Struktur Data

Diktat kuliah

Dr. Arya Adhyaksa Waskita

STMIK Eresha - 2017

Daftar Isi

Daftar Gambar

Daftar Program

Kata Pengantar

Diktat kuliah struktur data diperuntukkan bagi peserta mata kuliah struktur data STMIK Eresha semester II. Diktat ini menggunakan buku [Drozdek, 2001, Group, 2005] sebagai acuan utama. Dilengkapi juga acuan tambahan dari berbagai sumber online yang sesuai.

Serpong, 9 Mei 2017

Dr. Arya Adhyaksa Waskita

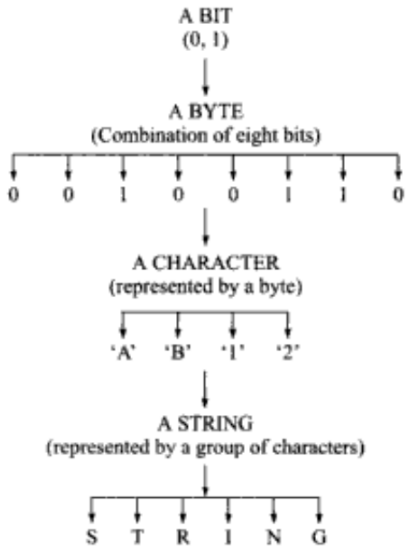
Bab 1

Pendahuluan

Memanfaatkan komputer dalam menyelesaikan suatu masalah menuntut pengetahuan tentang transformasi masalah tersebut agar dapat diselesaikan dengan komputer. Materi struktur data digunakan untuk memahami transfromasi data yang terlibat dalam suatu masalah serta operasi yang berlaku pada data sehingga dapat diselesaikan dengan komputer. Transformasi tersebut tentang bagaimana data diorganisasikan, dikendalikan serta struktur yang harus dirancang dan diterapkan. Targetnya adalah sebuah solusi yang sederhana dan efisien.

1.1 Representasi data

Dalam komputer, data dan instruksi dinyatakan dalam bentuk biner, berupa susunan angka 0 dan 1 dengan arti tertentu. Susunan 8 `bit` biner disebut dengan `byte` digunakan untuk merepresentasikan karakter. Sedangkan kumpulan karakter akan menjadi string. Ilustrasinya ditunjukkan pada Gambar 1.1.



Gambar 1.1: Representasi data dalam komputer

Acuan dalam merepresentasikan data ke dalam sistem biner antara adalah ASCII (*American Standard Code for Information Interchange*) dan BCD (*binary-coded decimal*). Dan setiap bahasa pemrograman telah mendefinisikan representasi untuk perintah dan data yang digunakannya seperti ilustrasi pada Gambar 1.1. Setiap jenis data memiliki skema representasi yang berbeda. Representasi data integer dilakukan menggunakan 8 bit (1 byte) biner. Misalnya, 27 dinyatakan dalam bilangan biner sebagai 00011011. Sementara untuk menyatakan bilangan negatifnya, dapat dilakukan dengan dua pendekatan. Yang pertama adalah melakukan operasi komplemen pada setiap bit biner, yaitu setiap bit 0 diubah menjadi 1, sedangkan bit 1 menjadi 0. Dengan demikian, angka -27 jika dinyatakan dalam biner dengan operasi komplemen menjadi 11100100. Sedangkan representasi yang kedua adalah menambahkan angka 1 pada hasil operasi komplemen pertama. Sehingga angka -27 dapat dinyatakan dalam bilangan biner dengan skenario kedua sebagai 11100101.

Selan itu, bilangan desimal dapat juga dinyatakan dalam bilangan biner dengan terlebih dahulu memisahkan bagian bilangan bulat (mantissa) dengan bagian pangkat (eksponen)nya. Sebagai contoh, bilangan 209.52 dapat dinyatakan sebagai 20952.10^{-2} dalam bilangan basis 10. 20952 adalah bagian mantissa, sedangkan -2 adalah bagian eksponennya. Dengan skenario sebelumnya, angka 20952 dinyatakan dalam biner sebagai 101000111011. Sedangkan bagian eksponennya dinyatakan dalam skenario komplemen sebagai 11111101. Sehingga keduanya dapat digunakan untuk menyatakan bilangan 20952.10^{-2} sebagai 101000111011.11111101.

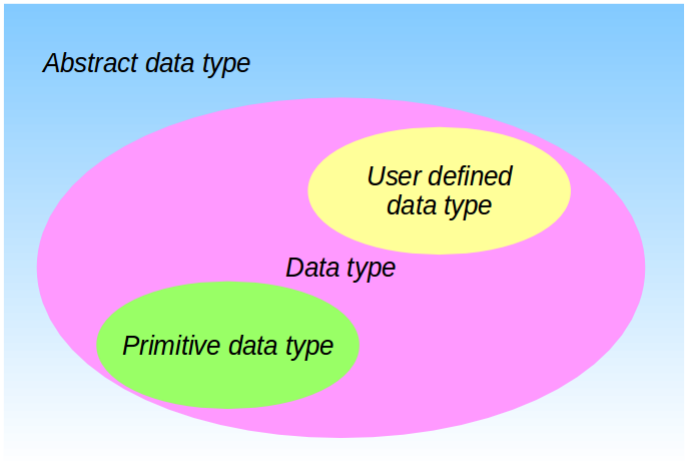
1.2 Tipe data abstrak

Pada kondisi tertentu, kita tidak dapat (sulit) merepresentasikan data atau obyek ke dalam komputer. Pada kondisi tersebut, bukan saja data yang perlu disimpan, tetapi operasi matematika yang berlaku pada data tersebut. Datanyapun seringkali berupa data jamak dari tipe yang berbeda. Bahasa pemorgraman modern menyebut tipe data ini sebagai tipa data abstrak (ADT/*Abstract Data Type*) yang muncul dalam bahasa pemrograman berorientasi obyek.

Untuk kasus yang lebih sederhana, di mana fokus utamanya justru pada sejumlah data yang perlu diacu bersama di saat yang sama, kita mengenal tipe data `structure`. Di sini, kita dapat mendefinisikan satu tipe data baru yang terdiri dari sejumlah tipe data yang telah didefinisikan dalam bahasa pemrograman yang kita gunakan.

Terdapat dua istilah terkait tipe data ini. Selain data struktur yang merupakan data yang tersusun dari sejumlah elemen data, terdapat juga tipe struktur *structured type*. *Structure type* merupakan hubungan yang terdapat pada data (elemen). Jika data struktur fokus pada data dengan elemen lebih dari satu, *structured type* fokus pada hubungan antar data.

Dapat disimpulkan bahwa ADT merupakan spesifikasi, sementara tipe data adalah penerapan dari ADT. ADT dapat juga dipandang sebagai merupakan bentuk umum dari tipe data. Jika telah didefinisikan dalam bahasa pemrograman tertentu sering disebut sebagai *primitive data type*, sedangkan jika didefinisikan oleh pengguna disebut sebagai *user defined data type*. Sementara data struktur adalah kumpulan data dengan tipe apapun. Hubungannya dapat diilustrasikan dalam Gambar 1.2 berikut.



Gambar 1.2: Hubungan tipe data

1.3 Prinsip dasar pemrograman

Prinsip dasar dalam pemrograman tahapan yang harus dilakukan dalam menyelesaikan masalah menggunakan bantuan komputer. Salah satu tahapan dalam pemrograman terstruktur dikenal sebagai *system development life cycle* (SDLC). Tahapan SDLC adalah sebagai berikut.

1. Analisis masalah: apa yang menjadi masalah, apa saja solusinya, apa dan siapa (pemrograman berorientasi obyek) yang terlibat, tahapan dalam penyelelesaian, indikasi keberhasilan, dll.
2. Membuat *prototype*, umunya dilakukan dalam bentuk *pseduo code* atau diagram alir.
3. Membangun algoritma:
   - merancang
   - verifikasi
   - analisis
   - memprogram
     - menguji
     - evaluasi
     - perbaikan (jika perlu)
     - optimasi
   - Merawat

Dalam merancang algoritma (termasuk juga program yang menerapkan algoritma) seperti disebutkan dalam tahapan ke-3 dari SDLC, diperlukan tahapan berikut.

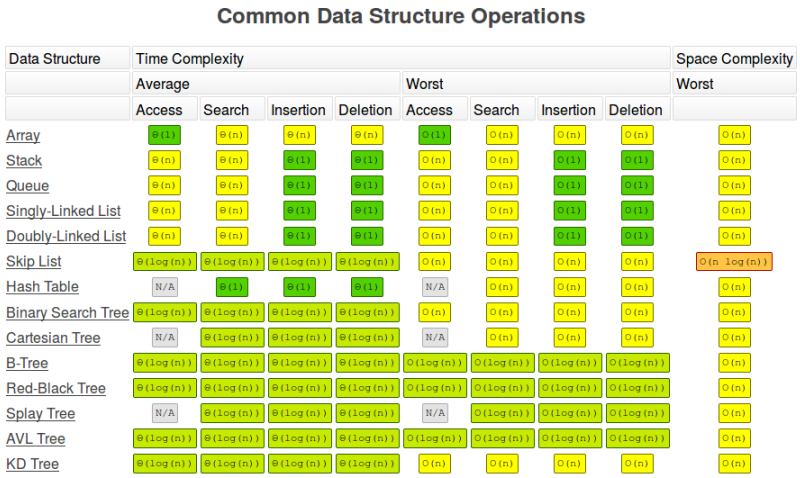
1. Program harus sejalan dengan masalah yang dihadapi, apakah itu prosedur, data maupun struktur datanya.

2. Bekerja baik pada semua kondisi di mana masalah terjadi. Jika ada kondisi di mana solusi yang digunakan tidak dapat mencapai hasil yang diinginkan, kondisi tersebut menjadi pengecualian dan harus disebutkan secara eksplisit.

3. Dokumentasi tentang bagaimana rancangan tersebut dihasilkan (dokumentasi rancangan)

4. Tersusun atas sejumlah modul, fungsi, subrutin. Modularisasi menjadi penting ketika algoritma yang dibangun komplek dan besar. Modularisasi akan memberikan keuntungan dari kemudahan pengembangan.

5. Waktu eksekusi dan ruang penyimpanan yang diperlukan. Kedua hal ini adalah kriteria sebuah algoritma disebut baik. Kebutuhan media penyimpanan yang dimaksud di sini meliputi media yang bersifat *volatile* (*Random Access Memory*/RAM) atau *non-volatile* (*hard disk*). Sementara waktu eksekusi tentu akan berbeda ketika dijalankan di mesin yang berbeda, sehingga ukuran tepatnya adalah jumlah operasi aritmatika dan logika yang dijalankan sebuah algoritma. Nilainya dinotasikan sebagai Θ atau $\mathbf{O}$. Nilai notasi Θ untuk beragam struktur data dan algoritmanya diilustasikan pada Gambar 1.3[1].



Gambar 1.3: Perbandingan nilai Θ sejumlah struktur data dan algoritma

Algoritma yang dirancang selanjutnya dianalisis kinerjanya dalam bentuk waktu eksekusi. Selain dipengaruhi oleh mesin di mana algoritma dijalankan, waktu eksekusi juga dipengaruhi oleh masukannya. Sebagai ilustrasi, mengurutkan sejumlah nilai yang telah terurut sebelumnya tentu membutuhkan waktu yang lebih singkat dibanding dengan nilai-nilai yang belum terurut. Waktu eksekusi yang dipengaruhi masukan dapat dikelompokkan dalam 3 kategori.

1. *Best*: waktu minimum yang diperlukan sebuah algoritma.

2. *Average*: waktu rerata dari eksekusi sebuah algoritma dalam menyelesaikan masalah yang sama dengan masukan yang berbeda. Jika satu masalah memiliki kemungkinan masukan

[1] http://bigocheatsheet.com/

sebanyak n, maka waktu rerata adalah rata-rata dari waktu untuk menyelesaikan masalah dengan n masukan tersebut. Jika waktu untuk menyelesaikan algoritma untuk satu jenis masukan adalah T_i, maka waktu rerata (T_{avg}) dapat diformulasikan dalam persamaan (1.1).

3. *Worst*: waktu terburuk sebuah algoritma dalam mengolah satu jenis masukan.

$$T_{avg} = \frac{\sum_{i=1}^{n} T_i}{n} \tag{1.1}$$

Algoritma sendiri dapat didefinisikan sebagai urutan instruksi yang harus diikuti untuk menyelesaikan suatu masalah. Atau dapat juga didefinisikan sebagai prosedur untuk mengubah *input* menjadi *output*. Karakteristik algoritma yang baik adalah sebagai berikut.

1. Setiap instruksi di dalamnya harus unik dan tepat.
2. Setiap instruksi tidak boleh dijalankan secara berulang tanpa batas.
3. Perulangan tugas yang sama harus dihindari
4. Harus memberikan hasil yang tepat untuk masalah yang dihadapi
5. Efisien dalam menyelesaikan masalah. Faktor efisiensi seperti yang telah disebutkan terdiri dari efisiensi terkait media penyimpanan dan waktu eksekusi.

Bab 2

Bahasa Pemrograman C++

2.1 Pendahuluan

Bahasa pemrograman `C++` dipilih dalam diktat ini karena `C++` secara eksplisit memanfaatkan *pointer* sebagai sarana membangun beragam struktur data. Sementara bahasa pemrograman lain yang lebih modern telah mengemas hal tersebut ke dalam class. Sebagai bahan ajar, `C++` dinilai lebih mampu memberikan pemahaman tentang merancang dan menerapkan rancangan struktur data ketimbang hanya menggunakan class yang telah terdefinisi dengan baik. Program 2.1 memperlihatkan program sederhana `C++`.

Program 2.1: hello.cpp

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5   cout << "Hello world\n";
6   return 0;
7 }
```

Penjelasannya adalah sebagai berikut.

1. Baris ke-1, merupakan pemanggilan pustaka di mana fungsi-fungsi yang digunakan dalam program diletakkan. Dalam contoh Program 2.1, digunakan fungsi `cout` yang digunakan untuk menampilkan data ke *standard output* (layar) dan fungsi tersebut didefinisikan di pustaka `iostream`.

2. Baris ke-2, digunakan untuk menyederhanakan penulisan fungsi. Karena `C++` adalah bahasa pemrograman berorientasi obyek, setiap fungsi tentu terdefinisi di dalam class tertentu, sehingga pemanggilan terhadap fungsi `cout` dilakukan apa adanya. Sebaliknya, tanpa deklarasi ini pemanggilan fungsi `cout` harus dilakukan dengan cara `std::cout`.

3. Baris ke-4, setiap program berbasis `C/C++` akan dieksekusi dari fugnsi yang bernama `main`. Tanpa fungsi ini, program `C/C++` tidak dapat dieksekusi (proses kompilasi tidak dapat menghasilkan *executable file*).

4. Baris ke-5, digunakan untuk menampilkan teks berupa "Hello World" ke layar.

5. Baris ke-6, digunakan untuk memenuhi syarat berupa *return value* dari fungsi main (Baris ke-4). Biasanya, fungsi yang mengembalikan nilai 0 disepakati sebagai fungsi yang menjalankan tugas dengan baik. Jika tidak ingin melakukan pengembalian fungsi, maka fungsi `main` harus diberi tanda *return value* sebagai `void`.

6. Setiap fungsi dalam `C/C++` dimulai oleh karakter "{" dan diakhiri oleh karakter "}", dan setiap instruksi di akhiri dengan karakter ";".

7. Di terminal, jalankan perintah `g++ -o hello hello.cpp`, dengan penjelasan.

   - `g++`: memanggil aplikasi kompilator `C++`
   - `-o`: opsi untuk melakukan kompilasi dan *linking* (meski dalam konteks Program 2.1 tidak diperlukan) sehingga dihasilkan *executable file*.
   - `hello`: nama executable file, jika dijalankan di Microsoft Windows dengan bantuan aplikasi MinGW, secara otomatis ditambahkan ekstensi `.exe`.
   - `hello.cpp`: kode sumber yang akan dikompilasi

2.2 IDE

IDE (*Integrated Development Environment*) adalah aplikasi yang digunakan dalam mengembangkan program. Untuk tujuan mempelajari materi struktur data, sebenarnya nyaris tidak diperlukan IDE. Kita hanya perlu memiliki kompilator *C++* sebagai bahasa pemrograman yang memiliki notasi eksplisit terkait *pointer* (dijelaskan dalam sub bab 2.3) serta editor teks untuk menulis sejumlah instruksi `C++`. Untuk pengguna sistem operasi GNU/Linux dalam berbagai variannya, nyaris semua jenis kompilator tersedia secara gratis. Saya sendiri menggunakan editor teks Geany[1] dengan kompilator `C++` dari GNU *Project*.

Namun, tidak demikian halnya dengan pengguna sistem Operasi Microsoft Windows. Umumnya, aplikasi di sistem operasi ini dikemas secara terintegrasi, bahkan dengan asumsi kita sedang berhadapan dengan sebuah proyek besar yang butuh cara mengelola proyek (dalam hal ini *software*) yang mumpuni. Padahal, di sisi lain, kita hanya memerlukan editor teks tempat di mana kita menyusun program serta kompilator yang akan membuat program tersebut dapat dijalankan. Beruntung, ada sejumlah aplikasi yang dapat memberikan kita lingkungan pengembangan yang sederhana seperti halnya di sistem operasi GNU/Linux. Misalnya dengan aplikasi MinGW (Minimalist GNU for Windows)[2] atau Cygwin[3]. Jangan lupa untuk memasang kompilator `C++` pada keduanya. Untuk yang masih kesulitan menggunakan aplikasi berbasis perintah baris seperti pada MinGW dan Cygwin, dapat menggunakan aplikasi IDE yang canggih seperti Visual Studio atau NetBeans dengan *plugin* kompilator `C/C++`.

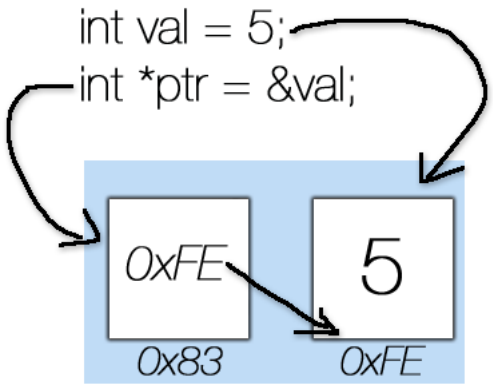
[1] https://www.geany.org/
[2] http://www.mingw.org/
[3] https://www.cygwin.com/

2.3 Tipe data `pointer` dan `struct`

Tipe data *pointer* adalah salah satu tipe data yang didefinisikan di `C/C++`. Tugasnya untuk menunjuk lokasi memori yang menjadi perhatian. Tipe data ini, tidak untuk menyimpan nilai suatu variabel tetapi alamat memori secara fisik di mana suatu nilai disimpan. Meskipun hanya bertugas menunjuk, tetapi tipe datanya harus disesuaikan dengan nilai yang akan ditunjuk.

Sebagai contoh, untuk mendefinisikan sebuah variabel *integer*, `C/C++` memiliki sintaks `int a;`. Untuk membuat sebuah tipe data *pointer* yang akan menunjuk variabel bertipe *integer*, `C/C++` memiliki sintaks `int *p;`. Sedangkan untuk membuat *pointer* `p` menunjuk variabel `a`, `C/C++` memilki sintaks `p=&a;`. Proses *assignment* tersebut juga dapat dilakukan sekaligus pada saat deklarasi dengan sintaks `int *p=&a;`. Intinya, tipe data *pointer* dicirikan dengan karakter "`*`" di awal variabel dan mengikuti variabel yang ditunjuk. Jika akan digunakan menunjuk vairabel *integer*, maka *pointer* juga harus dideklarasikan sebagai *integer*.

Sebagai contoh, di Gambar 2.1[4], terdapat dua deklarasi variabel. Yang pertama adalah `val` dengan tipe *integer* yang saat dieksekusi di simpan dalam RAM di alamat `0xFE`. Sedangkan yang kedua adalah `*ptr` dengan tipe `pointer` ke *integer*, tepatnya variabel `val`. Variabel `pointer` disimpan di alamat `0x83` saat diesekusi.



Gambar 2.1: Ilustrasi penggunaan variabel dengan tipe pointer

Kemudian, ada tipe data lainnya yang bertugas untuk mengemas sejumlah tipe data yang telah terdefinisi dalam `C/C++` yang disebut sebagai `struct`. Tipe data ini memungkinkan kita membuat tipe data baru yang mirip dengan sebuah obyek tanpa fungsi, karena hanya memiliki atribut. Sekali diacu dalam program, maka kita dapat mengakses semua data yang dikemas dalam `struct`. Program 2.2 mengilustrasikan penggunaan variabel `pointer` dan `struct`. Sedangkan Gambar 2.2 menunjukkan hasilnya saat dieksekusi.

Kedua tipe data inilah yang akan banyak digunakan dalam perkuliahan struktur data. Sedangkan yang lainnya hanya pelengkap saja.

[4]http://stackoverflow.com/questions/4483653/can-you-explain-the-concept-of-the-this-pointer

Program 2.2: intro.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  typedef struct data {
5          int nilai;
6          struct data *pointer;
7  }DATA;
8
9  int main() {
10         int a;
11         a=10;
12         int *p;
13         p=&a;
14         DATA x;
15         x.nilai=7;
16         x.pointer=NULL;
17
18         cout << "Nilai variabel a adalah " << a << " dan berlokasi di alamat " << &a << endl;
19         cout << "Nilai variabel *p adalah " << *p << " dan berlokasi di alamat " << p << endl;
20         cout << "x.nilai=" << x.nilai << ", x.pointer merujuk ke alamat " << x.pointer << endl;
21         return 0;
22 }
```

Penjelasannya adalah sebagai berikut.

1. Baris ke-4 s/d 7: definisi tipe data baru berupa `struct data`. Pernyataan `typedef` di depannya menunjukkan bahwa tipe data `struct data` memiliki nama alias `DATA`. Itu sebabnya di baris ke-4 tertulis `DATA` yang berarti nama alias untuk `struct data`.

   - Baris ke-5: definisi penyusun pertama `struct data`, yaitu `int nilai;`.
   - Baris ke-6: definisi penyusun kedua `struct data`, yaitu `struct data *pointer;`. Karena penyusun kedua ini ditugaskan untuk menujukan variabel dengan tipe `struct data`, maka ia harus didefinisikan dengan tipe yang sama.

2. Baris ke-10 dan 11: deklarasi dan *assignment* variabel *integer* dengan nama `a` dan dengan nilai `10`.

3. Baris ke-12 dan 13: deklarasi dan *assignment* variabel `pointer` dengan nama `*p` dan bernilai `&a` (alamat dari variabel `a` yang disimpan di RAM).

4. Baris ke-14: deklarasi variabel `DATA` dengan nama `x`. Perhatikan kembali cara mendeklarasi variabel di baris ke-10 dan 12.

5. Baris ke-15 dan 16: *assignment* nilai elemen penyusun variabel `x`, yaitu

   - `x.nilai=7`
   - `x.pointer=NULL`: `pointer` belum menunjuk ke variabel apapun yang bertipe `DATA`.

6. Baris ke-18 s/d 20: menuliskan nilai-nilai yang sebelumnya di-*assign* ke variabel untuk ditampilkan di layar.

7. Baris tanpa penjelasan memiliki penjelasan yang sama dengan penjelasan Program 2.1.

```
arya@arya-laptop:~/Documents/Kuliah/2017/StrukturData/diktat/coding$ ./intro
Nilai variabel a adalah 10 dan berlokasi di alamat 0x7ffd7556b294
Nilai variabel *p adalah 10 dan berlokasi di alamat 0x7ffd7556b294
x.nilai=7, x.pointer merujuk ke alamat 0
```

Gambar 2.2: Hasil eksekusi Program 2.2 di terminal

2.4 Pemanggilan fungsi

C/C++ memiliki dua skema pemanggilan fungsi, masing-masing adalah pemanggilan fungsi melalui nilai (*calling by value*) dan melalui referensi (*calling by reference*). Skema pertama, argumen fungsi adalah nilai dari yang akan dioperasikan. Jika nilai tersebut terwakili oleh sebuah variabel dalam fungsi asal, maka operasi yang dilakukan tidak akan mempengaruhi nilai pada variabel asalnya. Sebaliknya, pemanggilan fungsi dengan skema kedua memungkinkan perubahan pada nilai variabel asal. Hal ini disebakan karena pemanggilan fungsi dilakukan melalui lokasi absolut RAM di mana nilai tersebut di simpan. Program 2.3 akan menunjukkan perbedaannya. Sedangkan Gambar 2.3 adalah hasil dari eksekusi Program 2.3.

Program 2.3: function.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int byValue(int x) {
5          x=x*2;
6          return x;
7  }
8
9  int byReference1(int &x) {
10         x=x*2;
11         return x;
12 }
13
14 int byReference2(int *x) {
15         *x=*x*2;
16         return *x;
17 }
18
19 void tukar1(int a, int b) {
20         cout << "Nilai awal di fungsi tukar1: a=" << a << ", b=" << b << endl;
21         int i;
22         i=a;
23         a=b;
24         b=i;
25         cout << "Nilai akhir di fungsi tukar1: a=" << a << ", b=" << b << endl;
26 }
27
28 void tukar2(int &a, int &b) {
29 cout << "Nilai awal di fungsi tukar2: a=" << a << ", b=" << b << endl;
30         int i;
31         i=a;
32         a=b;
33         b=i;
34         cout << "Nilai akhir di fungsi tukar2: a=" << a << ", b=" << b << endl;
35 }
36
37 int main() {
38         int a,b;
39         cout << "a=";
```

```
40          cin >> a;
41          cout << "b=";
42          cin >> b;
43          tukar1(a,b);
44          cout << "Nilai di fungsi main: a=" << a << ", b=" << b << endl;
45          tukar2(a,b);
46          cout << "Nilai di fungsi main: a=" << a << ", b=" << b << endl;
47
48          int *p1,*p2;
49          b=byValue(a);
50          cout << "(a,b)=" << "(" << a << "," << b << ")" << endl;
51          b=byReference1(a);
52          cout << "(a,b)=" << "(" << a << "," << b << ")" << endl;
53          p1=&a;
54          b=byReference2(p1);
55          cout << "(a,b)=" << "(" << a << "," << b << ")" << endl;
56          cout << "*p1=" << *p1 << endl;
57          p2=p1;
58          cout << "*p2=" << *p2 << endl;
59
60          return 0;
61  }
```

```
a=5
b=7
Nilai awal di fungsi tukar1: a=5, b=7
Nilai akhir di fungsi tukar1: a=7, b=5
Nilai di fungsi main: a=5, b=7
Nilai awal di fungsi tukar2: a=5, b=7
Nilai akhir di fungsi tukar2: a=7, b=5
Nilai di fungsi main: a=7, b=5
(a,b)=(7,14)
(a,b)=(14,14)
(a,b)=(28,28)
*p1=28
*p2=28
```

Gambar 2.3: Hasil eksekusi Program 2.3

.

Penjelasan dari Program 2.3 adalah sebagai berikut.

1. Baris ke-4 s/d 7: isi dari fungsi yang menerima argumen berupa nilai. Nilai tersebut sejatinya dicopy ke lokasi lain dalam RAM untuk selanjutnya dioperasikan (dalam hal ini akan dikalikan dengan 2). Hasilnya dikembalikan ke fungsi yang memanggil.

2. Baris ke-9 s/d 12 serta baris ke-14 s/d 17: isi dari fungsi yang menerima argumen berupa alamat dari variabel yang akan dioperasikan. Operasi yang dilakukan fungsi pada variabel akan dilakukan di lokasi yang sama dengan lokasi variabel, sehingga hasilnya akan mengubah nilai asalnya. Kedua fungsi yang sama (`byReference1` dan `byReference2`) adalah fungsi dengan pemanggilan melalui referensi dengan notasi yang berbeda.

3. Baris ke-19 s/d 26: isi dari fungsi dengan pemanggilan melalui nilai, tetapi dengan yang berbeda dengan fungsi `byValue`. Di sini, fungsi hanya menjalankan tugas untuk melakukan pertukaran 2 nilai yang diberikan.

4. Baris ke-28 s/d 35: isi dari fungsi dengan pemanggilan melalui referensi, tetapi dengan tugas yang berbeda dengan fungsi `byReference1` dan `byReference2`. Di sini, fungsi hanya menjalankan tugas untuk melakukan pertukaran 2 variabel yang diberikan melalui alamatnya.

5. Baris ke-38 s/d 42: deklarasi 2 variabel *integer* serta *assignment* nilai ke kedua variabel tersebut secara langsung melalui *keyboard*.

6. Baris ke-43 dan 44: pemanggilan fungsi `tukar1` untuk kemudian nilainya ditampilkan kembali untuk melihat pengaruhnya. Maksud yang sama juga dilakukan baris ke-45 dan 46 untuk fungsi `tukar2`.

7. Baris ke-48: deklarasi variabel `pointer` dengan nama "`*p1`" dan "`*p2`".

8. Baris ke-49 dan 50: pemanggilan fungsi `byValue` untuk kemudian nilainya di-*assign* ke variabel `b`. Selanjutnya, nilai variabel `a` dan `b` untuk melihat pengaruhnya.

9. Baris ke-51 dan 52: pemanggilan fungsi `byReference1` yang menerima argumen berupa lokasi variabel `a` untuk kemudian nilainya di-*assign* ke variabel `b`. Selanjutnya, nilai variabel `a` dan `b` untuk melihat pengaruhnya. Maksud yang sama juga dilakukan baris ke-54 dan 55 dengan memanggil fungsi `byReference2`.

10. Sebelumnya, baris ke-53 dilakukan proses *assignment* alamat variabel `a` ke variabel pointer `*p1`. Argumen yang diterima fungsi `byReference1` dan `byReference2` sama-sama menerima argumen berupa alamat dari variabel `a`, hanya dengan notasi berbeda. Argumen ke fungsi `byReference1` adalah alamat variabel `a` secara langsung, sedangkan argumen ke fungsi `byReference2` alamat variabel `a` dilewatkan melalui variabel `pointer`. Perhatikan kembali Gambar 2.1.

11. Baris ke-56: menampilkan isi variabel `a` melalui variabel `*p1`.

12. Baris ke-57: melakukan *assigment* ke variabel `*p2` yang isinya adalah `*p1`. Dalam hal ini, variabel `*p1` dan `*p2` menunjuk lokasi yang sama, yaitu lokasi variabel `a`.

13. Baris ke-58: menampilkan isi variabel `a` melalui variabel `*p2`.

Bab 3

Array dan Linked list

3.1 *Array*

Kita dapat membuat variabel yang terdiri dari sejumlah elemen dengan tipe yang sama yang diacu sekaligus. Varibel tersebut, jika selama eksekusi jumlah elemennya tetap disebut sebagai *array*. Sedangkan jika jumlah elemennya berubah disebut sebagai *linked list*. Dengan *linked list*, elemen dari variabel dapat berubah selama eksekusi, baik nilai maupun jumlahnya. Perubahan jumlah elemen dapat terjadi di mana saja, apakah itu di awal, akhir maupun di tengah rangkaian data. Perhatikan Program 3.1 yang mengilustrasikan kegunaan array dalam `C++`. Dalam program tersebut, sebuah variabel *array* berelemen 10 bilangan *integer* diisi secara acak untuk kemudian ditampilkan isinya di layar.

Program 3.1: array.cpp

```
1  #include <iostream>
2  #include <cstdlib>
3  #include <ctime>
4  using namespace std;
5
6  int main() {
7          int a[10];
8          int i;
9          srand (time(NULL));
10         for (i=0;i<10;i++){
11                 a[i]=rand()%100;
12         }
13         for (i=0;i<10;i++){
14                 cout << "a[" << i << "]=" << a[i];
15                 if(i<10-1) {
16                         cout << ", ";
17                 }
18                 else {
19                         cout << endl;
20                 }
21         }
22         return 0;
23 }
```

Penjelasan dari Program 3.1 adalah sebagai berikut.

1. Baris ke-2: deklarasi pustaka untuk dapat membuat bilangan *pseudo random* (fungsi `srand`), sehingga pemanggilan fungsi untuk menghasilkan bilangan random selalu berbeda, dari satu eksekusi ke eksekusi selanjutnya.

2. Baris ke-3: deklarasi pustaka untuk fungsi `time`.

3. Baris ke-7: deklarasi variabel *array* dengan jumlah elemen 10 bilangan *integer*.

4. Baris ke-9: fungsi yang dapat membuat variabel `seed` selalu berubah pada setiap eksekusinya karena nilai bergantung pada saat program dieksekusi. Variabel `seed`[1] ini adalah variabel yang menentukan bilangan acak yang dihasilkan.

5. Baris ke-10 s/d 12: *looping* untuk mengisi *array* dengan bilangan acak.

6. Baris ke-13 s/d 21: *looping* untuk menampilkan nilai eleman *array*. Dari *looping* ini, terlihat bahwa elemen array dapat diakses langsung melalui nomor index.

3.2 *Linked list*

Untuk *linked list*, sebuah Program 3.2 dibuat untuk mensimulasikan penggunaannya. Di awal, *linked list* akan dibuat dengan jumlah 10 elemen. Kemudian, operasi penambahan dan pengurangan elemen dilakukan, baik di awal, maupun akhir elemen *linked list*. Sedangkan Gambar 3.1 menunjukkan hasil eksekusinya. Program 3.2 telah memanfaatkan fitur orientasi obyek dari `C++`.

Yang perlu diperhatikan adalah program ini tidak memiliki *destructor*, fungsi yang mendestruksi semua obyek saat program selesai dijalankan. Karena itu, dari sisi penggunaan RAM, program ini terbilang buruk. Hal ini disebabkan karena setiap selesai menjalankan program ini, semua elemen yang pernah dibuat tetap berada di dalam RAM dengan status terisi, meskipun tidak pernah digunakan lagi. Peran yang dijalankan oleh *destructor* disebut sebagai *garbage collector* di pemrograman Java[2]. Efisiensi penggunaan media penyimpanan, baik RAM maupun *hardisk*, baik saat disimpan, dijalankan bahkan paska dijalankan yang menjadi kriteria baik tidaknya sebuah program. Hal ini disajikan dalam sub bab 1.3.

Program 3.2: linkedList.cpp

```
1  #include <iostream>
2  #include <cstdlib>
3  #include <ctime>
4  using namespace std;
5
6  typedef struct data {
7          int nilai;
8          struct data *pointer;
9  }DATA;
10
11 class LinkedList {
12         private:
13                 DATA *awal;
```

[1] https://en.wikipedia.org/wiki/Random_seed
[2] http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

```
14                  DATA *akhir;
15          public:
16                  LinkedList();
17                  void insertAwal();
18                  void insertAkhir();
19                  void display();
20                  void removeAwal();
21                  void removeAkhir();
22  };
23
24  LinkedList::LinkedList() {
25          awal=NULL;
26          akhir=NULL;
27          int i;
28          for(i=0;i<10;i++) {
29                  DATA *temp;
30                  if(i==0) {
31                          temp=new DATA;
32                          temp->nilai=rand()%100;
33                          temp->pointer=NULL;
34                          awal=temp;
35                          akhir=temp;
36                  }
37                  else {
38                          temp=new DATA;
39                          temp->nilai=rand()%100;
40                          temp->pointer=NULL;
41                          akhir->pointer=temp;
42                          akhir=temp;
43                  }
44          }
45  }
46
47  void LinkedList::display() {
48          DATA *temp=new DATA;
49          temp=awal;
50          while(temp!=NULL) {
51                  cout << temp->nilai;
52                  temp=temp->pointer;
53                  if(temp!=NULL) {
54                          cout << ", ";
55                  }
56                  else {
57                          cout << endl;
58                          delete temp;
59                  }
60          }
61  }
62
63  void LinkedList::insertAwal() {
64          DATA *temp=new DATA;
65          temp->nilai=rand()%100;
66          temp->pointer=awal;
67          awal=temp;
68  }
69
70  void LinkedList::insertAkhir() {
71          DATA *temp=new DATA;
72          temp->nilai=rand()%100;
73          temp->pointer=NULL;
74          akhir->pointer=temp;
75          akhir=temp;
```

```
76  }
77
78  void LinkedList::removeAwal() {
79          if(awal==NULL && akhir==NULL) {
80                  cout << "Linked list kosong\n";
81          }
82          else {
83                  DATA *temp=new DATA;
84                  temp=awal->pointer;
85                  awal=temp;
86          }
87  }
88
89  void LinkedList::removeAkhir() {
90          if(awal==NULL && akhir==NULL) {
91                  cout << "Linked list kosong\n";
92          }
93          else {
94                  DATA *temp=new DATA;
95                  temp=awal;
96                  while(temp->pointer!=akhir) {
97                          temp=temp->pointer;
98                  }
99                  temp->pointer=NULL;
100                 akhir=temp;
101         }
102 }
103
104 int main() {
105         LinkedList list;
106         srand (time(NULL));
107         list.display();
108         list.insertAwal();
109         list.display();
110         list.insertAkhir();
111         list.display();
112         list.removeAwal();
113         list.display();
114         list.removeAkhir();
115         list.display();
116         return 0;
117 }
```

```
83, 86, 77, 15, 93, 35, 86, 92, 49, 21
70, 83, 86, 77, 15, 93, 35, 86, 92, 49, 21
70, 83, 86, 77, 15, 93, 35, 86, 92, 49, 21, 80
83, 86, 77, 15, 93, 35, 86, 92, 49, 21, 80
83, 86, 77, 15, 93, 35, 86, 92, 49, 21


------------------
(program exited with code: 0)
Press return to continue
```

Gambar 3.1: Hasil eksekusi linkedList.cpp

Penjelasan dari Program 3.2 adalah sebagai berikut.

1. Baris ke-11 s/d 22: deklarasi class LinkedList. Class ini terdiri dari 2 atribut dan 6 fungsi, termasuk *constructor*. Atribut yang terlibat adalah variabel *pointer* yang bertugas menandai elemen pertama (`awal`) dan terakhir (`akhir`).

2. Baris ke-24 s/d 45: isi dari fungsi *constructor*. Di sini didefinisikan 10 elemen pertama

dari *linked list*. Pengisian 10 elemen pertama dilakukan dalam 2 tahap. Yang pertama, saat elemen pertama diisi, semua pointer `awal` dan `akhir` menunjuk pada elemen tersebut (baris ke-34 dan 35). Selain elemen pertama, elemen yang sebelumnya ditunjuk oleh pointer akhir di-*assign* agar menunjuk elemen yang baru (baris ke-41). Sementara pointer akhir di-*assign* untuk menunjuk elemen yang baru.

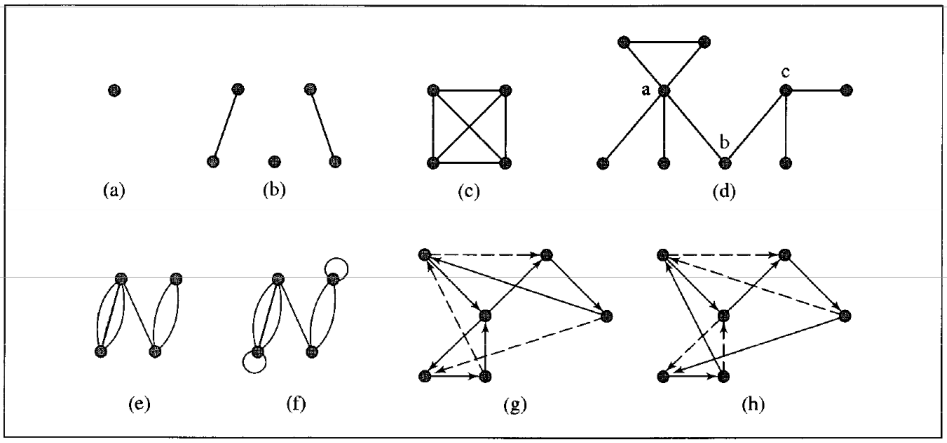
3. Baris ke-47 s/d 61: isi setiap elemen ditampilkan ke layar dimulai dari elemen pertama.
4. Baris ke-63 s/d 68: elemen baru ditambahkan sebelum elemen pertama, sehingga elemen baru tersebut sekarang menjadi elemen pertama dalam *linked list*.
5. Baris ke-70 s/d 76: elemen baru ditambahkan setelah elemen terakhir, sehingga elemen baru tersebut sekarang menjadi elemen terakhir dalam *linked list*.
6. Baris ke-78 s/d 87: elemen pertama dikeluarkan dari *linked list*.
7. Baris ke-89 s/d 102: elemen terakhir dikeluarkan dari *linked list*.
8. Baris ke-104 s/d 117: isi dari fungsi `main` berupa pendefinisian obyek `list`, `seed` serta pemanggilan terhadap fungsi-fungsi yang telah didefinisikan dalam *class* `LinkedList`.

Bab 4

Graph

4.1 Pendahuluan

Graph adalah sarana untuk menyimpan dan menganalisis metadata serta hubungan yang mungkin ada antar data tersebut. *Graph* terdiri dari dua elemen, masing-masing adalah **vertex** (yang dapat dianalogikan dengan *node*) serta **edge** yang menghubungkan sejumlah **vertex**. **Vertex** tanpa **edge** sudah dapat disebut sebagai *graph*, tetapi tidak sebaliknya. Gambar 4.1 menunjukkan beberapa jenis *graph*.



Gambar 4.1: Jenis-jenis *graph* [Drozdek, 2001]

Graph yang terdiri dari sejumlah **vertex**, dengan atau tanpa **edge** disebut sebagai *simple graph* (Gambar 4.1 (a)-(d)). Jika setiap **vertex** dalam *graph* saling terhubung oleh satu **edge**, *graph* disebut lengkap (*complete graph*). Contohnya ditunjukkan oleh Gambar 4.1 (c). Kemudian, ketika dua **vertex** berbeda saling terhubung melalui lebih dari satu **edge**, kondisi tersebut didefinisikan sebagai *multigraph*. Ilustrasinya ditunjukkan oleh Gambar 4.1 (e). Kondisi umum dari multigraph di mana **vertex** yang sama boleh dihubungkan dengan **edge** disebut sebagai *pseudograph*. Dengan demikian, akan terjadi *looping* **vertex** ke dirinya sendiri. Kondisi ini diilustrasikan oleh Gambar 4.1 (f).

Graph dengan **edge** berarah disebut sebagai *directed graph* atau *digraph*. Pada kondisi ini, dua vertex saling terhubung hanya pada arah tertentu. Sebaliknya, jika tidak ada batasan arah pada semua **edge** dalam *graph*, maka *graph* disebut sebagai *undirected graph*. Jika pada `edge` di-*assigned* sebuah nilai, maka *graph* disebut dengan *weighted graph*. Bobot dari *graph* bisa bermakna banyak hal, tergantung dalam penerapannya.

4.2 Contoh Program

Program 4.1 menunjukkan contoh penerapan *graph* dalam `C++` yang diambil dari laman http://www.sanfoundry.com/cpp-program-implement-adjacency-list/.

Program 4.1: adjList.cpp

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

struct AdjListNode {
        int dest;
    struct AdjListNode* next;
};

struct AdjList {
    struct AdjListNode *head;
};

class Graph {
        private:
                int V;
                struct AdjList* array;
        public:
                Graph(int V) {
                        this->V = V;
                        array = new AdjList [V];
                        for (int i = 0; i < V; ++i) {
                                array[i].head = NULL;
                        }
                }
                AdjListNode* newAdjListNode(int dest) {
                        AdjListNode* newNode = new AdjListNode;
                        newNode->dest = dest;
                        newNode->next = NULL;
                        return newNode;
                }
                void addEdge(int src, int dest) {
                        AdjListNode* newNode = newAdjListNode(dest);
                        newNode->next = array[src].head;
                        array[src].head = newNode;
                        newNode = newAdjListNode(src);
                        newNode->next = array[dest].head;
                        array[dest].head = newNode;
                }
                void printGraph() {
                        int v;
                        for (v = 0; v < V; ++v) {
                                AdjListNode* pCrawl = array[v].head;
                                cout<<"\n Adjacency list of vertex "<<v<<"\n head ";
                                while (pCrawl) {
```

```
46                                          cout<<"-> "<<pCrawl->dest;
47                                          pCrawl = pCrawl->next;
48                                      }
49                                      cout<<endl;
50                              }
51                      }
52  };
53
54  int main() {
55          Graph gh(5);
56          gh.addEdge(0, 1);
57          gh.addEdge(0, 4);
58          gh.addEdge(1, 2);
59          gh.addEdge(1, 3);
60          gh.addEdge(1, 4);
61          gh.addEdge(2, 3);
62          gh.addEdge(3, 4);
63          gh.addEdge(2, 0);
64          gh.printGraph();
65          return 0;
66  }
```

Bab 5

Pengurutan (*Sorting*)

5.1 Pendahuluan

Efisiensi pengelolaan data secara substansial dapat ditingkatkan jika data tersebut terurut [Drozdek, 2001]. Sebagai cotoh, sangat tidak praktis mencari nomor kontak seseorang ketika daftar kontak tidak terurut secara alfabet. *Sorting* adalah operasi yang dilakukan pada sejumlah data sedemikian rupa sehingga mereka terurut dengan aturan tertentu, baik secara alfabet atau alfa numerik maupun terurut secara *ascending* atau *descending* [Group, 2005].

Dalam operasi *sorting* efisiensi dalam waktu eksekusi lebih penting daripada penggunaan RAM [Group, 2005]. Hal ini disebabkan karena umumnya operasi *sorting* membutuhkan $\Theta(n)$ atau sebanyak data yang dioperasikan. Bahkan, algoritma *sorting* yang ideal melakukan pertukaran di tempat atau tidak memerlukan tambahan tempat selain yang digunakan untuk menyimpan data. Kita akan mempelajari beberapa teknik sorting yaitu *bubble sort*, *selection sort* dan *quick sort*. Gambar 5.1 menunjukkan perbandingan komplesitas beberapa algoritma *sorting*[1].

[1] http://bigocheatsheet.com/

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

Gambar 5.1: Perbandingan komplesitas algortima *sorting*

Secara praktis, operasi *sorting* tidak dapat dilakukan pada semua data. Sebagai contoh adalah struktur data *queue* yang terurut berdasarkan waktu kedatangan. Dengan demikian, tidak beralasan untuk mengurutkannya berdasarkan kriteria lain seperti alfabet atau alfanumerik. Tetapi, operasi *sorting* beralsan untuk dilakukan dalam kondisi jumlah elemen yang berubah sepanjang eksekusi. Sebagai contoh adalah pengurutan nomor kontak tadi. Karena itu, pengurutanpun harus dapat dilakukan di struktur data dinamis seperti *linked list*.

5.2 *Bubble sort*

Teknik ini adalah teknik yang sangat sederhana, yaitu dengan melakukan evaluasi pada dua elemen yang posisinya berurutan. Jika keduanya belum terurut, urutkan. Jika keduanya telah terurut, pindah ke posisi dua elemen berikutnya. Perhatikan contoh dalam Program 5.1 berikut. Perhatikan baris ke-19 dan 21. Perulangan yang dilakukan untuk mengurutkan elemen array membutuhkan kompleksitas algoritma sama dengan $\Theta(n^2)$.

Program 5.1: sorting.cpp

```
1  #include <iostream>
2  #include <cstdlib>
3  #include <ctime>
4  using namespace std;
5
6  int main() {
7          int n=10;
8          int a[n];
9          int i,j,temp;
10         srand(time(NULL));
11         for(i=0;i<n;i++) {
12                 a[i]=rand()%100;
13         }
14         cout << "Array awal:{";
15         for(i=0;i<n;i++) {
16                 cout << a[i] << ", ";
17         }
```

```
18          cout << "}" << endl << endl;
19          for(i=0;i<n;i++) {
20                  temp=0;
21                  for(j=0;j<n-1;j++) {
22                          if(a[j]>a[j+1]) {
23                                  temp=a[j];
24                                  a[j]=a[j+1];
25                                  a[j+1]=temp;
26                          }
27                  }
28          }
29          cout << "Sorted array:{";
30          for(i=0;i<n;i++) {
31                  cout << a[i] << ", ";
32          }
33          cout << "}" << endl;
34          return 0;
35  }
```

5.3 *Selection Sort*

Teknik pengurutan ini juga sederhana dan mirip dengan pengurutan *bubble*. Terdiri dari dua kali *looping* (baris ke-15 sebagai *loop* terluar dan baris ke-17 sebagai *loop* terdalam) serta mengasumsikan bahwa elemen terkecil terletak di elemen pertama (baris ke-16), algoritma ini berusaha untuk mencari nilai terkecil pada setiap *loop*. Di akhir *loop* terdalam, nilai elemen array ditukar, antara elemen pertama dengan posisi elemen di mana nilai terkecil ditemukan. Perhatikan baris ke-22, fungsi `swap` telah didefinisikan dalam `std` sehingga tidak terlihat deklarasi fungsinya. Harap berhati-hati dalam menggunakan pernyataan pada baris ke-4 jika kita berencana membuat fungsi dengan tugas yang dengan fungsi `swap` karena ada potensi penggunaan nama yang sama dan menyebabkan kesalahan. *Loop* kemudian dilanjutkan karena di setiap tahap, komparasi dilakukan hanya dengan satu elemen saja. Program 5.2[2] mengilustrasikan algoritma pengurutan *selection*.

Program 5.2: selection.cpp

```
1  #include <iostream>
2  #include <cstdlib>
3  #include <ctime>
4  using namespace std;
5
6  void print (int temp_ar[], int size) {
7          for (int i = 0; i < size; ++i) {
8                  cout << temp_ar[i] << " ";
9          }
10         cout << endl;
11 }
12
13 void selection_sort (int ar[], int size) {
14         int min_ele_loc;
15         for (int i = 0; i < 9; ++i) {
16                 min_ele_loc = i;
17                 for (int j = i + 1; j < 10; ++j) {
18                         if (ar[j] < ar[min_ele_loc]) {
```

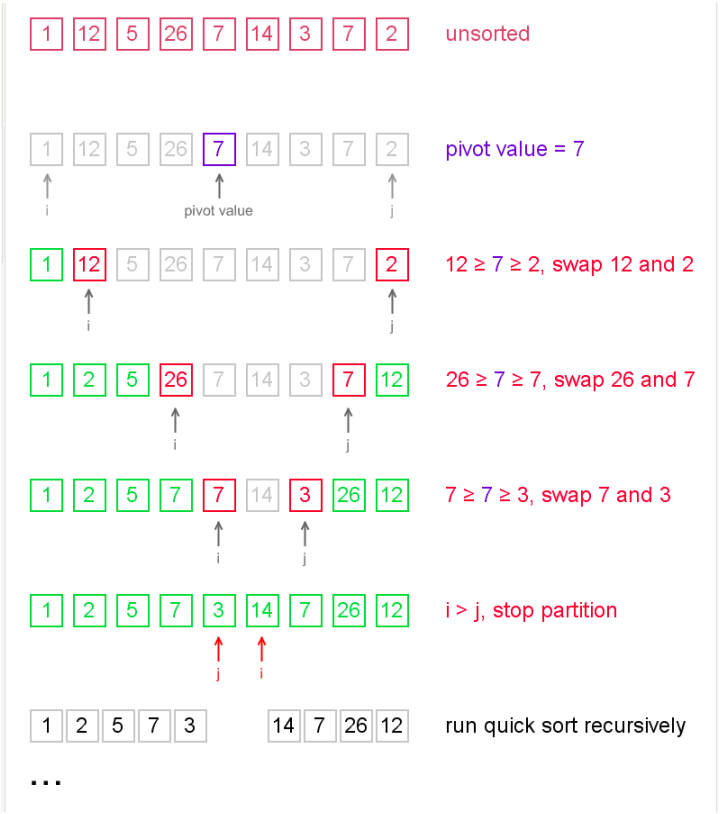
[2] http://www.sanfoundry.com/cplusplus-program-implement-selection-sort/

```
19                                      min_ele_loc = j;
20                              }
21                      }
22                      swap (ar[i], ar[min_ele_loc]);
23              }
24      }
25
26      int main () {
27              int i;
28              int ar[10];
29              srand(time(NULL));
30              for(i=0;i<10;i++) {
31                      ar[i]=rand()%100;
32              }
33              cout << "Array awal : ";
34              print (ar, 10);
35              selection_sort (ar, 10);
36              cout << "Array akhir : ";
37              print (ar, 10);
38              return 0;
39      }
```

5.4 *Quick Sort*

Sedikit berbeda dengan dua pendekatan *sorting* sebelumnya, *quick sort* melakukan dua perbandingan di setiap tahapnya. Acuan perbandingan dipilih sembarang, umumnya dari elemen yang posisinya di tengah dan disebut sebagai nilai *pivot*. Perbandingan terhadap nilai *pivot* dilakukan oleh elemen pertama (`i`) dan terakhir (`j`) dari array. Pertukaran dilakukan sedemikian sehingga terbentuk `array[i] < pivot < array[j]`. Posisi `i` digeser ke kanan (ke arah yang target nilainya semakin besar) sedangkan posisi `j` ke kiri. Tahapan pambandingan berakhir setelah jika posisi `i` sudah lebih besar daripada `j`. Tahapan yang sama kembali dilakukan secara rekursif. Gambar 5.2[3] mengilustrasikan algoritma ini. Sedangkan Program 5.3 menunjukkan penerapannya.

[3] http://www.algolist.net/Algorithms/Sorting/Quicksort



Gambar 5.2: Ilustrasi algoritma *quick sort*

Program 5.3: quicksort.cpp

```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

void quickSort(int arr[], int left, int right) {
        int i = left, j = right;
        int tmp;
        int pivot = arr[(left + right) / 2];
        while (i <= j) {
                while (arr[i] < pivot)
                        i++;
                        while (arr[j] > pivot)
                                j--;

                        if (i <= j) {
                                tmp = arr[i];
                                arr[i] = arr[j];
                                arr[j] = tmp;
                                i++;
                                j--;
                }
        };

      /* recursion */
        if (left < j)
```

```
27                  quickSort(arr, left, j);
28          if (i < right)
29                  quickSort(arr, i, right);
30  }
31
32  void print (int temp_ar[], int size) {
33          for (int i = 0; i < size; ++i) {
34                  cout << temp_ar[i] << " ";
35          }
36          cout << endl;
37  }
38
39  int main() {
40          int ar[10];
41          int i;
42          srand(time(NULL));
43          for(i=0;i<10;i++) {
44                  ar[i]=rand()%100;
45          }
46          cout << "Array awal : ";
47          print (ar, 10);
48          quickSort (ar, 0,9);
49          cout << "Array akhir : ";
50          print (ar, 10);
51  }
```

Bab 6

Tugas

Di kesempatan perkuliahan ini, mahasiswa diminta untuk membuat tulisan di blog dan media lainnya yang dapat diakses secara publik via internet. Tulisannya seputar penerapan struktur data dalam bahasa pemrograman, salah satunya `C++` yang telah dipelajari. Tulisan harus memiliki aspek berikut.

1. Skenario tentang struktur data seperti apa yang akan dibuat serta operasi apa saja yang dapat dijalankan pada struktur data tersebut.
2. Ilustrasi (berupa gambar) dari struktur data tersebut.
3. Penerapan program dari struktur data. Program dapat diambil dari kode sumber terbuka yang tersedia di internet. Pastikan menyertakan URL dari program tersebut. Sebagai contoh, mahasiswa dapat mengunjungi laman http://www.sanfoundry.com/cpp-programming-examples-data-structures/ untuk menyelesaikan tugas ini.
4. *Screenshot* hasil eksekusinya.

Bibliografi

[Drozdek, 2001] Drozdek, A. (2001). *Data Structures and Algorithms in C++*. Brooks/Cole.

[Group, 2005] Group, I. (2005). *Data Structures Using C*. Ace series. McGraw-Hill Education (India) Pvt Limited.