**LAPORAN**

**STRUKTUR DATA ALGORITMA**



**Disusun Oleh :**

1. M. Iqbal Zafarullah (G1A024007)

2. Nurmelizah (G1A024023)

3. Ali Akbar Bermano (G1A024041)

4. Anjelia Febiliyanti (G1A024053)

5. Ikhsan Rihan Farli (G1A024083)

**Dosen Pengampu :**

1. Arie Vatresia , S.T., M.TI, Ph.D.

2. Kurnia Anggraini, S.T., M.T., Ph.D.

**PROGRAM STUDI INFORMATIKA**

**FAKULTAS TEKNIK**

**UNIVERSITAS BENGKULU**

**2025**

**LANDASAN TEORI**

Algoritma merupakan fondasi yang harus dipahami atau dikuasai oleh seseorang yang akan menyelesaikan suatu masalah dengan komputer, dalam hal ini dengan membuat program. Algoritma berasal dari kata al-kwarizmi yang terdapat di buku Abu Ja'far Muhammad Ibnu Musa Al-Kwarizmi, seorang ahli Matematika dari Persia dengan judul bukunya "Aljabar wal Muqabala". Dalam beberapa buku terdapat beberapa definisi algoritma. Tetapi kalau kita cermati dengan baik, buku-buku tersebut mempunyai tujuan yang sama. Dalam buku ini, definisi Algoritma adalah susunan langkah penyelesaian suatu masalah secara sistematika dan logis. Terdapat dua kata yang menjadi perhatian dalam definisi ini, yaitu sistematis dan logis. Membangun sebuah program pada dasarnya adalah membuat alat bantu untuk menyelesaikan suatu masalah. Sebelum kita dapat menghasilkan program yang mampu membantu kita menyelesaikan masalah tersebut, kita dihadapkan pada 3 (tiga) tahapan pokok, yaitu memahami permasalahan, mampu menyusun rancangan dari masalah, dan mampu mengimplementasikan hasil.

Memahami permasalahan dan tujuan sebuah program dibuat. Pada tahap ini kita harus mampu mengidentifikasi jenis, bentuk dan karakteristik input serta output yang diharapkan. Tetapi untuk skala permasalahan yang besar, selain jenis, bentuk dan karakteristik, kita juga perlu mengetahui dengan pasti asal, frekuensi dan volume data input serta tujuan, frekuensi dan volume data output yang diharapkan. Mampu menyusun konsep/rancangan/desain penyelesain dari masalah yang akan kita selesaikan. Dari hasil pemahaman kita terhadap permasalahan di atas, kita harus mampu merancang sebuah alur proses untuk mengolah data input dan menghasilkan data output dengan jenis, bentuk dan karakteristik seperti yang diharapkan. Mampu mengimplementasikan hasil rancangan kita dalam bentuk program yang terstruktur. Program tersebut dapat kita buat dengan menggunakan sembarang bahasa pemrograman. Untuk itulah kita diharuskan memahami dan menguasai komponen bahasa pemrograman dan teknik pemrograman dengan baik. Pemahaman terhadap algoritma sangat berpengaruh terhadap kemampuan sesorang dalam membuat program.

Dalam pemrograman konsep algoritma tidak lepas dari struktur data, struktur data sangat berkaitan dengan algoritma. Pada awalnya struktur data adalah bagian dari algoritma kenapa bisa dibilang seperti itu karena berdasarkan beberapa petikan defenisi, bahwa algoritma merupakan urutan langkah yang logis dan mengikuti suatu urutan tertentu dan tidak boleh melompat-lompat. Struktur data merupakan cara menyimpan, pengorganisasian, dan pengurutan data didalam komputer agar bisa dipakai secara efisien. Sedangkan data adalah fakta mentah yang belum memiliki arti. Data adalah fakta yang dikumpulkan, disimpan dan diproses oleh sistem informasi Data adalah fakta atau obervasi mentah yang biasanya mengenai fenomena fisik atau transaksi data. Ada dua jenis struktur data yaitu statis dan dinamis. Struktur data statis adalah struktur data yang tidak berubah (*array, record,* himpunan), struktur data dinamis adalah Struktur data yang berubah (*queue, stack, list*). Dengan pemilihan struktur data yang baik maka problem yang kompleks dapat diselesaikan dengan algoritma yang dapat digunakan secara efisien, operasi operasi penting dapat dieksekusi dengan sumber daya yang lebih kecil, memori kecil dan wktu eksekusi yang lebih cepat.

Struktur data digunakan untuk meningkatkan efisiensi penggunaan memori pada saat program komputer sedang bekerja. Penggunaan struktur data yang tepat pada pemrograman dapat membuat algoritma menjadi lebih mudah, kemudahan ini membuat program lebih efisien dan sederhana. Meningkatkan efisien merupakan tujuan utama pengaplikasian struktur data. Dengan struktur data, proses reservasi memori yang tidak perlu akan diminimalisasi. Selain itu struktur data juga menjamin kemudahan pemahaman algortima, sehingga untuk menyelesaikan permasalahan seperti perkalian matriks, visualisasi matriks dan tabel akan menjadi lebih mudah.

Seorang programmer memerlukan teknik dalam menyelesaikan suatu masalah. Teknik-teknik tersebut yaitu teknik *TOP-Down* dan *Bottom-Up*. Teknik *Top-Down* bekerja dengan cara membagi suatu permasalahan yang kompleks ke dalam tingkatan-tingkatan masalah hingga yang paling kecil. Setiap masalah harus memiliki tanggung jawab yang jejelas.

Teknik *Top-Down* memiliki keteraturan yang dapat memudahkan perencanaan dan dalam mengorgranisir suatu pekerjaan, memungkinkan programmer terfokus akan gambaran besar sebelum ke detail yang lebih spesifik, dapat membantu memahamihubungan antar komponen yang berbeda dalam sistem dan mempermudah identifikasi dan pengelolaan kopleksitas proyek dengan membaginya menjadi bagian-bagian yang kecil. Meskipun demikian, teknik *Top-down* ini memiliki beberapa kekurangan yaitu fleksibilitas yang terbatas dalam mengakomodasi perubahan yang besar di tahap akhir, *overhead* karena memerlukan waktu dan usaha untuk mendesain dan mendokumentasikan dari atas hingga ke bawah sebelum kode di implementasikan dan memiliki ketergantungan terhadap kualitas analisis dan perencanaan awal.

Sedangkan Teknik *Bottom-Up* memiliki beberapa kelebihan yaitu dapat digunakan kembali pada proyek yang lain dengan cara mendorong pengembangan modul-modul independen, pada modul kecil dapat memungkinkan terjadinya perubahan penyesuaian yang lebih mudah tanpa mempengaruhi keseluruhan sistem, dapat mengidentifikasi dan memperbaiki masalah dengan cepat dandapat menghasilkan komponen-komponen yang digunakan kembali dalam berbagai konteks. Sedangkan kekurangan Teknik *Bottom-Up* ini yaitu kurang fokus terhadap gambaran besar, diperlukan banyak pengujian dan penyesuaian dan resiko duplikasi atau tumpang tindih.

Quicksort merupakan Algoritma Sortingyang dikembangkan oleh C.A.R Hoare pada tahun1960 yang secara kasus rata-rata, membuat pengurutan O(n logn) untuk mengurutkan item. Algoritma ini juga dikenal sebagai Partition-Exchange Sortatau disebut sebagai Sorting pergantian pembagi. Pada kasus terburuknya. algoritma ini membuat perbandingan O(n2), walaupun kejadian seperti ini sangat langka. Quick sort sering lebih cepat dalam praktiknya daripada algoritma O(n log n) yang lainnya. Dan juga, urutan dan referensi lokalisasi memori quicksort bekerja lebih baik dengan menggunakan cache CPU, jadi keseluruhan sortingdapat dilakukan hanya dengan ruang tambahan O(log n) [4]. Pada masalah pengurutan data bilangan bulat (*integer*) secara terindeks pada suatu list atau array dari bilangan yang paling besar sampai ke bilangan yang paling kecil atau sebaliknya. Tidak hanya dapat diterapkan pada pengindeksan bilangan saja, namun juga untuk pengindeksan huruf (abjad) dari A ke Z atau sebaliknya. Algoritma ini sangat baik diterapkan pada kasus pengindeksan kumpulan kata (library sort utility) atau kumpulan bilangan atau kombinasinya.

Counting sort adalah algoritma pengurutan waktu linier yang digunakan untuk mengurutkan item saat item tersebut termasuk dalam himpunan tetap dan terbatas. Counting sort [3] adalah algoritma efisien yang mengasumsikan bahwa semua elemen yang akan diurutkan bertipe integer dalam rentang 1 hingga k, di mana k adalah integer lainnya. Algoritma ini mengasumsikan bahwa setiap elemen adalah integer dalam rentang 1 hingga k, untuk beberapa integer k. Saat k = O(n), Counting-sort berjalan dalam waktu O(n). Ide dasar dari counting sort adalah untuk menentukan, untuk setiap elemen input x, jumlah elemen yang kurang dari x. Informasi ini dapat digunakan untuk menempatkan elemen secara langsung pada posisi yang benar.

Heap adalah struktur data berbentuk pohon biner yang memiliki sifat khusus, yaitu setiap node induk memiliki nilai yang lebih besar (untuk max heap) atau lebih kecil (untuk min heap) dari nilai-nilai anaknya. Binary heap adalah sebuah heap yang mana setiap node memiliki 2 children Proses membagi biner heap menjadi dua jenis: max heap dan min-heap. Dalam max-heap, setiap simpul memiliki nilai yang lebih besar atau sama dengan nilai anak-anaknya. Artinya, nilai terbesar berada di simpul akar (root). Sebaliknya, dalam min-heap, setiap simpul memiliki nilai yang lebih kecil atau sama dengan nilai anak-anaknya. Artinya, nilai terkecil berada di simpul akar.

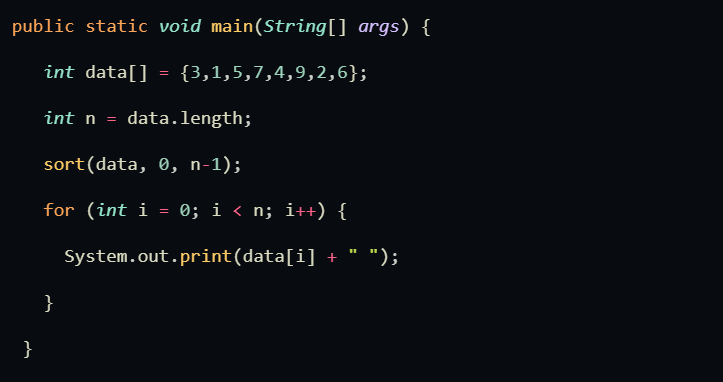
HeapSort adalah salah satu algoritma pengurutan yang berbasis pada struktur data heap, khususnya max heap atau min heap. Algoritma ini bekerja dengan membangun heap dari data yang diberikan, kemudian mengambil elemen maksimum (untuk max heap) atau minimum (untuk min heap) satu per satu dan menempatkannya di posisi yang benar dalam array. Kosnep binary heap dan heap sort juga sering di hubungkan dengan konsep priority queue.

**SOAL DAN PEMBAHASAN**

1. Quick Sort

1. Main method.

Print screen source code:



Gambar 1.1 Main method

Source code:

 public static *void* main(*String*[] *args*) {

*int* data[] = {3,1,5,7,4,9,2,6};

*int* n = data.length;

    sort(data, 0, n-1);

    for (*int* i = 0; i < n; i++) {

      System.out.print(data[i] + " ");

    }

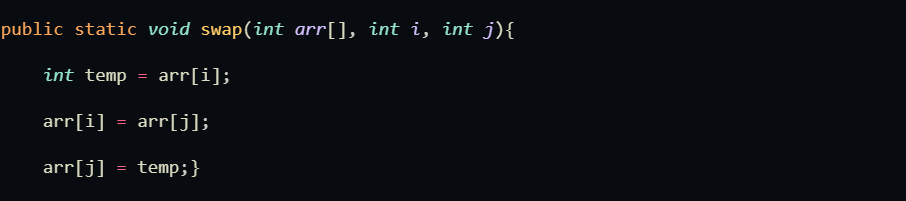
  }

Penjelasan souce code:

Method utama ini berisi pendeklarasian variabel data yang merupakan array dengan nilai {3,1,5,7,4,9,2,6}, lalu terdapat juga variabel n dengan nilai 8 yang didapat dari data.length. terdapat juga perulangan for untuk mencetak nilai dari variabel data yang sudah tersusun.

1. Method untuk merubah urutan array.

Print Screen Source code:



Gambar 1.2 Method Swap

Source code:

public static *void* swap(*int* *arr*[], *int* *i*, *int* *j*){

*int* temp = arr[i];

    arr[i] = arr[j];

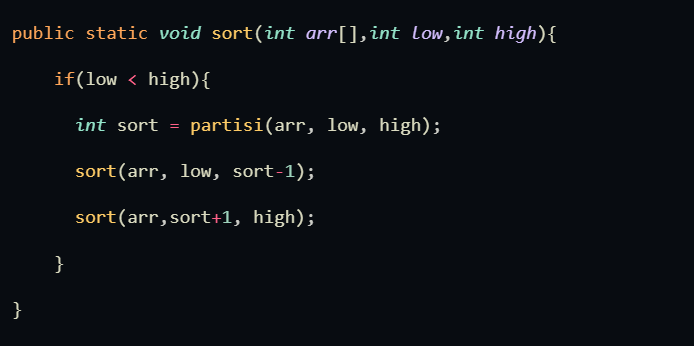
    arr[j] = temp;}

Penjelasan Source code:

Method ini merupakan unsur terpenting dalam sebuah algoritma sorting, karena algoritma sorting memerlukan sebuah metode untuk memindahkan data dari salah satu indeks array ke indeks lainnya. Method ini memiliki parameter *int* *arr*[] yang merupakan array yang ingin kita ubah susunannya, *int* *i*, dan *int*. *j* sebagai indeks dari array tersebut. Method ini bekerja dengan memasukkan data dalam array dengan indeks i, kedalam variabel sementara yang diberi nama temp, lalu array dengan indeks i akan mengambil data dari array indeks j, dan terakhir array indeks j akan mengambil data indeks i yang sudah dipindahkan kedalam variabel temp sebelumnya.

1. Method sort

Print screen source code:



Gambar 1.3 Method sort

Soruce code:

public static *void* sort(*int* *arr*[],*int* *low*,*int* *high*){

    if(low < high){

*int* sort = partisi(arr, low, high);

      sort(arr, low, sort-1);

      sort(arr,sort+1, high);

    }

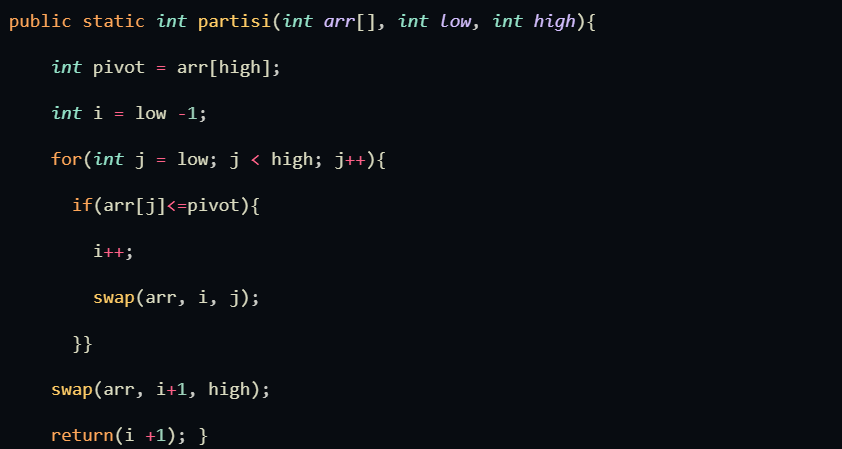
}

Penjelasan Source code:

Method ini tidak mengembalikan nilai dan memiliki parameter *int* *arr*[],*int* *low*,*int* *high.* Pada method ini terdapat conditional if dengan kondisi low < high, yang berarti method ini akan berjalan ketika nilai variabel low kurang dari high. Di dalam conditional tersebut, akan dideklarasikan variabel sort dengan tipe data integer, yang nilainya merupakan pivot (diambil dari method partisi). Setelah itu, masih di dalam if akan dideklarasikan pengunaan method sort sort(arr, low, sort-1) untuk menyusun data yang berada di sebelak kiri pivot dan sort(arr,sort+1, high) untuk menyusun data yang berada di sebelah kanan pivot.

1. Method untuk menentukan partisi pada quick sort.

Print screen source code:



Gambar 1.4 Method Partisi

Source code:

public static *int* partisi(*int* *arr*[], *int* *low*, *int* *high*){

*int* pivot = arr[high];

*int* i = low -1;

    for(*int* j = low; j < high; j++){

      if(arr[j]<=pivot){

        i++;

        swap(arr, i, j);

      }}

    swap(arr, i+1, high);

    return(i +1); }

Penjelasan source code:

Method ini digunakan untuk membagi array menjadi 2 bagian dengan memilih salah satu nilai pada suatu indeks sebagai pivot. Method ini memiliki return value berupa int, dengan parameter *int* *arr*[], *int* *low*, *int* *high.* Pada method ini akan dibuat sebuah variabel bernama pivot dengan tipe data integer yang akan mengambil nilai dari array pada indeks high. Method ini juga akan mendeklasarikan variabel i dengan dengan nilai low-1. Selanjutnya, akan dimulai sebuah perulangan for, yang yang mana perulangan akan terus terjadi ketika variabel j yagn bernilai low itu kuran dari high, dan di setiap iterasi nilai variabel j akan ditambah/increment. Di dalam perulangan tersebut akan terdapat kondisional if dengan kondisi arr[j]<=pivot yang berarti ketika nilai dari array dengan indeks j itu kurang dari atau sama dengan pivot, maka variabel i atan ditambah/increment, dan method swap akan dijalankan untuk merubah posisi array dengan indeks j dengan array dengan indeks i. untuk merubah posisi pivot yang mana tetap barada pada indeks terakhir, setelah perulangan selesai, maka jalankan method swap dengan parameter arr, i+1, high. Method ini akan mengembalikan nilai i+1 yang mana merupakan indeks tempat pivot berada.

2. Counting Sort

a. Main method.

Print screen source code:



Gambar 2.1 Main method

Source code:

 int main(){

    Mahasiswa mhs[] = {

        {"G1A024001", "LUDVIE", 87},

        {"G1A024002", "AHSAN", 98},

        {"G1A024003", "AGRA", 89},

        {"G1A024004", "AZKA", 78},

        {"G1A024005", "FEBRIYAN", 90},

        {"G1A024006", "FADLI", 88},

        {"G1A024007", "IQBAL", 91},

        {"G1A024008", "SULTHON", 95},

        {"G1A024OO9", "RAYYAN", 93},

        {"G1A024010", "RAFI", 85}

    };

    int n = sizeof(mhs)/sizeof(mhs[0]);

    countingSort(mhs, n);

    cout << "\nData berdasarkan nilai terkecil sampai terbesar:\n";

    print(mhs, n);

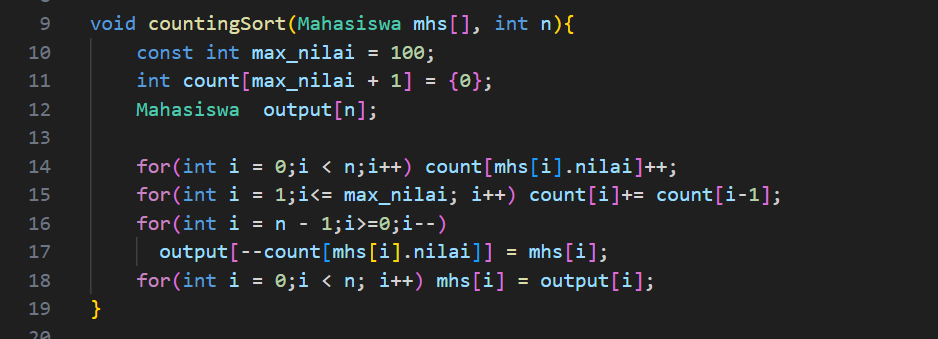
    return 0;

Penjelasan souce code:

Pada method main, dibuat array mhs bertipe Mahasiswa yang berisi 10 data, masing-masing berisi npm, nama, dan nilai. Variabel n digunakan untuk menghitung jumlah elemen dengan rumus sizeof(mhs)/sizeof(mhs[0]). Setelah itu, fungsi countingSort(mhs, n) dipanggil untuk mengurutkan mahasiswa berdasarkan nilai dari kecil ke besar. Hasil pengurutan kemudian ditampilkan menggunakan fungsi print(mhs, n), dan program diakhiri dengan return 0.

1. Method countingSort

Print Screen Source code:



Gambar 2.2 Method Counting Sort

Source code:

void countingSort(Mahasiswa mhs[], int n){

    const int max\_nilai = 100;

    int count[max\_nilai + 1] = {0};

    Mahasiswa  output[n];

    for(int i = 0;i < n;i++) count[mhs[i].nilai]++;

    for(int i = 1;i<= max\_nilai; i++) count[i]+= count[i-1];

    for(int i = n - 1;i>=0;i--)

      output[--count[mhs[i].nilai]] = mhs[i];

    for(int i = 0;i < n; i++) mhs[i] = output[i];

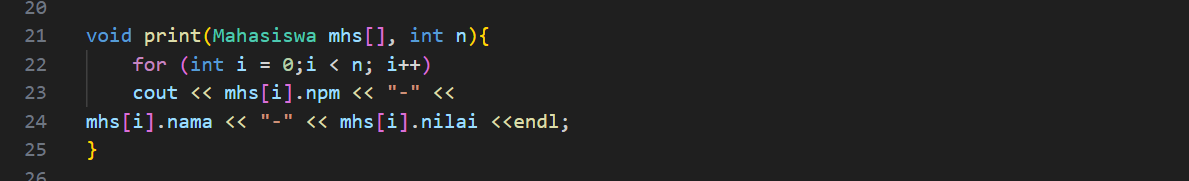
}

Penjelasan Source code:

Fungsi countingSort mengurutkan data mahasiswa menggunakan algoritma Counting Sort. Diawali dengan menentukan nilai maksimum (max\_nilai) sebesar 100, kemudian dibuat array count untuk mencatat jumlah kemunculan nilai dan array output untuk menyimpan hasil urutan. Setelah semua data mahasiswa ditempatkan sesuai urutan, hasilnya disalin kembali ke array mhs.

1. Method Print

Print screen source code:



Gambar 2.3 Method Print

Soruce code:

void print(Mahasiswa mhs[], int n){

    for (int i = 0;i < n; i++)

    cout << mhs[i].npm << "-" <<

mhs[i].nama << "-" << mhs[i].nilai <<endl;

}

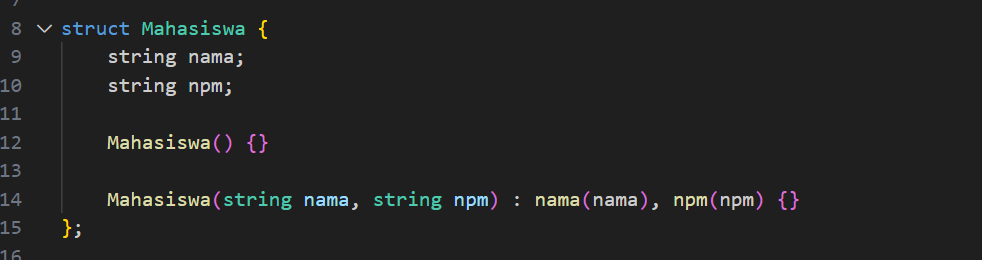
Penjelasan Source code:

Method print berfungsi untuk menampilkan seluruh data mahasiswa yang telah diurutkan. Di dalam method ini digunakan perulangan for untuk mengakses setiap elemen array mhs, dan pada setiap iterasi, program akan mencetak data mahasiswa dalam format npm-nama-nilai, kemudian dilanjutkan dengan mencetak baris baru. Dengan cara ini, seluruh data mahasiswa yang sudah tersusun berdasarkan nilai dapat ditampilkan dengan rapi ke layar.

3. Binary heaps

a. Struct Mahasiswa

Print screen source code:



Gambar 3.1 Struct mahasiswa

Source code:

 struct Mahasiswa {

string nama;

string npm;

Mahasiswa() {}

Mahasiswa(string nama, string npm) : nama(nama), npm(npm) {}

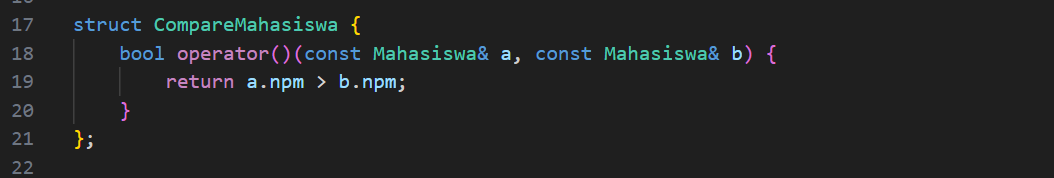
};

Penjelasan souce code:

Struct Mahasiswa berfungsi untuk membuat tipe data baru yang berisi dua atribut, yaitu nama dan npm, keduanya bertipe string. Struct ini punya dua constructor: satu constructor kosong untuk membuat objek tanpa data, dan satu lagi untuk langsung mengisi nama dan npm saat objek dibuat. Dengan struct ini, data mahasiswa bisa dikelola lebih rapi dan terstruktur.

1. Struct CompareMahasiswa

Print Screen Source code:



Gambar 3.2 Struct CompareMahasiswa

Source code:

void countingSort(Mahasiswa mhs[], int n){

    const int max\_nilai = 100;

    int count[max\_nilai + 1] = {0};

    Mahasiswa  output[n];

    for(int i = 0;i < n;i++) count[mhs[i].nilai]++;

    for(int i = 1;i<= max\_nilai; i++) count[i]+= count[i-1];

    for(int i = n - 1;i>=0;i--)

      output[--count[mhs[i].nilai]] = mhs[i];

    for(int i = 0;i < n; i++) mhs[i] = output[i];

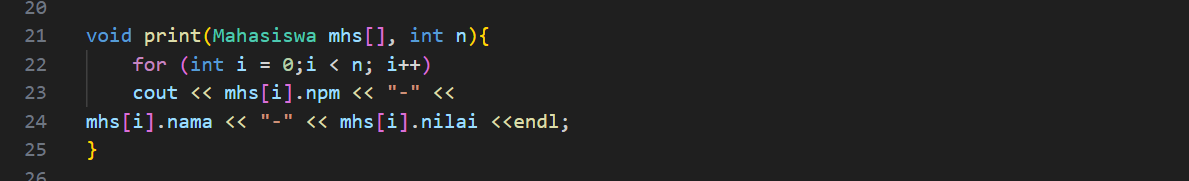
}

Penjelasan Source code:

Struct CompareMahasiswa dipakai untuk mengatur cara membandingkan dua Mahasiswa berdasarkan npm. Di dalamnya, operator () dioverride supaya mengembalikan true kalau npm objek pertama lebih besar dari yang kedua. Dengan aturan ini, priority queue berubah jadi min-heap, sehingga Mahasiswa dengan npm terkecil selalu ada di urutan atas dan keluar lebih dulu..

1. Main method

Print screen source code:



Gambar 3.3 Main method

Soruce code:

int main() {

priority\_queue<Mahasiswa, vector<Mahasiswa>, CompareMahasiswa> heap;

heap.push(Mahasiswa("Anjelia Febiliyanti", "G1A024053"));

heap.push(Mahasiswa("Nurmelizah", "G1A024023"));

heap.push(Mahasiswa("Ikhsan Rihan Farli", "G1A024083"));

heap.push(Mahasiswa("M. Iqbal Zafarullah", "G1A024007"));

heap.push(Mahasiswa("Ali Akbar Bermano", "G1A024041"));

cout << "Mahasiswa berdasarkan NPM terkecil:\n";

while (!heap.empty()) {

Mahasiswa mhs = heap.top();

cout << mhs.nama << " (" << mhs.npm << ")" << endl;

heap.pop();

}

return 0;

}

Penjelasan Source code:

Selain itu, dalam method main juga dibuat priority\_queue bernama heap yang menyimpan objek Mahasiswa berdasarkan aturan CompareMahasiswa. Lima data mahasiswa dimasukkan ke dalam heap. Melalui perulangan while, data dengan npm terkecil diambil dan ditampilkan satu per satu menggunakan heap.top(), lalu dihapus dengan pop.

**DAFTAR PUSTAKA**

Asih, V., Saputra, A., & Subagio, R. T. (2020). Penerapan Algoritma Fisher Yates Shuffle Untuk Aplikasi Ujian Berbasis Android. Jurnal Digit: Digital of Information Technology, 10(1), 59-70.

Pradita Eko Prasetyo Utomo, E. E. (2024). konsep struktur data. Padang: CV. gita lente