch Case**: Program menggunakan switch untuk memilih tindakan sesuai dengan pilihan pengguna. Pilihan tersebut antara lain adalah menambah, mengedit, menghapus, menampilkan, mengurutkan, atau mencari tugas.
5. **Pemrosesan Tugas**: Setiap aksi yang dipilih oleh pengguna dijalankan melalui pemanggilan fungsi yang sesuai seperti tambahTugas, editTugas, hapusTugas, tampilkanTugas, mergeSort, dan binarySearch.
6. **Keluar dari Program**: Pengguna dapat keluar dari program dengan memilih opsi keluar (case 8) yang menghentikan loop utama dan mengakhiri program.

Dengan struktur dan fitur-fitur tersebut, program main mengatur alur kerja program manajemen tugas secara efisien dan interaktif.

1. **Kesimpulan**

Program ini mengimplementasikan:

* **Singly linked list** untuk menyimpan daftar tugas.
* **Merge Sort** untuk mengurutkan daftar tugas berdasarkan tenggat waktu.
* **Binary Search** untuk mencari tugas berdasarkan nama setelah daftar diurutkan.
* **Manajemen Tugas** seperti menambah, mengedit, menghapus, dan menampilkan semua tugas.