Struktur Data

Oleh:

Agus Sujarwadi

UNIVERSITAS TEKNOLOGI YOGYAKARTA

2017

Diterbitkan Oleh:
Universitas Teknologi Yogyakarta
Jl. Siliwangi, Jombor, Sleman, Yogyakarta
Email: publikasi@uty.ac.id
Website: www.uty.ac.id

Oleh : Agus Sujarwadi

KATA PENGANTAR

Alhamdulillah, puji syukur penulis panjatkan atas rahmat dan nikmat yang Allah SWT berikan kepada kita semua, sehingga penyusunan buku praktikum ini dapat terselesaikan.

Di tengah pesatnya perkembangan teknologi dan informasi saat ini, komputerisasi menjadi kebutuhan utama dalam pengembangan sistem informasi berbasis komputer. Agar suatu sistem informasi berbasis komputer dapat berjalan secara optimal, diperlukan proses analisis, perancangan, dan implementasi yang terstruktur pada fase pembuatannya. Suatu sistem dikatakan baik jika memiliki algoritme yang benar dan ditulis dengan suata bahasa pemrograman secara baik pula.

Buku **Praktikum Struktur Data** ini merupakan salah satu pegangan bagi mahasiswa dalam rangka membangun pola pikir untuk menyelesaikan suatu masalah secara logis dan sistematis kemudian mengimplementasikannya menggunakan bahasa pemrograman. Buku praktikum ini disusun untuk kegiatan praktikum selama satu semester yang terdiri atas 14 kali pertemuan dengan masing-masing pertemuan setara dengan 2 sks.

Penulis sangat menyadari bahwa buku praktikum ini masih jauh dari kata sempurna. Oleh sebab itu, penulis mengharapkan kritik dan saran dari para pembaca untuk menyempurnakan buku praktikum ini. Penulis berharap kekurangan yang ada pada buku praktikum ini tidak mengurangi esensi dan manfaat buku ini.

Yogyakarta, November 2017

Penulis

DAFTAR ISI

BAB I
DASAR PEMROGRAMAN BAHASA C++

Kompetensi dan Indikator

Kompetensi Dasar :
- Mengetahui dasar-dasar pemrograman meliputi struktur dasar bahasa pemrograman.
- Mengetahui penulisan komentar, pengenal, konstanta dan variabel pada bahasa pemrograman
- Mengetahui perintah-perintah dasar untuk menampilkan tampilan pada monitor pada bahasa pemrograman.

Indikator Keberhasilan :
- Mampu menjelaskan dasar-dasar bahasa pemrograman.
- Mampu menuliskan komentar, pengenal, konstanta dan variabel pada suatu program.
- Mampu membuat program sederhana untuk menampilkan tulisan pada monitor.

Dasar Teori

A. Pengantar

Bahasa pemrograman Bahasa C atau *C Language Programming* pada awalnya dikembangkan untuk digunakan pada sistem operasi Unix dan dikembangkan dari bahasa pemrograman yang muncul terlebih dahulu, yaitu Algol-60 (1960), CPL (1963), BCPL (1967) dan B (1970). Seiring dengan perkembangan bahasa pemrograman, Bahasa C kemudian dimunculkan dengan konsep dan kemampuan baru yang kemudian dikenal dengan C++. Bahasa pemrograman ini berorientasi pada objek sehingga memungkinkan pembuatan program menjadi lebih mudah dan cepat.

Fasilitas penting yang ada pada C++ adalah pembentukan class yang akan mampu melakukan operasi-operasi pemrograman *enkapsulasi* (perlindungan atas data dan fungsi class), *inheritance* (pewarisan data dan fungsi class) dan *polymorphism* (menggunakan satu nama fungsi untuk beberapa tujuan operasi program).

B. Fungsi

Fungsi merupakan sub program yang sengaja ditulis untuk mengerjakan tugas tertentu sehingga memberikan suatu keluaran nilai. Ketika program sudah berjalan, fungsi dapat dipandang sebagai kotak hitam *(black box*) dimana kita tidak perlu berfikir apa yang dikerjakan dalam fungsi itu, tetapi dengan parameter yang telah disiapkan secara otomatis akan menghasilkan nilai yang kita inginkan.

Program yang dibuat dengan menggunakan bahasa C++ dapat memiliki beberapa fungsi, tetapi setidak-tidaknya memiliki fungsi utama, yaitu **main()**. Selanjutnya definisi fungsi-fungsi lain dilakukan diluar fungsi itu. Dalam C++ kode yang dibaca adalah kode-kode yang terdapat pada fungsi utama, sehingga jika terdapat pemanggilan fungsi lain maka program akan mencari nama fungsi tersebut untuk dieksekusi dan akan kembali ke fungsi utama setelah fungsi itu selesai dikerjakan.

C. Struktur Dasar Bahasa C++

Program yang dibuat dengan Bahasa C++ selalu diawali dengan ***Preprocessor directive***, yaitu perintah yang diawali dengan tanda *pound* (#). Khusus untuk bagian awal program, selalu menggunakan ***Directive #include*** yang digunakan untuk mendefinisikan *file*

header di dalam kode program. File header (file yang berekstensi **.h**) adalah file yang berisi fungsi-fungsi dan telah dikompilasi sebelumnya. Cara penulisannya adalah:

#include<nama_file> atau **#include "nama_file"**

Contoh :

```
#include <stdio.h>;
#include <conio.h>;
#include <uiostream.h>;
```

Setelah *preprocessor* didefinisikan, diikuti dengan deklarasi fungsi. Adapun fungsi pertama yang harus ada di dalam program C++ sudah di tentukan namanya, yaitu bernama **fungsi main()** yang ditulis dengan diawali tipe fungsi, misalnya **void main(), int main()** atau **char main()**. Fungsi selalu dibuka dengan kurung kurawal buka „**{**„ dan diakhiri dengan kurung kurawal tutup „**}**". Diantara kurung kurawal tersebut dituliskan statemen-statemen program. Penggunaan fungsi yang lain diletakkan di bawah fungsi utama dengan cara yang sama.

Struktur dasar program :

```
Preprosesor directive;
Fungsi Utama()
{
 deklarasi variabel;
 statement-statement;
}

Fungsi lain()
{
 deklarasi variabel;
 statement-statement;
}
```

D. Atribut Pemrograman

1. Komentar

   Komentar pada suatu baris program merupakan suatu hal yang sangat penting sebagai dokumentasi jika suatu saat dilakukan modifikasi dan tidak ikut dibaca pada saat proses kompilasi.

   Komentar menggunakan tanda //, digunakan untuk memberikan komentar atas perintah-perintah yang ada di bawahnya.

   Contoh :

   ```
   // Program latihan ini dibuat pertama untuk latihan
   int luas; //deklarasi variabel luas bertipe data integer
   ```

   Komentar menggunakan tanda /*komentar*/, digunakan untuk menuliskan komentar yang banyaknya satu baris atau lebih, mulai dari /* sampai */. Dengan menggunakan tanda ini dapat digunakan untuk memberi komentar sisipan pada satu baris.

   Contoh :

   ```
   /* Program latihan ini dibuat pertama untuk latihan
   Mahasiswa dalam matakuliah Struktur Data
   ```

```
Di Kampus UTY */

int /*bertipe data integer*/ luas;
```

2. Pengenal

Pengenal merupakan pengidentifikasi dari nama-nama yang akan dideklarasikan agar *kompiler* dapat mengenalinya, meliputi nama variabel, konstanta, fungsi, kelas, template, maupun *namespace*.

Pengenal dalam konstanta dan variabel berfungsi untuk menampung nilai yang digunakan dalam program. Identifikasi ini dilakukan untuk mempermudah proses penanganan data atau nilai. Penggunaan pengenal sebaiknya disesuaikan dengan kebutuhan dan sifat-sifat nilai yang akan ditampungnya dan hindarilah penggunaan pengenal yang mirip antar pengenal. Adapun hal-hal yang berkaitan dengan pengenal antara lain :

☑ C++ bersifat case sensitive, sehingga penulisan huruf kecil dan kapital dianggap berbeda. Misalnya variabel **Luas** berbeda dengan **luas**. Contoh:

```
int Luas; //variabel Luas dengan „L" huruf kapital berbeda dengan
int luas; //variabel luas dengan „l" huruf kecil
```

☑ Tidak boleh diawali dengan angka atau seluruh pengenal berupa angka. Tetapi jika huruf awal adalah karakter dan selanjutnya adalah angka diperbolehkan. Contoh:

```
int gaji; //benar
int gajike13; //benar
int 2018; //salah, karena seluruhnya angka dan diawali angka
int 2kali; //salah, karena diawali dengan angka
```

☑ Tidak boleh mengandung spasi. Contoh:

```
int tgllahir; //benar
int tgl lahir; //salah, karena menggunakan spasi antara „tgl" dan „lahir"
```

☑ Tidak boleh menggunakan tanda baca dan simbol-simbol (#, @, ?, !, &, dan lainnya). Contoh:

```
int tgllahir; //benar
int tgl&lahir; //salah, karena menggunakan simbol „&"
int tgl-lahir; //salah, karena menggunakan tanda baca „ – „
```

☑ Tidak boleh menggunakan kata kunci (*keywords*). Contoh:

```
int break; //salah, karena „break" merupakan keywords
int while; //salah, karena „while" merupakan keywords
int panjang; //benar
```

3. Konstanta

Konstanta adalah jenis pengenal yang bersifat konstan atau tetap, sehingga nilai dari konstanta di dalam program tidak dapat diubah. Konstanta berguna untuk menentukan nilai yang bersifat tetapan, misalnya phi selalu bernilai 3.14.

Penggunaan konstanta dapat dilakukan dengan 2 (dua) cara yaitu :

a. Menggunakan ***Preprosessor Directive #define***

Struktur penulisan konstanta menggunakan preprosessor ini adalah:

#define <pengenal> <nilai>;

Contoh:

```
#define phi 3.14; //mendefinisikan konstanta „phi" dengan nilai 3.14
```

```
#define max 1000; //mendefinisikan konstanta „max" dengan nilai 1000
```

b. Menggunakan kata kunci ***Const***

Struktur penulisan konstanta menggunakan preprosessor ini adalah:
const <tipe_data> <pengenal=nilai>;

Contoh:

```
const double phi=3.14; // „phi" dengan nilai 3.14 bertipe double
const long max=1000; // „max" dengan nilai 1000 bertipe long integer
const char lagi="Y"; // „lagi" dengan nilai „Y" bertipe karakter
```

4. Variabel

Variabel adalah pengenal yang mempunyai nilai dinamis sehingga nilai yang disimpan di dalamnya dapat diubah selama program berjalan sesuai dengan kebutuhan. Struktur penulisannya adalah :

<type_data> <pengenal>;

Contoh:

```
int panjang; // deklarasi variabel „panjang" bertipe integer
int lebar, luas; // deklarasi variabel „lebar" dan „luas",keduanya bertipe integer
```

Pada deklarasi variabel juga dapat dilakukan inisialisasi (pemberian nilai awal), dengan struktur penulisan sebagai berikut:
<type_data> <pengenal=nilai_awal>;

Contoh:

```
int panjang=10; //deklarasi variabel „panjang" bertipe integer dengan nilai awal 10
double tunjangan=1.25; // variabel „tunjangan" bertipe double dengan nilai awal
```

5. Tipe Data

Tipe data berfungsi untuk merepresentasikan jenis dari sebuah nilai yang terdapat dalam program. Penggunaan tipe data yang benar sangat mendukung efisiensi program karena tipe data juga berpengaruh terhadap kapasitas data.

Dalam C++ terdapat beberapa tipe data dasar yang telah didefinisikan yaitu tipe bilangan bulat (*integer*), bilangan riil (*floating point*), logika (*boolean*), dan teks (*character/string*).

Tipe Bilangan Bulat (*Integer*)

Tipe data ini digunakan untuk data-data angka bilangan bulat (yang tidak mengandung angka di belakang koma), misalnya 21, 100, 1929, dan seterusnya.

Tipe Bilangan Riil (*Floating point*)

Tipe data ini digunakan untuk data-data angka bilangan riil atau pecahan (mengandung angka di belakang koma), misalnya 21.76, 100.567, 1929.156, dan seterusnya.

Tipe Logika (*Boolean*)

Berbeda dengan kedua jenis data di atas, tipe data *boolean* digunakan untuk merepresentasikan data-data yang hanya mengandung dua buah nilai, yaitu nilai *True* (yang direpresentasikan selain 0) dan nilai *false* (direpresentasikan dengan 0).

Teks (*Character*/*String*)

Tipe data ini merepresentasikan data-data yang berupa karakter. Tipe data ini dinyatakan dengan tipe **char**, sedangkan untuk string (=kumpulan karakter) dinyatakan sebagai pointer dari **char**, yang ditulis dengan **char***. Tipe data string juga dapat dideklarasikan dengan **string**, namun fungsi harus didefinisikan terlebih dahulu.

E. Statement Menampilkan Teks atau Nilai Konstanta dan Variabel

Pada bahasa C standar, untuk menampilkan nilai atau teks menggunakan perintah **printff**. Pada C++ dapat dilakukan menggunakan perintah **cout<<** (perintah *cout* diikuti dengan tanda „kurang dari" sebanyak dua kali) kemudian diikuti dengan string atau teks yang akan ditampilkan diapit dengan tanda kutip ganda.

Contoh :

```
cout << "Belajar C itu mudah";
cout << "Fakultas Teknologi Informasi & Elektro";
```

Jika yang akan ditampilkan adalah nilai dari suatu konstanta atau variabel, maka setelah perintah **cout<<** diikuti dengan nama konstanta atau nama variabelnya.

Contoh :

```
cout << luas; //menampilkan nilai konstanta atau variabel „luas"
cout << nama; //menampilkan nilai konstanta atau variabel „nama"
```

F. Statement Input

Suatu program yang dinamis membutuhkan peran aktif dari user untuk memasukkan nilai atau data ke dalam program melalui *keyboard*. Pengguna akan memasukkan data yang akan disimpan atau diolah, kemudian pihak lain atau bahkan pengguna sendiri yang akan membutuhkan informasi dari data yang disimpan atau diolah itu.

Struktur penulisan perintah untuk memasukkan data dari *keyboard* menggunakan perintah **cin>>** (perintah *cin* diikuti dengan tanda „lebih dari „ sebanyak dua kali) kemudian diikuti dengan nama konstanta atau nama variabel. Syarat yang harus dipenuhi untuk memasukkan data atau nilai ini, konstanta atau variabel harus dideklarasikan tipe datanya terlebih dahulu.

Contoh :

```
cin >> panjang; //memasukkan isi data variabel „panjang"
cin >> nama; //memasukkan isi data variabel „nama"
```

Praktek

1. Menampilkan Tulisan "Selamat datang"

Berikut ini adalah program sederhana untuk menampilkan tulisan pada monitor.

Nama file : Latih01.cpp

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

void main()
{
 cout<<"Selamat datang di Fakultas Teknologi Informasi & Elektro UTY;
}
```

Catatan: Untuk menjalankan program di atas, simpanlah terlebih dahulu dengan **<Alt><F>** pilih **save**, beri nama file. Untuk mengeksekusi lakukan dengan perintah **<ctrl><F9>** secara bersamaan.

2. Menampilkan Tulisan "Selamat datang"

Berikut ini adalah program untuk menampilkan tulisan pada monitor dengan perintah 3 kali **cout<<**, namun yang tampil pada monitor keseluruhan tulisan ditampilkan dalam 1 baris yang sama.

Nama file : Latih02.cpp

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

void main(){
 cout<<"Selamat datang di ";
 cout<<"Fakultas Teknologi Informasi dan Elektro ";
 cout<<"Universitas Teknologi Yogyakarta";
}
```

3. Menampilkan Tulisan "Selamat datang…" Pada baris berbeda

Berikut ini adalah program pengembangan dari program sebelumnya untuk menampilkan tulisan pada monitor dengan perintah 3 kali **cout<<**, namun ditampilkan pada baris yang berbeda karena sudah diberi perintah **<<endl** untuk memindahkan pada baris berikutnya.

Nama file : Latih03.cpp

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

void main(){
 cout<<"Selamat datang di ";<<endl;
  //<<endl berfungsi untuk memindahkan tampilan pada baris berikutnya
 cout<<"Fakultas Teknologi Informasi dan Elektro";<<endl;
 cout<<"Universitas Teknologi Yogyakarta";
}
```

4. Menampilkan Nilai Konstanta

Program di bawah ini untuk menampilkan isi nilai dari suatu konstanta. Pada bagian atas dideklarasikan preprosessor directive untuk file header dan mendefinisikan konstanta dengan nama bilangan1. Kemudian pada fungsi utama dideklarasikan dengan cara yang berbeda dengan nama konstanta bilangan2. Baris selanjutnya menampilkan isi konstanta variabel bilangan1 dan bilangan2. Perhatikan cara menampilkan isi dari suatu konstanta bilangan1 dan bilangan2 yang dilakukan dengan cara yang berbeda.

Nama file : Latih04.cpp

```
#include <stdio.h>
#include <conio.h>
#define bilangan1=10;

void main(){
 int bilangan2=5;
 cout<<"Berikut ini adalah isi bilangan 1 : ";
 cout<<bilangan1;<<endl;
 cout<<"Berikut ini adalah isi bilangan 2 : ";<<bilangan2;
}
```

5. Menghitung Perkalian 2 Bilangan

Program di bawah ini untuk menampilkan isi nilai dari suatu konstanta dimana bilangan1 dan bilangan2 telah ditentukan kemudian dikalikan dan ditampilkan pada baris berikutnya dengan variabel hasil. Perhatikan cara deklarasi suatu konstanta dan proses penghitungan hasil.

Nama file : Latih05.cpp

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

void main(){
 int bilangan1;
 int bilangan2;
 int hasil;

 bilangan1 = 5;
 bilangan2 = 3;
 hasil = bilangan1 * bilangan2;
 cout<<"Bilangan Pertama : ";<<bilangan1;
 cout<<"Bilangan Kedua   : ";<<bilangan2;
 cout<<"Hasil Perkalian  : ";<<hasil;
}
```

6. Memasukkan bilangan dan Menampilkan kembali

Program di bawah ini mulai menggunakan variabel. Salah satu bagian dari penggunaan variabel adalah untuk memasukkan nilai dari *keyboard* kemudian ditampilkan kembali. Perhatikan perintah untuk memasukkan data dan perintah untuk menampilkan kembali data yang diinputkan.

Nama file : Latih06.cpp

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

void main(){
 int bilangan;

 cout<<"Masukkan suatu bilangan : ";
 cin>>bilangan;<<endl;
 cout<<endl;
 cout"Bilangan yang ada masukkan adalah : ";<<bilangan;
}
```

7. Menghitung Luas Lingkaran

Program di bawah ini adalah untuk menghitung luas suatu lingkaran. Masukan dari program ini adalah jari-jari dengan tipe data integer, kemudian phi dideklarasikan dengan konstanta 3.14. Luas lingkaran dihitung dengan rumus :
Luas=phi * r^2.
Hasil perhitungan luas kemudian ditampilkan ke monitor.

Nama file : Latih07.cpp

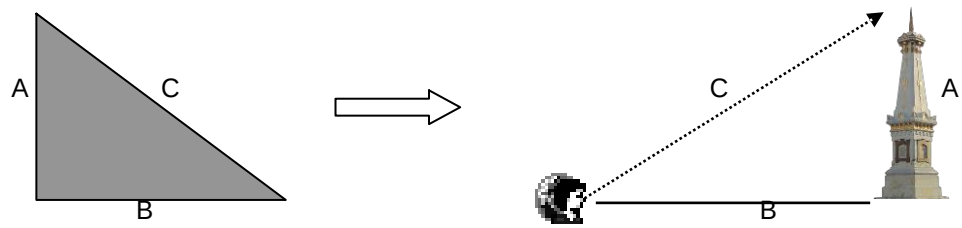
```
#include <stdio.h>
#include <conio.h>

void main()
{
 int jari;
 long int luas;
 double phi=3.14;

 cout<<"Masukkan jari-jari : ";
 cin>>jari;
 luas=phi*jari*jari;
 cout<<endl;
 cout"Luas lingkarannya adalah : ";<<luas;
}
```

8. Menghitung Sisi Miring Segitiga Siku-siku

Program di bawah ini adalah untuk menghitung sisi miring dari suatu segitiga siku-siku. Masukan dari program ini adalah panjang sisi bagian A dan panjang sisi bagian B. Panjang sisi bagian C adalah akar dari kuadrat panjang sisi A ditambah dengan kuadrat panjang sisi bagian B. Teori ini dikenal dengan theorema phytagoras yang juga dapat diaplikasikan untuk menghitung tinggi suatu gedung.



$$C = \sqrt{(A^2 + B^2)}$$

Nama file : Latih08.cpp

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

void main()
{
 int sisiA;
 int sisiB;
 double sisiC;

 cout<<"Masukkan Sisi A : ";
 cin>>sisiA;
 cout<<"Masukkan Sisi B : ";
 cin>>sisiB;
 sisiC=sqrt(sqr(sisiA)+sqr(sisiB));
 cout<<endl;
 cout"Sisi C adalah : ";<<sisiC;
}
```

Tugas Mandiri

1. Buatlah program untuk menampilkan teks berikut ini :

```
Daftar Program Studi
Fakultas Teknologi Informasi & Elektro
Universitas Teknologi Yogyakarta

1. D3-Sistem Informasi
2. S1-Sistem Informasi
3. S1-Informatika
4. S1-Teknik Komputer
5. S1-Teknik Elektro
```

2. Buatlah program untuk memasukkan data kemudian menampilkan kembali ke dalam bentuk lain. Gunakan fungsi **clrscr();** untuk membersihkan layar.

   Bentuk *input*:

```
Masukkan data :
Tanggal  : 01
Bulan    : 01
Tahun    : 1999
```

   Bentuk *output*:

```
Data yang Anda masukkan:
01-01-1999
```

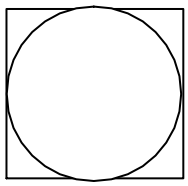
3. Buatlah program untuk menghitung luas segitiga. Masukan dari program ini adalah panjang alas dan tinggi. Kemudian luasnya dihitung dengan rumus 0.5 alas dikalikan tinggi.

4. Buatlah program untuk menghitung luas dan volume bola. Input yang dibutuhkan adalah jari-jari dan output yang diperlukan adalah volume dan luas permukaan bola. Adapun rumus untuk mencari volume dan luas permukaan bola adalah sebagai berikut :

   Volume = 4/3 * phi * jari-jari^3

   Luas permukaan = 4 * phi * jari-jari^2

5. Buatlah program untuk menyelesaikan permasalahan berikut ini:



   Jika sisi bujur sangkar tersebut adalah 4 centimeter, berapa luas bidang yang diarsir.

BAB II
TIPE DATA DAN OPERATOR

Kompetensi dan Indikator

Kompetensi Dasar : - Mengetahui tipe data dan operator pada C++.
- Mengetahui perbedaan penggunaan tipe dan dan operator.

Indikator Keberhasilan : - Mampu menjelaskan tipe data dan operator.
- Mampu menggunakan tipe data dan operator sesuai dengan kebutuhannya.

Dasar Teori

A. Pengantar

Sebuah program dibangun untuk memecahkan permasalahan yang kompleks dan terkadang dituntut untuk mendefinisikan operasi-operasi di dalamnya, baik berupa operasi perhitungan, perbandingan maupun operasi lainnya. Permasalahan yang kompleks membutuhkan ketepatan memilih jenis-jenis data dan operator yang mendukung kecepatan, ketepatan dan kehandalan program.

B. Tipe Data

Tipe data berfungsi untuk merepresentasikan jenis dari sebuah nilai yang terdapat dalam program. Kesalahan dalam menyebutkan tipe data program yang dibuat tidak dapat dijalankan atau seandainya jalan akan menghasilkan nilai yang tidak akurat. Sehingga tipe data harus digunakan sesuai dengan kebutuhan program.

1. Tipe Data Dasar

Dalam C++ terdapat beberapa tipe data dasar yang telah didefinisikan, yaitu yang digolongkan ke dalam tipe bilangan bulat *(integer)*, bilangan riil *(floating-point*), tipe logika *(boolean)*, tipe karakter/teks *(character/string)*. Tipe-tipe tersebut adalah tipe yang siap digunakan tanpa adanya proses manipulasi terlebih dahulu.

a. Tipe Bilangan Bulat

Tipe data ini digunakan untuk data-data angka yang tidak mengandung angka di belakang koma, misalnya 21, 3, 78 dan sebagainya. Tipe data yang termasuk ke dalam kategori ini adalah seperti yang terlihat pada tabel di bawah ini.

Tipe Data	Ukuran (bit)	Rentang
Int	16 atau 32	-32.768 sampai 32.767
Unsigned int	16 atau 32	0 - 65.535
Signed int	16 atau 32	Sama seperti **int**
Short int	16	-32.768 sampai 32.767
Unsigned short int	16	0 sampai 65.535
Signed short int	16	Sama seperti **short int**
Long int	32	-2.147.483.648 sampai 2.147.483.647
Signed long int	32	Sama seperti **long int**
Unsigned long int	32	0 sampai 4.294.967.295

b. Tipe Bilangan Riil

Tipe ini adalah tipe yang merepesentasikan data-data bilangan yang mengandung angka di belakang koma, misalnya 3.78, 21.03, dan banyak lagi yang lainnya. Adapun tipe data yang termasuk ke dalam kategori ini adalah seperti yang ditunjukkan pada tabel di bawah ini.

Tipe Data	Ukuran (bit)	Rentang	Presisi
float	32	1.2E-38 sampai 3.4E+38	6 digit presisi
double	64	2.3E-308 sampai 1.7E+308	15 digit presisi
Long double	80	3.4E-4932 to 1.1E+4932	19 digit presisi

c. Tipe Logika

Tipe ini merepresentasikan data-data yang hanya mengandung dua buah nilai, yaitu nilai logika *(boolean)*. Nilai logika itu sendiri hanya terdiri dari nilai benar (direpresentasikan dengan nilai selain nol, biasanya nilai **1**) dan salah (direpresentasikan dengan nilai **0**). Untuk sebagian kompiler C++ tertentu yang telah memenuhi standar ANSI/ISO, tipe ini telah dinyatakan dalam tipe bool. Dalam pemrograman, nilai ini umumnya lebih dikenal dengan nilai true (benar) dan false (salah).

d. Tipe Karakter/String

Tipe ini merepresentasikan data-data yang berupa karakter. Tipe data ini dinyatakan dengan tipe char, sedangkan untuk string (=kumpulan karakter) dinyatakan sebagai pointer dari tipe char, yaitu yang dituliskan dengan char*.

Tipe Data	Ukuran (bit)	Rentang
char	8	-128 sampai 127 atau 0 sampai 255
Unsigned char	8	0 sampai 255
Signed char	8	-128 sampai 127

C. Operator

Untuk memecahkan masalah-masalah tertentu tersebut dibutuhkan operator-operator yang sesuai. Beberapa pengertian penting dalam pembahasan operator antara lain :

```
C = 5+7
```

```
C           disebut dengan variabel
=           disebut dengan operator assignment
5 dan 7     disebut dengan operand
5+7         disebut dengan ekspresi
+           disebut dengan operator aritmetika (penambahan)
C=5+7       disebut dengan statemen aritmetika
```

Dalam bahasa C++ operator dikelompokkan ke dalam 4 bagian, yaitu operator assignment, unary, binary dan ternary.

1. Operator Assignment

Operator *assignment* adalah operator yang berfungsi untuk memasukkan *(assign)* nilai ke dalam suatu variabel ataupun konstanta. Operator ini dilambangkan dengan tanda **sama dengan** (=).

Contoh :

```
luas = panjang * lebar;
diskon = 0.01 * total_belanja;
```

2. Operator Unary

Dalam ilmu matematika yang disebut dengan operator *unary* adalah operator yang hanya melibatkan sebuah *operand*. Adapun yang termasuk ke dalam operator unary adalah seperti tampak pada tabel di bawah ini:

Operator	Jenis Operasi	Contoh
+	Membuat nilai positif	+7
-	Membuat nilai negatif	-7
++	Increment	C++
--	Decrement	C--

Increment

Increment adalah suatu penambahan nilai yang terjadi pada sebuah variabel. Adapun operator yang digunakan untuk melakukan *increment* adalah operator ++. Operator ini akan menambahkan nilai dari suatu variabel dengan standar nilai 1. Namun dalam C++ model ini dapat digunakan untuk dinamis, dimana increment dapat dirubah ke bentuk loncatan-loncatan lebih dari 1.

Terdapat dua buah jenis *increment* yang terdapat dalam bahasa C++, yaitu *pre-increment* dan *post-increment*. Arti dari *pre-increment* adalah melakukan penambahan nilai sebelum suatu variabel itu diproses, sedangkan post-increment merupakan kebalikannya, yaitu melakukan proses terlebih dahulu sebelum dilakukan penambahan nilai.

Decrement

Decrement merupakan kebalikan dari proses *increment*, yaitu menurunkan (mengurangi) nilai dari suatu variabel. Sama juga seperti pada *increment*, decrement juga dibagi ke dalam dua jenis yaitu *pre-decrement* dan *post-decrement*. Berikut ini program yang menunjukkan penggunaan *decrement.*

3. **Operator Binary**

Operator *binary* adalah operator yang digunakan dalam operasi yang melibatkan dua buah *operand*. Dalam bahasa C++, operator *binary* ini dikelompokkan lagi ke dalam empat jenis, yaitu operator *aritmetika, logika, relasional* dan *bitwise.*

a. Operator Aritmatika

Operator aritmatika adalah operator yang digunakan untuk melakukan operasi-operasi aritmatika seperti penjumlahan, pengurangan, perkalian, pembagian dan sebagainya. Adapun yang termasuk dalam operator aritmatika di dalam C++ adalah sebagai berikut :

Operator	Jenis Operasi	Contoh
+	Penjumlahan	2+3=5
-	Pengurangan	5-3=2
*	Perkalian	2*3=6
/	Pembagian	10.0/3.0=3.3333
%	Sisa bagi (modulus)	10%3=1

b. Operator Logika

Operator logika adalah operator yang digunakan untuk melakukan oerasi dimana nilai yang dihasilkan dari operasi tersebut hanya berupa nilai benar (*true*) dan salah (*false*). Nilai ini disebut dengan nilai boolean (ditemukan oleh matematikawan Inggris bernama George Bool).

Operator	Jenis Operasi	Contoh
&&	AND (dan)	1 && 1 = 1
||	OR (atau)	1 || 0 = 1
!	NOT (negasi)	!0 = 1

c. Operator Relasional

Operator relasional adalah operator yang digunakan untuk menentukan relasi atau hubungan dari dua buah *operand*. Operator ini ditempatkan di dalam sebuah ekspresi, yang kemudian akan menentukan benar atau tidaknya sebuah ekspresi.

Operator	Jenis Operasi	Contoh
>	Lebih besar	(5>2)=1
<	Lebih kecil	(5<2)=0
>=	Lebih besar atau sama dengan	(5>=5)=1
<=	Lebih kecil atau sama dengan	(5<=2)=0
==	Sama dengan	(5==2)=0
!=	Tidak sama dengan	(5!=2)=1

d. Operator Bitwise

Operator *bitwise* berguna untuk melakukan operasi-operasi yang berhubungandengan pemanipulasian bit. Operator bitwise ini hanya dapat dilakukan pada operand yang bertipe char dan int saja karena ini berkoresponden dengan tipe byte atau word di dalam bit.

Operator	Jenis Operasi	Contoh
&	AND	1&0=0
|	OR	1|0=1
^	*Exclusive OR (XOR)*	1^1=0
~	NOT	~1=0
>>	Shift Right	5<<1=10
<<	*Shift Left*	10>>1=5

4. **Operator Ternary**

Operator ternary adalah operator yang digunakan dalam operasi yang melibatkan tiga buah operand. Adapun operator yang digunakan untuk menyatakannya adalah operator **?:**
Konsep yang mendasari operasi ini adalah suatu percabangan (pemilihan) yang didasarkan atas kondisi tertentu.

Struktur penulisan operator *ternary* :

```
Ekspresi_1 ? Ekspresi_2 : ekspresi_3;
```

Jika **ekspresi_1** bernilai benar, maka program akan mengerjakan **ekspresi_2**. Sedangkan jika **ekspresi_1** bernilai salah maka yang akan dieksekusi adalah **ekspresi_3.**

Contoh :

```
Y = (X < 0) ? –X : X ;
```

Y akan bernilai negatif (-X) jika X (pada ekspresi pertama) bernilai benar, dan Y akan bernilai positif jika ekpresi pertama (X<0) bernilai salah.

Operator ternary ini jarang digunakan pada program aplikasi karena ada perintah khusus untuk menyelesaikan contoh-contoh ekspresi tersebut dengan menggunakan perintah **if.**

Praktek

1. Koversi Suhu

Rumus konversi suhu dari Celcius ke Fahrenheit dan Reamur dapat ditulis sebagai berikut :
$t_F = 9/5\ t_C + 32$
$t_R = 4/5\ t_C$

Algoritma dari program konversi ini adalah sebagai berikut :

1. Masukkan derajat celcius
2. $t_F \leftarrow 9/5\ t_C + 32$
3. $t_R \leftarrow 4/5\ t_C$
4. Tampilkan t_F
5. Tampilkan t_R

Nama file : Latih09.cpp

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

void main()
{
 //deklarasi variabel
 double tc, tf, tr;

 //memasukkan data derajat celcius
 cout<<"Masukkan derajat celcius : "; cin>>tc;

 //konversi suhu dari celcius ke fahrenheit dan reamur
 tf = ((9/5) * tc) + 32;
 tr = (4/5) * tc;

 //menampilkan hasil konversi
 cout<<endl;
 cout<<"Hasil konversi"<<endl;
 cout<<"Derajat Fahrenheit : "<<tf<<endl;
 cout<<"Derajat Reamur     : "<<tr<<endl;
}
```

2. Konversi Satuan Panjang

Konversi satuan panjang dari yard, kaki dan inchi ke meter menggunakan standar berikut ini :

1 yard = 3 kaki = 0,9144 meter
1 kaki = 12 inchi = 30,48 centimeter = 0,3048 meter
1 inchi = 25,4 milimeter = 0,0254 meter

Algoritma dari program ini adalah sebagai berikut :

1. Masukkan satuan panjang meter
2. meter $\leftarrow$ 0.9144 * yard + 0.3048 * kaki + 0.0254 * inchi
3. Tampilkan meter

Nama file : Latih10.cpp

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

int main()
{
 //Deklarasi variabel
```

```
 double yard, kaki, inchi, meter;
 cout<<"Masukkan satuan yard  : "; cin>>yard;
 cout<<"Masukkan satuan kaki  : "; cin>>kaki;
 cout<<"Masukkan satuan inchi : "; cin>>inchi;

 //menghitung konversi
 meter = 0.9144 * yard + 0.3048 * kaki + 0.0254 * inchi;
 cout<<endl;

 //menampilkan hasilnya
 cout<<yard<<" Yard"<<kaki<<" Kaki"<<inchi<<" Inchi setara
 dengan "<<meter<<" meter";
 return 0; //mengembalikan nilai 0 sebab fungsi main bertipe int
}
```

3. Operator Assignment

Berikut ini adalah program untuk memahami konsep operator assignment (*assignment* adalah memberikan suatu nilai ke dalam variabel). Hampir semua program menggunakan operator ini.

Nama file : Latih11.cpp

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

int main()
{
 //Mendeklarasikan variabel
 char karakter;
 int bulat;
 double pecahan;

 //Melakukan assignment pada variabel-variabel
 karakter = „A";
 bulat = 28;
 pecahan = 3.14;

 //Menampilkan nilai dari variabel-variabel
 cout<<"Isi karakter adalah : "<<karakter<<endl;
 cout<<"Isi bulat adalah    : "<<bulat<<endl;
 cout<<"Isi pecahan adalah  : "<<pecahan<<endl;
 return 0;
}
```

4. Operator Unary

Berikut ini adalah contoh program untuk memahami kegunaan operator unary pada aplikasi program bersifat *increment* (naik). Perhatikan perbedaan operasi ++A dan A++ pada saat program berjalan. Pada ++A nilai A akan dinaikkan satu sebelum ditampilkan dan A++ dinaikan setelah proses A++.

Nama file : Latih12.cpp

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

void main()
{
 int A; //deklarasi variabel
 A = -5; //assignment nilai -5 (negatif) ke variabel A
```

```
 A = -A; //merubah nilai -5 menjadi –(-5) = 5 (positif)

 //melakukan pre-increment
 cout<<"Nilai A awal  : "<<A<<endl;
 cout<<"Nilai ++A     : "<<++A<<endl;
 cout<<"Nilai A akhir : "<<A<<endl;

 A=10; //merubah nilai variabel A menjadi 10

 //melakukan post-increment
 cout<<"Nilai A awal  : "<<A<<endl;
 cout<<"Nilai A++     : "<<A++<<endl;
 cout<<"Nilai A akhir : "<<A<<endl;
}
```

5. Operator Unary

Berikut ini adalah contoh program untuk memahami kegunaan operator unary pada aplikasi program bersifat *decrement* (turun).

Nama file : Latih13.cpp

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

void main()
{
 int A; //deklarasi variabel
 A = 5; //assignment nilai 5 ke variabel A

 //melakukan pre-decrement
 cout<<"Nilai A awal  : "<<A<<endl;
 cout<<"Nilai --A     : "<<--A<<endl;
 cout<<"Nilai A akhir : "<<A<<endl;

 A=10; //merubah nilai variabel A menjadi 10

 //melakukan post-decrement
 cout<<"Nilai A awal  : "<<A<<endl;
 cout<<"Nilai A--     : "<<A--<<endl;
 cout<<"Nilai A akhir : "<<A<<endl;
}
```

6. Operator Binary

Program berikut ini memberikan gambaran penggunaan operator binary yang terdiri atas operator aritmatika penjumlahan, pengurangan, perkalian dan pembagian untuk bilangan bulat.

Nama file : Latih14.cpp

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

void main()
{
 int A, B; //deklarasi variabel

 //input bilangan pertama ke variabel A dan bilangan kedua ke B
 cout<<"Masukkan bilangan pertama : "; cin>>A;
 cout<<"Masukkan bilangan kedua   : "; cin>>B;
```

```
 //Proses perhitungan dan menampilkan
 cout<<"Penjumlahan bilangan pertama dan kedua : "<<A+B<<endl;
 cout<<"Pengurangan bilangan pertama dan kedua : "<<A-B<<endl;
 cout<<"Perkalian bilangan pertama dan kedua   : "<<A*B<<endl;
 cout<<"Pembagian bilangan pertama dan kedua   : "<<A/B<<endl;
}
```

Catatan :
Masukkan bilangan pertama lebih besar dari bilangan kedua untuk mendapatkan hasil bilangan pada proses A/B, karena A/B merupakan pembagian bilangan bulat, misalnya 10/3 akan menghasilkan nilai 3.

7. Sisa Bagi Bilangan Bulat (Modulo)

Dalam bahasa C++, sisa bagi hasil dilambangkan dengan operator %. Misalnya 7%3=1 (7 dibagi 3 dapat 2 sisa 1). Dalam bahasa Pascal menggunakan perintah **Mod,** misalnya 7 mod 3=1.

Berikut ini adalah contoh program penggunaan operator % untuk menghitung konversi dari detik ke jam, menit, dan detik. Masukan dari program ini adalah waktu dalam satuan detik, kemudian akan dikonversikan ke dalam satuan jam, menit dan detik. Misalnya jika dimasukkan 3700 detik maka akan tampil 1 jam 1 menit 40 detik.

Algoritma dari proses konversi ini adalah :

1. Masukkan satuan waktu detik
2. Jam ← detik dibagi 3600, karena 1 jam = 3600 detik. Hasil perhitungan ini adalah bilangan bulat.
3. sisajam ← detik modulo 3600, akan menghasilkan sisa dari perhitungan langkah 3.
4. menit ← sisajam dibagi 60;
5. sisamenit ← sisajam modulo 60;
6. detik ← sisamenit, karena sisa menit merupakan satuan terkecil yang dapat dikatakan sebagai detik.
7. Tampilkan jam, menit, detik.

Catatan :
1 jam = 60 menit = 3600 detik

Nama file : Latih15.cpp

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

int main()
{
 int detik, menit, jam, sisa; //deklarasi variabel

 //inputkan satuan waktu detik
 cout<<"Masukkan waktu dalam detik : ";
 cin>>detik;
 cout<<endl;

 //proses konvesi dari detik ke jam, menit dan detik
 jam = detik / 3600; //menghasilkan bilangan bulat
 sisajam = detik % 3600; //sisa bagi atau modulo
 //menit adalah sisa dari perhitungan di atas
 menit = sisajam / 60;
 //detik adalah sisa dari perhitungan di atas
```

```
 detik = sisajam % 60;

//menampilkan hasil operasi
 cout<<"Waktu tersebut setara dengan "<<jam<<" jam "<<menit<"
 menit "<<detik<<" detik";
 return 0;
}
```

8. Operator Logika

Berikut ini adalah contoh penggunaan operator logika AND (&&), OR (||) dan NOT (!)

Nama file : Latih16.cpp

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

int main()
{
 cout<<"1 && 1 = "<<(1&&1)<<endl;
 cout<<"1 && 0 = "<<(1&&0)<<endl;
 cout<<"0 && 1 = "<<(0&&1)<<endl;
 cout<<"0 && 0 = "<<(0&&0)<<endl;
 cout<<"1 || 1 = "<<(1||1)<<endl;
 cout<<"1 || 0 = "<<(1||0)<<endl;
 cout<<"0 || 1 = "<<(0||1)<<endl;
 cout<<"0 || 0 = "<<(0||0)<<endl;
 cout<<"!1 = "<<(!1)<<endl;
 cout<<"!0 = "<<(!0)<<endl;
 return 0;
}
```

9. Operator Bitwise

Berikut ini adalah contoh penggunaan operator bitwise (berfungsi untuk mengoperasikan suatu bilangan dalam bentuk biner, sehingga setiap operasi harus dikonversi ke dalam bilangan biner terlebih dahulu). Dalam program ini meliputi operator XOR (^), Shift Left/ menggeser bit ke kiri (<<) dan Shift Right / menggeser bit ke kanan (>>).

Nama file : Latih17.cpp

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

int main()
{
 cout<<"1 ^ 0   = "<<(1^0)<<endl;
 cout<<"19 ^ 4  = "<<(15^4)<<endl;
 cout<<"19 << 1 = "<<(19<<1)<<endl;
 cout<<"19 << 2 = "<<(19<<2)<<endl;
 cout<<"19 << 3 = "<<(19<<3)<<endl;
 cout<<"19 >> 1 = "<<(19>>1)<<endl;
 cout<<"19 >> 2 = "<<(19>>2)<<endl;
 cout<<"19 >> 3 = "<<(19>>3)<<endl;
 return 0;
}
```

Hasil dari perintah 15^4 di atas dapat diilustrasikan berikut:

0 0 0 1 0 0 1 1 19 dalam bentuk biner
0 0 0 0 0 1 0 0 4 dalam bentuk biner
---------- **XOR (**exsclusif OR) atau yang benar-benar berbeda.
0 0 0 1 0 1 1 1 23 dalam bentuk biner.

Kemudian untuk perintah 19<<1 dan berikutnya dapat diilustrasikan dalam tabel berikut ini :

Perintah	Operasi	Hasil
19	0 0 0 1 0 0 1 1	
19 << 1	0 0 **1 0 0 1 1** 0	38
19 << 2	0 **1 0 0 1 1** 0 0	88
19 << 3	**1 0 0 1 1** 0 0 0	152
19	0 0 0 1 0 0 1 1	
19 >> 1	0 **0 0 0 1 0 0 1**	9
19 >> 2	0 0 **0 0 0 1 0 0**	4
19 >> 3	0 0 0 **0 0 0 1 0**	2

Tugas Mandiri

1. Perhatikan program **latih10.**cpp. Buatlah program untuk mengkonversi satuan panjang dari meter ke yard, kaki dan inchi. Masukan dari program ini adalah panjang dalam satuan meter. Keluarannya adalah panjang dalam satuan yard, kaki dan inchi.

   Catatan standar panjang dapat dilihat pada latihan soal nomor 2.

2. Buatlah program untuk menghitung selisih dari dua waktu yang mempunyai format jam:menit:detik. Sebagai contoh, jika pengguna memasukkan 03:00:00 dan 04:30:15, maka program akan menampilkan selisihnya adalah 01:30:15.

   Masukan dari program ini adalah enam bilangan bulat yang masing-masing melambangkan pasangan jam, menit dan detik. Keluarannya juga terdiri atas enam bilangan bulat dengan format jam, menit dan detik selisih dari dua waktu tersebut.

3. Buatlah program untuk menentukan berat badan ideal seseorang jika diketahui tinggi badannya. Sebagai contoh jika tinggi badannya 180 centimeter maka berat badan yang ideal adalah 72 kilogram. Karena berat badan ideal dihitung dari tinggi badan dikurangi dengan 100 kemudian dikurangi lagi dengan 10% dari sisanya. Jadi 180 cm dikurangi 100 cm menjadi 80 cm. Dari 80 cm dikurangi 10% (8 cm) menjadi 72.

   Catatan:

   Berat badan ideal = tinggi badan – 100 – (0.1*(tinggi badan-100))

4. Buatlah program untuk mengkonversi dari bilangan desimal ke bilangan biner. Dimana masukan dari program ini adalah bilangan desimal dan keluarannya adalah bilangan binner. Adapun proses perubahannya adalah membagi bilangan desimal itu terus menerus dengan bilangan 2 dan bilangan biner adalah sisa dari pembagian itu. Misalnya bilangan desimal 19 dikonversi dalam bilangan biner adalah 10011 (jika standard 8 bit menjadi 0001 0011)

   Didapatkan dari :

   19
   --- sisa **1** (*19 dibagi 2 dapat 8 sisa 1*)
   9
   --- sisa **1** (*9 dibagi 2 dapat 4 sisa 1*)
   4
   --- sisa **0** (*4 dibagi 2 dapat 2 sisa 0*)
   2
   --- sisa **0** (*2 dibagi 2 dapat 1 sisa 0*)
   1
   Sehingga, nilai binernya dibaca dari bawah **10011**. Jika dirubah dalam standar 8 bit ditambah angka 0 di depannya sehingga semuanya berjumlah 8 angka, contoh **0001 0011**.

BAB III
MANIPULASI STRING

Kompetensi dan Indikator

Kompetensi Dasar : - Mengetahui jenis data berbentuk string.
- Mengetahui model manipulasi string.
- Mengetahui penggunaan string untuk aplikasi.

Indikator Keberhasilan : - Mampu menjelaskan dasar-dasar dan bentuk string.
- Mampu menggunakan manipulasi string untuk beberapa aplikasi.

Dasar Teori

String sangat memudahkan tugas pemrogram. Dengan menggunakan string, pemrogram dapat menampilkan pesan kesalahan, menampilkan prompt bagi masukan keyboard ataupun memberikan informasi pada layar dengan mudah.

Seperti halnya tipe data yang lain, string dapat berupa konstanta atau variabel. Konstanta string sudah biasa disertakan pada program. Misalnya pada pernyataan :

```
cout << “C++”<<endl;
```

Konstanta String

Suatu konstanta string ditulis dengan awalan dan akhiran tanda petik ganda (“_“), misalnya: “C++”. Konstanta string seperti diatas disimpan dalam memori secara berurutan. Setiap karakter menempati memory 1 byte. Setelah karakter yang terakhir terdapat karakter Null (karakter dengan nilai ASCII sama dengan nol atau disimbolkan dengan “\0”, yaitu tanda \ dikuti dengan nol).

Bila suatu string hanya berisi karakter NULL, string disebut sebagai string kosong.

Variabel String

Variabel string adalah variabel yang dipakai untuk menyimpan string. Misalnya:

```
char teks[10];
```

merupakan pernyataan untuk mendefinisikan variabel string dengan panjang maksimal 9 karakter(sudah termasuk karakter *NULL*). Perlu diketahui, pernyataan di atas tidak lain untuk mendefinisikan array bertipe karakter.

Praktek

Memasukan Data String

Setelah suatu variabel string didefinisikan, Anda bisa mengisikan data ke variabel tersebut. Pemasukan data dapat ditangani oleh data cin, seperti contoh program di bawah ini :

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

void main()
{
        char teks[13];  //string dengan panjang maksimal 12 karakter
        clrscr(); //hapus layar
        cout<<"Masukan sebuah kata :"<<endl;
        cin>> teks;
        cout<< "Kata yang tercetak    :"<<teks;
        getch();
}
```

Yang perlu diperhatikan adalah bahwa cin hanya dapat membaca sebuah kata. Artinya karakter-karakter yang terletak sesudah spasi tidak bisa ditampung pada teks. Ini disebabkan operator << pada cin hanya bisa membaca masukan hingga terdapat spasi, tab atau enter. Untuk menampilkan agar dapat terbaca solusinya adalah menambahkan fungsi **get()** pada objek cin (**cin.get()**) bisa dipakai untuk keperluan ini. Sebagai contoh seperti program dibawah ini :

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

void main()
{
        char teks[13];  //string dengan panjang maksimal 12 karakter
        clrscr(); //hapus layar
        cout<<"Masukan sebuah kata :"<<endl;
        cin.get(teks, 13);
        cout<< "Kata yang tercetak    :"<<teks;
        getch();
}
```

Karakter yang terletak sesudah spasi juga ikut disimpan pada teks. Tetapi jika data yang dimasukan lebih dari 13 karakter maka hanya 12 karakter pertama yang disimpan pada teks, mengingat argumen kedua dari fungsi **get()** diisi dengan 13 (satu karakter berisi Null).

Pada contoh program di atas dapat ditulis sebagai berikut :

cin.get(teks, 13)

Bisa juga ditulis sebagai berikut :

cin. Get(teks, sizeof(teks));

- **Fungsi getline()**

Suatu masalah akan timbul kalau cin.get() digunakan 2 kali seperti contoh program dibawah ini :

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

void main()
{
        char nama[25];
        char alamat[35];
        clrscr(); //hapus layar

        cout<<"Masukan Nama :"<<endl;
        cin.get(nama, sizeof(nama));
        cout<< "Masukan Alamat    :"<<endl;
        cin.get(alamat,sizeof(alamat));

        cout<<"Nama:"<<nama<<endl;
        cout<<"Alamat:"<<alamat<<endl;
        getch();
}
```

Pada contoh di atas **cin.get()** pertama digunakan untuk membaca nama yang kedua untuk membaca alamat. Ternyata program tidak memberikan kesempatan untuk mengisi alamat. Hal ini terjadi karena **get()** yang pertama tidak membuang kode newline(\n). Oleh karena **get()** tidak mengabaikan spasi putih(spasi, tab , atau newline) maka **get()** kedua menjadi tidak berfungsi sebagaimana mestinya. Cara untuk menyelesaikan masalah di atas dengan menggunakan fungsi **getline()**, karena fungsi ini dapat membuang sisa data yang tidak dibaca, termasuk newline itu sendiri. Pada contoh berikut, **get()** diganti dengan **getline().** Perhatikan contoh berikut ini :

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

void main(){
        char nama[25];
        char alamat[35];
        clrscr(); //hapus layar

        cout<<"Masukan Nama :"<<endl;
        cin.getline(nama, sizeof(nama));
        cout<< "Masukan Alamat    :"<<endl;
        cin.getline(alamat,sizeof(alamat));

        cout<<"Nama:"<<nama<<endl;
        cout<<"Alamat:"<<alamat<<endl;
        getch();
}
```

Catatan : Dengan menggunakan **getline()**, data nama dan alamat dapat diisi.

- **Fungsi strcpy**

Bentuk dari dari **strcpy()** :

strcpy(string_target,string_awal)

Prototipe fungsi di atas ada pada file header **string.h**

Contoh program yang menggunakan **strcpy()**:

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
void main()
{
        char teks[]=”C++ Oke”;
        char data [25];
        clrscr(); //hapus layar
        strcpy(data,teks); // mengcopy isi teks ke data
        cout<<”Isi data :”<<data<<endl;
        getch();
}
```

- **Fungsi toupper dan tolower**

Fungsi **toupper()** berguna untuk memperoleh huruf kapital dari suatu huruf kecil. Nilai balik dari fungsi ini akan berupa seperti argumennya kalau argumen tidak berisi huruf kecil.
Adapun fungsi **tolower()** adalah kebalikan dari **toupper()**. Fungsi ini memberikan nilai balik :

- Berupa huruf kecil kalau argumen berisi huruf kapital
- Berupa nilai seperti argumen kalau argumen tidak berupa huruf kecil

Kedua fungsi di atas memberikan nilai balik bertipe **int** dan memiliki protipe pada file **ctype.h**
Contoh program :

```
#include<iostream.h>
#include<conio.h>
#include<ctype.h>
void main()
{
        char st[]=”saya suka C++”;
        clrscr(); //hapus layer
        cout << “Kalimat asalnya : “ << st << endl;
        for(int i=0; st[i];i++)
        st[i]= toupper(st[i]);
        cout<< “Setelah diubah ke huruf kapital : “ <<st<<endl;
        getch();
}
```

Pada contoh di atas, ***st[i]= toupper(st[i]);*** menyebabkan setiap huruf kecil pada variabel st akan diganti dengan huruf kapital.

- **Fungsi strlen()**

Panjang suatu string dapat diketahui dengan mudah menggunakan fungsi **strlen()**. Misalnya saja, didefinisikan :

```
char bunga[15]= “mawar”;
int panjang;
```

Maka pernyataan yang menggunakan strlen :

panjang = strlen(bunga);

akan memberikan panjang string yang tersimpan pada variabel bunga ke panjang.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>

void main()
{
        char bunga[15]="mawar";
        clrscr();

        cout<<"Panjang karakter kata '" <<bunga<<"' adalah "<<strlen(bunga)<<endl;
  getch();
}
```

- **Fungsi strlwr() dan strupr**

Jika isi semua huruf kapital pada suatu string dinginkan untuk diubah menjadi huruf kapital, hal ini dapat dilakukan melalui fungsi **strlwr()**. Misalnya, didefinisikan :

char st[]=”AbCdEfGeHjKl”;

Maka st akan berisi : “abcdefgehjki”;

Sedangkan fungsi **strupr()** kebalikan dari fungsi **strlwr().** Kedua fungsi di atas sama seperti tolower dan toupper.

Contoh program untuk memperlihatkan efek **strlwr()** dan **strupr()** :

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
void main(){
        char st[]=”AbCdEfGhIjKl”;
        clrscr(); //hapus layar
        cout<<”Isi St mula-mula:”<<st<<endl;
        strlwr(st);
        cout<<”Isi setelah dikonversi strlwr: “<<endl;
        cout<<st<<endl;
        strupr(st);
        cout<<”Isi setelah dikonversi strupr: “<<endl;
        cout<<st<<endl;
        getch();
}
```

- **Konversi String ke Angka dan sebaliknya**

Untuk melakukan konversi dari suatu string ke bilangan, dapat menggunakan sejumlah fungsi bawahan. Fungsi-fungsi yang tersedia dapat dilihat pada tabel.

Fungsi	Prototipe	Keterangan Konversi
atoi	Stdlib.h	String argumen menjadi nilai bertipe **int**
atof	Stdlib.h	String argumen menjadi nilai bertipe **float**
atol	Stdlib.h	String argumen menjadi nilai bertipe **long int**
atold	Stdlib.h	String argumen menjadi nilai bertipe **long doubel**

Adapun fungsi yang berguna untuk mengubah suatu nilai bilangan menjadi suatu string diantaranya ditunjukan pada tabel di bawah ini:

Fungsi	Prototipe	Keterangan Konversi
itoa	Stdlib.h	**int** argumen menjadi nilai bertipe string
ltoa	Stdlib.h	**long int** argumen menjadi nilai bertipe string
ultoa	Stdlib.h	**unsigned long** argumen menjadi nilai bertipe string

Ketiga fungsi yang tercantum pada tabel di atas mempunyai tiga buah argumen.

- ✓ Argumen pertama berupa nilai yang akan dikonversi ke string
- ✓ Argumen kedua berupa variabel penerima string hasil konversi
- ✓ Argumen ketiga berupa basis bilangan yang digunakan. Bisa diisi dengan nilai antara 2 sampai dengan 36.

Misalnya, apabila hasil didefinisikan sebagai berikut :

```
 char hasil[24];
int nilai= 2345;
```

Pernyataan :

itoa(nilai,hasil,10);

membuat hasil berisi “2345”; Perhatikan contoh berikut ini :

```
#include <iostream.h>
#include <stdlib.h>
#include <conio.h>

void main()
{
        char hasil[24];
        int nilai= 2345;

        cout<<itoa(nilai,hasil,10);
        getch();
}
```

BAB IV
PERCABANGAN (*SELECTION*)

Kompetensi dan Indikator

Kompetensi Dasar : - Mengetahui dasar-dasar percabangan.
- Mengetahui penulisan percabangan.
- Mengetahui pemilihan jenis percabangan yang tepat sesuai dengan kebutuhannya.

Indikator Keberhasilan : - Mampu menjelaskan dasar-dasar percabangan.
- Mampu menuliskan contoh program menggunakan percabangan.
- Mampu memilih jenis percabangan sesuai dengan kasus yang sedang dikerjakan.

Dasar Teori

A. Pengantar

Salah satu permasalahan yang pasti ditemui dalam pembuatan program adalah suatu percabangan. Percabangan adalah suatu pemilihan statemen yang akan dieksekusi dimana pemilihan tersebut didasarkan atas kondisi tertentu. Di dalam C++, terdapat dua buah jenis struktur yang digunakan untuk mengimplementasikan suatu percabangan, yaitu dengan menggunakan struktur **if** dan struktur **switch.**

Statemen-statemen yang terdapat dalam sebuah blok percabangan akan dieksekusi hanya jika kondisi yagn didefinisikan terpenuhi (bernilai benar). Artinya jika kondisi tidak terpenuhi (bernilai salah), maka statemen-statemen tersebut juga tidak ikut dieksekusi atau dengan kata lain akan diabaikan oleh kompiler.

B. Struktur Satu Kondisi

Struktur ini hanya melibatkan satu buah ekspresi yang akan diperiksa. Bentuk umum dari struktur percabangan yang memiliki satu kondisi adalah sebagai berikut:

```
// jika terdapat lebih dari satu statemen
if (kondisi)
{
   Statemen1;
   Statemen2;
   ....
}

//Jika hanya satu statemen, dapat ditulis seperti di bawah
if (kondisi) Statemen;
```

C. Struktur Dua Kondisi

Struktur percabangan jenis ini terdapat sebuah statemen khusus yang berguna untuk mengatasi kejadian apabila kondisi yang didefinisikan tersebut tidak terpenuhi (bernilai salah). Bentuk umum dari struktur percabangan jenis ini adalah sebagai berikut:

```
// jika terdapat lebih dari satu statemen
If (kondisi) {
   Statemen_jika_kondisi_terpenuhi;
} else {
   Statemen_jika_kondisi_tidak_terpenuhi;
}

// jika terdapat hanya satu statemen
If (kondisi)
   Statemen_jika_kondisi_terpenuhi;
else
   Statemen_jika_kondisi_tidak_terpenuhi;
```

D. Struktur Tiga Kondisi

Struktur jenis ini merupakan perluasan dari struktur yang memiliki dua kondisi di atas, yaitu dengan menyisipkan (menambahkan) satu atau lebih kondisi ke dalamnya. Bentuk umum dari struktur percabangan yang memiliki lebih dari dua kondisi adalah seperti yang tampak di bawah ini:

```
If (kondisi1)  {
     Statemen_jika_kondisi1_terpenuhi;
} else if (kondisi2)  {
     Statemen_jika_kondisi2_terpenuhi;
} else if (kondisi3)  {
     Statemen_jika_kondisi3_terpenuhi;
}
Else  {
     Statemen_jika_semua_kondisi_diatas_tidak_terpenuhi;
}
```

E. Pemilihan Menggunakan Kata Kunci switch

Bentuk umum dari struktur percabangan yang menggunakan kata kunci switch ini adalah sebagai berikut:

```
Switch (ekspresi)  {
     case nilai_konstan1      :  Statemen_statemen; break;
     case nilai_konstan2      :  Statemen_statemen; break;
     .....
     case nilai_konstanN      :  Statemen_statemen; break;
     default
        Statemen_statemen_alternatif;
```

Tipe data dari ekspresi di atas haruslah bilangan bulat atau karakter. Dalam bahasa C standar kita diizinkan untuk menuliskan 257 buah statemen case dalam sebuah struktur C++ mengizinkan 16.384 buah.

Praktek

1. Struktur If dengan Satu Kondisi Satu Statement

Berikut ini adalah program sederhana untuk melakukan eksekusi jika kondisi bernilai benar saja, jika kondisi bernilai salah tidak ada yang dijalankan. Dalam kasus ini jika nilai lebih besar dari 50 maka dinyatakan lulus, dan jika nilai yang dimasukkan kurang dari 50 tidak ada komentar apa-apa karena kondisi yang salah tidak ada kemungkinan pilihannya.

Nama file : Latih174.cpp

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

void main()
{
 int nilai;
 cout<<"Masukkan nilai : "; cin>>nilai;
 cout<<endl;
 if (nilai>50)
 {
  cout<<"Selamat, Anda lulus...";
 }
}
```

2. Struktur If dengan Satu Kondisi Dua Statement

Berikut ini adalah contoh penggunaan If pada satu kondisi dengan dua statement kemungkinan. Jika nilai lebih besar dari 50 maka dinyatakan lulus, tetapi jika kurang dari 50 maka dinyatakan gagal.

Nama file : Latih184.cpp

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

void main()
{
 int nilai;
 cout<<"Masukkan nilai : "; cin>>nilai;
 cout<<endl;
 if (nilai>50)
 {
  cout<<"Selamat, Anda lulus...";
 } else
 {
  cout<<"Maaf, Anda gagal...";
 }
}
```

3. Struktur If dengan Penambahan Operator AND (&&)

Berikut ini adalah contoh penggunaan If dengan penambahan operator pembanding AND (&&). Pada operator && kondisi bernilai benar jika kedua kondisi bernilai benar semua.

Pada program ini akan dimasukkan suatu angka, kemudian diseleksi dengan kondisi jika angka lebih besar dari 0 dan lebih kecil dari 10 (berarti kondisi yang bernilai benar adalah nilai dengan interval 1 sampai dengan 9).

Nama file : Latih194.cpp

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

void main() {
 int angka;

 cout<<"Masukkan angka : "; cin>>angka;
 cout<<endl;
 //seleksi pembanding AND, kedua kondisi harus bernilai benar
 if ((angka > 0) && (angka < 10)){
  cout<<"Anda memasukkan angka antara 1 sampai 9";
 } else {
  cout<<"Yang Anda masukkan bukan antara 1 sampai 9";
 }
}
```

Pada program di atas jika diinputkan angka 5, maka seleksi pertama adalah 5 > 0 bernilai **benar**, dan seleksi kedua adalah 5 < 10 bernilai **benar**.

4. Struktur If dengan Penambahan Operator OR (||)

Berikut ini adalah contoh penggunaan If dengan penambahan operator pembanding OR (||). Pada operator || kondisi bernilai benar jika salah satu kondisi bernilai benar.

Pada program ini akan dimasukkan suatu huruf, kemudian diseleksi dengan kondisi jika huruf yang dimasukkan adalah a, i, u, e atau o maka seleksi akan memilih bahwa input merupakan huruf vokal, jika tidak maka huruf konsonan.

Nama file : Latih204.cpp

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

void main() {
 int huruf;

 cout<<"Masukkan huruf kecil : "; cin>>huruf;
 cout<<endl;
 //seleksi pembanding OR, salah satu kondisi bernilai benar
 if ((huruf == "a") || (huruf == "i") ||
    (huruf == "u") || (huruf == "e") ||
    (huruf == "o")){
  cout<<"Anda memasukkan huruf vokal";
 } else {
  cout<<"Yang Anda masukkan adalah huruf konsonan";
 }
}
```

Pada program di atas jika diinputkan huruf a, i, u, e, o, maka seleksi akan melihat apakah diantara lima kondisi itu ada yang bernilai benar? Jika ada yang bernilai benar salah satunya maka kesimpulannya adalah huruf vokal, jika tidak maka kesimpulannya adalah huruf konsonan.

5. Struktur If dengan Dua Kondisi untuk menentukan Bilangan nol, positif atau negatif

Berikut ini adalah contoh penggunaan If dengan dua kondisi dan tiga kemungkinan statement yang akan dipilih jika kondisinya bernilai benar.
Dalam menentukan suatu bilangan dikategorikan bilangan nol, genap atau ganjil menggunakan algoritma berikut ini :

1. Masukkan suatu bilangan
2. Jika bilangan = 0 maka keterangan ← bilangan nol,
   Jika bilangan > 1 maka keterangan ← bilangan positif,
   Jika tidak keduanya maka keterangan ← bilangan negatif,
3. Tampilkan keterangan

Nama file : Latih214.cpp

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

void main() {
 int bilangan;

 cout<<"Masukkan bilangan : "; cin>>bilangan;
 cout<<endl;
 if (bilangan == 0){
  cout<<"Bilangan nol";
 } if (bilangan > 0){
  cout<<"Bilangan Positif";
 } else {
  cout<<"Bilangan negatif";
 }
}
```

6. Menentukan Akar-akar persamaan kuadrat

Untuk menentukan akar-akar persamaan kuadrat pada umumnya menggunakan rumus ABC. Bentuk umum persamaan kuadrat adalah **$y=ax^2+bx+c$** dan untuk menentukan akar-akar dari persamaan tersebut sebelumnya kita harus melakukan perhitungan koefisien untuk mendapatkan nilai determinan (D) dengan menggunakan rumus **$d=b^2-4ac$**.

Setelah nilai determinan diketahui, terdapat tiga kemungkinan yang dapat menentukan akar-akar persamaan tersebut yaitu :

a. Jika D>0, maka x_1 dan x_2 bersifat riil dan berbeda. Rumus untuk penentuan nilainya adalah :

   $x_1 = (-b + \sqrt{D}) / 2a$

   $x_2 = (-b - \sqrt{D}) / 2a$

b. Jika D=0, maka x_1 dan x_2 bersifat riil dan sama

c. Jika D<0, maka x_1 dan x_2 bersifat imaginer.

Nama file : Latih224.cpp

```
#include <stdio.h>
#include <conio.h>
#include <cmath.h>

int main() {
 int a, b, c;
 float D, x1, x2;
 int flag;

 //meminta user memasukkan koefisien persamaan kuadrat
 cout<<"Masukkan nilai a : "; cin>>a;
 cout<<"Masukkan nilai b : "; cin>>b;
 cout<<"Masukkan nilai c : "; cin>>c;
```

```
//menghitung nilai determinan
D = (b*b) – (4*a*c);

//menentukan akar-akar persamaan kuadrat
if (D > 0){
  x1 = ((-b) + sqrt(D)) / (2 * a);
  x2 = ((-b) - sqrt(D)) / (2 * a);
} if (D == 0){
 x1 = ((-b) + sqrt(D)) / (2 * a);
 x2 = x1;
} else {
 cout<<”Akar-akar persamaan bersifat imaginer”;
 }
}
```

7. Menentukan Nilai Huruf dari Nilai Akhir

Program berikut merupakan contoh untuk seleksi dengan jumlah kondisi yang banyak dan menggunakan interval.

Nilai huruf dari nilai akhir dihitung dari prosentase tertentu yaitu 35% nilai Ujian Tengah Semester (UAS), 45% nilai Ujian Akhir Semester (UAS) dan 20% nilai tugas yang diinputkan oleh user. Berdasarkan perhitungan nilai akhir dilakukan seleksi untuk menentukan nilai huruf sebagai berikut :

A : nilai akhir >=85
B : 70 <= nilai akhir < 85
C : 55 <= nilai akhir < 70
D : 40 <= nilai akhir < 55
E : nilai akhir < 40

Nama file : Latih234a.cpp

```
#include <stdio.h>
#include <conio.h>
#include <cmath.h>

int main() {
 float uts, uas, tugas, nilaiakhir;
 char nilaihuruf;

 //meminta user memasukkan nilai uts, uas, dan tugas
 cout<<”Masukkan nilai UTS   : “; cin>>uts;
 cout<<”Masukkan nilai UAS   : “; cin>>uas;
 cout<<”Masukkan nilai Tugas : “; cin>>tugas;

 //menghitung nilai akhir
 nilaiakhir = ((0.35*uts)+(0.45*uas)+(0.2*tugas))/3;

 //menentukan nilai huruf
 if (nilaiakhir >= 85){
  nilaihuruf=”A”;
  } if (nilaiakhir >= 70){
  nilaihuruf=”B”;
  } if (nilaiakhir >= 55){
  nilaihuruf=”C”;
  } if (nilaiakhir >= 40){
  nilaihuruf=”D”;
  } else {
  nilaihuruf=”E”;
```

```
 }

 //menampilkan nilai angka dan nilai huruf
 cout<<"Nilai akhir angka : "<<nilaiangka<<endl;
 cout<<"Nilai akhir huruf : "<<nilaihuruf;
}
```

Dari program di atas, seleksi **if** dapat ditulis dengan menggunakan perintah **switch**. Berikut ini adalah contoh program menggunakan perintah itu.

Nama file : Latih234b.cpp

```
#include <stdio.h>
#include <conio.h>
#include <cmath.h>

int main() {
 float uts, uas, tugas, nilaiakhir;
 char nilaihuruf;

 //meminta user memasukkan nilai uts, uas, dan tugas
 cout<<"Masukkan nilai UTS   : "; cin>>uts;
 cout<<"Masukkan nilai UAS   : "; cin>>uas;
 cout<<"Masukkan nilai Tugas : "; cin>>tugas;

 //menghitung nilai akhir
 nilaiakhir = ((0.35*uts)+(0.45*uas)+(0.2*tugas))/3;

 //menentukan nilai huruf
 switch (nilaiakhir) {
  case 85..100 : nilaihuruf="A"; break;
  case 70..84  : nilaihuruf="B"; break;
  case 55..69  : nilaihuruf="C"; break;
  case 40..54  : nilaihuruf="D"; break;
  case 0..39   : nilaihuruf="E"; break;
  default      : cout<<"Nilai di luar 0 sampai 100";
  }


 //menampilkan nilai angka dan nilai huruf
 cout<<"Nilai akhir angka : "<<nilaiangka<<endl;
 cout<<"Nilai akhir huruf : "<<nilaihuruf;
}
```

Tugas Mandiri

1. Buatlah program untuk menentukan suatu bilangan yang dimasukkan berupa bilangan nol, genap atau ganjil.
2. Buatlah program untuk menentukan angka terbesar dari tiga angka yang dimasukkan. Masukan dari program ini adalah tiga angka secara bebas, kemudian dilakukan seleksi sehingga keluarannya adalah angka terbesar.
3. Buatlah program untuk mengecek suatu tahun dikategorikan tahun kabisat atau bukan. Masukan dari program ini adalah tahun bertipe bilangan bulat antara 0 sampai dengan 3000. Keluaran dari program ini adalah informasi apakah tahun yang dimasukkan termasuk tahun kabisat atau bukan.

   Catatan, suatu tahun dikatakan kabisat apabila habis dibagi 4 atau habis dibagi 400 jika tahun tersebut juga habis dibagi 100. Misalnya tahun 1996 merupakan tahun kabisat karena habis dibagi 4. Tahun 1900 bukan tahun kabisat karena habis dibagi 100 tetapi tidak habis dibagi 400.
4. Buatlah program untuk menentukan posisi suatu titik pada koordinat cartesius berada pada kuadran ke berapa. Masukan dari program ini adalah sumbu X dan sumbu Y. Untuk menentukan berada pada kuadran berapa didasarkan pada :

   Jika sumbu X positif dan sumbu Y positif maka berada pada kuadran I
   Jika sumbu X negatif dan sumbu Y positif maka berada pada kuadran II
   Jika sumbu X negatif dan sumbu Y negatif maka berada pada kuadran III
   Jika sumbu X positif dan sumbu Y negatif maka berada pada kuadran IV
5. Buatlah program untuk mengkonversi suatu angka numerik menjadi suatu kalimat. Misalnya jika dimasukkan angka 1500 maka akan ditampilkan tulisan “seribu lima ratus”.

BAB V
PERULANGAN (*LOOPING*)

Kompetensi dan Indikator

Kompetensi Dasar :
- Mengetahui dasar-dasar perulangan.
- Mengetahui penulisan perulangan.
- Mengetahui pemilihan jenis perulangan yang tepat sesuai dengan kebutuhannya.

Indikator Keberhasilan :
- Mampu menjelaskan dasar-dasar perulangan.
- Mampu menuliskan contoh program menggunakan perulangan.
- Mampu memilih jenis perulangan sesuai dengan kasus yang sedang dikerjakan.

Dasar Teori

A. Pengantar

Perulangan adalah suatu proses yang melakukan statemen-statemen dalam sebuah program secara terus-menerus sampai terdapat kondisi untuk menghentikannya. Struktur perulangan akan sangat membantu dalam efisiensi program.

Dalam konsep pemrograman terstruktur penggunaan perulangan sangat dianjurkan agar suatu program dapat terdefinisi dengan jelas sejak awal hingga akhir program. Kelemahan bahasa pemrograman yang *unstrutured language* adalah adanya lompatan-lompatan instruksi (menggunakan perintah **go to**) sehingga menyulitkan programmer jika terjadi kesalahan atau proses modifikasi yang dilakukan oleh orang lain.

B. Struktur for

Struktur pengulangan jenis ini digunakan untuk melakukan perulangan yang telah diketahui banyaknya statemen yang diulang. Untuk melakukan perulangan dengan menggunakan struktur ini, kita harus memiliki sebuah variabel sebagai indeksnya. Namun perlu sekali untuk diperhatikan bahwa tipe data dari variabel yang akan digunakan sebagai indeks haruslah tipe data yang mempunyai urutan yang teratur, misalnya tipe data **integer** (0,1,2,….) atau **char** („a", „b", „c",…..).

Adapun bentuk umum dari struktur **for** baik yang bersifat menaik maupun menurun seperti yang tampak di bawah ini.

```
// Untuk pengulangan yang sifatnya menaik (increment)
for (variabel=nilai_awal; kondisi; variabel ++)  {
   Statemen_yang_akan_diulang;
}


// Untuk pengulangan yang sifatnya menurun (decrement)
for (variabel=nilai_awal; kondisi; variabel --)  {
   Statemen_yang_akan_diulang;
}
```

Sebagai catatan bahwa jika kita melakukan pengulangan yang sifatnya menaik (*increment*) maka nilai awal dari variabel yang kita definisikan haruslah lebih kecil dari nilai akhir yang dituliskan dalam kondisi (ekspresi). Sebaliknya, jika kita akan melakukan pengulangan yang sifatnya menurun (decrement) maka nilai awal harus lebih besar dari nilai akhir.

Struktur for dengan Banyak Variabel

Struktur for di dalam C++ dapat juga melibatkan lebih dari satu variabel, namun yang jelas satu diantaranya akan digunakan sebagai indeks pengulangan. Adapun struktur **for** yang melibatkan tiga buah variabel adalah sebagai berikut :

```
for  (variabel1=nilai_awal1, variabel2=nilai_awal2, variabel3=nilai_awal3; kondisi1,
        kondisi2, kondisi3; variabel ++/--, variabel ++/--, variabel ++/--)
{
        Statemen_yang_akan_diulang;
}
```

Struktur for Bersarang

Sama halnya seperti pada percabangan, pada struktur pengulangan juga dapat juga diterapkan pengulangan bersarang (*nested looping*). Konsepnya sangat sederhana, yaitu dalam sebuah pengulangan terdapat pengulangan yang lainnya. Struktur penulisan *nested for* dapat dituliskan sebagai berikut :

```
for (variabel1=nilai_awal; kondisi1; variabel1++)  {
     for (variabel2=nilai_awal; kondisi2; variabel2++)  {
          for (variabel3=nilai_awal; kondisi3; variabel3++)  {
          .....
          }
     }
}
```

C. Struktur while

Struktur perulangan jenis ini adalah perulangan yang melakukan pengecekan kondisi di awal blok struktur. Biasanya digunakan untuk jumlah pengulangan yang belum diketahui jumlahnya, sehingga syarat berhentinya ditentukan pada saat proses perulangan berlangsung. Perulangan hanya akan dilakukan jika kondisi yang didefinisikan di dalamnya terpenuhi (bernilai benar).

Hal ini berarti jika kondisi yang didefinisikan tidak terpenuhi (bernilai salah) maka statemen-statemen yang terdapat dalam blok pengulangan pun tidak akan pernah dieksekusi oleh program.

Adapun bentuk umum dari struktur pengulangan **while** ini adalah seperti yang tampak di bawah ini.

```
While (kondisi)  {
        Statemen_statemen_yang_akan_diulang;
}
```

Perulangan ini biasa dikenal dengan perulangan syarat di awal (pada bahasa Pascal dikenal dengan perulangan **Do-While**), sehingga minimal terjadinya perulangan dengan menggunakan model ini adalah 0 (nol) kali. Karena jika pada saat pengecekan kondisi bernilai salah, maka statemen-statemen pada blok perulangan tidak dilakukan.

D. Struktur do-while

Berbeda dengan struktur while yang melakukan pengecekan kondisi di awal blok perulangan, pada struktur do-while kondisi justru ditempatkan di bagian akhir. Sehingga syarat berhentinya adalah jika kondisi bernilai salah (*false*).

Perulangan ini dikenal dengan perulangan syarat akhir (jika pada bahasa pemrograman Pascal menggunakan istilah **repeat-until**). Minimal terjadinya perulangan dengan menggunakan model ini adalah 1 (satu) kali. Karena perulangan langsung dikerjakan tanpa

adanya pengecekan kondisi, dan setelah satu kali perulangan baru dilakukan pengecekan kondisi.

Berikut ini bentuk umum dari struktur do-while.

```
Do {
   Statemen_statemen_yang_akan_diulang;
} while (kondisi);
```

E. Statemen Peloncatan

Dalam penggunaan perulangan, tidak jarang harus dilakukan peloncatan statemen yang memaksa untuk eksekusi statemen berjalan sesuai urutan yang kita inginkan, yaitu meloncat dari satu statemen ke statemen yang lain. Terdapat 4 model peloncatan, yaitu break, continue, goto dan exit.

Break

Berfungsi untuk menghentikan proses pengulangan dan program akan langsung meloncat keluar dari blok perulangan dan berpindah ke statemen yang berada di bawah blok perulangan yang bersangkutan.

Continue

Berfungsi untuk melanjutkan proses pengulangan dengan meloncat ke statemen awal yang terdapat dalam blok perulangan.

Goto

Berfungsi untuk meloncat dimana arah label yang menyertai perintah **goto.** Label ini hanya berfungsi sebagai tanda lokasi yang harus dituju, sehingga pemberian nama label tidak harus dideklarasikan terlebih dahulu.

Exit

Berfungsi untuk keluar dari program. Untuk melakukan terminasi program secara normal biasanya menggunakan parameter 0 (nol) pada fungsi exit, misalnya **exit(0);** Penggunaan fungsi **exit** diharuskan untuk mendefinisikan file header **cstdlib** pada bagian atas program.

Praktek

1. Menampilkan Tulisan “Fakultas Teknologi Informasi” 10 kali

Berikut ini adalah program sederhana untuk menampilkan tulisan sebanyak 10 kali menggunakan perulangan **for**.

Nama file : Latih245.cpp

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

void main(){
 int a;
 for (a=0; a<10; a++) {
  cout<<”Fakultas Teknologi Informasi;<<endl;
 }
}
```

2. Menampilkan Angka Naik dan Turun

Berikut ini adalah program untuk menampilkan angka (indeks) secara menaik (*increment*) dan menurun (*decrement*).

Nama file : Latih255.cpp

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

void main(){
 int a, b;

 //perulangan menaik atau increment dengan loncatan 1
 cout<<"Perulangan Menaik : "<<endl;
 for (a=0; a<5; a++) {
  cout<<a<<endl;
 }

 //perulangan menaik atau increment dengan loncatan 5
 cout<<"Perulangan Menaik : "<<endl;
 for (a=0; a<30; a++) {
  cout<<a+5<<endl;
 }

 //perulangan menurun atau decrement dengan loncatan 1
 cout<<"Perulangan Menurun : "<<endl;
 for (b=5; b>0; b--) {
  cout<<b<<endl;
 }

 //perulangan menurun atau decrement dengan loncatan 10
 cout<<"Perulangan Menurun : "<<endl;
  for (b=50; b>0; b--) {
   cout<<b-10<<endl;
  }
}
```

3. Struktur For dengan Banyak Variabel

Berikut ini disajikan sebuah contoh program dimana di dalamnya terdapat struktur for yang melibatkan tiga buah variabel, yaitu A, B dan C.

Nama file : Latih265.cpp

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

int main(){
 char A, B, C;
 for (A="a",B=0, C=1; A<="e"; A++, B=B+5, C=C*3) {
  cout<<"nilai A = "<<A<<endl;
  cout<<"nilai B = "<<B<<endl;
  cout<<"nilai C = "<<C<<endl;
  cout<<endl;
 }
}
```

Pada program di atas menunjukkan bahwa walaupun ketiga variabel akan diulang secara bersamaan, namun syarat berhentinya sebuah perulangan tergantung pada salah satu variabel. Dalam kasus di atas, syarat berhentinya adalah pada saat variabel A bernilai „e".

4. Struktur Nested For

Program di bawah ini berfungsi untuk menampilkan tampilan bintang yang bersifat dinamis. Tampilan bintang tergantung pada jumlah bintang yang dimasukkan oleh pengguna. Misalnya jika dimasukkan jumlah bintangnya adalah 5, maka tampilannya adalah :

```
*
* *
* * *
* * * *
* * * * *
```

Nama file : Latih275.cpp

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

void main(){
 int a, b, bintang;
 cout<<"Masukkan jumlah bintang : ";
 cin>>bintang;

 for (a=1; a<=bintang; a++) {
  for (b=1; b<=a; b++) {
   cout<<"* ";
  }
 cout<<endl;
 }
}
```

5. Menampilkan Hasil Perkalian Baris dan Kolom

Program di bawah ini berfungsi untuk menampilkan hasil perkalian antara baris dan kolom secara berurutan. Pada contoh ini tidak ada masukan, sehingga jumlah baris dan kolom ditentukan pada indek perulangan. Tampilan yang diinginkan adalah sebagai berikut :

```
1   2   3   4   5   6   7   8   9   10
2   4   6   8   10  12  14  16  18  20
3   6   9   12  15  18  21  24  27  30
4   8   12  16  20  24  28  32  36  40
5   10  15  20  25  30  35  40  45  50
6   12  18  24  30  36  42  48  54  60
7   14  21  28  35  42  49  56  63  70
8   16  24  32  40  48  56  64  72  80
9   18  27  36  45  54  63  72  81  90
10  20  30  40  50  60  70  80  90  100
```

Nama file : Latih285.cpp

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

void main(){
 int a, b;
 for (a=1; a<=10; a++) {
  for (b=1; b<=10; b++) {
   cout<<a*b;
  }
 cout<<endl;
```

```
 }
}
```

6. Menghitung Faktorial

Program di bawah ini berfungsi untuk menghitung nilai faktorial dari suatu bilangan yang dimasukkan. Contoh :
3! = 3x2x1 = 6
4! = 4x3x2x1 = 24
5! = 5x4x3x2x1 = 120

Nama file : Latih295.cpp

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

int main(){
 int bilangan;
 long faktorial=1;
 cout<<"Masukkan suatu bilangan : ";
 cin>>bilangan;
 cout<<endl;
 cout<<bil<<"! = ";
 while (bil >= 1) {
  faktorial *= bil;//bentuk singkat dari faktorial=faktorial*bil
  if (bil != 1) {
   cout<<bil<<" x ";
  } else {
  cout<<bil<<" = ";
  }
 bil--;
 }
}
cout<<faktorial;
return 0;
}
```

7. Menentukan Faktor Persekutuan Terbesar (FPB)

Berikut ini adalah program untuk menentukan niai faktor persekutuan terbesar dari dua buah bilangan bulat. Misalnya jika dimasukkan dua buah bilangan bulat yaitu 8 dan 12, maka faktor persekutuan terbesar dari kedua bilangan tersebut adalah 4.

8 → 1 2 **4** 8
12 → 1 2 3 **4** 6 12
Faktor yang sama dan terbesar antara bilangan 8 dan 12 adalah angka 4.

Adapun cara yang dapat digunakan untuk menyelesaikan permasalahan di atas adalah dengan algoritma berikut ini :

1. Masukkan bilangan pertama
2. Masukkan bilangan kedua
3. Jika bilangan pertama lebih kecil dari bilangan kedua maka tukar posisi antara bilangan pertama dengan bilangan kedua sehingga bilangan pertama menjadi lebih besar.
4. Selama sisa != 0, kerjakan langkah 5 sampai 7
5. sisa ← bilangan pertama % bilangan kedua
6. bilangan pertama ← bilangan kedua
7. Bilangan kedua ← sisa

8. FPB ← Bilangan pertama
9. Tampilkan FPB

Programnya adalah sebagai berikut :

Nama file : Latih305.cpp

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

int main(){
 int bil1, bil2, sisa;

 //input bilangan pertama dan kedua
 cout<<"Masukkan bilangan pertama : ";
 cin>>bil1;
 cout<<"Masukkan bilangan kedua   : ";
 cin>>bil2;

 //melakukan tukar bil1 ke bil2 jika bil1 < bil2
 if (bil1 < bil2) {
  int temp = bil1;
  bil1 = bil2;
  bil2 = temp;
 }

 do {
  sisa = bil1 % bil2;
  bil1 = bil2;
  bil2 = sisa;
 } while (sisa!=0);

 int FPB=bil1;
 cout<<"Faktor Persekutuan Terbesar dari "<<bil1<<" dan "<<bil2
 <<" adalah "<<FPB;
return 0;
}
```

Tugas Mandiri

1. Perhatikan contoh program **latih27.cpp.** Buatlah program sejenis tetapi dengan tampilan sebagai berikut :

```
* * * * *
* * * *
* * *
* *
*
```

2. Perhatikan contoh program **latih28.cpp**. Buatlah program untuk menampilkan hasil perkalian antara baris dan kolom seperti di bawah ini :

```
1   2   3   4   5   6   7   8   9   10
    4   6   8   10  12  14  16  18  20
        9   12  15  18  21  24  27  30
            16  20  24  28  32  36  40
                25  30  35  40  45  50
                    36  42  48  54  60
                        49  56  63  70
                            64  72  80
                                81  90
                                    100
```

3. Buatlah program untuk menentukan bilangan prima. Masukan dari program ini adalah sembarang bilangan. Keluarannya adalah informasi apakah angka yang dimasukkan berupa bilangan prima atau bukan.

4. Sebuah bola dipantulkan dari ketinggian 100 centimeter. Pantulan berikutnya adalah 70% dari pantulan pertama dan terus menerus sampai bola akan berhenti.

   Buatlah program untuk menghitung berapa kali bola melakukan pantulan sampai berhenti dan hitunglah berapa panjang lintasan yang dilalui mulai saat bola dijatuhkan sampai berhenti.

BAB VI
POINTER

Kompetensi dan Indikator

Kompetensi Dasar : - Mengetahui dasar-dasar pointer.
- Mengetahui penulisan pointer.
- Mengetahui penggunaan pointer untuk aplikasi.

Indikator Keberhasilan : - Mampu menjelaskan dasar-dasar pointer.
- Mampu menuliskan contoh program menggunakan pointer.
- Mampu menggunakan pointer pada contoh aplikasi.

Dasar Teori

A. Pengantar

Salah satu kelebihan dari bahasa C adalah karena bahasa ini mendukung sepenuhnya untuk pemanipulasian memori dengan menggunakan pointer. Namun di balik itu, pointer juga merupakan salah satu fitur C++ yang berbahaya karena dapat mengakibatkan sistem operasi pada komputer kita menjadi *crash* (rusak). Penggunaan pointer dengan cara yang salah juga dapt menyebabkan *bug* yang sangat sulit untuk ditemukan pada program kita.

Memahami pointer sama dengan sebuah rangkaian kereta api yang bersifat dinamis. Pada saat penumpang banyak dapat ditambah gerbong sesuai dengan jumlah penumpang, namun jika penumpang sedikit jumlah gerbong dapat dikurangi. Demikian juga pointer, alokasi memori dapat disesuaikan dengan jumlah datanya.

B. Variabel Pointer

Pointer dapat dikatakan sebagai suatu variabel yang menyimpan alamat memori. Pada bab-bab sebelumnya kita sudah terbiasa dengan penggunaan variabel, tapi variabel-variabel tersebut hanya berisi nilai, bukan alamat. Jika kita mempunyai sebuah variabel dengan tipe data tertentu maka untuk mendapatkan alamat dari variabel tersebut adalah dengan menggunakan operator &. Alamat inilah yang kemudian akan disimpan ke dalam variabel yang bertipe pointer. Sedangkan untuk mendeklarasikan variabel sebagai pointer, kita hanya menambahkan tanda *asterisk* (*) di depan nama variabel. Berikut ini bentuk umum dari pendeklarasian variabel yang bertipe pointer.

```
tipe_data *nama_pointer; //atau
tipe_data * nama_pointer; //atau
tipe_data* nama_pointer;
```

Tipe data di atas berguna untuk menyatakan bahwa pointer yang kita deklarasikan tersebut akan ditempati oleh data dengan tipe tertentu. Sebagai contoh, kita akan mendeklarasikan pointer P yang akan ditempati oleh tipe data long, maka bentuk pendeklarasiannya adalah sebagai berikut:

```
long *P; //mendeklarasikan pointer P menunjuk ke tipe longinteger
```

Jika kita mempunyai sebuah variabel yang bertipe long (misalnya X), maka kita dapat memerintahkan pointer P di atas untuk menunjuk ke alamat yang ditempati oleh variabel X. adapun sintaks untuk melakukan hal tersebut adalah seperti yang terlihat di bawah ini.

```
Long X; //mendeklarasikan variabel X bertipe longinteger
long *P; //mendeklarasikan pointer P menunjuk ke tipe longinteger
```

```
P = &X; //memerintahkan P untuk menunjuk alamat dari variabel X.
```

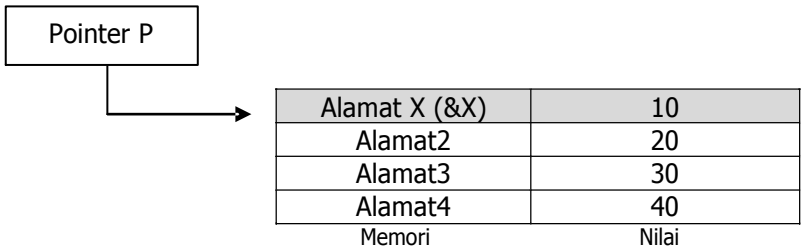
Apabila kita analisis potongan sintaks di atas, sebenarnya konsepnya sangat sederhana. Kita tahu bahwa P adalah pointer (berisi alamat) dan &X juga berisi alamat, maka kita dapat menyimpan alamat dari variabel X tersebut ke dalam pointer P. kita tidak diizinkan utnuk memasukkan sebuah nilai (bukan alamat) ke dalam pointer P. Misalnya dengan menuliskan sintaks seperti berikut:

```
P = X; //salah, karena X berupa nilai (bukan alamat)
```

Jika kita memang ingin mengisikan nilai ke dalam alamat yang disimpan oleh pointer P, maka seharusnya kita menggunakan tanda asterisk (*) di depan nama pointer tersebut, yaitu dengan mengubah sintaks di atas menjadi seperti di bawah ini.

```
*P = X; //benar, karena *P adalah nilai yang berada pada pointer P
```

Sebagai catatan bahwa *P ini disebut dengan dereference pointer. Untuk dapat lebih memahami konsep pointer. Berikut ini gambar yang mengilustrasikan kasus di atas.



Pada gambar di atas alamat 1 dari memori ditempati oleh variabel X yang bertipe long. Adapun nilai dari variabel X tersebut adalah 10. di atas kita mempunyai pointer P yang menunjuk ke alamat X, maka untuk mendapatkan nilai X kita dapat menggunakan dereference pointer, yaitu dengan *P. dengan demikian dapat disimpulkan bahwa jika:

```
P = &X; // keduanya menyimpan alamat
```

Maka:

```
*P = X; // keduanya menyimpan nilai
```

C. Memasukkan Nilai pada Pointer

Nilai yang dimaksud tentu berupa alamat, bukan berupa nilai data. Walaupun tampak mudah tapi kita juga harus berhati-hati dalam melakukan hal ini. Perlu diperhatikan bahwa tipe data dari pointer harus sama dengan tipe data dari variabel yang akan menempatinya. Hal ini merupakan hal yang biasa terabaikan oleh para programmer pemula. Misalnya kita mendeklarasikan pointer P ke tipe double dan kita juga memiliki variabel X yang bertipe int. pada kasus ini kita tidak diijinkan untuk menyimpan alamat dari variabel X ke pointer P karena tipenya berbeda.

Contoh :

```
double P;
int X; //mendefinisikan variabel bertipe int

P = &X; /*salah, karena P hanya menyimpan alamat dari variabel-variabel yang bertipe
double saja */
```

D. Menggunakan Kata Kunci New

Untuk mengalokasikan memori pada ruang yang masih kosong menggunakan kata kunci **new.** Kata kunci ini akan diikuti oleh tipe data yang akan dialokasikan sehingga kompiler akan mengetahui seberapa besar ruang memori yang dibutuhkan untuk proses pengalokasian tersebut. Misalnya jika mengalokasikan tipe data **long**, maka ruang yang dibutuhkan adalah 4 byte sedangkan unsigned short hanya membutuhkan 2 byte.

Pada saat mengalokasikan memori, alamat memori yang dialokasikan tersebut akan disimpan ke pointer, sehingga sebelumnya harus dideklarasikan pointernya terlebih dahulu. Struktur penulisan penggunaan kata kunci **new** adalah sebagai berikut :

```
Nama_pointer = new tipe_data;
```

Tapi jika ingin mengalokasikan beberapa ruang memori, misalnya akan mengalokasikan 10 buah ruang memori dengan menggunakan tipe data **long** (berukuran 4 byte) maka memori yang dibutuhkan adalah 10 x 4 – 40 byte. Adapun bentuk penulisannya adalah :

```
Nama_pointer = new tipe_data [n];
```

E. Menggunakan Kata Kunci Delete

Memori yang sudah tidak digunakan lagi dapat dihapus agar tidak memborosi memori komputer. Struktur penulisan penggunaan kata kunci **delete** adalah:

```
delete nama_pointer;
```

Praktek

1. Menampilkan Nilai dan Alamatnya

Berikut ini adalah contoh program penggunaan tipe data pointer. Perhatikan cara penulisan dan cara mendefinisikan pointer.

Nama file : Latih316.cpp

```
#include <iostream>

using namespace std;

int main() {

  long *P;
  long X;

  P = &X;

  X = 10;           // Mengisikan nilai 10 ke dalam variabel X

  cout<<"Nilai X : "<<X<<endl;
  cout<<"Nilai *P          : "<<*P<<endl;
  cout<<"Nilai P : "<<P<<endl;
  cout<<"Nilai &X : "<<&X<<endl;

  *P = 200;         // Mengisikan nilai 200 ke dalam *P

  cout<<"Nilai X : "<<X<<endl;
  cout<<"Nilai *P : "<<*P<<endl;
  cout<<"Nilai P : "<<P<<endl;
```

```
  cout<<"Nilai &X : "<<&X<<endl;

  return 0;
}
```

2. Pointer tanpa tipe data

Berikut ini adalah contoh program yang menggunakan pointer dapat menyimpan alamat dari variabel-variabel yang bertipe apapun.

Nama file : Latih326.cpp

```
#include <iostream>

using namespace std;

int main() {
  void *P;              // Mendeklarasikan pointer P
            // sebagai pointer tanpa tipe

  // Mendeklarasikan variabel X, Y dan Z dengan tipe berbeda
  int X;
  long Y;
  double Z;

  // Memerintahkan P untuk menunjuk ke alamat dari variabel X
  P = &X;
  X = 10;             // Mengisikan variabel X dengan nilai 10
  // Menampilkan hasil
  cout<<"Nilai X : "<<X<<endl;
  cout<<"Nilai P : "<<P<<endl;
  cout<<"Nilai &X           : "<<&X<<endl;
  cout<<endl;

  // Memerintahkan P untuk menunjuk ke alamat dari variabel Y
  P = &Y;
  Y = 2000;         // Mengisikan variabel Y dengan nilai 2000
  // Menampilkan hasil
  cout<<"Nilai Y : "<<Y<<endl;
  cout<<"Nilai P : "<<P<<endl;
  cout<<"Nilai &Y           : "<<&Y<<endl;
  cout<<endl;

  // Memerintahkan P untuk menunjuk ke alamat dari variabel Z
  P = &Z;
  Z = 21.0378;    // Mengisikan variabel Z dengan nilai 21.0378
  // Menampilkan hasil
  cout<<"Nilai      Z       :      "<<Z<<endl;
  cout<<"Nilai      P       :      "<<P<<endl;
  cout<<"Nilai &Z           : "<<&Z<<endl;

  return 0;
}
```

3. Menggunakan Kata Kunci New

Nama file : Latih336.cpp

```
#include <iostream>

#define MAX 5
```

```
using namespace std;

int main() {
  float *P1;
  int *P2;

  P1 = new float; // Mengalokasikan satu ruang memori
                  // dan disimpan ke pointer P1
  *P1 = 3.14; // Mengisikan nilai ke dalam ruang
              // yang telah dialokasikan

  // Menampilkan nilai dan alamat yang disimpan
  // dalam pointer P1
  cout<<"Nilai *P1          : "<<*P1<<endl;
  cout<<"Nilai P1           : "<<P1<<endl;
  cout<<'\n';

  // Mengalokasikan 5 buah ruang memori dan disimpan
  // ke pointer P2
  P2 = new int[MAX];

  // Mengisikan nilai ke dalam ruang-ruang memori
  // yang telah dialokasikan
  for (int C=0; C<5; C++) {
     *P2 = (C+1) * 10;
     P2 += 1;
  }

  // Mengembalikan pointer P2 agar menunjuk ke alamat
  // dari elemen ke-0
  P2 -= 5;

  // Menampilkan nilai dan alamat yang disimpan
  // dalam pointer P2
  for (int c=0; c<5; c++) {
      cout<<"Nilai *P2 ke-"<<c<<"   : "<<*P2<<endl;
      cout<<"Nilai P2 ke-"<<c<<"    : "<<P2<<endl<<endl;
      P2 += 1;
  }

  return 0;
}
```

4. Menggunakan Kata Kunci Delete

Nama file : Latih346.cpp

```
#include <iostream>

using namespace std;

int main() {
  int *P; // Mendeklarasikan pointer P yang menunjuk
          // ke tipe data int

  // Melakukan alokasi memori
  P = new int;

  // Menggunakan memori yang telah dialokasikan
  *P = 100;
```

```
  cout<<"Nilai *P : "<<*P<<endl;

  // Melakukan dealokasi memori
  delete P;

  return 0;
}
```

BAB VII
ARRAY

Kompetensi dan Indikator

Kompetensi Dasar : - Mengetahui dasar-dasar array.
- Mengetahui penulisan array.
- Mengetahui penggunaan array untuk aplikasi.

Indikator Keberhasilan : - Mampu menjelaskan dasar-dasar array.
- Mampu menuliskan contoh program menggunakan array.
- Mampu menggunakan array pada contoh aplikasi.

Dasar Teori

A. Pengantar

Array adalah variabel penyimpan sekumpulan data yang memiliki tipe sama. Setiap data menempati lokasi atau alamat memori yang berbeda-beda dan selanjutnya disebut dengan elemen array. Elemen array itu kemudian dapat diakses melalui indeks yang terdapat di dalamnya. Berbeda dengan Bahasa Pascal yang memulai indeks dari 1 (satu), indeks array pada Bahasa C dimulai dari 0 (nol).

Berikut ini adalah gambaran sederhana tentang array :

Alamat	indeks	Nilai
Alamat#1	0	Nilai ke-1
Alamat#2	1	Nilai ke-2
Alamat#3	2	Nilai ke-3
Alamat#4	3	Nilai ke-4
Alamat#5	4	Nilai ke-5
Alamat#6	5	Nilai ke-6
...	...	...
Alamat#n	n	Nilai ke-n

Sebelum digunakan, array harus dideklarasikan terlebih dahulu menggunakan tanda **[ ]** (*bracket*). Struktur penulisannya adalah sebagai berikut :

```
Tipe_data nama_array [jumlah_elemen];
```

Contoh :

```
int nilai[50];
char abjad [28];
float nilai_rata[30];
```

Pada contoh di atas dideklarasikan suatu array yang bernama **nilai** dengan tipe data integer yang jumlah elemennya maksimal 50 data dengan urutan elemen ditulis sebagai berikut:

Nilai[0] ← menunjukkan elemen data ke-1 pada indeks ke-0.

Nilai[1] ← menunjukkan elemen data ke-2 pada indeks ke-1.

Nilai[2] ← menunjukkan nilai elemen ke-3 pada indeks ke-2, dan seterusnya.

Sehingga jika menginginkan untuk mengakses elemen data ke 10, perintah yang digunakan adalah **nilai[9]** bukan **nilai[10]**, karena indeks dimulai dari **0.**

B. Memasukkan Nilai ke dalam Elemen Array

Untuk mengisikan nilai ke dalam elemen-elemen array dapat dilakukan langsung untuk setiap elemen jika diketahui alamat indeksnya. Misal jika akan memasukkan nilai 60 ke dalam array **nilai** pada indeks ke 7 maka perintahnya :

Nilai[7] = 60; ← *masukkan nilai 60 ke variabel nilai indeks ke-7.*

Namun cara ini tidak direkomendasikan karena tidak efisien dan bersifat statis. Karena indeks pada array bernilai urutan data secara teratur dari 0, 1, 2, 3, 4, dan seterusnya maka lebih umum dan banyak digunakan oleh para programmer untuk mengisikan nilai ke dalam elemen-elemen array dengan perulangan (*looping*). Cara ini akan lebih cepat dan bersifat dinamis.

Struktur penulisan array dengan menggunakan perulangan adalah :

```
Tipe_data nama_array [jumlah_elemen];

for (indeks=nilai_awal; indeks<=nilai_akhir; increment_indeks) {
  statemen nilai yang dimasukkan atau ditampilkan;
}
```

Contoh :

```
int data[10]; //deklarasi array

//memasukkan 10 data menggunakan perulangan for
for (int a=0; a<10; a++) {
  cout<<"Nilai elemen data pada indeks ke "<<a<<" = ";
  cin>>data[a];
}
```

C. Menampilkan Nilai Pada Elemen Array

Menampilkan nilai pada elemen array sama seperti pada saat memasukkan data. Perhatikan contoh berikut ini :

```
int data[10]; //deklarasi array

//memasukkan 10 data menggunakan perulangan for
for (int a=0; a<10; a++) {
  cout<<"Nilai elemen data pada indeks ke "<<a<<" = "<<data[a];
}
```

Pada contoh di atas, nilai elemen data akan ditampilkan berurutan seperti pada nilai yang dimasukkan pada saat input data dengan posisi indeks yang tetap.

D. Array Multidimensi

Array multidimensi yaitu array yang terdiri dari beberapa subskrip array. Sebagai contoh, array 2 dimensi adalah array yang mempunyai 2 subskrip array. Array seperti ini biasa digunakan untuk operasi matrik.

Struktur penulisannya adalah sebagai berikut :

```
Tipe_data nama_array [jumlah_elemen_1][jumlah_elemen_2];
```

Contoh :

```
int matrik[2][3]; //deklarasi array 2 dimensi

//memasukkan data menggunakan perulangan for
for (int i=0; i<2; i++) {
 for (int j=0; j<3; j++) {
  cout<<"Elemen data pada baris ke "<<i<<" kolom ke "<<j<<" = ";
  cin>>matrik[i][j];
 }
}

//menampilkan elemen data
for (int i=0; i<2; i++) {
 for (int j=0; j<3; j++) {
  cout<<"Data baris ke-"<<i<<" kolom ke-"<<j<<" = "<<matrik[i][j];
 }
}
```

Praktek

1. Memasukkan data ke variabel array

Berikut ini adalah program sederhana untuk memasukkan nilai ke dalam variabel array dan menampilkan pada bagian program berikutnya. Data yang dapat dimasukkan dalam program ini maksimal 5 angka dengan tipe bilangan bilat karena variabel **data** dideklarasikan dengan **int data[5];** dan dilakukan perulangan **for** mulai dari 0 sampai dengan kurang dari 5.

Nama file : Latih357.cpp

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


int main() {
 int data[5]; //deklarasi array
 //memasukkan data menggunakan perulangan for
 for (int i=0; i<5; i++) {
  cout<<"Masukkan data pada elemen ke "<<i<<" = ";
  cin>>data[i];
 }

 cout<<endl;
 //menampilkan elemen data
 for (int i=0; i<5; i++) {
  cout<<"Data baris ke-"<<i<<" = "<<data[i];
 }
 return 0;
}
```

2. Mencari data dalam Variabel Array

Dalam beberapa aplikasi program membutuhkan proses pencarian dalam suatu deretan elemen array. Misalnya mencari nama „**wibi**" dalam deretan array nama-nama mahasiswa.

Pada dasarnya pencarian data pada deret elemen array adalah membandingkan data yang cocok antara yang dicari dengan setiap elemen data secara berurutan dari indeks awal hingga akhir. Jika ditemukan data yang sesuai maka akan dicatat pada indeks ke berapa dan siap untuk ditampilkan. Tetapi jika hingga akhir indeks tidak ditemukan akan ditampilkan informasi bahwa „data yang dicari tidak ditemukan".

Nama file : Latih367.cpp

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

int main() {
 //deklarasi dan inisialisasi array
 int A[10]= {5, 12, 24, 53, 51, 26, 17, 62, 36, 68};
 int cari;

 //menampilkan elemen data
 for (int i=0; i<10; i++) {
  cout<<"Data baris ke-"<<i<<" = "<<A[i];
 }

 cout<<endl;

 //memasukkan data yang akan dicari
 cout<<"Masukkan data yang dicari : ";
 cin>>cari;

 //melakukan pencarian data
 for (int j=0; j<10; j++) {
  if A[j] == cari {
   cout<<"Nilai yang dicari berada pada indeks ke-"<<j;
  } else {
   cout<<"Data yang dicari tidak ditemukan";
  }
return 0;
}
```

3. Menentukan nilai tertinggi dan terendah dari deret array

Program di bawah ini merupakan contoh penggunaan array untuk mencari nilai tertinggi dan terendah dalam urutan array.

Algoritma yang dapat digunakan untuk menyelesaikan permasalahan ini adalah sebagai berikut :

1. Asumsikan nilai tertinggi ← elemen indeks pertama
2. Asumsikan nilai terendah ← elemen indeks pertama
3. Selama indeks belum berakhir, kerjakan langkah 4 sampai 5.
4. Jika elemen pada indeks lebih besar dari tertinggi maka tertinggi ← elemen indeks
5. Jika elemen pada indeks lebih kecil dari terrendah maka terrendah ← elemen indeks
6. Tampilkan elemen tertinggi.
7. Tampilkan elemen terrendah.

Nama file : Latih377.cpp

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

```
#include <conio.h>


int main() {
 //deklarasi dan inisialisasi array
 int A[10]= {5, 12, 24, 53, 51, 26, 17, 62, 36, 68};
 int tertinggi, terendah;

 //menampilkan elemen data
 for (int i=0; i<10; i++) {
  cout<<"Data baris ke-"<<i<<" = "<<A[i];
 }

 cout<<endl;
 tertinggi = A[0];
 terendah = A[0];

 //melakukan seleksi tertinggi dan terendah
 for (int j=0; j<10; j++) {
  if A[j] > tertinggi {
   tertinggi = A[j];
  } if A[j] < terendah {
   terendah = A[j];
  }

 //menampilkan nilai tertinggi dan terendah
 cout<<"Nilai tertinggi adalah "<<tertinggi;
 cout<<"Nilai terrendah adalah "<<terendah;

return 0;
}
```

4. Mengurutkan data pada deret array

Untuk mengurutkan deretan data menggunakan berbagai macam algoritma. Pada cotoh program ini disajikan menggunakan algoritma *buble sort* (metode pengurutan gelembung)l

Algoritma yang dapat digunakan adalah sebagai berikut :

1. i ← 0
2. Selama (i<N) kerjakan baris 3 sampai dengan 7
3. j ← N
4. Selama (j>=i) kerjakan baris 5 sampai dengan 7
5. Jika data[j-1] > data[j] maka tukar data[j-1] dengan data[j]
6. j ← j - 1;
7. i ← i + 1

Nama file : Latih387.cpp

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

int main() {
 //deklarasi array
 int data[5];
 int a, b, c, temp;
```

```
//memasukkan nilai array
 cout<<"INPUT DATA KE DALAM ARRAY :"<<endl;
 cout<<"---------------------------"<<endl;
 for (int i=0; i<5; i++) {
  cout<<"Masukkan data baris ke-"<<i<<" = ";
  cin>>data[i];
 }

//melakukan pengurutan elemen data
 for (int j=0; j<4; j++) {
  for (int k=5; k>0; k--) {
   if data[k] < data[k-1] {
    temp = data[k]; //menukar posisi
    data[k] = data[k-1];
    data[k-1] = temp;
   }
  }
 }
 //menampilkan nilai setelah urut
 cout<<"DATA SETELAH DIURUTKAN :"<<endl;
 cout<<"------------------------"<<endl;
 for (int i=0; i<5; i++) {
  cout<<"Data baris ke-"<<i<<" = "<<data[i];
 }
return 0;
}
```

5. Operasi Matrik Pada Array 2 Dimensi

Berikut ini adalah contoh operasi matrik memanfaatkan variabel array 2 dimensi.

Nama file : Latih397.cpp

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

int main() {
 //memasukkan tipedef yang berbentuk array
 typedef int matrik[3][2];

 //mendefinisikan A, B, C sebagai array 2 dimensi
 matrik A, B, C;
 int i, j, k;

 //memasukkan nilai ke dalam elemen-elemen matrik A
 cout<<"INPUT DATA KE DALAM MATRIK A :"<<endl;
 cout<<"------------------------------"<<endl;
 for (i=0; i<3; i++) {
  for (j=0; j<2; j++) {
   cout<<"A["<<i<<"]["<<j<<"]= "; cin>>A[i][j];
  }
 }
 cout<<endl;

 //memasukkan nilai ke dalam elemen-elemen matrik B
 cout<<"INPUT DATA KE DALAM MATRIK B :"<<endl;
 cout<<"------------------------------"<<endl;
 for (i=0; i<3; i++) {
```

```
  for (j=0; j<2; j++) {
   cout<<"B["<<i<<"]["<<j<<"]= "; cin>>B[i][j];
  }
 }
 cout<<endl;

 //melakukan penjumlahan matrik A + matrik B
 for (i=0; i<3; i++) {
  for (j=0; j<2; j++) {
   C[i][j]= A[i][j]+B[i][j];
  }
 }

//menampilkan hasil penjumlahan
 cout<<"HASIL PENJUMALAH MATRIK A + MATRIK B :"<<endl;
 cout<<"-------------------------------------"<<endl;
 for (i=0; i<3; i++) {
  for (j=0; j<2; j++) {
   cout<<"C["<<i<<"]["<<j<<"]= "<<C[i][j];
  }
 }
 return 0;
}
```

Tugas Mandiri

1. Buatlah program untuk mencari rerata dari sejumlah angka yang dimasukkan ke dalam deret array.
2. Dari program di atas, kembangkan menjadi program yang dapat menghitung standar deviasi.
3. Buatlah program untuk menampilkan tulisan yang berkurang satu huruf setiap barisnya hingga berakhir.

   Contoh :

   Masukan suatu kata : INFORMATIKA

   Keluaran yang diperlukan adalah :

   INFORMATIKA
   INFORMATIK
   INFORMATI
   INFORMAT
   INFORMA
   INFORM
   INFOR
   INFO
   INF
   IN
   I

BAB VIII
FUNGSI

Kompetensi dan Indikator

Kompetensi Dasar : - Mengetahui fungsi.
- Mengetahui penulisan fungsi.
- Mengetahui penggunaan dan manfaat fungsi pada aplikasi.

Indikator Keberhasilan : - Mampu menjelaskan tentang fungsi.
- Mampu menuliskan program menggunakan fungsi.
- Mampu menggunakan fungsi pada contoh aplikasi.

Dasar Teori

A. Pengantar

Suatu program merupakan kumpulan dari fungsi-fungsi, baik yang didefinisikan langsung dalam program maupun yang disimpan dalam suatu *header file.* Bahasa C memiliki fungsi utama yaitu **fungsi main()** yang harus ada setiap program C dan kompiler akan menjalankan printah-perintah yang terdapat dalam fungsi ini.

Fungsi merupakan sub program yang menjadikan program bersifat modular sehingga akan mempermudah proses penelusuran program. Dalam bahasa C tidak mengenal istilah prosedur. Fungsi dibedakan menjadi dua, yaitu *user defined function* dan *built-in function. User defined function* adalah fungsi-fungsi yang didefinisikan sendiri, sedangkan *built-in function* adalah fungsi-fungsi yang telah disediakan di dalam program.

B. Fungsi Tanpa Nilai Balik

Fungsi yang tidak memiliki nilai yang akan dikembalikan ke fungsi utama sama seperti pada bahasa Pascal dengan istilah *procedure* dimana subprogram ini digunakan melaksanakan proses tertentu. Karena tidak mengirimkan nilai balik (*return value*), maka fungsi ini menggunakan tipe **void.**

Struktur penulisan fungsi ini adalah :

```
void nama_fungsi () {
  statemen yang dilakukan;
  .....
}
```

Penulisan nama fungsi bersifat wajib dengan mengikuti aturan-aturan penulisan pengenal (*identifier*). Adapun pemanggilan fungsi pada fungsi utama adalah dengan menyebutkan nama fungsi.

C. Fungsi dengan Nilai Balik (*return value*)

Fungsi ini berguna untuk melakukan suatu proses yang dapat mengembalikan sebuah nilai ke dalam fungsi utama. Dalam membuat fungsi ini kita harus mendefinisikan tipe data dari nilai yang akan dikembalikan. Adapun tipe data yang akan dikembalikan harus sesuai dengan penggunaan pada fungsi utama.

Struktur penulisan dari fungsi ini adalah :

```
Tipe_data nama_fungsi () {
  statemen yang dilakukan;
  .....
  return nilai_yang_akan_dikembalikan;
}
```

D. Fungsi dengan Parameter

Pendefinisian fungsi yang tidak diikuti dengan daftar parameter, hasil yang didapatkan selalu bernilai tetap. Dengan pendefinisian fungsi berparameter, hasil yang diberikan dapat bersifat dinamis, tentunya tergantung dari nilai parameter yang dimasukkan.

Parameter yang terdapat pada pendefinisian sebuah fungsi disebut dengan *parameter formal*, sedangkan parameter pada saat pemanggilan disebut dengan *parameter aktual.* Jumlah serta tipe data antara parameter formal dan parameter aktual harus sesuai.

Jenis Parameter

Dalam C++ ada tiga jenis parameter yang dapat dilewatkan, yaitu parameter masukan, keluaran dan masukan/keluaran.

Parameter masukan, digunakan sebagai nilai masukan dalam sebuah fungsi. Niai tersebut kemudian akan diproses oleh fungsi untuk menghasilkan sebuah nilai kembalian (*return value*).

Parameter keluaran, berfungsi untuk menampung nilai yang dihasilkan dari proses di dalam fungsi. Parameter keluaran pada umumnya digunakan dalam fungsi bertipe **void** (fungsi yangtidak mengembalikan nilai).

Parameter masukan/keluaran, adalah parameter yang digunakan sebagai masukan dan juga keluaran. Mula-mula nilai dari parameter ini akan digunakan sebagai masukan untuk proses di dalam fungsi, setelah proses selesai maka hasilnya akan disimpan kembali ke dalam parameter tersebut sebagai nilai keluaran.

Melewatkan Parameter Berdasarkan Nilai (*Pass by Value*)

Fungsi ini akan melewatkan nilai parameter ke dalam sebuah fungsi untuk digunakan sesuai proses yang terdapat dalam fungsi tersebut. Jika melewatkan fungsi dengan cara ini maka nilai yang dihasilkan oleh fungsi tidak akan mempengaruhi nilai yang terdapat dalam program (di luar fungsi tersebut), sebab pada saat pemanggilan fungsi kompiler hanya akan membuat salinan (*copy*) dari nilai yang terdapat pada parameter aktual ke parameter formal. Dengan kata lain, yang akan berubah adalah nilai di dalam fungsi saja.

Melewatkan Parameter Berdasarkan Alamat (*Pass by Reference*)

Fungsi ini melewatkan parameter ke sebuah fungsi berdasarkan alamatnya. Jika membuat alias dari sebuah variabel maka alamat dari variabel dan alias tersebut adalah sama. Hal ini yang menjadi konsep dasar pengiriman parameter berdasarkan alamat.

Struktur penulisan fungsi dengan menggunakan parameter adalah sebagai berikut :

```
Tipe_data nama_fungsi (parameter1, parameter2, ...) {
  statemen yang dilakukan;
  .....
```

```
  return nilai_yang_akan_dikembalikan;
}
```

Praktek

1. Membuat dan memanggil fungsi tanpa nilai balik

Pada contoh berikut ini akan dibuat sebuah fungsi yang dapat menuliskan teks "Fakultas Teknologi Informasi" sebanyak 10 kali. Kemudian pada fungsi utama, nama fungsi akan dipanggil untuk dieksekusi.

Nama file : Latih408.cpp

```
#include <iostream>

using namespace std;

// Membuat fungsi dengan nama Tulis10Kali
void Tulis10Kali() {
  for (int C=0; C<10; C++) {
    cout<<"Aku Fakultas Teknologi Informasi"<<endl;
  }
}

// Fungsi utama dalam program C++
int main() {

  // Memanggil fungsi Tulis10Kali untuk dieksekusi
  Tulis10Kali();

  return 0;
}
```

2. Fungsi untuk mengembalikan nilai string

Pada contoh berikut ini, suatu fungsi akan mengembalikan nilai dalam bentuk string.

Nama file : Latih418.cpp

```
#include <iostream>

using namespace std;

// Membuat fungsi sederhana yang mengembalikan tipe string
char* TestFungsiString() {
  return "Ini adalah nilai dari fungsi";
}

// Fungsi utama
int main() {
  // Memanggil dan menampilkan hasil fungsi
  cout<<TestFungsiString();

  return 0;
}
```

3. Fungsi untuk mengembalikan tipe bilangan

Program di bawah ini merupakan contoh penggunaan fungsi untuk mengembalikan nilai berupa bilangan.

Nama file : Latih428.cpp

```
#include <iostream>

using namespace std;

// Membuat fungsi dengan nilai kembalian bertipe double
double TestFungsiBilangan() {
  return (3.14 * 2);
}

// Fungsi utama
int main() {

  cout<<"Nilai yang terdapat dalam fungsi : ";
  cout<<TestFungsiBilangan();

  return 0;
}
```

4. Menggunakan Fungsi Berparameter Masukan

Salah satu jenis parameter adalah parameter masukan, dimana pada contoh ini variabel X difungsikan sebagai parameter masukan.

Nama file : Latih438.cpp

```
#include <iostream>

using namespace std;

// Membuat fungsi dengan parameter bertipe masukan
int TambahSatu(int X) {
  int hasil;
  hasil = X + 1;
  return hasil;
}

int main() {

  /* Mendeklarasikan variabel yang akan digunakan sebagai nilai
     parameter pada saat pemanggilan */
  int Bilangan, HASIL;

  cout<<"Masukkan sebuah bilangan bulat : "<<endl;
  cin>>Bilangan;

  HASIL = TambahSatu(Bilangan);

  // Menampilkan nilai setelah diproses di dalam fungsi
  cout<<"Nilai akhir : "<<HASIL;

  return 0;
}
```

5. Melewatkan Paramater *Pass by Value*

Berikut ini adalah contoh sederhana yang melewatkan parameter berdasarkan nilai. Pada kasus ini fungsi akan melakukan pertukaran nilai data dari isi bilangan pertama ditukar dengan isi bilangan kedua.

Nama file : Latih448.cpp

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

// Mendefinisikan fungsi untuk menukarkan dua buah bilangan
void TukarBilangan(int X, int Y) {

  int Z = X;
  X = Y;
  Y = Z;

  // Menampilkan bilangan yang terdapat di dalam fungsi
  cout<<"Di dalam fungsi"<<endl;
  cout<<"Bilangan ke-1 : "<<X<<endl;
  cout<<"Bilangan ke-2 : "<<Y<<endl;
  cout<<endl;
}

// Fungsi utama
int main() {
  // Mendeklarasikan variabel Bilangan1 dan Bilangan2
  int Bilangan1, Bilangan2;
  cout<<"Masukkan bilangan pertama : "; cin>>Bilangan1;
  cout<<"Masukkan bilangan kedua    : "; cin>>Bilangan2;

  // Menampilkan nilai awal
  cout<<"Keadaan awal"<<endl;
  cout<<"Bilangan ke-1 : "<<Bilangan1<<endl;
  cout<<"Bilangan ke-2 : "<<Bilangan2<<endl;
  cout<<endl;

  // Memanggil fungsi TukarBilangan
  TukarBilangan(Bilangan1, Bilangan2);

  // Menampilkan nilai akhir setelah pemanggilan fungsi tukar
  cout<<"Keadaan akhir"<<endl;
  cout<<"Bilangan ke-1 : "<<Bilangan1<<endl;
  cout<<"Bilangan ke-2 : "<<Bilangan2<<endl;
  cout<<endl;
  return 0;
}
```

6. Melewatkan Paramater *Pass by Value*

Berikut ini adalah contoh sederhana yang melewatkan parameter berdasarkan referensi menggunakan kasus yang sama dengan contoh di atas. Perhatikan perbedaannya antara melewatkan parameter berdasarkan nilai dan referensi.

Nama file : Latih458.cpp

```
#include <iostream>

using namespace std;

// Mendefinsikan fungsi untuk menukarkan dua buah bilangan
```

```
void TukarBilangan(int& X, int& Y) {
  int Z = X;
  X = Y;
  Y = Z;

  // Menampilkan bilangan yang terdapat di dalam fungsi
  cout<<"Di dalam fungsi"<<endl;
  cout<<"Bilangan ke-1 : "<<X<<endl;
  cout<<"Bilangan ke-2 : "<<Y<<endl;
  cout<<endl;
}

// Fungsi utama
int main() {

  // Mendeklarasikan variabel Bilangan1 dan Bilangan2
  int Bilangan1, Bilangan2;

  cout<<"Masukkan bilangan pertama : "; cin>>Bilangan1;
  cout<<"Masukkan bilangan kedua    : "; cin>>Bilangan2;

  // Menampilkan nilai awal
  cout<<"Keadaan awal"<<endl;
  cout<<"Bilangan ke-1 : "<<Bilangan1<<endl;
  cout<<"Bilangan ke-2 : "<<Bilangan2<<endl;
  cout<<endl;

  // Memanggil fungsi TukarBilangan
  TukarBilangan(Bilangan1, Bilangan2);

  // Menampilkan nilai akhir setelah pemanggilan fungsi tukar
  cout<<"Keadaan akhir"<<endl;
  cout<<"Bilangan ke-1 : "<<Bilangan1<<endl;
  cout<<"Bilangan ke-2 : "<<Bilangan2<<endl;
  cout<<endl;

  return 0;
}
```

7. Menggunakan Fungsi dengan Parameter Bertipe Array

Berikut ini adalah contoh sederhana yang melewatkan parameter dengan tipe array.

Nama file : Latih468.cpp

```
#include <iostream>

using namespace std;

// Mendefinisikan fungsi untuk proses input array
void InputArray(int A[] , int N) {
  for (int C=0; C<N; C++) {
     cout<<"Masukkan nilai A["<<C<<"] : "; cin>>A[C];
  }
}

// Mendefinisikan fungsi untuk menghitung jumlah (sum)
// dari semua elemen array
long Jumlah(int A[], int N) {
  long jml = 0;
```

```
  // Menjumlahkan semua elemen array
  for (int C=0; C<N; C++) {
     jml += A[C];
  }
  return jml;
}

// Fungsi utama
int main() {
  int X[100];     // Elemen maksimal adalah 100
  int BanyakElemen;
  long HASIL;

  cout<<"Masukkan banyaknya elemen yang diinginkan : ";
  cin>>BanyakElemen;
  cout<<endl;

  // Memanggil fungsi InputArray
  InputArray(X, BanyakElemen);

  // Memanggil fungsi Jumlah dan menampung hasilnya
  // ke variabel HASIL
  HASIL = Jumlah(X, BanyakElemen);

  // Menampilkasn hasil
  cout<<"\nHasilnya = "<<HASIL;

  retun 0;
}
```

Tugas Mandiri

1. Buatlah program untuk menghitung operasi 2 buah bilangan yang disusun menjadi beberapa fungsi berdasarkan sifat operatornya. Misalnya, fungsi untuk memasukkan data, kemudian dibuat fungsi untuk menjumlahkan, fungsi untuk mengurangkan, fungsi untuk membagi, fungsi untuk mengalikan dan fungsi untuk menampilkan. Keenam fungsi itu dikendalikan ke dalam fungsi utama menggunakan sistem menu.
2. Buatlah program untuk mencari nilai tertinggi, nilai terendah, rata-rata, dan standar deviasi dalam bentuk fungsi-fungsi.
3. Buatlah program untuk mengurutkan nilai data yang diinputkan melalui keyboard menggunakan salah satu metode pengurutan. Bagilah setiap proses menjadi fungsi-fungsi.

BAB IX
STRUKTUR

Kompetensi dan Indikator

Kompetensi Dasar :
- Mengetahui *struct*.
- Mengetahui penulisan *struct*.
- Mengetahui penggunaan dan manfaat *struct* pada aplikasi.

Indikator Keberhasilan :
- Mampu menjelaskan tentang *struct*.
- Mampu menuliskan program menggunakan *struct*.
- Mampu menggunakan fungsi pada contoh *struct*.

Dasar Teori

A. Pengantar

Struct (atau yang dikenal dengan struktur) adalah sekumpulan variabel yang masing-masing dapat berbeda tipe, dan dikelompokkan ke dalam satu nama (menurut Pascal, struktur juga dikenal sebagai record). Struktur membantu mengatur data-data yang rumit, khususnya dalam program yang besar, karena struktur membiarkan sekelompok variabel diperlakukan sebagai satu unit daripada sebagai entity yang terpisah.

Salah satu contoh struktur tradisional adalah record daftar gaji karyawan, dimana karyawan digambarkan dengan susunan lambang seperti nama, alamat, nomor jaminan sosial, gaji dan sebagainya. Beberapa dari lambang tersebut biasanya berupa struktur, nama mempunyai komponen begitu juga alamat dan gaji.

Struktur ini sering digunakan untuk mendefinisikan suatu record data yang disimpan di dalam file. Struktur termasuk ke dalam tipe data yang dibangkitkan (derived data type), yang disusun dengan menggunakan obyek tipe lain.

Contoh :

```
struct mhs
{
        char *nama;
        char *nim;
        int tts, tas;
float akhir;
        char aksara;
}
```

Kata kunci struct menunjukkan definisi struktur, dan identitas mhs menunjukkan structure tag. Dengan demikian terdapat tipe data baru bernama struct mhs, yang terdiri dari nama mahasiswa, nilai tes tengah semester, tes akhir semester, nilai akhir, dan huruf aksara, yang masing-masing disebut dengan field.

Dapat dituliskan dengan :

struct mhs x, y[100], *z;

Variabel x adalah variabel tunggal, y adalah variabel array dengan 100 lokasi memori, dan z adalah variabel pointer, yang kesemuanya masing-masing berisi field di atas. Jadi, variabel y adalah daftar nama, nilai tts, tas, akhir, dan huruf aksara dari 100 mahasiswa.

Sehingga dapat ditulis :

```
struct mhs
{
        char *nama;
        char *nim;
        int tts, tas;
        float akhir;
        char aksara;
} x, y[100], *z;
```

Inisialisasi juga dapat dilakukan dengan :

struct mhs x = { “Garfield”, 80, 60, 76.8, „A‟ };

Untuk mengakses anggota dari struktur digunakan salah satu dari dua operator, yaitu operator titik (.), atau operator panah (->) tergantung tipe variabel yang dideklarasikan. Jika variabel tunggal (misal x) maka digunakan operator titik, sedangkan jika variabel pointer (misal z) digunakan operator panah.

Contoh :

```
printf (“%s”, x.nama);
printf (“%s”, z->nama);
```

B. DASAR STRUKTUR

Misal ada permasalahan grafis yang melibatkan koordinat x dan y. Objek dasar yang akan dibuat struktur adalah titik koordinatnya, yang diasumsikan sebagai koordinat x dan y dan keduanya bilangan bulat.

Deklarasi dari koordinat x dan y adalah :

```
struct point
{
        int x;
        int y;
};
```

kata kunci struct mengenalkan deklarasi struktur yang mana deklarasi list terlampir di kurung kurawal { }. Nama pilihan yang disebut structure tag mengikuti kata struct.

Deklarasi struktur yang tidak diikuti oleh variabel list tidak menyediakan tempat penyimpanan; deklarasi struktur hanya menjelaskan template atau bentuk struktur. Kalau deklarasi di tag, tag dapat digunakan dalam definisi contoh struktur. Sebagai contoh, memberikan deklarasi point diatas.

struct point pt;

Variabel pt yang berupa struktur tipe struct poin. Sebuah struktur dapat diletakan di depan dengan mengikuti difinisinya dengan daftar inisialisasi, masing-masing adalah lambang konstanta.

Praktek

C. STRUKTUR DAN FUNGSI

Operasi yang sering diterapkan pada struktur adalah proses menyalin atau menunjukkan struktur sebagai unit, menggunakan alamatnya dan mengakses anggotanya. Copy dan assignment mencakup memberi argumen ke fungsi dan menghasilkan nilai dari fungsinya juga. Struktur tidak bisa dibandingkan.

Struktur dapat diletakkan di awal oleh daftar value konstanta dan otomatis juga dapat ditempatkan di awal oleh operasi assignment. Sebuah struktur otomatis mungkin juga diletakkan di depan oleh tugas atau oleh panggilan fungsi yang menghasilkan struktur jenis yang tepat.
Untuk menghubungkan nama struktur dan nama anggota digunakan simbol “.”
Contoh :

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

struct time {
int jam;
int min;
};

struct rencana {
struct time awal;
struct time akhir;
int y;
int z;
};
struct rencana kerja = { 11,22,33,44,5,6 };

funct(struct rencana oo);

main()
{
kerja.akhir.min = 40;
```

```
kerja.z = 66;
cout << "proses main sebelum ke fungsi \n"<<kerja.awal.jam<<" "<<kerja.awal.min<<"
"<<kerja.akhir.jam<<" "<<kerja.akhir.min<<" "<<kerja.y<<" "<<kerja.z<<endl;

funct(kerja);      /* pengiriman struktur kerja ke fungsi */

cout <<"proses main sesudah ke fungsi\n"<<kerja.awal.jam<<" "<<kerja.awal.min<<"
"<<kerja.akhir.jam<<" "<<kerja.akhir.min<<" "<<kerja.y<<" "<<kerja.z<<endl;
getch();
}

funct(struct rencana oo)
/* nilai struktur kerja disalinkan ke oo */
{
cout <<"dalam fungsi (a)\n"<<oo.awal.jam<<" "<<oo.awal.min<<" "<<oo.akhir.jam<<"
"<<oo.akhir.min<<" "<<oo.y<<" "<<oo.z<<endl;
oo.awal.jam = 111;
oo.y = 555;       /* ubah nilai dalam fungsi */

cout <<"dalam fungsi (a)\n"<<oo.awal.jam<<" "<<oo.awal.min<<" "<<oo.akhir.jam<<"
"<<oo.akhir.min<<" "<<oo.y<<" "<<oo.z<<endl;
}
```

Bila program dijalankan maka :

```
proses main sebelum ke fungsi
11  22  33  40  5  66
dalam fungsi (a)
11  22  33  40  5  66
dalam fungsi (a)
111  22  33  40  555  66
proses main sesudah ke fungsi
11  22  33  40  5  66
```

D. ARRAY DALAM STRUKTUR

Array disini berfungsi untuk menyimpan nama dan bilangan bulat yang akan digunakan dalam proses perhitungan.

Contoh:

```
#include <iostream.h>
#include <string.h>
#include <stdlib.h>

struct movies_t {
        char title [50];
        int year;
} mine, yours;

void printmovie (movies_t movie);

int main ()
{
```

```
        char buffer [50];

        strcpy (mine.title, "Finding Nemo");
        mine.year = 2003;

        cout << "Masukkan judul film favorit: ";
        cin.getline (yours.title,50);
        cout << "Masukkan tahun: ";
        cin.getline (buffer,50);
        yours.year = atoi (buffer);

        cout << "Judul film favorit yang ada:\n ";
        printmovie (mine);
        cout << "Judul film favorit kamu adalah:\n ";
        printmovie (yours);
        return 0;
}

void printmovie (movies_t movie)
{
        cout << movie.title;
        cout << " (" << movie.year << ")\n";
}
```

Bila program diatas dijalankan maka hasilnya adalah :

Dapat dilihat deklarasi array dalam struktur terletak pada char title[50]; dimana dalam title dapat menyimpan judul film sebanyak 50 tempat dan title ini terletak pada :

```
Masukkan judul file favorit : Avp
Masukkan tahun : 2004
Judul film favorit yang ada :
Finding Nemo (2003)
Judul film favorit kamu adalah :
Avp (2004)
```

```
struct movies_t {
        char title [50];
        int year;
} mine, yours;
```

Contoh :

```
// array of structures
#include <iostream.h>
#include <stdlib.h>

#define N_MOVIES 5

struct movies_t {
        char title [50];
        int year;
```

```
} films [N_MOVIES];

void printmovie (movies_t movie);

int main ()
{
        char buffer [50];
        int n;
        for (n=0; n<N_MOVIES; n++)
        {
                cout << "Masukkan judul film: ";
                cin.getline (films[n].title,50);
                cout << "Masukkan tahun : ";
                cin.getline (buffer,50);
                films[n].year = atoi (buffer);
        }
        cout << "\nFilm yang menjadi favorit kamu:\n";
        for (n=0; n<N_MOVIES; n++)
                printmovie (films[n]);
        return 0;
}

void printmovie (movies_t movie)
{
        cout << movie.title;
        cout << " (" << movie.year << ")\n";
}
```

Bila program diatas dijalankan maka hasilnya adalah :

```
Masukkan judul file favorit : Matrix
Masukkan tahun : 2002
Masukkan judul file favorit : Alien Vs Predator
Masukkan tahun : 2004
...
Film yang menjadi favorit kamu :
Matrix (2002)
Alien Vs Predator (2004)
```

Deklarasi array dalam struktur terletak pada char title [50]; dan #define N_Movies 5, sehingga 5 buah data film harus dimasukkan.

E. POINTER DALAM STRUKTUR

Misalkan sebuah pointer yaitu ptpelajar, yang menunjuk kepada sebuah data yang mempunyai struktur PELAJAR seperti berikut :

struct PELAJAR *ptpelajar;

Seperti pada pointer yang lain, deklarasi tersebut tidak menyediakan sebarang tempat untuk record PELAJAR. Perlu dibuat record baru yang fungsinya menggantikan pointer. Misalnya pelajar_baru.

ptpelajar = &pelajar_baru;

Dengan kondisi tersebut, pointer ptpelajar boleh digunakan untuk menggantikan tempat alamat pelajar_baru, dan pointer ptpelajar ini ditunjukkan dengan menggunakan simbol ->.

Contoh :

```
ptpelajar->nama = Jill Valentine;
ptpelajar->nim = 672009001;
ptpelajar->fakultas = Teknologi Informasi;
ptpelajar->tahun = 2009;
ptpelajar->alamat = Salatiga;
```

Sama dengan :

```
*ptpelajar.nama = Jill Valentine;
*ptpelajar.nim = 672009001;
*ptpelajar.fakultas = Teknologi Informasi;
*ptpelajar.tahun = 2009;
*ptpelajar.alamat = Salatiga;
```

Contoh :

```
#include <iostream.h>
struct time {
        int jam;
        int min;
};
struct rencana {
        struct time *awal;    /* penunjuk bagi struktur */
        struct time *akhir;
};
struct time jk = { 1,2 };
struct time kl = { 3,4 };
struct rencana kerja = { &jk,&kl };

main()
{
        kerja.akhir->min = 37;
        cout << kerja.awal->jam << " " <<  kerja.awal->min << " "
        << kerja.akhir->jam << " " << kerja.akhir->min << endl;
}
```

Bila program diatas dijalankan maka hasilnya adalah :

1 2 3 37

F. STRUKTUR YANG MENUNJUK DIRINYA SENDIRI

Struktur yang mempunyai sifat menunjuk dirinya sendiri dapat dilihat pada contoh deklarasi berikut ini :

```
struct tnode {
        char *perkataan;
        int *perkiraan;
        struct tnode *kiri;
        struct tnode *kanan;
};
```

1. TYPEDEF

Kata kunci typedef merupakan mekanisme untuk membuat sinonim atau alias dari tipe data yang telah didefinisikan sebelumnya.

Contoh :

typedef struct mhs MHS;

Dari deklarasi tersebut dapat didefinisikan sebuah tipe data baru bernama MHS sebagai sinonim untuk struct mhs. Pernyataan struct mhs dapat diganti dengan MHS saja.

Contoh :

typedef int PANJANG;

Jenis PANJANG boleh digunakan untuk deklarasi variabel yang lain.

Contoh :

```
PANJANG len, maxlen;
PANJANG *lengths[];
```

Sama dengan :

int len, maxlen;
int *lengths[];

Contoh :

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

main()
{
static struct s1 {
char c[4], *s;
} s1 = { "abcd", "fghi" };

static struct s2 {
char *cp;
struct s1 ss1;
} s2 = { "klmn", { "pqrs", "uvwxy" } };

cout<<"s1.c[2] = "<<s1.c[2]<<", "<<"*s1.s = "<<*s1.s<<endl;
cout<<"s1.c = "<<s1.c<<", "<<"s1.s = "<<s1.s<<endl;
cout<<"s2.cp = "<<s2.cp<<", "<<"s2.ss1.s = "<<s2.ss1.s<<endl;
cout<<"++s2.cp = "<<++s2.cp<<", "<<"++s2.ss1.s = "<<++s2.ss1.s;
getch();
}
```

Bila program tersebut dijalankan maka :

```
s1.c[2] = c, *s1.s = f
s1.c = abcd s1.s = fghi
s2.cp = klmn   s2.ss1.s = uvwxy
++s2.cp = lmn   ++s2.ss1.s = vwxy
```

Penggunaan typedef terletak pada variabel s1 dan s2 yang mempunyai panjang karakter yang berbeda dan nantinya pada waktu dicetak akan menghasilkan output sesuai kondisi pada program.

2. UNION

Sama seperti struct, union juga merupakan tipe data yang dibangkitkan, dimana anggotanya menggunakan secara bersama ruang penyimpanan memori yang sama, berbeda dengan struktur yang masing-masing variabel menempati lokasi memori yang berbeda.

Jumlah bytes yang digunakan untuk menyimpan union adalah sedikitnya cukup untuk menyimpan data terbesar yang ditangani. Tipe union umumnya digunakan untuk menangani satu, dua, atau tiga variabel dengan tipe yang mirip.

Contoh :

```
#include <iostream.h>
typedef union int_or_float {
        int n;
        float x;
} number;

main()
{
        number temp;
        temp.n = 4444;
        cout << "(a) temp.n = " << temp.n << " temp.x = " << temp.x << endl;
        temp.x = 4444.0;
        cout << "(b) temp.n = " << temp.n << " temp.x = " << temp.x << endl;
        temp.n = 4444;
        cout << "(c) temp.n = " << temp.n << " temp.x = " << temp.x << endl;
}
```

Bila program dijalankan, maka :

```
(a) temp.n = 4444   temp.x = 0.574484
(b) temp.n = -8192   temp.x = 4444
(c) temp.n = 4444   temp.x = 4418.169922
```

Output pada baris (b), nilai temp.n berubah karena tempatnya telah digunakan untuk menempatkan temp.x. Hal sebaliknya terjadi kepada nilai temp.x pada baris (c).

Output program akan berbeda jika deklarasi union tadi diganti dengan deklarasi struktur berikut :

```
typedef struct int_or_float {
        int n;
        float x;
} number;
```

Hasilnya :

```
(a) temp.n = 4444   temp.x = -0.00000
(b) temp.n = 4444   temp.x = 4444.00000
(c) temp.n = 4444   temp.x = 4444.00000
```

C++ menyediakan tipe data yang dapat didefinisikan oleh pemrogram disebut dengan enumerasi. Enumerasi, didefinisikan dengan menggunakan kata kunci enum, adalah sekumpulan konstanta integer yang direpresentasikan dengan identikasi tertentu.

Nilai dalam enum dimulai dari 0, dapat diubah dengan nilai lainnya, dan menaik dengan penambahan 1 untuk nilai selanjutnya.

Contoh :

```
enum bulan {JAN, PEB, MAR, APR, MEI, JUN, JUL, AGU, SEP, OKT, NOP, DES};
```

Deklarasi tersebut akan menciptakan tipe baru yaitu enum bulan, yang secara otomatis menunjukkan deret nilai 0 untuk JAN hingga 11 untuk DES.
Nilai bulan ini dapat diubah menjadi 1 hingga 12 dengan cara :

```
enum bulan {JAN = 1, PEB, MAR, APR, MEI, JUN, JUL, AGU, SEP, OKT, NOP, DES};
```

Contoh :

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

enum bulan {JAN = 1, PEB, MAR, APR, MEI, JUN, JUL, AGU, SEP, OKT, NOP, DES};

main() {
        enum bulan Bulan;
char *namaBulan[] = {"", "Januari", "Pebruari", "Maret", "April", "Mei", "Juni", "Juli",
"Agustus", "September", "Oktober", "Nopember", "Desember" };

        for ( Bulan = JAN ; Bulan <= 12 ; Bulan++ )
        cout<<"Bulan ke - "<< Bulan <<"  "<< namaBulan[Bulan]<< endl;
getch();
return 0;
}
```

BAB X
SORTING

Kompetensi dan Indikator

Kompetensi Dasar :
- Mengetahui dasar-dasar algoritma pengurutan.
- Mengetahui penggunaan masing-masing algoritma.
- Mengetahui efisiensi masing-masing algoritma.

Indikator Keberhasilan :
- Mampu menjelaskan dasar-dasar algoritma pengurutan.
- Mampu menuliskan program dengan menterjemahkan algoritma ke dalam bahasa pemrograman.
- Mampu membandingkan efisiensi algoritma yang satu dengan lainnya.

Dasar Teori

A. Pengantar

Sort adalah suatu proses pengurutan data yang sebelumnya disusun secara acak atau tidak teratur menjadi urut dan teratur menurut suatu aturan tertentu.

Biasanya pengurutan terbagi menjadi 2 yaitu :
- Ascending (pengurutan dari karakter / angka kecil ke karakter / angka besar)
- Descending (pengurutan dari karakter / angka besar ke karakter / angka kecil)

Ada banyak cara yang dapat dilakukan untuk melakukan proses pengurutan dari paling tinggi ke paling rendah atau sebaliknya. Untuk masalah pengurutan pada array kita tidak dapat langsung menukar isi dari variabel yang ada, tetapi menggunakan metode penukaran (swap).

Contoh :
Data yang terdiri dari nilai dengan array, nilai[1] = 10 dan nilai[2] = 8 akan ditukar isi nilainya sehingga akan menghasilkan nilai[1] = 8 dan nilai[2] = 10.

Proses penukaran tidak dapat langsung dilakukan dengan cara :

```
nilai[1] = nilai[2];
nilai[2] = nilai[1];
```

Cara yang tepat adalah :

```
temp = nilai[1];
nilai[1] = nilai[2];
nilai[2] = temp;
```

Praktek

B. METODE SORTING

1. Bubble Sort

Bubble sort adalah suatu metode pengurutan yang membandingkan elemen yang sekarang dengan elemen berikutnya. Apabila elemen yang sekarang lebih besar dari elemen yang berikutnya, maka posisinya akan ditukar, bila tidak maka posisi akan tetap.

Misalkan data sebagai berikut :

5, 34, 32, 25, 75, 42, 22, 2

Pengurutan akan dilakukan dengan mengurutkan 8 data dari yang terkecil sampai yang terbesar.

- Dimulai dengan dua angka pertama yaitu angka 5 dan 34, untuk pengurutan dari kecil ke besar maka akan diperoleh posisi tidak berubah karena angka 5 < angka 34.
- Langkah berikutnya akan membandingkan angka 34 dan 32. Karena angka 34 > 32 maka akan terjadi pertukaran posisi, sehingga hasil urutan datanya menjadi 5, 32, 34, 25, 75, 42, 22, 2.
- 34 > 25 = 25, 34 (berubah)
- 34 < 75 = 34, 75 (tetap)
- 75 > 42 = 42, 75 (berubah)
- 75 > 22 = 22, 75 (berubah)
- 75 > 2 = 2, 75 (berubah)

Hasil sementara menjadi 5, 32, 25, 34, 42, 22, 2, 75

Langkah kedua :

- 5 < 32 = 5, 32 (tetap)
- 32 > 25 = 25, 32 (berubah)
- 32 < 34 = 32, 34 (tetap)
- 34 < 42 = 34, 42 (tetap)
- 42 > 22 = 22, 42 (berubah)
- 42 > 2 = 2, 42 (berubah)
- 42 < 75 = 42, 75 (tetap)

Hasil sementara menjadi 5, 25, 32, 34, 22, 2, 42, 75

Langkah ketiga :

- 5 < 25 = 5, 25 (tetap)
- 25 < 32 = 25, 32 (tetap)
- 32 < 34 = 32, 34 (tetap)
- 34 > 22 = 22, 34 (berubah)
- 34 > 2 = 2, 34 (berubah)
- 34 < 42 = 34, 42 (tetap)
- 42 < 75 = 42, 75 (tetap)

Hasil sementara menjadi 5, 25, 32, 22, 2, 34, 42, 75

Sampai langkah terakhir yaitu langkah ketujuh :

- 5 > 2 = 2, 5 (berubah)
- 5 < 22 = 5, 22 (tetap)
- 22 < 25 = 22, 25 (tetap)
- 25 < 32 = 25, 32 (tetap)
- 32 < 34 = 32, 34 (tetap)
- 34 < 42 = 34, 42 (tetap)

- 42 < 75 = 42, 75 (tetap)

Hasil sementara menjadi 2, 5, 22, 25, 32, 34, 42, 75

Proses pengurutan data tersebut membutuhkan tujuh langkah atau tujuh kali perulangan.

Contoh program:

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

void main()
{
        int NumList[8] = {5, 34, 32, 25, 75, 42, 22, 2};
        int Swap;
        cout<<"Data sebelum diurutkan: \n";
        for(int ctr=0; ctr<8; ctr++)
        {
                cout<< setw( 3 ) <<NumList[ctr];
        }
        cout<<"\n\n";
        for(int i=0; i<7; i++)
                for(int ii=0; ii<7; ii++)
                        if (NumList[ii] > NumList[ii + 1])
                        {
                                Swap = NumList[ii];
                                NumList[ii] = NumList[ii + 1];
                                NumList[ii + 1] = Swap;
                        }
        cout<<"Data setelah diurutkan: \n";

        for (int iii=0; iii<8; iii++)
                cout<< setw( 3 ) << NumList[iii];
        cout<< endl <<endl;
}
```

Bila program tersebut dijalankan maka :

```
Data sebelum diurutkan :
        5, 34, 32, 25, 75, 42, 22, 2
Data setelah diurutkan :
        2, 5, 22, 25, 32, 34, 42, 75
```

Data pada program diatas akan diurutkan menurut ascending atau dari kecil ke besar.

2. Selection Sort

Selection sort adalah suatu metode pengurutan yang membandingkan elemen yang sekarang dengan elemen berikutnya sampai ke elemen yang terakhir. Jika ditemukan elemen lain yang lebih kecil dari elemen sekarang maka dicatat posisinya dan langsung ditukar.

Misalkan data sebagai berikut :

5, 34, 32, 25, 75, 42, 22, 2

Data tersebut akan diurutkan dengan ascending atau dari kecil ke besar.

Langkah pertama :

Posisi	:	1	2	3	4	5	6	7	8
Data	:	5	34	32	25	75	42	22	2

Pembanding	Posisi
• 5 < 34	1
• 5 < 32	1
• 5 < 25	1
• 5 < 75	1
• 5 < 42	1
• 5 < 22	1
• 5 > 2	8

Tukar data pada posisi 1 dengan data posisi 8.
Hasil sementara menjadi 2, 34, 32, 25, 75, 42, 22, 5

Langkah kedua :

Posisi	:	1	2	3	4	5	6	7	8
Data	:	2	34	32	25	75	42	22	5

Pembanding	Posisi
• 34 > 32	3
• 32 > 25	4
• 25 < 75	4
• 25 < 42	4
• 25 > 22	7
• 22 > 5	8

Tukar data pada posisi 2 dengan data posisi 8.
Hasil sementara menjadi 2, 5, 32, 25, 75, 42, 22, 34

Langkah ketiga :

Posisi	:	1	2	3	4	5	6	7	8
Data	:	2	5	32	25	75	42	22	34

Pembanding	Posisi
• 32 > 25	4

- 25 > 75 4
- 25 < 42 4
- 25 > 22 7
- 22 < 34 7

Tukar data pada posisi 3 dengan data posisi 7.
Hasil sementara menjadi 2, 5, 22, 25, 75, 42, 32, 34

Langkah keenam (langkah terakhir) :

Posisi	:	1	2	3	4	5	6	7	8
Data	:	2	5	22	25	32	34	75	42

Pembanding Posisi

- 75 > 42 8

Tukar data pada posisi 7 dengan data posisi 8.
Hasil sementara menjadi 2, 5, 22, 25, 32, 34, 42, 75

Proses pengurutan data tersebut membutuhkan enam langkah atau enam kali perulangan.

Contoh program:

```
void SelectionSort (int Array[ ], const int Size){
        int i, j, smallest, temp;
        for (i=0; i<Size; i++)
        {
                smallest=i;
                for (j=i; j<Size; j++)                  {
                        if (array[smallest]>array[j])
                        {
                                smallest=j;
                        }
                }
                temp=array[i];
                array[i]=array[smallest];
                array[smallest]=temp;
        }
}
```

3. Quick Sort

Quick sort adalah suatu metode pengurutan yang membandingkan suatu elemen (pivot) dengan elemen yang lain dan menyusunnya sedemikian rupa sehingga elemen yang lain yang lebih kecil daripada pivot terletak disebelah kiri pivot sedangkan elemen yang lebih besar dari pivot diletakkan di sebelah kanan pivot.
Sehingga akan terbentuk dua sub list yaitu yang teletak disebelah kiri pivot dan sebelah kanan pivot.

List yang sebelah kiri pivot juga diterapkan aturan seperti pivot, yaitu membandingkan dengan elemen yang lain. Jika lebih kecil akan diletakkan di sebelah kiri, jika lebih besar akan diletakkan di sebelah kanan.

Contoh program:

```
void QuickSort (array A, int L, int N)
{
        if L<N
                M:=Partition (A, L, N)
                QuickSort (A, L, M-1)
                QuickSort (A, M+1, N)
        endif
}
void Partition (array A, int L, int N)
{
        select M, where L <= M <=N
                reorder A(L) ... A(N) so that I<M implies A(I) <= A(M), and I>M implies
                A(I) >= A(M)
        return M
}
```

Contoh :

Data yang akan diurutkan adalah : 20, 10, 15, 5, 8, 3
Bilangan yang terletak diantara kurung buka dan kurung tutup adalah pivot

i bergerak dari kiri ke kanan sampai mendapat nilai >= pivot
j bergerak dari kanan ke kiri sampai mendapat nilai < pivot

Langkah 1

posisi	1	2	3	4	5	6
data	20	10	(15)	5	8	3
	i					*j*

i berhenti pada posisi 1 karena langsung mendapatkan nilai yang lebih besar dari pivot (15) yaitu 20
j berhenti pada posisi 6 karena langsung mendapatkan nilai yang lebih kecil dari pivot (15) yaitu 3
karena i>j maka data yang ditunjuk oleh i ditukar dengan data yang ditunjuk j, sehingga data berubah menjadi :

3 10 15 5 8 20

Langkah 2

posisi	1	2	3	4	5	6
data	3	10	(15)	5	8	20
			i		*j*	

i berhenti pada posisi 3 karena tidak dapat menemukan nilai yang lebih besar dari pivot (15)

j berhenti pada posisi 5 karena langsung mendapatkan nilai yang lebih kecil dari pivot (15) yaitu 8

karena i>j maka data yang ditunjuk oleh i ditukar dengan data yang ditunjuk j, sehingga data berubah menjadi :

3 10 8 5 15 20

Langkah 3

posisi	1	2	3	4	5	6
data	3	10	(8)	5	15	20
		i		*j*		

i berhenti pada posisi 2 karena langsung mendapatkan nilai yang lebih besar dari pivot (8) yaitu 10

j berhenti pada posisi 4 karena langsung mendapatkan nilai yang lebih kecil dari pivot (8) yaitu 5

karena i>j maka data yang ditunjuk oleh i ditukar dengan data yang ditunjuk j, sehingga data berubah menjadi :

3 5 8 10 15 20

Hasil akhirnya adalah : 3 5 8 10 15 20

CONTOH LATIHAN :

Buatlah program untuk melakukan pengurutan data secara menaik dengan menukar data berikut menggunakan bubble sort.
Data : 102, 21, 83, 42, 11, 10, 9, 3, 20, 27, 15, 92, 2

program:

```
#include <iostream.h>

void SwapMembers (int items[], int index1, int index2)
{
        int temp;
        temp=items[index1];
        items[index1]=items[index2];
        items[index2]=temp;
}

main ()
{
        int n,m;
        int numbers [] = {102,21,83,42,11,10,9,3,20,27,15,92,2};

        const int size = sizeof(numbers) / sizeof(numbers[0]);
        for (n=size-1; n>0; n--)
                for (m=0; m<n; m++)
```

```
                        if ( numbers[m] > numbers[m+1] )
                                SwapMembers (numbers, m, m+1 );
        for (n=0;n<size;n++)
                cout << numbers[n] << ", ";
        return 0;
}
```

Bila program dijalankan maka :

2, 3, 9, 10, 11, 15, 20, 21, 27, 42, 83, 92, 102,

BAB XI
SEARCHING

Kompetensi dan Indikator

Kompetensi Dasar : - Mengetahui dasar-dasar pencarian.
- Mengetahui penggunaan algoritma pencarian.

Indikator Keberhasilan : - Mampu menjelaskan dasar-dasar pencarian.
- Mampu menuliskan program dengan menggunakan pencarian.
- Mampu membuat program aplikasi menggunakan teori pencarian.

Dasar Teori

Pencarian/ search merupakan suatu proses di mana mencari data pada suatu deretan obyek.

Seperti halnya pengurutan/sort, algoritma pencarian juga bermacam macam, beberapa di antaranya adalah:

- Squential Search
- Binary Search

A. Squential Search

Salah satu metode pencarian yang sering digunakan dan merupakan metode paling mudah adalah Squential Search, yaitu memandingkan satu-persatu antara data yang dicri dengan data yang ada pada array dari awal sampai akhir atau data yang dicari ditemukan. Metode pencarian squential ini tidak memandang apakah data sudah urut belum.

jika secara kebetulan data yang dicari berada di elemen-eleman awal dari array, maka proses pencarian cepat dilakukan. sebaliknya jika data yang dicari berada di akhir atau bahkan tidak ditemukan, maka proses pencarian menjadi lebih lama. jumlah langkah pembandingan jika data yang dicari tidak ditemukan adalah sejumlah elemen array (n).

Algoritma

1. masukkan array
2. ketahui jumlah elemen(n), mulai dari data pertama, i=1;
3. bandingkan array ke-i dengan data yang dicari. jika tidak sama, maka lanjutkan dengan menambahkan i=i+1
4. jika i>jumlah elemen array (n), maka pencarian dihentikan dan dianggap tidak ketemu dan selesai. jika i<=jumlah elemen array, maka ulangi dari langkah 3

Contoh:

Terdapat data sebagai berikut: {2,4,26,82,49}

Dari data ini akan dicari angka 4

Proses pencarian:

- Ketahui jumlah elemen (n), n=5, i=1

- Bandingkan data yang ada pada i (dt[i]) dengan data yang dicari. dt[i]=dt[1]=2 sedangkan data yang dicari=4, sehingga tidak sama, maka lanjutkan pencarikan ke denga cara menambahkan i, i=i+1=2
- i=2, sedangkan n=5, sehingga i<=n, pencarian dilanjutkan. Bandingkan data yang ada pada i (dt[i]) dengan data yang dicari. dt[i]=dt[2]=4 sedangkan data yang dicari=4, sehingga data yang dicari ketemu. proses pencarian selesai.

Jika yang dicari adalah angka 50,

Proses pencarian:

- Ketahui jumlah elemen (n), n=5, i=1
- Bandingkan data yang ada pada i (dt[i]) dengan data yang dicari. dt[i]=dt[1]=2 sedangkan data yang dicari=50, sehingga tidak sama, maka lanjutkan pencarikan ke data berikutnya dengan cara menambahkan i, i=i+1=2
- i=2, sedangkan n=5, sehingga i<=n, pencarian dilanjutkan. Bandingkan data yang ada pada i (dt[i]) dengan data yang dicari. dt[i]=dt[2]=4 sedangkan data yang dicari=50, sehingga tidak sama, maka lanjutkan pencarian ke data berikutnya dengan cara menambahkan i, i=i+1=3
- i=3, sedangkan n=5, sehingga i<=n, pencarian dilanjutkan. Bandingkan data yang ada pada i (dt[i]) dengan data yang dicari. dt[i]=dt[3]=26 sedangkan data yang dicari=50, sehingga tidak sama, maka lanjutkan pencarian ke data berikutnya dengan cara menambahkan i, i=i+1=4
- i=4, sedangkan n=5, sehingga i<=n, pencarian dilanjutkan. Bandingkan data yang ada pada i (dt[i]) dengan data yang dicari. dt[i]=dt[4]=82 sedangkan data yang dicari=50, sehingga tidak sama, maka lanjutkan pencarian ke data berikutnya dengan cara menambahkan i, i=i+1=5
- i=2, sedangkan n=5, sehingga i<=n, pencarian dilanjutkan. Bandingkan data yang ada pada i (dt[i]) dengan data yang dicari. dt[i]=dt[5]=49 sedangkan data yang dicari=50, sehingga tidak sama, maka lanjutkan pencarian ke data berikutnya dengan cara menambahkan i, i=i+1=6
- i=6, sedangkan n=5, sehingga i>n, maka pencarian dihentikan dan dinyatakan bahwa data yang dicari tidak ditemukan. proses selesai.

B. Binary Search

Secara umum. Untuk mencari suatu nila di array yang tidak urut, diperlukan pencarian dengan cara membandingkan saut persatu.sampai nilai yang dicari ketemu . apabila nilai yang dicari tidak ditemukan di dalam array, maka secara otomatis semua elemen array sudah dibandingkan satu persatu.

Berbeda secara signifikan ketika array sudah urut sebelumnya. Pencarian dapat dilakukan jauh lebih cepat. Salah satu cara pencarian dengan cepat pada array yang sudah uruta adalah pencarian secara biner (binary search).bisa dibayangkan, jika terdapat sebuah array dengan elemen sebanyak 1000000 (satu juta) elemen , maka untuk mencari suatu jnilai dalam array tersebut cukup 20 langkah pencarian yang dihitung dari log2(1000000).

Algoritma

5. masukkan array
6. urutkan array
7. ketahui jumlah elemen array (n), batas bawah(bb), batas atas (ba), dan titik tengah (tt) yang dihitung dengan cara bb=1 ba= jumlah elemen array, dan tt= int((ba+bb)/2)
8. cari nilai dari elemen tengah-tengah (tt)
9. jika nilai elemen tengah-tengah sama dengan yang dicari, maka **algoritma dihentikan dan dinyatakan pencarian ketemu dan selesai**.
10. jika tidak sama, ada 2 kemungkinan:

- jika nilai yang dicari < nilai elemen tengah, ini menunjukan bahwa nilai yang dicari jika ada pasti berada di bawah elemen tengah, sehingga pencarian dipersempit. ba=tt-1
- jika nilai yang dicari > nilai elemen tengah, ini menunjukan bahwa nilai yang dicari jika ada pasti berada di setelah elemen tengah, sehingga pencarian dipersempit. bb=tt+1

11. Ada 3 kriteria
    - jika bb>ba, maka **pencarian dihentikan dan dinyatakan tidak ketamu dan selesai**,
    - jika bb<=ba, ulangi dari langkah 3

Contoh:

Terdapat data sebagai berikut:

int dt[9]={2,4,6,8,9,21,40,45,87}

Dari data ini akan dicari angka 4

Proses pencarian:

- Ketahui jumlah elemen (n),batas bawah (bb), batas atas(ba) dan titik tengah(tt) n=9, bb=1, ba=9, tt= int((1+9)/2)=5
- Bandingkan data yang ada pada tt dengan yang dicari dt[5]=9, sedangkan yang dicari 4, sehingga 4<9, alias dinyatakan belum ketemu. lanjutkan ke langkah berikutnya
- karena data elemen tengah (dt[tt]) tidak sama dengan yang dicari yang dicari, maka ada 2 kemungkinan, data yang dicari<dt[tt] atau data yang dicari>dt[tt]. dalam hal ini, data yang dicari angka 4 dan data tengah dt[tt] adalah 9, sehingga: tidak mungkin data yang dicari berada di elemen kelima sampak akhir, sehingga batas atas(ba) diubah menjadi titik tengah (tt)-1, ba=5-1=4
- setelah dilakukan perubahan batas, tampak bahwa bb tidak berubah, yaitu bb=1, sedangkan ba=4, bb<ba, proses dilanjutkan untuk mencari nilai tengah yang baru (kembali ke langkah 3)
- tt=int((bb+ba)/2) = int((1+4) /2) = 2
- bandingkan antara data tengah (dt[tt]) dengan data yang dicari dt[tt]=dt[2]=4, sedangkan yang dicari=4, sehingga dt[tt]=data yang dicari. hal ini menunjukkan bahwa data yang dicari sudah ketemu. pencarian dihentikan dan dinyatakan selesai.

Jika yang dicari adalah angka 50,

Proses pencarian:

- Ketahui jumlah elemen (n),batas bawah (bb), batas atas(ba) dan titik tengah(tt) n=9, bb=1, ba=9, tt= int((1+9)/2)=5
- Bandingkan data yang ada pada tt dengan yang dicari dt[5]=9, sedangkan yang dicari 50, sehingga 50>9, alias dinyatakan belum ketemu. lanjutkan ke langkah berikutnya
- karena data elemen tengah (dt[tt]) tidak sama dengan yang dicari yang dicari, maka ada 2 kemungkinan, data yang dicari<dt[tt] atau data yang dicari>dt[tt]. dalam hal ini, data yang dicari angka 50 dan data tengah dt[tt] adalah 9, sehingga: tidak mungkin data yang dicari berada di elemen pertama sampak tengah (5), sehingga batas bawah(bb) diubah menjadi titik tengah (tt)+1, bb=5+1=6

- setelah dilakukan perubahan batas, tampak bahwa ba tidak berubah, yaitu ba=9, sedangkan bb=6, bb<ba, proses dilanjutkan untuk mencari nilai tengah yang baru (kembali ke langkah 3)
- tt=int((bb+ba)/2) = int((6+9) /2) = 7
- bandingkan antara data tengah (dt[tt]) dengan data yang dicari dt[tt]=dt[7]=40, sedangkan yang dicari=50, sehingga dt[tt]<data yang dicari. pencarian dilanjutkan
- karena data elemen tengah (dt[tt]) tidak sama dengan yang dicari yang dicari, maka ada 2 kemungkinan, data yang dicari<dt[tt] atau data yang dicari>dt[tt]. dalam hal ini, data yang dicari angka 50 dan data tengah dt[tt] adalah 40, sehingga: tidak mungkin data yang dicari berada di elemen pertama sampak tengah (5), sehingga batas bawah(bb) diubah menjadi titik tengah (tt)+1, bb=7+1=8
- setelah dilakukan perubahan batas, tampak bahwa ba tidak berubah, yaitu ba=9, sedangkan bb=8, bb<ba, proses dilanjutkan untuk mencari nilai tengah yang baru (kembali ke langkah 3)
- tt=int((bb+ba)/2) = int((8+9) /2) = 8
- bandingkan antara data tengah (dt[tt]) dengan data yang dicari dt[tt]=dt[8]=45, sedangkan yang dicari=50, sehingga dt[tt]<data yang dicari. pencarian dilanjutkan
- karena data elemen tengah (dt[tt]) tidak sama dengan yang dicari yang dicari, maka ada 2 kemungkinan, data yang dicari<dt[tt] atau data yang dicari>dt[tt]. dalam hal ini, data yang dicari angka 50 dan data tengah dt[tt] adalah 45, sehingga: tidak mungkin data yang dicari berada di elemen pertama sampak tengah (), sehingga batas bawah(bb) diubah menjadi titik tengah (tt)+1, bb=8+1=9
- setelah dilakukan perubahan batas, tampak bahwa ba tidak berubah, yaitu ba=9, sedangkan bb=9, bb=ba, proses dilanjutkan untuk mencari nilai tengah yang baru (kembali ke langkah 3)
- tt=int((bb+ba)/2) = int((9+9) /2) = 9
- bandingkan antara data tengah (dt[tt]) dengan data yang dicari dt[tt]=dt[9]=87, sedangkan yang dicari=50, sehingga dt[tt]>data yang dicari. pencarian dilanjutkan
- karena data elemen tengah (dt[tt]) tidak sama dengan yang dicari yang dicari, maka ada 2 kemungkinan, data yang dicari<dt[tt] atau data yang dicari>dt[tt]. dalam hal ini, data yang dicari angka 50 dan data tengah dt[tt] adalah 87, sehingga: sehingga batas bawah(bb) diubah menjadi titik tengah (tt)+1, bb=9+1=10
- setelah dilakukan perubahan batas, maka ba tidak berubah, yaitu 9, sedangkan bb=10, karena bb>ba, maka proses dihentikan dan dinyatakan tidak ketemu dan selesai.

Tugas Mandiri

- Buatlah sebuah program untuk mempraktikan pencarian dengan metode squential search
- Buat pula program untuk mempraktekkan metode Binary Search

BAB XII
STRUKTUR DATA DINAMIS

Kompetensi dan Indikator

Kompetensi Dasar :
- Mengetahui dasar-dasar struktur data dinamis.
- Mengetahui penggunaan struktur data dinamis.

Indikator Keberhasilan :
- Mampu menjelaskan dasar-dasar struktur data dinamis.
- Mampu menuliskan program dengan menggunakan struktur data dinamis.
- Mampu membuat program aplikasi menggunakan teori struktur data dinamis.

Dasar Teori

A. Pengantar

Konsep dasar struktur data dinamis adalah alokasi memori yang dilakukan secara dinamis. Pada konsep ini, terdapat suatu struktur yang disebut dengan struktur referensi diri (self-referential structure), mempunyai anggota pointer yang menunjuk ke struktur yang sama dengan dirinya sendiri.

Struktur data dinamis sederhana dapat dibagi menjadi empat jenis, yaitu :

1. Linked list
2. Stack
3. Queue
4. Binary tree

Definisi ini dapat dituliskan secara sederhana dengan struktur :

```
struct node {
        int info;
        struct node *nextPtr;
}
```

Struktur ini mempunyai dua anggota, yaitu bilangan bulat info dan pointer nextPtr. Variabel pointer nextPtr ini menunjuk ke struktur bertipe struct node, yang sama dengan deklarasi induknya.

Untuk membuat dan menangani struktur data dinamis dibutuhkan alokasi memori dinamis agar tersedia ruang bagi node-node yang ada. Fungsi malloc, free, dan operator sizeof banyak digunakan dalam alokasi memori dinamis ini.
Contoh :

```
typedef struct node NODE;
typedef NODE *NODEPTR;
NODEPTR newPtr = malloc (sizeof (NODE));
```

```
        newPtr -> info = 15;
        newPtr -> nextPtr = NULL;
```

Fungsi free digunakan untuk menghapus memori yang telah digunakan untuk node yang ditunjuk oleh pointer tertentu. Jadi free (newPtr); akan menghapus memori tempat node yang ditunjuk oleh newPtr.

B. DEFINISI LINKED LIST

Linked list adalah suatu cara untuk menyimpan data dengan struktur sehingga dapat secara otomatis menciptakan suatu tempat baru untuk menyimpan data yang diperlukan. Program akan berisi suatu struct atau definisi kelas yang berisi variabel yang memegang informasi yang ada didalamnya, dan mempunyai suatu pointer yang menunjuk ke suatu struct sesuai dengan tipe datanya.

Struktur dinamis ini mempunyai beberapa keuntungan dibanding struktur array yang bersifat statis. Struktur ini lebih dinamis, karena banyaknya elemen dengan mudah ditambah atau dikurangi, berbeda dengan array yang ukurannya bersifat tetap.
Manipulasi setiap elemen seperti menyisipkan, menghapus, maupun menambah dapat dilakukan dengan lebih mudah.

Contoh :

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>

typedef struct nod {
        int data;
        struct nod *next;
} NOD, *NODPTR;

void CiptaSenarai(NODPTR *s)
{
        *s = NULL;
}

NODPTR NodBaru(int m){
        NODPTR n;

        n = (NODPTR) malloc(sizeof(NOD));
        if(n != NULL) {
                n -> data = m;
                n -> next = NULL;
        }
        return n;
}

void SisipSenarai (NODPTR *s, NODPTR t, NODPTR p)
{
        if (p==NULL) {
                t -> next = *s;
                *s = t;
        }
        else {
                t -> next = p -> next;
```

```
                p ->next = t;
        }
}

void CetakSenarai (NODPTR s)
{
        NODPTR ps;
        for (ps = s; ps != NULL; ps = ps -> next)
                printf("%d -->", ps -> data);
        printf("NULL\n");
}

int main ()
{
        NODPTR pel;
        NODPTR n;

        CiptaSenarai(&pel);
        n = NodBaru(55);
        SisipSenarai(&pel, n, NULL);
        n= NodBaru(75);
        SisipSenarai(&pel, n, NULL);
        CetakSenarai(pel);
        return 0;
}
```

Bila program dijalankan maka :

75->55->NULL

C. TEKNIK DALAM LINKED LIST

Pengulangan Linked List

Secara umum pengulangan ini dikenal sebagai while loop. Kepala pointer (head pointer) dikopikan dalam variabel lokal current yang kemudian dilakukan perulangan dalam linked list. Hasil akhir dalam linked list dengan current!=NULL. Pointer lanjut dengan current=current->next.

Contoh :

```
        // Return the number of nodes in a list (while-loop version)
        int Length(struct node * head) {
                int count = 0;
                struct node* current = head;
                while (current != NULL) {
                        count++;
                        current=current->next;
                }
                return (count);
        }
```

Mengubah Pointer Dengan Referensi Pointer

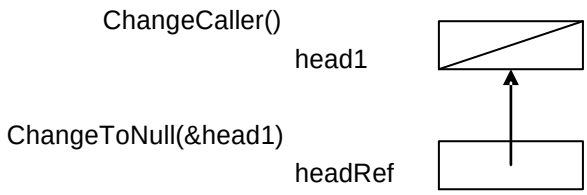
Banyak fungsi pada linked list perlu untuk merubah pointer kepala. Pointer ke pointer disebut dengan “reference pointer”. Langkah utamanya dalam teknik ini adalah :

- Merancang fungsi untuk mengambil pointer ke pointer kepala. Untuk melewati pointer ke “value of interest” yang dibutuhkan untuk diubah. Untuk mengubah struct node*, lewati struct node**.
- Gunakan „&" dalam panggilan untuk menghitung dan melewati pointer ke value of interest.
- Gunakan „*" pada parameter dalam fungsi pemanggil untuk mengakses dan mengubah value of interest.

Contoh :

```
void ChangeToNull (struct node** headRef)
*headRef=NULL;
void ChangeCaller() {
        struct node* head1;
        struct node* head2;
        ChangeToNull (&head1);
        ChangeToNull (&head2);
}
```

Fungsi sederhana tersebut akan membuat pointer kepala ke NULL dengan menggunakan parameter reference. Gambar berikut menunjukkan bagaimana pointer headRef dalam ChangeToNull menunjuk ke head1 pada Change Caller.



Gambar 1. Pointer headRef dalam ChangeToNull menunjuk ke head1

Membuat Kepala Senarai Dengan Perintah Push()

Cara termudah untuk membuat sebuah senarai dengan menambah node pada “akhir kepala” adalah dengan push().

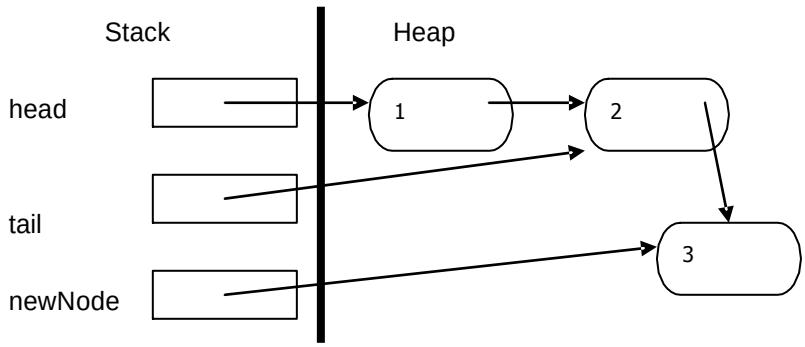
Contoh :

```
struct node* AddAtHead() {
        struct node* head = NULL;
        int i;
        for ( i=1; i<6; i++) {
                push (&head, i);
        }
        // head == { 5, 4, 3, 2, 1 };
        return (head);
}
```

Dalam membuat tambahan node pada akhir senarai (list) dengan menggunakan perintah Push(), jika terjadi kesalahan maka urutan akan terbalik.

Membuat Ekor Pada Akhir Senarai

Menambah ekor pada senarai akan melibatkan penempatan node terakhir, dan kemudian merubahnya .next field dari NULL untuk menunjuk node baru seperti variabel tail dalam contoh yaitu menambah node 3 ke akhir daftar {1,2}



Gambar 2. Membuat ekor pada akhir senarai

Untuk memasukkan atau menghapus node di dalam senarai, pointer akan dibutuhkan ke node sebelum posisi, sehingga akan dapat mengubah .next field.

Agar dapat menambahkan node di akhir ekor pada data senarai {1, 2, 3, 4, 5}, dengan kesulitan node pertama pasti akan ditambah pada pointer kepala, tetapi semua node yang lain dimasukkan sesudah node terakhir dengan menggunakan pointer ekor. Untuk menghubungkan dua hal tersebut adalah dengan menggunakan dua fungsi yang berbeda pada setiap permasalahan.
Fungsi pertama adalah menambah node kepala, kemudian melakukan perulangan yang terpisah yang menggunakan pointer ekor untuk menambah semua node yang lain. Pointer ekor akan digunakan untuk menunjuk pada node terakhir, dan masing-masing node baru ditambah dengan tail->next.

Contoh :

```
struct node* BuildWithSpecialCase () {
        struct node* head = NULL;
        struct node* tail;
        int i;
        push (&head, 1);
        tail = head;
        for (i=2; i<6; i++) {
                push (&(tail->next), i);
                tail = tail->next;
        }
        return (head);
}
```

Membuat Referansi Lokal

Dengan menggunakan “reference pointer” lokal, pointer akan selalu menunjuk ke pointer terakhir dalam senarai. Semua tambahan pada senarai dibuat dengan reference pointer. Reference pointer tidak menunjuk ke pointer kepala, tetapi menunjuk ke .next field didalam node terakhir dalam senarai.

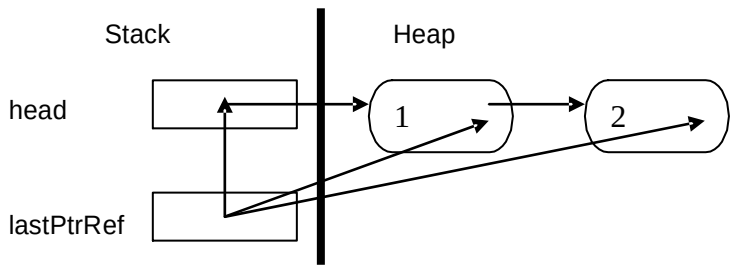
Contoh :

```
struct node* BuildWithLocalRef() {
        struct node* head = NULL;
        struct node** lastPtrRef=&head;
        int i;

        for (i=2; i<6; i++) {
                Push (lastPtrRef, i);
                lastPtrRef=&((*lastPtrRef)->next);
        }
        return (head);
}
```

Cara kerja teknik ini adalah :

- Pada puncak putaran, lastPtrRef menunjuk pointer terakhir dalam senarai. Pada permulaannya menunjuk ke pointer kepala itu sendiri. Berikutnya menunjuk ke .next field di dalam node terakhir dalam senarai.
- Push(lastPtrRef); menambah node baru pada pointer akhir. Node baru menjadi node terakhir dalam senarai.
- lastPtrRef=&((*lastPtrRef)->next; lastPtrRef lanjut ke .next field dan .next field sekarang menjadi pointer terakhir dalam senarai.



Gambar 3. Membuat ekor pada akhir senarai

D. Operasi dalam Link List

Menambah Node Baru

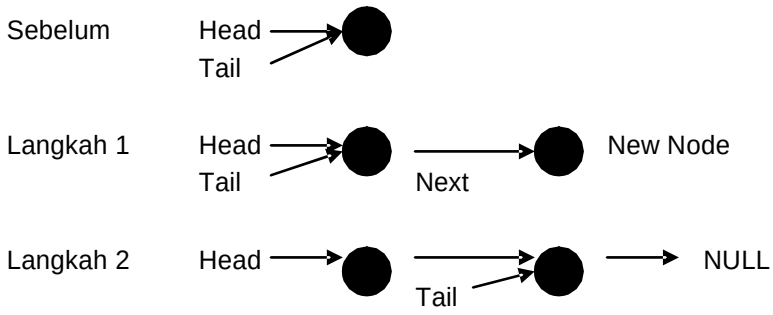
Untuk menambahkan sebuah node di dalam senarai, perlu didefinisikan sebuah kepala (head) dan ekor (tail). Pada awal senarai, posisi kepala dan ekor masih menjadi satu. Setelah ada tambahan node yang baru, maka posisinya berubah. Posisi kepala tetap pada awal senarai dan posisi ekor pada akhir senarai atau node yang baru. Bila ada tambahan

node, maka posisi ekor akan bergeser sampai ke belakang dan akhir senarai ditunjukkan dengan NULL.

Contoh :

```
void SLList::AddANode()
{
        Tail->Next = new List;
        Tail=Tail->Next;
}
```

Ilustrasi penambahan Node adalah sebagai berikut :



Menghapus Node

Berikut ini adalah contoh program untuk menghapus node :

```
void SLList::DeleteANode(ListPtr corpse)
{
        ListPtr temp;
        if (corpse==Head) {
                temp=Head;
                Head=Head->Next;
                delete temp;
        }
        else if (corpse==Tail) {
                temp=Tail;
                Tail=Previous(Tail);
                Tail->Next=NULL;
                delete temp;
        }
        else {
                temp=Previous(corpse);
                temp->Next=corpse->Next;
                delete corpse;
        }
        CurrentPtr=Head;
}
```

Praktek

Buatlah program untuk memasukkan beberapa data dalam sebuah senarai (linked list), jika akan mengakhiri tekan n maka akan muncul semua node yang masuk ke dalam linked list tersebut.

Contoh program :

```
#include <stdio.h>
#include <conio.h>
#include <ctype.h>
#include <stdlib.h>

typedef struct node {
        int lkey;
        char name[10];
        struct node* next;
} TNODE;

TNODE *first, *last;

int LoadNode(TNODE *p);
void FreeNode(TNODE *p);
void PrintNode(TNODE *p);

void CreateList()
{
        TNODE *p;
        int n=sizeof(TNODE);

        first=last=0;
        for(;;)
        {
                if( (p=(TNODE*)malloc(n))==0 )
                {
                        printf("\nmemori tidak cukup");
                        break;
                }
                if(LoadNode(p)!=1)
                {
                        FreeNode(p);
                        break;
                }
                p->next=0;
                if (first==0)
                        first=last=p;
                else {
                        last->next=p;
```

```
                        last=p;
                }
        }
}

int LoadNode(TNODE *p)
{
        char opt;
        printf("\nMasukkan node baru?");
        opt=getche();
        opt=toupper(opt);
        if(opt!='N') {
                puts("\nMasukkan data untuk node:");
                printf("\nlkey:\t");
                if (scanf("%d",&(p->lkey))!=1) return 0;

                printf("\nname:\t");if (scanf("%s",p->name)!=1)
return 0;
                return 1;
        }
        else
                return -1;
}

void FreeNode(TNODE *p) {
        free(p);
}

void ViewAllList()
{
        TNODE *p;
        p=first;
        while(p) {
                PrintNode(p);
                p=p->next;
        }
}

TNODE* FindNode(int key)
{
        TNODE *p;
        p=first;
        while(p) {
                if(p->lkey == key) return p;
                p=p->next;
        }
        return 0;
```

```
}

void PrintNode(TNODE *p)
{
	if(p) printf("\n%d\t%s",p->lkey,p->name);
}

TNODE* InsertBeforeFirst()
{
	TNODE *p;
	int n=sizeof(TNODE);
	if (((p=(TNODE*)malloc(n))!=0) && (LoadNode(p)==1))
	{
		if (first==0) {
			p->next=0;
			first=last=p;
		}
		else {
			p->next=first;
			first=p;
		}
		return p;
	}
	if(p==0)
		printf("\nMemori tidak cukup");
	else
		FreeNode(p);
	return 0;
}

TNODE* InsertBeforeKey(int key)
{
	TNODE *p, *q, *q1;
	int n=sizeof(TNODE);

	q1=0;
	q=first;
	while(q) {
		if(q->lkey == key) break;
		q1=q;
		q=q->next;
	}
	if(q==0) {
		printf("\nTidak ada node yang mempunyai kunci atau senarai kosong");
		return 0;
	}
	if (((p=(TNODE*)malloc(n))!=0) && (LoadNode(p)==1)) {
```

```
                if(q==first) {
                        p->next=first;
                        first=p;
                }
                else {
                        p->next=q;
                        q1->next=p;
                }
                return p;
        }
        if(p==0)
                printf("\nMemori tidak cukup");
        else
                FreeNode(p);
        return 0;
}
TNODE* InsertAfterKey(int key)
{
        TNODE *p, *q;
        int n=sizeof(TNODE);

        q=first;
        while(q) {
                if(q->lkey == key) break;
                q=q->next;
        }
        if(q==0) {
                printf("\nTidak ada node yang mempunyai kunci atau senarai kosong");
                return 0;
        }
        if (((p=(TNODE*)malloc(n))!=0) && (LoadNode(p)==1))
        {
                if(q==last) {
                        p->next=0;
                        last->next=p;
                        last=p;
                }
                else {
                        p->next=q->next;
                        q->next=p;
                }
                return p;
        }
        if(p==0)
                printf("\nMemori tidak cukup");
        else
                FreeNode(p);
```

```
	return 0;
}

TNODE* InsertAfterLast()
{
	TNODE *p;
	int n=sizeof(TNODE);
	if (((p=(TNODE*)malloc(n))!=0) && (LoadNode(p)==1))
	{
		p->next=0;
		if (first==0)
			first=last=p;
		else {
			last->next=p;
			last=p;
		}
		return p;
	}
	if(p==0)
		printf("\nMemori tidak cukup");
	else
		FreeNode(p);
	return 0;
}


void RemoveFirst()
{
	TNODE *p;

	if(first==0)
		return;
	if(first==last) {
		FreeNode(first);
		first=last=0;
		return;
	}
	p=first;
	first=first->next;
	FreeNode(p);
}

void RemoveLast()
{
	TNODE *p, *q;

	if(first==0)
```

```
                return;
        if(first==last) {
                FreeNode(first);
                first=last=0;
                return;
        }
        q=0;
        p=first;
        while(p!=last)    {
                q=p;
                p=p->next;
        }
        p=last;
        FreeNode(p);
        q->next=0;
        last=q;
}

void RemoveByKey(int key)
{
        TNODE *p, *q;

        if(first==0)
                return;
        q=0;
        p=first;
        while(p) {
                if(p->lkey == key) break;
                q=p;
                p=p->next;
        }
        if(!p) {
                printf("\nTidak ada node yang mempunyai kunci");
                return;
        }

        if(first==last) {
                FreeNode(first);
                first=last=0;
                return;
        }
        if(p==first) {
                first=first->next;
                FreeNode(p);
                return;
        }
        if(p==last) {
```

```
                q->next=0;
                last=q;
                FreeNode(p);
                return;
        }
        q->next=p->next;
        FreeNode(p);
}

void DeleteList()
{
        TNODE *p;

        p=first;
        while(p) {
                first=first->next;
                FreeNode(p);
                p=first;
        }
        last=0;
}

void main()
{
        CreateList();
        ViewAllList();
        InsertAfterLast();
        ViewAllList();
        RemoveFirst();
        ViewAllList();
        InsertAfterKey(1);
        ViewAllList();
        RemoveByKey(1);
        ViewAllList();
        DeleteList();
        ViewAllList();
}
```

BAB XIII
TUMPUKAN

Kompetensi dan Indikator

Kompetensi Dasar : - Mengetahui dasar-dasar tumpukan.
- Mengetahui penggunaan tumpukan

Indikator Keberhasilan : - Mampu menjelaskan dasar-dasar tumpukan.
- Mampu membuat program aplikasi menggunakan tumpukan

Dasar Teori

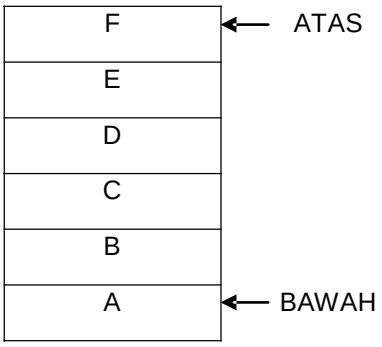
Tumpukan (*Stack*)

Stack merupakan bentuk khusus dari suatu struktur data, dimana node yang ditambahkan ke dalam list dan diambil dari list hanya pada kepalanya, atau dengan prinsip pengolahannya adalah last-in first-out (LIFO). Pada struktur ini hanya ada dua fungsi utama, yaitu push (memasukkan node ke dalam stack), dan pop (mengambil node dari stack).

Secara sederhana tumpukan bisa diartikan sebagai kumpulan data yang seolah-olah diletakkan di atas data yang lain. Dalam suatu tumpukan akan dapat dilakukan operasi penambahan (penyisipan) dan pengambilan (penghapusan) data melalui ujung yang sama, ujung ini merupakan ujung atas tumpukan.

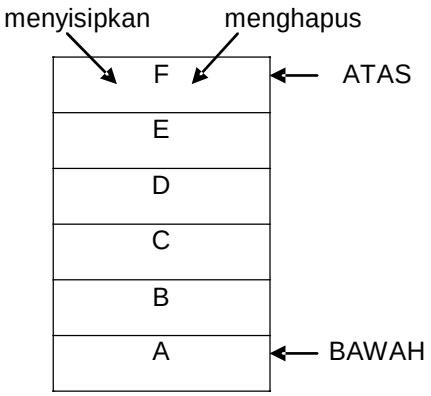
Contoh Kasus :

Terdapat dua buah kotak yang ditumpuk, kotak yang satu akan ditumpuk diatas kotak yang lainnya. Jika kemudian tumpukan 2 kotak tadi, ditambah kotak ketiga, keempat, kelima, dan seterusnya, maka akan diperoleh sebuah tumpukan kotak yang terdiri dari N kotak.



Dapat dilihat bahwa kotak B terletak diatas kotak A dan ada dibawah kotak C. Berdasarkan pada tumpukan tersebut dapat disimpulkan bahwa penambahan dan pengambilan sebuah kotak hanya dapat dilakukan dengan melewati satu ujung saja, yaitu bagian atas.

Kotak F adalah kotak paling atas sehingga jika ada kotak lain yang akan disisipkan maka kotak tersebut akan diletakkan pada posisi paling atas (diatas kotak F). Demikian juga pada saat pengambilan kotak, kotak F akan diambil pertama kali.



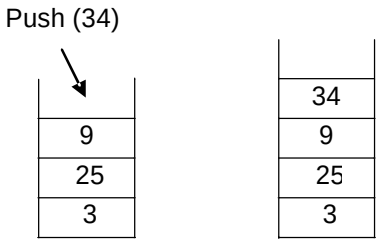
1. Operasi pada Tumpukan

Ada 2 operasi dasar yang bisa dilaksanakan pada sebuah tumpukan, yaitu :

- Menyisipkan data (push)
- Menghapus data (pop)

Operasi Push

Perintah push digunakan untuk memasukkan data ke dalam tumpukan.



Contoh :

```
        void Push (NOD **T, char item)
        {
                NOD *n;
                n=NodBaru (item);
                n->next=*T;
                *T=n;
    }
```

Operasi Pop

Operasi pop adalah operasi untuk menghapus elemen yang terletak pada posisi paling atas dari sebuah tumpukan.

Contoh :

```
        char Pop (NOD **T)
        {
                NOD *n; char item;
                if (!TumpukanKosong(*T)) {
                        P=*T;
                        *T=(*T)->next;
```

```
                item=P->data;
                free(P);
        }
        return item;
}
```

Untuk dapat mengetahui kosong tidaknya suatu tumpukan adalah dengan suatu fungsi yang menghasilkan suatu data bertipe boolean.

Contoh :

```
BOOL TumpukanKosong (NOD *T)
{
        return ((BOOL) (T==NULL));
}
```

2. Aplikasi Tumpukan

Pemanfaatan tumpukan antara lain untuk menulis ungkapan dengan menggunakan notasi tertentu.

Contoh :

(A + B) * (C – D)

Tanda kurung selalu digunakan dalam penulisan ungkapan numeris untuk mengelompokkan bagian mana yang akan dikerjakan terlebih dahulu. Dari contoh (A + B) akan dikerjakan terlebih dahulu, kemudian baru (C – D) dan terakhir hasilnya akan dikalikan.

A + B * C – D

B * C akan dikerjakan terlebih dahulu, hasil yang didapat akan berbeda dengan hasil notasi dengan tanda kurung.

Notasi Infix Prefix

Cara penulisan ungkapan yaitu dengan menggunakan notasi infix, yang artinya operator ditulis diantara 2 operator. Seorang ahli matematika bernama Jan Lukasiewiccz mengembangkan suatu cara penulisan ungkapan numeris yang disebut prefix, yang artinya operator ditulis sebelum kedua operand yang akan disajikan.

Contoh :

Infix	Prefix
A + B	+ A B
A + B – C	- + A B C
(A + B) * (C – D)	* + A B – C D

Proses konversi dari infix ke prefix :

= (A + B) * (C – D)
= [+ A B] * [- C D]
= * [+ A B] [- C D]
= * + A B - C D

Notasi Infix Postfix

Cara penulisan ungkapan yaitu dengan menggunakan notasi postfix, yang artinya operator ditulis sesudah operand.

Contoh :

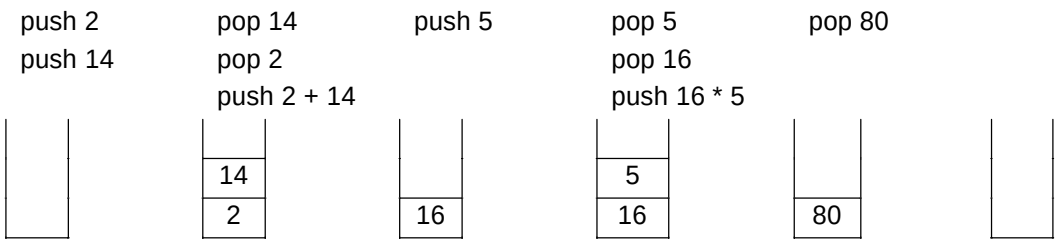
Infix	Postfix
16 / 2	16 2 /
(2 + 14) * 5	2 14 + 5 *
2 + 14 * 5	2 14 5 * +
(6 – 2) * (5 + 4)	6 2 – 5 4 + *

Proses konversi dari infix ke postfix :

= (6 - 2) * (5 + 4)
= [6 2 -] * [5 4 +]
= [6 2 -] [5 4 +] *
= 6 2 - 5 4 + *

Contoh :

Penggunaan notasi postfix dalam tumpukan, misal : 2 14 + 5 * = 80



BAB XIV

ANTRIAN

Kompetensi dan Indikator

Kompetensi Dasar : - Mengetahui dasar-dasar antrian.
- Mengetahui penggunaan antrian.

Indikator Keberhasilan : - Mampu menjelaskan dasar-dasar antrian.
- Mampu membuat program aplikasi menggunakan antrian.

Dasar Teori

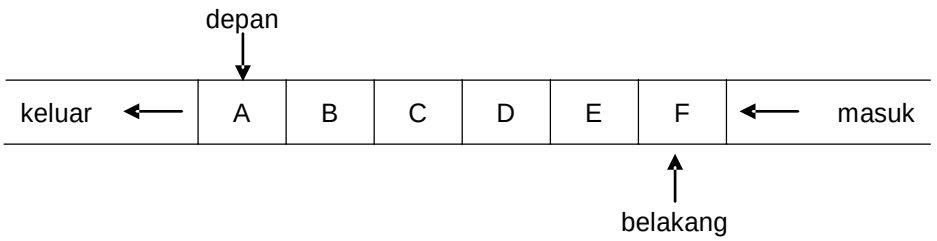
Antrian (*queue*)

Queue (Antrian) adalah suatu kumpulan data yang penambahan elemennya hanya bisa dilakukan pada suatu ujung (disebut dengan sisi belakang atau rear), dan penghapusan atau pengambilan elemen dilakukan lewat ujung yang lain (disebut dengan sisi depan atau front)

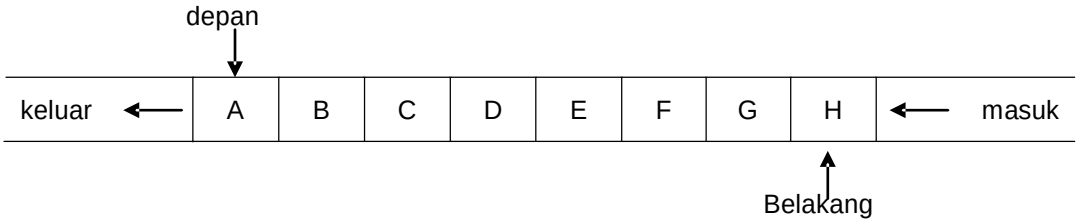
Jika pada tumpukan dikenal dengan menggunakan prinsip LIFO (Last In First Out), maka pada antrian prinsip yang digunakan adalah FIFO (First In First Out).

1. Implementasi Antrian dengan Array

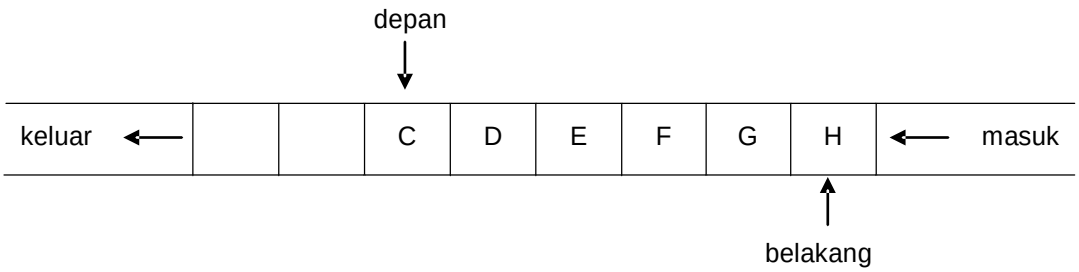
Karena antrian merupakan suatu kumpulan data, maka tipe data yang sesuai untuk menyajikan antrian adalah menggunakan array atau list (senarai berantai).



Antrian tersebut berisi 6 elemen, yaitu A, B, C, D, E, dan F. Elemen A terletak dibagian depan antrian dan elemen F terletak di bagian belakang antrian. Jika terdapat elemen baru yang akan masuk, maka elemen tersebut akan diletakkan disebelah kanan F. Jika ada elemen yang akan dihapus, maka A akan dihapus terlebih dahulu.



Antrian dengan penambahan elemen baru, yaitu G dan H.



Antrian dengan penghapusan elemen antrian, yaitu A dan B.

Seperti dalam tumpukan atau stack, maka di dalam antrian juga dikenal dua operasi dasar yaitu menambah elemen baru yang akan diletakkan di bagian belakang antrian dan menghapus elemen yang terletak di bagian depan antrian. Selain itu juga harus dilihat kondisi antrian mempunyai isi atau masih kosong.

Contoh :

```
 #define MAXN 6
typedef enum {NOT_OK, OK} Tboolean;
typedef struct {
        Titem array[MAXN];
        int first;
        int last;
        int number_of_items;
} Tqueue;
```

Elemen antrian dinyatakan dalam tipe integer yang semuanya terdapat dalam struktur. Variabel first menunjukkan posisi elemen pertama dalam array, dan variabel last menunjukkan posisi elemen terakhir dalam array.

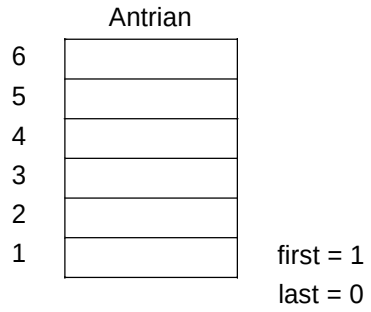
Untuk menambah elemen baru dan mengambil elemen dari antrian diperlukan deklarasi.

Contoh :

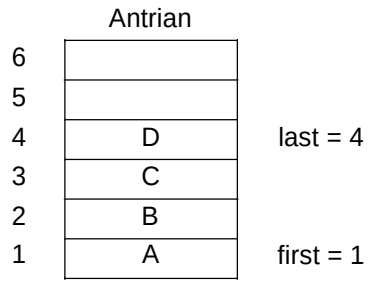
```
void initialize_queue (Tqueue *Pqueue) {
        Pqueue->first=0;
        Pqueue->last=-1;
        Pqueue->number_of_items=0;
}
Tboolean enqueue (Tqueue *Pqueue, Titem item) {
        if (Pqueue->number_of_items >= MAXN)
                return (NOT_OK);
        else {
                Pqueue->last++;
                if (Pqueue->last > MAXN – 1)
                        Pqueue->last=0;
                Pqueue->array[Pqueue->last]=item;
```

```
                Pqueue->number_of_items++;
                return (OK);
        }
}
Tboolean dequeue (Tqueue *Pqueue, Titem *Pitem) {
        if (Pqueue->number_of_items==0)
                return (NOT_OK);
        else {
                *Pitem=Pqueue->array[Pqueue->first++];
                if (Pqueue->first > MAXN – 1)
                        Pqueue->first=0;
                Pqueue->number_of_items--;
                return (OK);
        }
}
```

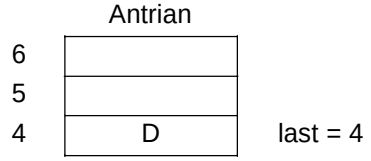
Kondisi awal sebuah antrian dapat digambarkan sebagai berikut.



Pada kondisi awal ini antrian terdiri dari last dibuat = 0 dan first dibuat = 1. Antrian dikatakan kosong jika last < first. Banyaknya elemen yang terdapat dalam antrian dinyatakan sebagai last – first + 1.



Dengan MAXN = 6 antrian telah terisi empat buah data yaitu A, B, C, dan D. Kondisi first = 1 dan last = 4.



3	C	first = 3
2		
1		

Pada antrian dilakukan penghapusan dua buah data yaitu A dan B. Sehingga kondisi first = 3 dan last = 4.

	Antrian	
6	F	last = 6
5	E	
4	D	
3	C	first = 3
2		
1		

Pada antrian dilakukan penambahan dua buah data yaitu E dan F. Elemen E akan diletakkan setelah D dan elemen F akan diletakkan setelah E. Sehingga kondisi first = 3 dan last = 6. Dapat diperoleh jumlah elemen yaitu 6 – 3 + 1 = 4. Dengan pembatasan data maksimal 6 elemen akan terdapat sisa 2 elemen. Jika pada antrian akan ditambahkan elemen yang baru, misal G, elemen G akan diletakkan setelah elemen F. Hal ini akan menyebabkan elemen yang baru tersebut tidak dapat masuk (MAXN = 6), meskipun masih tersisa 2 buah elemen yang kosong.

2. Implementasi Antrian dengan Pointer

Pada prinsipnya, antrian dengan pointer akan sama dengan antrian yang menggunakan array. Penambahan akan selalu dilakukan di belakang antrian dan penghapusan akan selalu dilakukan pada elemen dengan posisi paling depan. Antrian sebenarnya merupakan bentuk khusus dari suatu senarai berantai (linked list).

Pada antrian bisa digunakan dua variabel yang menyimpan posisi elemen paling depan dan elemen paling belakang. Jika menggunakan senarai berantai maka dengan dua pointer yang menunjuk elemen kepala (paling depan) dan elemen ekor (paling belakang) dapat dibentuk antrian.

Contoh :

```
struct queueNode {
        char data;
        struct queueNode * nextPtr;
};
typedef struct queueNode QUEUENODE;
typedef QUEUENODE *QUEUENODEPTR;
QUEUENODEPTR headPtr = NULL, tailPtr = NULL;
```

Berikut ini adalah contoh program untuk menambah elemen baru yang selalu diletakkan di belakang senarai.

```
void enqueue (QUEUENODEPTR *headPtr, QUEUENODEPTR *tailPtr, char
value) {
        QUEUENODEPTR newPtr = malloc (sizeof(QUEUENODE));
        if (newPtr != NULL) {
                newPtr->data=value;
                newPtr->nextPtr=NULL;
                if (isEmpty(*headPtr))
                        *headPtr=newPtr;
                else
                        (*tailPtr)->nextPtr=newPtr;
                *tailPtr=newPtr;
        }
        else
                printf("%c not inserted. No memory available. \n", value);
}
```

Berikut ini adalah contoh program untuk menghapus akan dilakukan pemeriksaan terlebih dahulu antrian dalam keadaan kosong atau tidak.

```
int isEmpty (QUEUENODEPTR headPtr) {
        return headPtr==NULL;
}
```