UNIVERSITAS ISLAM NEGERI
ALAUDDIN
MAKASSAR

STRUKTUR DATA

MODUL PRAKTIKUM

Prodi Sistem Informasi
Fakultas Sains & Teknologi
UIN Alauddin Makassar
2019

Kartu Kontrol

Nama Mahasiswa : ……..

NIM : ……..

Pertemuan	Hari/Tanggal	Paraf Dosen / Asisten			
		Tugas Pendahuluan	Respon	Kehadiran	Laporan Praktikum
1	/				
2	/				
3	/				
4	/				
5	/				
6	/				
7	/				
8	/				
9	/				
10	/				

DAFTAR ISI

Table of Contents

Praktikum Struktur Data

Nama Dosen: Rahman, S. Kom., M.T.

Pendahuluan

Struktur data adalah pendekatan pengorganisasian data secara digital untuk keperluan pemrosesan pada komputer. Data yang terorganisir memiliki banyak manfaat diantaranya memudahkan proses temu kembali data, penyimpanan hingga meningkatkan efisiensi komputasi. Sebaliknya data yang tidak didesain secara baik akan sulit diproses bahkan bisa jadi menimbulkan inkonsistensi data seperti duplikasi data atau data hilang.

Struktur data berarti menyusun informasi dengan teknik tertentu yang mempertimbangkan cara akses elemen data, cara penyimpanan dan reorganisasi data saat terjadi operasi penghapusan atau penambahan elemen data.

Pendekatan struktur data paling awal dimulai dari pemilihan tipe data. Struktur data yang baik seharusnya menggunakan tipe data yang representative dengan sifat informasi data tersebut. Data numeric mesti direperesentasikan oleh tipe numeric, demikian juga numeric bilangan bulat sebaiknya dideklarasikan sebagai bilangan bulat. Demikian pula halnya jika tipe data yang diperlukan adalah tipe data bentukan atau ADT maka perancang ADT harus benar memperhatikan konstruksi ADT dengan kandungan informasinya.

Pada modul praktikum ini, mahasiswa akan mempelajari teknik mengorganisasi data menggunakan beberapa model struktur data yang sudah umum digunakan seperti Linked List, Stack, Queue, Tree dan lainnya. Praktikan akan mempelajari bagaimana elemen-elemen data dan relasinya dihubungkan secara logis menurut jenis-jenis struktur data yang dipelajari. Mahasiswa praktikan diharapkan bisa menganalisa dan memecahkan kasus dunia ril dengan menggunakan pendekatan struktur data.

Deskripsi Praktikum

Pertemuan 1: Tipe Data, Struct dan Abstract Data Type
Pertemuan 2: Array dan Matriks Data
Pertemuan 3: Linked List
Pertemuan 4: Stack, Queue dan Deque
Pertemuan 5: List dan Iterator
Pertemuan 6: Tree
Pertemuan 7: Heaps dan Priority Queue
Pertemuan 8: Hash Table
Pertemuan 9: Map dan Skip List
Pertemuan 10: Tries dan Graph

Modul Praktikum 1: Tipe Data, Struct dan Abstract Data Type

Deskripsi Pertemuan

Modul ini akan merefresh pengetahuan tentang tipe data dasar yang telah diperkenalkan pada mata kuliah permograman dasar, dan sekaligus akan memperkenalkan tipe data stuktur dan tipe data UDT (User Defined Type) dan ADT (Abstract Data Type) yaitu tipe data custom yang didefiniskan pengguna.

Syarat Kompetensi

1. Memahami dan mampu membuat program dalam bahasa C++.
2. Memahami dan mampu menggunakan tipe data primitif C++
3. Mampu menganalisa dan melakukan debug program.

Standar Kompetensi

1. Mahasiswa mampu mengimplementasikan Tipe data struktur UDT dan ADT dalam program.
2. Mahasiswa mampu menganalisis kasus dan merancang algoritma penyelesaian kasus menggunakan UDT dan ADT.

Dasar Teori

Setiap bahasa pemrograman telah menyediakan tipe data primitif untuk merepresentasikan data atau informasi yang akan diproses. Misalnya data nama dapat menggunakan array char atau string. Demikian juga data numerik dapat menggunakan tipe data integer atau double. Namun bagaimana jika programmer ingin merepresentasikan suatu struktur data?

Pada dasarnya struktur data itu adalah koleksi data-data primitif yang disatukan dalam satu bentuk data karena faktor keterhubungan data-data tersebut. Sebagai contoh data suatu objek atau entitas mahasiswa seperti nama, nim, alamat, umur, angkatan adalah

data-data yang saling berkaitan yang terikat pada satu identitas mahasiswa. Dalam pemrograman kita bisa menyatakan data-data tersebut sebagai bagian data-data primitif terpisah. Misalnya pada program berikut ini, kita bisa menuliskan data 2 entitas mahasiswa dengan memberi penomoran pada variabel:

```
#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

int main(){

	/*
		Deklarasi mahasiswa 1
		dengan memberi penomoran pada variabel
	*/
	string nama_1 = "Arifudding";
	char nim_1[10] = "080940101";
	char jenisKelamin_1 = 'L';
	int umur_1 = 19;
	int angkatan_1 = 2019;

	/*
		Deklarasi mahasiswa 2
		dengan memberi penomoran berbeda pada variabel
	*/
	string nama_2 = "Salma Roshidin";
	char nim_2[10] = "080940102";
	char jenisKelamin_2 = 'P';
	int umur_2 = 19;
	int angkatan_2 = 2019;

	cout << "Berikut identitas mahasiswa 2 : \n"
		 << setw(16) << left << "Nama " << ": "
		 << nama_2 << endl
		 << setw(16) << left << "NIM " << ": "
		 << nim_2 << endl
		 << setw(16) << left << "Jenis Kelamin " << ": "
		 << jenisKelamin_2 << endl
		 << setw(16) << left << "Umur " << ": "
		 << umur_2 << endl
		 << setw(16) << left << "Angkatan " << ": "
		 << angkatan_2 << endl;

	system("pause");
	return 0;
}
```

Cara deklarasi variabel untuk menampung informasi tentang 2 mahasiswa berbeda pada program di atas akan berjalan dengan baik dan tidak menimbulkan masalah berarti untuk mengelolah 2 data. Akan tetapi bertambahnya jumlah mahasiswa yang mesti dimasukan kedalam program akan membuat kesulitan baru khususnya dalam menuliskan variabel berulang-ulang dan memastikan penomoran variabelnya tepat sesuai urutan identitas mahasiswa.

Cara penomoran variabel seperti diatas selain tidak efektif, juga menimbulkan masalah tersendiri yakni tidak adanya kesatuan atau integrity informasi yang menjelaskan mahasiswa spesifik. Semua variabel-variabel tersebut kenyataannya variabel berdiri sendiri yang kesatuannya hanya pada level pemahaman programmer atau pembaca kode program. Padahal nama, nim, alamat, umur dan jenis kelamin adalah satu kesatuan informasi yang menjelaskan tentang entitas mahasiswa.

Solusi atas masalah integritas informasi tersebut dapat diatasi dengan menggunakan custom tipe data dengan User Defined Type (UDT) atau Abstract Data Type (ADT). Perhatikan contoh berikut dengan menggunakan tipe Struct UDT:

```
#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

/*
          Deklarasi tipe data struktur
          mahasiswa  dengan variabel primitif didalamnya
*/
struct Mahasiswa{
     string nama;
     string nim;
     char jenisKelamin;
     int umur;
     int angkatan;
};

int main(){

     /*
          Membuat variabel mahasiswa
          dengan tipe data  Mahasiswa
     */
     Mahasiswa mhs1, mhs2;

     // Memberi nilai mahasiswa 1
     mhs1.nama = "Arifudding";
```

```
	mhs1.nim = "080940101";
	mhs1.jenisKelamin = 'L';
	mhs1.umur = 19;
	mhs1.angkatan = 2019;

	// Memberi nilai mahasiswa 2
	mhs2.nama = "Salma Roshidin";
	mhs2.nim = "080940102";
	mhs2.jenisKelamin = 'P';
	mhs2.umur = 19;
	mhs2.angkatan = 2019;

	cout << "Berikut identitas mahasiswa 1 : \n"
		<< setw(16) << left << "Nama " << ": "
		<< mhs1.nama << endl
		<< setw(16) << left << "NIM " << ": "
		<< mhs1.nim << endl
		<< setw(16) << left << "Jenis Kelamin " << ": "
		<< mhs1.jenisKelamin << endl
		<< setw(16) << left << "Umur " << ": "
		<< mhs1.umur << endl
		<< setw(16) << left << "Angkatan " << ": "
		<< mhs1.angkatan << endl;

	system("pause");
	return 0;
}
```

Cara deklarasi tipe data custom atau User Defined Type yaitu tipe struct Mahasiswa pada bagian deklarasi program mengatasi keharusan deklarasi berulang-ulang variabel nama, nim, umur, jenis kelamin dan angkatan untuk merepresentasikan data mahasiswa dalam program. Cara ini lebih efektif dan sekaligus menampilkan integritas data. Secara eksplisit terlihat semua variabel anggota UDT Mahasiswa adalah satu kesatuan atribut yang melambangkan entitas mahasiswa.

Lebih lanjut, tipe UDT dapat dikembangkan menjadi ADT. ADT adalah tipe abstrak yang dinyatakan oleh class program. ADT selain memiliki atribut variabel juga memiliki deklarasi operasional dari suatu tipe data. ADT secara konsep dipelajari mendalam pada pemrograman berorientasi objek (PBO).

Perhatikan contoh penyelesaian organisasi data mahasiswa dengan menggunakan ADT class program:

```
#include <iostream>
#include <iomanip>
#include <string>
```

```cpp
using namespace std;

/*
          Deklarasi tipe data struktur
          mahasiswa  dengan ADT
*/
class Mahasiswa{
public:
      string nama;
      string nim;
      char jenisKelamin;
      int umur;
      int angkatan;

public:
      //consructor
      Mahasiswa();

      //Destructor
      ~Mahasiswa();
};

Mahasiswa::Mahasiswa(){
      cout << "Constructor Mahasiswa dieksekusi\n";
}

Mahasiswa::~Mahasiswa(){
      cout << "Destructor Mahasiswa dieksekusi\n";
}

int main(){

      /*
            Membuat variabel mahasiswa
            dengan tipe data  Mahasiswa
      */
      Mahasiswa mhs1, mhs2;

      // Memberi nilai mahasiswa 1
      mhs1.nama = "Arifudding";
      mhs1.nim = "080940101";
      mhs1.jenisKelamin = 'L';
      mhs1.umur = 19;
      mhs1.angkatan = 2019;

      // Memberi nilai mahasiswa 2
      mhs2.nama = "Salma Roshidin";
      mhs2.nim = "080940102";
      mhs2.jenisKelamin = 'P';
      mhs2.umur = 19;
      mhs2.angkatan = 2019;
```

```
    cout << "Berikut identitas mahasiswa 1 : \n"
         << setw(16) << left << "Nama " << ": "
         << mhs1.nama << endl
         << setw(16) << left << "NIM " << ": "
         << mhs1.nim << endl
         << setw(16) << left << "Jenis Kelamin " << ": "
         << mhs1.jenisKelamin << endl
         << setw(16) << left << "Umur " << ": "
         << mhs1.umur << endl
         << setw(16) << left << "Angkatan " << ": "
         << mhs1.angkatan << endl;

    system("pause");
    return 0;
}
```

Pada program menggunakan Abstract Data Type (ADT) diatas deklarasi struktur data menggunakan class. Pada contoh kita menambahkan public constructor dan destructor yang akan dieksekusi saat instasiasi objek dan destructor akan dieksekusi saat objek didealokasikan/dibuang dari memory.

Tugas Pendahuluan

1. Buatlah sebuah program kasir yang menampilkan transaksi pembelian saat pembayaran. Informasi transaksi harus memperlihatkan nama barang, kode barang, harga satuan setiap barang.
2. Gunakan UDT struct untuk mengerjakan program kasir tersebut.

Langkah-Langkah Praktikum

1. Input dan output data mahasiswa dengan struktur UDT (User Defined Type) menggunakan struct.
   a. Buat file baru dengan nama praktikum11.cpp
   b. Ketik kode program berikut:

```
#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

struct Mahasiswa{
    string nama;
    string nim;
    char jenisKelamin;
    int umur;
```

```
      int angkatan;
};

int main(){

      char opsi = '2';

      do{
            Mahasiswa mhs;
            cout << endl
                  << "Masukan identitas mahasiswa:"
                  << endl
                  << "Nama : "; cin >> mhs.nama;
            cout << "NIM : "; cin >> mhs.nim;
            cout << "Jenis Kelamin [L/P] : ";
            cin >> mhs.jenisKelamin;
            cout << "Umur : "; cin >> mhs.umur;
            cout << "Angkatan : "; cin >> mhs.angkatan;

            cout << "Berikut identitas mahasiswa : \n"
             << setw(16) << left << "Nama " << ": "
             << mhs.nama << endl
             << setw(16) << left << "NIM " << ": "
             << mhs.nim << endl
             << setw(16) << left << "Jenis Kelamin " << ": "
             << mhs.jenisKelamin << endl
             << setw(16) << left << "Umur " << ": "
             << mhs.umur << endl
             << setw(16) << left << "Angkatan " << ": "
             << mhs.angkatan << endl;

            cout << endl
                  << "Pilihan: " << endl
                  << "1. Ulang" << endl
                  << "2. Exit" << endl
                  << "Pilihan anda:";
            cin >> opsi;
      }while(opsi=='1');
      cout << endl
            << "Program selesai!" << endl;
      system("pause");
      return 0;
}
```

c. Jalankan program

2. Input dan output data mahasiswa dengan struktur ADT (Abstract Data Type) menggunakan class.
   a. Buat file baru dengan nama praktikum12.cpp
   b. Ketik kode berikut:

```cpp
#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

class Mahasiswa{
public:
	string nama;
	string nim;
	char jenisKelamin;
	int umur;
	int angkatan;
};

int main(){

	char opsi = '2';

	do{
		Mahasiswa mhs;
		cout << endl
			<< "Masukan identitas mahasiswa:" << endl
			<< "Nama : "; cin >> mhs.nama;
		cout << "NIM : "; cin >> mhs.nim;
		cout << "Jenis Kelamin [L/P] : ";
		cin >> mhs.jenisKelamin;
		cout << "Umur : "; cin >> mhs.umur;
		cout << "Angkatan : "; cin >> mhs.angkatan;

		cout << "Berikut identitas mahasiswa : \n"
		 << setw(16) << left << "Nama " << ": "
		 << mhs.nama << endl
		 << setw(16) << left << "NIM " << ": "
		 << mhs.nim << endl
		 << setw(16) << left << "Jenis Kelamin " << ": "
		 << mhs.jenisKelamin << endl
		 << setw(16) << left << "Umur " << ": "
		 << mhs.umur << endl
		 << setw(16) << left << "Angkatan " << ": "
		 << mhs.angkatan << endl;

		cout << endl
			<< "Pilihan: " << endl
			<< "1. Ulang" << endl
			<< "2. Exit" << endl
			<< "Pilihan anda:";
		cin >> opsi;
	}while(opsi=='1');
	cout << endl
```

```
            << "Program selesai!" << endl;
      system("pause");
      return 0;
}
```

c. Jalankan program

Tugas Laporan

1. Buatlah program dengan menggunakan tipe data struct untuk menyelesaikan studi kasus berikut:
2. Pak Maman adalah pemilik penginapan Home-Stay dengan 20 kamar yang disewakan harian. Pak Maman membutuhkan sebuah perangkat lunak untuk mengelola penginapannya. Dia harus tahu setiap saat kodisi setiap kamar, yaitu info tentang kosong atau tidaknya, nomor setiap kamar, nama penyewa, lama sewa dan nomor kontak penyewa agar setiap saat bisa dihubungi jika ada satu dan lain hal terjadi. Demikian juga Pak Maman menginginkan setiap program dijalankan di komputer, dia bisa secara dinamis mengeset status kamar jika ada penyewa yang sewa atau selesai sewa.

Modul Praktikum 2: Array dan Matriks Data

Deskripsi Pertemuan

Percobaan ini akan membahas tentang struktur array sebagai struktur data yang simple. Mahasiswa akan diperkenalkan penggunaan array sebagai koleksi data primitive maupun UDT\ADT.

Syarat Kompetensi

1. Memahami dengan baik tipe data dan penggunaannya.
2. Mampu merancang dan mengimplementasikan User Defined Type atau Abstract Data Type dalam pemrograman.
3. Memahami dan mampu menggunakan logika perulangan dalam program.

Standar Kompetensi

1. Mahasiswa mampu mengimplementasikan Array dan array multi dimensi dalam program.
2. Mahasiswa mampu menganalisis dan merancang penyelesaian kasus menggunakan Array dan array multi dimensi dalam program

Dasar Teori

Pada contoh praktikum pertama, organisasi atribut-atribut data mahasiswa terbungkus dalam suatu struktur data baik menggunakan struct maupun menggunakan class ADT. Namun pada bagian deklarasi variabel untuk data mahasiswa 1 dan 2, tetap saja menggunakan variabel mhs1 dan mhs2. Cara ini tentu tidak optimal jika harus mengelola puluhan, ratusan atau bahkan ribuan mahasiswa. Demikian pula seandainya data mahasiswa tersebut hendak diorganisasikan dalam kaitannya dengan informasi tertentu misalnya relasi mahasiswa dengan nomor kursi tempat duduk di dalam kelas perkuliahan. Contoh lain misalkan relasi buku dengan lokasi dan nomor lemari penyimpanan pada perpustakaan.

Dalam hal ini kita membutuhkan suatu representasi variabel yang praktis untuk mengelola objek data mahsiswa dalam jumlah tertentu. Dalam hal ini kita bisa menggunakan tipe data array untuk mendeklarasikan kumpulan objek mahasiswa. Ini mungkin dilakukan selama objek array bertipe data sama. Perhatikan contoh berikut **program-array-satu-dimensi**:

```
#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

/*
          Deklarasi tipe data struktur
          mahasiswa  dengan variabel primitif didalamnya
*/
struct Mahasiswa{
     string nama;
     string nim;
     char jenisKelamin;
     int umur;
     int angkatan;
};

int main(){

     /*
          Membuat variabel array mahasiswa
          dengan kapasitas 2
     */
     Mahasiswa mhs[2];

     // Memberi nilai mahasiswa 1 dengan indeks array 0
     mhs[0].nama = "Arifudding";
     mhs[0].nim = "080940101";
     mhs[0].jenisKelamin = 'L';
     mhs[0].umur = 19;
     mhs[0].angkatan = 2019;

     // Memberi nilai mahasiswa 2 dengan indeks array 1
     mhs[1].nama = "Salma Roshidin";
     mhs[1].nim = "080940102";
     mhs[1].jenisKelamin = 'P';
     mhs[1].umur = 19;
     mhs[1].angkatan = 2019;

     //mengakses objek mhs 2 dengan indeks array 1
     cout << "Berikut identitas mahasiswa 2 : \n"
          << setw(16) << left << "Nama " << ": "
          << mhs[1].nama << endl
```

```
            << setw(16) << left << "NIM " << ": "
            << mhs[1].nim << endl
            << setw(16) << left << "Jenis Kelamin " << ": "
            << mhs[1].jenisKelamin << endl
            << setw(16) << left << "Umur " << ": "
            << mhs[1].umur << endl
            << setw(16) << left << "Angkatan " << ": "
            << mhs[1].angkatan << endl;

      system("pause");
      return 0;
}
```

Penggunaan array objek pada contoh, mengoptimalkan penggunaan variabel dengan hanya 1 variabel untuk merepresentasikan banyak data dengan pengkodean indeks pada variabel array. Cara ini akan memudahkan dalam menulis kode untuk mengimplementasikan algoritma pengurutan atau pencarian data. Data dalam array dapat lebih mudah diperiksa dalam skope logika perulangan dengan mengakses masing-masing objek melalui indeksnya secara mandiri. Ini tentu sulit dilakukan jika setiap entitas mahasiswa dideklarasikan dengan nama variabel berbeda.

Array objek mahasiswa yang telah dicontohkan adalah array satu dimensi. Array dapat pulan dikonstruksi dengan ukuran multi dimensi. Pemetaan posisi duduk mahasiswa didalam kelas dapat secara kreatif kita nyatakan dalam bentuk array multi dimensi.

Misalnya suatu kelas terdiri atas 20 kursi yang terisi penuh oleh mahasiswa. Kursi-kursi diatur 5 kursi setiap baris sehingga total ada 4 banjar kursi yang terisi mahasiswa.

Kondisi tersebut dapat direpresentasikan menggunkan array 5 kali 4. Selanjutnya setiap elemen array dapat diakses menggunakan alamat vektor array misal [0,0] untuk mengakses representasi kuris pada baris pertama dan banjar pertama. Vektor [3,2] adalah representasi baris keempat dan banjar ke tiga dan seterusnya. Kondisi ini bisa kita sebut sebagai matriks data yang setiap elemennya diakses dengan vektor elemen baris dan kolom.

Perhatikan contoh berikut **program-array-multi-dimensi**:

```
#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

/*
           Deklarasi tipe data struktur
           mahasiswa  dengan variabel primitif didalamnya
```

```
*/
struct Mahasiswa{
	string nama;
	string nim;
	char jenisKelamin;
	int umur;
	int angkatan;
};

int main(){

	/*
		Deklarasi variabel array multidimensi
		5x4 atau matriks [5][4]
	*/
	Mahasiswa mhs[5][2];

	/*
		Mahsiwa baris pertama banjar pertama
		Vektor array [0][0]
	*/
	mhs[0][0].nama = "Arifudding";
	mhs[0][0].nim = "080940101";
	mhs[0][0].jenisKelamin = 'L';
	mhs[0][0].umur = 19;
	mhs[0][0].angkatan = 2019;

	/*
		Mahsiwa baris keempat banjar ketiga
		Vektor array [3][2]
	*/
	mhs[3][2].nama = "Salma Roshidin";
	mhs[3][2].nim = "080940102";
	mhs[3][2].jenisKelamin = 'P';
	mhs[3][2].umur = 19;
	mhs[3][2].angkatan = 2019;

	//mengakses objek mhs 1 dengan Vektor array [0][0]
	cout << "Berikut identitas mahasiswa 2 : \n"
	       << setw(16) << left << "Nama " << ": "
	       << mhs[0][0].nama << endl
	       << setw(16) << left << "NIM " << ": "
	       << mhs[0][0].nim << endl
	       << setw(16) << left << "Jenis Kelamin " << ": "
	       << mhs[0][0].jenisKelamin << endl
	       << setw(16) << left << "Umur " << ": "
	       << mhs[0][0].umur << endl
	       << setw(16) << left << "Angkatan " << ": "
	       << mhs[0][0].angkatan << endl;

	system("pause");
```

```
	return 0;
}
```

Yang perlu diperhatikan adalah indeks array selalui dimulai dari angka 0, sehingga indeks terbesar array adalah n-1.

Tugas Pendahuluan

1. Buatlah program untuk mengorganisasi data-data mahsiswa dan asal tempat tinggalnya serta jarak rumah tinggal ke kampus.
2. Kemudian melalui program tersebut, implementasikan agloritma pengurutan untuk menampilkan urutan mahasiswa yang paling jauh lokasi domisilinya dari kampus dan sebaliknya dengan urutan yang paling dekat.
3. Pastikan anda menggunakan struktur data dan array dalam mengerjakan tugas program anda.

Langkah-Langkah Praktikum

1. Array dan matriks data
   a. Buatlah file baru dengan nama praktikum21.cpp
   b. Ketikan kode **program-array-satu-dimensi** di atas
   c. Jalankan program
   d. Buatlah file baru lagi dengan nama praktikum22.cpp
   e. Ketik kode **program-array-multi-dimensi** di atas
   f. Jalankan program.
2. Array struct didalam object data struct (struct di dalam struct).
   a. Buat file baru dengan nama praktikum23.cpp
   b. Ketik kode berikut:

```
#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

struct MataKuliah{
	string mataKuliah;
	double nilai;
	/*MataKuliah(){}
	MataKuliah(string mk, double nil){
		mataKuliah = mk;
		nilai = nil;
	}*/
};
```

```
struct Mahasiswa{
	string nama;
	int nim;
	MataKuliah mk[2];
	/*Mahasiswa(){}
	Mahasiswa(string nma, int nm){
		nama = nma;
		nim = nm;
	}*/
};

void TambahkanMataKuliah(Mahasiswa &mhs, \
	MataKuliah mk1, MataKuliah mk2){
	mhs.mk[0] = mk1;
	mhs.mk[1] = mk2;
}


int main(){

	char opsi = '2';

	do{
		Mahasiswa mhs;
		cout << endl
			<< "Masukan identitas mahasiswa:" << endl
			<< "Nama [string]: "; cin >> mhs.nama;
		cout << "NIM [integer]: "; cin >> mhs.nim;
		MataKuliah mk1, mk2;
		cout << "Matakuliah 1 [string] : ";
		cin >> mk1.mataKuliah;
		cout << "Nilai [Double] : "; cin >> mk1.nilai;
		cout << "Matakuliah 2 [string] : ";
		cin >> mk2.mataKuliah;
		cout << "Nilai [Double] : "; cin >> mk2.nilai;
		cout << endl;
		TambahkanMataKuliah(mhs,mk1,mk2);
	cout << "Berikut data mahasiswa dan nilai MKnya: \n"
		 << setw(16) << left << "Nama [string]" << ": "
		 << mhs.nama << endl
		 << setw(16) << left << "NIM [integer]" << ": "
		 << mhs.nim << endl
		 << setw(16) << left << "\tMata Kuliah 1 "
		 << ": "
		 << mhs.mk[0].mataKuliah << endl
		 << setw(16) << left << "\tNilai  " << ": "
		 << mhs.mk[0].nilai << endl
		 << setw(16) << left << "\tMata Kuliah 2 "
		 << ": "
		 << mhs.mk[1].mataKuliah << endl
		 << setw(16) << left << "\tNilai  " << ": "
```

```
            << mhs.mk[1].nilai << endl;
      cout << endl << endl;

            cout << endl
                  << "Pilihan: " << endl
                  << "1. Ulang" << endl
                  << "2. Exit" << endl
                  << "Pilihan anda:";
            cin >> opsi;
      }while(opsi=='1');
      cout << endl
             << "Program selesai!" << endl;
      system("pause");
      return 0;
}
```

c. Jalankan program

Tugas Laporan

1. Buatlah program dengan menggunakan struktur struct, array serta gunakan algoritma pegurutan dan perulangan untuk menyelesaikan tugas berikut:
2. Jono dan Yulian mendapat tugas berbeda dari gurunya. Jono mendapat tugas untuk mencatat nama-nama siswa kelas VIII. Guru Jono terkadang meminta daftar seluruh nama siswa yang terurut mulai yang paling pendek hingga paling panjang namanya. Kadang pula dengan urutan sebaliknya. Namun suatu ketika, Guru meminta lagi hanya sepuluh nama siswa yang paling panjang namanya. Melayani permintaan gurunya, Jono cukup kerepotan, karena itu dia memutuskan membuat program komputer yang bisa membantunya seketika menyediakan data untuk gurunya.
3. Sementara itu Yulian diminta guru mencatat seluruh kelas di sekolah negeri tersebut dan nama wali kelas masing-masing kelas. Maing-masing angkatan yaitu kelas VII sampai IX memiliki kode kelas abjad A sampai J, seperti VIIA hingga IXJ. Setiap kelas memiliki satu Wali kelas, misalnya kelas VIIIC wali kelasnya adalah Umar Sulaiman, S. Kom. Supaya mudah setiap saat memenuhi permintaan data-data wali kelas sekolah tersebut, maka Yulian berniat membuat program yang bisa membantunya mengelolah data wali kelas.
4. Buatlah program untuk membantu Jono dan Yulian!

Modul Praktikum 3: Linked List

Deskripsi Pertemuan

Praktikum tiga akan memperkenalkan struktur data Linked-List yaitu model simple linked list dan doubly linked list. Mahasiswa diharapkan mampu mendalami model struktur linked list sehingga mampu mengimplementasikannya secara bersamaan dengan tipe data UDT maupun ADT.

Syarat Kompetensi

1. Memahami dan mampu menggunakan tipe data UDT\ADT.
2. Memahami dan mampu membuat program dengan menggunakan logika seleksi atau struktur kontrol.
3. Memahami variabel referensi dan pointer.

Standar Kompetensi

1. Mahasiswa mampu mengimplementasikan struktur single linked list dan double linked list dalam program.
2. Mahasiswa mampu menganalisis dan merancang program penyelesaian kasus menggunakan single linked list dan double linked list.

Dasar Teori

Deklarasi koleksi data dengan struktur array seperti pada praktikum sebelumnya sejauh ini terlihat bisa memecahkan masalah. Namun jika kita kembangkan sedikit jauh, akan terlihat bahwa tipe array bersifat statis, yaitu kita harus mendeklarasikan ukuran array tertentu yang kita cocokan dengan ukuran data yang hendak kita kelola. Namun ada kalanya koleksi data yang akan dikelola tidak dalam jumlah yang bisa diprediksi. Bahkan terkadang suatu koleksi data bisa mengalami penambahan atau pengurangan.

Selain itu, sifat statis array juga punya konskuensi di kapasitas penggunaan memory. Array yang dideklarasikan dengan ukuran 10 akan mengambil alokasi untuk sepuluh data

di memori, meskipun data yang saat itu kita tambahkan pada variabel array kurang dari 10.

Karena itu, struktur Linked List menjadi alternatif untuk mengorganisasi data yang bersifat dinamis. Suatu struktur linked list dapat ditambahkan elemen secara dinamis pada saat runtime dan tidak perlu definisi ukuran pada saat deklarasi.

Linked list dibentuk dengan memanfaatkan pointer yang menyimpan alamat memory dari setiap element data. Setiap elemen dari linked list akan menyimpan alamat pointer dari elemen lain. Dengan demikian elemen-elemen tersebut dapat dirangkaikan secara logis membentuk linked-list.

Perhatikan kode program **Linear List** berikut:

```
#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

//deklarasi struktur node
struct Node{
      string nama;
      Node * nextNode;
};

//method addNode dan cetak list
void addNode(string nama);
void cetakList();

//deklarasi head
Node *head;


int main(){

      string nama1 = "Arifudding";
      string nama2 = "Salma Roshidin";
      string nama3 = "Joko Subando";

      cetakList();
      addNode(nama1);
      cetakList();
      addNode(nama2);
      cetakList();
      addNode(nama3);
      cetakList();

      cout << endl;
```

```
	system("pause");
	return 0;
}

void addNode(string nama){
	Node * newNodePtr = new Node;
	newNodePtr->nama = nama;
	newNodePtr->nextNode = nullptr;
	if(head == NULL){
		head = newNodePtr;
	}else{
		Node *prevHeadPtr = head;
		head = newNodePtr;
		newNodePtr->nextNode = prevHeadPtr;
	}
}

void cetakList(){
	int i = 0;
	cout << "Berikut daftar mahasiswa dalam list: \n";
	if(head == NULL){
		cout << "List kosong!\n\n";
	}else{
		Node *p = head;
		while(p != NULL){
			i++;
			cout << i << ". "
				<< p->nama << "." << endl;
			p = p->nextNode;
		}
	cout << endl << endl;
	}
}
```

Program Linear List diatas adalah bentuk single linked list atau linear linked list yang hanya diidentifikasi dengan penanda head. Setiap node baru yang ditambahkan, akan menjadi head baru dan head yang lama akan digeser dan direferensi oleh node baru melalui atribut **nextNode** node baru.

Pada contoh, setiap element linked list hanya bisa dilacak dari referensi head ke data berikutnya. Namun tidak ada cara menakses sebelumnya dari posisi node yang direferensi. Ini karena struktur node hanya didesain memiliki 1 pointer node.

Untuk mengakses data, menambah atau menghapus data dari arah depan atau belakang untaian data, maka perlu menggunakan double linked list. Setiap node pada double linked list dapat melacak baik node didepan maupun dibelakangnya dalam untaian node. Perhatikan program **Double Linked List** berikut:

```
#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

//deklarasi struktur double linked node
struct Node{
	string nama;
	Node * nextNode;
	Node * prevNode;
};

//method add node dan cetak list
void addNodeAtFront(string nama);
void addNodeAtBack(string nama);
void cetakList();

//deklarasi head dan tail
Node *head;
Node *tail;


int main(){

	string nama1 = "Arifudding";
	string nama2 = "Salma Roshidin";
	string nama3 = "Joko Subando";
	string nama4 = "Susanna Edy";
	string nama5 = "Daeng Bali";
	string nama6 = "Puang Cani";

	cetakList();
	cout << "Tambah data di depan:\n";
	addNodeAtFront(nama1);
	cetakList();
	cout << "Tambah data di belakang:\n";
	addNodeAtBack(nama2);
	cetakList();
	cout << "Tambah data di depan:\n";
	addNodeAtFront(nama3);
	cetakList();
	cout << "Tambah data di belakang:\n";
	addNodeAtBack(nama4);
	cetakList();
	cout << "Tambah data di belakang:\n";
	addNodeAtBack(nama5);
	cetakList();
	cout << "Tambah data di depan:\n";
	addNodeAtFront(nama6);
```

```
	cetakList();

	cout << endl;

	system("pause");
	return 0;
}

void addNodeAtFront(string nama){
	Node * newNodePtr = new Node;
	newNodePtr->nama = nama;
	newNodePtr->nextNode = nullptr;
	newNodePtr->prevNode = nullptr;
	if(head == NULL){
		head = newNodePtr;
		tail = nullptr;
	}else if(tail == NULL){
		tail = newNodePtr;
		tail->prevNode = head;
		head->nextNode = tail;
	}else{
		Node *prevHeadPtr = head;
		head = newNodePtr;
		newNodePtr->nextNode = prevHeadPtr;
		prevHeadPtr->prevNode = newNodePtr;
	}
}

void addNodeAtBack(string nama){
	Node * newNodePtr = new Node;
	newNodePtr->nama = nama;
	newNodePtr->nextNode = nullptr;
	newNodePtr->prevNode = nullptr;
	if(head == NULL){
		head = newNodePtr;
		tail = nullptr;
	}else if(tail == NULL){
		tail = newNodePtr;
		tail->prevNode = head;
		head->nextNode = tail;
	}else{
		Node *prevTailPtr = tail;
		prevTailPtr->nextNode = newNodePtr;
		newNodePtr->prevNode = prevTailPtr;
		tail = newNodePtr;
	}
}

void cetakList(){
	int i = 0;
	cout << "Berikut daftar mahasiswa dalam list: \n";
```

```
      if(head == NULL){
            cout << "List kosong!\n\n";
      }else{
            Node *p = head;
            while(p != NULL){
                  i++;
                  cout << i << ". "
                        << p->nama << "." << endl;
                  p = p->nextNode;
            }
      cout << endl << endl;
      }
}
```

Program double linked list memperlihatkan bahwa senarai data dapat diakses dari arah depan maupun arah belakang. Demikian juga setiap posisi node dapat merujuk pada node didepan atau dibelakangnya tanpa perlu mengulang dari posisi head linked list.

Tugas Pendahuluan

1. Ubahlah program **Linear List** pada bagian teori sehingga setiap penambahan node dilakukan bukan pada bagian head tapi ditambahkan dibagian belakang data.
2. lengkapi kode program **Double Linked List** sehingga mampu melakukan penambahan data, pencarian, penghapusan, hingga penyisipan data setelah data tertentu dalam senarai.

Langkah-Langkah Praktikum

1. Linked list menambah dan menghapus node ADT
   a. Buat file baru dengan nama praktikum31.cpp
   b. Ketik kode berikut:

```
#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

//deklarasi ADT
class Mahasiswa{
private:
      string nama;
      int nim;
public:
      Mahasiswa(){}
      ~Mahasiswa(){
            cout << "Dihapus " << nama << endl;
```

```
	}
	Mahasiswa(string nma, int nm){
		nama = nma;
		nim = nm;
	}
	string getNama(){
		return nama;
	}
	int getNim(){
		return nim;
	}
};

//deklarasi struktur node
struct Node{
	Mahasiswa mahasiswa;
	Node * nextNode;
	Node * prevNode;
};

//method tambah data
void tambahDepan(Mahasiswa mhs);
//method  keluarkan data
void hapusBelakang();
void cetakData();

//deklarasi head dan tail
Node *head;
Node *tail;


int main(){

	Mahasiswa mhs1 = Mahasiswa\
			("Arifudding", 123);
	Mahasiswa mhs2 = Mahasiswa\
			("Salma Roshidin", 456);
	Mahasiswa mhs3 = Mahasiswa\
			("Joko Subando", 876);


	cetakData();
	tambahDepan(mhs1);
	cetakData();
	tambahDepan(mhs2);
	cetakData();
	tambahDepan(mhs3);
	cetakData();
	tambahDepan(Mahasiswa("Sugeng Yellor", 6789));
	cetakData();
	tambahDepan(Mahasiswa("Budi Sumantri", 870));
```

```
	cetakData();
	hapusBelakang();
	cetakData();
	hapusBelakang();
	cetakData();
	hapusBelakang();
	cetakData();

	cout << endl;
	cout << "SELESAI!";
	cout << endl;
	system("pause");
	return 0;
}

// add node as new head
void tambahDepan(Mahasiswa mhs){
	cout << "Tambah data depan\n\n";
	Node * newQueueNode = new Node;
	newQueueNode->mahasiswa = mhs;
	newQueueNode->nextNode = nullptr;
	newQueueNode->prevNode = nullptr;
	if(head == NULL){
		head = newQueueNode;
		tail = NULL;
	}else if(tail == NULL){
		tail = head;
		head = newQueueNode;
		head->nextNode = tail;
		tail->prevNode = head;
	}else{
		Node *nextTailNode = head;
		head = newQueueNode;
		newQueueNode->nextNode = nextTailNode;
		nextTailNode->prevNode = newQueueNode;
	}
}

void hapusBelakang(){

	if(tail == NULL){
		cout << "linked list kosong!\n\n";
	}else{
		Node * delQueueNode = tail;
		tail = delQueueNode->prevNode;
		tail ->nextNode = nullptr;
		cout << "Memproses hapus node belakang:"
			<< "Nama : "
			<< delQueueNode->mahasiswa.getNama()
			<< endl
			<< "NIM : "
```

```
                    << delQueueNode->mahasiswa.getNim()
                    << endl;
              delete delQueueNode;
        }
}


void cetakData(){
        int i = 0;
        cout << "Cetak linked list dari depan: \n";
        if(head == NULL){
              cout << "linked list kosong!\n\n";
        }else{
              Node *p = head;
              while(p != NULL){
                    i++;
                    cout << i << ". "
                          << "Nama : " << p->mahasiswa.getNama()
                                << "." << endl
                           << "    "
                           << "NIM : " << p->mahasiswa.getNim()
                                << "." << endl;
                    p = p->nextNode;
              }
        cout << endl << endl;
        }
}
```

c. Jalankan program

2. Linked list menambah dan menghapud head dan tail.
   a. Buat file baru dengan nama praktikum32.cpp
   b. Ketik kode berikut:

```
#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

//deklarasi ADT
class Mahasiswa{
private:
      string nama;
      int nim;
public:
      Mahasiswa(){}
      ~Mahasiswa(){
```

```
            cout << "Dihapus " << nama << endl;
      }
      Mahasiswa(string nma, int nm){
            nama = nma;
            nim = nm;
      }
      string getNama(){
            return nama;
      }
      int getNim(){
            return nim;
      }
};

//deklarasi struktur node
struct Node{
      Mahasiswa mahasiswa;
      Node * nextNode;
      Node * prevNode;
};

//method tambah
void tambahDepan(Mahasiswa mhs);
void tambahBelakang(Mahasiswa mhs);
//method keluarkan
void keluarDepan();
void keluarBelakang();
void cetakDataDariDepan();
void cetakDataDariBelakang();

//deklarasi dengan head dan tail
Node *head;
Node *tail;


int main(){

      Mahasiswa mhs1 = Mahasiswa\
                  ("Arifudding", 123);
      Mahasiswa mhs2 = Mahasiswa\
                  ("Salma Roshidin", 456);
      Mahasiswa mhs3 = Mahasiswa\
                  ("Joko Subando", 876);


      cetakDataDariDepan();
      cetakDataDariBelakang();
      tambahDepan(mhs1);
      tambahBelakang(mhs2);

      cetakDataDariDepan();
```

```
	cetakDataDariBelakang();
	tambahDepan(mhs3);
	tambahBelakang(Mahasiswa("Sugeng Yellor", 6789));
	cetakDataDariDepan();
	cetakDataDariBelakang();
	tambahBelakang(Mahasiswa("Budi Sumantri", 870));
	cetakDataDariDepan();
	cetakDataDariBelakang();

	//keluarDepan();
	//cetakDataDariDepan();
	keluarBelakang();
	cetakDataDariBelakang();
	//keluarDepan();
	keluarBelakang();
	//cetakDataDariDepan();
	cetakDataDariBelakang();

	cout << endl;
	cout << "SELESAI!";
	cout << endl;
	system("pause");
	return 0;
}

// add node as new head
void tambahDepan(Mahasiswa mhs){
	cout << "Tambah data depan\n\n";
	Node * newQueueNode = new Node;
	newQueueNode->mahasiswa = mhs;
	newQueueNode->nextNode = nullptr;
	newQueueNode->prevNode = nullptr;
	if(head == NULL){
		head = newQueueNode;
		tail = NULL;
	}else if(tail == NULL){
		tail = head;
		head = newQueueNode;
		head->nextNode = tail;
		tail->prevNode = head;
	}else{
		Node *nextTailNode = head;
		head = newQueueNode;
		newQueueNode->nextNode = nextTailNode;
		nextTailNode->prevNode = newQueueNode;
	}
}

// add node as new tail
void tambahBelakang(Mahasiswa mhs){
	cout << "Tambah data belakang\n\n";
```

```
	Node * newQueueNode = new Node;
	newQueueNode->mahasiswa = mhs;
	newQueueNode->nextNode = nullptr;
	newQueueNode->prevNode = nullptr;
	if(head == NULL){
		head = newQueueNode;
		tail = NULL;
	}else if(tail == NULL){
		tail = newQueueNode;
		head->nextNode = tail;
		tail->prevNode = head;
	}else{
		Node *prevTailNode = tail;
		tail = newQueueNode;
		prevTailNode->nextNode = tail;
		tail->prevNode = prevTailNode;
	}
}

//Antrian keluar dari head
void keluarDepan(){

	if(head == NULL){
		cout << "linked list kosong!\n\n";
	}else{
		Node * delQueueNode = head;
		head = delQueueNode->nextNode;
		 head->prevNode = nullptr;
		cout << "Memproses antrian keluar depan:"
			<< "Nama : "
			<< delQueueNode->mahasiswa.getNama()
			<< endl
			<< "NIM : "
			<< delQueueNode->mahasiswa.getNim()
			<< endl;
		delete delQueueNode;
	}
}

void keluarBelakang(){

	if(tail == NULL){
		cout << "linked list kosong!\n\n";
	}else{
		Node * delQueueNode = tail;
		tail = delQueueNode->prevNode;
		tail->nextNode = nullptr;
		cout << "Memproses antrian keluar belakang:"
			<< "Nama : "
			<< delQueueNode->mahasiswa.getNama()
			<< endl
```

```
                    << "NIM : "
                    << delQueueNode->mahasiswa.getNim()
                    << endl;
              delete delQueueNode;
        }
}
void cetakDataDariDepan(){
        int i = 0;
        cout << "Cetak linked list dari depan: \n";
        if(head == NULL){
              cout << "linked list kosong!\n\n";
        }else{
              Node *p = head;
              while(p != NULL){
                    i++;
                    cout << i << ". "
                          << "Nama : " << p->mahasiswa.getNama()
                                << "." << endl
                           << "    "
                           << "NIM : " << p->mahasiswa.getNim()
                                << "." << endl;
                    p = p->nextNode;
              }
        cout << endl << endl;
        }
}

void cetakDataDariBelakang(){
        int i = 0;
        cout << "Cetak linked list dari belakang: \n";
        if(tail == NULL){
              cout << "linked list kosong!\n\n";
        }else{
              Node *p = tail;
              while(p != NULL){
                    i++;
                    cout << i << ". "
                          << "Nama : " << p->mahasiswa.getNama()
                                << "." << endl
                           << "    "
                           << "NIM : " << p->mahasiswa.getNim()
                                << "." << endl;
                    p = p->prevNode;
              }
        cout << endl << endl;
        }
}
```

c. Jalankan program

Tugas Laporan

1. Buatlah sebuah program single linked list dengan fungsi menambah, menghapus dan menampilkan data UDT mahasiswa!
2. Ubah program nomor 1 dengan menggunakan double linked list.

Modul Praktikum 4: Stack, Queue dan Deque

Deskripsi Pertemuan

Praktikum ini akan memperkenalkan cara menyusun data dalam bentuk struktur tumpukan dan antrian menggunakan struktur data linear dan double linked list yang telah dipraktekan sebelumnya.

Syarat Kompetensi

1. Memahami dan mampu mengimplementasikan UDT/ADT dalam program.
2. Memahmi dan mampu menggunakan logika seleksi.
3. Memahami dan mampu menggunakan logika perulangan atau looping dalam program.

Standar Kompetensi

1. Mahasiswa mampu mengimplementasikan struktur stack, queue dan deque dalam program.
2. Mahasiswa mampu menganalisis dan merancang program penyelesaian kasus menggunakan stack, queue dan deque.

Dasar Teori

Stack atau tumpukan adalah struktur data yang menerapkan prinsip LIFO. LIFO merupakan agregasi dari Last In First Out yang artinya data yang terakhir masuk pada stack akan menjadi data yang pertama dikeluarkan dari stack. Seperti penamaannya yaitu ‘tumpukan’, data yang dimasukan pada stack/tumpukan seolah-olah ditumpuk. Data yang baru ditambahkan ditempatkan pada bagian atas data yang sudah ada. Seperti jika seseorang menumpuk buku diatas tumpukan buku-buku. Sehingga ketika mengambil data secara urut dari tumpukan, maka data paling atas tumpukan yang akan diambil, yaitu data yang paling akhir ditempatkan pada tumpukan. Inilah arti Last In First Out (LIFO) atau Terakhir Masuk Pertama Keluar.

Struktur tumpukan atau stack dapat dibentuk secara logis dengan array atau Linked List. Stack yang dibuat dengan array akan berukuran statik seperti array. Stack yang dinamis sebaliknya dapat disusun dengan menggunakan linked list.

Stack yang dibentuk dari linked list adalah senarai linear yang memiliki head. Single linked list yang asalnya hanya memiliki satu head atau satu tail pada dasarnya dapat langsung dibentuk menjadi stack. Penggunaan head atau tail pada senarai bersifat opsional. Bagian paling penting dari sisi implementasi logika LIFO pada senarai tersebut. Berikut contoh stack dengan linked list:

```
#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

//deklarasi struktur node
struct Node{
	string nama;
	Node * nextNode;
};

//method push tambah tumpukan
void push(string nama);
//method pop keluarkan tumpukan
void pop();
void cetakList();

//deklarasi stack sebagai head
Node *stack;


int main(){

	string nama1 = "Arifudding";
	string nama2 = "Salma Roshidin";
	string nama3 = "Joko Subando";

	cetakList();
	push(nama1);
	cetakList();
	push(nama2);
	cetakList();
	push(nama3);
	cetakList();

	pop();
	cetakList();
	pop();
```

```
	cetakList();

	cout << endl;

	system("pause");
	return 0;
}

void push(string nama){
	Node * newNodePtr = new Node;
	newNodePtr->nama = nama;
	newNodePtr->nextNode = nullptr;
	if(stack == NULL){
		stack = newNodePtr;
	}else{
		Node *prevHeadPtr = stack;
		stack = newNodePtr;
		newNodePtr->nextNode = prevHeadPtr;
	}
}

void pop(){

	if(stack == NULL){
		cout << "Stack kosong!\n\n";
	}else{
		Node * delNodePtr = stack;
		stack = delNodePtr->nextNode;
		cout << "Mengeluarkan data:"
			<< delNodePtr->nama << endl;
		delete delNodePtr;
	}
}

void cetakList(){
	int i = 0;
	cout << "Berikut daftar mahasiswa dalam Stack: \n";
	if(stack == NULL){
		cout << "Stack kosong!\n\n";
	}else{
		Node *p = stack;
		while(p != NULL){
			i++;
			cout << i << ". "
				<< p->nama << "." << endl;
			p = p->nextNode;
		}
	cout << endl << endl;
	}
}
```

Berbeda dengan stack, queue menerapkan prinsip FIFO, yang agregasi dari First In Firs Out. Queue atau disebut juga antrian akan mengeluarkan data sesuai urutan masuk data. Data yang pertama masuk akan pertama kali keluar. Sistem ini sama persis dengan antrian, siapa yang masuk terlebih dahulu kedalam antrian, akan mendapatkan pelayanan lebih dahulu.

Struktur Queue dapat dibentuk dengan menggunakan sigle atau linear linked list seperti halnya dengan stack. Akan tetapi penggunaan single linked list membutuhkan komputasi lebih pada operasi **dequeue()** karena bagian akhir dari linked list hanya bisa dirujuk dari head. Pendekatan yang lebih baik adalah dengan menambahkan poiter referensi tail pada bagian akhir linked list. Sehingga dengan mudah bagian akhir dari linked list bisa dirujuk.

Namun kali ini kita akan menggunakan struktur double linked list untuk membuat antrian. Pada Queue, kita bisa menambahkan tail atau ekor pada senarai double linked list. Tail atau ekor berfungsi sebagai tempat penambahan node data baru. Sedangkan head sebagai tempat keluarnya data antrian.

Perhatikan kode berikut:

```
#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

//deklarasi struktur node
struct Node{
      string nama;
      Node * nextNode;
      Node * prevNode;
};

//method enqueue tambah antrian
void enqueue(string nama);
//method dequeue keluarkan antrian
void dequeue();
void cetakQueue();

//deklarasi queue dengan head dan tail
Node *head;
Node *tail;


int main(){

      string nama1 = "Arifudding";
      string nama2 = "Salma Roshidin";
```

```
	string nama3 = "Joko Subando";

	cetakQueue();
	enqueue(nama1);
	cetakQueue();
	enqueue(nama2);
	cetakQueue();
	enqueue(nama3);
	cetakQueue();

	dequeue();
	cetakQueue();
	dequeue();
	cetakQueue();
	enqueue("Sugeng Yellor");
	cetakQueue();
	enqueue("Budi Sumantri");
	cetakQueue();
	dequeue();
	cetakQueue();
	dequeue();
	cetakQueue();

	cout << endl;

	system("pause");
	return 0;
}

// add node as new tail
void enqueue(string nama){
	Node * newQueueNode = new Node;
	newQueueNode->nama = nama;
	newQueueNode->nextNode = nullptr;
	newQueueNode->prevNode = nullptr;
	if(head == NULL){
		head = newQueueNode;
		tail = NULL;
	}else if(tail == NULL){
		tail = newQueueNode;
		head->nextNode = tail;
		tail->prevNode = head;
	}else{
		Node *prevTailNode = tail;
		tail = newQueueNode;
		prevTailNode->nextNode = tail;
		tail->prevNode = prevTailNode;
	}
}
```

```
//Antrian keluar dari head
void dequeue(){

	if(head == NULL){
		cout << "Queue kosong!\n\n";
	}else{
		Node * delQueueNode = head;
		head = delQueueNode->nextNode;
		cout << "Memproses antrian keluar:"
			<< delQueueNode->nama << endl;
		delete delQueueNode;
	}
}

void cetakQueue(){
	int i = 0;
	cout << "Berikut daftar mahasiswa dalam Queue: \n";
	if(head == NULL){
		cout << "Queue kosong!\n\n";
	}else{
		Node *p = head;
		while(p != NULL){
			i++;
			cout << i << ". "
				<< p->nama << "." << endl;
			p = p->nextNode;
		}
	cout << endl << endl;
	}

}
```

Versi lain dari queue atau antrian adalah **deque.** Deque adalah double-ended queue yang memungkinkan penambahan dan penghapusan queue pada head dan tail. Deque dapat diimplementasikan dengan single maupun double linked list dengan konsekuensi komputasi yang berbeda. Deque memungkinkan prinsip FIFO dan LIFO dalam satu struktur queue.

Tugas Pendahuluan

1. Buatlah sebuah stack dengan menggunakan double linked list.
2. Buatlah sebuah queue dengan menggunakan single linked list menggunakan penanda pointer head dan tail.
3. Buatlah sebuah deque dengan menggunakan single linked list.
4. Buatlah sebuah deque dengan menggunakan double linked list.

Langkah-Langkah Praktikum

1. Stack dengan object UDT
   a. Buatlah file baru dengan nama praktikum41.cpp
   b. Ketik kode berikut:

```
#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

//deklarasi Mahasiswa
struct Mahasiswa{
	string nama;
	int nim;

	Mahasiswa(){}

	Mahasiswa(string nma, int nm){
		nama = nma;
		nim = nm;
	}
};
//deklarasi struktur node
struct Node{
	Mahasiswa mahasiswa;
	Node * nextNode;

	Node(){}

};

//method push tambah tumpukan
void push(Mahasiswa mhs);
//method pop keluarkan tumpukan
void pop();
void cetakStack();

//deklarasi head stack
Node *atas;


int main(){

	Mahasiswa mhs1 = Mahasiswa("Arifudding", 123);
	Mahasiswa mhs2 = Mahasiswa("Salma Roshidin", 123);
	Mahasiswa mhs3 = Mahasiswa("Joko Subando", 123);

	cetakStack();
	push(mhs1);
```

```
	cetakStack();
	push(mhs2);
	cetakStack();
	push(mhs3);
	cetakStack();

	pop();
	cetakStack();
	pop();
	cetakStack();

	cout << endl;

	system("pause");
	return 0;
}

void push(Mahasiswa mhs){
	Node * newNodePtr = new Node;
	//Mahasiswa mhs1 = new Mahasiswa("Arifudding", 123);
	newNodePtr->mahasiswa = mhs;
	newNodePtr->nextNode = nullptr;
	if(atas == NULL){
		atas = newNodePtr;
	}else{
		Node *prevHeadPtr = atas;
		atas = newNodePtr;
		newNodePtr->nextNode = prevHeadPtr;
	}
}

void pop(){

	if(atas == NULL){
		cout << "Stack kosong!\n\n";
	}else{
		Node * delNodePtr = atas;
		atas = delNodePtr->nextNode;
		cout << "Mengeluarkan data:"
			<< delNodePtr->mahasiswa.nama << endl;
		delete delNodePtr;
	}
}

void cetakStack(){
	int i = 0;
	cout << "Berikut daftar mahasiswa dalam Stack: \n";
	if(atas == NULL){
		cout << "Stack kosong!\n\n";
	}else{
		Node *p = atas;
```

```
            while(p != NULL){
                  i++;
                  cout << i << ". "
                        << "Nama : " << p->mahasiswa.nama
                        << "." << endl
                        << "    "
                        << "NIM : " << p->mahasiswa.nim
                              << "." << endl;
                  p = p->nextNode;
            }
      cout << endl << endl;
      }
}
```

c. Jalankan program
d. Uji dengan beberapa skenario pada kodenya

2. Antrian dengan object ADT
   a. Buat file baru dengan nama praktikum42.cpp
   b. Ketik kode berikut:

```
#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

//deklarasi ADT
class Mahasiswa{
private:
      string nama;
      int nim;
public:
      Mahasiswa(){}
      Mahasiswa(string nma, int nm){
            nama = nma;
            nim = nm;
      }
      string getNama(){
            return nama;
      }
      int getNim(){
            return nim;
      }
};

//deklarasi struktur node
struct Node{
      Mahasiswa mahasiswa;
```

```
      Node * nextNode;
      Node * prevNode;
};

//method enqueue tambah antrian
void enqueue(Mahasiswa mhs);
//method dequeue keluarkan antrian
void dequeue();
void cetakQueue();

//deklarasi queue dengan head dan tail
Node *head;
Node *tail;


int main(){

      Mahasiswa mhs1 = Mahasiswa\
                  ("Arifudding", 123);
      Mahasiswa mhs2 = Mahasiswa\
                  ("Salma Roshidin", 456);
      Mahasiswa mhs3 = Mahasiswa\
                              ("Joko Subando", 876);


      cetakQueue();
      enqueue(mhs1);
      cetakQueue();
      enqueue(mhs2);
      cetakQueue();
      enqueue(mhs3);
      cetakQueue();

      dequeue();
      cetakQueue();
      dequeue();
      cetakQueue();
      enqueue(Mahasiswa("Sugeng Yellor", 6789));
      cetakQueue();
      enqueue(Mahasiswa("Budi Sumantri", 870));
      cetakQueue();
      dequeue();
      cetakQueue();
      dequeue();
      cetakQueue();

      cout << endl;

      system("pause");
      return 0;
}
```

```
// add node as new tail
void enqueue(Mahasiswa mhs){
	Node * newQueueNode = new Node;
	newQueueNode->mahasiswa = mhs;
	newQueueNode->nextNode = nullptr;
	newQueueNode->prevNode = nullptr;
	if(head == NULL){
		head = newQueueNode;
		tail = NULL;
	}else if(tail == NULL){
		tail = newQueueNode;
		head->nextNode = tail;
		tail->prevNode = head;
	}else{
		Node *prevTailNode = tail;
		tail = newQueueNode;
		prevTailNode->nextNode = tail;
		tail->prevNode = prevTailNode;
	}
}

//Antrian keluar dari head
void dequeue(){

	if(head == NULL){
		cout << "Queue kosong!\n\n";
	}else{
		Node * delQueueNode = head;
		head = delQueueNode->nextNode;
		cout << "Memproses antrian keluar:"
			<< "Nama : "
			<< delQueueNode->mahasiswa.getNama()
			<< endl
			<< "NIM : "
			<< delQueueNode->mahasiswa.getNim()
			<< endl;
		delete delQueueNode;
	}
}

void cetakQueue(){
	int i = 0;
	cout << "Berikut daftar mahasiswa dalam Queue: \n";
	if(head == NULL){
		cout << "Queue kosong!\n\n";
	}else{
		Node *p = head;
		while(p != NULL){
			i++;
			cout << i << ". "
```

```
                        << "Nama : " << p->mahasiswa.getNama()
                        << "." << endl
                        << "    "
                         << "NIM : " << p->mahasiswa.getNim()
                         << "." << endl;
                    p = p->nextNode;
               }
          cout << endl << endl;
          }
}
```

c. Jalankan program

Tugas Laporan

1. Sebuah klinik kesehatan menerima pengunjung yang sangat ramai hampir setiap hari. Untuk mengatur antrian konsultasi pada 2 dokter berbeda yaitu dokter Yuan Lisna dan dokter Alex Jane, maka klinik berencana menggunakan suatu program komputer. Program ini akan mengatur antrian pengujung. Setiap kedatangan pengunjung akan direkam datanya pada sistem dan menerima nomor antrian. Setiap proses antrian yang mendapat giliran, maka progam akan menampilkan nama, nomor urut antrian dan dokter tujuan konsultasi. Dokter tujuan konsultasi dipilih oleh sistem berdasarkan kesediaan waktu dokter yang telah selesai dengan pasiennya.
2. Buatlah program untuk menyelesaikan kasus klinik kesehatan tersebut.

Modul Praktikum 5: List dan Iterator

Deskripsi Pertemuan

Praktikum lima akan memperkenalkan struktur List dan Iterator List dengan menggunakan struktur data yang telah dipelajari sebelumnya. Selanjutnya pada bab ini akan diperkenalkan penggunaan iterator pada beberapa tipe koleksi data.

Syarat Kompetensi

1. Memahami dan mampu menggunakan tipe data UDT/ADT dan tipe data koleksi.
2. Memahami dan mampu mengimplementasikan struktur array dalam pemrograman.
3. Memahmi dan mampu menggunakan struktur kendali/logika seleksi dalm program.
4. Memahami dan mampu mengimplementasika logika perulangan dalm program.

Standar Kompetensi

1. Mahasiswa mampu mengimplementasikan struktur list dan iterator dalam program.
2. Mahasiswa mampu menganalisis dan merancang program penyelesaian kasus menggunakan struktur list dan iterator.

Dasar Teori

Pada struktur array, penambahan data tidak boleh melebihi kapasitas array yang telah dideklarasikan. Jika tetap ingin menambah data pada koleksi array maka tidak ada jalan lain kecuali membuat array baru dengan kapasitas yang diinginkan kemudian menyalin array sebelumnya. Cara ini selain membutuhkan komputasi lebih juga tidak efektif dari sisi kode program.

Masalah lainnya dengan array adalah, jika ingin mengakses setiap elemen array maka cara standar yang telah diperkenalkan adalah melalui logika looping menggunakan

struktur for defenitif. Artinya kode program harus tahu berapa elemen array yang ada sehingga tepat diberlakukan perulangan. Jika tidak resiko error saat runtime akan terjadi dan memaksa program berhenti.

Oleh karena itu, struktur List dan Iterator digunakan untuk mengorganisasi koleksi data, mengakses dan memodifikasi elemen data tanpa harus tahun pasti berapa elemen dalam setiap List. Sebagai contoh perbandingan akan diperlihatkan koleksi data dengan akses elemen data tanpa List dan dengan menggunakan List dan Iterator. Perhatikan contoh berikut:

```
#include <iostream>
#include <vector>

using namespace std;

int main()
{
	// Declaring a vector
	vector<int> v;
	for(int x=10;x<=100;x+=10) v.push_back(x);

	// Declaring an iterator
	vector<int>::iterator i;

	int j;

	cout << "Tanpa iterator = ";

	// Akses elemen tanpa iterator
	for (j = 0; j < 10; ++j){
		cout << v[j] << " ";
	}

	cout << "\nDengan iterator = ";

	// Akses elemen dengan iterator
	for (i = v.begin(); i != v.end(); ++i) {
		cout << *i << " ";
	}

	// Tambahkan elemen ke vector
	v.push_back(12389);

	cout << "\nTanpa iterator = ";

	// Akses elemen tanpa iterator,
	//  elemen tambahan diluar indeks array
	for (j = 0; j < 10; ++j){
		cout << v[j] << " ";
```

```
	}

	cout << "\nDengan iterator = ";

	// Akses elemen dengan iterator
	for (i = v.begin(); i != v.end(); ++i){
		cout << *i << " ";
	}
	cout << endl;
	system("pause");
	return 0;
}
```

Pada contoh diatas, iterator menggunakan fungsi begin() dan end() untuk merujuk awal dan akhir data dalam koleksi data. Sementara itu perulangan dengan mengakses indeks data bersifat statis, sehingga penambahan data pada koleksi data tidak tercetak karena indeks koleksi data tidak diupdate pada kondisi perulangannya. Hal yang tidak terjadi pada looping menggunakan iterator.

Selain sifat dinamis tersebut, iterator dengan struktur List yang dipakai bersamaan memberikan beberapa variasi iterasi terhadap elemen-elemen objek data dalam list. Beberapa contoh penggunaan List dengan iterator untuk data primitif integer sebagai berikut:

```
#include <iostream>
#include <list>

using namespace std;

int main(){
	list<int> lst;
	int i;

	//inisialisasi elemen
	for(i=0; i<10; i++) lst.push_back(i);

	cout << "Size = " << lst.size() << endl;

	//cetak elemen list
	list<int>::iterator p = lst.begin();
	while(p != lst.end()) {
		cout << *p << endl;
		p++;
	}

   // Modifikasi nilai elemen
	p = lst.begin();
```

```
	while(p != lst.end()) {
		*p = *p + 100;
		p++;
	}

	// cetak elemen yg tlh diubah
	p = lst.begin();
	while(p != lst.end()) {
		cout << *p << " ";
		p++;
	}
	cout << endl;
	system("pause");
	return 0;
}
```

List dan iterator dapat juga digunakan untuk iterasi tipe data typedef. Perhatikan contoh berikut:

```
#include <iostream>
#include <list>

using namespace std;

// deklarasi tipe data
typedef list<int> LISTINT;
int main(void){
	LISTINT listOne;
	LISTINT::iterator i;
	listOne.push_front (2);
	listOne.push_front (1);
	listOne.push_back (3);
	for (i = listOne.begin(); i != listOne.end(); ++i)
		cout << *i << " ";
	cout << endl;
	for (i = listOne.end(); i != listOne.begin(); --i)
		cout << *i << " ";
	cout << endl;
	cout << endl;
	system("pause");
	return 0;
}
```

Selain for, penggunaan for loop dapat digunakan pula pada list dan iterator. Perhatikan contoh berikut:

```
#include <iostream>
#include <list>
#include <algorithm>

using namespace std;

void print (int elemen){
     cout << elemen << " ";
}
int main(){
    list<int> coll;
    // masukan elemen data 1-9
    for (int i=1; i<=9; ++i) {
        coll.push_back(i);
    }
    // cetak elemen
    for_each (coll.begin(), coll.end(),
              print);
    cout << endl;
    // cetak urutan terbalik-reverse
    for_each (coll.rbegin(), coll.rend(),
              print);
    cout << endl;
      cout << endl;
      system("pause");
      return 0;
}
```

Seperti pada vector, pengubahan nilai elemen dapat pula dilakukan pada struktur List menggunakan iterator. Perhatikan contoh berikut:

```
#include <iostream>
#include <list>

using namespace std;

int main(){
     list<int> lst; // deklarasi list
     int i;

     //inisialisasi nilai awal
     for(i=0; i<10; i++) lst.push_back(i);

     //cetak tampilan
     cout << "Size = " << lst.size() << endl;
     cout << "Elemen sekarang: ";
     list<int>::iterator p = lst.begin();
```

```
      while(p != lst.end()) {
            cout << *p << " ";
            p++;
      }
      cout << "\n\n";

      // ubah nilai elemen
      p = lst.begin();
      while(p != lst.end()) {
            *p = *p + 100;
            p++;
      }

      //cetak hasil perubahan
      cout << "Elemen setelah diubah: ";
      p = lst.begin();
      while(p != lst.end()) {
            cout << *p << " ";
            p++;
      }
    cout << endl;
      cout << endl;
      system("pause");
      return 0;
}
```

Contoh penggunaan list dan iterator diatas menggunakan data primitif sebagai elemen data. Untuk data objek seperti struct atau class maka terlebih dahulu struktur data objek harus dideklarasikan sebagai elemen List.

Tugas Pendahuluan

1. Buatlah sebuah List objek data dan Vector objeck data menggunakan struct sebagai elemen data
2. Lakukan operasi perulangan for_each dan reverse loop.

Langkah-Langkah Praktikum

1. List dan iterator dengan primitif type
   a. Buat file baru dengan nama praktikum51.cpp
   b. Ketik kode berikut ini:

```
#include <iostream>
#include <list>
using namespace std;

int main()
{
```

```
	// deklarasi list
	list<int> mylist;
	for(int i=1; i<6; i++) mylist.push_back(i);

	// gunakan begin() akses elemen pertama list
	for (auto it = mylist.begin(); \
			it != mylist.end(); ++it)
		cout << ' ' << *it;
	cout << endl;
	system("pause");
	return 0;
}
```

c. Jalankan program

2. List dan iterator dengan ADT.
   a. Buat file baru dengan nama praktikum52.cpp
   b. Ketik kode berikut:

```
#include <list>
#include <iostream>
#include <string>
#include <iterator>
#include <algorithm>

using namespace std;

struct Player
{
	int id;
	string name;
	Player(int playerId, string playerName) :
	id(playerId), name(playerName)
	{
	}
};
int main()
{
	Player arr[] = { Player(22, "Amel"), \
	Player(3, "Colleng"), Player(43, "Joko"), \
	Player(30,"Wiro"), Player(2, "Tunning") };
	list<Player> listofPlayers(begin(arr), end(arr));

	cout << "*******Iterators*******" << endl;
	list<Player>::iterator it;

	for (it = listofPlayers.begin(); \
		it != listofPlayers.end(); it++)
	{
		int id = it->id;
```

```
            string name = it->name;
            cout << id << " :: " << name << endl;
      }
      cout
            << "*******Lambda function *********"
            << endl;
      for_each(listofPlayers.begin(), listofPlayers.end(),
            [](const Player & player)
      {
            cout<<player.id<< " :: "<<player.name<<endl;
      });

      cout
            << "*******Reverse Iterators *********"
            << endl;
      list<Player>::reverse_iterator revIt;
      for (revIt = listofPlayers.rbegin(); \
            revIt != listofPlayers.rend(); revIt++)
      {
            int id = revIt->id;
            string name = revIt->name;
            cout << id << " :: " << name << endl;
      }
      cout << endl;
      system("pause");
      return 0;
}
```

c. Jalankan program

Tugas Laporan

1. Kantor dinas lingkungan hidup menerapkan sistem absensi online. Setiap bulan laporan kehadiran pegawai akan dirangking dan hasilnya akan ditampilkan sepuluh pegawai paing rajin dan sepuluh pegawai paling malas. Kriteria penilaiannya adalah rata-rata kehadiran individu dalam satu bulan kerja. Buatlah progam untuk menampilkan data pegawai-pegawai tersebut dari paling rajin dan paling malas serta persentasi kehadirannya.
2. Gunakan struktur List dan iterator serta iterator reverse untuk menampilkan data pegawai tersebut.

Modul Praktikum 6: Tree

Deskripsi Pertemuan

Praktikum ke enam akan meperkenalkan struktur tree atau pohon dalam mengorganisasikan data.

Syarat Kompetensi

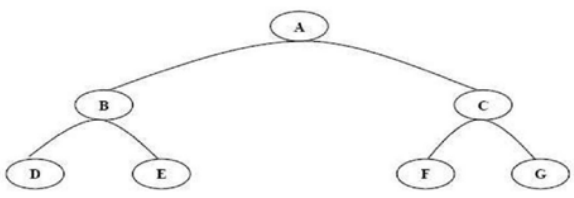
1. Memahami dan mampu mengimplementasikan struktur list dalam program.
2. Memahami dan mampu mengimplementasikan logika seleksi dalam program.
3. Memahami dan mampu mengimplementasikan logika perulangan dalam program.
4. Memahami teknik pengurutan data.

Standar Kompetensi

1. Mahasiswa mampu mengimplementasikan struktur tree dalam program.
2. Mahasiswa mampu menganalisis dan merancang program penyelesaian kasus menggunakan tree.

Dasar Teori

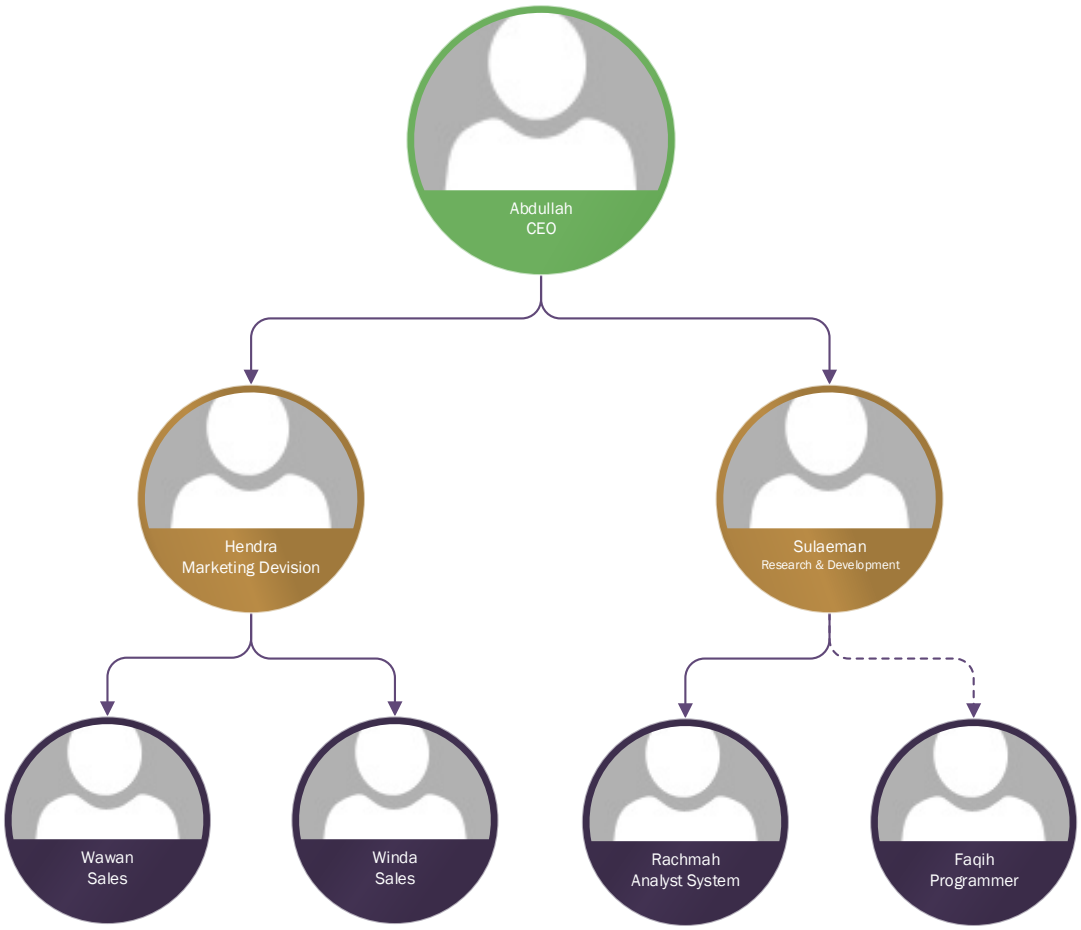
Tree adalah node-node data yang dihubungkan secara hirarki. Setiap node data kecuali node daun akan memiliki sedikitnya 1 hirarki node di bawahnya yang disebut node anak. Kemudian node-node anak itu pun memiliki lagi hirarki node anak dibawahnya sehingga membentuk rangkaian hirarki node dengan level 1, 2, 3 dan seterusnya.



Ilustrasi hirarki tree dengan 2 node anak

Ada beragam struktur data model tree yang dibedakan berdasarkan jumlah anak setiap node. Tree secara general adalah struktur hirarki data yang memiliki node root sebagai node awal kemudian juga memiliki hirarki node anak dibawahnya. Binary tree adalah hirarki struktur data dengan kriteria maksimal node anak setiap node adalah 2. Sedangkan Full binary tree adalah hirarki node yang setiap node parentnya pasti memiliki 2 node anak kecuali node daun.

Tree dapat diimplementasikan menggunakan double linked list. Perhatikan struktur tree berikut ini:



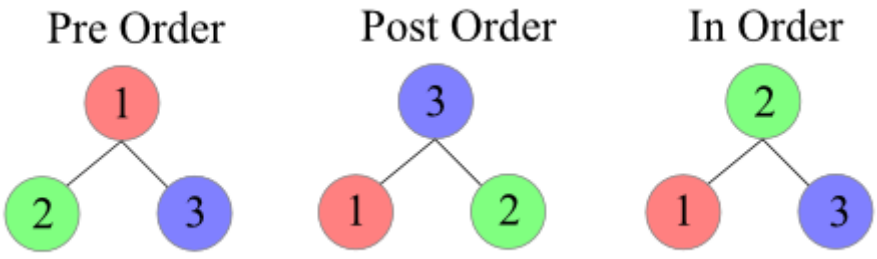
Tree diatas dapat diimplementasikan menggunakan double linked list seperti berikut:

```
#include <iostream>
#include <iomanip>
#include <string>

using namespace std;

//deklarasi struktur double linked node
struct Node{
      string nama;
      string posisi;
      Node * induk;
      Node * kiri;
```

```
      Node * kanan;

      Node(string nam, string pos){
            nama = nam;
            posisi = pos;
      }

};

//method cetak tree
void cetakTree(Node * node, string space);

//deklarasi root
Node *root;

int i = 0;

int main(){

      Node * root = new Node("Abdullah","CEO");
      Node * level_11 = new Node("Hendra","Marketing Devision");
      Node * level_12 = new Node("Sulaeman",\
                                 "Research & Development");
      Node * level_111 = new Node("Wawan","Sales");
      Node * level_112 = new Node("Winda","Sales");
      Node * level_121 = new Node("Rachmah","Analyst System");
      Node * level_122 = new Node("Faqih","Programmer");


      root->induk = nullptr;
      root->kiri = level_11;
      root->kanan = level_12;

      level_11->induk = root;
      level_11->kiri = level_111;
      level_11->kanan = level_112;

      level_12->induk = root;
      level_12->kiri = level_121;
      level_12->kanan = level_122;

      level_111->induk = level_11;
      level_111->kiri = nullptr;
      level_111->kanan = nullptr;

      level_112->induk = level_11;
      level_112->kiri = nullptr;
      level_112->kanan = nullptr;

      level_121->induk = level_12;
      level_121->kiri = nullptr;
```

```
	level_121->kanan = nullptr;

	level_122->induk = level_12;
	level_122->kiri = nullptr;
	level_122->kanan = nullptr;

	cout << "Berikut ini data dalam struktur tree:\n";
	cetakTree(root, "");

	cout << endl;

	system("pause");
	return 0;
}



void cetakTree(Node * node, string space){
	if(node != NULL){
		i++;
		cout << space << i << ". Node: \n"
			<< space << "Nama: " << node->nama << endl
			<< space << "Posisi: " << node->posisi
			<< endl << endl;
		cetakTree(node->kiri, space+"     ");
		cetakTree(node->kanan, space+"     ");
	}
}
```

Untuk mencetak elemen data setiap node maka terdapat 3 pendekatan yang bisa digunakan. Perhatikan opsi cetak struktur tree berikut:



Masing-masing pendekatan tersebut diimplementasikan sebagai berikut:

```
void cetakTreePreOrder(Node * node, string space){
	if(node != NULL){
		i++;
		cout << space << i << ". Node: \n"
			<< space << "Nama: " << node->nama << endl
			<< space << "Posisi: " << node->posisi
```

```
                << endl << endl;
            cetakTree(node->kiri, space+"     ");
            cetakTree(node->kanan, space+"     ");
      }
}
```

```
void cetakTreeInOrder(Node * node, string space){
      if(node != NULL){

            i++;
            cetakTree(node->kiri, space+"     ");
            cout << space << i << ". Node: \n"
                  << space << "Nama: " << node->nama << endl
                  << space << "Posisi: " << node->posisi
                  << endl << endl;
            cetakTree(node->kanan, space+"     ");

      }
}
```

```
void cetakTreePostOrder(Node * node, string space){
      if(node != NULL){

            i++;
            cetakTree(node->kiri, space+"     ");
            cetakTree(node->kanan, space+"     ");
            cout << space << i << ". Node: \n"
                  << space << "Nama: " << node->nama << endl
                  << space << "Posisi: " << node->posisi
                  << endl << endl;

      }
}
```

Logika cetak di atas berlaku juga dalam hal urutan akses node-node data dalam struktur data tree.

Tugas Pendahuluan

1. Buatlah sebuah struktur data tree yang dapat melakukan penambahan dan penghapusan node secara dinamis.
2. Buatlah struktur binary tree untuk mengorganisasikan kode morse. Gunakan referensi kode morse yang anda ketahui.

Langkah-Langkah Praktikum

1. Simple binary tree
   a. Kita akan membuat struktur pohon biner untuk menyimpan data dalambentuk struktur tree sederhana.
   b. Buat file baru dengan nama praktikum61.cpp
   c. Ketik kode berikut:

```
#include <stdio.h>
#include <iostream>

class Node
{
public:
    Node( int v )
    {
        data = v;
        left = 0;
        right = 0;
    }

    int data;
    Node* left;
    Node* right;
};

void Add( Node** root, Node* n )
{
    if ( !*root  )
    {
        *root = n;
        return;
    }

    if ( (*root)->data < n->data )
    {
        Add( &(*root)->right, n );
    }
    else
    {
        Add( &(*root)->left, n );
    }
}

void Print( Node* node )
{
    if ( !node )  return;
    Print( node->left );
    printf( "value = %i\n", node->data );
    Print( node->right );
}
```

```
int main()
{
    Node* root = 0;

    Add( &root, new Node( 1 ) );
    Add( &root, new Node( 2 ) );
   Add( &root, new Node( -1 ) );
    Add( &root, new Node( 12 ) );

    Print( root );
      system("pause");
    return 0;
}
```

d. Jalankan program

2. Compelete binary tree
   a. Buat file baru dengan nama praktikum62.cpp
   b. Ketik kode berikut:

```
#include <iostream>

using namespace std;


struct node{
      int value;
      node *left;
      node *right;
};

class btree{
public:
      btree();
      ~btree();

      void insert(int key);
      node *search(int key);
      void destroy_tree();
      void inorder_print();
      void postorder_print();
      void preorder_print();

private:
      void destroy_tree(node *leaf);
      void insert(int key, node *leaf);
      node *search(int key, node *leaf);
      void inorder_print(node *leaf);
      void postorder_print(node *leaf);
```

```
      void preorder_print(node *leaf);

      node *root;
};


btree::btree(){
      root = NULL;
}

btree::~btree(){
      destroy_tree();
}

void btree::destroy_tree(node *leaf){
      if(leaf != NULL){
            destroy_tree(leaf->left);
            destroy_tree(leaf->right);
            delete leaf;
      }
}

void btree::insert(int key, node *leaf){

      if(key < leaf->value){
            if(leaf->left != NULL){
                  insert(key, leaf->left);
            }else{
                  leaf->left = new node;
                  leaf->left->value = key;
                  leaf->left->left = NULL;
                  leaf->left->right = NULL;
            }
      }else if(key >= leaf->value){
            if(leaf->right != NULL){
                  insert(key, leaf->right);
            }else{
                  leaf->right = new node;
                  leaf->right->value = key;
                  leaf->right->right = NULL;
                  leaf->right->left = NULL;
            }
      }

}

void btree::insert(int key){
      if(root != NULL){
            insert(key, root);
      }else{
            root = new node;
```

```
            root->value = key;
            root->left = NULL;
            root->right = NULL;
      }
}

node *btree::search(int key, node *leaf){
      if(leaf != NULL){
            if(key == leaf->value){
                  return leaf;
            }
            if(key < leaf->value){
                  return search(key, leaf->left);
            }else{
                  return search(key, leaf->right);
            }
      }else{
            return NULL;
      }
}

node *btree::search(int key){
      return search(key, root);
}

void btree::destroy_tree(){
      destroy_tree(root);
}

void btree::inorder_print(){
      inorder_print(root);
      cout << "\n";
}

void btree::inorder_print(node *leaf){
      if(leaf != NULL){
            inorder_print(leaf->left);
            cout << leaf->value << ",";
            inorder_print(leaf->right);
      }
}

void btree::postorder_print(){
      postorder_print(root);
      cout << "\n";
}

void btree::postorder_print(node *leaf){
      if(leaf != NULL){
            inorder_print(leaf->left);
            inorder_print(leaf->right);
```

```
            cout << leaf->value << ",";
      }
}

void btree::preorder_print(){
      preorder_print(root);
      cout << "\n";
}

void btree::preorder_print(node *leaf){
      if(leaf != NULL){
            cout << leaf->value << ",";
            inorder_print(leaf->left);
            inorder_print(leaf->right);
      }
}

int main(){

      //btree tree;
      btree *tree = new btree();

      tree->insert(10);
      tree->insert(6);
      tree->insert(14);
      tree->insert(5);
      tree->insert(8);
      tree->insert(11);
      tree->insert(18);

      tree->preorder_print();
      tree->inorder_print();
      tree->postorder_print();

      delete tree;
      system("pause");
    return 0;
}
```

c. Jalankan program.

Tugas Laporan

1. Sebuah bank memiliki daftar nasabah dan jumlah saldo yang diurut berdasarkan jumlah saldo yang selalu diundi setiap bulan untuk undian berhadiah untuk 4 orang nasabah. 2 orang dipilih dari yang paling tinggi saldo dan yang paling rendah. Karena itu pihak bank sangat membutuhkan koleksi data yang selalu terbaharukan.
2. Susunlah data nasabah bank tersebut dalam bentuk struktur binary tree yang memudahkan operasi pencarian.

Modul Praktikum 7: Heaps dan Priority Queue

Deskripsi Pertemuan

Praktikum ke tujuh akan memperkenalkan struktur heaps dan antrian prioritas.

Syarat Kompetensi

1. Memahami dan mampu mengimplementasikan struktur list dalam program.
2. Memahami dan mampu mengimplementasikan logika seleksi dalam program.
3. Memahami dan mampu mengimplementasikan logika perulangan dalam program.
4. Memahami stuktur antrian dalam program.

Standar Kompetensi

1. Mahasiswa mampu mengimplementasikan struktur heaps dan queue prioritas dalam program.
2. Mahasiswa mampu menganalisis dan merancang program penyelesaian kasus menggunakan heaps dan queue prioritas.

Dasar Teori

Priority queue adalah sistem antrian yang berdasarkan pada prioritas. Berbeda dengan antrian biasa (queue), tipe antrian ini tidak menganut prinsip FIFO atau pertama masuk pertama dilayani. Sebaliknya, setiap node dalam struktur data akan diproses terlebih dahulu sesuai dengan prioritas node tersebut.

Dalam kehidupan nyata, model antrian seperti ini dapat pula kita temui. Misalnya antrian pelayanan rumah sakit tentu akan memprioritaskan alias mendahulukan pelayanan kesehatan bagi pasien darurat meskipun yang bersangkutan datang belakangan. Demikian juga seorang pasien yang datang dan hendak melahirkan akan lebih diperhatikan dulu untuk mendapatkan pelayanan dan seterusnya.

Dalam struktur data, prioritas ditentukan berdasarkan keperluan dan programmer saat membentuk struktur data. Prioritas bisa ditandai dengan bilangan integer yang terurut dari angkan minimum ke angka maksimun yang menunjukan prioritas data tersebut.

Antrian prioritas paling mudah dibentuk dengan menggunakan array koleksi dari struct data prioritas. Misalnya sebuah struktur data disusun dengan prioritas berikut:

```
struct Node {
   string nama;
   int priority;
}
```

Selanjutnya kita dapat mendeklarasikan sebuah array yang berisi elemen-elemen Node dengan penanda prioritas berbeda. Dalam hal ini array berfungsi sebagai container antrian dan elemen node didalamnya adalah antrian data. Pada operasi **dequeue()** atau **pop()**, elemen akan dikeluarkan dari array menurut nilai priority dari node bukan dari indeks posisinya dalam array. Elemen array yang telah dikeluarkan akan diset menjadi NULL.

Sebaliknya dalam operasi **enqueue()** atau **push()**, elemen ditambahkan pada posisi mana saja dalam array selama elemen array tersebut masih kosong atau bernilai NULL.

Selain array, struct linked list yang telah dipraktekan pada pertemuan sebelumnya juga bisa dimodifikasi menjadi stuktur antrian prioritas. Hal yang perlu dilakukan hanya menambahkan atribut **priority** dalam struct Node seperti contoh berikut:

```
struct Node{
      string nama;
     int priority;
      Node * nextNode;
      Node * prevNode;
};
```

Hal yang perlu diperhatikan dalam memodifikasi program sebelumnya menjadi antrian prioritas adalah pada operasi **dequeue()** atau **pop()**. Operasi tersebut tidak lagi mengeluarkan array dari head antrian, tapi harus memeriksa setiap Node dalam linked list dan mengeluarkan alias menghapus node dengan nilai priority tertinggi.

Meski bisa menggunakan array dan linked list dalam membuat struktur antrian prioritas, tapi cara yang paling baik adalah dengan menggunakan heap. Heap adalah struktur data tree dengan penanda min atau max. Bahkan heap disebut antrian prioritas itu sendiri. Perhatikan implementasi antrian prioritas berikut menggunakan struktur heap-max:

```
#include <iostream>
#include <queue>


using namespace std;


void showpq(priority_queue <int> gq)
{
    priority_queue <int> g = gq;
    while (!g.empty())
    {
        cout << '\t' << g.top();
        g.pop();
    }
    cout << '\n';
}

int main ()
{

    priority_queue <int> gquiz;
    gquiz.push(10);
    gquiz.push(30);
    gquiz.push(20);
    gquiz.push(5);
    gquiz.push(1);

    cout << "The priority queue gquiz is : ";
    showpq(gquiz);

    cout << "\ngquiz.size() : " << gquiz.size();
    cout << "\ngquiz.top() : " << gquiz.top();


    cout << "\ngquiz.pop() : ";
    gquiz.pop();
    showpq(gquiz);

      cout << endl;
      system("pause");
      return 0;
}
```

Pada contoh antrian MAX diatas, elemen MAX akan selalu dikeluarkan lebih dahulu saat operasi **pop()**, meskipun data tersebut bukan yang pertama di **push()** kedalam antrian.

Selanjutnya perhatikan contoh dengan heap-MIN berikut ini:

```
// program  min heap
#include <iostream>
#include <queue>


using namespace std;

void showpq(priority_queue <int, vector<int>, greater<int>> gq)
{
    priority_queue <int, vector<int>, greater<int>> g = gq;
    while (!g.empty())
    {
        cout << '\t' << g.top();
        g.pop();
    }
    cout << '\n';
}

int main ()
{

    priority_queue <int, vector<int>, greater<int>> gquiz;
    gquiz.push(10);
    gquiz.push(30);
    gquiz.push(20);
    gquiz.push(5);
    gquiz.push(1);

    cout << "The priority queue gquiz is : ";
    showpq(gquiz);

    cout << "\ngquiz.size() : " << gquiz.size();
    cout << "\ngquiz.top() : " << gquiz.top();


    cout << "\ngquiz.pop() : ";
    gquiz.pop();
    showpq(gquiz);

      cout << endl;
      system("pause");
      return 0;
}
```

Sama seperti pada contoh pertama, antrian prioritas MIN akan mengeluarkan nilai sesuai urutan MIN pada saat operasi **pop()** meskipun data tersebut bukan yang pertama masuk kedalam struktur antrian.

Tugas Pendahuluan

1. Buatlah sebuah antrian prioritas dengan menggunakan pendekatan array sebagai kontainer antrian.
2. Pelajari antrian pada praktikum pertemuan sebelumnya kemudian kembangkanlah program antrian tersebut menjadi antrian prioritas.
3. Buatlah program antrian Deque dengan prioritas.

Langkah-Langkah Praktikum

1. Antrian prioritas dengan UDT struct menggunakan STL
   a. Buat file baru dengan nama praktikum71.cpp
   b. Ketik kode berikut:

```
// program priority_queue dgn structure

#include <iostream>
#include <queue>
using namespace std;
#define ROW 5
#define COL 2

struct Person {

	int age;

	float height;

	Person(int age, float height)
		: age(age), height(height)
	{
	}
};

struct CompareHeight {
	bool operator()(Person const& p1, Person const& p2)
	{
		return p1.height < p2.height;
	}
};

int main()
{
	priority_queue<Person, vector<Person>, \
			CompareHeight> Q;

	float arr[ROW][COL] = { { 30, 5.5 }, { 25, 5 },
				{ 20, 6 }, { 33, 6.1 },
				{ 23, 5.6 } };
```

```
	for (int i = 0; i < ROW; ++i) {

		Q.push(Person(arr[i][0], arr[i][1]));

	}

	while (!Q.empty()) {
		Person p = Q.top();
		Q.pop();
		cout << p.age << " " << p.height << "\n";
	}
	system("pause");
	return 0;
}
```

c. Jalankan program
d. Lakukan perubahan kode dan pelajari outputnya saat dijalankan

2. Antrian prioritas dengan ADT class menggunakan STL
   a. Buat file baru dengan nama praktikum72.cpp
   b. Ketik kode berikut:

```
// program priority_queue dgn class
#include <iostream>
#include <queue>
using namespace std;

#define ROW 5
#define COL 2

class Person {

public:
	int age;

	float height;

	Person(int age, float height)
		: age(age), height(height)
	{
	}
};

bool operator<(const Person& p1, const Person& p2)
{

	return p1.height < p2.height;
}
```

```
int main()
{

	priority_queue<Person> Q;

	float arr[ROW][COL] = { { 30, 5.5 }, { 25, 5 },
				{ 20, 6 }, { 33, 6.1 }, { 23, 5.6 } };

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

		Q.push(Person(arr[i][0], arr[i][1]));

	}

	while (!Q.empty()) {

		Person p = Q.top();

		Q.pop();

		cout << p.age << " " << p.height << "\n";
	}
	cout << endl;
	system("pause");
	return 0;
}
```

c. Jalankan program
d. Lakukan percobaan perubahan beberapa kode dan pelajari hasilnya.

Tugas Laporan

1. Sebuah klinik kesehatan menerapkan sistem antrian prioritas untuk melayani konsultasi pasien pada dokter. Prioritas pelayanan pasien ditandai dengan bilangan angka yang ditempelkan pada karcis antrian. Kode nomor yang paling besar akan memiliki prioritas paling tinggi. Setiap kali pemanggilan pasien, maka petugas staf klinik akan memanggil pasien dan nomor antriannya. Nomor antrian ini berbeda dengan nomor prioritas antrian. Keduanya tercatat pada karcis antrian.
2. Buatlah program untuk membantu sistem antrian klinik tersebut.

Modul Praktikum 8: Hash Table

Deskripsi Pertemuan

Praktikum 8 akan mempelajari struktur Hash Table dan cara mengorganisasikan data menggunakan hash table.

Syarat Kompetensi

1. Memahami dan mampu mengimplementasikan struktur list dalam program.
2. Memahami dan mampu mengimplementasikan logika seleksi dalam program.
3. Memahami dan mampu mengimplementasikan logika perulangan dalam program.
4. Memahami algoritma hashing code.

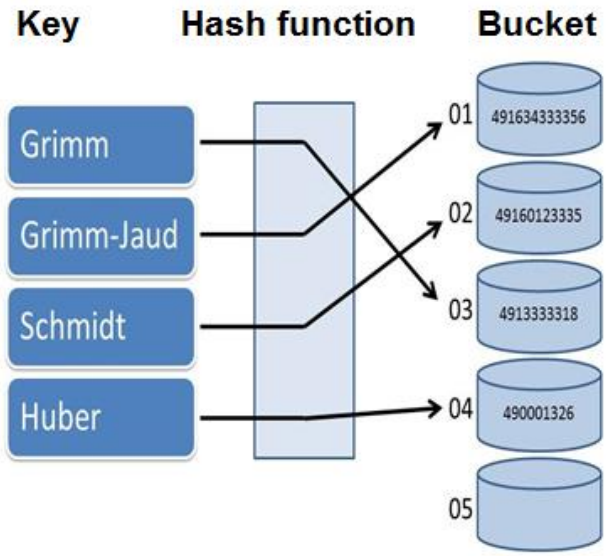
Standar Kompetensi

1. Mahasiswa mampu mengimplementasikan struktur hash tabel dalam program.
2. Mahasiswa mampu menganalisis dan merancang program penyelesaian kasus menggunakan hash tabel.

Dasar Teori

Hash table adalah sebuah struktur data yang memetakan data kedalam tabel key-value dengan menggunakan fungsi hashing. Fungsi hash bisa menggunakan banyak alternatif sepanjang dapat digunakan untuk memetakan key dan value kedalam suatu bucket atau hash table.

Berbeda dengan array yang menggunakan indeks sebagai key untuk mengakses value, struktur hash tabel bisa menggunakan key apa saja sepanjang dapat dipetakan melalui fungsi hash. Key bisa berupa string yang dilewatkan pada fungsi hash. Fungsi hash ini yang mengembalikan hash value yang akan disimpan dalam bucket. Setiap hash value akan menunjuk pada data value yang ikut disimpan pada saat operasi insert data. Perhatikan gambar ilustrasi fungsi hash berikut:



Dari gambar, untuk membuat sebuah struktur hash table, maka harus tersedia fungsi hash yang akan memproses key dengan output hash value. Pada contoh, key '**Grimm**' akan menghasilkan hash value '**03**' sehingga data value '**490001326**' disimpan pada bucket atau container urutan ke 3.

Dengan cara yang sama kita bisa membuat struktur hash table sederhanan dengan menggunakan array sebagai bucket atau container, fungsi panjang karakter key sebagai fungsi hash. Perhatikan contoh:

```
#include <iostream>
#include <string>

using namespace std;

struct HashTable{
      string alamat[100];

      void insert(string key, string value){
            int index = hashFunction(key);
            alamat[index] = value;
      }

      string getData(string key){
            int index = hashFunction(key);
            return alamat[index];
      }

      int hashFunction(string key){
            return key.length();
      }
};


int main(){
```

```
      HashTable hashTable;

      hashTable.insert("Acox","Jl. Toddopuli 4 No 2");
      hashTable.insert("Wawan","Jl. Perintis Kemerdekaan VII No 14");

      cout << "\nAlamat Acox : " << hashTable.getData("Acox");
      cout << "\nAlamat Wawan : " << hashTable.getData("Wawan");

      cout << endl;
      system("pause");
      return 0;
}
```

Hashtable menggunakan array seperti contoh adalah struktur yang sederhana. Selain sederhana, strukturnya juga bersifat statis dan tidak mengimplementasikan solusi untuk hash value yang sama atau solusi atas collision hash value.

Salah satu kesulitan mengorganisasi struktur data dengan fungsi hash adalah ketidak pastian keunikan hasil fungsi hash. Dua atau lebih string key bisa jadi menghasilkan hash value yang sama. Konskuensinya dua data value yang sama akan dirujuk oleh satu lokasi bucket yang sama jika tidak ditangani secara benar. Kondisi ini disebut sebagai collision atau tabrakan.

Ada beberapa metode yang telah dikembangkan untuk mengatasi collision hash. Salah satunya dengan mengganti lokasi penyimpanan data dengan memajukan lokasinya dari lokasi hash value aktual sepanjang lokasi data tersebut masih kosong.

Pada contoh program dengan Hash Table dibawah ini, fungsi hash modulus digunakan untuk memetakan key kedalam sebuah array container:

```
#include<iostream>
#include<cstdlib>
#include<string>
#include<cstdio>

using namespace std;

const int T_S = 200;

class HashTableEntry {
   public:
      int k;
      int v;
      HashTableEntry(int k, int v) {
         this->k= k;
         this->v = v;
      }
};
```

```
class HashMapTable {
   private:
      HashTableEntry **t;
   public:
      HashMapTable() {
         t = new HashTableEntry * [T_S];
         for (int i = 0; i< T_S; i++) {
            t[i] = NULL;
         }
      }
      int HashFunc(int k) {
         return k % T_S;
      }
      void Insert(int k, int v) {
         int h = HashFunc(k);
         while (t[h] != NULL && t[h]->k != k) {
            h = HashFunc(h + 1);
         }
         if (t[h] != NULL)
            delete t[h];
         t[h] = new HashTableEntry(k, v);
      }
      int SearchKey(int k) {
         int h = HashFunc(k);
         while (t[h] != NULL && t[h]->k != k) {
            h = HashFunc(h + 1);
         }
         if (t[h] == NULL)
            return -1;
         else
            return t[h]->v;
      }
     void Remove(int k) {
         int h = HashFunc(k);
         while (t[h] != NULL) {
            if (t[h]->k == k)
               break;
            h = HashFunc(h + 1);
         }
         if (t[h] == NULL) {
            cout<<"No Element found at key "<<k<<endl;
            return;
         } else {
            delete t[h];
         }
         cout<<"Element Deleted"<<endl;
      }
      ~HashMapTable() {
         for (int i = 0; i < T_S; i++) {
            if (t[i] != NULL)
               delete t[i];
```

```
                delete[] t;
            }
        }
};
int main() {
   HashMapTable hash;
   int k, v;
   int c;
   while (1) {
      cout<<"1.Insert element into the table"<<endl;
      cout<<"2.Search element from the key"<<endl;
      cout<<"3.Delete element at a key"<<endl;
      cout<<"4.Exit"<<endl;
      cout<<"Enter your choice: ";
      cin>>c;
      switch(c) {
         case 1:
            cout<<"Enter element to be inserted: ";
            cin>>v;
            cout<<"Enter key at which element to be inserted: ";
            cin>>k;
            hash.Insert(k, v);
         break;
         case 2:
            cout<<"Enter key of the element to be searched: ";
            cin>>k;
            if (hash.SearchKey(k) == -1) {
               cout<<"No element found at key "<<k<<endl;
               continue;
            } else {
               cout<<"Element at key "<<k<<" : ";
            cout<<hash.SearchKey(k)<<endl;
            }
         break;
         case 3:
            cout<<"Enter key of the element to be deleted: ";
            cin>>k;
            hash.Remove(k);
         break;
         case 4:
            exit(1);
         default:
            cout<<"\nEnter correct option\n";
      }
   }
      cout << endl;
      system("pause");
      return 0;
}
```

Pada contoh diatas, baik key maupun value menggunakan data integer. Tentu saja data string atau tipe data lain dapat digunakan. Demikian juga fungsi hashing selain modulus juga bisa kita gunakan sepanjang dapat memetakan key-value kedalam container data secara tepat.

Tugas Pendahuluan

1. Buatlah sebuah struktur data hash table menggunakan fungsi lain selain modulus untuk memetakan key-value.
2. Buatlah sebuah struktur data hash table untuk memetakan data mahasiswa dan alamat mahasiswa. Gunakan array sebagai container data.

Langkah-Langkah Praktikum

1. Hash table dengan heaps
   a. Buatlah sebuah file baru dengan nama praktikum81.cpp
   b. Ketik kode berikut:

```
/*
 * Program Hash Tables Chaining with List Heads
 */
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <cmath>
#include <vector>
#include <cstdlib>
using namespace std;
const int TABLE_SIZE = 5;
/*
 * Link List Class Declaration
 */
class LinkedHash
{
      public:
         int key, value;
         LinkedHash *next;
         LinkedHash(int key, int value)
         {
             this->key = key;
             this->value = value;
             this->next = NULL;
         }
};
/*
 * HashMap Class Declaration
```

```
 */
class HashMap
{
    private:
        LinkedHash **htable;
    public:
        HashMap()
        {
            htable = new LinkedHash*[TABLE_SIZE];
            for (int i = 0; i < TABLE_SIZE; i++)
            {
                htable[i] = NULL;
            }
        }
        ~HashMap()
        {
            for (int i = 0; i < TABLE_SIZE; i++)
            {
                if (htable[i] != NULL)
                {
                    LinkedHash *prev = NULL;
                    LinkedHash *entry = htable[i];
                    while (entry != NULL)
                    {
                        prev = entry;
                        entry = entry->next;
                        delete prev;
                    }
                }
                delete[] htable;
            }
        }
        /*
           * Hash Function
         */
        int HashFunc(int key)
           {
            return key % TABLE_SIZE;
        }
        /*
           * Insert Element at a key
         */
        void Insert(int key, int value)
        {
            int hash_val = HashFunc(key);
            if (htable[hash_val] == NULL)
                htable[hash_val] = new LinkedHash(key,
value);
            else
            {
                LinkedHash *entry = htable[hash_val];
```

```
            while (entry->next != NULL)
                entry = entry->next;
                if (entry->key == key)
                    entry->value = value;
                else
                    entry->next = new LinkedHash(key, \
                                  value);
        }
    }
    /*
       * Search Element at a key
     */
    int Find(int key)
    {
        int hash_val = HashFunc(key);
        if (htable[hash_val] == NULL)
            return -1;
        else
        {
            LinkedHash *entry = htable[hash_val];
            while (entry != NULL && entry->key != key)
            entry = entry->next;
            if (entry == NULL)
                return -1;
            else
                return entry->value;
        }
    }
    /*
       * Delete Element at a key
     */
    void Delete(int key)
    {
        int hash_val = HashFunc(key);
        if (htable[hash_val] != NULL)
        {
            LinkedHash *entry = htable[hash_val];
            LinkedHash *prev = NULL;
            while (entry->next != NULL && \
                   entry->key != key)
            {
                prev = entry;
                entry = entry->next;
            }
            if (entry->key == key)
            {
                if (prev == NULL)
                {
                    LinkedHash *next = entry->next;
                    delete entry;
                    htable[hash_val] = next;
```

```
                    }
                    else
                    {
                        LinkedHash *next = entry->next;
                        delete entry;
                        prev->next = next;
                    }
                }
            }
        }
};
/*
 * Main Contains Menu
 */
int main()
{
    HashMap hash;
    int key, value;
    int choice;
    while(1)
    {
      cout<<"\n----------------------"<<endl;
        cout<<"Operations on Hash Table"<<endl;
        cout<<"\n----------------------"<<endl;
        cout<<"1.Insert element into the table"<<endl;
        cout<<"2.Search element from the key"<<endl;
        cout<<"3.Delete element at a key"<<endl;
        cout<<"4.Exit"<<endl;
        cout<<"Enter your choice: ";
        cin>>choice;
        switch(choice)
        {
        case 1:
            cout<<"Enter element to be inserted: ";
            cin>>value;
     cout<<"Enter key at which element to be inserted: ";
            cin>>key;
            hash.Insert(key, value);
            break;
        case 2:
        cout<<"Enter key of the element to be searched: ";
            cin>>key;
            if (hash.Find(key) == -1)
            cout<<"No element found at key "<<key<<endl;
            else
            {
                cout<<"Elements at key "<<key<<" : ";
                cout<<hash.Find(key)<<endl;
            }
            break;
        case 3:
```

```
            cout<<"Enter key of the element to be deleted: ";
              cin>>key;
              if (hash.Find(key) == -1)
                  cout<<"Key "<<key
               <<" is empty, nothing to delete"<<endl;
              else
              {
                  hash.Delete(key);
                  cout<<"Entry Deleted"<<endl;
              }
              break;
          case 4:
              exit(1);
          default:
             cout<<"\nEnter correct option\n";
        }
    }
      cout << endl;
      system("pause");
    return 0;
}
```

c. Jalankan program
d. Lakukan simulasi interaksi dengan program

2. Hash Table dengan doubly linked list
   a. Buat file baru dengan nama praktikum82.cpp
   b. Ketik kode berikut:

```
/*
 * Program Hash Tables Chaining with
 * Doubly Linked Lists
 */
#include <iostream>
#include <string>
#include <vector>
#include <cstdlib>
const int TABLE_SIZE = 25;
using namespace std;
/*
 * Node Declaration
 */
struct HashNode
{
    int data, key;
    HashNode *next;
    HashNode *prev;
};
/*
 * Class Declaration
```

```
 */
class HashMap
{
    public:
        HashNode **htable, **top;
        HashMap()
        {
            htable = new HashNode*[TABLE_SIZE];
            top = new HashNode*[TABLE_SIZE];
            for (int i = 0; i < TABLE_SIZE; i++)
            {
                htable[i] = NULL;
                top[i] = NULL;
            }
        }
        ~HashMap()
        {
            delete [] htable;
        }

        /*
         * Hash Function
         */
        int HashFunc(int key)
        {
            return key % TABLE_SIZE;
        }

        /*
       * Insert Element at a key
         */
        void insert(int key, int value)
        {
            int hash_val = HashFunc(key);
            HashNode *entry = htable[hash_val];
            if (entry == NULL)
            {
                entry = new HashNode;
                entry->data = value;
                entry->key = key;
                entry->next = NULL;
                entry->prev = NULL;
                htable[hash_val] = entry;
                top[hash_val] = entry;
            }
            else
            {
                while (entry != NULL)
                    entry = entry->next;
                entry = new HashNode;
                entry->data = value;
```

```
                entry->key = key;
                entry->next = NULL;
                entry->prev = top[hash_val];
                top[hash_val]->next = entry;
                top[hash_val] = entry;
            }
        }

        /*
         * Remove Element at a key
         */
        void remove(int key)
        {
            int hash_val = HashFunc(key);
            HashNode *entry = htable[hash_val];
            if (entry->key != key || entry == NULL)
            {
                cout<<"No Element found at key:
"<<key<<endl;
                return;
            }
            while (entry != NULL)
            {
                if (entry->next == NULL)
                {
                    if (entry->prev == NULL)
                    {
                        htable[hash_val] = NULL;
                   top[hash_val] = NULL;
                        delete entry;
                        break;
                    }
                    else
                    {
                        top[hash_val] = entry->prev;
                        top[hash_val]->next = NULL;
                        delete entry;
                        entry = top[hash_val];
                    }
                }
                entry = entry->next;
            }
        }
        /*
         * Search Element at a key
         */
        void get(int key)
        {
            int hash_val = HashFunc(key);
            bool flag = false;
            HashNode* entry = htable[hash_val];
```

```
            if (entry != NULL)
            {
                while (entry != NULL)
                {
                    if (entry->key == key)
                    {
                        flag = true;
                    }
                    if (flag)
                    {
                        cout<<"Element found at key
"<<key<<": ";
                        cout<<entry->data<<endl;
                    }
                    entry = entry->next;
                }
            }
            if (!flag)
                cout<<"No Element found at key "<<key<<endl;
        }
};

/*
 * Main Contains Menu
*/
int main()
{
   HashMap hash;
    int key, value;
    int choice;
    while (1)
    {
        cout<<"\n----------------------"<<endl;
        cout<<"Operations on Hash Table"<<endl;
        cout<<"\n----------------------"<<endl;
        cout<<"1.Insert element into the table"<<endl;
        cout<<"2.Search element from the key"<<endl;
        cout<<"3.Delete element at a key"<<endl;
        cout<<"4.Exit"<<endl;
        cout<<"Enter your choice: ";
        cin>>choice;
        switch(choice)
        {
        case 1:
            cout<<"Enter element to be inserted: ";
            cin>>value;
        cout<<"Enter key at which element to be inserted: ";
            cin>>key;
            hash.insert(key, value);
            break;
        case 2:
```

```
            cout<<"Enter key of the element to be searched: ";
              cin>>key;
              hash.get(key);
              break;
          case 3:
            cout<<"Enter key of the element to be deleted: ";
              cin>>key;
              hash.remove(key);
              break;
          case 4:
              exit(1);
          default:
             cout<<"\nEnter correct option\n";
          }
      }
        cout << endl;
        system("pause");
      return 0;
}
```

c. Jalankan program
d. Lakukan simulasi data

Tugas Laporan

1. Buatlah sebuah program interaktif struktur data mahasiswa dan data-data mahasiswa seperti nama, nim, alamat, angkatan menggunakan hash table dengan doubly linked list.
2. Implementasikan ulang soal nomor 1 dengan hash table menggunakan singly linked list.

Modul Praktikum 9: Map dan Skip List

Deskripsi Pertemuan

Praktikum 9 akan membahas dan memperkanalkan cara menyususn struktur map dan skip list.

Syarat Kompetensi

1. Memahami dan mampu mengimplementasikan struktur list dalam program.
2. Memahami dan mampu mengimplementasikan logika seleksi dalam program.
3. Memahami dan mampu mengimplementasikan logika perulangan dalam program.

Standar Kompetensi

1. Mahasiswa mampu mengimplementasikan struktur Map dan Skip List dalam program.
2. Mahasiswa mampu menganalisis dan merancang program penyelesaian kasus menggunakan Map dan Skip List.

Dasar Teori

Map adalah struktur data yang memetakan KEY-VALUE secara berpasangan. Map tidak memungkinkan adanya 2 VALUE yang memiliki KEY yang sama. Map secara internal akan menyimpan data secara urut.

Map adalah bagian dari Standard Template Library (STL). Seperti List yang telah diperkenalkan pada praktikum sebelumnya, iterasi untuk mengakses elemen-elemen pasangan map dilakukan dengan menggunakan iterator. Map menyediakan fungsi-fungsi dasar seperti **begin()**, **end()**, **size()** dan beberapa fungsi standar lain.

```
#include <iostream>
#include <iterator>
#include <map>
```

```
using namespace std;

int main(){

	// deklarasi map container
	map<int, int> map1;

	// insert elemen secara acak
	map1.insert(pair<int, int>(1, 40));
	map1.insert(pair<int, int>(2, 30));
	map1.insert(pair<int, int>(3, 60));
	map1.insert(pair<int, int>(4, 20));
	map1.insert(pair<int, int>(5, 50));
	map1.insert(pair<int, int>(6, 50));
	map1.insert(pair<int, int>(7, 10));

	// cetak map1
	map<int, int>::iterator itr;
	cout << "\nBerikut isi map1 : \n";
	cout << "\tKEY\tELEMENT\n";
	for (itr = map1.begin(); itr != map1.end(); ++itr) {
		cout << '\t' << itr->first
			<< '\t' << itr->second << '\n';
	}
	cout << endl;

	// salin elemen dari map1 ke map2
	map<int, int> map2(map1.begin(), map1.end());

	// cetak elemen map2
	cout << "\nHasil salin ke map2"
		<< " dari map1 : \n";
	cout << "\tKEY\tELEMEN\n";
	for (itr = map2.begin(); itr != map2.end(); ++itr) {
		cout << '\t' << itr->first
			<< '\t' << itr->second << '\n';
	}
	cout << endl;

	// Hapus elemen dari map
	// elemen dengan key=3 dalam map2
	map2.erase(map2.begin(), map2.find(3));

	cout << "\nmap2 setelah penghapusan"
			" elemen key=1-3 : \n";
	cout << "\tKEY\tELEMENT\n";

	for (itr = map2.begin(); itr != map2.end(); ++itr) {
		cout << '\t' << itr->first
			<< '\t' << itr->second << '\n';
	}
```

```
      // hapus semua elemen dengan key = 4
      int num;
      num = map2.erase(4);
      cout << "\nmap2.erase(4) : ";
      cout << num << " dihapus \n";
      cout << "\tKEY\tELEMENT\n";
      for (itr = map2.begin(); itr != map2.end(); ++itr) {
            cout << '\t' << itr->first
                  << '\t' << itr->second << '\n';
      }

      cout << endl;

      // lower bound dan upper bound untuk map1 key = 5
      cout << "map1.lower_bound(5) : "
            << "\tKEY = ";
      cout << map1.lower_bound(5)->first << '\t';
      cout << "\tELEMENT = "
            << map1.lower_bound(5)->second << endl;
      cout << "map1.upper_bound(5) : "
            << "\tKEY = ";
      cout << map1.upper_bound(5)->first << '\t';
      cout << "\tELEMENT = "
            << map1.upper_bound(5)->second << endl;

      cout << endl;
      system("pause");
      return 0;
}
```

Map sangat dinamis dalam mengorganisasikan elemen data yang memungkinkan diakses secara cepat dengan pasangan kuncinya. Ini lebih efektif dibandingkan struktur array yang mengharuskan terikat pada penomoran indeks array.

Sementara skip-list adalah single linked list yang memiliki fungsi dinamis untuk proses insert dan remove node tidak saja pada ujung head atau tail tapi juga pada posisi tengan senarai. Suatu catatan bahwa teknik implementasi skip-list yang baik akan meningkatkan performansi struktur data ini melebihi performa B-tree.

Perhatikan contoh insertion skip-list berikut:

```
#include <iostream>
#include <time.h>

using namespace std;

// Class ADT sebagai Node
class Node
```

```
{
public:
	int key;

	// Array pointer menunjuk ke node level berbeda
	Node **forward;
	Node(int, int);
};

Node::Node(int key, int level)
{
	this->key = key;


	forward = new Node*[level+1];

	// Inisialisasi forward array dengan 0(NULL)
	memset(forward, 0, sizeof(Node*)*(level+1));
};

// Class  Skip list
class SkipList
{
	// Maximum level skip list
	int MAXLVL;
	float P;

	// current level skip list
	int level;

	// pointer header node
	Node *header;
public:
	SkipList(int, float);
	int randomLevel();
	Node* createNode(int, int);
	void insertElement(int);
	void displayList();
};

SkipList::SkipList(int MAXLVL, float P)
{
	this->MAXLVL = MAXLVL;
	this->P = P;
	level = 0;
	header = new Node(-1, MAXLVL);
};

// membuat random level node
int SkipList::randomLevel() {
	float r = (float)rand()/RAND_MAX;
```

```
      int lvl = 0;
      while (r < P && lvl < MAXLVL)
      {
            lvl++;
            r = (float)rand()/RAND_MAX;
      }
      return lvl;
};

// buat new node
Node* SkipList::createNode(int key, int level)
{
      Node *n = new Node(key, level);
      return n;
};

// Insert key
void SkipList::insertElement(int key)
{
      const int MAXLVL = 3;
      Node *current = header;

      Node *update[MAXLVL+1];
      memset(update, 0, sizeof(Node*)*(MAXLVL+1));

      for (int i = level; i >= 0; i--)
      {
            while (current->forward[i] != NULL &&
                  current->forward[i]->key < key)
                  current = current->forward[i];
            update[i] = current;
      }

      current = current->forward[0];

      if (current == NULL || current->key != key)
      {

            int rlevel = randomLevel();

            if (rlevel > level)
            {
                  for (int i=level+1;i<rlevel+1;i++)
                        update[i] = header;

                  // Update the list current level
                  level = rlevel;
            }

            Node* n = createNode(key, rlevel);
```

```
            for (int i=0;i<=rlevel;i++)
            {
                  n->forward[i] = update[i]->forward[i];
                  update[i]->forward[i] = n;
            }
            cout << "Successfully Inserted key " << key << "\n";
      }
};

// Display skip list
void SkipList::displayList()
{
      cout<<"\n*****Skip List*****"<<"\n";
      for (int i=0;i<=level;i++)
      {
            Node *node = header->forward[i];
            cout << "Level " << i << ": ";
            while (node != NULL)
            {
                  cout << node->key<<" ";
                  node = node->forward[i];
            }
            cout << "\n";
      }
};

// Menguji skip list
int main() {
      // membuat data acak
      srand((unsigned)time(0));

      // membuat skiplist obejct dengan MAXLVL dan P
      SkipList lst(3, 0.5);

      lst.insertElement(3);
      lst.insertElement(6);
      lst.insertElement(7);
      lst.insertElement(9);
      lst.insertElement(12);
      lst.insertElement(19);
      lst.insertElement(17);
      lst.insertElement(26);
      lst.insertElement(21);
      lst.insertElement(25);
      lst.displayList();

      cout << endl;
      system("pause");
      return 0;
}
```

Skip list diatas menerapkan proses insertion atau penambahan data. Skip list yang interaktif akan dipraktekan pada bagian praktikum.

Tugas Pendahuluan

1. Buatlah sebuah struktur data Map yang memetakan harga setiap barang-barang kebutuhan pokok yang diperjualbelikan di pasar.
2. Buatlah sebuah skip-list sederhana menggunakan linked list.

Langkah-Langkah Praktikum

1. Map ADT object
   a. Kita akan praktikum memetakan objek dari Abstract Data Type yaitu objek class dalam struktur Map
   b. Buat file baru dengan nama praktikum91.cpp
   c. Ketik kode berikut:

```
#include <iostream>
#include <map>
#include <string>

using namespace std;

class User
{
      string m_id;
      string m_name;
public:
      User(string name, string id) :
        m_id(id), m_name(name)
        {
        }
        const string& getId() const
        {
              return m_id;
        }
        const string& getName() const
        {
              return m_name;
        }
        bool operator<(const User& userObj) const
        {
              if (userObj.m_id < this->m_id)
                    return true;
        }
};
void example_1()
{
```

```
	map<User, int> m_UserInfoMap;
	m_UserInfoMap.insert(make_pair<User,
		int>(User("Mr.X", "3"), 100));
	m_UserInfoMap.insert(make_pair<User,
		int>(User("Mr.X", "1"), 120));
	m_UserInfoMap.insert(make_pair<User,
		int>(User("Mr.Z", "2"), 300));
	map<User, int>::iterator it = m_UserInfoMap.begin();
	for (; it != m_UserInfoMap.end(); it++)
	{
		cout << it->first.getName() << " :: "
			<< it->second << endl;
	}
}
struct UserNameComparator{
	bool operator()(const User &left, const User &right)
const
	{
		return (left.getName() > right.getName());
	}
};
void example_2(){
	map<User, int, UserNameComparator> m_UserInfoMap;
	m_UserInfoMap.insert(make_pair<User,
		int>(User("Mr.X", "3"), 100));
	m_UserInfoMap.insert(make_pair<User,
		int>(User("Mr.X", "1"), 120));
	m_UserInfoMap.insert(make_pair<User,
		int>(User("Mr.Z", "2"), 300));
	map<User, int, UserNameComparator>::iterator it =
		m_UserInfoMap.begin();
	for (; it != m_UserInfoMap.end(); it++)
	{
		cout << it->first.getName() << " :: "
			<< it->second << endl;
	}
}
int main()
{
	cout << "EXAMPLE 1 :: Comparing by ID"
		<< endl;
	example_1();
	cout << "EXAMPLE 1 :: Comparing by NAME"
		<< endl;
	example_2();

	cout << endl;
	system("pause");
	return 0;
}
```

d. Jalankan program
e. Cobalah ubah beberapa baris kode dan uji dengan menjalankannya kembali.

2. Program interaktif Skip-List.
   a. Kita akan mempraktekan program interaktif struktur data skip-list yang melibatkan operasi menambah data, dan mencari data dan menghapus data.
   b. Buatlah file baru dengan nama praktikum92.cpp
   c. Ketik kode berikut:

```
/*
 * Program Skip List
 */
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <cstring>
#include <time.h>
#define MAX_LEVEL 6
const float P = 0.5;
using namespace std;
/*
 * Skip Node Declaration
 */
struct snode
{
    int value;
    snode **forw;
    snode(int level, int &value)
    {
        forw = new snode * [level + 1];
        memset(forw, 0, sizeof(snode*) * (level + 1));
        this->value = value;
    }
    ~snode()
    {
        delete [] forw;
    }
};
/*
 * Skip List Declaration
 */
struct skiplist
{
    snode *header;
    int value;
    int level;
    skiplist()
    {
        header = new snode(MAX_LEVEL, value);
```

```
        level = 0;
    }
    ~skiplist()
    {
        delete header;
    }
    void display();
    bool contains(int &);
    void insert_element(int &);
    void delete_element(int &);
};
/*
 * Main: Contains Menu
 */
int main()
{
    skiplist ss;
    int choice, n;
    while (1)
    {
        cout<<endl<<"-----------------------"<<endl;
        cout<<endl<<"Operations on Skip list"<<endl;
        cout<<endl<<"-----------------------"<<endl;
        cout<<"1.Insert Element"<<endl;
        cout<<"2.Delete Element"<<endl;
        cout<<"3.Search Element"<<endl;
        cout<<"4.Display List "<<endl;
        cout<<"5.Exit "<<endl;
        cout<<"Enter your choice : ";
        cin>>choice;
        switch(choice)
        {
        case 1:
             cout<<"Enter the element to be inserted: ";
             cin>>n;
             ss.insert_element(n);
             if(ss.contains(n))
                 cout<<"Element Inserted"<<endl;
             break;
        case 2:
             cout<<"Enter the element to be deleted: ";
             cin>>n;
             if(!ss.contains(n))
             {
                 cout<<"Element not found"<<endl;
                 break;
             }
             ss.delete_element(n);
             if(!ss.contains(n))
                 cout<<"Element Deleted"<<endl;
             break;
```

```
        case 3:
             cout<<"Enter the element to be searched: ";
             cin>>n;
             if(ss.contains(n))
                 cout<<"Element "
                  <<n<<" is in the list"<<endl;
             else
                 cout<<"Element not found"<<endl;
        case 4:
             cout<<"The List is: ";
           ss.display();
             break;
        case 5:
             exit(1);
             break;
        default:
             cout<<"Wrong Choice"<<endl;
        }
    }
      cout << endl;
      system("pause");
    return 0;
}

/*
 * Random Value Generator
 */
float frand()
{
    return (float) rand() / RAND_MAX;
}

/*
 * Random Level Generator
 */
int random_level()
{
    static bool first = true;
    if (first)
    {
        srand((unsigned)time(NULL));
        first = false;
    }
    int lvl = (int)(log(frand()) / log(1.-P));
    return lvl < MAX_LEVEL ? lvl : MAX_LEVEL;
}

/*
 * Insert Element in Skip List
 */
void skiplist::insert_element(int &value)
```

```
{
    snode *x = header;
    snode *update[MAX_LEVEL + 1];
    memset(update, 0, sizeof(snode*) * (MAX_LEVEL + 1));
    for (int i = level;i >= 0;i--)
    {
        while (x->forw[i] != NULL && \
               x->forw[i]->value < value)
        {
            x = x->forw[i];
      }
        update[i] = x;
    }
    x = x->forw[0];
    if (x == NULL || x->value != value)
    {
        int lvl = random_level();
        if (lvl > level)
        {
            for (int i = level + 1;i <= lvl;i++)
            {
                update[i] = header;
            }
            level = lvl;
        }
        x = new snode(lvl, value);
        for (int i = 0;i <= lvl;i++)
        {
            x->forw[i] = update[i]->forw[i];
            update[i]->forw[i] = x;
        }
    }
}

/*
 * Delete Element from Skip List
 */
void skiplist::delete_element(int &value)
{
    snode *x = header;
    snode *update[MAX_LEVEL + 1];
    memset (update, 0, sizeof(snode*) * (MAX_LEVEL + 1));
    for (int i = level;i >= 0;i--)
    {
        while (x->forw[i] != NULL && \
               x->forw[i]->value < value)
        {
            x = x->forw[i];
        }
        update[i] = x;
    }
```

```
    x = x->forw[0];
    if (x->value == value)
    {
        for (int i = 0;i <= level;i++)
        {
            if (update[i]->forw[i] != x)
                break;
            update[i]->forw[i] = x->forw[i];
        }
        delete x;
      while (level > 0 && header->forw[level] == NULL)
        {
            level--;
        }
    }
}

/*
 * Display Elements of Skip List
 */
void skiplist::display()
{
    const snode *x = header->forw[0];
    while (x != NULL)
    {
        cout << x->value;
        x = x->forw[0];
        if (x != NULL)
            cout << " - ";
    }
    cout <<endl;
}

/*
 * Search Elemets in Skip List
 */
bool skiplist::contains(int &s_value)
{
    snode *x = header;
    for (int i = level;i >= 0;i--)
    {
        while (x->forw[i] != NULL && \
               x->forw[i]->value < s_value)
        {
            x = x->forw[i];
        }
    }
    x = x->forw[0];
    return x != NULL && x->value == s_value;
}
```

d. Jalankan program
e. Lakukan beberapa simulasi untuk lebih memahami program.

Tugas Laporan

1. Buatlah sebuah program yang memetakan data-data perpustakaan seperti judul buku, pengarang, genre, penerbit, tahun, dan ringkasan isi buku kedalam struktur data menggunakan map container.
2. Buatlah sebuah struktur data terurut yang memudahkan pencarian identitas mahasiswa yang menampilkan nama, nim, asal daerah, jurusan dan fakultas menggunakan struktur Skip-List.

Modul Praktikum 10: Tries dan Graph

Deskripsi Pertemuan

Praktikum 10 akan membahas tentang Tries dan graph

Syarat Kompetensi

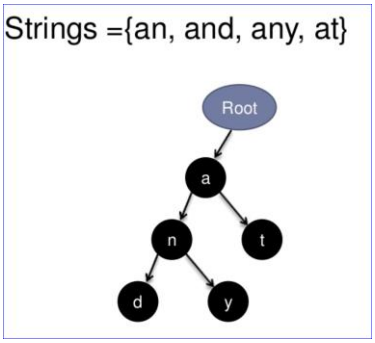
1. Memahami dan mampu mengimplementasikan struktur list dalam program.
2. Memahami dan mampu mengimplementasikan logika seleksi dalam program.
3. Memahami dan mampu mengimplementasikan logika perulangan dalam program.
4. Memahami dan mampu mengimplementasikan stuktur Tree dalam program.

Standar Kompetensi

1. Mahasiswa mampu mengimplementasikan struktur Tries dan Graph dalam program.
2. Mahasiswa mampu menganalisis dan merancang program penyelesaian kasus menggunakan Tries dan Graph.

Dasar Teori

Trie adalah struktur data serupa tree yang disusun dari node-node linked list. Perbedaan trie dengan struktur tree adalah pada trie hirarki node tidak lain adalah hirarki alphabet yang disusun untuk membentuk kata kunci.



Jadi trie digunakan untuk menyimpan informasi kata atau kalimat dalam struktur pohon data.

Pada contoh trie, kata **an, and, any,** dan **at,** disusun sebagai abjad berjenjang yang dapat dirujuk pada hirarki pohon data. Huruf **a** menjadi parent dari semua huruf lain pembentuk kata. Dengan demikian penyebutan huruf **a** dalam pohon data tidak berulang dalam satu level.

Berikut ini adalah contoh sederhanan program **trie-sample**:

```
#include <iostream>
#include <string>
#include <map>
#include <vector>

using namespace std;

struct Node {
      char ch;
      map<char, Node*> children;
};

class Trie {
      public:
            Trie() { head.ch = -1; };
            ~Trie();

    void build_trie(string words[], int length);
    void insert(string word);
    void search(string word, bool &result);

    void print_tree(map<char, Node*> tree);
    void print();

protected:
    Node head;
    vector<Node*> children;
};

Trie::~Trie() {
      for (int i=0; i<children.size(); ++i) {
            delete children[i];
      }
}

void Trie::build_trie(string words[], int length) {
      for (int i=0; i<length; ++i) {
            insert(words[i]);
      }
}
```

```
void Trie::insert(string word) {
      map<char, Node*> *current_tree = &head.children;
      map<char, Node*>::iterator it;

      for (int i=0; i<word.length(); ++i) {
            char ch = word[i];

    if ((it = current_tree->find(ch)) != current_tree->end()) {
            current_tree = &it->second->children;
            continue;
    }

    if (it == current_tree->end()) {

            Node* new_node = new Node();
            new_node->ch = ch;
            (*current_tree)[ch] = new_node;

            current_tree = &new_node->children;

            children.push_back(new_node);
    }
  }
}

void Trie::search(string word, bool &result) {
      map<char, Node*> current_tree = head.children;
      map<char, Node*>::iterator it;

      for (int i=0; i<word.length(); ++i) {
            if ((it = current_tree.find(word[i])) ==\
            current_tree.end()) {
            result = false;
            return;
            }
      current_tree = it->second->children;
      }

      result = true;
      return ;
}

void Trie::print_tree(map<char, Node*> tree) {
      if (tree.empty()) {
            return;
      }

      for (map<char, Node*>::iterator it=tree.begin();\
     it!=tree.end(); ++it) {
            cout << it->first << endl;
            print_tree(it->second->children);
```

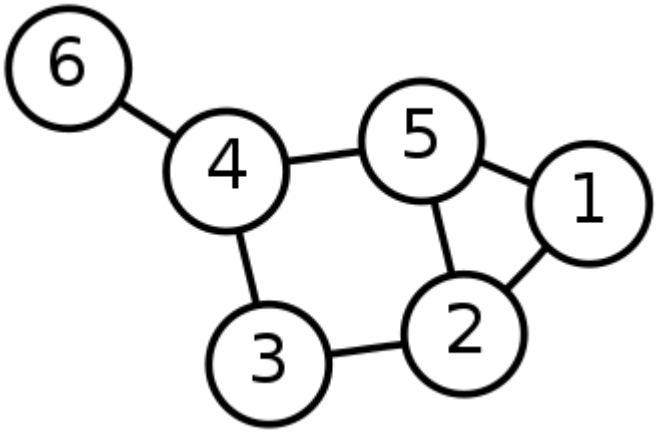
```
        }
}

void Trie::print() {
        map<char, Node*> current_tree = head.children;
        print_tree(current_tree);
}


int main(int argc, char** argv){
        string words[] = {"foo", "bar", "baz", "barz"};
        Trie trie;
        trie.build_trie(words, 4);
        cout << "All nodes..." << endl;
        trie.print();

        cout << "Searching..." << endl;
        bool in_trie = false;
        trie.search("foo", in_trie);
        cout << "foo " << in_trie << endl;

        trie.search("fooz", in_trie);
        cout << "fooz " << in_trie << endl;

        trie.search("bar", in_trie);
        cout << "bar " << in_trie << endl;

        trie.search("baz", in_trie);
        cout << "baz " << in_trie << endl;

        trie.search("barz", in_trie);
        cout << "barz " << in_trie << endl;

        trie.search("bbb", in_trie);
        cout << "bbb " << in_trie << endl;

        cout << endl;
        system("pause");
        return 0;
}
```

Trie pada contoh menggunakan Abstract Data Type yaitu class dan User Defined Type yaitu struct untuk membuat struktur pohon data.

Berbeda dengan trie, graph dalam struktur data adalah koleksi node-node yang saling berhubungan dan tidak dalam bentuk hirarki. Dalam graph, setiap node bisa merepresentasikan data apa saja, misalnya data tentang kota-kota yang saling

berhubungan. Garis-garis penghubungnya diibaratkan jalan-jalan yang menghubungkan kota danseterusnya.

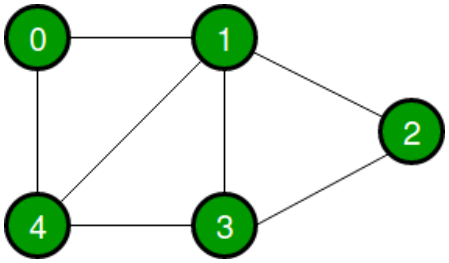
Berikut ini adalah contoh graph:



Graph tersebut terdiri atas 6 node-node yang saling terhubung. Tidak ada pola yang menjadi dasar hubungan setiap node. Pada prakteknya, graph bisa digunakan untuk merepresentasikan jaringan komputer dalam rancangan protokol komunikasi data.

Sesuai dengan sifat relasi node, graph dikelompokan menjadi graph terarah dan graph tidak terarah. Dalm pemrograman, masing-masing struktur graph memiliki implementasi sesuai dengan kasus yang ingin dipecahkan.

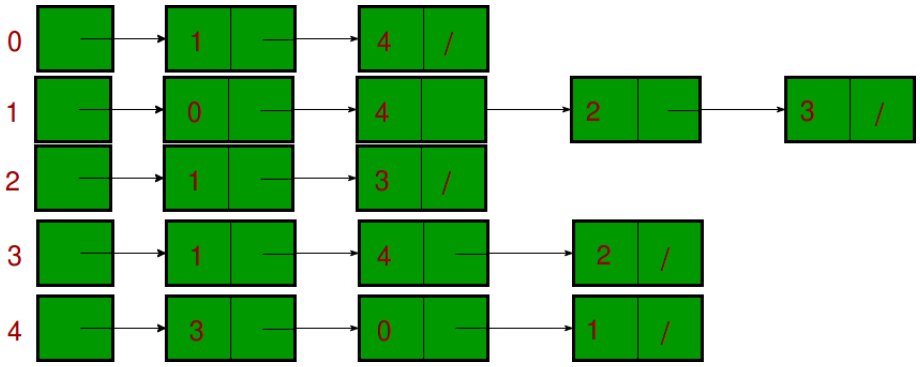
Graph secara teori direpresentasikan melalui 2 cara yaitu Adjacency Matrix dan Adjacency List. Sebagai contoh berikut adalah graph dengan masing-masing representasinya:



Representasi Adjacency Matrix graph tersebut adalah:

	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

Sedangkan representasi Adjacency List graph tersebut adalah:



Berikut ini adalah contoh sederhana program **graph-sample** yang mengimplementasikan representasi Adjacency List:

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;


void addEdge(vector<int> adj[], int u, int v) {
	adj[u].push_back(v);
	adj[v].push_back(u);
}

void printGraph(vector<int> adj[], int V) {
	for (int v = 0; v < V; ++v) {
		cout << "\n Adjacency list of vertex "
			<< v << "\n head ";
		for (auto x = adj[v].begin(); x != adj[v].end(); ++x)
		cout << "-> " << *x;

		printf("\n");
	}
}

int main()
{
	const int V = 5;
	vector<int> adj[V];
	addEdge(adj, 0, 1);
	addEdge(adj, 0, 4);
	addEdge(adj, 1, 2);
	addEdge(adj, 1, 3);
	addEdge(adj, 1, 4);
	addEdge(adj, 2, 3);
	addEdge(adj, 3, 4);
	printGraph(adj, V);
```

```
	cout << endl;
	system("pause");
	return 0;
}
```

Graph diatas tidak merekam bobot koneksi setiap node. Graph ini disebut **unweighted graph**.

Tugas Pendahuluan

1. Pelajari program trie-sample diatas lalu buatlah program yang sama untuk menampung data-data nama mahasiswa dalam struktur Trie.
2. Pelajari program **graph-sample** setelah itu buatlah program yang sama dengan representasi Adjacency Matrix

Langkah-Langkah Praktikum

1. Program kamus kata dengan struktur Trie
   a. Buat file baru dengan nama praktikum101.cpp
   b. Ketik program berikut:

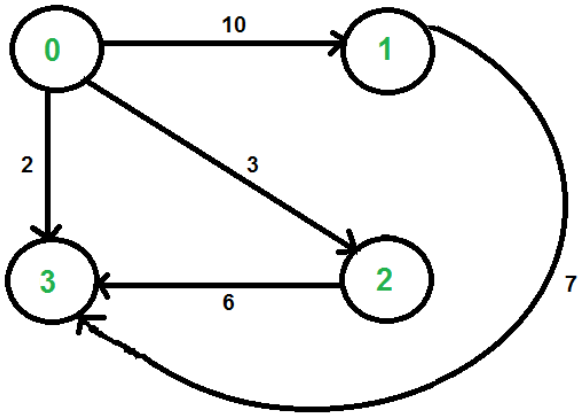
```
#include <iostream>
#include <vector>
#include <string>
#include <unordered_map>

using namespace std;


struct Trie {
	bool isEndOfWord;
	unordered_map<char, Trie*> map;
	string meaning;
};


Trie* getNewTrieNode()
{
	Trie* node = new Trie;
	node->isEndOfWord = false;
	return node;
}


void insert(Trie*& root, const string& str,
		const string& meaning)
{
```

```
	if (root == NULL)
		root = getNewTrieNode();

	Trie* temp = root;
	for (int i = 0; i < str.length(); i++) {
		char x = str[i];

		if (temp->map.find(x) == temp->map.end())
			temp->map[x] = getNewTrieNode();

		temp = temp->map[x];
	}

	temp->isEndOfWord = true;
	temp->meaning = meaning;
}

string getMeaning(Trie* root, const string& word)
{

	if (root == NULL)
		return "";

	Trie* temp = root;

	for (int i = 0; i < word.length(); i++) {
		temp = temp->map[word[i]];
		if (temp == NULL)
			return "";
	}

	if (temp->isEndOfWord)
		return temp->meaning;
	return "";
}

int main()
{
	Trie* root = NULL;

	insert(root, "ayam", "ayam adalah manu");
	insert(root, "bale", \
		"bale adalah ikan yang bisa digoreng");
	insert(root, "sekolah", \
		"sekolah adalah tempat orang pintar");
	insert(root, "baju", \
		"baju adalah pakaian dan pakaian adalah baju");
	insert(root, "sepatu", \
		"sepatu itu bukan tas");

	string str = "bale";
```

```
	cout << getMeaning(root, str);

	cout << endl;
	system("pause");
	return 0;
}
```

c. Jalankan program
d. Simulasikan dengan mengedit beberapa baris kota dan uji jalankan kembali

2. Program graph dengan bobot link
   a. Buat file baru dengan nama praktikum102.cpp
   b. Perhatikan gambar graph



   c. Graph akan diterjemahkan kedalam program
   d. Ketik kode berikut:

```
#include <iostream>
#include <vector>
#include <string>
#include <unordered_map>


using namespace std;

void addEdge(vector <pair<int, int> > adj[], int u,
                     int v, int wt)
{
	adj[u].push_back(make_pair(v, wt));
	adj[v].push_back(make_pair(u, wt));
}

void printGraph(vector<pair<int,int> > adj[], int V)
{
	int v, w;
	for (int u = 0; u < V; u++)
	{
		cout << "Node " << u << " makes an edge with \n";
		for (auto it = adj[u].begin(); it!=adj[u].end();\
```

```
                it++)
            {
                  v = it->first;
                  w = it->second;
                  cout << "\tNode " << v << \
                          " with edge weight ="
                       << w << "\n";
            }
            cout << "\n";
      }
}

int main()
{
      const int V = 5;
      vector<pair<int, int> > adj[V];
      addEdge(adj, 0, 1, 10);
      addEdge(adj, 0, 4, 20);
      addEdge(adj, 1, 2, 30);
      addEdge(adj, 1, 3, 40);
      addEdge(adj, 1, 4, 50);
      addEdge(adj, 2, 3, 60);
      addEdge(adj, 3, 4, 70);
      printGraph(adj, V);

      cout << endl;
      system("pause");
      return 0;
}
```

e. Jalankan program
f. Lakukan perubahan baris kode kemudian uji dengan menjalankannya kembali.

Tugas Laporan

1. Buatlah program kamus kata bahasa bugis dan makassar.
2. Buatlah program graph yang menyimpan jaringan setiap kecamatan di makassar dan jarak dari satu kecamatan ke kecamatan lain. Program dapat menampilkan jarak antara kecamatan berdasarkan input pengguna.

Lampiran-Lampiran

Format Tugas Pendahuluan

TUGAS PENDAHULUAN PRAKTIKUM 1

Nama :

Nim :

Soal :

……..

Penyelesaian:

a. **Penjelasan**

   …….

b. **Program**

   …….

c. **Penjelasan kode program**

   …….

Tanda Tangan Asistensi

Asisten

Format Laporan

TUGAS LAPORAN PRAKTIKUM 1

Nama :

Nim :

Soal :

……..

Penyelesaian:

a. **Penjelasan**

   …….

b. **Program**

   …….

c. **Penjelasan kode program**

   …….

Tanda Tangan Asistensi

Asisten