

Modul 1 : Algoritma

1.1 Waktu Pelaksanaan Praktikum

Durasi kegiatan praktikum = 170 menit, dengan rincian sebagai berikut :

- a. 15 menit untuk pengerjaan Tes Awal atau wawancara Tugas Pendahuluan
- b. 60 menit untuk penyampaian materi
- c. 45 menit untuk pengerjaan tugas / Studi Kasus
- d. 50 menit Pengayaan

1.2 Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat:

1. Mengetahui dan membuat tentang macam-macam algoritma, dan langkah-langkah membuat algoritma.
2. Mengetahui dan membuat tentang simbol-simbol flowchart, dan menyelesaikan masalah pemrograman dengan menggunakan flowchart.
3. Mengetahui dan membuat bagian-bagian pseudocode menyelesaikan masalah pemrograman dengan menggunakan pseudocode.

1.3 Alat & Bahan

1. Komputer
2. Microsoft Word atau Microsoft Visio

1.4 Dasar Teori

Logika pemrograman dapat diartikan sebagai suatu metode atau cara berpikir terstruktur yang diperlukan untuk menyelesaikan permasalahan di dalam menyusun suatu program. Ketika seseorang telah memahami langkah-langkah di dalam menyelesaikan masalah di dalam membuat suatu program, maka ia akan lebih mudah mengimplementasikannya ke dalam suatu bahasa pemrograman.

Setiap permasalahan pemrograman dapat diselesaikan dengan metode atau langkah yang berbeda-beda, tetapi penggunaan metode yang baik (optimal) ditentukan oleh tujuan dari program tersebut dibuat. Kemampuan seseorang di dalam menentukan metode atau cara yang tepat di dalam menyelesaikan masalah pemrograman tergantung pada kemampuan analisis dan pengalaman yang dimilikinya. Oleh karena itu latihan penyelesaian permasalahan pemrograman sangat diperlukan dan akan membantu melatih dan membentuk cara berpikir yang baik di dalam menyelesaikan permasalahan pemrograman.

Metode atau cara menyelesaikan permasalahan pemrograman dapat dilakukan dengan menggunakan algoritma dan *flowchart* ataupun *pseudocode*.

1.4.1 Algoritma

Algoritma dapat diartikan sebagai suatu cara di dalam menyelesaikan masalah melalui serangkaian langkah-langkah atau tahap terstruktur dan ditulis secara

sistematis. Pada umumnya algoritma banyak diimplementasikan dalam bentuk tulisan, tetapi ada pula yang diimplementasikan dalam bentuk gambar.

Untuk menyusun algoritma di dalam menyelesaikan permasalahan, maka kita harus memahami permasalahannya dan mengetahui langkah-langkah penyelesaian masalah tersebut untuk dituangkan ke dalam bentuk tulisan/gambar.

1.4.2 Algoritma Pemrograman

Algoritma yang digunakan untuk membuat suatu program disebut sebagai algoritma pemrograman. Sebagaimana algoritma secara umum, algoritma pemrograman juga berisi langkah-langkah terstruktur untuk menyelesaikan permasalahan di bidang pemrograman (untuk membuat suatu program).

Algoritma pemrograman yang baik adalah algoritma yang jelas dan mudah dipahami oleh yang membaca algoritma tersebut, sehingga memudahkan *programmer* ketika akan mengimplemen-tasikannya ke dalam bahasa pemrograman.

Secara umum, algoritma pemrograman dimulai dengan menentukan nilai suatu variabel, yang diinputkan oleh pengguna, ataupun yang berupa nilai awal. Kemudian dilanjutkan dengan memproses variabel tersebut sesuai dengan tujuan dibuatnya program, dan hasil proses tersebut ditampilkan kepada pengguna ataupun digunakan sebagai nilai awal untuk proses berikutnya, sampai akhirnya tujuan program tercapai dan program selesai.





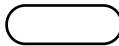


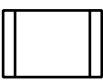
1.4.3 Diagram Alir (*Flowchart*)

Selain menggunakan algoritma, representasi logika pemrograman di dalam menyelesaikan permasalahan di dalam pemrograman adalah diagram alir (*flowchart*). *Flowchart* dapat diartikan sebagai suatu aliran proses yang berupa simbol/bagan yang menggambarkan urutan-urutan penyelesaian permasalahan, dimana terjadi hubungan antara proses yang satu dengan yang lainnya. *Flowchart* dituangkan ke dalam bentuk gambar atau simbol yang telah menjadi kesepakatan para ahli komputer di dalam representasi logika pemrograman untuk menyusun program komputer.

1.4.4 Program Flowchart

Adalah suatu *flowchart* yang digunakan untuk menyelesaikan permasalahan pada pembuatan program. Setiap penyelesaian permasalahan pemrograman dapat digambarkan dengan menggunakan simbol-simbol yang terdapat pada *program flowchart*. *Flowchart* inilah yang akan kita gunakan ketika kita akan menyelesaikan permasalahan pemrograman yang pada akhirnya akan diimplementasikan ke dalam suatu bahasa pemrograman. Simbol-simbol yang digunakan pada *program flowchart* dapat dilihat pada gambar di bawah.

Simbol-simbol yang digunakan pada *program flowchart* antara lain :

 <p>Proses</p>	<ul style="list-style-type: none"> • Menunjukkan suatu proses/ pengolahan • Digunakan untuk melambangkan : <ul style="list-style-type: none"> - perhitungan - perubahan nilai variabel
 <p>Operasi Input/Output</p>	<ul style="list-style-type: none"> • Menunjukkan operasi input/ouput • Digunakan untuk melambangkan : <ul style="list-style-type: none"> - menunggu input/ masukan - mengeluarkan output/ keluaran
 <p>Persiapan (<i>Preparation</i>)</p>	<ul style="list-style-type: none"> • Menunjukkan suatu persiapan • Digunakan untuk melambangkan : <ul style="list-style-type: none"> - memberikan nilai awal pada suatu variabel - permulaan dari suatu perulangan
 <p>Keputusan (<i>Decision</i>)</p>	<ul style="list-style-type: none"> • Menunjukkan proses pembuatan keputusan • Digunakan untuk melambangkan : <ul style="list-style-type: none"> - suatu pilihan/ percabangan (ya/tidak)
 <p>Terminal (<i>Terminator</i>)</p>	<ul style="list-style-type: none"> • Digunakan untuk menunjuk kan awal dan akhir suatu program/<i>flowchart</i>
 <p>Penghubung (<i>connector</i>)</p>	<ul style="list-style-type: none"> • Digunakan sebagai penghubung antar simbol <i>flowchart</i> yang terpisah (simbol yang terpisah masih berada dalam satu halaman)
 <p>Penghubung antar halaman (<i>Offpage Connector</i>)</p>	<ul style="list-style-type: none"> • Digunakan sebagai penghubung antar simbol <i>flowchart</i> yang terpisah (simbol yang terpisah berada pada halaman yang berbeda)
 <p>Modul</p>	<ul style="list-style-type: none"> • Menunjukkan suatu proses / Subproses yang telah ditentukan • Dapat berupa suatu : <ul style="list-style-type: none"> - Prosedur - Fungsi

Panah ↓	<ul style="list-style-type: none"> • Menunjukkan arah dari suatu proses •
------------	---

Gambar Simbol-simbol pada *program flowchart*

Di dalam menyusun atau membuat suatu *program flowchart* ada kaidah atau aturan-aturan yang harus dipenuhi. Aturan-aturan tersebut antara lain :

1. Digunakan simbol yang sesuai dengan fungsi simbol tersebut dan berurutan berdasarkan langkah yang digunakan.
2. Jalannya metode penyelesaian dibuat sesingkat-singkatnya, tetapi tetap dapat dengan mudah dipahami.
3. Dibuat metode penyelesaian yang sederhana, mudah dipahami, dan diimplementasikan. Hilangkan langkah-langkah (logika) yang tidak perlu dan berbelit-belit.
4. Digunakan simbol penghubung, jika metode penyelesaian tidak dapat diselesaikan dengan langkah yang sederhana, atau membutuhkan penyelesaian lebih dari 1 halaman
5. Digunakan simbol pengulangan untuk menunjukkan langkah yang diulang-ulang.
6. Digunakan modul, jika terdapat langkah atau metode yang sering dilakukan
7. Penyusunan simbol digambarkan dari atas ke bawah, dan dari kiri ke kanan, dan diberi arah panah yang menunjukkan langkah yang dilakukan sebelumnya dan setelahnya. Setiap simbol (kecuali *start* dan *stop* minimal memiliki sebuah panah masuk dan sebuah panah keluar.

Sebuah *flowchart* diawali dengan simbol “Mulai” (*Start*) dan diakhiri dengan “Selesai” (*Stop*)

1.4.5 Pseudocode

Representasi lain dari logika pemrograman di dalam menyelesaikan permasalahan pemrograman yang sering digunakan adalah *pseudocode*. *Pseudocode* adalah suatu urutan langkah-langkah atau prosedur penyelesaian masalah yang dituliskan dalam bentuk yang sistematis yang mendekati pernyataan/instruksi atau *syntax* yang digunakan oleh suatu bahasa pemrograman. Representasi *pseudocode* ini banyak digunakan di dalam penulisan ilmiah yang dipublikasikan (misalnya pada makalah, atau jurnal) karena penggunaan ruang tulisan yang lebih sedikit dibandingkan *flowchart*, dan juga kedekatannya dengan struktur bahasa pemrograman.

Struktur penulisan dan perintah yang digunakan tidak memiliki notasi/symbol yang baku, tetapi berdasarkan kesepakatan yang dianggap dapat mempermudah

seseorang di dalam memahami langkah-langkah penyelesaian masalah di dalam pemrograman. Misalnya, untuk pernyataan-pernyataan yang saling berhubungan atau saling tergantung, biasanya ditulis dengan indentasi, seperti keputusan, dan perulangan.

Notasi pada *pseudocode* yang digunakan biasanya berkorespondensi dengan bahasa pemrograman yang umum, dan biasa disebut **notasi algoritmik**.

1.4.6 Struktur *Pseudocode*

Walaupun tidak ada notasi baku di dalam menyusun atau membuat suatu *pseudocode*, tetapi umumnya suatu *pseudocode* memiliki suatu struktur sebagai berikut :

- Kepala *pseudocode*

Kepala *pseudocode* terdiri atas nama *pseudocode* dan komentar yang berisi penjelasan (spesifikasi) tentang *pseudocode* tersebut. Pada kepala *pseudocode* ini biasanya juga dicantumkan nama-nama pembuat program dan versi atau revisi dari program tersebut. Jika merupakan pengembangan dari *pseudocode* sebelumnya, maka biasanya juga berisi fitur-fitur baru yang telah ditambahkan pada *pseudocode* tersebut.

- Deklarasi

Bagian deklarasi biasanya berisi tentang pencantuman atau penentuan penggunaan semua nama (konstanta, peubah/*variable*, tipe atau jenis data, prosedur atau fungsi) yang digunakan di dalam *pseudocode*. Biasanya digunakan beberapa kesepakatan untuk penentuan nama untuk konstanta, dan peubah yang digunakan. Misalnya untuk nama konstanta digunakan huruf kapital, seperti PHI=3.14, AWAL=0, dll, dan huruf kecil untuk peubah, misalnya bil, nilai, dll.

Pada bagian ini juga biasanya dituliskan tentang tipe-tipe data yang digunakan dan juga tipe data baru yang dibuat, yang akan dipergunakan di dalam deskripsi atau badan *pseudocode* tersebut.

- Deskripsi atau badan

Bagian deskripsi adalah bagian yang berisi uraian langkah-langkah penyelesaian permasalahan. Deskripsi juga berisi tentang proses, pembuatan keputusan, dan perulangan. Umumnya untuk tiap-tiap bagian atau kelompok penyelesaian akan diberi komentar, yang umumnya ditulis menggunakan tanda kurung “{” dan “}”, atau “//”.

Bagian ini adalah bagian yang paling penting dari suatu *pseudocode*, yang merupakan bagian utama yang berisi langkah-langkah yang dilakukan di dalam penyelesaian masalah pemrograman.

Pseudocode di atas bertujuan untuk menampilkan bilangan yang diinputkan oleh pengguna ke layar. Pada *pseudocode* tersebut dideklarasikan (digunakan) peubah bernama "Bil" yang bertipe *integer* yang akan menyimpan nilai yang diinputkan oleh pengguna. Tipe *integer* yang digunakan, menunjukkan bahwa input yang diberikan oleh pengguna harus berupa bilangan yang memiliki nilai jangkauan (*range*) tertentu. Pada bagian deskripsi dijelaskan tentang proses yang dilakukan, yaitu proses untuk meminta masukan dari pengguna, dan proses menampilkan atau mencetak bilangan yang sudah diinputkan oleh pengguna ke layar.

1.5 Prosedur Praktikum

Pelajari contoh-contoh Algoritma Pemrograman, flowchart dan Pseudocode berikut :

a. Algoritma Pemrograman

Perhatikan dan analisis algoritma pemrograman berikut :

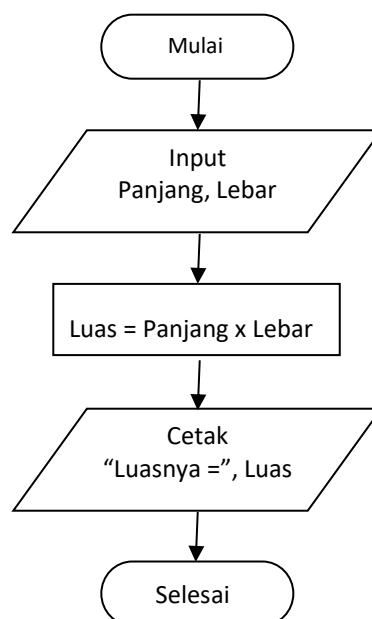
1. Tentukan isi dari variabel pertama.
2. Tentukan isi dari variabel kedua.
3. Buat variabel ketiga untuk menampung isi variabel secara sementara.
4. Masukkan isi variabel pertama ke variabel ketiga.
5. Masukkan isi variabel kedua ke variabel pertama.
6. Masukkan isi variabel ketiga ke variabel kedua.
7. Tampilkan kembali isi variabel pertama dan kedua.

Apa judul yang tepat untuk algoritma di atas ?

.....

b. Flowchart

Perhatikan dan analisis flowchart berikut :



Apa judul yang tepat untuk flowchart di atas ?

.....

c. Pseudocode

Pseudocode Menghitung.....; {pseudocode ini akan menghitung luas dari sebuah segiempat yang merupakan perkalian dari panjang dan lebarnya}
Deklarasi Panjang, Lebar, Keliling : word;
Deksripsi Cetak "Inputkan nilai panjang : "; Input Panjang; Cetak "Inputkan nilai lebar : ", Input Lebar; Keliling = 2*Panjang * Lebar; Cetak "Kelilingnya = ", Keliling;

Apa judul yang tepat untuk pseudocode di atas ?

.....

1.6 Analisis Hasil

Jelaskan proses yang terjadi pada Algoritma Pemrograman, Flowchart dan Pseudocode di atas :

a. Algoritma Pemrograman

.....

.....

.....

.....

.....

.....

b. Flowchart

.....

.....

.....

.....

.....

.....

c. Pseudocode

.....

.....

.....

.....

.....

.....

1.7 Kesimpulan

Berikan kesimpulan yang kamu dapatkan dari praktikum ini :

.....

.....

.....

1.8 Latihan

Buatlah Algoritma Pemrograman, Flowchart dan Pseudocode untuk proses di bawah ini :

- Menghitung luas bidang datar
- Mencari hasil perhitungan aritmetika 2 buah bilangan

1.9 Tugas

Kerjakan soal-soal berikut :

- Buat algoritma pemrograman untuk mencari 5 buah bilangan kuadrat yang pertama.
- Buat algoritma pemrograman untuk menghitung jumlah 10 bilangan ganjil yang pertama.
- Buat *pseudocode* untuk menukar nilai dua buah peubah/variabel. Misal variabel A = 5 dan variabel B = 10, diproses sehingga menjadi variabel A = 10 dan variabel B = 5.
- Buat *flowchart* untuk menjumlahkan 5 buah bilangan pertama.
- Buat *pseudocode* untuk menjumlahkan 5 buah bilangan **kuadrat** yang pertama.
- Buat *flowchart* untuk menghitung pangkat 3 dari sebuah bilangan.

1.10 Daftar Pustaka

1. Goodrich, M.T., Tamassia, R., dan Mount, D., 2003, "Data Structures and Algorithms in C++ 2nd edition", John Wiley & Sons.
2. Sahni, S., 2005, "Data Structures, Algorithms and Applications in C++ 2nd edition", Universities Press.
3. Sanjaya, D., 2005, "Asyiknya Belajar Struktur Data di Planet C++", PT Elex Media Komputindo.

Modul 2 : *Struct dan Pointer*

2.1 Waktu Pelaksanaan Praktikum

Durasi kegiatan praktikum = **170 menit**, dengan rincian sebagai berikut:

1. 15 menit untuk pengerjaan Tes Awal atau wawancara Tugas Pendahuluan
2. 60 menit untuk penyampaian materi
3. 45 menit untuk pengerjaan tugas / Studi Kasus
4. 50 menit **Pengayaan**

2.2 Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat:

1. Mengimplementasikan Array, Struct dan Pointer
2. Mengimplementasikan Malloc, Calloc, Realloc dan Free

2.3 Alat & Bahan

1. Komputer
2. Dev C++
3. GCC

2.4 Dasar Teori

1. Array dan Struct

a. Array

Array adalah jenis data yang mana suatu variable dapat menyimpan lebih dari satu item, dengan catatan tipe datanya juga sama, dalam hal ini adalah tipe data integer, float, double, char, atau string.

Suatu array berdimensi satu dideklarasikan dengan cara:

```
tipe_data nama_var[ukuran];
```

tipe_data : untuk menyatakan tipe dari elemen array, misalnya int, char, float.

nama_var : nama variabel array

ukuran : untuk menyatakan jumlah maksimal elemen array.

```
int a[10];
```

Contoh array di atas adalah variabel “a” yang bertipe “integer” dan batas item yang dimiliki array kurang dari atau sama dengan 10 (tidak boleh lebih dari 10). Array bisa digunakan dalam looping for, berbeda dengan struct yang tidak dapat bersarang dalam for.

```
nilai_tes[0] = 70;          /* contoh 1 */
cin >> nilai_tes[2];       /* contoh 2 */
```

Contoh pertama merupakan pemberian nilai 70 ke nilai_tes[0]. Sedangkan contoh 2 merupakan perintah untuk membaca data bilangan dari keyboard dan diberikan ke nilai_tes[2].

b. Struct

Struct adalah koleksi dari variabel yang dinyatakan dengan sebuah nama, dengan sifat setiap variabel dapat memiliki tipe yang berlainan. Struct dapat digunakan untuk mengelompokkan beberapa informasi yang berkaitan menjadi sebuah satu kesatuan. Bentuk umum deklarasi struct adalah sebagai berikut:

```

struct nama_tipe_struct
{
    tipe field1;
    .
    .
    tipe fieldN;
} variabel_struct1, ..., variabel_structM;

```

Elemen dari struktur dapat diakses dengan menggunakan bentuk:

```

variabel_struct.nama_field

```

c. Struct of Array

Struct of array atau disebut juga struct dari array, artinya kita mendeklarasikan sebuah struct yang elemen-elemennya berupa array. Field nama dan nim merupakan array of char, sedangkan field nilai merupakan suatu array. Contoh di bawah ini yang menandakan struct of array adalah adanya array pada field struct mahasiswa, yaitu field nilai.

```

struct Mahasiswa{
    char nama[30];
    char nim[15];
    float nilai[3];
};
Mahasiswa mhs;

```

Cara mengakses Struct of Array

```

mhs.nilai[1]=90;

```

d. Array of Struct

Array of struct atau array dari struct berarti kita mendeklarasikan sebuah array yang elemennya berupa struct.

```

struct Mahasiswa{
    char nama[30];
    char nim[15];
    float nilai[3];
};
Mahasiswa mhs[20];

```

Cara mengakses Array of Struct

```

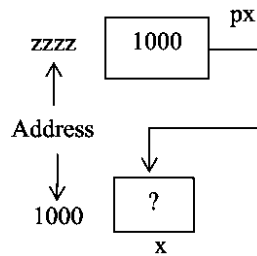
strcpy(mhs[2].nim, '123456789012345');

```

Angka yang ada di dalam tanda [] adalah alamat array dimana isinya berupa variabel yang digunakan untuk perulangan. Fungsi strcpy digunakan untuk menyalin isi string sumber ke array tujuan. Fungsi tersebut dapat digunakan apabila menggunakan cstring di dalam include.

2. Pointer

Variabel pointer sering dikatakan sebagai variabel yang menunjuk ke obyek lain. Pada kenyataan yang sebenarnya, variabel pointer berisi alamat dari suatu obyek lain (yaitu obyek yang dikatakan ditunjuk oleh pointer). Sebagai contoh, "px" adalah variabel pointer dan "x" adalah variabel yang ditunjuk oleh "px". Kalau "x" berada pada alamat memori (alamat awal) 1000, maka "px" akan berisi 1000. Sebagaimana diilustrasikan pada Gambar di bawah ini.



a. Mendeklarasikan Variabel Pointer

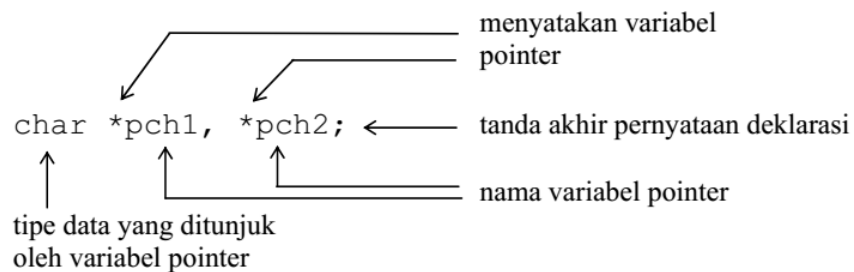
Suatu variabel pointer dideklarasikan dengan bentuk sebagai berikut :

```
tipe *nama_variabel
```

dengan *tipe* dapat berupa sembarang tipe data dalam bahasa C++. Adapun *nama_variabel* adalah nama dari variabel pointer. Sebagai contoh :

```
int *px;                / *contoh 1 */
char *pch1, *pch2;      / *contoh 2 */
```

Contoh pertama menyatakan bahwa “*px*” adalah variabel pointer yang menunjuk ke suatu data bertipe int, sedangkan contoh kedua “*pch1*” dan “*pch2*” merupakan variabel pointer yang menunjuk ke data bertipe char.



b. Pengaturan Pointer agar Menunjuk ke variabel lain

Agar suatu pointer menunjuk ke variabel lain, mula-mula pointer harus diisi dengan alamat dari variabel yang akan ditunjuk. Untuk menyatakan alamat dari suatu variabel, operator “&” (operator alamat, bersifat unary) bisa dipergunakan, dengan menempatkannya di depan nama variabel. Sebagai contoh, bila “*x*” dideklarasikan sebagai variabel bertipe int, maka:

```
&x
```

berarti “alamat dari variabel *x*”. Adapun contoh pemberian alamat *x* ke suatu variabel pointer *px* (yang dideklarasikan sebagai pointer yang menunjuk ke data bertipe *int*) yaitu :

```
px = &x;
```

Pernyataan di atas berarti bahwa *px* diberi nilai berupa alamat dari variabel “*x*”. Setelah pernyataan tersebut dieksekusi barulah dapat dikatakan bahwa *px* menunjuk ke variabel “*x*”.

c. Mengakses Variabel Melalui Pointer

Jika suatu variabel sudah ditunjuk oleh pointer, variabel yang ditunjuk oleh pointer tersebut dapat diakses melalui variabel itu sendiri (pengaksesan langsung) ataupun melalui pointer (pengaksesan tak langsung). Pengaksesan tak langsung dilakukan dengan menggunakan

operator *indirection* (tak langsung) berupa simbol “*” (bersifat unary). Contoh penerapan operator * yaitu :

```
*px
```

yang menyatakan “isi atau nilai variabel/data yang ditunjuk oleh pointer *px*” . Sebagai contoh jika *y* bertipe int, maka sesudah dua pernyataan berikut

```
px = &x;  
y = *px;
```

“*y*” akan berisi nilai yang sama dengan nilai “*x*”.

d. Pointer dan Array

Hubungan antara pointer dan array pada C++ sangatlah erat. Sebab sesungguhnya array secara internal akan diterjemahkan dalam bentuk pointer. Pembahasan berikut akan memberikan gambaran hubungan antara pointer dan array. Misalnya dideklarasikan di dalam suatu fungsi

```
static int tgl_lahir[3] = { 01, 09, 64 };
```

dan

```
int *ptgl;
```

Kemudian diberikan instruksi

```
ptgl = &tgl_lahir[0];
```

maka *ptgl* akan berisi alamat dari elemen array *tgl_lahir* yang berindeks nol. Instruksi di atas bisa juga ditulis menjadi

```
ptgl = tgl_lahir;
```

sebab nama array tanpa tanda kurung menyatakan alamat awal dari array. Sesudah penugasan seperti di atas,

```
*ptgl;
```

dengan sendirinya menyatakan elemen pertama (berindeks sama dengan nol) dari array “*tgl_lahir*”.

3. Alokasi Memori

a. Malloc

Malloc merupakan fungsi standart untuk mengalokasikan memori. Struktur dari fungsi malloc adalah:

```
void* malloc (size_t jml_byte);
```

Banyaknya byte yang akan dipesan dinyatakan sebagai parameter fungsi. Untuk *return value* dari fungsi ini adalah sebuah pointer yang tak bertipe (*pointer to void*) yang menunjuk ke *buffer* (tempat penyimpanan sementara) yang dialokasikan. Pointer harus dikonversi pada tipe data yang sesuai (dengan menggunakan type cast) agar bisa mengakses data yang disimpan dalam buffer. Kegunaan dari malloc adalah mengembalikan pointer sejumlah “*n byte*” ruang memori yang belum diinisialisasi. Apabila tidak terpenuhi akan mengembalikan ke nilai NULL.

Contoh dari malloc() :

```
int *x;
x = (int*) malloc (3 * sizeof(int));
if(x==NULL) {
    cout << "Error di malloc" << endl;
    exit(0);
} else {
    cout << "Lakukan operasi memori dinamis" << endl;
}
```

Sintak tersebut artinya kita mengalokasikan ruang memori sebanyak 12 byte (berasal dari 3x4) untuk menyimpan tipe data int dan mencatat ruang tersebut kedalam pointer x (*x). Nilai 4 berasal dari ukuran tipe data int yang dituliskan dengan sizeof(int). Ukuran dari tipe data int adalah 4 byte. Jadi apabila akan mengalokasikan ruang memori untuk tipe data double berukuran 8 byte, kita dapat menuliskan :

```
double *x;
x = (double*) malloc (8) ;
```

b. Calloc

Struktur dari fungsi calloc() :

```
void* calloc (size_t n, size_t size);
```

Fungsi calloc() akan mengembalikan pointer ke sebuah array yang terdiri dari n elemen data dengan size (ukuran) yang telah ditentukan sebelumnya. Apabila tidak terpenuhi akan mengembalikan ke nilai NULL. Berbeda dengan fungsi malloc() yang tidak melakukan inisialisasi, pada fungsi calloc() ini ruang yang dialokasikan akan diinisialisasi dengan 0. Contoh sintak:

```
int *x;
x = (int*) calloc (50, sizeof(int));
```

Jadi sintak tersebut mengalokasikan 200 byte ruang memori (yang berasal dari 50 x 4). Jadi terdapat 50 elemen array yang masing-masing elemen memiliki ukuran 4 byte.

c. Realloc

Kegunaan dari fungsi dari realloc adalah mengalokasikan ulang memori yang telah digunakan dalam fungsi malloc () dan calloc ().

Struktur dari Realloc

```
void* realloc(void* p , size_t size);
```

Fungsi realloc hanya digunakan jika alokasi memori yang digunakan pada fungsi malloc() dan fungsi calloc() kurang besar. Fungsi dari realloc() adalah mengembalikan pointer ke ruang baru yang ditambahkan, atau mengembalikan nilai NULL apabila permintaan ruang tersebut tidak terpenuhi.

Contoh :

```
int *x;
x = (int *) calloc (10, sizeof (int)) ;
/* memesan ruang baru 40 byte*/
realloc((int*) x, 80 );
/* memesan ruang sebanyak 40 byte lagi */
```

d. Free

Fungsi dari free adalah untuk membebaskan memori yang telah dipakai dalam fungsi malloc(), calloc(), atau realloc().

Struktur dari free

```
void free (void* p)
```

fungsi ini untuk menghindari adanya pemborosan memori atau terjadi kebocoran memori (memori leak) maka harus melakukan dealokasi terhadap ruang-ruang memori yang sebelumnya dialokasikan.

contoh :

```
int *x ;  
x=(int *) malloc (sizeof(int));  
free(x);
```

Jika kita menggunakan bahasa C++ terbaru (versi 11) fungsi standar yang dipakai untuk mengalokasikan memori adalah new

contoh:

```
int* nilai = new int;
```

hasil pengalokasikan diatas ialah suatu alamat yang menunjukkan byte pertama dari memori yang dialokasikan diheap.jika alokasi gagal bisa dikarenakan tidak mencukupinya memori pada heap sehingga akan mengembalikan nilai NULL. Jika kita ingin mendealokasikan memori fungsi standar yang dipakai dalam bahasa C++ yaitu delete sama dengan free sebelumnya.

contoh :

```
int* p1 = new int;  
delete p1;  
  
int* p2 = new int[10];  
delete[] p2;
```

2.5 Prosedur Praktikum

Jalankan masing-masing percobaan di bawah ini

1. Array dan Struct

a. Array

```
1  #include <cstdlib>  
2  #include <iostream>  
3  
4  #define MAX 20  
5  
6  using namespace std;  
7  
8  int fibo[MAX];  
9  
10 int main()  
11 {  
12     int i;  
13     fibo[1] = 1;  
14     fibo[2] = 1;  
15     for (i=3;i<=MAX;i++)  
16         fibo[i]=fibo[i-2]+fibo[i-1];  
17 }
```

```

18     cout<<MAX<<" Bilangan Fibonacci Pertama adalah : "<<endl;
19
20     for (i=1;i<MAX;i++)
21         cout<<fibo[i]<<endl;
22
23     cin.get();
24     return 0;
25 }

```

b. Penggunaan Struktur pada Konversi Koordinat Polar ke Koordinat Cartesian

```

1  #include <cstdlib>
2  #include <iostream>
3  #include <cmath>
4
5  using namespace std;
6
7  struct polar {
8      double r;
9      double alpha;
10 };
11
12 struct kartesian {
13     double x;
14     double y;
15 };
16
17 int main()
18 {
19     struct polar p1;
20     struct kartesian k1;
21
22     cout << "Masukkan nilai r untuk koordinat polar : ";
23     cin >> p1.r;
24
25     cout << "Masukkan nilai alpha untuk koordinat polar : ";
26     cin >> p1.alpha;
27
28     k1.x = p1.r * cos(p1.alpha);
29     k1.y = p1.r * sin(p1.alpha);
30
31     cout << "Nilai koordinat kartesian untuk koordinat polar r =
        " << p1.r << " alpha = " << p1.alpha << " adalah:" <<
        endl;
32     cout << "x = " << k1.x << " y = " << k1.y << endl;
33
34     cin.get();
35     return 0;
36 }

```

c. Struct of Array

```

1  struct Mahasiswa{
2      char nama[30];
3      char nim[15];
4      float nilai[3];
5  };
6
7  int main()
8  {
9      Mahasiswa mhs;
10     int sum=0, jmlNilai=sizeof(mhs.nilai)/sizeof(*mhs.nilai);

```



```

11
12     mhs.nilai[0] = 95;
13     mhs.nilai[1] = 90;
14     mhs.nilai[2] = 85;
15
16     for (int i = 0; i < jmlNilai ; i++)
17         sum += mhs.nilai[i];
18
19     cout << "Rata-rata yang diperoleh adalah : ";
20     cout << sum/jmlNilai;
21
22     cin.get();
23     return 0;
24 }

```

d. Array of Struct

```

1  #include <cstring>
2
3  using namespace std;
4
5  struct Mahasiswa{
6      char nama[30];
7      char nim[15];
8      float nilai[3];
9  };
10
11 int main()
12 {
13     Mahasiswa mhs[3];
14     int jmlMhs = sizeof(mhs)/sizeof(*mhs);
15
16     strcpy(mhs[0].nama, "Sulistyo");
17     strcpy(mhs[0].nim, "123456789054321");
18
19     strcpy(mhs[1].nama, "Kornelia");
20     strcpy(mhs[1].nim, "987654321012345");
21
22     strcpy(mhs[2].nama, "Firmansyah");
23     strcpy(mhs[2].nim, "543210123456789");
24
25     cout << "Data mahasiswa : " << endl;
26     for (int i = 0; i < jmlMhs ; i++) {
27         cout << "Mahasiswa ke-" << i+1 << ":" << endl;
28         cout << "\tNama : " << mhs[i].nama << endl;
29         cout << "\tNIM : " << mhs[i].nim << endl << endl;
30     }
31
32     cin.get();
33     return 0;
34 }

```

2. Pointer

a. Mengubah Isi Variabel Melalui Pointer

```

1  int main()
2  {
3      int y, x = 87;    /* x & y bertipe int */
4      int *px;         /* var pointer yang menunjuk ke data yang bertipe
5                          int */
6      px = &x;         /* px diisi dengan alamat dari variabel x */

```

```

6   y = *px; /* y diisi dengan nilai yg ditunjuk oleh px */
7   cout << "Alamat x = " << &x << endl;
8   cout << "Isi px = " << px << endl;
9   cout << "Isi x = " << x << endl;
10  cout << "Nilai yang ditunjuk oleh px = " << *px << endl;
11  cout << "Nilai y = " << y << endl;
12
13  cin.get();
14  return 0;
15 }

```

b. Mengakses dan Mengubah Isi Suatu Variabel Pointer

```

1  int main()
2  {
3      float d = 54.5f, *pd;
4      cout << "Isi d mula-mula = " << d << endl;
5      pd = &d;
6      *pd += 10;
7      cout << "Isi d sekarang = " << d << endl;
8
9      cin.get();
10     return 0;
11 }

```

c. Penggunaan Pointer untuk Bilangan Fibonacci

```

1  #include <cstdlib>
2  #include <iostream>
3
4  #define MAX 20
5
6  using namespace std;
7
8  int *fibo;
9  int main()
10 {
11     int i;
12     fibo = (int*) malloc(MAX * sizeof(int));
13     *(fibo + 1) = 1;
14     *(fibo + 2) = 1;
15
16     for (i=3;i<=MAX;i++)
17         *(fibo + i) = (*(fibo + i - 2) + *(fibo + i - 1));
18     cout<<MAX<<" Bilangan Fibonacci Pertama adalah : "<<endl;
19
20     for (i=1;i<MAX;i++)
21         cout << *(fibo+i) << endl;
22
23     cin.get();
24     return 0;
25 }

```

3. Alokasi Memori

a. Malloc

```

1  int main()
2  {
3      int num, i, *ptr, sum = 0;
4      cout << "Enter number of elements: ";
5      cin >> num;
6      ptr = (int*) malloc(num * sizeof(int));

```

```

7   if(ptr == NULL)
8   {
9       cout << "Error! memory not allocated.";
10      exit(0);
11  }
12
13  cout << "Enter elements of array: ";
14  for(i = 0; i < num; ++i)
15  {
16      cin >> *(ptr + i);
17      sum += *(ptr + i);
18  }
19  cout << "Sum = " << sum;
20  free(ptr);
21
22  cin.get();
23  return 0;
24  }

```

b. Calloc

```

1   int main()
2   {
3       int num, i, *ptr, sum = 0;
4       cout << "Enter number of elements: ";
5       cin >> num;
6       ptr = (int*) calloc(num, sizeof(int));
7       if(ptr == NULL)
8       {
9           cout << "Error! memory not allocated.";
10          exit(0);
11      }
12
13      cout << "Enter elements of array: ";
14      for(i = 0; i < num; ++i)
15      {
16          cin >> *(ptr + i);
17          sum += *(ptr + i);
18      }
19      cout << "Sum = " << sum;
20      free(ptr);
21
22      cin.get();
23      return 0;
24  }

```

c. Realloc

```

1   int main()
2   {
3       int *ptr, i, n1, n2;
4       cout << "Enter size of array: ";
5       cin >> n1;
6
7       ptr = (int*) malloc(n1 * sizeof(int));
8
9       cout << "Address of previously allocated memory: " << endl;
10      for(i = 0; i < n1; ++i)
11          cout << (ptr + i) << endl;
12
13      cout << "Enter new size of array: ";
14      cin >> n2;

```

15	realloc((int*) ptr, n2);
16	for(i = 0; i < n2; ++i)
17	cout << (ptr + i) << endl;
18	
19	cin.get();
20	return 0;
21	}

2.6 Hasil Percobaan

1. Array dan Struct

a. Array

.....

.....

b. Penggunaan Struktur pada Konversi Koordinat Polar ke Koordinat Cartesian

.....

.....

c. Struct of Array

.....

.....

d. Array of Struct

.....

.....

2. Pointer

a. Mengubah Isi Variabel Melalui Pointer

.....

.....

b. Mengakses dan Mengubah Isi Suatu Variabel Pointer

.....

.....

c. Penggunaan Pointer untuk Bilangan Fibonacci

.....

.....

3. Alokasi Memori

a. Malloc

.....

.....

b. Calloc

.....

-
- c. Realloc
-
-

2.7 Analisis Hasil

1. Array dan Struct

a. Array

Jelaskan bagaimana cara membuat dan mengakses suatu array! Jelaskan fungsi dari pernyataan baris 13-14, 15-16, dan 20-21!

.....

.....

b. Penggunaan Struktur pada Konversi Koordinat Polar ke Koordinat Cartesian

Jelaskan cara membuat objek dari struct dan cara mengakses variabel dari struct! Jelaskan bagaimana cara memberi nilai pada variabel struct berdasarkan masukan dari pengguna!

.....

.....

c. Struct of Array dan Array of Struct

Jelaskan perbedaan antara SoA dan AoS! Jelaskan maksud pernyataan `sizeof(var)/sizeof(*var)!`

.....

.....

2. Pointer

a. Mengubah Isi Variabel Melalui Pointer

Jelaskan dengan gambar blok memori, apa yang terjadi pada blok memori sehingga didapatkan hasil seperti pada pernyataan baris 7-11!

.....

.....

b. Mengakses dan Mengubah Isi Suatu Variabel Pointer

Jelaskan cara mengubah nilai dari suatu variabel dengan menggunakan pointer!

.....

.....

c. Penggunaan Pointer untuk Bilangan Fibonacci

Jelaskan apa yang terjadi pada blok memori ketika pernyataan pada baris ke-12 dijalankan! Dimanakah letak alamat `(fibo+i)` pada pernyataan baris ke-17?

.....

.....

3. Alokasi Memori

a. Malloc dan Calloc

Jelaskan dengan diagram blok memori, apa yang terjadi ketika pernyataan pada baris 6! Dari hasil yang diperoleh pada percobaan Malloc dan Calloc, apakah hasil yang didapatkan sama? Jelaskan mengapa hasilnya seperti itu!

.....

.....

b. Realloc

Jelaskan apa hubungannya antara pernyataan baris ke-7 dengan 15! Apa fungsi dari realloc berdasarkan pernyataan tersebut?

.....

.....

2.8 Kesimpulan

1. Array dan Struct

a. Array

.....

.....

b. Penggunaan Struktur pada Konversi Koordinat Polar ke Koordinat Cartesian

.....

.....

c. Struct of Array

.....

.....

d. Array of Struct

.....

.....

2. Pointer

a. Mengubah Isi Variabel Melalui Pointer

.....

.....

b. Mengakses dan Mengubah Isi Suatu Variabel Pointer

.....

.....

c. Penggunaan Pointer untuk Bilangan Fibonacci

.....

.....

3. Alokasi Memori

a. Malloc

.....
.....

b. Calloc

.....
.....

c. Realloc

.....
.....

2.9 Latihan

1. Array dan Struct

a. Ubahlah nilai dari define MAX dengan nilai lain. Apa hasilnya dan apa fungsi dari statement define?

.....
.....

b. Buatlah fungsi untuk menghitung panjang suatu array!

.....
.....

c. Buatlah data sebanyak 10 mahasiswa dari struct Mahasiswa dan tampilkan seluruh data mahasiswa tersebut! Tampilkan mahasiswa yang memiliki nilai total paling tinggi! Tampilkan rata-rata dari setiap nilai matakuliah!

.....
.....

2. Pointer

```
1  int main()
2  {
3      int i = 5, j = 10;
4      int *ptr;
5      int **pptr;
6
7      ptr = &i;
8      pptr = &ptr;
9      *ptr = 3;
10     **pptr = 7;
11     ptr = &j;
12     **pptr = 9;
13
14     cin.get();
15     return 0;
16 }
```

a. Tulis kode di atas, dan tampilkan keluaran nilai i dan j pada pernyataan baris ke-7 hingga 12!

-
-
- b. Gambar blok memori ketika pernyataan tersebut dijalankan! Jelaskan maksud dari setiap pernyataan tersebut!
-
-

3. Alokasi Memori

```

1  int main()
2  {
3      int *p1 = (int*) malloc(4 * sizeof(int));
4      int *p2 = (int*) malloc(2 * sizeof(int));
5      int *p3 = (int*) malloc(3 * sizeof(int));
6
7      free(p2);
8
9      int *p4 = (int*) malloc(1 * sizeof(int));
10
11     cin.get();
12     return 0;
13 }

```

- a. Tampilkan alamat dari setiap pointer!
-
-
- b. Ubahlah nilai pengali 1 dari baris 9 dengan nilai 2, 3, dan 4! Jalankan kembali dan lihat alamat dari setiap pointer! Apa yang terjadi? Mengapa demikian? Jelaskan!
-
-
- c. Ubahlah pernyataan baris ke-9 dengan menggunakan realloc dari pointer p2 dengan ukuran array baru 1, 2, 3, dan 4, dengan terlebih dahulu menghilangkan pernyataan free(p2)! Jalankan kembali dan lihat alamat dari setiap pointer! Apa yang terjadi? Mengapa demikian? Jelaskan!
-
-

2.10 Tugas

- Masalah aritmetika polinom adalah membuat sekumpulan subrutin manipulasi terhadap polinom simbolis (symbolic Polynomial).

Misalnya:

$$P1 = 6x^8 + 8x^7 + 5x^5 + x^3 + 15$$

$$P2 = 3x^9 + 4x^7 + 3x^4 + 2x^3 + 2x^2 + 10$$

$$P3 = x^2 + 5$$

Terdapat empat operasi aritmetika polinom dasar antara lain:

- Penambahan ($P1 + P2 = 3x^9 + 6x^8 + 12x^7 + 5x^5 + 3x^4 + 3x^3 + 2x^2 + 25$)
- Pengurangan ($P1 - P2 = -3x^9 + 6x^8 + 4x^7 + 5x^5 - 3x^4 - x^3 - 2x^2 + 5$)

c. Perkalian ($P1 * P3 = 6x^{10} + 8x^9 + 5x^7 + x^5 + 15x^2 + 30x^8 + 40x^7 + 25x^5 + 5x^3 + 75 = 6x^{10} + 8x^9 + 30x^8 + 45x^7 + 26x^5 + 5x^3 + 15x^2 + 75$)

d. Turunan ($P2' = 27x^8 + 28x^6 + 12x^3 + 6x^2 + 4x$)

Representasikan bilangan polinom dengan array dan buatlah prosedur-prosedur yang melakukan keempat operasi aritmetika di atas! Representasikan juga dengan menggunakan pointer!

2. Bilangan kompleks berbentuk $a + bi$, dimana a dan b adalah bilangan nyata dan $i^2 = -1$. Terdapat empat operasi aritmatika dasar untuk bilangan kompleks, yaitu:

a. Penambahan : $(a+bi) + (c+di) = (a+c) + (b+d)i$

b. Pengurangan : $(a+bi) - (c+di) = (a-c) + (b-d)i$

c. Perkalian : $(a+bi) * (c+di) = (ac-bd) + (ad+bc)i$

d. Pembagian : $(a+bi) / (c+di) = [(ac+bd) / (a^2+b^2)] + [(bc-ad)/(c^2+d^2)]i$

Tulis program yang membaca dua bilangan kompleks dan simbol operasi yang perlu dilakukan, kemudian lakukan operasi yang diminta! Gunakan struct untuk merepresentasikan bilangan kompleks dan gunakan prosedur untuk implementasi tiap operasi!

DAFTAR PUSTAKA

- Goodrich, M.T., Tamassia, R., dan Mount, D., 2003, "Data Structures and Algorithms in C++ 2nd edition", John Wiley & Sons.
- Sahni, S., 2005, "Data Structures, Algorithms and Applications in C++ 2nd edition", Universities Press.
- Sanjaya, D., 2005, "Asyiknya Belajar Struktur Data di Planet C++", PT Elex Media Komputindo.

Modul 3 : Single Linked List

Waktu Pelaksanaan Praktikum

Durasi kegiatan praktikum = **170 menit**, dengan rincian sebagai berikut :

5. 15 menit untuk pengerjaan Tes Awal atau wawancara Tugas Pendahuluan
6. 60 menit untuk penyampaian materi
7. 45 menit untuk pengerjaan tugas / Studi Kasus
8. 50 menit **Pengayaan**

Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat:

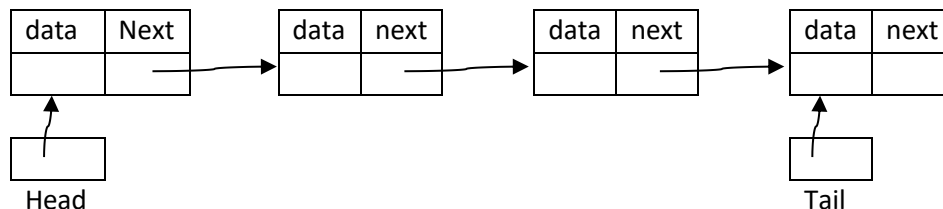
3. Mampu memahami konsep linked list
4. Mampu mengaplikasikan single linked list

Alat & Bahan

4. Komputer
5. Dev C++ IDE

Dasar Teori

Linked List merupakan struktur data yang dibangun dari satu atau lebih node yang menempati alokasi memori secara dinamis. Dalam setiap node menyimpan dua informasi utama yakni data dan pointer yang menunjuk ke node yang lain sehingga membentuk rangkaian node sebagaimana digambarkan berikut :



Jika linked list hanya berisi satu node maka pointer-nya akan menunjuk ke NULL. Jika linked list memiliki lebih dari satu node maka pointer menyimpan alamat dari node berikutnya. Sehingga antara node satu dengan node yang lain akan terhubung. Kecuali node paling ujung akan menunjuk ke NULL.

Single Linked List (SLL)

Single artinya pointer-nya hanya satu buah dan satu arah, yaitu menunjuk ke node sesudahnya. Node terakhir akan menunjuk ke NULL yang akan digunakan sebagai kondisi berhenti pada saat pembacaan isi linked list.

Dari ilustrasi gambar di atas ADT single Linked-list dapat direpresentasikan sebagai berikut :

```
struct node {
    //bagian data
    typedata data 1;
    typedata data 2;
    ...
    typedata data n;
    //pointer ke node selanjutnya
    struct node *next;
};
typedef struct node node;
```

Prosedur Praktikum

A. Percobaan Pertama: Memahami *Node* yang digunakan pada *Single Linked List*

1. Buatlah file percobaan_1.cpp di IDE Dev C++ dan tuliskan kode di bawah :

```
1  #include <cstdlib>
2  #include <iostream>
3  #include <conio.h>
4
5  using namespace std;
6
7  struct node {
8      //bagian data
9      char data;
10     //pointer ke node selanjutnya
11     struct node *next;
12 };
13
14 typedef struct node node;
15 node *head, *tail;
16
17 int main()
18 {
19
20
21
22
23
24
25     getch();
26     return EXIT_SUCCESS;
27 }
```

2. Lakukan beberapa langkah berikut untuk memahami konsep Node pada Single Linked List:
 - a. Tambahkan kode `head = (node *) malloc(sizeof(node));` pada baris ke 19.
 - b. Tambahkan kode `cout << "Data:" << head->data <<endl;` pada baris ke 22, kemudian jalankan program.
 - c. Tambahkan kode `cout << "Pointer:" << head->next <<endl;` pada baris ke 23, kemudian jalankan program.
 - d. Tambahkan kode `head->data='A';` pada baris ke 20, kemudian jalankan program.
 - e. Tambahkan kode `head->next=NULL;` pada baris ke 21, kemudian jalankan program.
3. Tuliskan hasil percobaan, analisis hasil dan kesimpulannya.

B. Percobaan Kedua: Memahami Operasi sederhana pada *Single Linked List*

1. Buatlah file *percobaan_2.cpp* di IDE Dev C++ dan Tuliskan kode di bawah :

1	#include <cstdlib>
2	#include <iostream>
3	#include <conio.h>
4	
5	using namespace std;
6	
7	struct node {
8	//bagian data
9	char data;
10	//pointer ke node selanjutnya
11	struct node *next;
12	};
13	typedef struct node node;
14	node *head, *tail;
15	
16	void AddLast(char item){ // Add Node After Tail
17	struct node *pNew;
18	pNew = (node *) malloc(sizeof(node));
19	pNew->data = item;
20	if (head == NULL){
21	head= pNew;
22	}
23	else {
24	tail->next = pNew;
25	}
26	tail=pNew;
27	}
28	
29	void AddFirst(char item){ // AddNodeBeforeHead
30	struct node *pNew;
31	pNew = (node *) malloc(sizeof(node));
32	pNew->data = item;
33	if (head == NULL){
34	tail= pNew;
35	}
36	else {
37	pNew->next = head;
38	}
39	head=pNew;
40	}

2. Tambahkan method main pada file *percobaan_2.cpp* dengan dengan kode sebagai berikut kemudian jalankan.

1	int main()
2	{

3	head=tail=NULL;
4	AddFirst('A');
5	cout << "head : " << head->data <<endl;
6	cout << "tail : " << tail->data <<endl;
7	AddFirst('B');
8	cout << "head : " << head->data <<endl;
9	cout << "tail : " << tail->data <<endl;
10	AddFirst ('C');
11	cout << "head : " << head->data <<endl;
12	cout << "tail : " << tail->data <<endl;
13	AddFirst ('D');
14	cout << "head : " << head->data <<endl;
15	cout << "tail : " << tail->data <<endl;
16	
17	getch();
18	return EXIT_SUCCESS;
19	}

3. Ganti isi dari method main pada file *percobaan_2.cpp* dengan kode sebagai berikut kemudian jalankan.

1	int main()
2	{
3	head=tail=NULL;
4	AddLast ('A');
5	cout << "head : " << head->data <<endl;
6	cout << "tail : " << tail->data <<endl;
7	AddLast ('B');
8	cout << "head : " << head->data <<endl;
9	cout << "tail : " << tail->data <<endl;
10	AddLast ('C');
11	cout << "head : " << head->data <<endl;
12	cout << "tail : " << tail->data <<endl;
13	AddLast ('D');
14	cout << "head : " << head->data <<endl;
15	cout << "tail : " << tail->data <<endl;
16	
17	getch();
18	return EXIT_SUCCESS;
19	}

4. Tuliskan hasil percobaan, analisis hasil dan kesimpulannya!

Hasil Percobaan

1. Tuliskan hasil dari percobaan pertama di atas!
2. Tuliskan hasil dari percobaan kedua di atas!

Analisis Hasil

1. Tuliskan Analisis hasil dari percobaan pertama di atas!
2. Tuliskan Analisis hasil dari percobaan kedua di atas!

Kesimpulan

1. Tuliskan kesimpulan dari percobaan pertama di atas!
2. Tuliskan kesimpulan dari percobaan kedua di atas!

Latihan

Operasi pada Single Linked List secara lengkap adalah sebagai berikut:

1. Penambahan
2. Penghapusan
3. Penyisipan
4. Pencarian
5. Pengaksesan

Lengkapi kode pada file *percobaan_2.cpp* di atas untuk operasi-operasi (method) yang belum ada pada Single Linked List!

Tugas

1. Modifikasilah program pada file *percobaan_2.cpp* di atas sehingga dapat menampung data struct Mahasiswa sebagai berikut!

Mahasiswa
String nim
String nama
double ipk

2. Tambahkan method penyisipan data yang bisa membentuk linked listurut sejak awal dengan pengurutan berdasarkan ipk!

DAFTAR PUSTAKA

- Goodrich, M.T., Tamassia, R., dan Mount, D., 2003, "Data Structures and Algorithms in C++ 2nd edition", John Wiley & Sons.
- Sahni, S., 2005, "Data Structures, Algorithms and Applications in C++ 2nd edition", Universities Press.
- Sanjaya, D., 2005, "Asyiknya Belajar Struktur Data di Planet C++", PT Elex Media Komputindo.

Modul 4 : ADT Double Linked List

Waktu Pelaksanaan Praktikum

Durasi kegiatan praktikum = **170 menit**, dengan rincian sebagai berikut:

9. 15 menit untuk pengerjaan Tes Awal atau wawancara Tugas Pendahuluan
10. 60 menit untuk penyampaian materi
11. 45 menit untuk pengerjaan tugas / Studi Kasus
12. 50 menit **Pengayaan**

Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat:

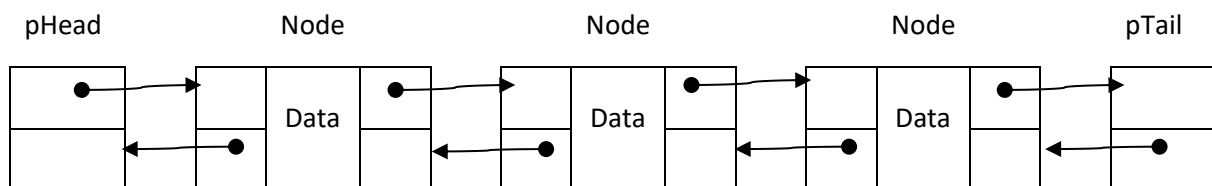
5. Mampu memahami konsep double linked list
6. Mampu mengaplikasikan double linked list

Alat & Bahan

6. Komputer
7. Dev C++ IDE

Dasar Teori

Double Linked List merupakan representasi data yang disimpan dalam suatu rangkaian node dimana setiap node mempunyai penunjuk ke node sebelumnya dan sesudahnya. Double : artinya field pointer-nya dua buah dan dua arah, yang menunjuk ke node sebelum dan sesudahnya. Berguna bila perlu melakukan pembacaan linkedlist dari dua arah. Double linked list memiliki 2 buah pointer yaitu pointer next dan prev. Pointer next : mengarah ke node belakang (tail). Pointer prev : mengarah ke node depan (head). Ilstrasi double Linked List dapat digambarkan berikut :



Ketika masih ada satu node maka kedua pointer (next dan prev) akan menunjuk ke NULL. Ketika double linked list memiliki banyak node maka node yang paling depan pointer prev-nya menunjuk ke NULL. Sedangkan node yang paling belakang pointer next-nya yang menunjuk ke NULL. Pada double linked dibutuhkan pointer bantu yaitu head dan tail. Head : menunjuk pada node yang paling depan. Tail : menunjuk pada node yang paling belakang. Dari ilustrasi gambar di atas ADT double Linked-list dapat direpresentasikan sebagai berikut :

```
struct node {  
    //bagian data  
    typedata data 1;  
    typedata data 2;  
    ...  
    typedata data n;  
    //pointer ke node selanjutnya  
    struct node *next;  
    struct node *prev;  
};  
typedef struct node node;
```


Prosedur Praktikum

C. Percobaan Pertama: Memahami *Node* yang digunakan pada *Double Linked List*

1. Buatlah file percobaan_1.cpp di IDE Dev C++ dan tuliskan kode di bawah :

```
1  #include <cstdlib>
2  #include <iostream>
3  #include <conio.h>
4
5  using namespace std;
6
7  struct node {
8      //bagian data
9      char data;
10     //pointer ke node selanjutnya
11     struct node *next;
12     struct node *prev;
13 };
14 typedef struct node node;
15 node *head, *tail;
16
17 int main()
18 {
19
20
21
22
23
24
25     getch();
26     return EXIT_SUCCESS;
27 }
```

2. Lakukan beberapa langkah berikut untuk memahami konsep Node pada Double Linked List:
 - a. Tambahkan kode `head = (node *) malloc(sizeof(node));` pada baris ke 19.
 - b. Tambahkan kode `cout << "Data:" << head->data <<endl;` pada baris ke 22, kemudian jalankan program.
 - c. Tambahkan kode `cout << "Pointer next:" << head->next <<endl;` pada baris ke 23, dan `cout << "Pointer prev:" << head->prev <<endl;` pada baris ke 24 kemudian jalankan program.
 - d. Tambahkan kode `head->data='A';` pada baris ke 20, kemudian jalankan program.
 - e. Tambahkan kode `head->next=head->prev=NULL;` pada baris ke 21, kemudian jalankan program.
3. Tuliskan hasil percobaan, analisis hasil dan kesimpulannya.

D. Percobaan Kedua: Memahami Operasi sederhana pada Double Linked List

5. Buatlah file *percobaan_2.cpp* di IDE Dev C++ dan Tuliskan kode di bawah :

```
1  #include <cstdlib>
2  #include <iostream>
3  #include <conio.h>
4
5  using namespace std;
6
7  struct node {
8      //bagian data
9      char data;
10     //pointer ke node selanjutnya
11     struct node *next;
12     struct node *prev;
13 };
14 typedef struct node node;
15 node *head, *tail;
16
17 void AddLast(char item){ // Add Node After Tail
18     struct node *pNew;
19     pNew = (node *) malloc(sizeof(node));
20     pNew->data = item;
21     if (head == NULL){
22         head= pNew;
23     }
24     else {
25         tail->next = pNew;
26         pNew->prev=tail;
27     }
28     tail=pNew;
29 }
30
31 void AddFirst(char item){ // AddNodeBeforeHead
32     struct node *pNew;
33     pNew = (node *) malloc(sizeof(node));
34     pNew->data = item;
35     if (head == NULL){
36         tail= pNew;
37     }
38     else {
39         pNew->next = head;
40         head->prev=pNew;
41     }
42     head=pNew;
43 }
```

6. Tambahkan method main pada file *percobaan_2.cpp* dengan dengan kode sebagai berikut kemudian jalankan.

```
1  int main()
2  {
3      head=tail=NULL;
4      AddFirst('A');
5      cout << "head : " << head->data <<endl;
6      cout << "tail : " << tail->data <<endl;
7      AddFirst('B');
8      cout << "head : " << head->data <<endl;
9      cout << "tail : " << tail->data <<endl;
10     AddFirst ('C');
11     cout << "head : " << head->data <<endl;
12     cout << "tail : " << tail->data <<endl;
13     AddFirst ('D');
14     cout << "head : " << head->data <<endl;
15     cout << "tail : " << tail->data <<endl;
16
17     getch();
18     return EXIT_SUCCESS;
19 }
```

7. Ganti isi dari method main pada file *percobaan_2.cpp* dengan kode sebagai berikut kemudian jalankan.

```
1  int main()
2  {
3      head=tail=NULL;
4      AddLast ('A');
5      cout << "head : " << head->data <<endl;
6      cout << "tail : " << tail->data <<endl;
7      AddLast ('B');
8      cout << "head : " << head->data <<endl;
9      cout << "tail : " << tail->data <<endl;
10     AddLast ('C');
11     cout << "head : " << head->data <<endl;
12     cout << "tail : " << tail->data <<endl;
13     AddLast ('D');
14     cout << "head : " << head->data <<endl;
15     cout << "tail : " << tail->data <<endl;
16
17     getch();
18     return EXIT_SUCCESS;
19 }
```

8. Tuliskan hasil percobaan, analisis hasil dan kesimpulannya!

Hasil Percobaan

3. Tuliskan hasil dari percobaan pertama di atas!
4. Tuliskan hasil dari percobaan kedua di atas!

Analisis Hasil

3. Tuliskan Analisis hasil dari percobaan pertama di atas!
4. Tuliskan Analisis hasil dari percobaan kedua di atas!

Kesimpulan

3. Tuliskan kesimpulan dari percobaan pertama di atas!
4. Tuliskan kesimpulan dari percobaan kedua di atas!

Latihan

Operasi pada Double Linked List secara lengkap adalah sebagai berikut:

6. Penambahan
7. Penghapusan
8. Penyisipan
9. Pencarian
10. Pengaksesan

Lengkapi kode pada file *percobaan_2.cpp* diatas untuk operasi-operasi (method) yang belum ada pada Double Linked List!

Tugas

3. Modifikasilah program pada file *percobaan_2.cpp* di atas sehingga dapat menampung data struct Mahasiswa sebagai berikut!

Mahasiswa
String nim
String nama
double ipk

4. Gunakan dan modifikasi method penyisipan data pada tugas di modul 3 yang bisa membentuk linked listurut sejak awal dengan pengurutan berdasarkan ipk!
5. Lengkapi method pada file *percobaan_2.cpp* yang bisa menampilkan data secara *ascending* dan *descending* berdasarkan ipk!

DAFTAR PUSTAKA

- Goodrich, M.T., Tamassia, R., dan Mount, D., 2003, "Data Structures and Algorithms in C++ 2nd edition", John Wiley & Sons.
- Sahni, S., 2005, "Data Structures, Algorithms and Applications in C++ 2nd edition", Universities Press.
- Sanjaya, D., 2005, "Asyiknya Belajar Struktur Data di Planet C++", PT Elex Media Komputindo.

Modul 5 : Circular Linked List

Waktu Pelaksanaan Praktikum

Durasi kegiatan praktikum = **170 menit**, dengan rincian sebagai berikut:

13. 15 menit untuk pengerjaan Tes Awal atau wawancara Tugas Pendahuluan
14. 30 menit untuk penyampaian materi
15. 65 menit untuk pengerjaan tugas / Studi Kasus
16. 60 menit **Pengayaan**

Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat:

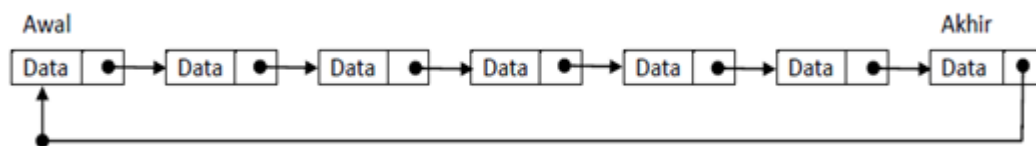
7. Mampu membedakan antara Linked List dengan Circular Linked List
8. Mampu menerapkan Circular Single Linked List dan Circular Double Linked List
9. Mampu mengimplementasikan Circular Linked List pada suatu studi kasus

Alat & Bahan

8. Komputer
9. Dev C++
10. GCC

Dasar Teori

Salah satu varian dari Linked List adalah Circular Linked List yang mempunyai dua field setiap node-nya, yaitu field pointer yang menunjuk ke node setelahnya dan sebuah field yang berisi data untuk node tersebut. Perbedaan antara Linked List dengan Circular Linked List adalah pada Circular Linked List pointer node terakhirnya mengarah ke node awal. Adapun struktur dari Circular Linked List ditampilkan pada gambar berikut.



Circular Linked List terdiri dari Circular Single Linked List dan Circular Double Linked List. Pada gambar di atas merupakan contoh Circular Single Linked List, karena hanya memiliki satu buah pointer dan arahnya hanya ke node setelahnya. Ketika node baru terbentuk, pointer-nya menunjuk ke dirinya sendiri. Jika sudah lebih dari satu node, maka pointer-nya menunjuk ke node setelahnya dan pointer node akhir menuju ke node awal.

Adapun UDT dari Circular Single Linked List adalah sebagai berikut.

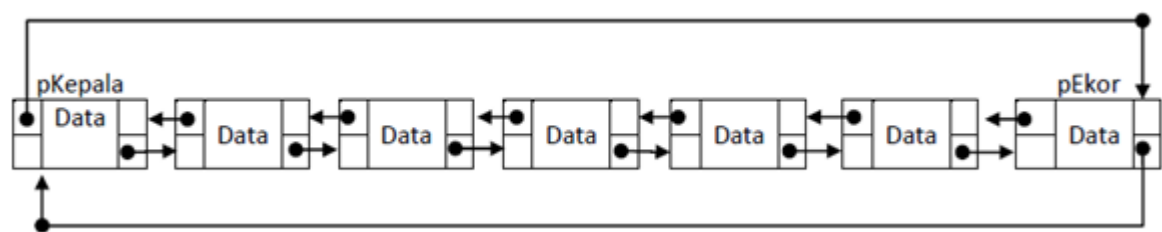
NodeCSLL
Object data
NodeCSLL setelah

CircularLinkedList
NodeCSLL pAwal
NodeCSLL pAkhir
int jumlah
CircularLinkedList ()
public void sisipDtDiAwal(Object dt)
public void sisipDtDiAkhir(Object dt)
public void hapusDt(Object dtHapus)
public void hapusSatuDataDiAwal()
public void hapusSatuDataDiAkhir()
public void cetakDt(String komentar)

Berbeda dengan Circular Single Linked List, Circular Double Linked List mempunyai dua pointer, yaitu pointer yang mengarah ke node setelahnya dan ke node sebelumnya. Pada pembentukan

node baru, kedua pointer mengarah ke dirinya sendiri. Jika sudah lebih dari satu node, maka pointer pada node baru akan mengarah ke node setelahnya dan node sebelumnya. Pada node akhir pointer setelahnya mengarah ke node awal, dan pada node awal pointer sebelumnya mengarah ke node akhir.

Adapun struktur dan UDT dari Circular Double Linked List adalah sebagai berikut.



NodeCDLL
Object data
NodeCDLL sebelum
NodeCDLL setelah

CircularDLinkedList
NodeCDLL pAwal
NodeCDLL pAkhir
int jumlah
CircularDLinkedList ()
public void sisipDtDiAwal(Object dt)
public void sisipDtDiAkhir(Object dt)
public void hapusDt(Object dtHapus)
public void cetakDt(String komentar)

Prosedur Praktikum

A. Percobaan Pertama: Implementasi Circular Single Linked List

1. Buatlah proyek di IDE DevCpp dengan nama CircularSingleLinkedList.
2. Buatlah Struct CircularSingleLinkedList di proyek yang telah dibuat, dan tambahkan Struct NodeCSLL seperti kode di bawah ini sesuai dengan UDT Struct NodeCSLL.

1	<code>struct NodeCSLL {</code>
2	<code>int data;</code>
3	<code>NodeCSLL *setelah;</code>
4	<code>}</code>
5	<code>struct CircularSingleLinkedList {</code>
6	
7	
8	<code>}</code>

3. Tambahkan beberapa fungsi sesuai dengan UDT Struct CircularLinkedList dan implementasikan sesuai dengan kode di bawah ini.

1	<code>struct NodeCSLL {</code>
2	<code>int data;</code>
3	<code>NodeCSLL *setelah;</code>
4	<code>}</code>
5	<code>struct CircularSingleLinkedList {</code>
6	<code>NodeCSLL *pAwal, *pAkhir;</code>
7	<code>int jumlah;</code>
8	<code>CircularSingleLinkedList(){</code>
9	<code>pAwal = NULL;</code>
10	<code>pAkhir = NULL;</code>
11	<code>jumlah = -1;</code>
12	<code>}</code>
13	<code>void sisipDataDiAwal(int data){</code>
14	<code>NodeCSLL *pBaru = new NodeCSLL();</code>
15	<code>pBaru->data = data;</code>
16	<code>pBaru->setelah = pBaru;</code>

```

17     if (pAwal == NULL) {
18         pAwal = pBaru;
19         pAkhir = pBaru;
20         jumlah = 0;
21     } else {
22         pBaru->setelah = pAwal;
23         pAkhir->setelah = pBaru;
24         pAwal = pBaru;
25         jumlah++;
26     }
27 }
28 void sisipDataDiAkhir(int data){
29     // lengkapi bagian ini
30 }
31 void hapusData(int dtHapus){
32     if(pAwal != NULL) {
33         NodeCSLL *pSbl, *pHapus;
34         pSbl = NULL; pHapus = pAwal;
35         bool ketemu = false;
36         int i = 0;
37         while(!ketemu && (i <= jumlah)){
38             if (pHapus->data == dtHapus) {
39                 ketemu = true;
40             }
41             else {
42                 pSbl = pHapus;
43                 pHapus = pHapus->setelah;
44             }
45             i++;
46         }
47         if (ketemu){
48             if(pSbl == null) {
49                 pAwal = pHapus->setelah;
50                 pAkhir->setelah = pAwal;
51                 delete pHapus;
52             } else {
53                 if (pAkhir == pHapus) {
54                     pAkhir = pSbl;
55                 }
56                 pSbl->setelah = pHapus->setelah;
57                 delete pHapus;
58             }
59             jumlah--;
60         }
61     }
62 }
63 void hapusSatuDataDiAwal(){
64     // lengkapi bagian ini
65 }
66 void hapusSatuDataDiAkhir(){
67     // lengkapi bagian ini
68 }
69 void cetak(string komentar){
70     cout << komentar << endl;
71     NodeCSLL *pCetak;
72     pCetak = pAwal;
73     int i = -1;

```

```

74     while((i < jumlah) ){
75         cout << pCetak->data << "->";
76         pCetak = pCetak->setelah;
77         i++;
78     }
79     cout << endl;
80 }
81 int main() {
82     CircularSingleLinkedList *csll =
83         new CircularSingleLinkedList();
84     csll->sisipDataDiAwal(50);
85     csll->sisipDataDiAwal(60);
86     csll->sisipDataDiAwal(70);
87     csll->sisipDataDiAwal(8);
88     csll->sisipDataDiAwal(9);
89     csll->sisipDataDiAwal(90);
90     csll->sisipDataDiAwal(19);
91     csll->cetak("csll Asal");
92     csll->hapusData(8);
93     csll->cetak("csll stl 8 dihapus");
94     csll->hapusData(90);
95     csll->cetak("csll stl 90 dihapus");
96 }

```

4. Lakukan kompilasi dan tuliskan hasil percobaan, analisis hasil, dan kesimpulannya.

B. Percobaan Kedua: Implementasi Circular Double Linked List

1. Buatlah projek di IDE DevCpp dengan nama CircularDoubleLinkedList.
2. Buatlah Struct CircularDoubleLinkedList di projek yang telah dibuat, dan tambahkan Struct NodeCDLL seperti kode di bawah ini sesuai dengan UDT Class NodeCDLL.

```

1 struct NodeCDLL {
2     int data;
3     NodeCDLL *sebelum;
4     NodeCDLL *setelah;
5 }
6 struct CircularDoubleLinkedList {
7
8
9 }

```

3. Tambahkan beberapa prosedur sesuai dengan UDT Class CircularDLinkedList dan implementasikan sesuai dengan kode di bawah ini.

```

1 struct NodeCDLL {
2     int data;
3     NodeCDLL *sebelum;
4     NodeCDLL *setelah;
5 }
6 struct CircularDoubleLinkedList {
7     NodeCSLL *pAwal, *pAkhir;
8     int jumlah;
9     CircularDoubleLinkedList(){
10         pAwal = NULL;
11         pAkhir = NULL;
12         jumlah = -1;
13     }
14     void sisipDataDiAwal(int data){
15         NodeCDLL *pBaru = new NodeCDLL();

```



```

16     pBaru->data = data;
17     pBaru->sebelum = pBaru;
18     pBaru->setelah = pBaru;
19     if (pAwal == NULL){
20         pAwal = pBaru;
21         pAkhir = pBaru;
22         jumlah = 0;
23     } else {
24         pBaru->sebelum = pAkhir;
25         pBaru->setelah = pAwal;
26         pAwal->sebelum = pBaru;
27         pAkhir->setelah = pBaru;
28         pAwal = pBaru;
29         jumlah++;
30     }
31 }
32 void sisipDataDiAkhir(int data){
33     // lengkapi bagian ini
34 }
35 void hapusData(int dtHapus){
36     // lengkapi bagian ini
37 }
38 void cetak(string komentar){
39     cout << komentar << endl;
40     NodeCDLL *pCetak;
41     pCetak = pAwal;
42     int i = -1;
43     while((i < jumlah) ){
44         cout << pCetak->data << "->";
45         pCetak = pCetak->setelah;
46         i++;
47     }
48     cout << endl;
49 }
50 int main() {
51     CircularDoubleLinkedList *cdll =
52         new CircularDoubleLinkedList();
53     cdll->sisipDataDiAwal(70);
54     cdll->sisipDataDiAwal(60);
55     cdll->sisipDataDiAwal(50);
56     cdll->sisipDataDiAwal(8);
57     cdll->sisipDataDiAwal(9);
58     cdll->sisipDataDiAwal(19);
59     cdll->sisipDataDiAwal(90);
60     cdll->cetak("cdll Asal");
61 }

```

4. Lakukan kompilasi dan tuliskan hasil percobaan, analisis hasil, dan kesimpulannya.

Hasil Percobaan

1. Tuliskan hasil dari percobaan pertama di atas!
2. Tuliskan hasil dari percobaan kedua di atas!

Analisis Hasil

4. Percobaan pertama

- d. Jelaskan apa fungsi dari variabel jumlah! Mengapa nilai jumlah ada yang 0, 1, dan di-increment?
 - e. Jelaskan fungsi if-else pada baris ke-17 hingga 26! Dalam kondisi apa pernyataan di dalam if dan else dijalankan?
 - f. Jelaskan fungsi dari variabel pSbl dan pHapus pada fungsi hapusData! Mengapa diperlukan kedua variabel tersebut!
 - g. Jelaskan apa yang terjadi apabila kondisi pada baris ke-48 diganti menjadi "pSbl != null" dan isi pernyataan dari if dan else ditukar!
5. Percobaan kedua
- a. Jelaskan apa kegunaan dari fungsi pada baris ke-9!
 - b. Jelaskan apa yang terjadi apabila pernyataan pada baris ke-24 hingga 27 ditukar-tukar posisinya!

Kesimpulan

1. Tuliskan kesimpulan dari percobaan pertama di atas!
2. Tuliskan kesimpulan dari percobaan kedua di atas!

Latihan

1. Lengkapi kode Class CircularSingleLinkedList dan CircularDoubleLinkedList di atas pada bagian prosedur yang memiliki komentar 'lengkapi bagian ini'!
2. Tambahkan perintah untuk menjalankan prosedur tersebut di method main untuk mengetahui apakah prosedur tersebut sudah benar!

Tugas

1. Modifikasi program Latihan CircularSingleLinkedList dan CircularDoubleLinkedList tanpa menggunakan variabel jumlah!
2. Modifikasi program Latihan CircularSingleLinkedList dan CircularDoubleLinkedList tanpamenggunakan pointer pAkhir!

DAFTAR PUSTAKA

- Goodrich, M.T., Tamassia, R., dan Mount, D., 2003, "Data Structures and Algorithms in C++ 2nd edition", John Wiley & Sons.
- Sahni, S., 2005, "Data Structures, Algorithms and Applications in C++ 2nd edition", Universities Press.
- Sanjaya, D., 2005, "Asyiknya Belajar Struktur Data di Planet C++", PT Elex Media Komputindo.

Modul 6 : ADT Stack

7.1 Waktu Pelaksanaan Praktikum

Durasi kegiatan praktikum = **170 menit**, dengan rincian sebagai berikut:

- 15 menit untuk pengerjaan Tes Awal atau wawancara Tugas Pendahuluan
- 60 menit untuk penyampaian materi
- 45 menit untuk pengerjaan tugas / Studi Kasus
- 50 menit Pengayaan

7.2 Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat:

1. Mampu membuat stack menggunakan array
2. Mampu membuat stack menggunakan linked list

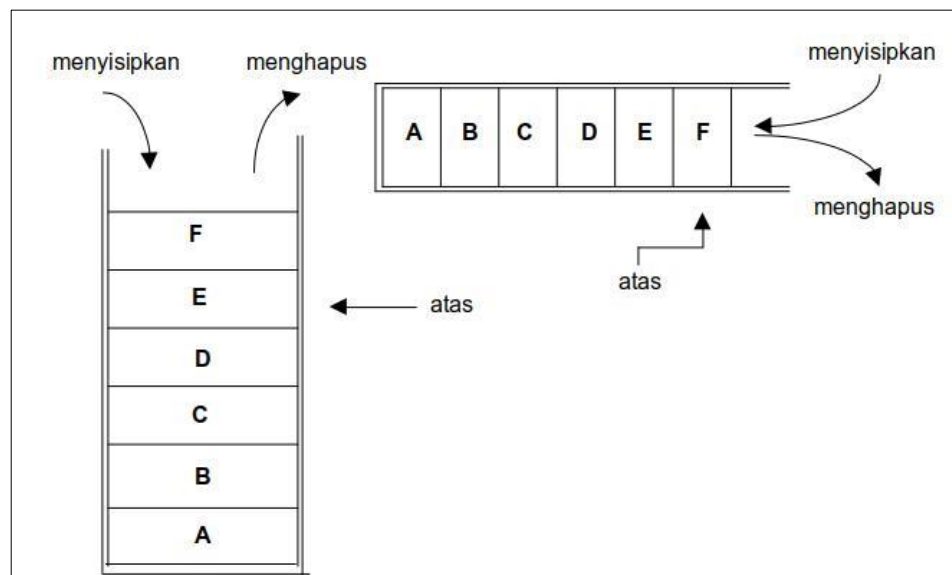
7.3 Alat & Bahan

1. Komputer
2. C++ compiler

7.4 Dasar Teori

A. Stack

Stack atau tumpukan adalah kumpulan data yang hanya bisa dilakukan penambahan (penyisipan) data dan penghapusan data pada salah satu ujung yang sama.



Dengan memperlihatkan ilustrasi-ilustrasi yang disebutkan maka kita bisa melihat bahwa stack merupakan suatu list yang mempunyai karakteristik “masuk terakhir keluar pertama” (last in first out – LIFO).

Operasi dasar pada stack antara lain pop dan push. **Push** adalah operasi menambah/menyisipkan item pada stack. **Pop** adalah proses mengambil item dari stack. Stack bisa dibuat dengan Array atau Linked List. Tabel dibawah memberikan ilustrasi operasi pada stack yang dibuat dengan array.

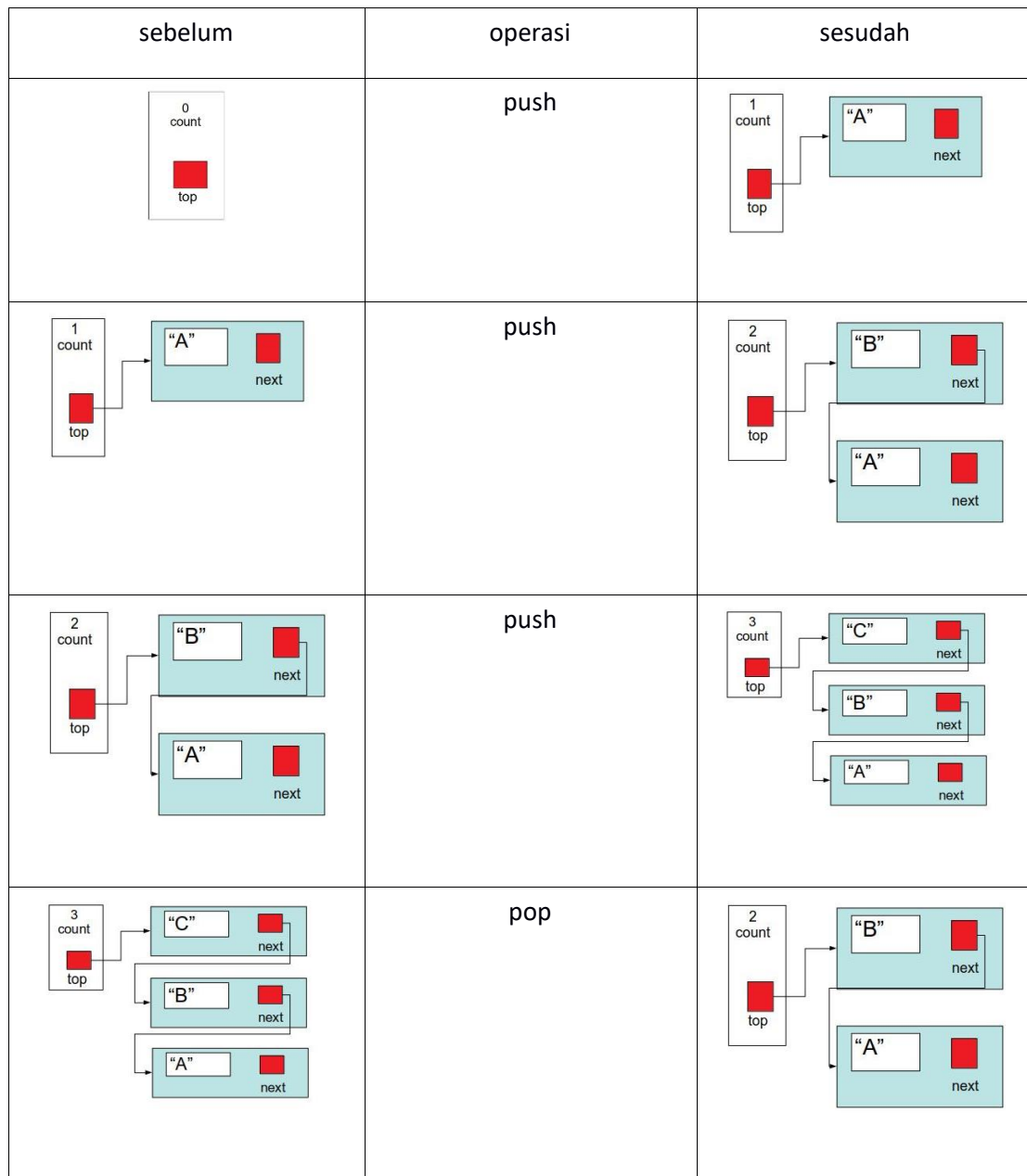
sebelum	operasi	sesudah
[]	push("A")	["A"]
["A"]	push("B")	["A","B"]
["A","B"]	push("C")	["A","B","C"]
["A","B","C"]	pop()	["A","B"]
["A","B"]	push("D")	["A","B","D"]
["A","B","D"]	pop()	["A","B"]

Dalam implementasi stack ini disimpan variabel **count** yang menghitung index atau nomor urut terakhir item dalam stack. Operasi push menambahkan item ke dalam array pada index/urutan **count+1** atau **setelah item terakhir** dalam array. Operasi pop akan mengambil item dari dalam array pada index **count** atau **item terakhir** dalam array.

Pada tabel dibawah diilustrasikan operasi stack menggunakan array. Perbedaan dengan stack diatas adalah implementasi proses **push** akan menambahkan item pada **array index ke 0**. Proses **pop** akan mengambil item pada **array index ke 0**.

sebelum	operasi	sesudah
[]	push("A")	["A"]
["A"]	push("B")	["B","A"]
["B","A"]	push("C")	["C","B","A"]
["C","B","A"]	pop()	["B","A"]
["B","A"]	push("D")	["D","B","A"]
["D","B","A"]	pop()	["B","A"]

Gambar dibawah memberikan ilustrasi stack menggunakan single linked list. Top merupakan pointer yang menunjuk pada item paling atas yang merupakan item terakhir yang dimasukkan dalam stack. Dalam islutrasi ini operasi **push** pada stack menggunakan operasi **add first** pada single linked list dan operasi **pop** menggunakan operasi **remove first**. Jika operasi **push** menggunakan operasi **add last** maka operasi **pop** akan menggunakan **remove last**.



7.5 Prosedur Praktikum

1. Buatlah file percobaan stack_array.cpp berikut

```

1  #include <iostream>
2  #include <stdlib.h>
3
4  #define MAX_SIZE 5
5  using namespace std;
6  int choice, i;

```

```

7  char item;
8  char arr_stack[MAX_SIZE];
9  int count = 0;
10 int keluar = 0;
11
12 void push(char item)
13 {
14     if (count == MAX_SIZE)
15     {
16         cout << "\n# Stack Penuh";
17     }
18     else
19     {
20         .....
21         .....
22         .....
23     }
24 }
25
26 void pop()
27 {
28     if (count == 0)
29         cout << "\n## Stack kosong";
30     else
31     {
32         .....
33         .....
34     }
35 }
36
37 void printAll()
38 {
39     cout << "\n## Stack Size : " << count;
40     for (i = (count - 1); i >= 0; i--)
41         cout << "\n## No Urut/index : " << i << ", Value : "
42         << arr_stack[i];
43 }
44
45 void menu()
46 {
47     cout << "\nMasukkan operasi yang akan dilakukan
48     (1:push, 2:pop, 3:print) : ";
49     cin >> choice;
50     switch (choice)
51     {
52     case 1:
53     {
54         cout << "\nMasukkan huruf yang akan dipush : ";

```


54	cin >> item;
55	push(item);
56	}
57	break;
58	case 2:pop();
59	break;
60	case 3:printAll();
61	break;
62	default:
63	cout << "\n1:push, 2:pop, 3:print\n";
64	keluar = 1;
65	break;
66	}
67	}
68	
69	int main()
70	{
71	do{
72	menu();
73	} while (keluar == 0);
74	}

2. Tambahkan kode berikut pada baris 20 sampai 22

1	arr_stack[count] = item;
2	cout << "\n# PUSH No urut/index : " << count << ", Push : " << item;
3	count++;

3. Tambahkan kode berikut pada baris 32 dan 33

1	--count;
2	cout << "\n##POP No urut/index : " << count << ", Value : " << arr_stack[count];

4. Jalankan program dan

- pilih menu push dan masukkan "A"
- pilih menu push dan masukkan "B"
- pilih menu push dan masukkan "C"
- pilih menu print
- pilih menu pop
- pilih menu print
- pilih menu push dan masukkan "D"
- pilih menu print

5. Buatlah program stack_llist.cpp berikut :

```

1  #include <iostream>
2  #include <stdlib.h>
3  using namespace std;
4  struct node
5  {
6      char data;
7      struct node *next;
8  };
9  typedef struct node node;
10 node *top;
11
12 int choice;
13 char item;
14 int count = 0;
15 int keluar = 0;
16
17 void push(char item)
18 {
19     node *temp = new node;
20     temp->data = item;
21     temp->next = NULL;
22     if (top == NULL)
23     {
24         top = temp;
25         temp = NULL;
26     }
27     else
28     {
29         .....
30         .....
31     }
32     cout << "\n# PUSH : No urut/index : " << count << ",
33 Push : " << item;
34     count++;
35 }
36
37 void pop()
38 {
39     if (top==NULL)
40     cout << "\n## Stack kosong";
41     else
42     {
43         --count;
44         char item=top->data;
45         node *temp = new node;
46         .....
47         .....

```

```

48     cout << "\n##POP hasil:" << item;
49     cout << "\n##jumlah item dalam stack : " << count ;
50 }
51 }
52
53 void printAll()
54 {
55     cout << "\n## Stack Size : " << count;
56     node *temp=new node;
57     temp=top;
58     int i=count;
59     while(temp!=NULL)
60     {
61         cout << "\n## No Urut/index : " << i << ", Value : "
62         << temp->data;
63         temp=temp->next;
64         i--;
65     }
66
67 void menu()
68 {
69     cout << "\nMasukkan operasi yang akan dilakukan
70     (1:push, 2:pop, 3:print) : ";
71     cin >> choice;
72     switch (choice)
73     {
74     case 1:
75     {
76         cout << "\nMasukkan huruf yang akan dipush : ";
77         cin >> item;
78         push(item);
79         break;
80     case 2:pop();
81     break;
82     case 3:printAll();
83     break;
84     default:
85         cout << "\n1:push, 2:pop, 3:print\n";
86         keluar = 1;
87         break;
88     }
89 }
90
91 int main()
92 {
93     do

```

94	{
95	menu();
96	} while (keluar == 0);
97	}

6. Tambahkan kode berikut pada baris 29 dan 30

1	temp->next = top;
2	top = temp;

7. Tambahkan kode berikut pada baris 46 dan 47

1	temp=top->next;
2	top=temp;

8. Jalankan program kemudian :

- pilih menu push dan masukkan "A"
- pilih menu push dan masukkan "B"
- pilih menu push dan masukkan "C"
- pilih menu print
- pilih menu pop
- pilih menu print
- pilih menu push dan masukkan "D"
- pilih menu print

7.6 Hasil Percobaan

1. Tuliskan hasil dari percobaan pertama diatas.
2. Tuliskan hasil dari percobaan kedua diatas.

7.7 Analisis Hasil

1. Tuliskan Analisis hasil dari percobaan pertama diatas.
2. Tuliskan Analisis hasil dari percobaan kedua diatas.

7.8 Kesimpulan

1. Tuliskan kesimpulan dari percobaan pertama diatas.
2. Tuliskan kesimpulan dari percobaan kedua diatas.

7.9 Latihan

1. Buat pseudocode untuk operasi pop dan push pada stack menggunakan array dengan ketentuan operasi push akan menambahkan item pada array index ke 0 dan operasi pop akan mengembalikan item pada array index ke 0
2. Buat pseudocode untuk operasi push dan pop stack menggunakan single linked list dengan ketentuan operasi push menggunakan add last pada lisnked list dan operasi pop akan menggunakan remove last.

7.10 Tugas

1. Modifikasi program `stack_array.cpp` sehingga operasi push akan menambahkan item pada array index ke 0 dan operasi pop akan mengembalikan item pada array index ke 0.
2. Modifikasi program `stack_llist.cpp` sehingga operasi push menggunakan add last pada linked list dan operasi pop akan menggunakan remove last.

DAFTAR PUSTAKA

- Goodrich, M.T., Tamassia, R., dan Mount, D., 2003, "Data Structures and Algorithms in C++ 2nd edition", John Wiley & Sons.
- Sahni, S., 2005, "Data Structures, Algorithms and Applications in C++ 2nd edition", Universities Press.
- Sanjaya, D., 2005, "Asyiknya Belajar Struktur Data di Planet C++", PT Elex Media Komputindo.

Modul 7 : ADT Queue

8.1 Waktu Pelaksanaan Praktikum

Durasi kegiatan praktikum = **170 menit**, dengan rincian sebagai berikut:

- 15 menit untuk pengerjaan Tes Awal atau wawancara Tugas Pendahuluan
- 60 menit untuk penyampaian materi
- 45 menit untuk pengerjaan tugas / Studi Kasus
- 50 menit Pengayaan

8.2 Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat:

1. Mampu membuat queue menggunakan array
2. Mampu membuat queue menggunakan linked list

8.3 Alat & Bahan

1. Komputer
2. C++ compiler

8.4 Dasar Teori

Queue atau antrian adalah kumpulan data yang bisa dilakukan penambahan (penyisipan) data pada satu sisi dan penghapusan data pada sisi/ujung yang lain.



Dengan memperhatikan ilustrasi diatas maka kita bisa melihat bahwa queue merupakan suatu list yang mempunyai karakteristik “masuk pertama keluar pertama” (first in first out – FIFO). Struktur data ini banyak dipakai dalam informatika misalnya untuk merepresentasi :

1. Antrian job dalam sistem operasi
2. Antrian dalam dunia nyata

Operasi dasar pada queue antara lain **enqueue** dan **dequeue**. **Enqueue** adalah operasi menambah/menyisipkan item pada akhir queue. Pada proses enqueue item ditambahkan di belakang item terakhir dalam queue. **Dequeue** adalah proses mengambil item dari queue. Ketika proses dequeue item dalam queue akan berkurang yaitu item paling depan dalam queue atau item yang pertama kali ditambahkan kedalam queue. Queue bisa dibuat dengan Array atau Linked List. Tabel dibawah memberikan ilustrasi operasi pada queue yang dibuat dengan array.

sebelum	operasi	sesudah
[]	queue("A")	["A"]
["A"]	queue("B")	["A","B"]
["A","B"]	queue("C")	["A","B","C"]
["A","B","C"]	dequeue()	["B","C"]
["A","B"]	queue("D")	["A","B","D"]
["A","B","D"]	dequeue()	["B","D"]

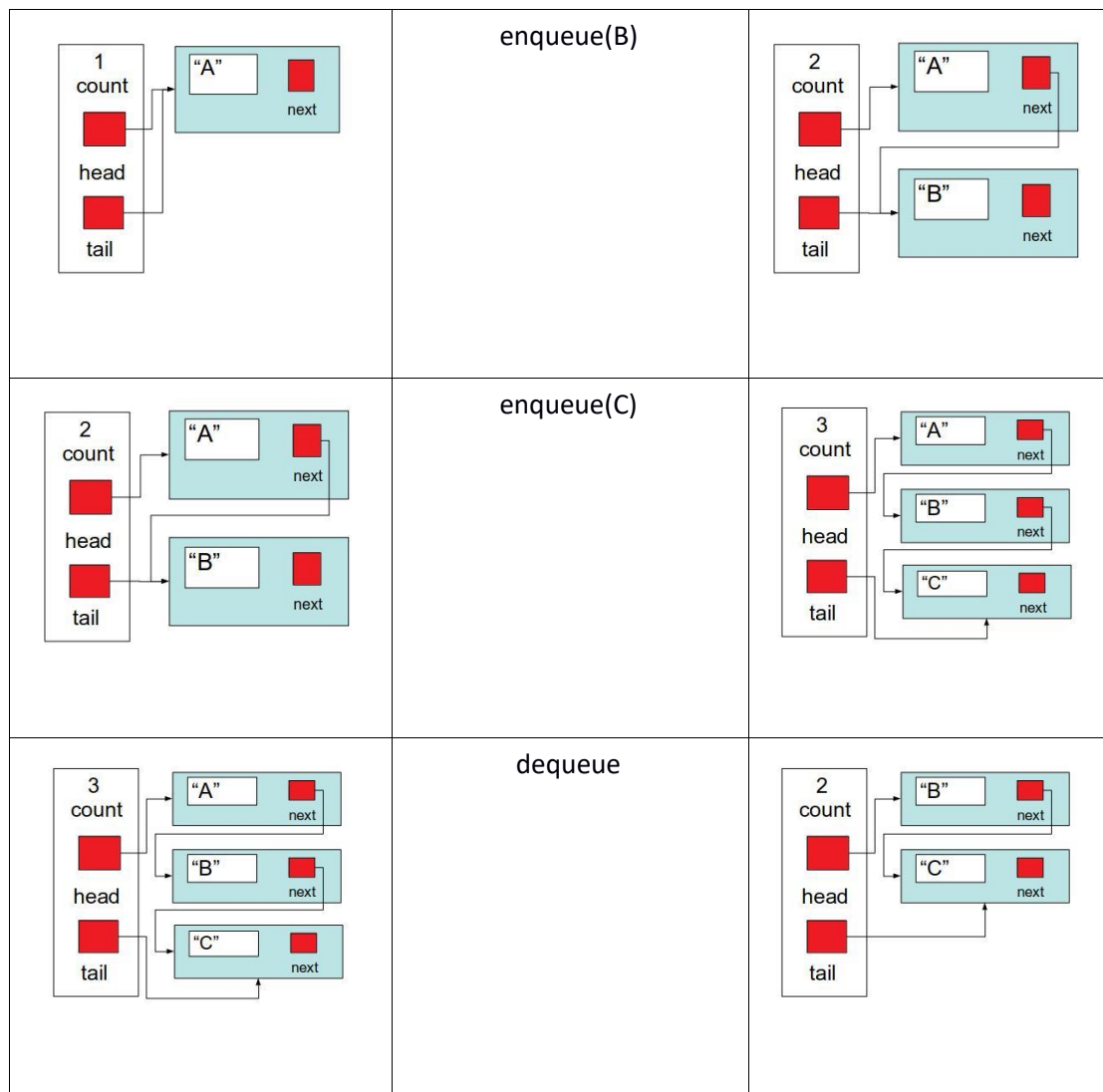
Dalam implementasi queue menggunakan array ini disimpan variabel **rear** yang menunjukkan index atau nomor urut terakhir item dalam queue. Operasi **enqueue** menambahkan item ke dalam array pada index/urutan **rear+1** atau **setelah item terakhir** dalam array. Operasi **dequeue** akan mengambil item dari dalam array pada index 0 atau **item pertama/paling depan** dalam array. Kemudian dilakukan pergeseran isi array ke kiri sebanyak satu dan mengurangi rear sebanyak satu(rear-1).

Pada tabel dibawah diilustrasikan operasi queue menggunakan array. Perbedaan dengan queue diatas adalah implementasi proses **enqueue** akan menggeser item dalam array kekanan sebanyak satu dan menambahkan/menyisipkan item pada **array index ke 0**. Proses **dequeue** akan mengambil item pada **array index terakhir atau rear**.

sebelum	operasi	sesudah
[]	queue("A")	["A"]
["A"]	queue("B")	["B","A"]
["B","A"]	queue("C")	["C","B","A"]
["C","B","A"]	dequeue()	["C","B"]
["C","B"]	queue("D")	["D","C","B"]
["D","C","B"]	dequeue()	["D","C"]

Gambar dibawah memberikan ilustrasi queue menggunakan single linked list. Head merupakan pointer yang menunjuk pada item paling atas yang merupakan item yang pertama dimasukkan dalam queue, item paling depan dalam antrian. Tail merupakan pointer yang selalu menunjuk pada item yang terakhir ditambahkan dalam queue, item paling belakang dalam antrian. Dalam ilustrasi ini operasi **enqueue** pada queue menggunakan operasi **add last** pada single linked list dan operasi **dequeue** menggunakan operasi **remove first**. Jika operasi **enqueue** menggunakan operasi **add first** maka operasi **dequeue** akan menggunakan **remove last**.

sebelum	operasi	sesudah
<div> <div>0</div> <div>count</div> <div></div> <div>head</div> <div></div> <div>tail</div> </div>	enqueue(A)	<div> <div>1</div> <div>count</div> <div></div> <div>head</div> <div></div> <div>tail</div> </div> <div> <div> <div>"A"</div> <div>next</div> </div> </div>



8.5 Prosedur Praktikum

1. Buatlah file percobaan queue_array.cpp berikut

```

1  #include <iostream>
2  #include <stdlib.h>
3
4  #define MAX_SIZE 5
5  using namespace std;
6  int choice, i;
7  char item;
8  char arr_stack[MAX_SIZE];
9  int count = 0;
10 int keluar = 0;
11 int front, rear=0;
12
13 void enqueue(char item)
14 {
15     if (rear == MAX_SIZE)

```

```

15     {
16         cout << "\n# Queue Penuh";
17     }
18     else
19     {
20         cout << "\n# Queue No urut/index : " << rear << ",
21 Queue : " << item;
22         .....
23     }
24 }
25 void dequeue()
26 {
27     if (rear == 0)
28         cout << "\n## Queue kosong";
29     else
30     {
31         .....
32         .....
33         .....
34         .....
35         .....
36         .....
37         .....
38     }
39 }
40
41 void printAll()
42 {
43     cout << "\n## Queue Size : " << rear;
44     for (i = 0; i < rear; i++)
45         cout << "\n## No Urut/index : " << i << ", Value : " <<
46 arr_stack[i];
47 }
48 void menu()
49 {
50     cout << "\nMasukkan operasi yang akan dilakukan
51 (1:enqueue, 2:dequeue, 3:print) : ";
52     cin >> choice;
53     switch (choice)
54     {
55     case 1:
56     {
57         cout << "\nMasukkan huruf yang akan di-enqueue : ";
58         cin >> item;
59         enqueue(item);

```

60	<code>break;</code>
61	<code>case 2:</code>
62	<code> dequeue();</code>
63	<code> break;</code>
64	<code>case 3:</code>
65	<code> printAll();</code>
66	<code> break;</code>
67	<code>default:</code>
68	<code> cout << "\n1:enqueue, 2:dequeue, 3:print\n";</code>
69	<code> keluar = 1;</code>
70	<code> break;</code>
71	<code> }</code>
72	<code>}</code>
73	
74	<code>int main()</code>
75	<code>{</code>
76	<code> do</code>
77	<code> {</code>
78	<code> menu();</code>
79	<code> } while (keluar == 0);</code>
80	<code>}</code>

2. Tambahkan kode berikut pada baris 22

22	<code>arr_stack[rear++] = item;</code>
----	--

3. Tambahkan kode berikut pada baris 32 sampai 38

32	<code> cout << "\n##Dequeue Value :" << arr_stack[0];</code>
33	<code> for(i=1;i<=rear;i++)</code>
34	<code> {</code>
35	<code> char temp=arr_stack[i];</code>
36	<code> arr_stack[i-1]=temp;</code>
37	<code> }</code>
38	<code> rear--;</code>

4. Jalankan program kemudian

- pilih menu enqueue dan masukkan “A”
- pilih menu enqueue dan masukkan “B”
- pilih menu enqueue dan masukkan “C”
- pilih menu print
- pilih menu dequeue
- pilih menu print
- pilih menu enqueue dan masukkan “D”
- pilih menu print

5. Buatlah program queue_llist.cpp berikut :

1	<code>#include <iostream></code>
---	--

```

2  #include <stdlib.h>
3  using namespace std;
4  struct node
5  {
6      char data;
7      struct node *next;
8  };
9  typedef struct node node;
10
11 node *head,*tail;
12 int choice;
13 char item;
14 int count = 0;
15 int keluar = 0;
16
17 void enqueue(char item)
18 {
19     node *temp = new node;
20     temp->data = item;
21     temp->next = NULL;
22     if (head == NULL)
23     {
24         head = temp;
25         tail=temp;
26         temp = NULL;
27     }
28     else
29     {
30         .....
31         .....
32     }
33     cout << "\n# Queue : No urut/index : " << count << ",
Value :" << item;
34     count++;
35 }
36
37 void dequeue()
38 {
39     if (head==NULL)
40         cout << "\n## Queue kosong";
41     else
42     {
43         .....
44         .....
45         .....
46         .....
47         cout << "\n##Dequeue result:" << item;
48         cout << "\n##jumlah item dalam queue : " << count ;

```

```

49     }
50 }
51
52 void printAll()
53 {
54     cout << "\n## Queue Size : " << count;
55     node *temp=new node;
56     temp=head;
57     int i=0;
58     while(temp!=NULL)
59     {
60         cout << "\n## No Urut/index : " << i << ", Value : " <<
temp->data;
61         temp=temp->next;
62         i++;
63     }
64 }
65
66 void menu()
67 {
68     cout << "\nMasukkan operasi yang akan dilakukan
69 (1:enqueue, 2:dequeue, 3:print) : ";
70     cin >> choice;
71     switch (choice)
72     {
73     case 1:
74     {
75         cout << "\nMasukkan huruf yang akan dimasukkan dalam
queue : ";
76         cin >> item;
77         enqueue(item);
78         break;
79     }
80     case 2:
81         dequeue();
82         break;
83     case 3:
84         printAll();
85         break;
86     default:
87         cout << "\n1:enqueue, 2:dequeue, 3:print\n";
88         keluar = 1;
89         break;
90     }
91 }
92
93 int main()
94 {

```

95	<code>do</code>
96	<code>{</code>
97	<code> menu();</code>
98	<code> }<code>while</code> (keluar == 0);</code>
99	<code>}</code>

6. Tambahkan kode berikut pada baris 30 dan 31

30	<code>tail->next = temp;</code>
31	<code>tail = temp;</code>

7. Tambahkan kode berikut pada baris 43 sampai 46

43	<code>--count;</code>
44	<code>char item=head->data;</code>
45	<code>node *temp=head->next;</code>
46	<code>head=temp;</code>

8. Jalankan program kemudian :

- pilih menu enqueue dan masukkan “A”
- pilih menu enqueue dan masukkan “B”
- pilih menu enqueue dan masukkan “C”
- pilih menu print
- pilih menu dequeue
- pilih menu print
- pilih menu enqueue dan masukkan “D”
- pilih menu print

8.6 Hasil Percobaan

1. Tuliskan hasil dari percobaan pertama diatas.
2. Tuliskan hasil dari percobaan kedua diatas.

8.7 Analisis Hasil

1. Tuliskan Analisis hasil dari percobaan pertama diatas.
2. Tuliskan Analisis hasil dari percobaan kedua diatas.

8.8 Kesimpulan

1. Tuliskan kesimpulan dari percobaan pertama diatas.
2. Tuliskan kesimpulan dari percobaan kedua diatas.

8.9 Latihan

Buatlah program queue menggunakan array dengan ketentuan operasi enqueue akan selalu menambahkan item kedalam array pada index ke 0.

8.10 Tugas

1. Buat pseudocode untuk operasi enqueue dan dequeue pada queue menggunakan single linked list dengan ketentuan operasi enqueue menggunakan add last pada linked list dan operasi dequeue akan menggunakan remove first.
2. Buatlah program queue menggunakan double linked list dengan ketentuan operasi enqueue akan menambahkan item pada node head(add first) dan operasi dequeue akan mengambil item dari node pada tail(remove last).

DAFTAR PUSTAKA

- Goodrich, M.T., Tamassia, R., dan Mount, D., 2003, "Data Structures and Algorithms in C++ 2nd edition", John Wiley & Sons.
- Sahni, S., 2005, "Data Structures, Algorithms and Applications in C++ 2nd edition", Universities Press.
- Sanjaya, D., 2005, "Asyiknya Belajar Struktur Data di Planet C++", PT Elex Media Komputindo.

Modul 8 : Sorting

Waktu Pelaksanaan Praktikum

Durasi kegiatan praktikum = **170 menit**, dengan rincian sebagai berikut:

17. 15 menit untuk pengerjaan Tes Awal atau wawancara Tugas Pendahuluan
18. 60 menit untuk penyampaian materi
19. 45 menit untuk pengerjaan tugas / Studi Kasus
20. 50 menit **Pengayaan**

Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat:

10. Memahami konsep dasar sorting
11. Mengimplementasikan Bubble dan Shell Sort
12. Mengimplementasikan Selection dan Insertion Sort
13. Memahami perbedaan masing-masing metode sorting

Alat & Bahan

11. Komputer
12. Dev C++
13. GCC

Dasar Teori

Sorting

Pengurutan data (sorting) didefinisikan sebagai suatu proses untuk menyusun kembali himpunan obyek menggunakan aturan tertentu. Ada dua macam urutan yang biasa digunakan dalam proses pengurutan yaitu :

- a) urut naik (ascending) yaitu dari data yang mempunyai nilai paling kecil sampai paling besar
- b) urut turun (descending) yaitu data yang mempunyai nilai paling besar sampai paling kecil.

Contoh : data bilangan 5, 2, 6 dan 4 dapat diurutkan naik menjadi 2, 4, 5, 6 atau diurutkan turun menjadi 6, 5, 4, 2. Pada data yang bertipe char, nilai data dikatakan lebih kecil atau lebih besar dari yang lain didasarkan pada urutan relatif (collating sequence) seperti dinyatakan dalam tabel ASCII

Keuntungan dari data yang sudah dalam keadaan terurutkan antara lain :

- a) data mudah dicari (misalnya dalam buku telepon atau kamus bahasa), mudah untuk dibetulkan, dihapus, disisipi atau digabungkan.
- b) Selain itu dalam keadaan terurutkan, kita mudah melakukan pengecekan apakah ada data yang hilang atau tidak.
- c) melakukan kompilasi program komputer jika tabel-tabel simbol harus dibentuk
- d) mempercepat proses pencarian data yang harus dilakukan berulang kali.

Data yang diurutkan sangat bervariasi, dalam hal jumlah data maupun jenis data yang akan diurutkan. Tidak ada algoritma terbaik untuk setiap situasi yang kita hadapi, bahkan cukup sulit untuk menentukan algoritma mana yang paling baik untuk situasi tertentu karena ada beberapa faktor yang mempengaruhi efektifitas algoritma pengurutan. Beberapa faktor yang berpengaruh pada efektifitas suatu algoritma pengurutan antara lain:

- a) banyak data yang diurutkan
- b) kapasitas pengingat apakah mampu menyimpan semua data yang kita miliki
- c) tempat penyimpanan data, misalnya disket, flashdisk, harddisk atau media penyimpanan yang lain

Bubble Sort

Bubble sort (metode gelembung) adalah metode/algorithm pengurutan dengan dengan cara melakukan penukaran data dengan tepat disebelahnya secara terus menerus sampai bisa dipastikan dalam satu iterasi tertentu tidak ada lagi perubahan. Jika tidak ada perubahan berarti data sudah terurut. Disebut pengurutan gelembung karena masing-masing kunci akan dengan lambat menggelembung ke posisinya yang tepat.

Metode gelembung (bubble sort) sering juga disebut dengan metode penukaran (exchange sort) adalah metode yang mengurutkan data dengan cara membandingkan masing-masing elemen, kemudian melakukan penukaran bila perlu. Metode ini mudah dipahami dan diprogram, tetapi bila dibandingkan dengan metode lain yang kita pelajari, metode ini merupakan metode yang paling tidak efisien.

Proses pengurutan metode bubble sort ini menggunakan dua loop. Loop pertama melakukan pengulangan dari elemen ke 2 sampai dengan elemen ke N-1 (misalnya variable i), sedangkan kalang kedua melakukan pengulangan menurun dari elemen ke N sampai elemen ke i (misalnya variable j). Pada setiap pengulangan, elemen ke j-1 dibandingkan dengan elemen ke j. Apabila data ke j-1 lebih besar daripada data ke j, dilakukan penukaran.

Contoh: Elemen array / larik dengan N=6 buah elemen dibawah ini.

25	27	10	8	76	21
1	2	3	4	5	6

Langkah 1:

K=N=6					21	76
K=5				8	21	76
K=4			8	10	21	76
K=3		8	27	10	21	76
K=2	8	25	27	10	21	76

Hasil akhir langkah 1:

8	25	27	10	21	76
1	2	3	4	5	6

Langkah 2:

K=N=6					21	76
K=5				10	21	76
K=4			10	27	21	76
K=3		10	25	27	21	76

Hasil akhir langkah 2 :

8	10	25	27	21	76
1	2	3	4	5	6

Langkah 3:

K=N=6					21	76
K=5				21	27	76
K=4			21	25	27	76

Hasil akhir langkah 3 :

Proses ke-	Jarak	0	1	2	3	4	5
1	Awal	25	40	5	8	33	13
2	Jarak = 3	<u>25</u>	40	5	<u>8</u>	33	13
3		8	<u>40</u>	5	25	<u>33</u>	13
4		8	33	<u>5</u>	25	40	<u>13</u>
5	Jarak = 1	<u>8</u>	<u>33</u>	5	25	40	13
6		8	<u>33</u>	<u>5</u>	25	40	13
7		8	5	<u>33</u>	<u>25</u>	40	13
9		8	5	25	<u>33</u>	<u>40</u>	13
10		8	5	25	33	<u>40</u>	<u>13</u>
11		<u>8</u>	<u>5</u>	25	33	13	40
12		5	<u>8</u>	<u>25</u>	33	13	40
13		5	8	25	33	13	40

Proses ke-	Jarak	0	1	2	3	4	5
14		5	8	25	<u>33</u>	<u>13</u>	40
15		5	8	25	13	<u>33</u>	<u>40</u>
16		<u>5</u>	<u>8</u>	25	13	33	40
17		5	<u>8</u>	<u>25</u>	13	33	40
18		5	8	<u>25</u>	<u>13</u>	33	40
19		5	8	13	<u>25</u>	<u>33</u>	40
20		<u>5</u>	<u>8</u>	<u>13</u>	<u>25</u>	<u>33</u>	<u>40</u>

Keterangan

Proses ke-1: Data dalam bentuk array awal yang dimiliki, data berjumlah 6. Kemudian kita tentukan dengan jarak 3.

Proses ke-2: Dengan jarak 3, kita bandingkan data pada indeks ke 0 dengan indeks ke 3. Data pada indeks ke-0 bernilai 25 dibandingkan dengan indeks ke-3 bernilai 8. Karena data pada indeks ke-3 bernilai lebih kecil, maka dilakukan pertukaran data. Sehingga data yang bernilai 8 ditaruh pada indeks ke-0 dan sebaliknya. Setelah dilakukan pertukaran, akan maju 1 indeks hingga selesai.

Proses ke-3: Dengan cara yang sama dilakukan seperti proses ke-2, antara indeks ke-1 dan indeks ke-4, indeks ke-1 bernilai 40 dan indeks ke-4 bernilai 33 kemudian dilakukan pertukaran.

Proses ke-4: Indeks ke-2 dibandingkan dengan indeks ke-5, karena data sudah sesuai, sehingga tidak dilakukan pertukaran.

Proses ke-5 sampai dengan proses ke-19 : proses perbandingan data dan pertukaran data mirip dengan proses ke-2 hingga proses ke-4, namun perbedaannya adalah nilai jarak 1. Nilai jarak biasanya akan dilakukan setting decrement 1 dari indeks awal, atau dapat ditentukan untuk proses selanjutnya. Pada contoh tersebut dilakukan pemberian jarak sebesar 1 untuk proses ini. Proses ini akan terus berjalan dan panjang proses tidak menentu hingga tidak ada lagi data yang akan ditukarkan dan dibandingkan hingga tidak ada data yang dapat dibandingkan (lebih kecil atau lebih besar)

Proses ke-20 akan dilakukan lagi perbandingan terakhir pada masing-masing data untuk memastikan tidak ada data yang lebih kecil dari indeks sebelumnya.

Prosedur Praktikum

Jalankan Program berikut

Bubble Sort

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #define MAX 10
4
5  int Data[MAX];
6
7  // Prosedur menukar data
8  void Tukar (int *a, int *b)
9  {
10     int temp;
11     temp = *a;
12     *a = *b;
13     *b = temp;
14 }
15
16 // Prosedur pengurutan metode gelembung
17 void BubbleSort()
18 {

```

```

19     int i, j;
20     for(i=1; i<MAX-1; i++)
21         for(j=MAX-1; j>=i; j--)
22             if(Data[j-1] > Data[j])
23                 Tukar(&Data[j-1], &Data[j]);
24 }
25
26 void main()
27 {
28     int i;
29     srand(0);
30     // Membangkitkan bilangan acak
31     printf("DATA SEBELUM TERURUT");
32     for(i=0; i<MAX; i++)
33     {
34         Data[i] = (int) rand()/1000+1;
35         printf("\nData ke %d : %d ", i, Data[i]);
36     }
37     BubbleSort();
38     // Data setelah terurut
39     printf("\nDATA SETELAH TERURUT");
40     for(i=0; i<MAX; i++)
41     {
42         printf("\nData ke %d : %d ", i, Data[i]);
43     }
44 }

```

Shell Sort

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #define MAX 10
4
5  int Data[MAX];
6
7  // Prosedur menukar data
8  void Tukar (int *a, int *b)
9  {
10     int temp;
11     temp = *a;
12     *a = *b;
13     *b = temp;
14 }
15
16 // Prosedur pengurutan metode Shell
17 void ShellSort()
18 {
19     int Jarak, i, j;
20     bool Sudah;
21     Jarak = MAX;
22     while(Jarak > 1)
23     {
24         Jarak = Jarak / 2;
25         Sudah = false;
26         while(!Sudah)
27         {
28             Sudah = true;

```

29	for(j=0; j<MAX-Jarak; j++)
30	{
31	i = j + Jarak;
32	if(Data[j] > Data[i])
33	{
34	Tukar(&Data[j], &Data[i]);
35	Sudah = false;
36	}
37	}
38	}
39	}
40	}
41	
42	void main()
43	{
44	int i;
45	srand(0);
46	
47	// Membangkitkan bilangan acak
48	printf("DATA SEBELUM TERURUT");
49	for(i=0; i<MAX; i++)
50	{
51	Data[i] = (int) rand()/1000+1;
52	printf("\nData ke %d : %d ", i, Data[i]);
53	}
54	ShellSort();
55	// Data setelah terurut
56	printf("\nDATA SETELAH TERURUT");
57	for(i=0; i<MAX; i++)
58	{
59	printf("\nData ke %d : %d ", i, Data[i]);
60	}
61	}

Hasil Percobaan

1. Apakah yang anda dapatkan dari Bubble Sort tersebut?

.....

.....

2. Apakah yang anda dapatkan dari Shell Sort tersebut?

.....

.....

Analisis Hasil

1. Apakah perbedaan mendasar pada bubble sort dan shell sort tersebut?

.....

.....

2. Jika diberikan data secara berbeda (misal 10, 20, 30 dst) proses sorting manakah yang paling efektif? Sebutkan alasan anda dengan tepat!

.....
.....

Kesimpulan

Berilah kesimpulan tentang bubble sort dan shell sort setelah anda menjalankan program tersebut

.....
.....

Latihan

1. Buatlah Kedua Sorting tersebut dalam Bentuk Descending

.....
.....

2. Tambahkan kode program untuk menampilkan perubahan setiap iterasi dari proses pengurutan dengan bubble dan shell sort.

.....
.....

3. Tambahkan kode program untuk menghitung banyaknya perbandingan dan pergeseran pada algoritma bubble dan shell.

.....
.....

Tugas

1. Pada bubble dan shell sort, berilah data secara acak sebanyak 50 data. Masukkan data tersebut, dan berapa banyak iterasi yang dibutuhkan untuk sorting data tersebut?

.....
.....

2. Pilih salah satu antara bubble atau shell sort, implementasikan pengurutan data Pegawai pada tugas pendahuluan dengan ketentuan sebaga berikut
 - a. Proses pengurutan dapat dipilih secara ascending maupun descending
 - b. Pengurutan dapat dilakukan berdasarkan NIP dan Nama
 - c. Gunakan struktur data array

.....

DAFTAR PUSTAKA

- Goodrich, M.T., Tamassia, R., dan Mount, D., 2003, "Data Structures and Algorithms in C++ 2nd edition", John Wiley & Sons.
- Sahni, S., 2005, "Data Structures, Algorithms and Applications in C++ 2nd edition", Universities Press.
- Sanjaya, D., 2005, "Asyiknya Belajar Struktur Data di Planet C++", PT Elex Media Komputindo.

Modul 9 : Sorting (Insertion dan Selection Sort)

Waktu Pelaksanaan Praktikum

Durasi kegiatan praktikum = **170 menit**, dengan rincian sebagai berikut:

21. 15 menit untuk pengerjaan Tes Awal atau wawancara Tugas Pendahuluan
22. 60 menit untuk penyampaian materi
23. 45 menit untuk pengerjaan tugas / Studi Kasus
24. 50 menit **Pengayaan**

Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat:

14. Memahami konsep dasar sorting
15. Mengimplementasikan Bubble dan Shell Sort
16. Mengimplementasikan Selection dan Insertion Sort
17. Memahami perbedaan masing-masing metode sorting

Alat & Bahan

14. Komputer
15. Dev C++
16. GCC

Dasar Teori

Selection Sort

Metode seleksi melakukan pengurutan dengan cara mencari data yang terkecil kemudian menukarkannya dengan data yang digunakan sebagai acuan atau sering dinamakan pivot.

Proses pengurutan dengan metode selection sort secara ascending dapat dijelaskan sebagai berikut:

Contoh: Elemen array / larik dengan $N=6$ buah elemen dibawah ini.

29	27	10	8	76	21
1	2	3	4	5	6

Langkah 1:

Cari elemen maksimum di dalam larik $L[1..6] \rightarrow \text{maks} = L[5] = 76$

Tukar maks dengan $L[N]$, hasil akhir langkah 1:

29	27	10	8	21	76
1	2	3	4	5	6

Langkah 2:

(berdasarkan susunan larik hasil langkah 1)

Cari elemen maksimum di dalam larik $L[1..5] \rightarrow \text{maks} = L[1] = 29$

Tukar maks dengan $L[5]$, hasil akhir langkah 2:

21	27	10	8	29	76
1	2	3	4	5	6

Langkah 3:

(berdasarkan susunan larik hasil langkah 2)

Cari elemen maksimum di dalam larik $L[1..4] \rightarrow \text{maks} = L[2] = 27$

Tukar maks dengan L[4], hasil akhir langkah 3:

21	8	10	27	29	76
1	2	3	4	5	6

Langkah 4:

(berdasarkan susunan larik hasil langkah 3)

Cari elemen maksimum di dalam larik L[1..3] → maks = L[1] = 21

Tukar maks dengan L[3], hasil akhir langkah 4:

10	8	21	27	29	76
1	2	3	4	5	6

Langkah 5:

(berdasarkan susunan larik hasil langkah 4)

Cari elemen maksimum di dalam larik L[1..2] → maks = L[1] = 10

Tukar maks dengan L[2], hasil akhir langkah 5:

8	10	21	27	29	76
1	2	3	4	5	6

Jika nilai pada array/larik sudah terurutkan maka proses sorting sudah selesai.

Insertion Sort

Proses pengurutan dengan metode penyisipan langsung (straight) dapat dijelaskan sebagai berikut :

Data dicek satu per satu mulai dari yang kedua sampai dengan yang terakhir. Apabila ditemukan data yang lebih kecil daripada data sebelumnya, maka data tersebut disisipkan pada posisi yang sesuai. Akan lebih mudah apabila membayangkan pengurutan kartu. Pertama-tama anda meletakkan kartu-kartu tersebut di atas meja, kemudian melihatnya dari kiri ke kanan. Apabila kartu di sebelah kanan lebih kecil daripada kartu di sebelah kiri, maka ambil kartu tersebut dan sisipkan di tempat yang sesuai.

Contoh: Elemen array / larik dengan N=6 buah elemen dibawahini.

29	27	10	8	76	21
1	2	3	4	5	6

Langkah 1:

Elemen L[1] dianggap sudah terurut

29	27	10	8	76	21
1	2	3	4	5	6

Langkah 2:

(berdasarkan susunan larik pada langkah 1)

Cari posisi yang tepat untuk L[2] pada L[1..2], diperoleh :

27	29	10	8	76	21
1	2	3	4	5	6

Langkah 3:

(berdasarkan susunan larik pada langkah 2)

Cari posisi yang tepat untuk L[3] pada L[1..3], diperoleh :

10	27	29	8	76	21
1	2	3	4	5	6

Langkah 4:

(berdasarkan susunan larik pada langkah 3)

Cari posisi yang tepat untuk L[4] pada L[1..4],diperoleh :

8	10	27	29	76	21
1	2	3	4	5	6

Langkah 5:

(berdasarkan susunan larik pada langkah 4)

Cari posisi yang tepat untuk L[5] pada L[1..5],diperoleh :

8	10	27	29	76	21
1	2	3	4	5	6

Langkah 6:

(berdasarkan susunan larik pada langkah 5)

Cari posisi yang tepat untuk L[6] pada L[1..6],diperoleh :

8	10	21	27	29	76
1	2	3	4	5	6

Jika nilai pada array/larik sudah terurutkan maka proses sorting sudah selesai.

Sedikit berbeda dengan insertion, untuk binary insertion sort memperbaiki metode pengurutan dengan algoritma penyisipan langsung dengan melakukan proses perbandingan yang lebih sedikit sehingga proses pengurutan lebih cepat.

Metode penyisipan biner melakukan proses perbandingan dengan membagi dua bagian data dari posisi 0 sampai dengan $i-1$ yang disebut dengan bagian kiri dan kanan. Apabila data pada posisi ke i berada pada jangkauan kiri maka proses perbandingan dilakukan hanya pada bagian kiri dan menggeser posisi sampai i .

Prosedur Praktikum

Jalankan program berikut

Selection Sort

SelectionSort.cpp	
1	#include <stdio.h>
2	#include <stdlib.h>
3	#define MAX 10
4	
5	int Data[MAX];
6	// Fungsi pertukaran bilangan
7	void Tukar (int *a, int *b)
8	{
9	int temp;
10	temp = *a;
11	*a = *b;
12	*b = temp;
13	}
14	
15	// Fungsi pengurutan penyisipan biner
16	void SelectionSort()

17	{
18	int i, j, k;
19	for(i=0; i<MAX-1;i++){
20	k = i;
21	for (j=i+1; j<MAX; j++){
22	if(Data[k] > Data[j])
23	k = j;
24	Tukar(&Data[i], &Data[k]);
25	}
26	}
27	
28	void main()
29	{
30	int i;
31	srand(0);
32	
33	// Membangkitkan bilangan acak
34	printf("DATA SEBELUM TERURUT");
35	for(i=0; i<MAX; i++){
	{
	Data[i] = (int) rand()/1000+1;
	printf("\nData ke %d : %d ", i, Data[i]);
	}
	SelectionSort();
	// Data setelah terurut
	printf("\nDATA SETELAH TERURUT");
	for(i=0; i<MAX; i++){
	{
	printf("\nData ke %d : %d ", i, Data[i]);
	}
	}

Insertion Sort (Straight)

StraightInstertionSort.cpp	
1	#include <stdio.h>
2	#include <stdlib.h>
3	#define MAX 10
4	
5	int Data[MAX];
6	// Fungsi pengurutan penyisipan langsung
7	void StraightInsertSort()
8	{
9	int i, j, x;
10	for(i=1; i<MAX; i++){
11	x = Data[i];
12	j = i - 1;
13	while (x < Data[j]){
14	Data[j+1] = Data[j];
15	j--;
16	}
17	Data[j+1] = x;
18	}
19	}
20	
21	void main()

```

22 {
23     int i;
24     srand(0);
25     // Membangkitkan bilangan acak
26     printf("DATA SEBELUM TERURUT");
27     for(i=0; i<MAX; i++)
28     {
29         Data[i] = (int) rand()/1000+1;
30         printf("\nData ke %d : %d ", i, Data[i]);
31     }
32
33     StraightInsertSort();
34     // Data setelah terurut
35     printf("\nDATA SETELAH TERURUT");
36     for(i=0; i<MAX; i++)
37     {
38         printf("\nData ke %d : %d ", i, Data[i]);
39     }
40 }

```

Insertion Sort (Binary)

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #define MAX 10
4
5  int Data[MAX];
6  // Fungsi pengurutan penyisipan biner
7  void BinaryInsertSort()
8  {
9      int i, j, l, r, m, x;
10     for (i=1; i<MAX; i++){
11         x = Data[i];
12         l = 0;
13         r = i - 1;
14         while(l <= r){
15             m = (l + r) / 2;
16             if(x < Data[m])
17                 r = m - 1;
18             else
19                 l = m + 1;
20         }
21         for(j=i-1; j>=l; j--)
22             Data[j+1] = Data[j];
23         Data[l]=x;
24     }
25 }
26 void main()
27 {
28     int i;
29     srand(0);
30
31     // Membangkitkan bilangan acak
32     printf("DATA SEBELUM TERURUT");
33     for(i=0; i<MAX; i++)
34     {

```

35	Data[i] = (int) rand()/1000+1;
36	printf("\nData ke %d : %d ", i, Data[i]);
37	}
38	
39	BinaryInsertSort();
40	// Data setelah terurut
41	printf("\nDATA SETELAH TERURUT");
42	for(i=0; i<MAX; i++)
43	{
44	printf("\nData ke %d : %d ", i, Data[i]);
45	}
46	}

Hasil Percobaan

1. Apakah yang anda dapatkan dari Selection Sort tersebut?

.....

.....

2. Apakah yang anda dapatkan dari Insertion Sort (Straight dan binary) tersebut?

.....

.....

Analisis Hasil

3. Apakah perbedaan mendasar pada Selection sort dan Insertion sort tersebut?

.....

.....

4. Jika diberikan data secara berbeda (misal 10, 20, 30 dst) proses sorting manakah yang paling efektif? Sebutkan alasan anda dengan tepat!

.....

.....

Kesimpulan

Berilah kesimpulan tentang bubble sort dan shell sort setelah anda menjalankan program tersebut

.....

.....

Latihan

4. Buatlah Kedua Sorting tersebut dalam Bentuk Descending

.....

.....

5. Tambahkan kode program untuk menampilkan perubahan setiap iterasi dari proses pengurutan dengan Selection dan Insertion.

-
-
6. Tambahkan kode program untuk menghitung banyaknya perbandingan dan pergeseran pada algoritma Selection dan Insertion.

.....

.....

Tugas

3. Pada Selection dan Insertion, berilah data secara acak sebanyak 50 data. Masukkan data tersebut, dan berapa banyak iterasi yang dibutuhkan untuk sorting data tersebut?

-
-
4. Pilih salah satu antara Selection dan Insertion, implementasikan pengurutan data Pegawai pada tugas pendahuluan dengan ketentuan sebaga berikut
- a. Proses pengurutan dapat dipilih secara ascending maupun descending
 - b. Pengurutan dapat dilakukan berdasarkan NIP dan Nama
 - c. Gunakan struktur data array

.....

DAFTAR PUSTAKA

- Goodrich, M.T., Tamassia, R., dan Mount, D., 2003, "Data Structures and Algorithms in C++ 2nd edition", John Wiley & Sons.
- Sahni, S., 2005, "Data Structures, Algorithms and Applications in C++ 2nd edition", Universities Press.
- Sanjaya, D., 2005, "Asyiknya Belajar Struktur Data di Planet C++", PT Elex Media Komputindo.

Modul 10 : UDT Binary Tree

Waktu Pelaksanaan Praktikum

Durasi kegiatan praktikum = **170 menit**, dengan rincian sebagai berikut:

25. 15 menit untuk pengerjaan Tes Awal atau wawancara Tugas Pendahuluan
26. 60 menit untuk penyampaian materi
27. 45 menit untuk pengerjaan tugas / Studi Kasus
28. 50 menit **Pengayaan**

Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat:

18. Mengetahui dan memahami konsep tree, binary tree, dan tree transversal
19. Menerapkan konsep tree, binary tree, dan tree transversal dengan bahasa pemrograman

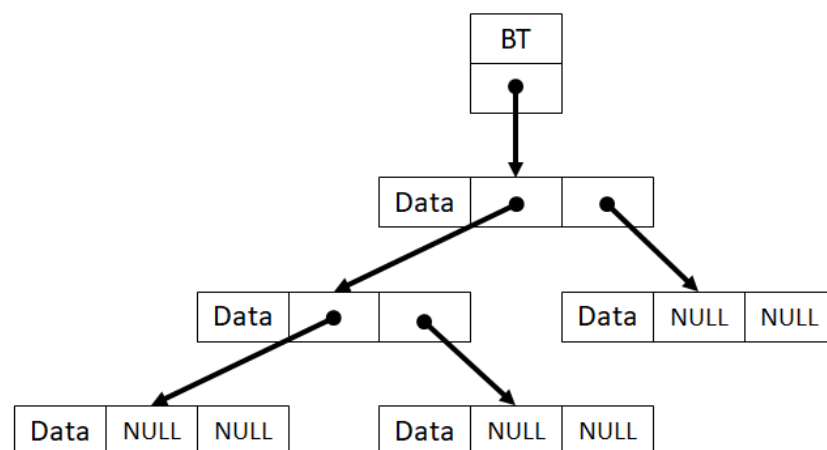
Alat & Bahan

17. Komputer
18. Dev C++
19. GCC

Dasar Teori

Binary Tree adalah non-linear linked list yang mana masing-masing nodenya menunjuk paling banyak 2 node yang lain. Elemen pada tree disebut node, yang mana sebuah node hanya bisa mempunyai satu parent dan bisa mempunyai child paling banyak 2. Root adalah node yang memiliki hirarki tertinggi dan dibentuk pertama kali sehingga tidak memiliki parent. Ada 4 macam cara penelusuran node pada binary tree, yaitu preorder, postorder, inorder, dan level order.

Gambaran pohon biner ini seperti gambar berikut.



UDT Binary Search Tree

Binary Search Tree merupakan bagian dari Binary Tree namun dengan kondisi subtree kiri lebih kecil dari subtree kanan. Dari ilustrasi gambar di atas UDT Binary Search Tree dapat direpresentasikan sebagai berikut.

Node	Tree
int data;	Node root;
Node nodeKiri;	Tree()
Node nodeKanan;	void sisipDtNode(int dtSisip)
	void preorderTraversal()
Node(int)	void inorderTraversal()
	void postorderTraversal()

Prosedur Praktikum

- Buatlah program sesuai dengan source code Program Latihan Praktikum 10.1
- Jalankan program

	Program Latihan Praktikum 10.1
1	<code>#include <iostream></code>
2	<code>#include <stdlib.h> // srand, rand</code>
3	<code>#include <time.h> // time</code>
4	
5	<code>using namespace std;</code>
6	
7	<code>struct Node {</code>
8	<code> int data;</code>
9	<code> Node *anakKiri;</code>
10	<code> Node *anakKanan;</code>
11	
12	<code> Node(int) ;</code>
13	<code>};</code>
14	
15	<code>struct Tree {</code>
16	<code> Node *root;</code>
17	
18	<code> Tree() ;</code>
19	<code> void sisipDtNode(int) ;</code>
20	<code> void preorderTraversal() ;</code>
21	<code> void inorderTraversal() ;</code>
22	<code> void postorderTraversal() ;</code>
23	
24	<code>private:</code>
25	<code> void sisipDt(Node*, int) ;</code>
26	<code> void preorder(Node*) ;</code>
27	<code> void inorder(Node*) ;</code>
28	<code> void postorder(Node*) ;</code>
29	<code>};</code>
30	
31	<code>Node::Node(int dt) {</code>
32	<code> data = dt;</code>
33	<code> anakKiri = anakKanan = NULL;</code>

```

34     }
35
36     Tree::Tree() {
37         root = NULL;
38     }
39
40     void Tree::sisipDtNode(int dtSisip) {
41         if (root == NULL)
42             root = new Node(dtSisip);
43         else
44             sisipDt(root, dtSisip);
45     }
46
47     void Tree::preorderTraversal() {
48         preorder(root);
49     }
50
51     void Tree::inorderTraversal() {
52         inorder(root);
53     }
54
55     void Tree::postorderTraversal() {
56         postorder(root);
57     }
58
59     void Tree::sisipDt(Node *node, int dtSisip) {
60         if (dtSisip < node->data) {
61             if (node->anakKiri == NULL)
62                 node->anakKiri = new Node(dtSisip);
63             else
64                 sisipDt(node->anakKiri, dtSisip);
65         }
66         else if (dtSisip > node->data) {
67             if (node->anakKanan == NULL)
68                 node->anakKanan = new Node(dtSisip);
69             else
70                 sisipDt(node->anakKanan, dtSisip);
71         }
72     }
73
74     void Tree::preorder(Node* node) {
75         if (node == NULL) return;
76
77         cout << node->data << ", ";
78         preorder(node->anakKiri);
79         preorder(node->anakKanan);
80     }
81
82     void Tree::inorder(Node* node) {
83         if (node == NULL) return;
84

```

```

85     inorder(node->anakKiri);
86     cout << node->data << ", ";
87     inorder(node->anakKanan);
88 }
89
90 void Tree::postorder(Node* node) {
91     if (node == NULL) return;
92
93     postorder(node->anakKiri);
94     postorder(node->anakKanan);
95     cout << node->data << ", ";
96 }
97
98 int main() {
99     Tree *tree = new Tree();
100     int nilai;
101
102     cout << "Sisip nilai data berikut: " << endl;
103
104     // sisip data 10 bilangan acak dari 0-99 ke dalam tree
105     srand(time(NULL));
106     for (int i = 0; i < 10; i++) {
107         nilai = rand() % 100;
108         cout << nilai << " ";
109         tree->sisipDtNode(nilai);
110     }
111
112     cout << "\n\nPreorder Traversal" << endl;
113     tree->preorderTraversal();
114
115     cout << "\n\nInorder Traversal" << endl;
116     tree->inorderTraversal();
117
118     cout << "\n\nPostorder Traversal" << endl;
119     tree->postorderTraversal();
120 }

```

Hasil Percobaan

1. Tulislah hasil dari Program Latihan Praktikum 10.1!

.....

.....

.....

2. Gambarkan tree yang telah dihasilkan oleh Program Latihan Praktikum 10.1!

.....

.....

.....

Analisis Hasil

1. Jelaskan apa perbedaan dari 3 macam penelusuran node yang ada di Program Latihan Praktikum 10.1, yaitu pre order, inorder, dan postorder !

.....

.....

.....

2. Apa tipe data yang dapat disimpan di node Program Latihan Praktikum 10.1? Jika ingin menyimpan data yang berupa String bagian kode program manakah yang harus diganti?

.....

.....

.....

3. Perhatikan urutan yang dibentuk oleh masing-masing traversal! Jelaskan apa yang dapat disimpulkan dari masing-masing traversal berdasarkan urutan yang dihasilkan!

.....

.....

.....

Kesimpulan

.....

.....

.....

Latihan

1. Tambahkan method untuk menghitung banyaknya node pada Binary Search Tree!
2. Tambahkan method untuk menghitung banyaknya daun!
3. Tambahkan method untuk menghitung tinggi dari pohon!

Tugas

1. Buatlah program Binary Search Tree dengan menggunakan array! Dengan asumsi jumlah maksimum dari indeks array adalah 40. Apabila suatu node menempati pada indeks lebih dari 40, maka node tersebut tidak akan dimasukkan. Jumlah node yang ada di dalam BST sebesar 10 node.

DAFTAR PUSTAKA

- Goodrich, M.T., Tamassia, R., dan Mount, D., 2003, "Data Structures and Algorithms in C++ 2nd edition", John Wiley & Sons.
- Sahni, S., 2005, "Data Structures, Algorithms and Applications in C++ 2nd edition", Universities Press.
- Sanjaya, D., 2005, "Asyiknya Belajar Struktur Data di Planet C++", PT Elex Media Komputindo.

Modul 11 : AVL TREE

Waktu Pelaksanaan Praktikum

Durasi kegiatan praktikum = **170 menit**, dengan rincian sebagai berikut:

29. 15 menit untuk pengerjaan Tes Awal atau wawancara Tugas Pendahuluan
30. 60 menit untuk penyampaian materi
31. 45 menit untuk pengerjaan tugas / Studi Kasus
32. 50 menit **Pengayaan**

Tujuan

Setelah mengikuti praktikum ini mahasiswa diharapkan dapat:

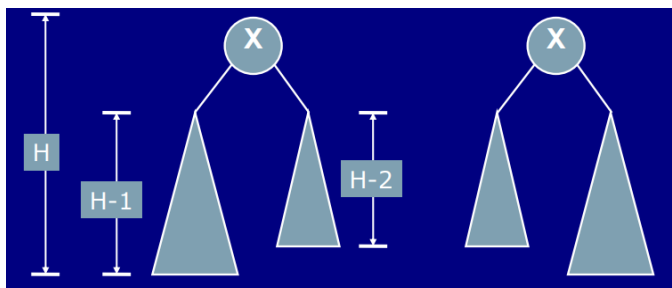
1. Mengetahui dan memahami konsep AVL Tree
2. Menerapkan konsep AVL Tree dengan menggunakan bahasa pemrograman
3. Menganalisis penerapan konsep AVL Tree

Alat & Bahan

20. Komputer
21. Dev C++ IDE

Dasar Teori

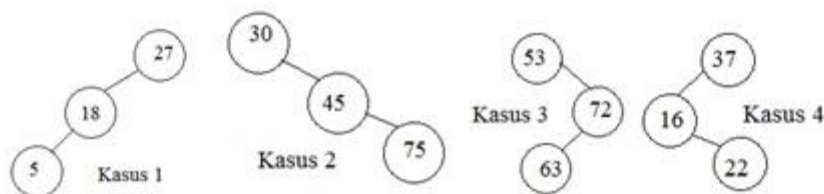
AVL Tree adalah Binary Search Tree yang memiliki perbedaan tinggi/ level maksimal 1 antara subtree kiri dan subtree kanan. AVL Tree muncul untuk menyeimbangkan Binary Search Tree. Dengan AVL Tree, waktu pencarian dan bentuk tree dapat dipersingkat dan disederhanakan. Selain AVL Tree, terdapat pula Height Balanced n Tree, yakni Binary Search Tree yang memiliki perbedaan level antara subtree kiri dan subtree kanan maksimal adalah n sehingga dengan kata lain AVL Tree adalah Height Balanced 1 Tree.



Gambar 3 Perbedaan level AVL Tree maksimal 1 antara subtree kiri dan kanan

Ada 4 kasus yang biasanya terjadi saat operasi *insert* dilakukan, yaitu : anggap T adalah node yang harus diseimbangkan kembali:

- a. Kasus 1 : node terdalam terletak pada subtree kiri dari anak kiri T (left-left)
- b. Kasus 2 : node terdalam terletak pada subtree kanan dari anak kanan T (right-right)
- c. Kasus 3 : node terdalam terletak pada subtree kanan dari anak kiri T (right-left)
- d. Kasus 4 : node terdalam terletak pada subtree kiri dari anak kanan T (left-right)



Ke-4 kasus tersebut dapat diselesaikan dengan melakukan rotasi

- a. Kasus 1 dan 2 dengan single rotation
- b. Kasus 3 dan 4 dengan double rotation

Menghapus node pada AVL Tree sama dengan menghapus binary search tree procedure dengan perbedaan pada penanganan kondisi tidak balance. . Penanganan kondisi tidak balance pada operasi menghapus node AVL tree, serupa dengan pada operasi penambahan. Mulai dari node yang diproses (dihapus) periksa seluruh node pada jalur yang menuju root (termasuk root) untuk menentukan node tidak balance yang pertama. Terapkan single atau double rotation untuk menyeimbangkan tree.

Prosedur Praktikum

E. Percobaan Pertama: Implementasi AVL Tree

Beberapa method sama atau serupa dengan Binary Search Tree. Perbedaan utama terdapat pada tambahan proses balancing dengan single dan double rotation. Perlu tidak nya dilakukan balancing perlu diperiksa setiap kali melakukan insert dan remove.

a. Buatlah program sesuai dengan source code Program Latihan Praktikum 12.1

b. Jalankan program

```

1  #include<iostream>
2  #include<cstdio>
3  #include<sstream>
4  #include<algorithm>
5  #define pow2(n) (1 << (n))
6  using namespace std;
7
8  struct avl_node{
9      int data;
10     struct avl_node *left;
11     struct avl_node *right;
12 }*root;
13
14 class avlTree{
15     public:
16         int height(avl_node *);
17         int diff(avl_node *);
18         avl_node *rr_rotation(avl_node *);
19         avl_node *ll_rotation(avl_node *);
20         avl_node *lr_rotation(avl_node *);
21         avl_node *rl_rotation(avl_node *);
22         avl_node* balance(avl_node *);
23         avl_node* insert(avl_node *, int );
24         void display(avl_node *, int);
25         avlTree(){
26             root = NULL;
27         }
28 };
29
30 // Height of AVL Tree
31 int avlTree::height(avl_node *temp){
32     int h = 0;
33     if (temp != NULL){
34         int l_height = height (temp->left);
35         int r_height = height (temp->right);

```



```

36         int max_height = max (l_height, r_height);
37         h = max_height + 1;
38     }
39     return h;
40 }
41
42 //Height Difference
43 int avlTree::diff(avl_node *temp){
44     int l_height = height (temp->left);
45     int r_height = height (temp->right);
46     int b_factor= l_height - r_height;
47     return b_factor;
48 }
49
50 //Right- Right Rotation
51 avl_node *avlTree::rr_rotation(avl_node *parent){
52     avl_node *temp;
53     temp = parent->right;
54     parent->right = temp->left;
55     temp->left = parent;
56     return temp;
57 }
58
59 //Left- Left Rotation
60 avl_node *avlTree::ll_rotation(avl_node *parent){
61     avl_node *temp;
62     temp = parent->left;
63     parent->left = temp->right;
64     temp->right = parent;
65     return temp;
66 }
67
68 //Left - Right Rotation
69 avl_node *avlTree::lr_rotation(avl_node *parent){
70     avl_node *temp;
71     temp = parent->left;
72     parent->left = rr_rotation (temp);
73     return ll_rotation (parent);
74 }
75
76 // Right- Left Rotation
77 avl_node *avlTree::rl_rotation(avl_node *parent){
78     avl_node *temp;
79     temp = parent->right;
80     parent->right = ll_rotation (temp);
81     return rr_rotation (parent);
82 }
83
84 // Balancing AVL Tree
85 avl_node *avlTree::balance(avl_node *temp){
86     int bal_factor = diff (temp);
87     if (bal_factor > 1){
88         if (diff (temp->left) > 0)

```

```

89         temp = ll_rotation (temp);
90     else
91         temp = lr_rotation (temp);
92     }
93     else if (bal_factor < -1){
94         if (diff (temp->right) > 0)
95             temp = rl_rotation (temp);
96         else
97             temp = rr_rotation (temp);
98     }
99     return temp;
100 }
101
102 // Insert Node Kedalam Tree
103 avl_node *avlTree::insert(avl_node *root, int value){
104     if (root == NULL){
105         root = new avl_node;
106         root->data = value;
107         root->left = NULL;
108         root->right = NULL;
109         return root;
110     }
111     else if (value < root->data){
112         root->left = insert(root->left, value);
113         root = balance (root);
114     }
115     else if (value >= root->data){
116         root->right = insert(root->right, value);
117         root = balance (root);
118     }
119     return root;
120 }
121
122 /* Menampilkan AVL Tree
123 void avlTree::display(avl_node *ptr, int level){
124     int i;
125     if (ptr!=NULL){
126         .....
127     }
128 }*/
129
130 int main(){
131     avlTree avl;
132     int nilai;
133     cout << "Masukkan nilai data berikut:" << endl;
134     // sisip data 10 bilangan acak dari 0-99 ke dalam tree
135     srand(time(NULL));
136     for (int i = 0; i < 5; i++) {
137         nilai = rand() % 100;
138         cout << nilai << " ";
139         root = avl.insert(root, nilai);
140     }
141

```

142	// avl.display(root, 1);
143	return 0;
144	}

Hasil Percobaan

5. Tulislah urutan data random yang dimasukkan dari Program Latihan Praktikum 12.1!

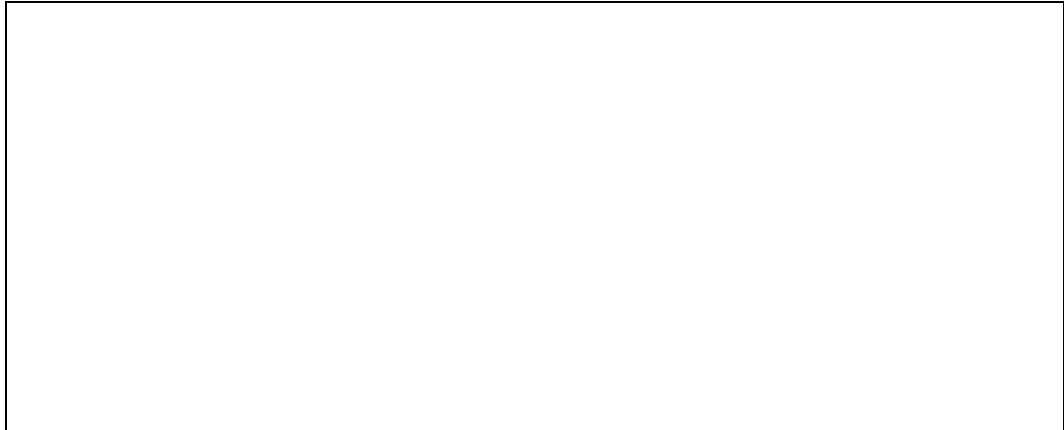
.....

.....

6. Gambarkan secara manual AVL Tree untuk setiap data yang dimasukkan sesuai urutan data pada hasil percobaan 1.

Gambar AVL untuk data pertama dimasukkan	Gambar AVL untuk data kedua dimasukkan
Gambar AVL untuk data ketiga dimasukkan	Gambar AVL untuk data keempat dimasukkan
Gambar AVL untuk data kelima dimasukkan	

7. Gambarkan juga hasil akhir **BST** berdasarkan urutan data pada hasil percobaan 1 yang nanti akan digunakan sebagai pembanding pada analisis hasil.



Analisis Hasil

5. Berdasarkan hasil gambar BST dan AVL Tree pada percobaan diatas, Apa yang membedakan antara BST dengan AVL Tree?
6. Kenapa perlu dilakukan proses single rotation dan double rotation untuk membentuk AVL Tree? Apa yang membedakan kedua proses tersebut?

Kesimpulan

Tuliskan kesimpulan dari percobaan diatas.

Latihan

Pada Program Latihan Praktikum 12.1, hapus tanda “//” pada baris 142 dan lengkapi kode method untuk Menampilkan AVL Tree pada baris 123. Contoh tampilan adalah sebagai berikut:

```
Masukkan nilai data berikut:
4 33 36 92 32

                                92
                               /
                             36
                             /
Root -> 33                 /
                          4
                           \
                           32
```

Gambar 4 Contoh Tampilan AVL Tree

Tugas

Kembangkan Program Latihan Praktikum 12.1 dengan menambahkan method untuk menghapus suatu node pada pohon AVL dengan 3 kondisi yaitu jika node yang dihapus adalah daun, jika node yang dihapus mempunyai satu anak dan jika node yang dihapus mempunyai 2 anak.

DAFTAR PUSTAKA

- Goodrich, M.T., Tamassia, R., dan Mount, D., 2003, "Data Structures and Algorithms in C++ 2nd edition", John Wiley & Sons.
- Sahni, S., 2005, "Data Structures, Algorithms and Applications in C++ 2nd edition", Universities Press.
- Sanjaya, D., 2005, "Asyiknya Belajar Struktur Data di Planet C++", PT Elex Media Komputindo.