MODUL ALGORITMA DAN PEMROGRAMAN DASAR

CYNTHIA HAYAT SKOM MMSI

UNIVERSITAS KRISTEN KRIDA WACANA
Program Studi Sistem Informasi
TAHUN AKADEMIK GANJIL 2021/2022

Kata Pengantar

Puji Syukur kehadirat Tuhan Yang Maha Esa karena atas berkat dan kasih Karunia-Nya sehingga saya dapat menyelesaikan modul Pemrograman Berorientasi Objek ini. Modul ini disusun berdasarkan Kurikulum Program Studi (S1) Sistem Informasi 2017, dibentuk mengacu pada Permenristek Dikti No. 44 Tahun 2015 mengenai Standar Nasional Pendidikan Tinggi (SNPT). Capaian Pembelajaran Lulusan (CPL) yang disusun merujuk pada KKNI (Kualifikasi Kompetensi Nasional Indonesia) level 6, untuk jenjang pendidikan S1.

Modul ini juga dilengkapi dengan latihan soal untuk menguji pemahaman mahasiswa terkait dengan materi yang terdapat pada modul. Terima kasih kepada berbagai pihak yang telah membantu proses penyelesaian modul ini. Semoga modul ini dapat bermanfaat bagi kita semua, khususnya para peserta didik.

Jakarta, 17 Februari 2021

Penyusun

Cynthia Hayat S.Kom., M.MSI

DAFTAR ISI

Belajar C++: Pengenalan Algoritma

CYNTHIA HAYAT S.KOM., M.MSI

KRIDA WACANA CHRISTIAN UNIVERSITY
Faculty of Engineering and Computer Science
Departement of Information System

UKRIDA
Universitas Kristen Krida Wacana

Pengantar

- Logika merupakan dasar dari semua penalaran.
- Contoh : masakan ditambah garam menjadi asin,
- 7 lebih besar dari 5, dll
- Algoritma merupakan langkah-langkah logis penyelesaian masalah yang disusun secara sistematis.
- Contoh algoritma :
  - Masalah : membuat kue

    Algoritma : resep kue
  - Masalah : Menggunakan mesin

    Algoritma : petunjuk pemakaian alat

Cara Menyusun Algoritma

- Untuk menyusun sebuah algoritma, harus diketahui keadaan awal dan keadaan akhir yang diinginkan.
- Seringkali terdapat lebih dari satu algoritma untuk menyelesaikan suatu masalah.
- Contoh : Algoritma membuat secangkir teh manis

  Keadaan Awal :
  - Cangkir, panci, teh, gula, sendok, ada di lemari.
  - Air diambil dari keran.

  Keadaan Akhir :
  - Secangkir teh manis di meja.
  - Semua alat dikembalikan ke tempat semula

Contoh Solusi Algoritma Membuat Teh Manis

1. Ambil cangkir dari lemari letakkan di meja.
2. Ambil panci dari lemari.
3. Isi panci dengan air keran.
4. Panaskan panci (berisi air) di atas kompor.
5. Ambil sendok dari lemari.
6. Ambil gula dari lemari, masukkan satu sendok ke dalam cangkir.
7. Ambil teh dari lemari, masukkan satu sachet ke dalam cangkir.
8. Periksa apakah air sudah mendidih
9. Jika belum tunggu hingga mendidih
10. Jika sudah mendidih, matikan kompor. Tuangkan air ke dalam cangkir.
11. Aduk sampai gula larut.
12. Buang sisa air panas. Simpan panci, teh, gula ke lemari.

4 Langkah sistematis membuat program

1. Definisikan masalah
2. Design algoritma
3. Implementasikan
4. Uji coba dan Dokumentasi

Latihan

Disediakan 3 wadah kosong berukuran 3 liter, 5 liter dan 6 liter. Tuliskan algoritma sehingga keadaan akhir terdapat 4 liter air di wadah berukuran 5 liter, wadah lainnya kosong.

Solusi

1. Isi wadah 6ltr sampai penuh
2. Tuang isi wadah 6ltr ke wadah 5ltr sampai penuh
3. Buang isi wadah 5ltr
4. Isi wadah 3ltr sampai penuh
5. Tuang isi wadah 3ltr ke wadah 5ltr
6. Tuang isi wadah 6ltr ke wadah 5ltr

Latihan

Diberikan 4 buah angka sembarang. Tuliskan algoritma untuk mencari angka yang paling besar (searching)

Solusi

1. Bandingkan angka ke-1 & ke-2
2. Simpan angka terbesar
3. Bandingkan angka ke-3 & angka yg tersimpan
4. Simpan angka terbesar
5. Bandingkan angka ke-4 & angka yg tersimpan
6. Simpan angka terbesar
7. Tampilkan angka yg tersimpan

Catatan:

Komputer hanya dapat membandingkan dua buah angka pada satu saat.

Thank You

ukrida.ac.id

UKRIDA
Universitas Kristen Krida Wacana

Belajar C++: Struktur Dasar C++

CYNTHIA HAYAT S.KOM., M.MSI

KRIDA WACANA CHRISTIAN UNIVERSITY
Faculty of Engineering and Computer Science
Departement of Information System

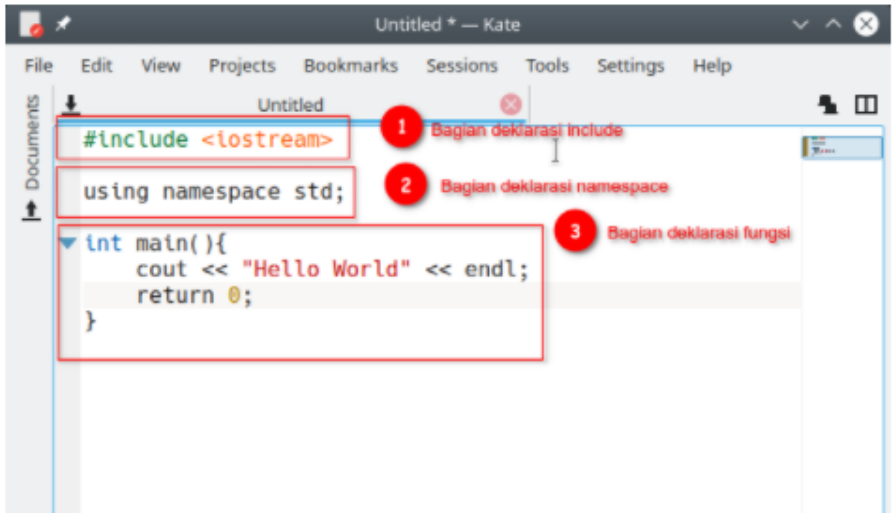
UKRIDA
Universitas Kristen Krida Wacana

1. Struktur Dasar Program C++

Bentuk atau struktur dasar program yang dibuat dengan C++ terdiri dari tiga bagian:

1. Bagian include
2. Bagian namespace
3. Bagian fungsi

Mari kita lihat contohnya:



1. Bagian Deklarasi Include

Pada bagian ini, kita mendefinisikan *library* (pustaka) apa saja yang akan kita gunakan di dalam program.

Library bisa kita anggap sebagai program lain yang ingin kita gunakan di dalam program kita.

Pada contoh di atas, kita menggunakan pustaka `iostream`. Library ini berisi fungsi-fungsi untuk melakukan input dan output.

Kadang kita juga akan menemukan *library* yang di-include dengan ekstensi `.h`, `.cpp`, `.hpp`, `.cc`, `.c`, dsb.

Contoh:

```
#include <math.h>
#include <signal.h>
#include <time.h>
#include "hello.h"
```

Semua memiliki arti yang sama, yaitu: gunakan *library* yang lain ke dalam program ini.

Perbedaanya pada jenis file yang akan diimpor:

- `.h` artinya header file dari C atau C++;
- `.cpp` artinya source code dari C++;
- `.hpp` artinya header file dari C++;
- `.cc` dan `.c` artinya header file dari C.

Lalu perbedaan yang lain tedapat pada simbol yang digunakan untuk include.

Jika menggunakan tanda kurung siku `<...>` maka program akan mencari library ke dalam sistem komputer kita.

Sedangkan yang menggunakan tanda petik, akan mencari ke lokasi yang ditentukan di sana.

Contoh:

```
#include "/home/dian/hello.h"
```

Maka program akan mencari library ke dalam direktori `/home/dian/`.

2. Bagian Deklarasi Namespace

Bagian ini sebenarnya bersifat opsional, bisa ditulis bisa tidak.

Pada contoh di atas, kita menggunakan namespace `std`. Karena fungsi-fungsi pada iostream dibungkus dalam namespace `std`.

Apabila kita tidak menggunakan namespace `std`, maka untuk menggunakan fungsi `cin` dan `cout` yang ada pada *iostream* harus diawali dengan `std::`.

Seperti ini:

```
std::cout << "Hello World!" << endl;
std::cout << "Belajar C++!" << endl;
```

Jika tidak ingin menulis `std::` terus-menerus, maka gunakanlah namespace `std`.

3. Bagian Fungsi

Bagian ini adalah baigan terpenting, di sinilah kita akan banyak menulis kode program.

Pada contoh di atas, terdapat fungsi main.

Fungsi `main()` adalah fungsi yang akan dieksekusi pertamakali saat program dibuka.

Fungsi ini wajib ada di setiap program yang dibuat untuk dieksekusi.

Tapi...

Jika hanya membuat program yang berfungsi sebagai library saja, fungsi `main()` boleh tidak dibuat.

Selain fungsi `main()` kita juga bisa membuat fungsi yang lain pada bagian ini.

Contoh:

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

int main(){
   cout << "Hello world!" << endl;
   return 0;
}

void hello(){
   cout << "Hello apa kabar?" << endl;
}
```

Pada contoh di atas kita membuat fungsi `hell()` di bawah fungsi `main()`.

Apakah boleh ditulis di atas fungsi `main()`?

Boleh.

2. Penulisan Statement dan Ekspresi

Statemen dan ekspresi adalah perintah-perintah yang ditulis di dalam fungsi.

Contoh:

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

int main(){
   cout << "Hello world!" << endl;
   cout << "Hari ini saya belajar c++" << endl;
   cout << "Belajar C++ itu mudah";
   return 0;
}
```

Perhatikan baris-baris ini:

```
cout << "Hello world!" << endl;
cout << "Hari ini saya belajar c++" << endl;
cout << "Belajar C++ itu mudah";
return 0;
```

Ini adalah statement dan ekspresi.

Penulisan statement dan ekspresi wajib diakhiri dengan titik-koma (;).

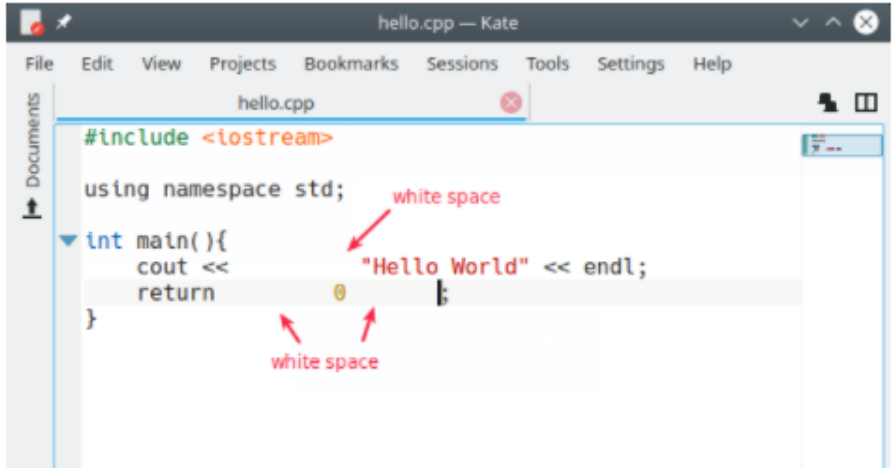
Apabila tidak ada titik-koma, maka program akan error.

3. White Space

White Space biasanya dibuat dengan tab dan spasi.

C++ tidak memiliki aturan yang ketat dalam penulisan *White space*.

Misalnya kita buat program seperti ini:



4. Penulisan Blok Kode

Blok kode adalah kumpulan dari beberapa statemen yang dibungkus dengan tanda kurung kurawal `{...}`.

Contoh:

Pada contoh di atas terdapat dua blok program. Blok pertama adalah blok fungsi `main()` lalu blok kedua adalah blok perulangan *for*.

Blok perulangan *for* berada di dalam blok fungsi `main()`.

Lalu pertanyaannya:

Apakah boleh blok `main()` berada di dalam blok *for*?

Jawabannya: tidak!

Karena blok `main()` merupakan fungsi.

Selain blok fungsi dan class, blok yang lainnya bisa ditulis di dalam blok *for*. Contohnya seperti blok *if*, *while*, *for*, *do/while*, dsb.

5. Penulisan Komentar

Komentar adalah bagian kode yang akan diabaikan oleh komputer. Ada dua cara penulisan komentar pada C++:

1. Menggunakan garis miring ganda `//`;
2. dan menggunakan garis miring bintang `/**/`.

Contoh:

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

/* ini adalah komentar
yang lebih dari satu baris */

int main(){
   // ini adalah komentar satu baris
   cout << "Hello world!" << endl;
   return 0;
}
```

Komentar biasanya digunakan untuk memberikan informasi tambahan pada kode program dan juga menonaktifkan statement maupun blok kode.

6. Penulisan String dan Karakter

String merupakan kumpulan karakter...

...atau kita juga bisa sebut dengan teks.

String dalam program C++ ditulis dengan diapit tanda ptik ganda (`"..."`) dan untuk karakter ditulis dengan tanda petik tunggal (`'...'`).

Contoh:

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

int main(){
   // contoh penulisan string
   cout << "Tutorial C++ untuk Pemula" << endl;
   // contoh penulisan karakter
   cout << 'a' << endl;
   return 0;
}
```

Thank You

ukrida.ac.id

UKRIDA
Universitas Kristen Krida Wacana

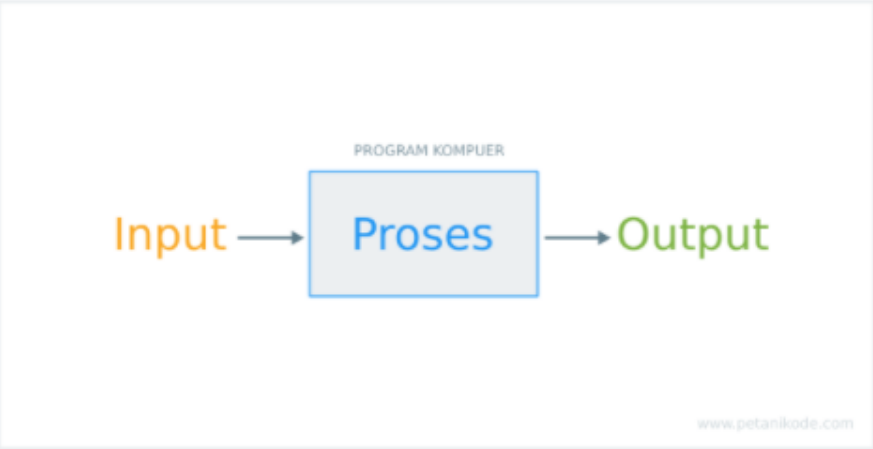
Belajar C++: Fungsi Input dan Output pada C++

CYNTHIA HAYAT S.KOM., M.MSI

KRIDA WACANA CHRISTIAN UNIVERSITY
Faculty of Engineering and Computer Science
Departement of Information System

UKRIDA
Universitas Kristen Krida Wacana

Pada dasarnya, program komputer hanya terdiri dari tiga bagian:



Input adalah sesuatu data yang kita masukan ke dalam program.

Input biasanya diambil dari perangakat inputan seperti keyboard, mouse, kamera, microphone, dll.

Proses adalah langkah-langkah yang harus dilakukan program untuk menghasilkan output.

Output adalah informasi yang dihasilkan setelah dilakukan proses. Output biasanya ditampilkan ke layar komputer.

Pada bahasa pemrograman C++, terdapat beberapa fungsi dasar untuk menampilkan output dan mengambil input.

Fungsi Output pada C++

C++ memiliki empat fungsi dasar untuk menampilkan output:

1. `cout` untuk menampilkan teks ke layar;
2. `cerr` untuk menampilkan error;
3. `clog` untuk menampilkan log;
4. `printf()` untuk menampilkan output, fungsi ini dari C;

Kita akan fokus memabahas yang dua saja, yakni `cout` dan `printf()`.

Soalnya `cerr` dan `clog`, cara pakainya sama seperti `cout`.

Bedanya sih pada konteks penggunaanya, yakni untuk error dan log.

1. Fungsi `cout`

Fungsi `cout` adalah fungsi standar pada C++ untuk menampilkan output ke layar.

Berikut ini struktur dasar fungsi `cout`:

Setelah simbol `<<` kita bisa menuliskan teks yang akan ditampilkan ke layar.

Teks harus diapit dengan tanda petik dan untuk membuat baris baru bisa menggunakan `endl` atau simbol `\n`.

Pembuatan baris baru bersifat opsional, terserah kita mau ditambahkan atau tidak.

Contoh:

```
cout << "Namaku adalah ";
cout << "Petani kode";
```

Maka outputnya akan ditampilkan dalam satu baris:

```
Namaku adalah Petani Kode
```

Sedangkan kalau kita menggunakan `endl` atau `\n`.

```
cout << "Namaku adalah " << endl;
cout << "Petani kode";
```

Hasilnya akan ditampilkan dalam dua baris:

```
Namaku adalah
Petani kode
```

2. Fungsi `printf()`

Fungsi `printf()` merupakan fungsi yang aslinya dari bahasa C, tapi bisa juga dipakai pada C++.

Fungsi `printf()` merupakan fungsi untuk menampilkan output ke layar komputer.

Fungsi ini terdapat pada library `<stdio.h>` dan juga `<iostream>`.

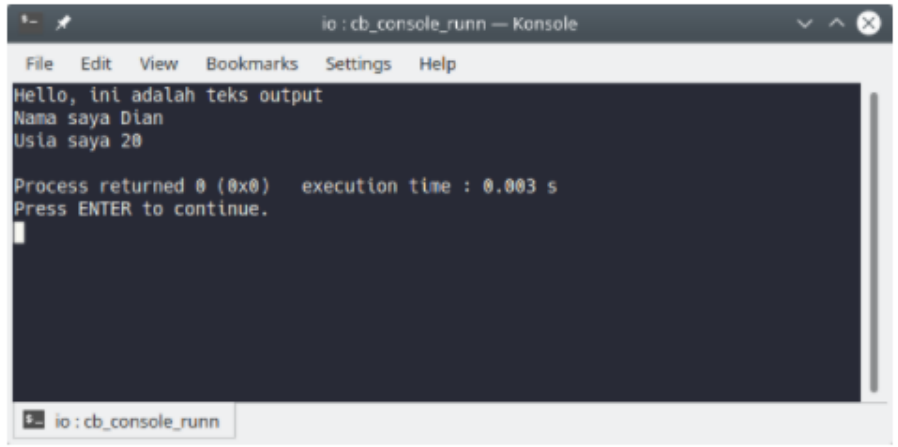
Berikut ini struktur dasar fungsi `printf()`:

Perhatikan:

`"format"` adalah sebuah teks (string) untuk ditampilkan. Lalu tanda `...` akan berisi sebuah variabel atau nilai untuk ditampilkan berdasarkan format yang diberikan pada teks `"format"`.

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

int main(){
    printf("Hello, ini adalah teks output\n");
    printf("Nama saya %s\n", "Dian");
    printf("Usia saya %d\n", 20);
    return 0;
}
```

Hasilnya:



Ada beberapa hal yang perlu di perhatikan...

Pada fungsi `printf()` kita menggunakan simbol `%s`, `%d`, dan `\n` untuk format teks.

Mari kita bahas arti dari simbol tersebut:

- `%s` adalah simbol untuk menampilkan nilai string;
- `%d` adalah simbol untuk menampilkan nilai angka atau bilangan desimal;
- `\n` adalah simbol untuk membuat baris baru.

Selain tiga simbol tersebut, masih banyak lagi simbol yang lain.

Simbol	Arti atau Fungsi
`%c`	untuk menampilkan karakter
`%s`	untuk menampilkan teks (string)
`%d, %1`	untuk menampilkan bilangan desimal
`%f`	untuk menampilkan bilangan pecahan
`%o`	untuk menampilkan bilangan oktal
`%x`	untuk menampilkan bilangan heksadesimal
`\t`	untuk membuat tabs

Fungsi Input pada C++

Sementara untuk mengambil input, C++ memiliki fungsi `cin` dan bisa juga menggunakan `scanf()` dari bahasa C.

1. Fungsi `cin`

Fungsi `cin` (c input) adalah fungsi untuk mengambil input dari keyboard.

Berikut ini bentuk dasar fungsi `cin`:

Fungsi `cin` membutuhkan variabel untuk menyimpan data yang diinputkan.

Intinya variabel berfungsi untuk meyimpan data saat program berjalan.

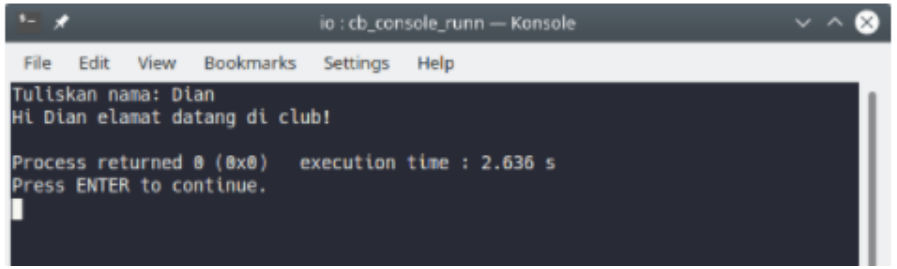
Mari kita coba menggunakan fungsi `cin`.

Buatlah program baru bernama `program_input.cpp`, kemudian isi dengan kode berikut:

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

int main(){
    string nama;
    cout << "Tuliskan nama: ";
    cin >> nama;

    cout << "Hi " << nama;
    cout << " elamat datang di club!" << endl;

    return 0;
}
```

Setelah itu, coba *compile* dan jalankan.

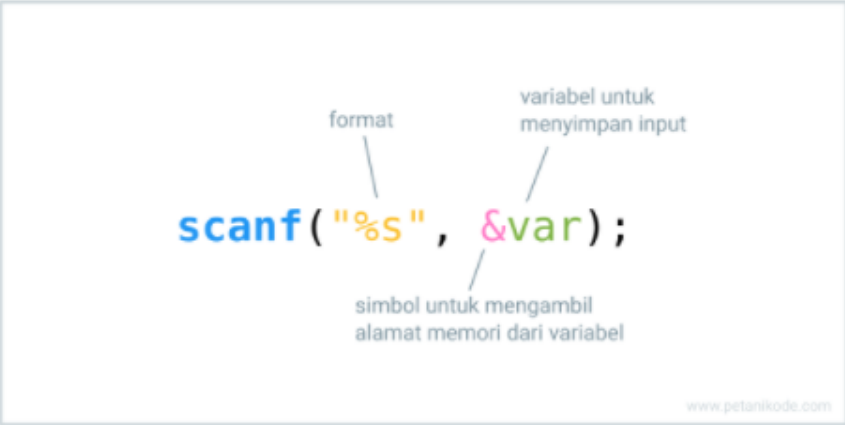
Maka hasilnya:



2. Fungsi `scanf()`

Fungsi `scanf()` sebenarnya dari bahasa C, tapi bisa juga digunakan pada C++.

Fungsi `scanf()` adalah fungsi untuk mengambil input dari keyboard. Fungsi ini memiliki format seperti fungsi `printf()`.



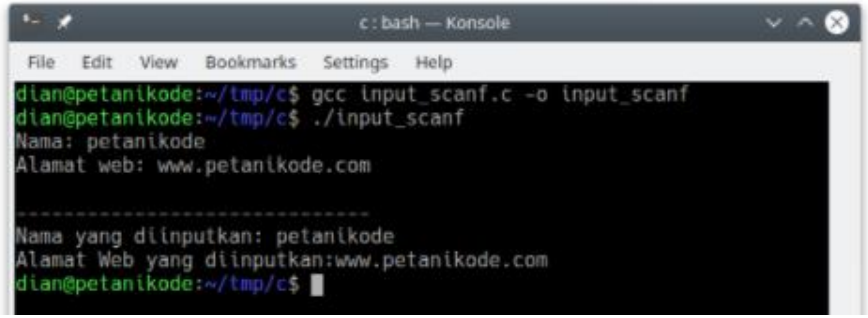
Format yang diberikan tergantung dari jenis tipe data apa yang ingin kita ambil.

Misalnya mau mengambil angka, maka kita bisa pakai `%d` atau `%i`.

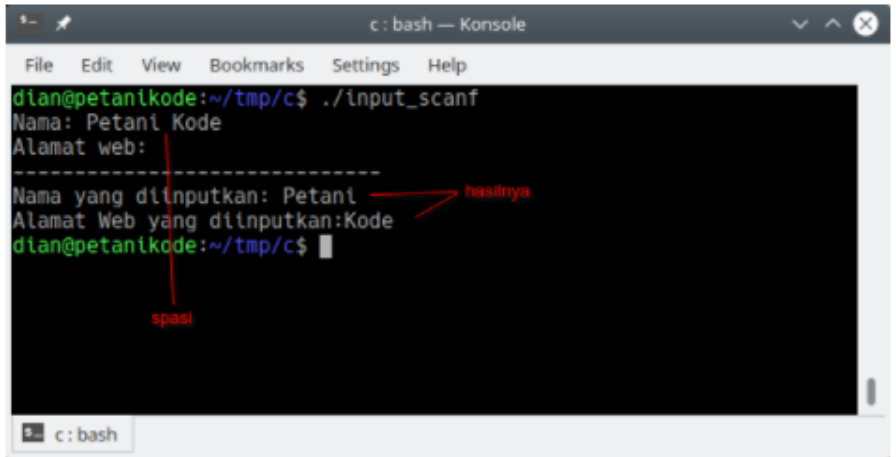
Mari kita lihat contohnya:

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

int main () {
    // membuat variabel
    char name[20], web_address[30];

    printf("Nama: ");
    scanf("%s", &name);

    printf("Alamat web: ");
    scanf("%s", &web_address);

    printf("\n------------------------------\n");
    printf("Nama yang diinputkan: %s\n", name);
    printf("Alamat Web yang diinputkan: %s\n", web_address);

    return 0;
}
```

Hasilnya:



Tapi saat kita menginputkan teks yang mengandung spasi, hasilnya akan dipecah menjadi dua seperti ini:



Untuk mengatasi masalah ini, kita bisa ubah format yang digunakan pada `scanf()` menjadi seperti ini:

```
printf("Nama: ");
scanf("%[^\n]s", name);
```

Maka fungsi `scanf()` akan menerima spasi.

Dalam menggunakan `scanf()`, kita dianjurkan menggunakan simbol `&` sebelum nama variabel.

Contoh:

```
#include <stdio.h>

void main(){

    int a, b, c;

    printf("Inputkan nilai a: ");
    scanf("%i", &a);

    printf("Inputkan nilai b: ");
    scanf("%i", &b);

    c = a + b;

    printf("Hasil a + b: %i", c);

}
```

Simbol `&` berfungsi untuk mengambil alamat memori dari sebuah variabel.

Fungsi `scanf()` membutuhkan tempat untuk menyimpan nilai yang akan diinputkan.

Karena itu kita memberikan simbol `&` di depan nama variabel untuk menentukan alamat memori yang akan digunakan oleh `scanf()`.

Thank You

ukrida.ac.id

UKRIDA
Universitas Kristen Krida Wacana

Belajar C++: Mengenal Variabel, Konstanta, Tipe data, dan Operator

CYNTHIA HAYAT S.KOM., M.MSI

KRIDA WACANA CHRISTIAN UNIVERSITY
Faculty of Engineering and Computer Science
Departement of Information System

UKRIDA
Universitas Kristen Krida Wacana

Inti dari sebuah program komputer adalah menerima input, melakukan pemrosesan, dan menghasilkan output.

Nilai input bisa kita dapatkan dari keyboard, file, kamera, mikrofon, dan sebagainya.

Sementara output dapat kita tampilkan ke monitor, cetak ke dokumen, atau ke dalam sebuah file.

Pada tahap pemrosesan, program membutuhkan bantuan variabel untuk menyimpan nilai sementara.

Sama seperti waktu kita berpikir, kita membutuhkan beberapa ingatan untuk memproses informasi.

Apa itu Variabel dan Tipe Data?

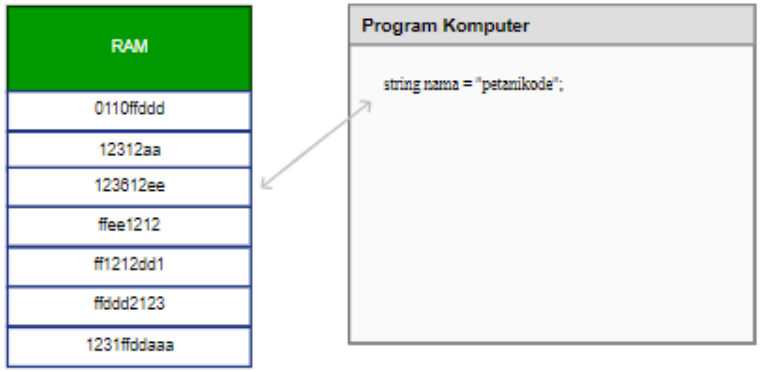
Semua program komputer yang sedang berjalan akan menyimpan data sementara di dalam RAM (Random Access Memori).

Data-data yang tersimpan di dalam RAM memiliki alamat yang direpresentasikan dalam bilangan heksa desmial.

Bagaiaman cara program menyimpan nilai ke RAM?

Jawabannya dengan menggunakan variabel.

Semakin banyak variabel yang kamu buat semakin besar pula memori yang akan digunakan di dalam RAM.



Macam-macam tipe data di C++ dapat dilihat dari tabel berikut.

Nama Tipe Data	Ukuran dalam Memori	Rentang Nilai
char	1byte	-127 sampai 127 atau 0 sampai 255
unsigned char	1byte	0 sampai 255
signed char	1byte	-127 sampai 127
int	4bytes	-2147483648 sampai 2147483647
unsigned int	4bytes	0 sampai 4294967295
signed int	4bytes	-2147483648 sampai 2147483647
short int	2bytes	-32768 sampai 32767
unsigned short int	Range	0 sampai 65,535
signed short int	Range	-32768 sampai 32767
long int	4bytes	-2,147,483,648 sampai 2,147,483,647
signed long int	4bytes	same as long int
unsigned long int	4bytes	0 sampai 4,294,967,295
float	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	8bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	2 or 4 bytes	1 wide character
boolean	4bytes	`true` atau `false`

Cara Membuat Variabel di C++

Pembuatan variabel atau deklarasi variabel di C++ dapat kita lakukan seperti ini:

```
string nama;
int umur;
char jenis_kelamin;
```

Tipe datanya ditulis terlebih dahulu, lalu diikuti dengan nama variabelnya.

> FYI: untuk C++11 kita bisa gunakan tipe data `auto` sebagai placeholder untuk tipe data yang belum jelas.

Variabel-variabel di atas akan menyimpan nilai `null` (kosong), karena belum kita isi.

Kita juga dapat membuat variabel dengan mengisinya langsung.

Contoh:

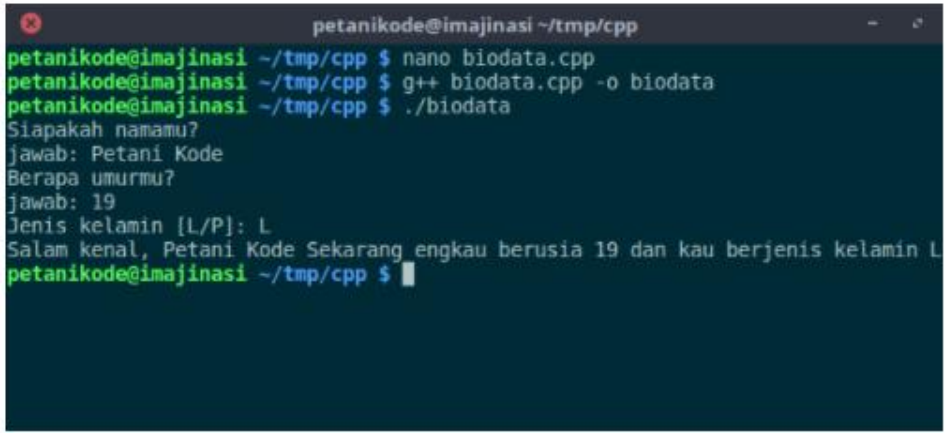
```
string nama = "Petani Kode";
float tinggi = 172.43;
```

Membuat Porgram C++ dengan Variabel dan Tipe Data

Silahkan buat sebuah file baru bernama `biodata.cpp`, kemudian isi dengan kode berikut:

```
#include <iostream>

using namespace std;

int main(){

  // deklarasi tipe data variabel
  string nama;
  int umur;
  char jenis_kelamin;

  // --- proses input ---
  cout << "Siapakah namamu?" << endl;
  cout << "jawab: ";
  // menyimpan data ke variabel
  getline(cin,nama);

  cout << "Berapa umurmu?" << endl;
  cout << "jawab: ";
  // menyimpan data ke variabel
  cin >> umur;

  cout << "Jenis kelamin [L/P]: ";
  // menyimpan data ke variabel
  cin >> jenis_kelamin;

  // --- proses output ---
  cout << "Salam kenal, " << nama << " Sekarang engkau berusia ";
  cout << umur << " dan kau berjenis kelamin "<< jenis_kelamin;

  return 0;

}
```

Setelah itu lakukan kompilasi dan eksekusi programnya.



Pertama dimulai dari membuat variabel.

```
string nama;
int umur;
char jenis_kelamin;
```

Pada baris kode tersebut, kita membuat tiga buah variabel dengan tipe data yang berbeda-beda.

Lalu kita mengisi nilainya berdasarkan input yang diberikan dari keyboard dengan perintah `cin`.

```
cin >> umur;
cin >> jenis_kelamin;
```

Khusus untuk tipe data `string`, kita menggunakan fungsi `getline()` untuk mengambil satu baris nilai yang diinputkan.

```
getline(cin,nama);
```

Terakhir kita menampilkan isi variabelnya dengan perintah `cout`.

Apa itu Operator?

Operator adalah sebuah simbol...

Simbol yang digunakan untuk melakukan operasi tertentu.

Misalnya:

Kita ingin menjumlahkan nilai dari variabel x dan y, maka kita bisa menggunakan operator penjumlahan (+).

```
x + y
```

Ada enam jenis kelompok operator dalam pemrograman C++ yang harus kamu ketahui:

1. Operator Artimatika;
2. Operator Penugasan;
3. Operator Pembanding;
4. Operator Logika;
5. Operator Bitwise;
6. dan Operator Lain-lain.

1. Operator Aritmatika

Operator aritmatika merupakan operator yang digunakan untuk melakukan operasi aritmatika.

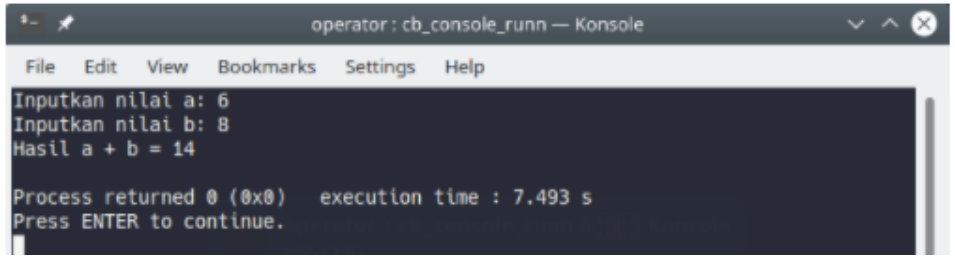
Operator ini terdiri dari:

Nama Operator	Simbol
Penjumlahan	+
Pengurangan	-
Perkalian	*
Pembagian	/
Sisa Bagi	%

Buatlah program baru bernama `operator_aritmatika.cpp`, kemudian isi dengan kode berikut.

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

int main(){

    int a, b, c;

    cout << "Inputkan nilai a: ";
    cin >> a;
    cout << "Inputkan nilai b: ";
    cin >> b;

    // menggunakan operator penjumlahan
    c = a + b;

    cout << "Hasil a + b = " << c << endl;

    return 0;
}
```

Maka hasilnya:



2. Operator Penugasan

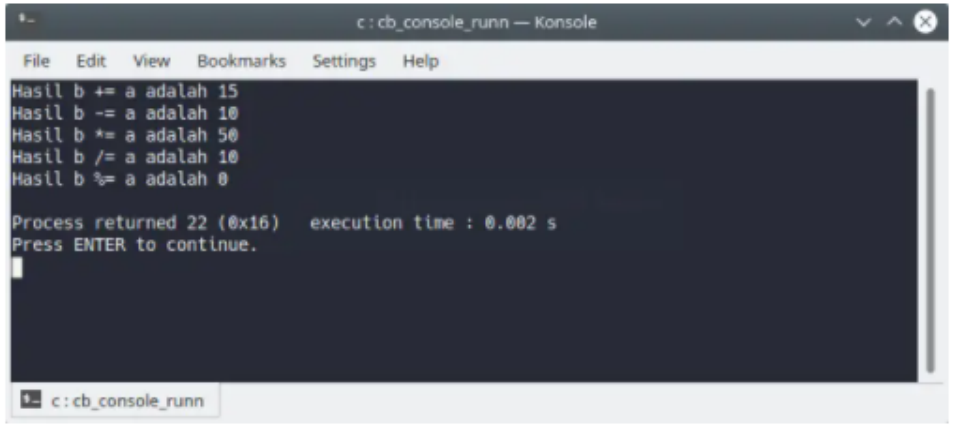
Operator penugasan (Assignment Operator) merupakan operator untuk meberikan tugas pada variabel. Biasanya untuk **mengisi nilai**.

Operator Penugasan terdiri dari:

Nama Operator	Sombol
Pengisian Nilai	=
Pengisian dan Penambahan	+=
Pengisian dan Pengurangan	-=
Pengisian dan Perkalian	*=
Pengisian dan Pembagian	/=
Pengisian dan Sisa bagi	%=
Pengisian dan shift left	<<=
Pengisian dan shift right	>>=
Pengisian dan bitwise AND	&=
Pengisian dan bitwise OR	|=
Pengisian dan bitwise XOR	^=

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

int main(){

    int a, b;

    // pengisian nilai dengan operator =
    a = 5;
    b = 10;

    // pengisian sekaligus penambahan
    b += a; // ini sama seperti b = b + a
    cout << "Hasil b += a adalah " << b << endl;

    // pengisian sekaligus pengurangan
    b -= a; // ini sama seperti b = b - a
    cout << "Hasil b -= a adalah " << b << endl;

    // pengisian sekaligus perkalian
    b *= a; // ini sama seperti b = b * a
    cout << "Hasil b *= a adalah " << b << endl;

    // pengisian sekaligus pembagian
    b /= a; // ini sama seperti b = b / a
    cout << "Hasil b /= a adalah " << b << endl;

    // pengisian sekaligus penambahan
    b %= a; // ini sama seperti b = b % a
    cout << "Hasil b %= a adalah " << b << endl;

    return 0;
}
```

Hasilnya:



Pada program tersebut, variabel `b` kita isi ulang dengan operator penugasan.

Sebagai contoh, operasi:

```
b += a
```

Sama seperti operasi:

```
b = b + a
```

Artinya kita akan mengisi nilai untuk `b` dengan nilai `b` ditambah nilai `a`.

3. Operator Pembanding

Operator pembanding adalah operator untuk memabndingkan dua buah nilai. Operator ini juga dikenal dengan operator relasi.

Operator pembanding terdiri dari:

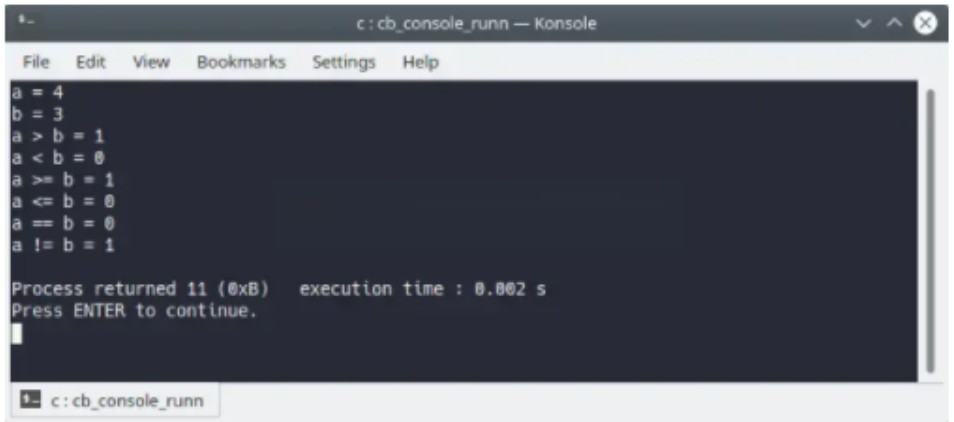
Nama Operator	Simbol
Lebih Besar	`>`
Lebih Kecil	`<`
Sama Dengan	`==`
Tidak Sama dengan	`!=`
Lebih Besar Sama dengan	`>=`
Lebih Kecil Sama dengan	`<=`

Nilai yang dihasilkan dari operasi pembanding akan berupa `true` dan `false`.

Pada bahasa C++, nilai `true` akan samadengan `1` dan `false` akan samadengan `0`.

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

int main(){
    int a = 4, b = 3;
    bool hasil;

    cout << "a = " << a << endl;
    cout << "b = " << b << endl;

    // menggunakan operator pembanding
    hasil = a > b;
    cout << "a > b = " << hasil << endl;

    hasil = a < b;
    cout << "a < b = " << hasil << endl;

    hasil = a >= b;
    cout << "a >= b = " << hasil << endl;

    hasil = a <= b;
    cout << "a <= b = " << hasil << endl;

    hasil = a == b;
    cout << "a == b = " << hasil << endl;

    hasil = a != b;
    cout << "a != b = " << hasil << endl;


    return 0;
}
```

Maka hasilnya:



Operator perbandingan biasanya akan kita pakai saat membuat percabangan.

4. Operator Logika

Kalau kamu pernah belajar logika matematika, pasti tidak akan asing dengan operator ini.

Nama Operator	Simbol
Logika AND	`&&`
Logika OR	`||`
Negasi/kebalikan	`!`

Operator Logika digunakan untuk membuat operasi logika.

Misalnya seperti ini:

- Pernyataan 1: Petani Kode seorang programmer
- Pernyattan 2: Petanikode menggunakan Linux

Jika ditanya, apakah Petani Kode programmer yang menggunakan Linux?

Tentu kita akan cek dulu kebenarannya

- Pernyataan 1: Petani Kode seorang programmer = `true`.
- Pernyataan 2: Petanikode menggunakan Linux = `true`.

Apa petanikode programmer dan menggunakan Linux?

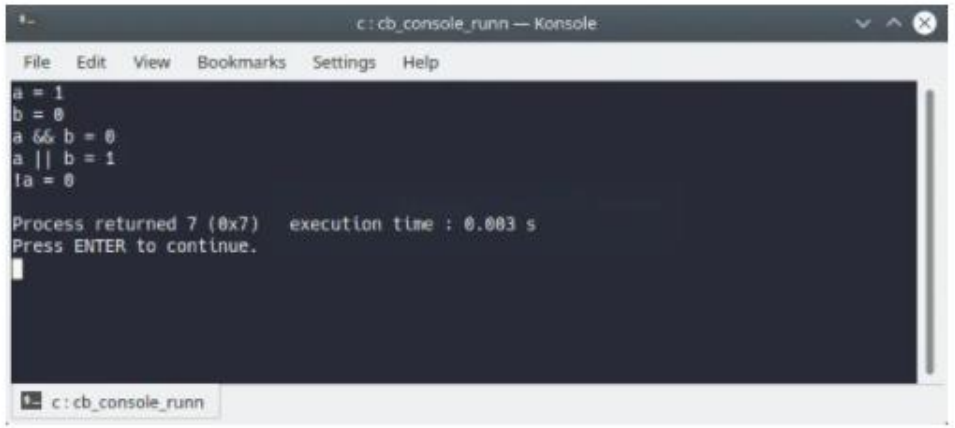
Coba cek lagi tabel kebenaran untuk logika AND.

Pernyatan 1	Pernytaan 2	Hasil Logika AND
`true`	`true`	`true`
`true`	`false`	`false`
`false`	`true`	`false`
`false`	`false`	`false`

Buatlah program baru bernama `operator_logika.cpp`, kemudia ini dengan kode berikut.

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

int main(){
    int a = 1; // true
    int b = 0; // false
    bool hasil;

    cout << "a = " << a << endl;
    cout << "b = " << b << endl;

    // logika AND
    hasil = a && b;
    cout << "a && b = " << hasil << endl;

    // logika OR
    hasil = a || b;
    cout << "a || b = " << hasil << endl;

    // logika NOT
    cout << "!a = " << !a << endl;

    return 0;
}
```

Hasilnya:



5. Operator Bitwise

Operator bitwise merupkan operator yang digunakan untuk operasi berdasarkan bit (biner) dari sebuah nilai.

Operator bitwise terdiri dari:

Nama Operator	Simbol di Java
AND	&
OR	|
XOR	^
NOT/komplemen	~
Left Shift	<<
Right Shift	>>

Buat kamu yang sudah pernah belajar sistem bilangan dan sistem digital akan mudah memahami cara kerja operator ini.

Misalkan kita punya nilai 6 dan 3.

Nilai 6 dan 3 kemudian kita ubah ke dalam bentuk bilangan biner.

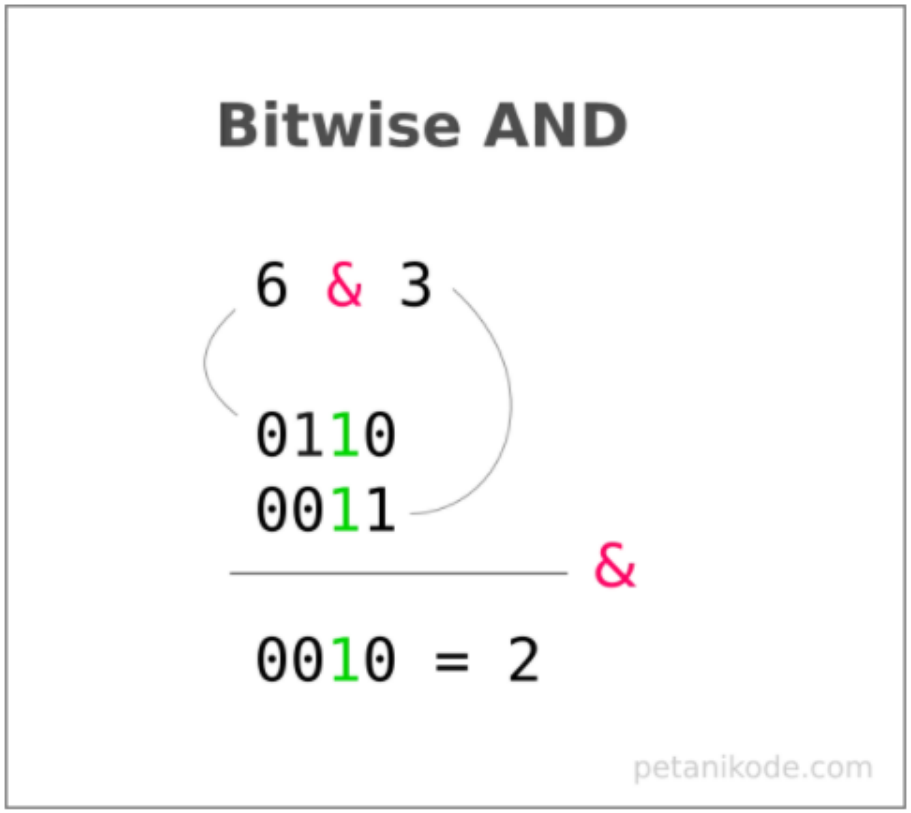
Sehiga akan menjadi seperti ini:

```
6 = 0110
3 = 0011
```

Nah, operator bitwise akan melakukan operasi berdasarkan biner-biner tersebut.

Bitwise AND (&)

Bitwise AND merupakan operasi bit berdasarkan logika AND, perhatikan gambar ini.

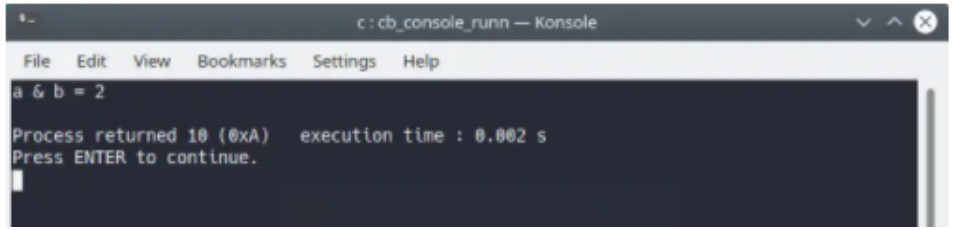


Perhatikan bilangan biner untuk nilai 6 dan 3.

Buatlah program baru bernama `bitwise_and.c`, kemudian isi dengan kode berikut:

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

int main(){
    int a = 6;
    int b = 3;
    int hasil;

    // menggunakan operator bitwise and
    hasil = a & b;
    cout << "a & b = " << hasil << endl;

    return 0;
}
```

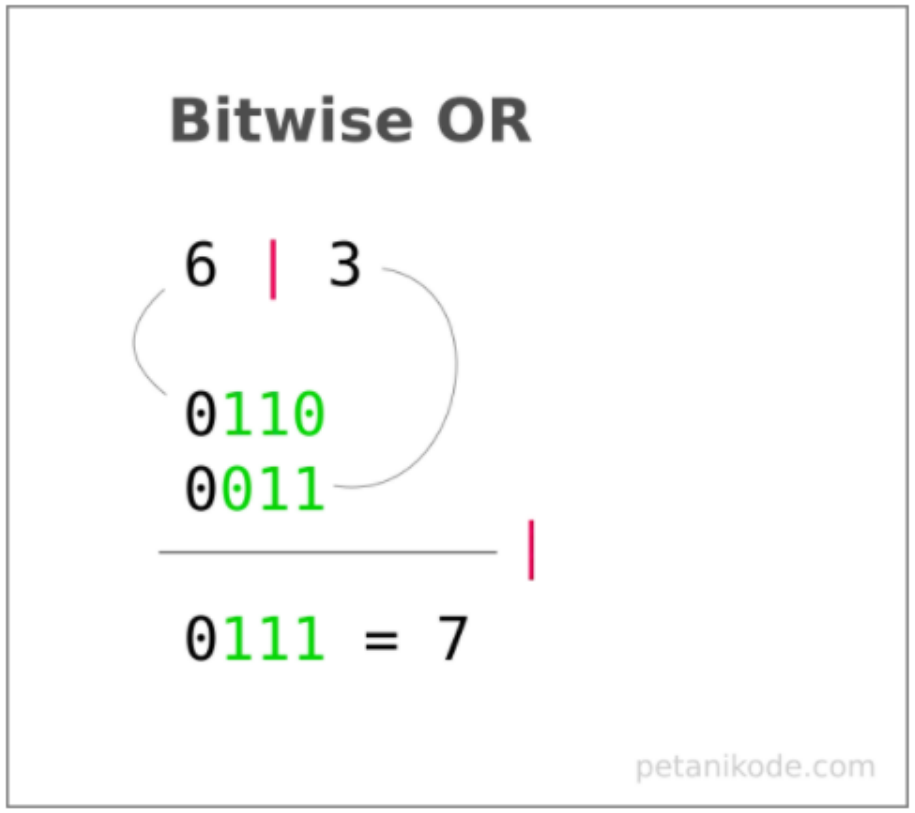
Maka hasilnya:



Bitwise OR (|)

Operator bitwise OR juga sama seperti bitwise AND.

Operator bitwise OR akan menghasilkan nilai `false` atau `0` saat keduanya `false`.



Contoh Program: `bitwise_or.c`

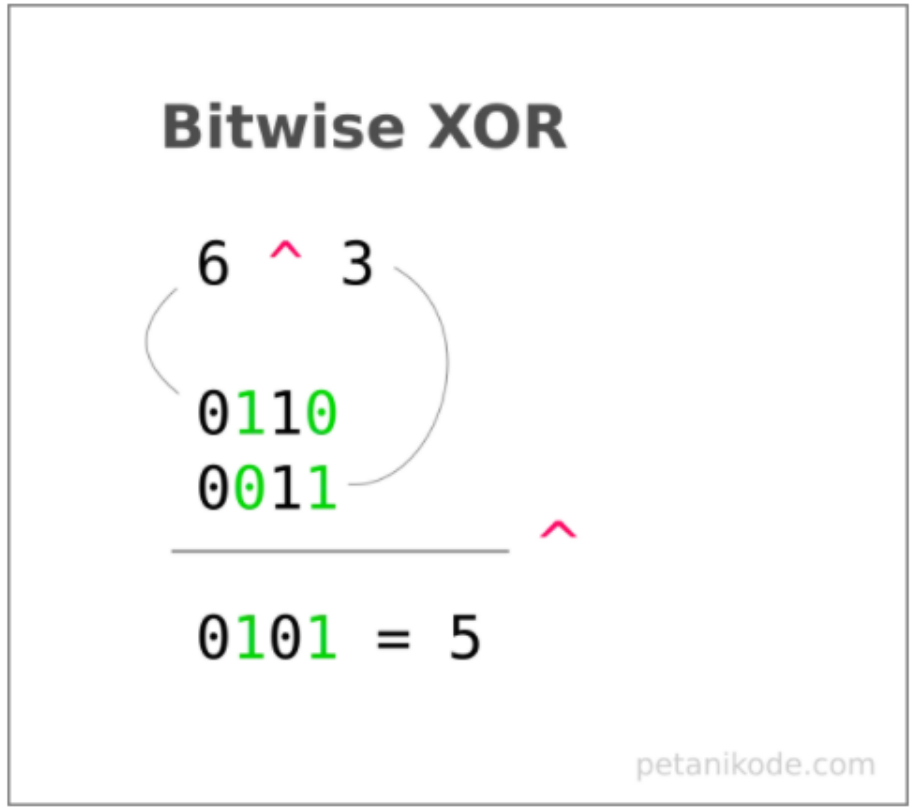
```
#include <iostream>
using namespace std;

int main(){
    int a = 6;
    int b = 3;
    int hasil;

    // menggunakan operator bitwise or
    hasil = a | b;
    cout << "a & b = " << hasil << endl;

    return 0;
}
```

Hasil outputnya:

```
a | b = 7
```

Bitwise XOR (^)

Operator XOR *(Exclusive OR)* akan menghasilkan nilai 1 saat kedua nilai tidak sama.



Contoh Program: bitwise_xor.c

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

int main(){
    int a = 6;
    int b = 3;
    int hasil;

    // menggunakan operator bitwise xor
    hasil = a ^ b;
    cout << "a ^ b = " << hasil << endl;

    return 0;
}
```

Hasil outputnya:

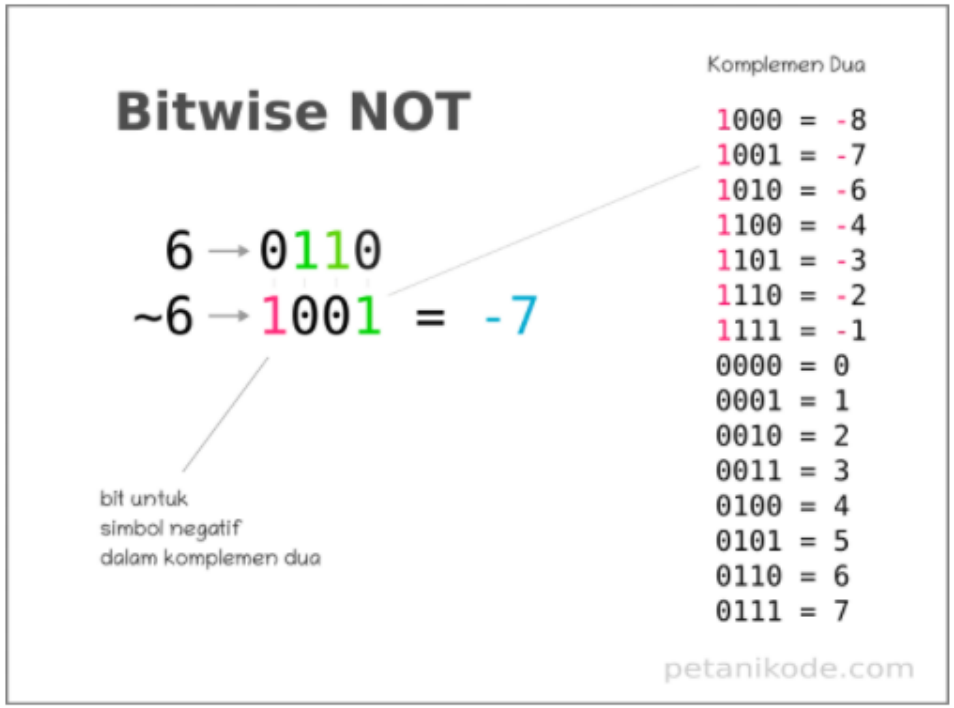
```
a ^ b = 5
```

Bitwise NOT (~)

Bitwise NOT dikenal juga dengan komplemen.

Operator ini akan menghasilkan nilai biner yang terbalik dari biner aslinya.

Kemudian direpresentasikan dengan komplemen dua↗.



Contoh program: bitwise_not.c

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

int main(){
    int a = 6;
    int hasil;

    // menggunakan operator bitwise not
    hasil = ~a;
    cout << "~a = " << hasil << endl;

    return 0;
}
```

Hasil outputnya:

```
~a = -7
```

6. Operator Lain-lain

Selain dari operator yang kita bahas di atas, terdapat beberapa operator lain yang harus diketahui:

Nama Operator	Simbol	Keterangan
Alamat memori	`&`	untuk mengambil alamat memori
Pointer	`*`	untuk membuat pointer
Ternary	`? :`	untuk membuat kondisi
Increment	`++`	untuk menambah `1`
Decrement	`--`	untuk mengurangi `1`

Operator `&` jika digunakan pada satu variabel, akan berfungsi untuk mengambil alamat memori dari variabel tersebut.

Dan operator `*` kan berfungsi untuk membut sebuah pointer.

Thank You

ukrida.ac.id

UKRIDA
Universitas Kristen Krida Wacana

Belajar C++: Pseudocode

CYNTHIA HAYAT S.KOM., M.MSI

KRIDA WACANA CHRISTIAN UNIVERSITY
Faculty of Engineering and Computer Science
Departement of Information System

UKRIDA
Universitas Kristen Krida Wacana

Pseudocode & Flowchart

- Menuliskan algoritma menggunakan bahasa sehari-hari : tidak efisien, terlalu panjang.
- Programmer mengenal dua standar penulisan algoritma, yaitu :
  - Pseudocode
  - Flowchart

Pseudocode

- Pseudocode adalah bahasa yang digunakan untuk menyederhanakan penulisan algoritma.
- Pseudocode **bukan bahasa pemrograman.**

Flowchart

- Cara lain menuliskan algoritma adalah dengan flowchart. Cara ini sudah jarang dipakai karena sangat menyita waktu dan kertas.
- Keunggulan flowchart: Alur program sangat mudah dibaca/ditelusuri.

Aturan Penulisan Pseudocode

- Isikan Nilai x kedalam min
- Min ← x
- Notasi "← " berarti mengisi peubah (variabel) min dengan nilai peubah x.

Pada dasarnya pseudocode disusun 3 bagian, yaitu :

-Bagian kepala (header)

-Bagian deklarasi

-Bagian deskripsi

Bagian Kepala

Bagian yang terdiri dari atas nama algoritma dan penjelasan tentang algoritma tersebut.

Contoh :

Algoritma Luas_Lingkaran

{Menghitung luas lingkaran untuk jari-jari tertentu}

Bagian Kamus

Bagian untuk mendefinisikan semua nama yang dipakai dalam algoritma.

Nama dapat berupa nama peubah (variabel),nama tipe, nama prosedur,dll.

Contoh :

Kamus:

A,B,C :integer

D:char

Bagian deskripsi

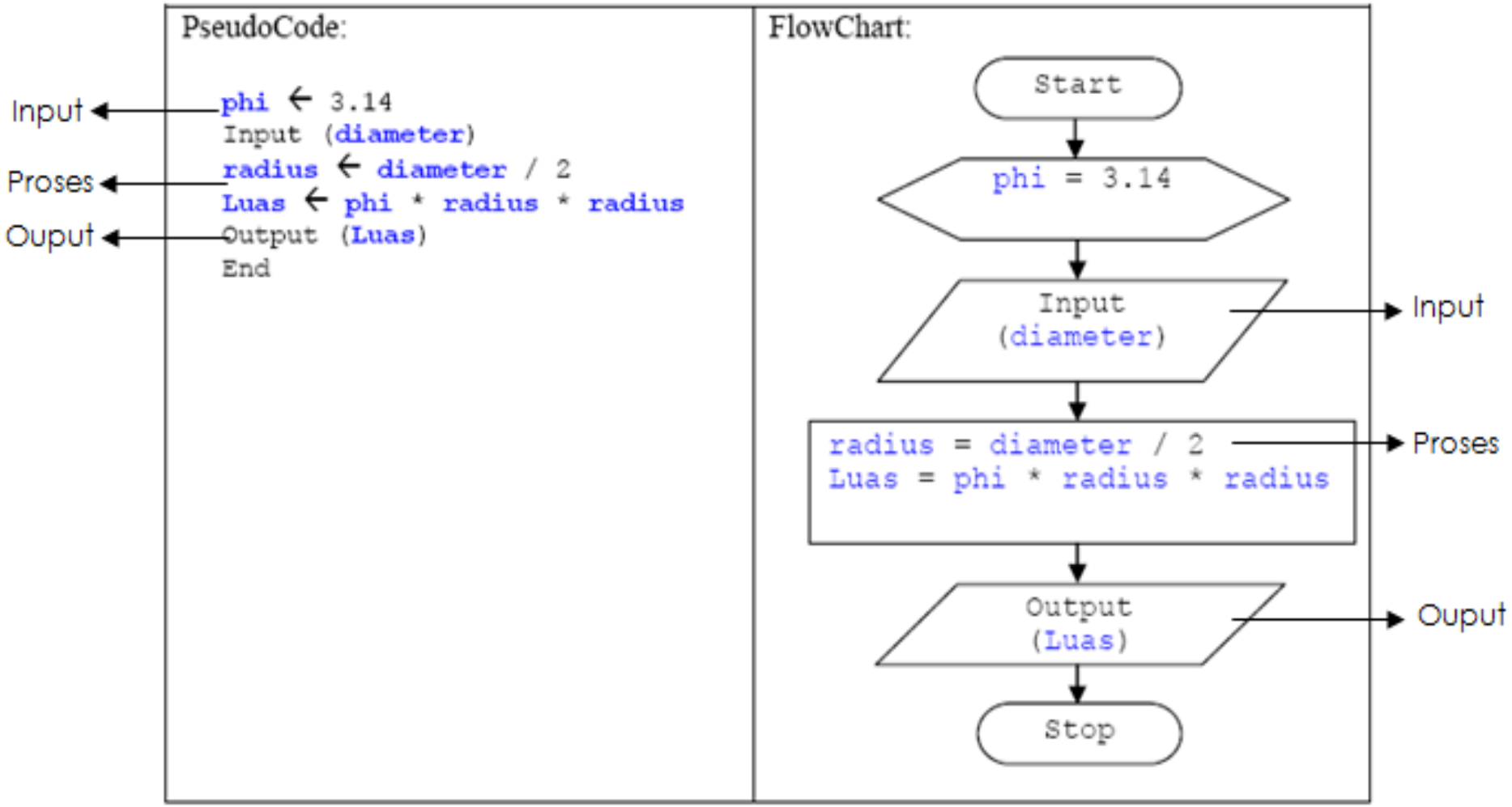
Bagian ini berisi uraian langkah-langkah penyelesaian masalah.

Contoh :

Deskripsi

```
read(a,b)
if a<b then
        c←a+b
else
        c←b-a
end if
write c
```

Contoh Pseudocode



UKRIDA

LATIHAN

Buat algoritma dengan flowchart dan pseudocodenya:

1. Konversi dari Celcius ke Reamur. R=(4/5)*C
2. Konversi dari Celcius ke Farenheit. F=(9/5)*C+32
3. Menghitung sisi miring dari suatu segitiga siku-siku, jika diketahui panjang sisi yang membentuk sudut siku-siku. [clue : Phitagoras]
4. Menghitung usia berdasarkan tahun lahir (tl) dan tahun sekarang (ts)
5. Menghitung rata-rata 5 bilangan
6. Konversi dari Rupiah ke USD. [USD=Rp*13.000]

Thank You

ukrida.ac.id

UKRIDA
Universitas Kristen Krida Wacana

Belajar C++: Memahami Blok Percabangan pada C++

CYNTHIA HAYAT S.KOM., M.MSI

KRIDA WACANA CHRISTIAN UNIVERSITY
Faculty of Engineering and Computer Science
Departement of Information System

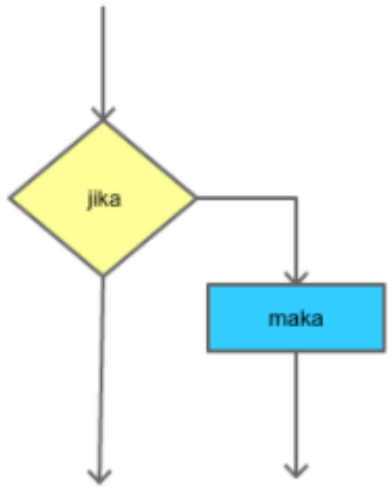
UKRIDA
Universitas Kristen Krida Wacana

Apa itu percabangan dan kenapa dinamakan percabangan?

Buat yang belum pernah kuliah atau belajar tentang algoritma dan flowchart, mungkin ini istilah yang baru pertama kamu dengar.

Isitilah ini sebenarnya untuk menggambarkan alur program yang bercabang.

Pada flow chart, logika "**jika...maka**" digambarkan dalam bentuk cabang.



Karena itu, ini disebut percabangan.

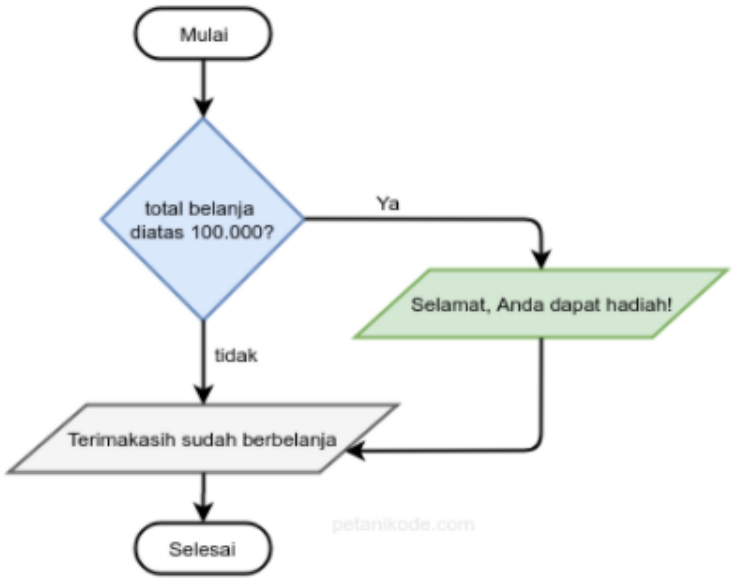
Selain percabangan, struktur ini juga disebut: *control flow*, *decision*, struktur kondisi, Struktur if, dsb.

Percabangan akan mampu membuat program berpikir dan menentukan tindakan sesuai dengan logika/kondisi yang kita berikan.

1. Percabangan *if*

Percabangan *if* merupakan percabangan yang hanya memiliki **satu blok pilihan** saat kondisi bernilai benar.

Coba perhatikan *flowchart* berikut ini:



Flowchart tersebut dapat kita baca seperti ini:

"Jika total belanja lebih besar dari Rp 100.000, Maka tampilkan pesan **Selamat, Anda dapat hadiah**"

Kalau dibawah Rp 100.000 bagaimana?

Ya pesannya tidak akan ditampilkan.

Mari kita coba dalam program C++.

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

int main(){

    cout << "=== Program Pembayaran ===" << endl;
    unsigned int total_belanja;

    cout << "Masukan total belanja: ";
    cin >> total_belanja;

    // menggunakan percabangan if
    if(total_belanja > 100000){
        cout << "Selamat! anda dapat hadiah" << endl;
    }

    cout << "Terimakasih sudah berbelanja di toko kami" << endl;

    return 0;
}
```

Perhatikan pada bagian ini:

```cpp
// menggunakan percabangan if
if(total_belanja > 100000){
    cout << "Selamat! anda dapat hadiah" << endl;
}
```

Inilah yang disebut blok program.

Blok program berisi sekumpulan ekpresi dan statement untuk dikerjakan oleh komputer.

Blok program pada C++, selalu diawali dengan tanda buka kurung kurawal { dan akan diakhiri dengan tutup kurung kurawal }.

Apabila di dalam blok hanya terdapat satu baris ekspresi atau statement, maka boleh tidak ditulis tanda kurungnya.

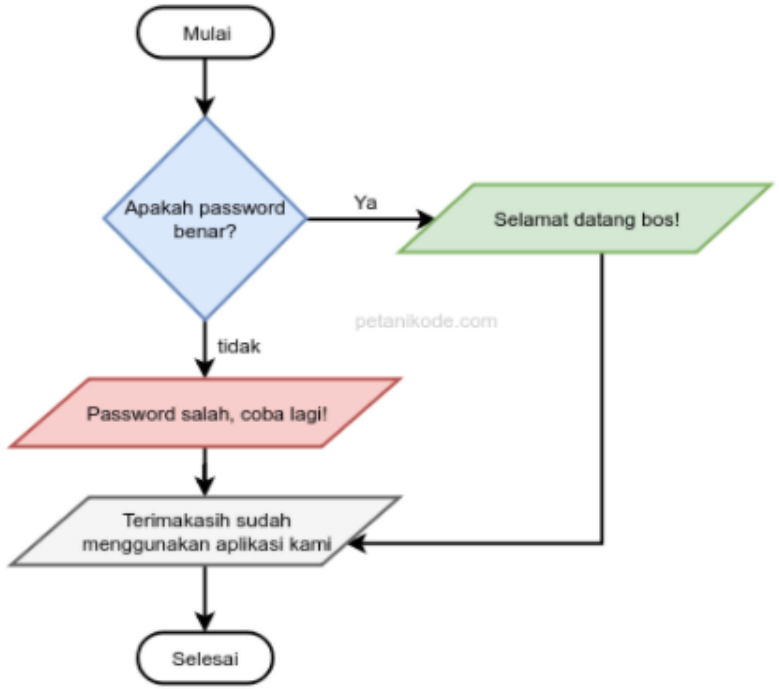
```cpp
if (total_belanja > 100000)
    cout << "Selamat! anda dapat hadiah" << endl;
```

2. Percabangan *if/else*

Percabangan *if/else* merupakan percabangan yang memiliki **dua blok pilihan**.

Blok pilihan pertama untuk kondisi **benar**, dan pilihan kedua untuk kondisi **salah** (*else*).

Coba perhatikan flowchart ini:



Ini adalah flowchart untuk mengecek password.

Apabila password benar, pesan yang ada pada blok hijau akan ditampilkan: **"Selamat datang bos!"**

Tapi kalau salah, maka pesan yang ada di blok merah yang akan ditampilkan: **"Password salah, coba lagi!"**

Kemudian, pesan yang berada di blok abu-abu akan tetap ditampilkan, karena dia bukan bagian dari blok percabangan *if/else*.

Perhatikan arah panahnya, setiap blok *if/else* mengarah ke sana...

Untuk lebih jelasnya, mari kita coba dalam program.

Buatlah file baru bernama `if_else.cpp`, kemudian isi dengan kode berikut:

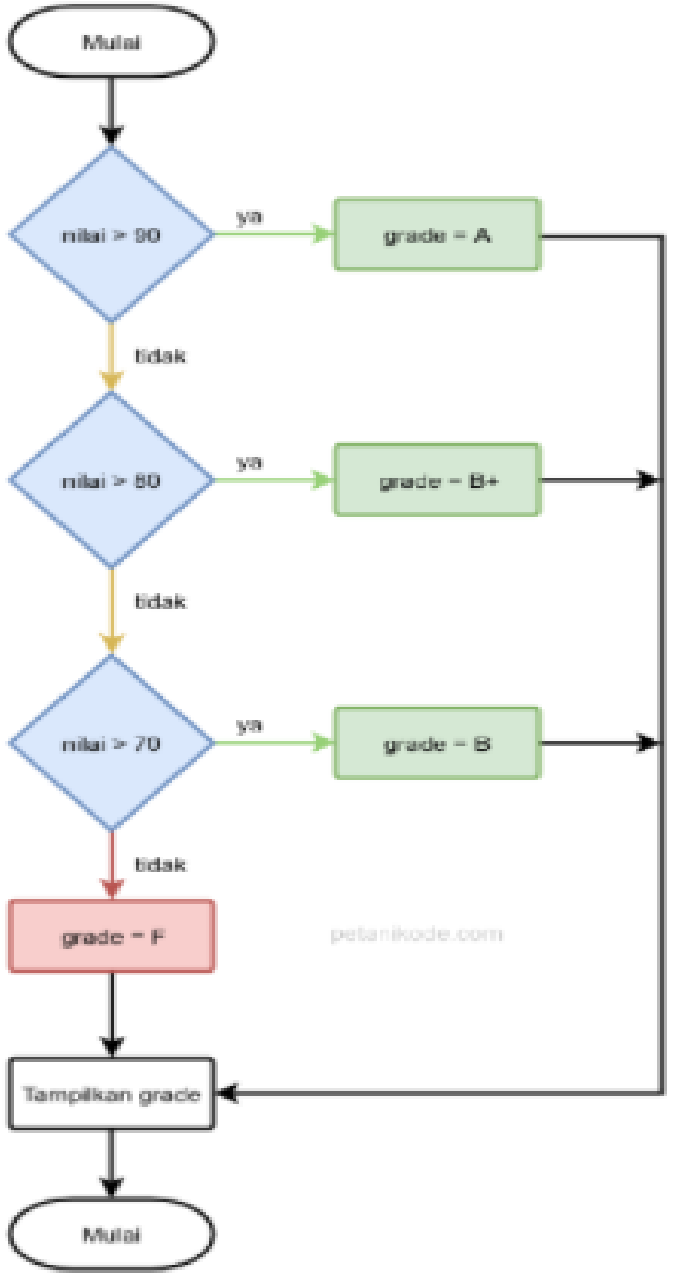
```cpp
#include <iostream>
using namespace std;

int main(){

    string password;

    cout << "========= Login =========" << endl;
    cout << "Masukan password: ";
    cin >> password;

    // percabangan if/else
    if (password == "kopi"){
        cout << "Selamat datang bos!" << endl;
    } else {
        cout << "Password salah, coba lagi!"  << endl;
    }

    cout << "Terimakasih sudah menggunakan aplikasi ini!" << endl;

    return 0;
}
```

3. Percabangan *if/else/if*

Percabangan *if/else/if* merupakan percabangan yang memiliki **lebih dari dua blok pilihan**.

Contoh Program:

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

int main(){

    int nilai;
    string grade;

    cout << "=== Program Grade Nilai ===" << endl;
    cout << "Inputkan nilai akhir: ";
    cin >> nilai;

    // menggunakan percabangan if/esle/if
    if (nilai >= 90) {
        grade = "A";
    } else if (nilai >= 80) {
        grade = "B+";
    } else if (nilai >= 70) {
        grade = "B";
    } else if (nilai >= 60) {
        grade = "C+";
    } else if (nilai >= 50) {
        grade = "C";
    } else if (nilai >= 40) {
        grade = "D";
    } else if (nilai >= 30) {
        grade = "E";
    } else {
        grade = "F";
    }

    cout << "Grade anda: " << grade << endl;

    return 0;
}
```



4. Percabangan *Switch/Case*

Percabangan *switch/case* adalah bentuk lain dari percabangan *if/else/if*.

Strukturnya seperti ini:

```
switch(variabel){
    case <value>:
        // blok kode
        break;
    case <value>:
        // blok kode
        break;
    default:
        // blok kode
}
```

Kita dapat membuat blok kode (`case`) sebanyak yang diinginkan di dalam blok switch.

Pada `<value>`, kita bisa isi dengan nilai yang nanti akan dibandingkan dengan `varabel`.

Setiap `case` harus diakhiri dengan `break`. Khusus untuk `default`, tidak perlu diakhiri dengan `break` karena dia terletak di bagian akhir.

Pemberian `break` bertujuan agar program berhenti mengecek `case` berikutnya saat sebuah `case` terpenuhi.

Contoh:

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

int main(){

    char grade;

    cout << "Inputkan grade: ";
    cin >> grade;

    switch (toupper(grade)){
        case 'A':
            cout << "Luar biasa!" << endl;
            break;
        case 'B':
        case 'C':
            cout << "Bagus!" << endl;
            break;
        case 'D':
            cout << "Anda lulus" << endl;
            break;
        case 'E':
        case 'F':
            cout << "Anda remidi" << endl;
            break;
        default:
            cout << "Grade Salah!" << endl;
    }

    return 0;
}
```

5. Percabangan dengan Opertor Ternary

Percabangan menggunakan opreator ternary merupakan bentuk lain dari percabangan if/else.

Bisa dibilang:

Bentuk singkatnya dari if/else.

Operator ternary juga dikenal dengan sebutan operator kondisi *(conditional operator)*.

Bentuk strukturnya seperti ini:

```
(kondisi) ? true : false
```

Bagian `kondisi` dapat kita isi dengan ekspresi yang menghasilkan nilai `true` dan `false`.

Lalu setlah tanda tanya `?` adalah bagian pilihan.

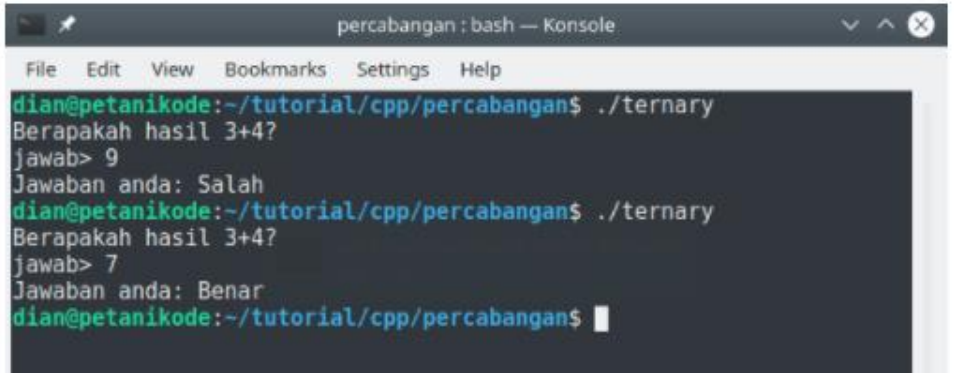
Jika `kondisi` bernilai benar, maka `true` yang akan dipilih. Tapi kalau salah, maka `false` yang akan dipilih.

Untuk lebih jelasnya, mari kita coba...

Buatlah file baru bernama `ternary.cpp`, kemudian isi dengan kode berikut:

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

int main(){

    int jawaban;

    cout << "Berapakah hasil 3+4?" << endl;
    cout << "jawab> ";
    cin >> jawaban;

    string hasil = (jawaban == 7) ? "Benar" : "Salah";

    cout << "Jawaban anda: " << hasil << endl;

    return 0;
}
```

Hasilnya:



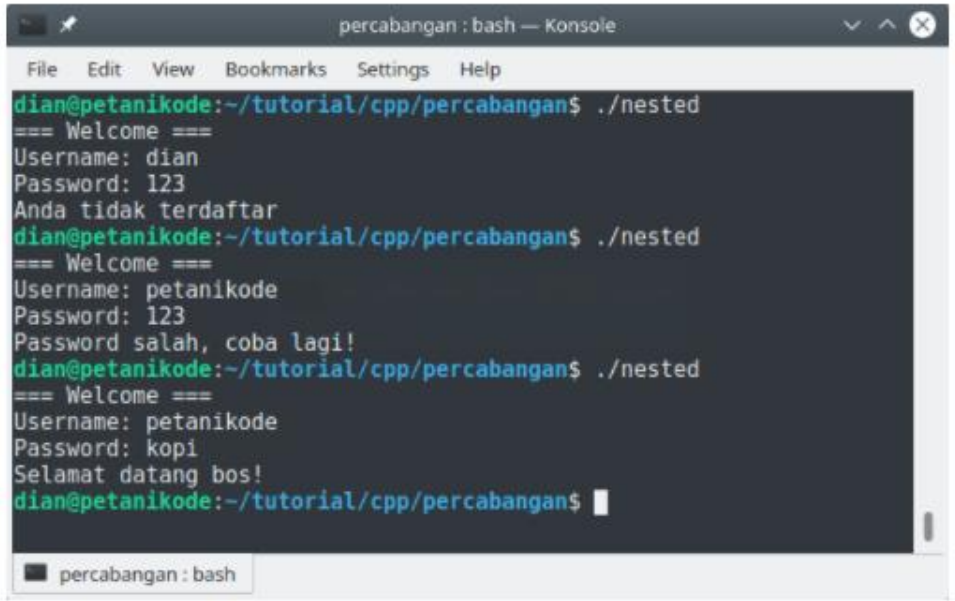
6. Percabangan Bersarang *(Nested)*

Kita juga dapat membuat blok percabangan di dalam percabangan. Ini disebut percabangan bersarng atau *nested if*.

Contoh: nested.cpp

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

int main(){

    string username, password;

    cout << "=== Welcome ===" << endl;
    cout << "Username: ";
    cin >> username;
    cout << "Password: ";
    cin >> password;

    if (username == "petanikode"){
        if (password == "kopi"){
            cout << "Selamat datang bos!" << endl;
        } else {
            cout << "Password salah, coba lagi!" << endl;
        }
    } else {
        cout << "Anda tidak terdaftar" << endl;
    }

    return 0;
}
```

Hasilnya:



Thank You

ukrida.ac.id

UKRIDA
Universitas Kristen Krida Wacana

Belajar C++: Memahami Blok Perulangan pada C++

CYNTHIA HAYAT S.KOM., M.MSI

KRIDA WACANA CHRISTIAN UNIVERSITY
Faculty of Engineering and Computer Science
Departement of Information System

UKRIDA
Universitas Kristen Krida Wacana

Apa yang akan kamu lakukan saat disuruh mencetak kalimat berulang-ulang?

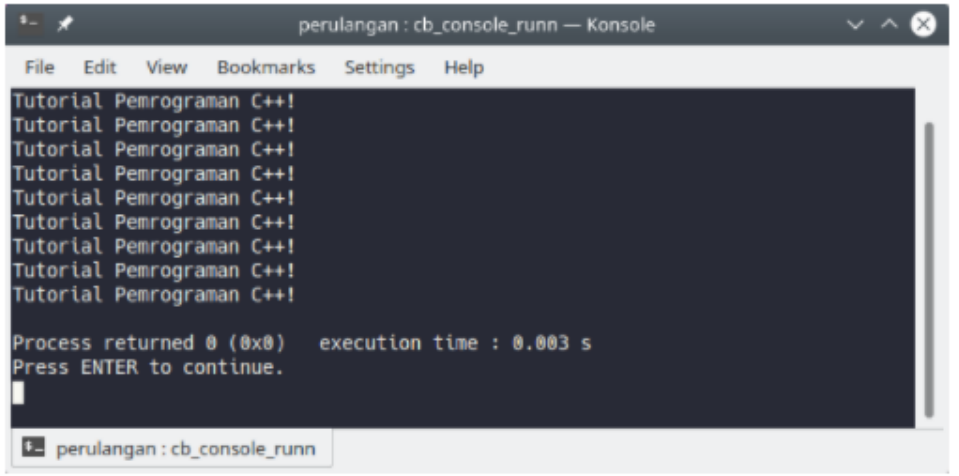
Misalnya:

Tolong tampilkan kalimat `"Tutorial Pemrograman C++!"` sebanyak 10x ke layar!

Mungkin kamu akan menggunakan `cout` sebanyak 10 kali seperti ini:

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

int main(){
    cout << "Tutorial Pemrograman C++!" << endl;
    cout << "Tutorial Pemrograman C++!" << endl;
    cout << "Tutorial Pemrograman C++!" << endl;
    cout << "Tutorial Pemrograman C++!" << endl;
    cout << "Tutorial Pemrograman C++!" << endl;
    cout << "Tutorial Pemrograman C++!" << endl;
    cout << "Tutorial Pemrograman C++!" << endl;
    cout << "Tutorial Pemrograman C++!" << endl;
    cout << "Tutorial Pemrograman C++!" << endl;
    cout << "Tutorial Pemrograman C++!" << endl;
    return 0;
}
```

Hasilnya:



Apakah boleh seperti ini?

Ya, boleh-boleh saja.

Tapi...

Bagaimana kalau nanti mau menampilkan sebanyak `1000` kali.

Pasti capek donk ngetiknya.

Karena itu, kita harus menggunakan **perulangan**.

Perulangan akan membantu kita mengeksekusi kode yang berulang-ulang, berapapun yang kita mau.

Ada empat macam bentuk perulangan pada C.

Secara umum, dibagi menjadi dua kelompok.

Yaitu: *counted loop* dan *uncounted loop*.

Perbedaanya:

- **Counted Loop** merupakan perulangan yang jelas dan sudah tentu banyak kali perulangannya.
- Sedangkan **Uncounted Loop**, merupakan perulangan yang tidak jelas berapa kali ia harus mengulang.

Perulangan yang termasuk dalam *Counted Loop*:

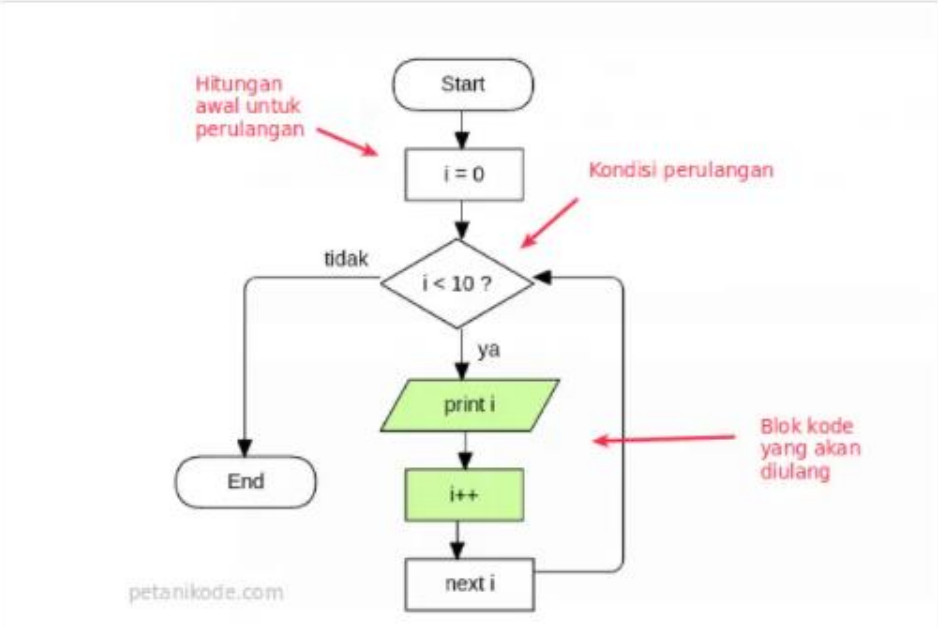
1. Perulangan For

Perulangan yang termasuk dalam *Uncounted Loop*:

1. Perulangan While
2. Perulangan Do/While

1. Blok Perulangan For

Perulangan `for` merupakan perulangan yang termasuk dalam *couted loop*, karena sudah jelas berapa kali ia akan mengulang.



Bentuknya kodenya seperti ini:

```
for(int i = 0; i < 10; i++){
    printf("Perulangan ke-%i\n", i);
}
```

Yang perlu diperhatikan adalah kondisi yang ada di dalam kurung setelah kata `for`.

Kondisi ini akan menentukan:

- Hitungan akan dimulai dari `0` (`i = 0`);
- Hitungannya sampai berapa? Sampai `i < 10`;
- Lalu di setiap perulangan `i` akan bertambah `+1` (`i++`).

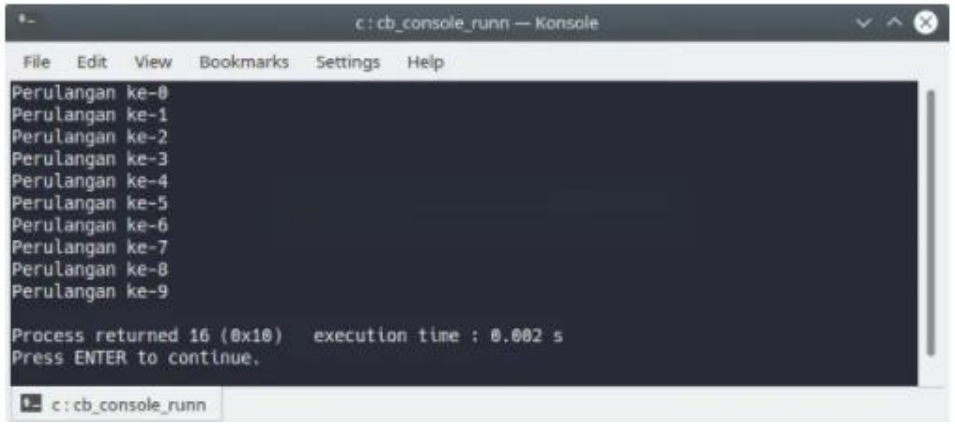
Variabel `i` pada perulangan `for` berfungsi untuk menyimpan nilai hitungan.

Jadi setiap perulangan dilakukan nilai `i` akan selalu bertambah satu. Karena kita menentukannya di bagian `i++`.

Contoh program loop dengan for:

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

int main(){
    for(int i = 0; i < 10; i++){
        printf("Perulangan ke-%i\n", i);
    }

    return 0;
}
```

Ini hasil outputnya:



Bagaimana kalau *counter* perulangannya dimulai dari akanga yang lebih besar sampai yang ke paling kecil?

Ini biasanya kita buat saat ingin menghitung mundur...

Caranya gampang.

Kita tinggal isi nilai counter dengan nilai terbesarnya.

Misalnya kita akan mulai hitungan dari `10` sampai ke `0`.

Maka nilai `counter`, kita isi awalnya dengan `10`.

Lalu di kondisi perbandingannya, kita berikan `counter > 0`. Artinya perulangan akan dilakukan selama nilai counter lebih besar dari `0`.

Lalu kita kurangi (`-1`) nilai counter di setiap perulangan (`counter--`).

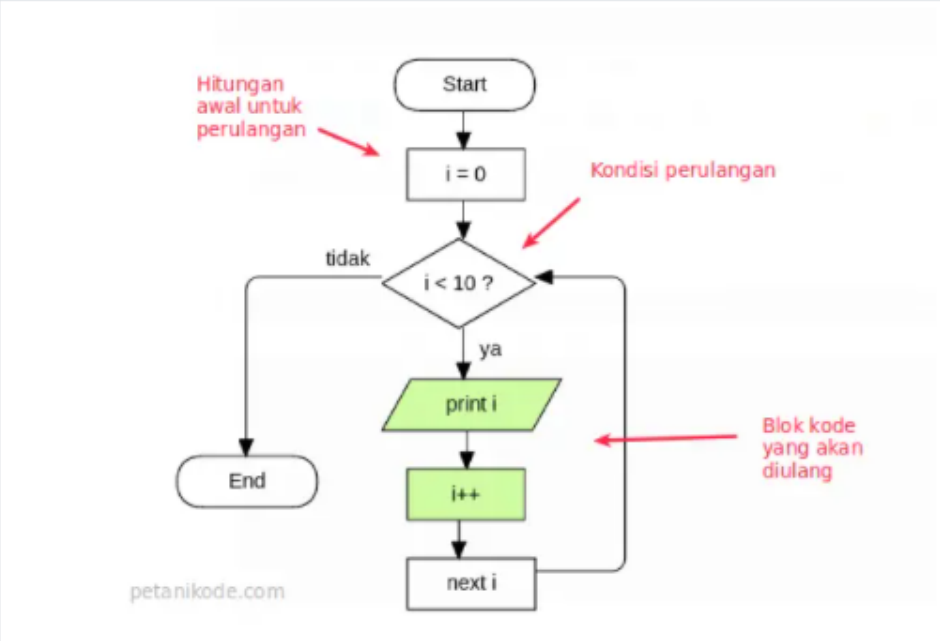
```
for(int counter = 10; counter > 0; counter--){
    printf("Perulangan ke-%i\n", counter);
}
```

2. Perulangan While pada C++

Perulangan `while` merupakan perulangan yang termasuk dalam perulangan *uncounted loop*.

Perulangan `while` juga dapat menjadi perulangan yang *counted loop* dengan memberikan *counter* di dalamnya.

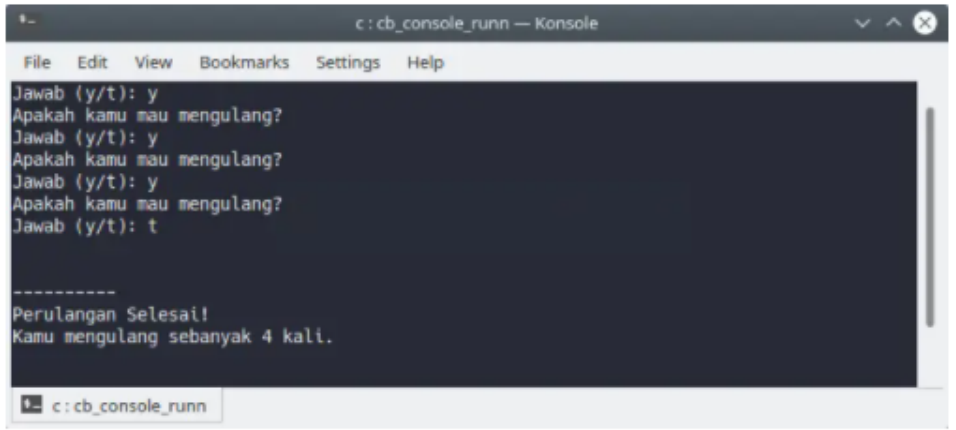
Bentuk flow chart-nya sama seperti *flow chart for*.



Untuk memahami perulangan ini...

...mari kita coba lihat contohnya:

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

int main(){
    char ulangi = 'y';
    int counter = 0;

    // perulangan while
    while(ulangi == 'y'){
        printf("Apakah kamu mau mengulang?\n");
        printf("Jawab (y/t): ");
        cin >> ulangi;

        // increment counter
        counter++;
    }

    printf("\n\n----------\n");
    printf("Perulangan Selesai!\n");
    printf("Kamu mengulang sebanyak %i kali.\n", counter);

    return 0;
}
```



Coba perhatikan blok kode `while`:

```
// perulangan while
while(ulangi == 'y'){
    printf("Apakah kamu mau mengulang?\n");
    printf("Jawab (y/t): ");
    cin >> ulangi;
    // increment counter
    counter++;
}
```

Di sana... Perulangan akan terjadi selama variabel `ulangi` brenilai `y`.

Lalu kita menggunakan fungsi `scanf()` untuk mengambil input.

Selama kita menjawab **y** pada input, maka perulangan akan terus dilakukan.

3. Perulangan Do/While pada C++

Perulangan `do/while` sama seperti perulangan `while`.

Perbedaanya:

Perulangan `do/while` akan melakukan perulangan sebanyak `1` kali terlebih dahulu, lalu mengecek kondisi yang ada di dalam kurung `while`.

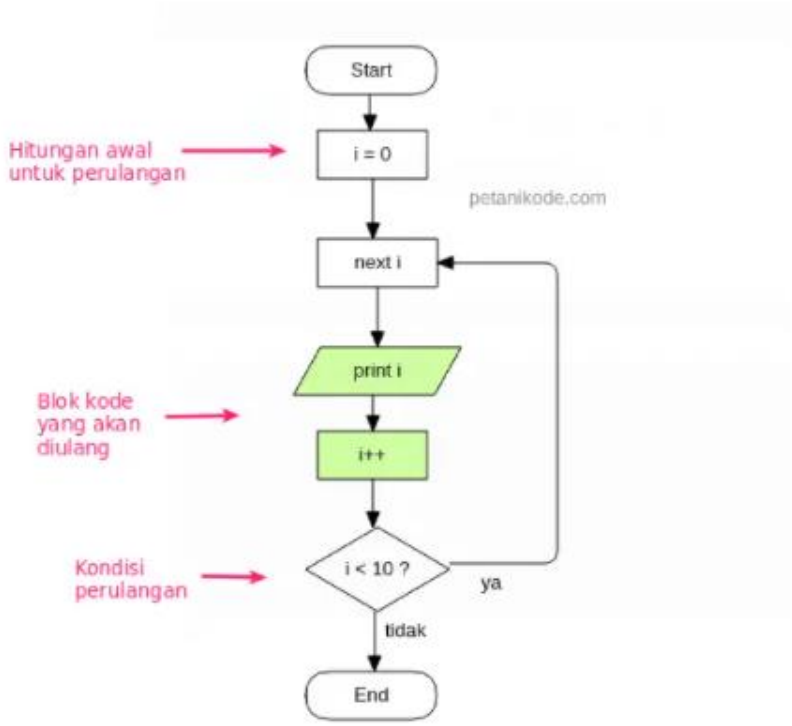
Bentuk kodenya seperti ini:

```
do {
    // blok kode yang akan diulang
} while (<kondisi>);
```

Jadi perbedaanya:

> Perulangan `do/while` akan mengecek kondisi di belakang (sesudah mengulang), sedangkan `while` akan mencek kondisi di depan atau awal (sbelum mengulang).

Flow chart perulangan do/while:



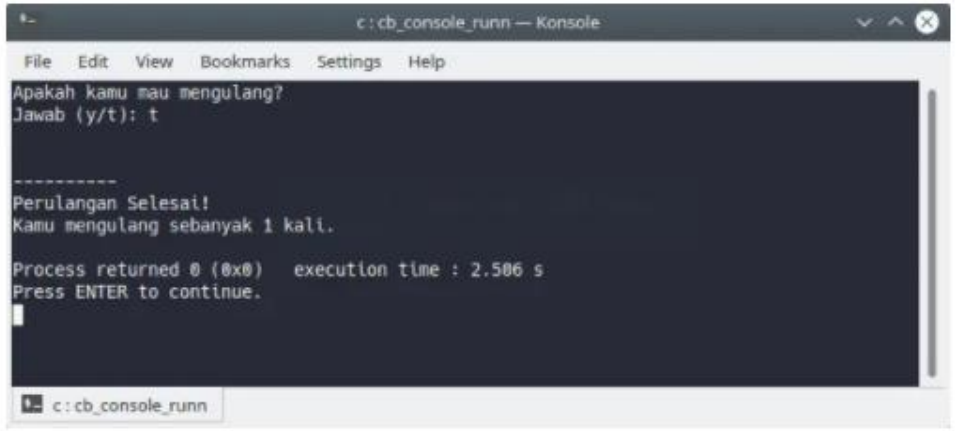
Mari kita coba lihat contohnya:

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

int main(){
    char ulangi = 'y';
    int counter = 0;

    do {
        printf("Apakah kamu mau mengulang?\n");
        printf("Jawab (y/t): ");
        cin >> ulangi;

        // increment counter
        counter++;
    } while(ulangi == 'y');

    printf("\n\n----------\n");
    printf("Perulangan Selesai!\n");
    printf("Kamu mengulang sebanyak %i kali.\n", counter);

    return 0;
}
```

Contoh tersebut sama seperti contoh pada perulangan `while`.

Saat perulangan pertama, cobalah untuk membatalkan perulangannya dengan menjawab `t`.

Maka hasilnya:



4. Perulangan Bersarang *(Nested Loop)*

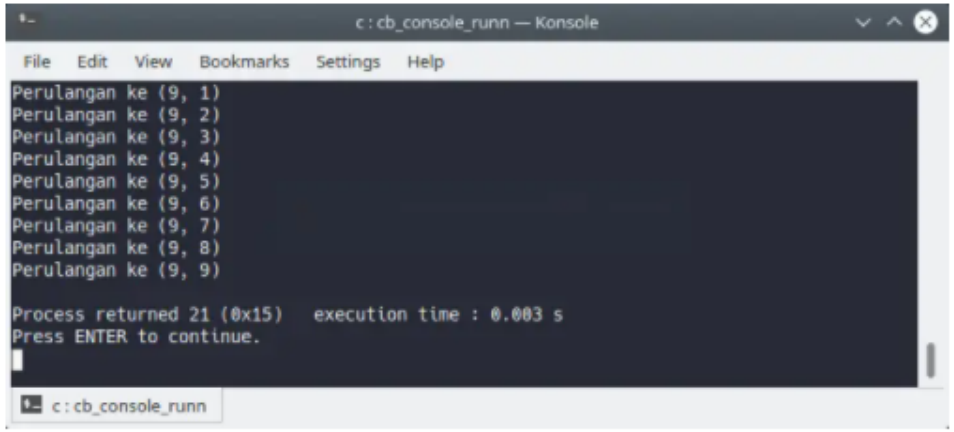
Di dalam blok perulangan, kita juga dapat membuat perulangan.

Ini disebut dengan *nested loop* atau perulangan bersarang atau perulangan di dalam perualangan.

Mari kita coba lihat contohnya:

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

int main(){
    for(int i = 0; i < 10; i++){
        for(int j = 0; j < 10; j++){
            printf("Perulangan ke (%d, %d)\n", i, j);
        }
    }

    return 0;
}
```

Hasilnya:



Pada perulangan tersebut, kita menggunakan dua perulangan `for`.

Perulangan pertama menggunakan variabel `i` sebagai *counter*, sedangkan perultangan kedua menggunakan variabel `j` sebagai *counter*.

Thank You

ukrida.ac.id

UKRIDA
Universitas Kristen Krida Wacana

Belajar C++: Memahami Prosedur dan Fungsi pada C++

CYNTHIA HAYAT S.KOM., M.MSI

KRIDA WACANA CHRISTIAN UNIVERSITY
Faculty of Engineering and Computer Science
Departement of Information System

UKRIDA
Universitas Kristen Krida Wacana

Dalam pemrograman, fungsi atau prosedur sering digunakan untuk membungkus program menjadi bagian-bagian kecil.

Tujuannya agar program tidak menumpuk pada fungsi `main()` saja.

Bayangkan saja, kalau program kita tambah besar dan kompleks..

Kalau semua kodenya ditulis di dalam fungsi `main()`, maka kita akan kesulitan membacanya.

Karena itu, kita harus menggunakan Fungsi.

Apa itu fungsi?

dan Bagaimana cara menggunakannya di C++?

Apa itu Fungsi?

Fungsi adalah sub-program yang bisa digunakan kembali baik di dalam program itu sendiri, maupun di program yang lain.

Fungsi dapat menerima input dan menghasilkan output.

Contoh fungsi yang sering kita buat adalah fungsi `main()`.

Fungsi ini memang wajib ada di setiap program C++, karena fungsi inilah yang akan dieksekusi pertama kali saat program berjalan.

Seperti yang saya bilang tadi:

“Kalau kita menumpuk semua kode program di dalam fungsi `main()`, maka program akan menjadi sulit terbaca.”

Karena itu, kita perlu membuat fungsi tersendiri agar program tidak berantakan.

Cara Membuat Fungsi pada Bahasa C++

Fungsi pada C++ dapat kita buat dengan cara seperti ini:

Fungsi biasanya akan mengembalikan sebuah nilai dari hasil prosesnya. Karena itu, kita harus menentukan tipe data untuk nilai yang akan dikembalikan.

Apabila fungsi tersebut tidak memiliki nilai kembalian, maka kita harus menggunakan tipe `void` untuk menyatakan kalau fungsi tersebut tidak akan mengembalikan nilai apa-apa.

Contoh:

```
void nama_fungsi(){
    cout << "Ini adalah sebuah fungsi\n";
}
```

Lalu untuk parameter bersifat opsional, boleh ada boleh tidak.

Tergantung dari fungsi yang dibuat. Jika fungsi itu membutuhkan input, maka kita harus membuatkan paramter.

Tapi kalau tidak menerima input apapun, ya tidak perlu dibuat.

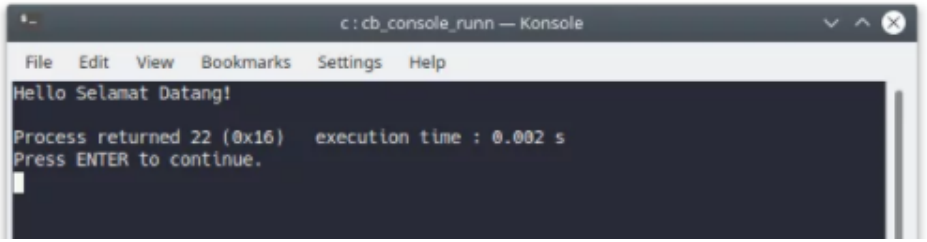
Fungsi yang tidak menerima input, kadang juga disebut dengan **prosedur**.

Sekarang mari kita coba membuat fungsi pada program C++.

Silahkan buat file baru bernama `contoh_fungsi.cpp` kemudian isi dengan kode berikut:

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

// membuat fungsi say_hello()
void say_hello(){
    cout << "Hello Selamat Datang!\n";
}

int main(){
    // memanggil fungsi say_hello()
    say_hello();

    return 0;
}
```

Hasilnya:



Deklarasi dan Definisi Fungsi

Pada contoh di atas, kita membuat fungsi dengan cara mendefinisikan langsung fungsinya.

Kita juga bisa membuatnya dengan deklarasi.

Contoh:

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

// deklarasi fungsi
void say_hello();

int main(){
    // memanggil fungsi say_hello()
    say_hello();
    say_hello();
    say_hello();

    return 0;
}

// Definisi fungsi
void say_hello(){
    cout << "Hello Selamat Datang!\n";
}
```

Apa bedanya dengan yang tadi?

Jika kita membuat fungsi secara **definisi**, kita harus membuat fungsinya di atas fungsi main.

Jika dibuat di bawah fungsi main, maka program akan error.

Soalnya program C++ dieksekusi dari atas ke bawah.

Tapi berkat **deklarasi**, masalah ini bisa teratasi.

Jadi kamu mau pilih cara yang mana?

Deklarasi dulu atau langsung definisikan fungsinya?

Fungsi dengan Parameter

Parameter adalah variabel yang menyimpan nilai untuk diproses di dalam fungsi.

Parameter berfungsi untuk menyimpan nilai yang akan diinputkan ke fungsi.

Contoh:

```
void say_hello(string name){
    cout << "Hello " << name << "!\n";
}
```

Perhatikan!

`name` adalah sebuah parameter dengan tipe string.

Parameter ini akan menyimpan nilai yang diinputkan ke fungsi `say_hello()`.

Lalu, bagaimana cara kita memberikan input ke fungsi?

Berikut caranya:

```
say_hello("Petani Kode");
```

Perhatikan!

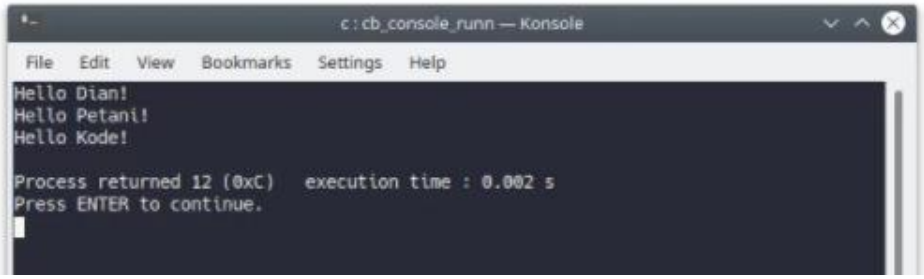
`"Petani Kode"` adalah nilai yang akan kita berikan ke fungsi.

Biar lebih paham... mari kita coba dalam program.

Silahkan buat program baru dengan nama `fungsi_parameter.cpp`, kemudian isi dengan kode berikut:

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

void say_hello(string name){
    cout << "Hello " << name << "!\n";
}

int main(){
    say_hello("Dian");
    say_hello("Petani");
    say_hello("Kode");
    return 0;
}
```

Hasilnya:



Fungsi yang Mengembalikan Nilai

Pada contoh di atas, kita memberikan nilai input ke fungsi berupa integer...

...lalu di dalamnya dilakukan operasi penjumlahan.

```
void add(int a, int b){
    printf("%d + %d = %d\n", a, b, a+b);
}
```

Fungsi ini tidak mengembalikan apa-apa, karena tipe data yang diberikan pada nilai kembalian adalah `void`.

Fungsi juga kadang harus menghasilkan output.

Mengapa?

Karena kadang kita membutuhkan hasil dari fungsi tersebut untuk digunakan pada proses berikutnya.

Kita bisa menggunakan kata kunci `return` untuk mengembalikan nilai dari fungsi.

Contoh:

```
int add(int a, int b){
    return a+b;
}
```

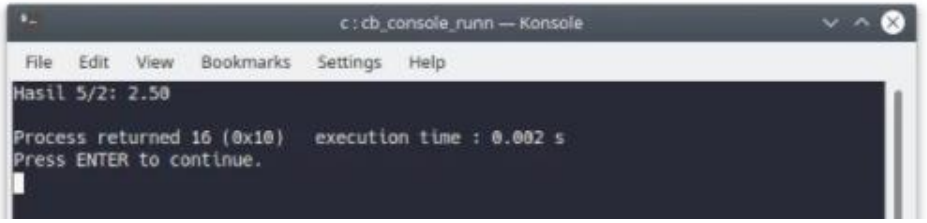
Maka fungsi `add()` akan mengembalikan nilai berupa integer dari hasil penjumlahan nilai `a` dan `b`.

Mari kita coba contoh yang lain...

Silahkan buat program baru bernama `fungsi_bagi.cpp`, kemudian isi dengan kode berikut:

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

float bagi(int a, int b){
    float hasil = (float)a / (float)b;
    return hasil;
}

int main(){
    printf("Hasil 5/2: %.2f\n", bagi(5, 2));
    return 0;
}
```

Hasilnya:



Variabel Lokal dan Variabel Global

Variabel lokal dan variabel global akan sering kita temukan dalam pembuatan fungsi.

Variabel global adalah variabel yang bisa diakses dari semua fungsi. Sedangkan variabel lokal adalah variabel yang hanya bisa diakses dari dalam fungsi itu sendiri.

Contoh:

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

// membuat variabel global
int nilai = 9;

int main(){
    // membuat variabel lokal
    int nilai = 7;

    // mencetak variabel
    printf("Nilai: %d\n", nilai);

    return 0;
}
```

Pada contoh di atas, kita membuat variabel global bernama `nilai`.

Fungsi ini berada di luar fungsi main.

Lalu di dalam fungsi `main()`, kita membuat variabel lagi bernama `nilai` dengan nilai yang berbeda.

Variabel yang ada di dalam fungsi `main()` adalah variabel lokal.

Lalu, pertanyaanya:

Berapakah hasil outputnya?

Jawabannya: `7`

Mengapa bisa `7`?

Karena variabel `nilai` kita buat ulang di dalam fungsi main.

Thank You

ukrida.ac.id

UKRIDA
Universitas Kristen Krida Wacana

Apa itu Struktur Data?

CYNTHIA HAYAT S.KOM., M.MSI
KRIDA WACANA CHRISTIAN UNIVERSITY
Faculty of Engineering and Computer Science
Departement of Information System

UKRIDA
Universitas Kristen Krida Wacana

Struktur data adalah cara penyimpanan, penyusunan dan pengaturan data di dalam media penyimpanan komputer sehingga data tersebut dapat digunakan secara efisien

Struktur data atau data struktur berfungsi Untuk menyimpan data dalam bentuk yang efisien, Untuk memudahkan pembacaan data, Membantu kinerja algoritma.

Kemudian dari Stuktur Data sendiri berguna untuk pengorganisasian data yang disimpan agar lebih mudah untuk dibaca di modifikasi, dan diolah kembali dikemudian waktu. Akan tetapi kita perlu ingat Struktur Data mengalami evolusi dan setiap pengembangannya selalu memiliki trade-off.

Inilah beberapa contoh Struktur Data:

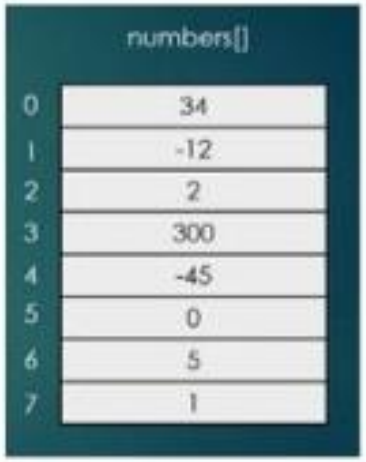
1. Array
2. Linked List: Single dan Double
3. Stacks
4. Queue
5. Tree
6. Graph

Lalu kenapa kita perlu belajar Struktur data ? akan lebih mudah bayangkan jika kamus tidak memiliki urutan huruf yang baik, bayangkan jika jutaan data pengguna facebook tidak tersimpan dengan rapi, dan bayangkan jika data di GPS tidak menyimpan data kota dengan baik.

1. Array

Array adalah sebuah koleksi dari elemen atau suatu nilai yang dapat diindentifikasi dengan menggunakan indeks yang memiliki tipe data yang sama dan dinyatakan dengan nama yang sama. array memungkinkan untuk menyimpan data maupun referensi objek dalam jumlah banyak dan terindeks. Array menggunakan indeks integer untuk menentukan urutan elemen-elemennya, dimana elemen pertamanya dimulai dari indeks 0,elemen kedua memiliki indeks 1, dan seterusnya.jadi dengan adanya indeks, dimungkinkan untuk pengaksesan secara acak (random Access)

Contoh Array :



Numbers[3]=>300
Numbers[6]=>5

Salah satu ciri Array adalah, perlu ditentukan ukurannya sebelum dapat digunakan. Karena perlu pengalokasian di memory.

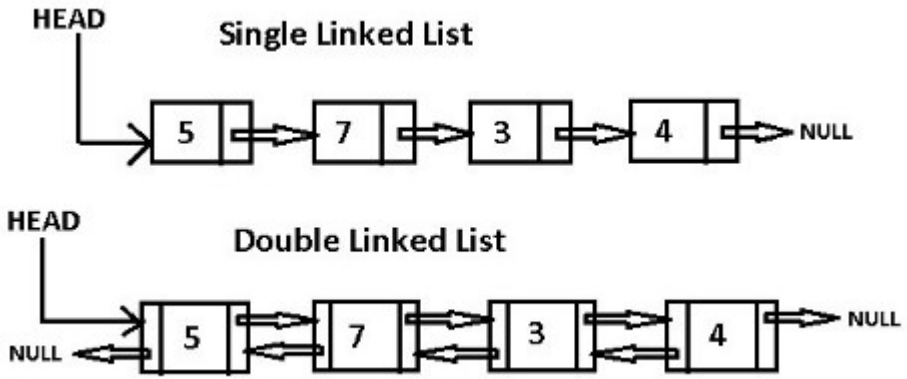
Ketika Array yang kita buat sudah penuh, maka perlu inisiasi baru untuk Array dengan ukuran yang lebih besar. Dengan catatan, tipe data yang dapat disimpan hanya satu jenis saja

Di dalam Array terdapat beberapa Operasi yang perlu kita ketahui, yaitu:

- **Insert** = Data pada array dapat ditambahkan baik di awal, tengah, maupun akhir.
- **Search/Read** = Untuk mengambil data pada array kita lakukan dengan menggunakan index dari data tersebut.
- **Remove** = Mengapus data pada array dapat dilakukan berdasarkan index ataupun langsung tertuju pada data apa yang akan kita hapus.

2. Linked List

Linked List adalah Struktur Data yang terdiri dari suatu node dan *referensi/pointer* yang menghubungkan satu node dengan yang lain. Dimana Node terakhir memiliki referensi ke *null.* Setiap node memiliki data dan referensi ke node selanjutnya (*singly*) dan atau ke node sebelumnya (*doubly*). Digunakan sebagai dasar implementasi *Stacks* dan *Queue.* Tidak ada unsur *Random Access* via indeks seperti Array dan setiap operasi biasanya perlu operasi sekuensial dari node awal.



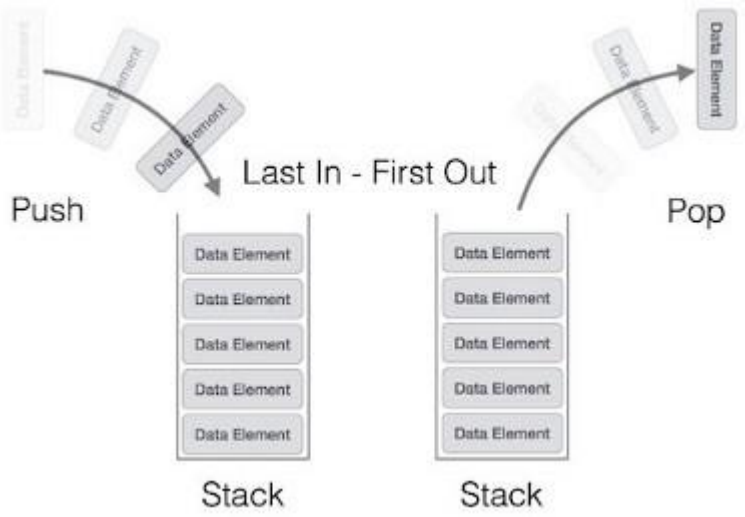
Enaknya Linked List bersifat dinamis secara ukuran, Alokasi penggunaan *memory* yang dibutuhkan pada *run-time,* Mudah di implementasikan. Akan tetapi kurangnya Linked List lebih boros *memory,* pembacaan node hanya bisa melalui proses sekuensial dari awal, tidak layaknya via indeks di Array. Pada Singly Linked List, mustahil bisa melakukan traversal dari belakang ke depan, namun dapat disolusikan dengan Doubly Linked List

Di dalam Linked List pun ada beberapa Operasi yang perlu diketahui yaitu:

- Insert di awal
- Insert di akhir
- Remove sebuah nilai
- Remove node pertama
- Traversing forward and reverse

3. Stacks

Stack merupakan sebuah kumpulan data atau item dengan cara penambahan item baru serta penghapusan, selalu terjadi pada tempat atau ujung yang sama. Stack ini biasa di analogikan seperti tumpukan pada piring. Dimana kita mengambil maupun meletakkan piring selalu pada sisi atasnya. Stack ini memiliki konsep LIFO (last in first out) atau dalam bahasa indonesianya adalah data yang terakhir masuk, maka ialah yang pertama akan dikeluarkan.

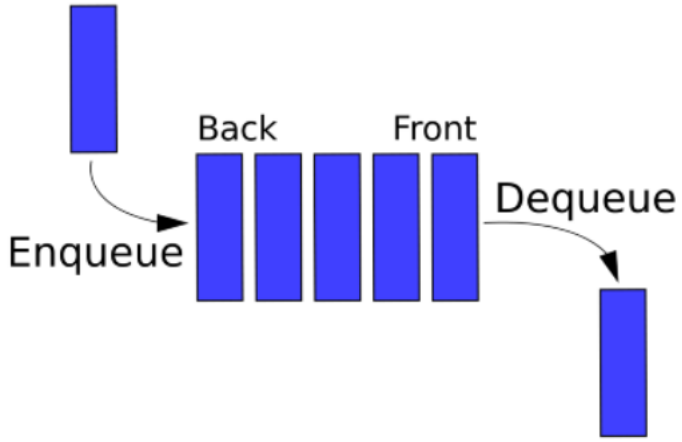


stack ini memiliki beberapa operasi, yaitu:

- **push(item)** menambahkan suatu item baru ke atas (top) dari stack. Perlu item dan tidak mengembalikan apapun.
- **pop()** menghapus item teratas dari stack. Tidak perlu parameter dan mengembalikan item. Stack berubah.
- **peek()** mengintip top item dari stack tetapi tidak menghapusnya. Tidak memerlukan parameter dan stack tidak berubah.

4. Queue

Queue merupakan koleksi item yang cara penambahan itemnya terjadi pada sebuah ujung yang biasa disebut sebagai “ekor” atau (rear) dan untuk penghapusannya, terjadi pada ujung yang satunya. Atau biasa kita beri nama “kepala” atau (head). Jadi konsep dari queue ini menggunakan konsep layak nya FIFO yang merupakan kepanjangan dari First in First out. Dalam kehidupan sehari — hari, konsep ini biasa di analogikan sebagai sebuah antrian. Dimana setiap orang yang datang terlebih dahulu, maka ia lah yang akan di layani terlebih dahulu. Nah konsep ini sangat berbeda dengan konsep yang ada pada stack. Namun, sama seperti stack, kelas ini juga memiliki beberapa operasi.



Operasi tersebut adalah sebagai berikut :

- **enqueue(item)** menambahkan suatu item baru ke ujung saru antrian. Perlu item dan tidak mengembalikan sesuatu.
- **dequeue()** menghapus item depan dari antrian. Tidak memerlukan parameter dan mengembalikan itemnya. Antrian termodifikasi.
- **peek()** mengintip top item dari stack tetapi tidak menghapusnya. Tidak memerlukan parameter dan stack tidak berubah.

5. Trees

Tree merupakan salah satu bentuk struktur data tidak linear yang menggambarkan hubungan yang bersifat hirarkis (hubungan one to many) antara elemen-elemen. Tree bisa didefinisikan sebagai kumpulan simpul/node dengan satu elemen khusus yang disebut Root dan node lainnya terbagi menjadi himpunan-himpunan yang saling tak berhubungan satu sama lainnya (disebut subtree). Untuk jelasnya, di bawah akan diuraikan istilah-istilah umum dalam tree :

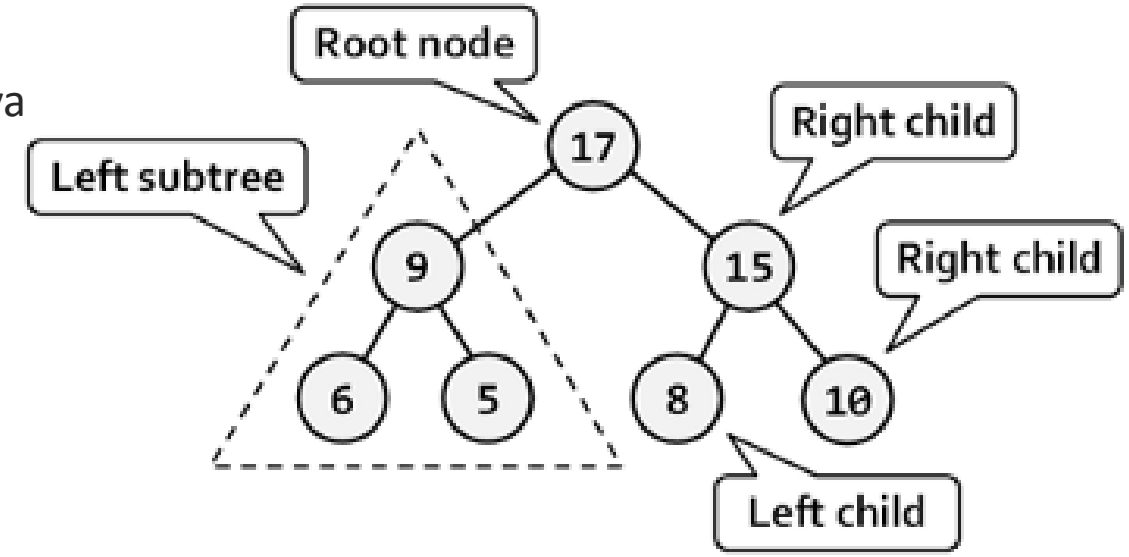
a) Prodecessor : node yang berada diatas node tertentu.
b) Successor : node yang berada di bawah node tertentu.
c) Ancestor : seluruh node yang terletak sebelum node tertentu dan terletak pada jalur yang sama.
d) Descendant : seluruh node yang terletak sesudah node tertentu dan terletak pada jalur yang sama.
e) Parent : predecssor satu level di atas suatu node.
f) Child : successor satu level di bawah suatu node.

g) Sibling : node-node yang memiliki parent yang sama dengan suatu node.
h) Subtree : bagian dari tree yang berupa suatu node beserta descendantnya dan memiliki semua karakteristik dari tree tersebut.
i) Size : banyaknya node dalam suatu tree.
j) Height : banyaknya tingkatan/level dalam suatu tree.
k) Root : satu-satunya node khusus dalam tree yang tak punya predecssor.
l) Leaf : node-node dalam tree yang tak memiliki seccessor.
m) Degree : banyaknya child yang dimiliki suatu node.

A. Binary Tree

Binary Tree adalah tree dengan syarat bahwa tiap node hanya boleh memiliki maksimal dua subtree dan kedua subtree tersebut harus terpisah. Sesuai dengan definisi tersebut, maka tiap node dalam binary tree hanya boleh memiliki paling banyak dua child.

Jenis Binary Tree secara strukturnya ada 3 jenis yaitu:
1. Strict Binary Tree dimana harus punya 2 anak atau gak punya sama sekali
2. Completed Binary Ttee dimana sebelum anaknya 2 belum bisa nambah anak disebelah kanan, jika sudah 2 yang kanan boleh puyna anak.
3. Perfect Binary Tree dimana kiri kanan harus punya anak dengan jumlah sama dan seluruh segitiga seimbang.Jenis



dalam Binary Tree ada Operasi-operasi yang mesti diketahui sebelumnya, yaitu :

- Create : Membentuk binary tree baru yang masih kosong.
- Clear : Mengosongkan binary tree yang sudah ada.
- Empty : Function untuk memeriksa apakah binary tree masih kosong.
- Insert : Memasukkan sebuah node ke dalam tree. Ada tiga pilihan insert: sebagai root, left child, atau right child. Khusus insert sebagai root, tree harus dalam keadaan kosong.
- Find : Mencari root, parent, left child, atau right child dari suatu node. (Tree tak boleh kosong).
- Update : Mengubah isi dari node yang ditunjuk oleh pointer current. (Tree tidak boleh kosong)
- Retrieve : Mengetahui isi dari node yang ditunjuk pointer current. (Tree tidak boleh kosong)
- DeleteSub : Menghapus sebuah subtree (node beserta seluruh descendantnya) yang ditunjuk current. Tree tak boleh kosong. Setelah itu pointer current akan berpindah ke parent dari node yang dihapus.
- Characteristic : Mengetahui karakteristik dari suatu tree, yakni : size, height, serta average lengthnya. Tree tidak boleh kosong.
- Traverse : Mengunjungi seluruh node-node pada tree, masing-masing sekali. Hasilnya adalah urutan informasi secara linier yang tersimpan dalam tree. Adatiga cara traverse : Pre Order, In Order, dan Post Order.

Langkah-Langkahnya Traverse :

1. PreOrder : Cetak isi node yang dikunjungi, kunjungi Left Child, kunjungi Right Child.
2. InOrder : Kunjungi Left Child, Cetak isi node yang dikunjungi, kunjungi Right Child.
3. PostOrder : Kunjungi Left Child, Kunjungi Right Child, cetak isi node yang dikunjungi.

7. Graph

Graph merupakan sekumpulan dari node dan sekumpulan garis(edge) bersifat non-linier yang kemungkinan bisa hirarki bisa juga tidak. Dan pemakaian graph dalam dunia nyata biasanya seperti lingkaran pertemanan, dimana dari lingkaran tersebut kita bisa mempresentasikan dari satu teman ke teman yang lainnya.

Berbeda dengan tree yang memiliki jumlah edge sebanyak node-1, pada graph jumlah edge jauh lebih bebas, karena edge yang bisa berasal dari node mana saja, bahkan self loop.

Graph bisa dibedakan dari edge nya, ada yang memiliki arah (*directed*) dan tidak memiliki arah (*undirected*).
Graph kadang memiliki *cost* di setiap edge nya (*weight*) ada juga yang tidak memilikinya (*unweight*). bertujuan mencari jalur terendek degan menggunakan tranversal dengan mempertimbangkan *weight.*

Berikutnya yang perlu diperhatikan dari graph adalah *adjacency* (node yang dekat/berhubungan dengan node lainnya). A*djacency* dapat dibuat dengan dua pendekatan, yaitu:

- *Adjacency List*
  Digunakan ketika jumlah edge nya mendekati jumlah minimum dari edge yang bisa dibuat.
- *Adjacency Matrix*
  Digunakan ketika jumlah edge nya mendekati jumlah maksimum dari edge yang bisa dibuat.

Pada graph terdapat dua jenis traversal, yaitu:

- DFS (*Depth First Search*), melakukan kunjungan ke node-node dengan cara mengunjungi node terdalam/kebawah ,setelah itu mencari ketempat yang lainnya , dan sistemnya menggunakan stack
- BFS (*Breadth First Search*), melakukan visit ke node- node dengan cara melebar kesamping, dan sistemnya menggunakan queue

Thank You

ukrida.ac.id

UKRIDA
Universitas Kristen Krida Wacana

Menggunakan Array untuk Menyimpan Banyak Data

CYNTHIA HAYAT S.KOM., M.MSI

KRIDA WACANA CHRISTIAN UNIVERSITY
Faculty of Engineering and Computer Science
Departement of Information System

UKRIDA
Universitas Kristen Krida Wacana

Apa yang akan kamu lakukan jika diminta untuk menyimpan banyak data di program?

Misalkan kita ingin menyimpan nama-nama teman untuk data kontak.

Mungkin saja, kita akan menyimpannya seperti ini:

```
string namaKontak1 = "Ayu";
string namaKontak2 = "Bunga";
string namaKontak3 = "Cyntia";
string namaKontak4 = "Deni";
string namaKontak5 = "Elisa";
```

Hal ini boleh-boleh saja..

Akan tetapi, masalahnya:

“Gimana nanti kalau ada banyak sekali data, pasti capek bikin variabel terus?”

Karena itu, kita membutuhkan Array.

Apa itu Array, dan bagaimana cara menggunakannya?

Mari kita bahas...

Apa itu Array?

Array merupakan struktur data yang digunakan untuk **menyimpan sekumpulan data** dalam satu tempat.

Setiap data dalam Array memiliki indeks, sehingga kita akan mudah memprosesnya.

Indeks array selalu dimulai dari angka nol (0).

Pada teori struktur data...

...ukuran array akan bergantung dari banyaknya data yang disimpan di dalamnya.

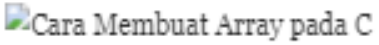
Cara Membuat Array pada C++

Pada C++, array dapat kita buat dengan cara seperti ini.

```
// membuat array kosong dengan tipe data integer dan panjang 10
int nama_array[10];

// membuat array dengan langsung diisi
int nama_arr[3] = {0, 3, 2}
```

Cara membaut array hampir sama seperti cara membuat variabel biasa.

Bedanya... pada array kita harus menentukan panjangnya.


Cara Membuat Array pada C

Cara Mengambil Data dari Array

Seperti yang sudah kita kethaui...

Array akan menyimpan sekumpulan data dan memberinya nomer indeks agar mudah diakses.

Indeks array selalu dimauli dari nol `0`.

Misalkan kita punya array seperti ini:

```
char huruf[5] = {'a', 'b', 'c', 'd', 'e'};
```

Bagaimana cara mengambil huruf `c`?

Jawabannya:

```
huruf[2];
```

Mengapa bukan `huruf[3]`?

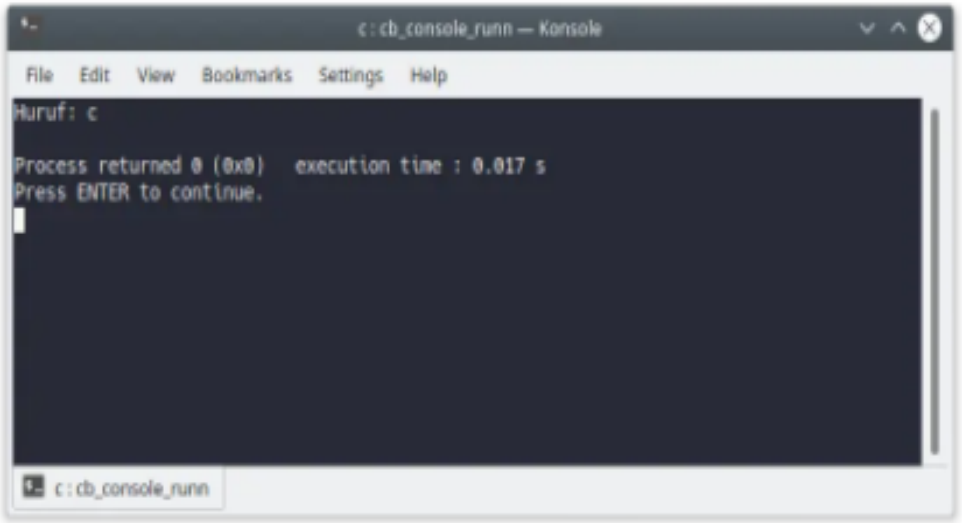
Ingat: indeks array selalu dimulai dari nol.

Ingat: indeks array selalu dimulai dari nol.

Biar lebih jelas, mari kita coba dalam program. Silahkan buat file baru dengan nama `contoh_array.cpp`, kemudian isi dengan kode berikut:

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

int main(){
    char huruf[5] = {'a', 'b', 'c', 'd', 'e'};

    // mengambil data pada array
    cout << "Huruf: " << huruf[2] << endl;

    return 0;
}
```

Maka hasilnya:



Mengisi Ulang Data pada Array

Data pada array dapat kita isi ulang dengan cara seperti ini:

```
huruf[2] = 'z';
```

Maka isi array `huruf` pada indeks ke-2 akan bernilai `'z'`.

Kalau tidak percaya...

Kamu bisa coba buktikan sendiri.

Silahkan ubah kode pada `contoh_array.cpp` menjadi seperti ini:

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

int main(){
    // isi awal array
    char huruf[5] = {'a', 'b', 'c', 'd', 'e'};

    // mengubah isi data array
    huruf[2] = 'z';

    // mencetak isi array
    cout << "Huruf: " << huruf[2] << endl;

    return 0;
}
```

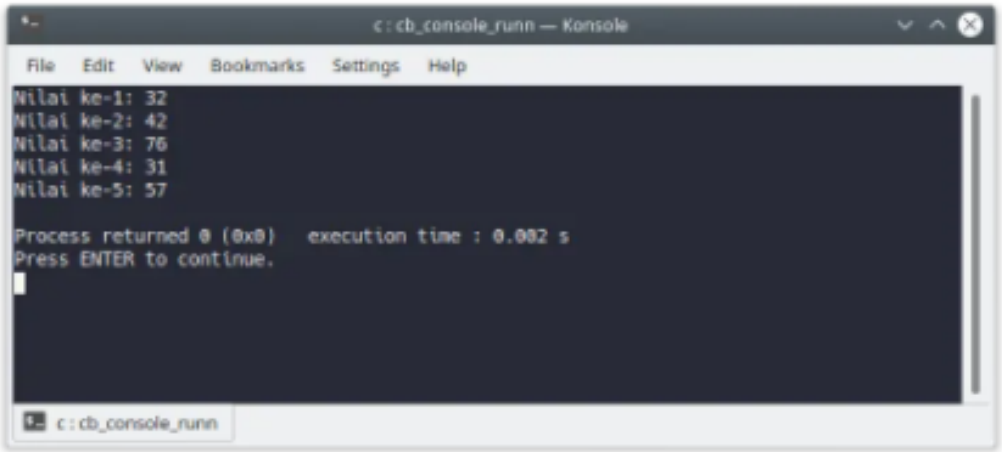
Hasil outputnya:

```
Huruf: z
```

Contoh lain: array_data.cpp

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

int main(){
    // membuat array kosong
    int nilai[5];

    // mengisi array
    nilai[0] = 32;
    nilai[1] = 42;
    nilai[2] = 76;
    nilai[3] = 31;
    nilai[4] = 57;

    // mencetak isi array
    cout << "Nilai ke-1: " << nilai[0] << endl;
    cout << "Nilai ke-2: " << nilai[1] << endl;
    cout << "Nilai ke-3: " << nilai[2] << endl;
    cout << "Nilai ke-4: " << nilai[3] << endl;
    cout << "Nilai ke-5: " << nilai[4] << endl;

    return 0;
}
```

Hasilnya:



Menggunakan Perulangan dan Array

Pada contoh di atas, kita menggunakan perintah `cout` secara berulang untuk mencetak semua isi array.

```
cout << "Nilai ke-1: " << nilai[0] << endl;
cout << "Nilai ke-2: " << nilai[1] << endl;
cout << "Nilai ke-3: " << nilai[2] << endl;
cout << "Nilai ke-4: " << nilai[3] << endl;
cout << "Nilai ke-5: " << nilai[4] << endl;
```

Kalau isi array-nya ada ribuan, apa kamu akan sanggup menulis ini berulang-ulang?

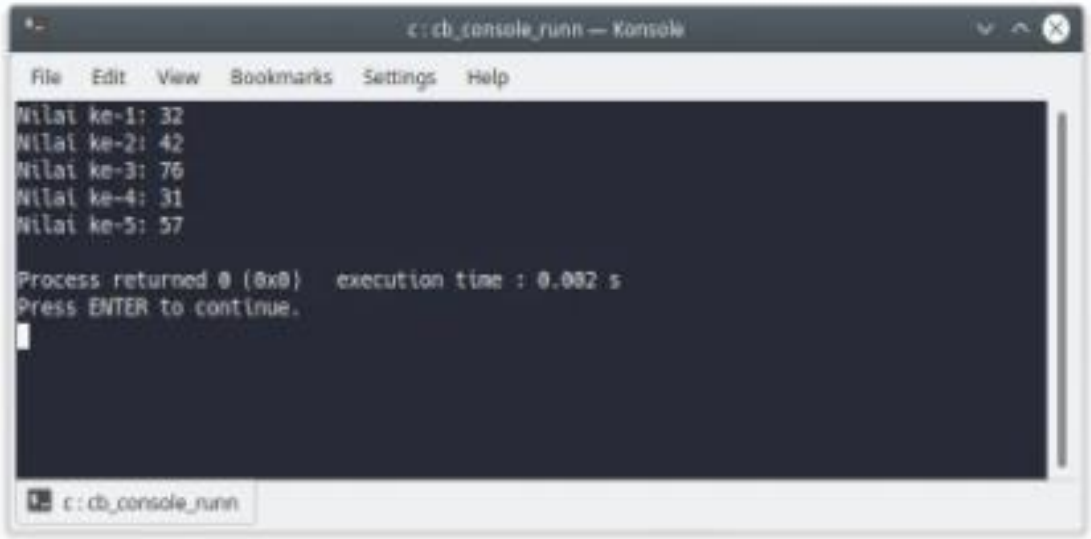
Tentu saja tidak!

Karena itu, kita bisa memanfaatkan perulangan untuk mencetaknya.

Contoh: `array_loop.cpp`

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

int main(){
    // membuat array kosong
    int nilai[5];

    // mengisi array
    nilai[0] = 32;
    nilai[1] = 42;
    nilai[2] = 76;
    nilai[3] = 31;
    nilai[4] = 57;

    // mencetak isi array dengan perulangan
    for(int i; i < 5; i++){
        printf("Nilai ke-%d: %d\n", i, nilai[i]);
    }

    return 0;
}
```

Hasilnya:



Jauh lebih sederhana 'kan...

Tapi ada yang masih kurang.

Pada perulangan di atas kita memberikan batas maksimal secara manual, yaitu: `i < 5`.

Nilai `5` seharusnya diganti dengan ukuran atau jumlah isi array.

Sehingga akan menjadi seperti ini:

```
int length = sizeof(nilai) / sizeof(*nilai);
for (int i = 0; i < length; i++){
    printf("Nilai ke-1: %d\n", nilai[i]);
}
```

Mengambil Panjang Array

Ada dua cara untuk mendapatkan panjang array:

1. Partama cara seperti di atas (seperti bahasa C);
2. Menggunakan Class array dari C++.

Kita bahas dulu cara yang pertama..

Pada contoh di atas, kita menggunakan fungsi `sizeof()` untuk mengambil panajang atau ukuran array.

Fungsi `sizeof()` sebenarnya akan mengambil ukuran memori dari array.

Misalkan saya punya array seperti ini:

```
int nilai[2] = {1, 2};
```

Isi array-nya berupa integer, tipe data integer memiliki ukuran `4` byte dalam memori. Maka ukuran array tersebut adalah `8` byte.

Lalu bagaimana cara mendapatkan banyaknya isi array?

Karena di bahasa C belum memiliki fungsi khusus untuk mengambil banyaknya isi array.

Kita harus membaginya dengan panjang pointernya.

Contoh:

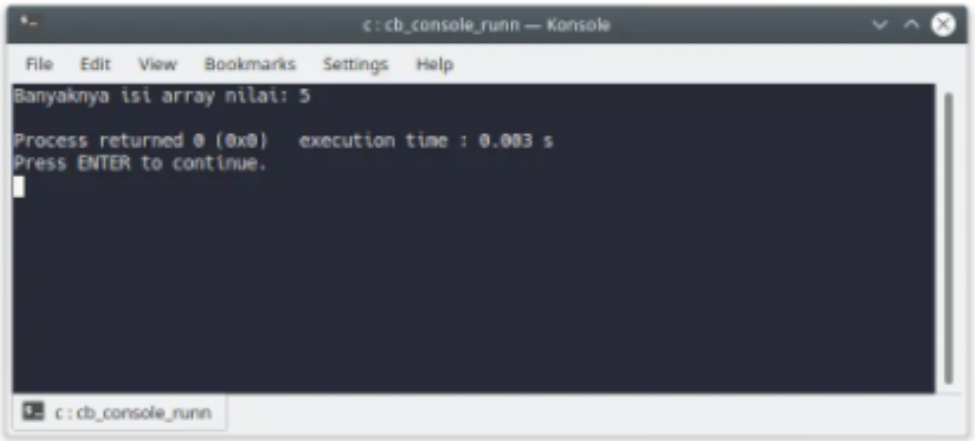
```
int array[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

/* ukuran array dalam byte */
int size = sizeof(array);

/* banyaknya isi array `array` */
int length = sizeof(array) / sizeof(*array);
```

Mari kita coba...

Buatlah program dengan nama `panjang_array.cpp`, kemudian isi dengan kode berikut:

```cpp
#include <stdio.h>

void main(){
    // membuat array
    string contact[] = {"Ami", "Ayu", "Budi", "Agus", "Mila"};

    //  mengambil banyaknya isi array
    int length = sizeof(contact) / sizeof(*contact);

    printf("Banyaknya isi array nilai: %d\n", length);
}
```

Hasilnya:



Cara kedua, kita bisa gunakan *Class Template* dari C++. *Class Template* ini mulai ditambahkan pada C++11.

Contoh:

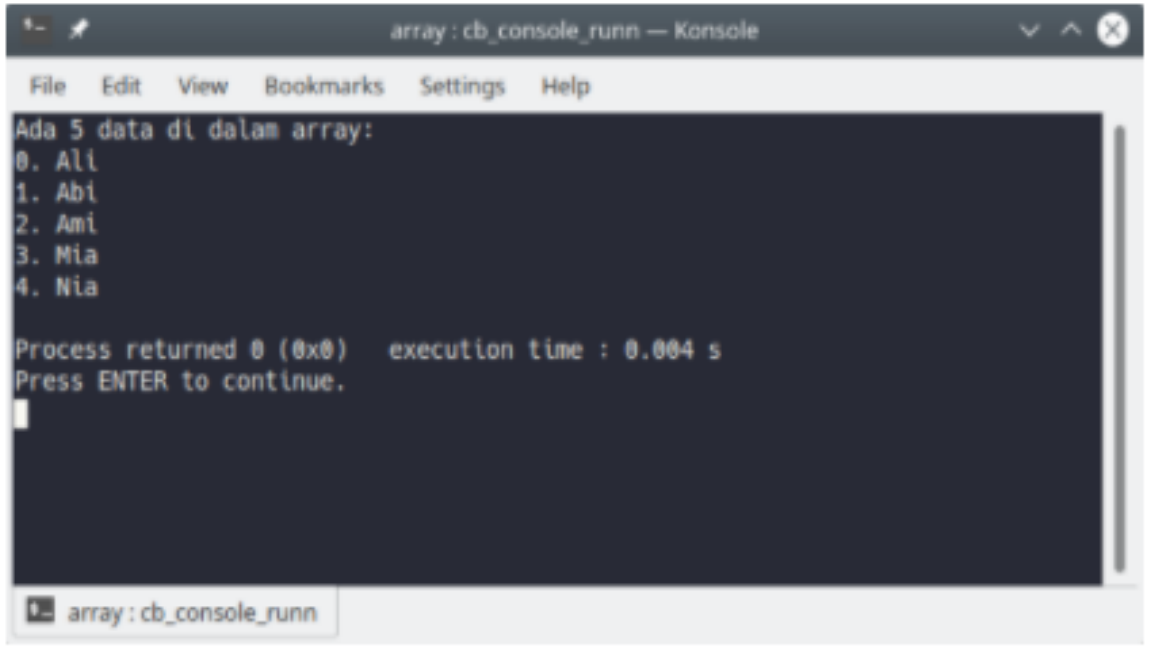
```cpp
// membuat array dengan class
array<string, 5> names = {"Ali", "Abi", "Ami", "Mia", "Nia"};
// mengambil ukuran array
cout << names.size();
```

Tapi sebelum kita dapat menggunakan *Class Template* `array<>`, kita harus mengimpornya dengan `#include`.

Contoh lengkap: `array_class.cpp`

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

int main(){
    array<string, 5> names = {"Ali", "Abi", "Ami", "Mia", "Nia"};

    printf("Ada %d data di dalam array: \n", names.size());

    for(int i = 0; i < names.size(); i++){
        cout << i << ". " << names[i] << endl;
    }

    return 0;
}
```

Hasilnya:



Bisa dibilang, *Class Template* array ini adalah pengembangan dari cara yang sebelumnya.

Array dianggap sebagai sebuah objek..

Objek itu apa?

Latihan: Program Hitung Rata-Rata

Anggap saja kita diminta untuk membuat program untuk menghitung tinggi rata-rata dari sepuluh orang.

Pertama kita pasti akan membutuhkan sebuah array dengan panjang `10` yang berisi kumpulan tinggi badan.

```
int tinggi_badan[10] = {175, 165, 166, 157, 184, 156, 163, 176, 171, 169};
```

Berikutnya kita harus menghitung nilai rata-rata dari sekumpulan nilai tersebut.

Rumus nilai rata-rata:

$$\overline{x} = \frac{x_1 + x_2 + x_3 + \cdots + x_n}{n}$$

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

Mari kita buat programnya...

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

int main(){
    // membuat array dari tinggi badan
    int tinggi_badan[10] = {175, 165, 166, 157, 184, 156, 163, 176, 171, 169

    //  length itu sama dengan n
    int length = sizeof(tinggi_badan) / sizeof(tinggi_badan[0]);
    int sum = 0;

    for(int i = 0; i < length; i++){
        sum += tinggi_badan[i];
    }

    // rata-rata tinggi badan
    float rata_rata = (float)sum / (float)length;

    printf("Rata-rata tinggi badan: %.2f\n", rata_rata);

    return 0;
}
```

Array Multi Dimensi

Array yang kita buat pada contoh-contoh program di atas adalah array satu dimensi.

Array bisa dibuat dua dimensi bahkan lebih.

Array multidimensi biasanya digunakan untuk membuat matriks.

Contoh array dua dimensi:

```
int matriks[3][3] = {
    {1, 3, 5},
    {5, 3, 1},
    {6, 2, 3}
};
```

Array dua dimensi biasanya digunakan untuk membuat matriks.

Lalu bagaimana cara mengambil data dari array dua dimensi?

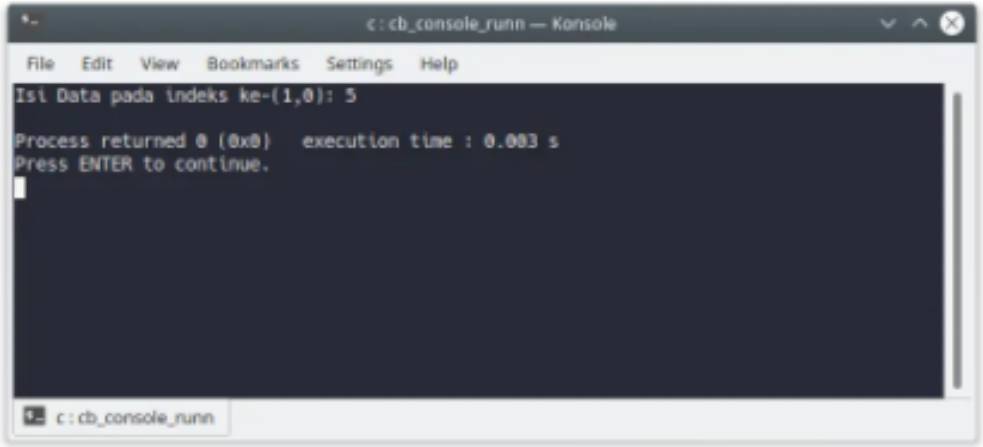
Begini caranya:

```
matriks[1][2];
```

Contoh program:

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

int main(){
    int matriks[3][3] = {
        {1, 3, 5},
        {5, 3, 1},
        {6, 2, 3}
    };

    printf("Isi Data pada indeks ke-(1,0): %d\n", matriks[1][0]);

    return 0;
}
```

Hasilnya:



Thank You

ukrida.ac.id

UKRIDA
Universitas Kristen Krida Wacana

Linked List

CYNTHIA HAYAT S.KOM., M.MSI
KRIDA WACANA CHRISTIAN UNIVERSITY
Faculty of Engineering and Computer Science
Departement of Information System

UKRIDA
Universitas Kristen Krida Wacana

Single Linked List pada C++

Linked List adalah struktur berupa rangkaian elemen saling berkait dimana tiap elemen dihubungkan ke elemen yang lain melalui pointer. Pointer adalah alamat elemen. Penggunaan pointer untuk mengacu elemen berakibat elemen-elemen bersebelahan secara logik walaupun tidak bersebelahan secara fisik di memori.

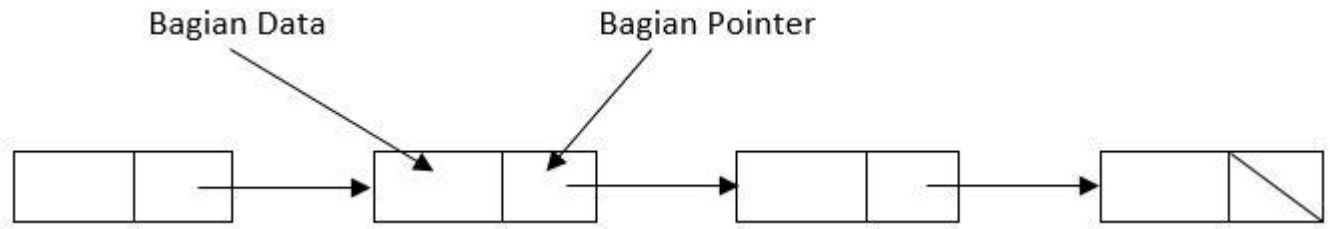
Penyimpanan dan pengolahan data dari sekelompok data yang telah terorganisir dalam sebuah urutan tertentu dapat dilakukan dengan menggunakan array seperti yang telah dibahas sebelumnya. Cara lain untuk menyimpan dan mengolah sekumpulan data seperti di atas juga dapat dilakukan dengan tipe pointer.

Penggunaan pointer sangat mendukung dalam pembentukan struktur data dinamis. Salah satu struktur data dinamis adalah linked list. Berarti Linked List merupakan kumpulan komponen yang saling berkaitan satu dengan yang lain melalui pointer. Masing-masing komponen sering disebut dengan simpul atau node atau verteks. Setiap simpul pada dasarnya dibagi atas dua bagian. Bagian pertama disebut bagian Isi atau Informasi atau Data yaitu bagian yang berisi nilai yang disimpan oleh simpul. Bagian kedua disebut bagian Pointer, yaitu berisi alamat dari simpul berikutnya dan atau sebelumnya.

Linked List dapat disajikan dengan 2 bagian besar yaitu *Singly List* dan *Doubly List*. Baik Singly List maupun Doubly List dapat juga disajikan secara melingkar (*circular*).

1. Singly Linked List

Single Linked List merupakan Linked List yang paling sederhana. Setiap simpul dibagi menjadi dua bagian yaitu bagian Isi dan bagian Pointer. Bagian Isi merupakan bagian yang berisi data yang disimpan oleh simpul, sedangkan bagian Pointer merupakan bagian yang berisi alamat dari simpul berikutnya.



terlihat bahwa simpul pertama dihubungkan ke simpul kedua melalui bagian Pointer simpul pertama. Bagian Pointer simpul kedua dihubungkan ke simpul ketiga. Demikian seterusnya hingga simpul terakhir. Bagian Pointer simpul terakhir tidak dihubungkan ke simpul lain yang disebut sebagai NULL.

Deklarasi Single Linked List pada C++

Dari ilustrasi di atas kita lihat bahwa masing-masing simpul terbagi atas dua bagian, yaitu bagian Isi dan bagian Pointer, maka dengan demikian dapat dibuat dalam struct yang terdiri dari 2 field, seperti berikut ini.

```
typedef struct
struct node
{

   type_data Isi;
   simpul->next;
}
```

3. Operasi pada Singly Linked List

Ada sejumlah operasi yang dapat dilakukan pada sebuah Singly Linked List, seperti menambah simpul, menghapus simpul, membaca isi Linked List, atau pencarian informasi pada suatu Linked List. Dalam buku ini Linked List ditunjuk oleh Pointer L.

3.1. Menambah Simpul

Operasi yang digunakan untuk menyisipkan simpul di posisi tertentu. Penyisipan simpul dapat dilakukan di posisi depan, penyisipan simpul di belakang, penyisipan simpul di antara dua simpul (simpul tengah).

a. Menambah Simpul Depan

Operasi ini akan menyisipkan simpul baru selalu berada pada posisi pertama atau depan dari linked list. Langkah-langkah penyisipan simpul depan dapat dilakukan dengan:

•Ciptakan simpul Baru yang akan disisipkan.

•Jika Linked List belum ada maka simpul Baru menjadi Linked List (L = Baru).

•Jika Linked List sudah ada maka penyisipan dilakukan dengan cara:

- Pointer Next simpul Baru menunjuk L (Baru->Next = L).
- Pointer L dipindahkan ke Baru (L = Baru).

a. Menambah Simpul Depan

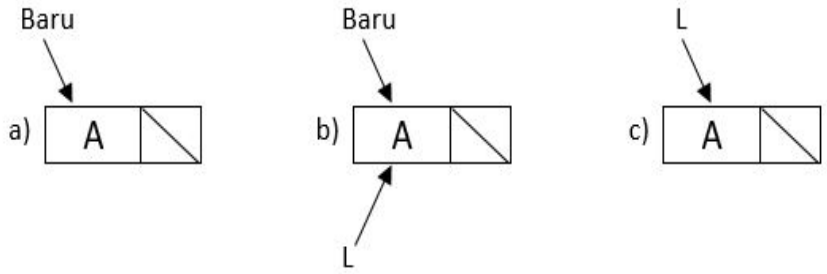
Operasi ini akan menyisipkan simpul baru selalu berada pada posisi pertama atau depan dari linked list. Langkah-langkah penyisipan simpul depan dapat dilakukan dengan:

•Ciptakan simpul Baru yang akan disisipkan.

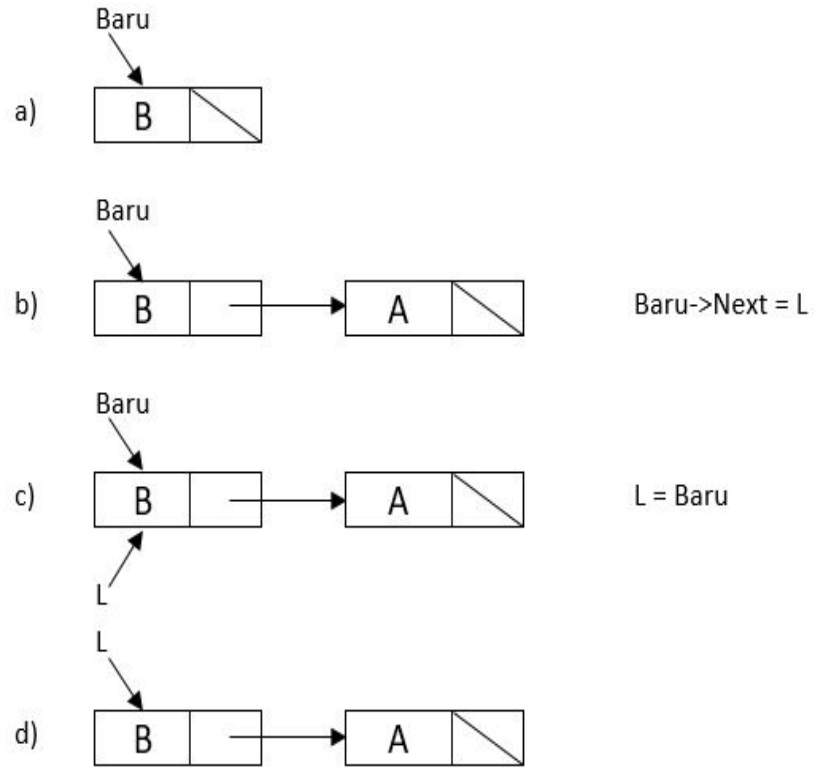
•Jika Linked List belum ada maka simpul Baru menjadi Linked List (L = Baru).

•Jika Linked List sudah ada maka penyisipan dilakukan dengan cara:

- Pointer Next simpul Baru menunjuk L (Baru->Next = L).
- Pointer L dipindahkan ke Baru (L = Baru).



Gambar 2. Penyisipan Simpul dengan Linked List Belum Ada



Gambar 3. Penyisipan Simpul Depan

Fungsi yang digunakan untuk menyisipkan simpul Depan dengan mengikuti langkah-langkah dan gambar 7.3 di atas dapat dilihat berikut ini.

```
void Sisip_Depan(simpul &L, char elemen)
{
     simpul baru;
     baru = (simpul) malloc(sizeof(simpul));
     baru->Isi = elemen;
     baru->Next = NULL;
     if(L == NULL)
          L=baru;
     else
     {
          baru->Next;
          L=baru;
     }
}
```

b. Menambah Simpul Belakang

Operasi ini akan menyisipkan simpul Baru selalu berada pada posisi terakhir atau belakang dari Linked List.

Langkah-langkah penyisipan simpul Belakang dapat dilakukan dengan:

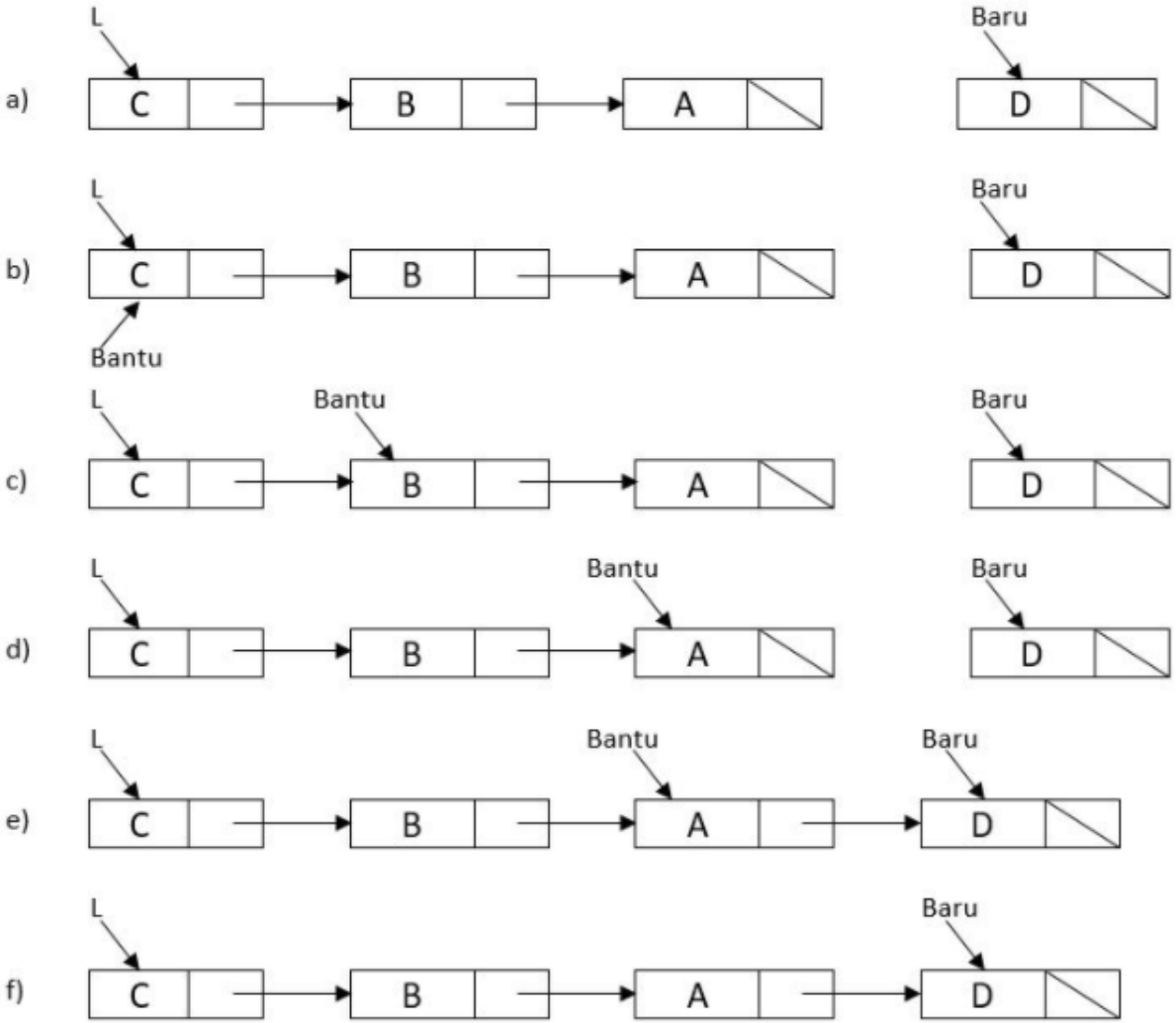
•Ciptakan simpul Baru yang akan disisipkan.

•Jika Linked List belum ada maka simpul baru menjadi Linked List (L = Baru).

•Jika Linked List sudah ada maka penyisipan dilakukan dengan cara:

- Buat suatu Pointer yang dapat digerakkan, misalkan Pointer Bantu yang menunjuk simpul pertama dari Linked List (*Bantu = L*). Hal ini dilakukan karena Pointer L tidak boleh digerakkan dari simpul Depan. Karena jika Pointer L digerakkan maka informasi dari simpul yang ditinggalkan oleh Pointer L tidak dapat lagi diakses.
- Gerakkan Pointer Bantu hingga ke simpul paling Belakang dari Linked List (while(Bantu->Next != NULL) Bantu=Bantu->Next;).
- Sambungkan Linked List dengan simpul Baru (*Bantu->Next = Baru*).

Skema penyisipan simpul Belakang dapat dilihat pada gambar 7.2. (Linked List belum ada) dan gambar 7.4.



c. Menambah Simpul Tengah

Operasi ini akan menyisipkan simpul Baru selalu berada di antara dua simpul dari Linked List. Simpul dapat diletakkan sebelum simpul tertentu atau setelah simpul tertentu. Penyisipan simpul Tengah hanya dapat dilakukan jika Linked List tidak kosong.

3.2. Menghapus Simpul

Maksudnya adalah operasi menghapus suatu simpul dari suatu Linked List. Dalam melakukan penghapusan simpul, ada yang perlu diperhatikan, bahwa Linked List tidak boleh kosong dan Linked List tidak boleh terputus. Sama halnya dengan penyisipan, penghapusan simpul juga dapat dilakukan terhadap simpul depan, simpul belakang, dan simpul tengah.

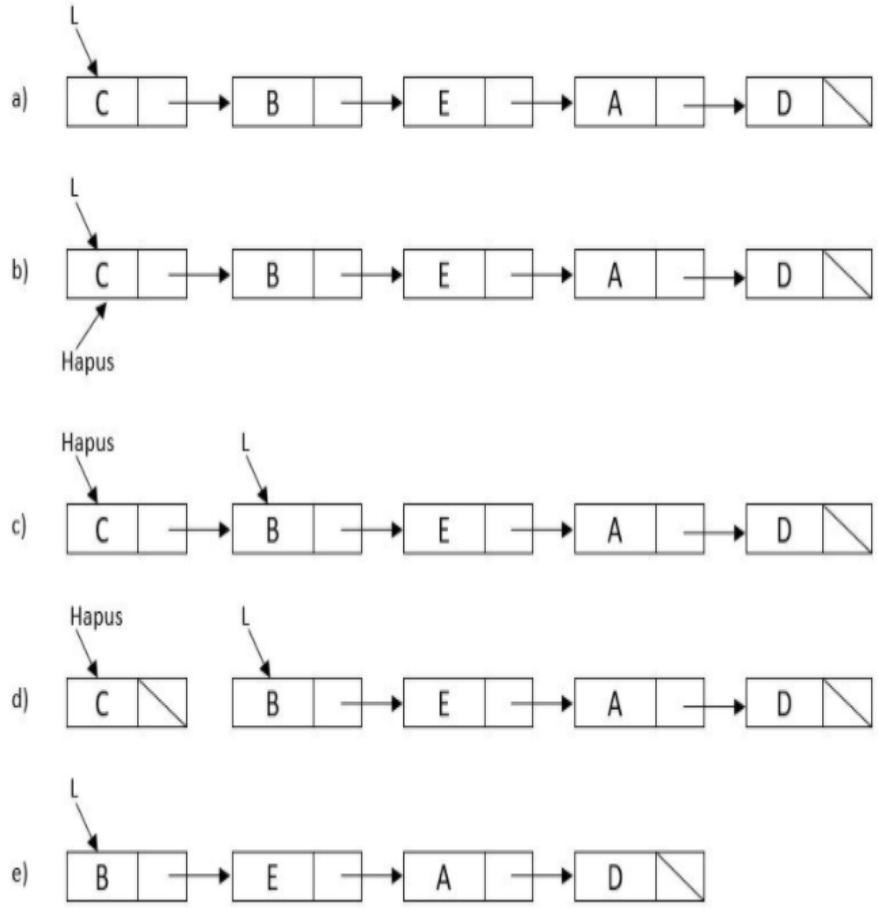
a. Menghapus Simpul Depan

Selalu menghapus simpul dengan dari Linked List. Linked List tidak boleh kosong.

Langkah-langkah penghapusan simpul dengan dapat dilakukan dengan cara sebagai berikut:

- Buat suatu pointer misalnya pointer hapus untuk menunjuk simpul yang akan dihapus dan L untuk menunjuk Linked List.
- Simpul pertama ditunjuk oleh pointer hapus (Hapus = L).
- Pointer L digerakkan satu simpul berikutnya (L = L->Next).
- Putuskan simpul pertama dari L (Hapus->Next = NULL).

Skema penghapusan simpul depan dapat dilihat pada gambar 7.7.




```
1.  void Hapus_Depan(simpul &L)
2.  {
3.    simpul Hapus;
4.    if(L == NULL)
5.      cout<<"Linked List Kosong .....";
6.    else
7.    {
8.      Hapus = L;
9.      L = L->N;
10.     Hapus->Next = NULL;
11.     free(Hapus);
12.   }
13. }
```

b. Menghapus Simpul Belakang

Selalu menghapus simpul terakhir atau belakang dari Linked List. Linked List tidak boleh kosong. Perlu diingat bahwa pointer L tidak boleh digerakkan, maka buat suatu pointer misalnya pointer bantu yang dapat digerakkan dalam Linked List. Dengan menggunakan pointer bantu maka memungkinkan tidak adanya simpul yang hilang atau terputus. Di samping itu juga buatlah pointer hapus yang digunakan untuk menunjuk simpul yang akan dihapus.

Langkah-langkah penghapusan simpul belakang dapat dilakukan sebagai berikut:

•Letakkan pointer bantu pada simpul pertama (Bantu = L).

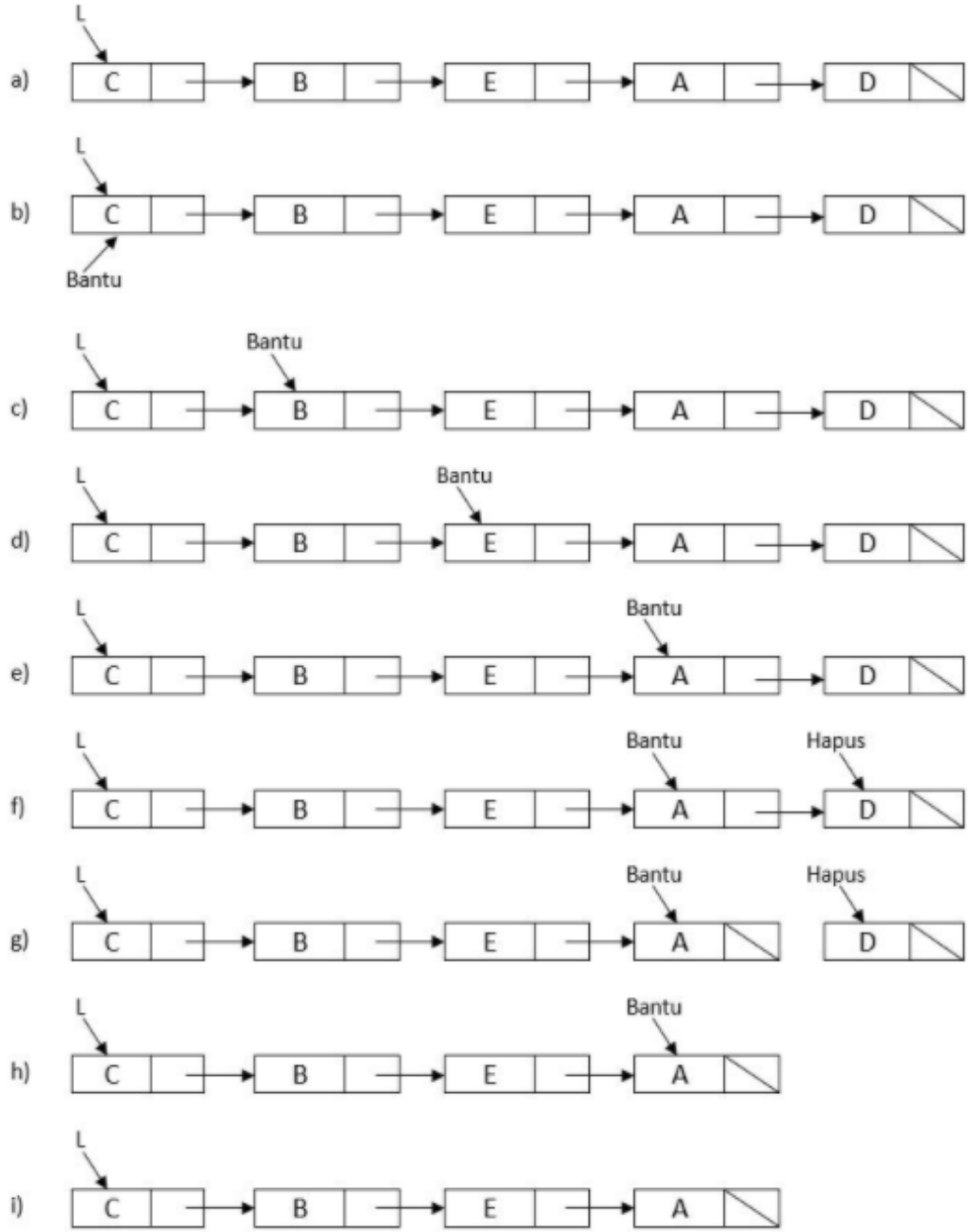
•Gerakkan pointer bantu hingga pada satu simpul sebelum siimpul terakhir (while(Bantu->Next->Next != NULL) Bantu=Bantu->Next;).

•Simpul terakhir ditunjuk oleh pointer **hapus** (Hapus = Bantu->Next).

•Putuskan simpul terakhir dari L (Bantu->Next = NULL).

Fungsi penghapusan simpul belakang dengan mengikuti langkah-langkah dan gambar 7.8. di atas dapat dilihat berikut ini.

```
1.  void Hapus_Belakang(simpul &L)
2.  {
3.    simpul bantu, hapus;
4.    if(L==NULL)
5.      cout<<"Linked List Kosong .....";
6.    else
7.    {
8.      bantu = L;
9.      while(bantu->Next->Next != NULL)
10.       bantu=bantu->Next;
11.     hapus = bantu->Next;
12.     bantu->Next = NULL;
13.     free(hapus);
14.   }
15. }
```



c. Menghapus Simpul Tengah

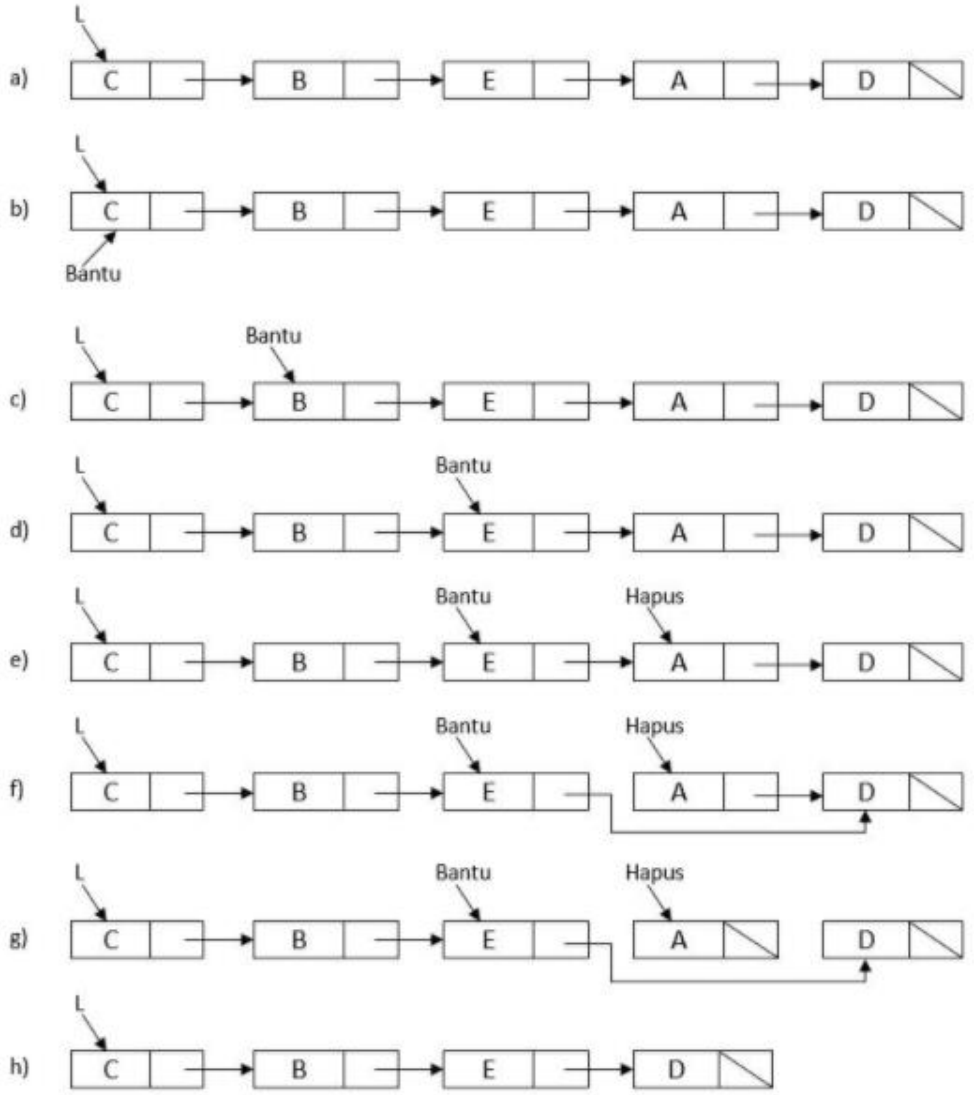
Menghapus simpul tertentu yang ada di posisi tengah dari Linked List. Linked List tidak boleh kosong. Perlu diingat bahwa pointer L tidak boleh digerakkan, maka buat suatu pointer misalnya pointer bantu yang dapat digerakkan dalam Linked List. Dengan menggunakan pointer bantu maka memungkinkan tidak adanya simpul yang hilang atau terputus. Di samping itu juga buatlah pointer hapus yang digunakan untuk menunjuk simpul yang akan dihapus.

Langkah-langkah penghapusan simpul belakang dapat dilakukan sebgai berikut:

- Letakkan pointer bantu pada simpul pertama (Bantu=L).
- Gerakkan pointer bantu hingga satu simpul sebelum simpul yang akan dihapus (while(Bantu->Next->Isi != “info”) Bantu=Bantu->Next;)
- Letakkan pointer hapus pada simpul yang akan dihapus (Hapus = Bantu->Next).
- Pointer bantu menunjuk simpul setelah simpul yang ditunjuk oleh hapus (Bantu->Next=Hapus->Next atau Bantu->Next = Bantu->Next->Next).
- Putuskan simpul yang ditunjuk hapus dari Linked List (Hapus->Next = NULL).

Fungsi untuk menghapus simpul tertentu di tengah dengan mengikuti langkah-langkah dan gambar 7.9 di atas dapat dilihat berikut ini.

```
1.  void Hapus_Tengah(simpul &L, char elemen)
2.  {
3.    simpul bantu, hapus;
4.    if(L==NULL)
5.      cout<<"Linked List Kosong ......";
6.    else
7.    {
8.      bantu = L;
9.      while(bantu->Next->Isi != elemen)
10.       bantu=bantu->Next;
11.     hapus = bantu->Next;
12.     bantu->Next = bantu->Next->Next;
13.     hapus->Next = NULL;
14.     free(hapus);
15.   }
16. }
```



Gambar 9. Operasi Penghapusan Simpul Tertentu di Tengah

3.3. Mencetak Isi Simpul

Nilai masing-masing simpul dapat dicetak mulai dari isi simpul pertama atau simpul depan hingga simpul belakang dengan fungsi berikut ini.

```
1.  void Cetak(simpul L)
2.  {
3.    simpul bantu;
4.    if(L==NULL)
5.      cout<<"Linked List Kosong ......"<<endl;
6.    else
7.    {
8.      bantu=L;
9.      cout<<"Isi Linked List : ";
10.     while (bantu->Next != NULL)
11.     {
12.       cout<<bantu->Isi<<"-->";
13.       bantu=bantu->Next;
14.     }
15.     cout<<bantu->Isi;
16.   }
17. }
```

Berikut ini merupakan program lengkap untuk operasi Singly Linked List.

```
/*  ================================
    ================================
    ==PROGRAM SINGLE LINKED LIST==
    =====BY : LAMHOT SITORUS======
    ================================*/

#include<iostream.h>
#include<conio.h>
#include<stdlib.h>
typedef struct node *simpul;
struct node
{
  char Isi;
  simpul Next;
};
//======================
//==Prototype Function==
//======================
void Sisip_Depan(simpul &L, char elemen);
void Sisip_Belakang(simpul &L, char elemen);
void Sisip_Tengah1(simpul &L, char elemen1, char elemen2);
void Sisip_Tengah2(simpul &L, char elemen1, char elemen2);
void Hapus_Depan(simpul &L);
void Hapus_Belakang(simpul &L);
void Hapus_Tengah(simpul &L, char elemen);
void Cetak(simpul L);

//================
//==Function Main==
//================
main()
{
  char huruf, huruf2;
  simpul L = NULL;  //Pastikan bahwa L kosong
  cout<<"==OPERASI PADA SINGLE LINKED LIST=="<<endl<<endl;
  //===============
  //==Sisip Depan==
  //===============
  cout<<"Penyisipan Simpul Di Depan"<<endl<<endl;
  cout<<"Masukkan Huruf :"; cin>>huruf;
  Sisip_Depan(L, huruf);
  cout<<"Masukkan Huruf :"; cin>>huruf;
  Sisip_Depan(L, huruf);
  cout<<"Masukkan Huruf :"; cin>>huruf;
  Sisip_Depan(L, huruf);
  cout<<"Masukkan Huruf :"; cin>>huruf;
  Sisip_Depan(L, huruf);
  Cetak(L);
  //======================
  //====Sisip Belakang====
  //======================
  //==Hapus Simpul Depan==
  //======================
  cout<<endl<<endl<<"Setelah Hapus Simpul Depan "<<;
  Hapus_Depan(L);
  Cetak(L);
  //==========================
  //==Hapus Simpul Belakang==
  //==========================
  cout<<endl<<endl<<"Setelah Hapus Simpul Belakang "<<endl;
  Hapus_Belakang(L);
  Cetak(L);
  //======================
  //==Hapus Simpul TENGAH==
  //======================
  cout<<"\n\nMasukkan Huruf Tengah Yang Akan Dihapus :";
  cin>>huruf;
  Hapus_Tengah(L,huruf);
  Cetak(L);
  getch();
}
```

```
72.
73.   //************************************
74.   //**FUNCTION SISIP SIMPUL DI DEPAN**
75.   //************************************
76.   void Sisip_Depan(simpul &L, char elemen)
77.   {
78.     simpul baru;  // = new simpul;
79.     baru = (simpul) malloc(sizeof(simpul));
80.     baru->Isi = elemen;
81.     baru->Next = NULL;
82.     if(L == NULL)
83.       L=baru;
84.     else
85.     {
86.       baru->Next = L;
87.       L=baru;
88.     }
89.   }
90.
91.   //*****************************************************
92.   //**FUNCTION SISIP SIMPUL SETELAH SIMPUL TERTENTU**
93.   //*****************************************************
94.   void Sisip_Tengah1(simpul &L, char elemen1, char elemen2)
95.   {
96.     simpul bantu, baru;
97.     baru = (simpul) malloc(sizeof(simpul));
98.     baru->Isi = elemen1;
99.     baru->Next = NULL;
100.    if(L == NULL)
101.      cout<<"List Kosong............."<<endl;
102.    else
103.    {
104.      bantu = L;
105.      while(bantu->Isi != elemen2) bantu=bantu->Next;
106.      baru->Next = bantu->Next;
107.      bantu->Next = baru;
108.    }
109.  }
```

```
110.
111.  //*****************************************************
112.  //**FUNCTION SISIP SIMPUL SEBELUM SIMPUL TERTENTU**
113.  //*****************************************************
114.  void Sisip_Tengah2(simpul &L, char elemen1, char elemen2)
115.  {
116.    simpul bantu, baru;
117.    baru = (simpul) malloc(sizeof(simpul));
118.    baru->Isi = elemen1;
119.    baru->Next = NULL;
120.    if(L == NULL)
121.      cout<<"List Kosong ..........."<<endl;
122.    else
123.    {
124.      bantu = L;
125.      while(bantu->Next->Isi != elemen2) bantu=bantu->Next;
126.      baru->Next = bantu->Next;
127.      bantu->Next = baru;
128.    }
129.  }
130.
131.  //************************************
132.  //**FUNCTION SISIP SIMPUL DI BELAKANG**
133.  //************************************
134.  void Sisip_Belakang(simpul &L, char elemen)
135.  {
136.    simpul bantu, baru;
137.    baru = (simpul) malloc(sizeof(simpul));
138.    baru->Isi = elemen;
139.    baru->Next = NULL;
140.    if(L == NULL)
141.      L=baru;
142.    else
143.    {
144.      bantu=L;
145.      while(bantu->Next != NULL)
146.        bantu=bantu->Next;
147.      bantu->Next=baru;
148.    }
149.  }
```

```
149.  }
150.  //***************************************
151.  //**FUNCTION MENCETAK ISI LINKED LIST**
152.  //***************************************
153.  void Cetak(simpul L)
154.  {
155.    simpul bantu;
156.    if(L=NULL)
157.      cout<<"Linked List Kosong ......"<<endl;
158.    else
159.    {
160.      bantu=L;
161.      cout<<"Isi Linked List : ";
162.      while (bantu->Next != NULL)
163.      {
164.        cout<<bantu->Isi<<"-->";
165.        bantu=bantu->Next;
166.      }
167.      cout<<bantu->Isi;
168.    }
169.  }
170.
171.  //*********************************
172.  //**FUNCTION HAPUS SIMPUL DEPAN**
173.  //*********************************
174.  void Hapus_Depan(simpul &L)
175.  {
176.    simpul Hapus;
177.    if(L==NULL)
178.      cout<<"Linked List Kosong ............";
179.    else
180.    {
181.      Hapus = L;
182.      L = L->Next;
183.      Hapus->Next = NULL;
184.      free(Hapus);
185.    }
186.  }
```

```
187.
188.  //************************************
189.  //**FUNCTION HAPUS SIMPUL BELAKANG**
190.  //************************************
191.  void Hapus_Belakang(simpul &L)
192.  {
193.    simpul bantu, hapus;
194.    if(L==NULL)
195.      cout<<"Linked List Kosong ............";
196.    else
197.    {
198.      bantu = L;
199.      while(bantu->Next->Next != NULL) bantu=bantu->Next;
200.      hapus = bantu->Next;
201.      bantu->Next = NULL;
202.      free(hapus);
203.    }
204.  }
205.
206.  //************************************
207.  //**FUNCTION HAPUS SIMPUL DI TENGAH**
208.  //************************************
209.  void Hapus_Tengah(simpul &L, char elemen)
210.  {
211.    simpul bantu, hapus;
212.    if(L==NULL)
213.      cout<<"Linked List Kosong ............";
214.    else
215.    {
216.      bantu = L;
217.      while(bantu->Next->Isi != elemen) bantu=bantu->Next;
218.      hapus = bantu->Next;
219.      bantu->Next = bantu->Next->Next;
220.      hapus->Next = NULL;
221.      free(hapus);
222.    }
223.  } //====================eof====================
```

Thank You

ukrida.ac.id

UKRIDA
Universitas Kristen Krida Wacana

Belajar C++: Stack dan Queue

CYNTHIA HAYAT S.KOM., M.MSI

KRIDA WACANA CHRISTIAN UNIVERSITY
Faculty of Engineering and Computer Science
Departement of Information System

UKRIDA
Universitas Kristen Krida Wacana

- **Graf** adalah **diagram** yang digunakan untuk menggambarkan berbagai struktur yang ada.
- **Contoh :**

  Struktur Organisasi, Peta, Diagram Rangkaian Listrik.
- **Tujuan :**

  Sebagai visualisasi objek-objeknya agar mudah dimengerti.

Dasar-Dasar Graf (1)

- Suatu Graf terdiri dari 2 himp. yang berhingga, yaitu **himp. titik-titik** tak kosong (simbol **V(G)**) dan **himp. garis-garis** (simbol **E(G)**).
- Setiap garis berhubungan dg satu atau dua titik. Titik-titik tsb disebut **Titik Ujung.**
- Garis yang berhubungan dg satu titik disebut **Loop**.
- Dua garis yang menghubungkan titik yang sama disebut **Garis Paralel.**
- Dua titik dikatakan **berhubungan** bila ada garis yg menghubungkan keduanya.
- Titik yang tidak punya garis yang berhubungan dengannya disebut **Titik Terasing**.

Dasar-Dasar Graf (2)

- Dua garis yang menghubungkan titik yang sama disebut **Garis Paralel.**
- Dua titik dikatakan **berhubungan** bila ada garis yg menghubungkan keduanya.
- Titik yang tidak punya garis yang berhubungan dengannya disebut **Titik Terasing**.
- **Graf Kosong** adalah graf yang tidak punya titik dan garis.
- **Graf Berarah** adalah graf yang semua garisnya memiliki arah (*Directed Graph / Digraph*).
- **Graf Tak Berarah** adalah graf yang semua garisnya tidak memiliki arah

Thank You

ukrida.ac.id

UKRIDA
Universitas Kristen Krida Wacana

Graph dan Tree

CYNTHIA HAYAT S.KOM., M.MSI

KRIDA WACANA CHRISTIAN UNIVERSITY
Faculty of Engineering and Computer Science
Departement of Information System

UKRIDA
Universitas Kristen Krida Wacana

- **Graf** adalah **diagram** yang digunakan untuk menggambarkan berbagai struktur yang ada.
- **Contoh :**

  Struktur Organisasi, Peta, Diagram Rangkaian Listrik.
- **Tujuan :**

  Sebagai visualisasi objek-objeknya agar mudah dimengerti.

Dasar-Dasar Graf (1)

- Suatu Graf terdiri dari 2 himp. yang berhingga, yaitu **himp. titik-titik** tak kosong (simbol **V(G)**) dan **himp. garis-garis** (simbol **E(G)**).
- Setiap garis berhubungan dg satu atau dua titik. Titik-titik tsb disebut **Titik Ujung.**
- Garis yang berhubungan dg satu titik disebut **Loop**.
- Dua garis yang menghubungkan titik yang sama disebut **Garis Paralel.**
- Dua titik dikatakan **berhubungan** bila ada garis yg menghubungkan keduanya.
- Titik yang tidak punya garis yang berhubungan dengannya disebut **Titik Terasing**.

Dasar-Dasar Graf (2)

- Dua garis yang menghubungkan titik yang sama disebut **Garis Paralel.**
- Dua titik dikatakan **berhubungan** bila ada garis yg menghubungkan keduanya.
- Titik yang tidak punya garis yang berhubungan dengannya disebut **Titik Terasing**.
- **Graf Kosong** adalah graf yang tidak punya titik dan garis.
- **Graf Berarah** adalah graf yang semua garisnya memiliki arah (*Directed Graph / Digraph*).
- **Graf Tak Berarah** adalah graf yang semua garisnya tidak memiliki arah

Contoh 1.

- Ada 7 kota (A,…,G) yang diantaranya dihubungkan langsung dg jalan darat. Hubungan antar kota didefinisikan sebagai berikut :

  A terhubung dg B dan D

  B terhubung dg D

  C terhubung dg B

  E terhubung dg F

  Buatlah graf yang menunjukkan keadaan transportasi di 7 kota tersebut !

- Gambarlah graf dengan titik-titik dan garis berikut :

  V(G) = { v1,v2,v3,v4 }

  E(G) = { e1,e2,e3,e4,e5 }

  Titik-titik ujung garis adalah :

Garis	Titik Ujung
e1	{v1,v3}
e2	{v2,v4}
e3	{v1}
e4	{v2,v4}
e5	{v3}

Graf Tak Berarah

- **Graf Sederhana** adalah graf yang tidak memiliki Loop ataupun Garis Paralel.

Contoh 3.

- Gambarkan semua graf sederhana yang dapat dibentuk dari 4 titik {a,b,c,d} dan 2 garis !

Derajat

- Misal titik v adalah suatu titik dalam graf G. **Derajat titik v** (simbol **d(v)**) adalah *jumlah garis yang berhubungan dengan titik v.*
- Derajat titik yang berhubungan dengan sebuah loop adalah **2**.

- **Derajat total** suatu graf G adalah **jumlah derajat semua titik dalam G.**
- **Derajat total** suatu graf selalu **genap.**

- Dalam sembarang graf **jumlah titik yang berderajat ganjil selalu genap.**

Path dan Sirkuit (1)

Misalkan G adalah suatu graf, v0 danvn adalah 2

titik di dalam G.

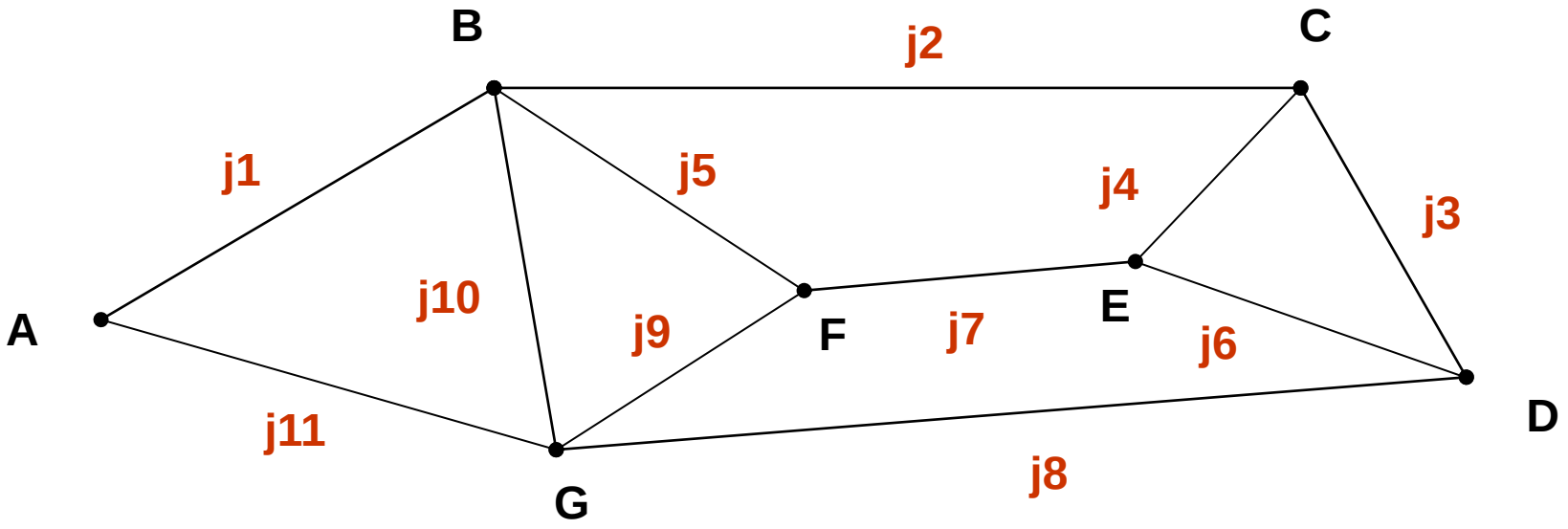
- ***Walk*** dari titik v0 ke titik vn adalah barisan titik-titik berhubungan dan garis secara berselang-seling diawali dari titik v0 dan diakhiri pada titik vn.
- ***Path*** dari titik v0 ke titik vn adalah **walk** dari titik v0 ke titik vn yang **semua garisnya berbeda.**
- **Panjang** walk atau path **=** jumlah garis yang dilalui
- ***Path sederhana*** dari titik v0 ke titik vn adalah **path** dari titik v0 ke titik vn yang **semua titiknya berbeda.**
- ***Sirkuit*** adalah **path** yang dimulai dan diakhiri pada titik yang sama.
- ***Sirkuit sederhana*** adalah **sirkuit** semua titiknya berbeda kecuali untuk titik awal dan titik akhir.

Graf Terhubung dan Tidak Terhubung

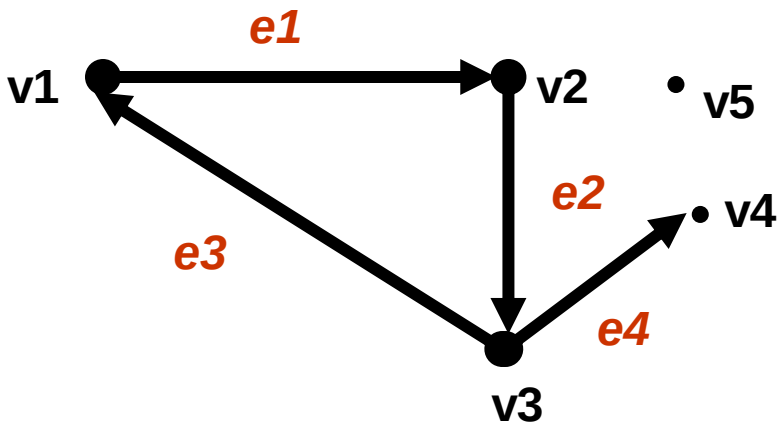
Misalkan G adalah suatu graf

- 2 titik dalam G ,v1 dg v2 **terhubung** bila ada **walk** dari v1 ke v2.
- Graf G dikatakan
  - ***Terhubung*** ⇔ **setiap 2 titik** dalam G **terhubung**.
  - ***Tidak terhubung*** ⇔ **ada 2 titik** dalam G yang **tidak terhubung**.
  - Suatu **graf terhubung G** memiliki **Sirkuit Hamilton** bila ada sirkuit yang mengunjungi setiap titiknya **tepat satu kali** (kecuali titik awal dan titik akhir).

- Gambar di bawah menyatakan peta kota A..G dan jalan-jalan yang menghubungkan kota-kota tsb. Seorang salesman akan mengunjungi tiap kota masing-masing 1 kali dari kota A kembali lagi ke kota A. Carilah rute perjalanan yang harus dilalui salesman tsb !



- Contoh graf G berikut :



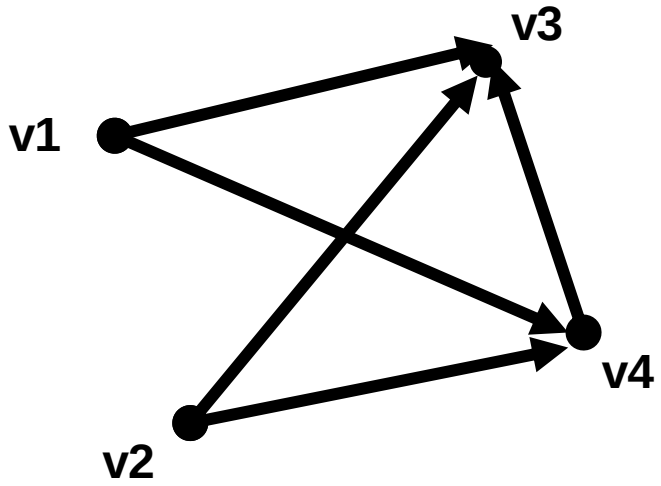
- Titik v1 adalah **titik awal** e1, titik v2 adalah **titik akhir** e1. Arah garis dari v1 ke v2.

Path Berarah dan Sirkuit Berarah

- Dalam graf berarah, perjalanan harus mengikuti arah garis.

- Suatu graf yang tidak memuat sirkuit berarah disebut **ASIKLIK.**

  Contoh :



Pohon (Tree)

- Struktur Pohon adalah salah satu kasus dalam graf.
- Penerapannya pada Teori Struktur Data.
- Graf G disebut **Pohon** ⇔ G merupakan *graf sederhana* yang *tidak memuat sirkuit dan terhubung*.
- **Daun** adalah titik di dalam Pohon yang berderajat 1.
- Titik dalam Pohon yang berderajat > 1 disebut **Titik Cabang.**

Teorema

Suatu pohon dengan n titik memiliki

(n-1) garis

Pohon Rentang

- **Pohon Rentang** dari graf terhubung G adalah subgraf G yang merupakan *pohon* dan *memuat semua titik dalan G.*

Graf Berlabel

- ***Graf Berlabel*** : graf tanpa garis paralel yang setiap garisnya berhubungan dengan bilangan riil positif yang menyatakan **bobot** garis tersebut.
- Simbol : w(e).
- **Total Bobot :** jumlah bobot semua garis dalam graf.
- **Bobot** suatu garis dapat mewakili “jarak”, “biaya”, “panjang”, “kapasitas”, dll.

Pohon Rentang Minimum

- Masalah : mencari pohon rentang dengan total bobot seminimal mungkin.

- Metode : ***Algoritma Kruskal***

Algoritma Kruskal (1)

- Mula-mula urutkan semua garis dalam graf dari yang bobotnya terkecil sampai terbesar.

- **G** : graf mula-mula dg **n** titik,
- **T** : Pohon Rentang Minimum,
- **E** : himpunan semua garis dlm G

Algoritma :

1. Isi T dengan semua titik dalam G tanpa garis.
2. m = 0
3. Selama m < (n-1) lakukan :
   a. Pilih garis e dalam E dg bobot terkecil. Jika ada beberapa garis, pilih salah satu.
   b. Hapus garis e dari E.
   c. Jika garis e ditambahkan ke T tidak menghasilkan sirkuit, maka
      I. Tambahkan e ke T.
      II. m = m+1 (Nilai m dinaikkan satu).

Lintasan Terpendek

- Mencari path dengan total bobot paling minimal dari sebuah graf berlabel.
- Metode : Algoritma Djikstra

Algoritma Djikstra

V = {v1, v2, …, vn} → titik awal : v1, titik akhir : vn

L(j) = jumlah bobot lintasan terpendek dari v1 ke vj

w(i,j) = bobot garis dari titik v1 ke titik vj

T = himp. titik yg sudah terpilih dlm alur lintasan terpendek

ALGORITMA

1. T = { }

   L(v1) = 0

   L(v2) = L(v3) = … = L(vn) = ~

Algoritma Djikstra

2. Selama vn $\notin$ T lakukan :

   a. Pilih titik vk $\in$ V – T dengan L(vk) terkecil

      T = T $\cup$ { vk }

   b. Untuk setiap vj $\in$ V – T hitung :

      L(vj) = min[L(vj) , L(vk) + w(vk,vj)]

3. Telusuri alur path minimum mulai dari titik akhir (vn) sampai titik awal (v1)

Thank You

ukrida.ac.id

UKRIDA
Universitas Kristen Krida Wacana

Searching Data

CYNTHIA HAYAT S.KOM., M.MSI

KRIDA WACANA CHRISTIAN UNIVERSITY
Faculty of Engineering and Computer Science
Departement of Information System

UKRIDA
Universitas Kristen Krida Wacana

Pengertian Searching pada C++. Pencarian merupakan proses yang mendasar di dalam pemrograman. Pencarian (*Searching*) merupakan tindakan untuk mendapatkan suatu data dalam kumpulan data berdasarkan satu kunci (*key*) atau acuan data. Dalam kehidupan sehari-hari, seringkali kita berurusan dengan pencarian; misalnya untuk menemukan nomor telepon seseorang pada buku telepon atau mencari istilah dalam kamus, dan masih banyak lagi. Pada aplikasi komputer, pencarian kerapkali dilakukan. Misalnya untuk proses penghapusan data/*record* atau mengubah data/*record* tertentu di dalam suatu tabel atau file, langkah pertama yang harus dilakukan adalah mencari apakah data tersebut terdapat di dalam tabel/file atau tidak. Jika ada maka dapat dihapus atau diubah.

Kegunaan beberapa struktur data dalam hubungannya untuk menyimpan data sehingga memudahkan proses pencarian kembali tergantung pada:

- Media penyimpanan data (memori, disk, tape).
- Karakteristik jenis data yang disimpan (data numerik/character/string).
- Jumlah data yang akan disimpan dan keperluan untuk akses data secepat-cepatnya.

Pencarian dapat dilakukan berdasarkan tempat penyimpanan, yang dibagi atas dua, yaitu pencarian internal dan pencarian eksternal. Pencarian internal yaitu pencarian yang dilakukan terhadap data yang berada dalam memori komputer. Pencarian eksternal yaitu pencarian yang dilakukan terhadap data yang berada dalam memori eksternal. Di dalam pembahasan ini hanya membahas tentang pencarian internal.

Pada umumnya dikenal tiga metode searching, antara lain: Sequensial Search, Binary Search, dan Interpolation Search.

1. Sequential Search

Sequential Search (pencarian beruntun) adalah metode pencarian yang paling mudah. Pencarian beruntun adalah proses membandingkan setiap elemen array satu per satu secara beruntun yang dimulai dari elemen pertama hingga elemen yang dicari ditemukan atau hingga elemen terakhir dari array. Pencarian beruntun dapat dilakukan terhadap elemen array yang belum terurut atau terhadap elemen array yang terurut. Perbedaan dari keduanya terletak pada efisiensi operasi pembandingan yang dilakukan.

Dengan kata lain sequential search akan mencari data dengan cara membandingkannya satu-persatu dengan data yang ada. Prosesnya tentu saja akan singkat jika data yang diolah sedikit, dan akan lama jika data yang diolah banyak. Metode ini disarankan untuk digunakan pada data yang sedikit saja.

Contoh: Diberikan suatu array nilai dengan banyak elemen 8 seperti berikut:

10	15	9	3	25	65	15	30
1	2	3	4	5	6	7	8

Misalkan nilai yang dicari adalah: X = 15

- Kalau yang diharapkan hanya menyatakan ada atau tidak ada maka pemeriksaan dilakukan terhadap 10 dan 15 maka tampil pesan “15 ditemukan”.

Berikut program Lat_Searching_01a pencarian untuk kasus pertama di atas:

```
/* Program Sequential Search
  Nama File : Lat_Searching_01a */

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

main()
{
  int Nilai[20];
  int i, N, Bilangan;
  bool ketemu;

  cout<<"Masukkan Banyaknya Bilangan = ";
  cin>>N;

  //Membaca elemen Array
  for(i=0; i<N; i++)
  {
    cout<<"Masukkan elemen ke-"<<i<<" = ";
    cin>>Nilai[i];
  }

  //Mencetak Elemen Array
  cout<<"\n\Deretan Bilangan = ";
  for(i=0; i<N; i++)
  cout<<Nilai[i]<<" ";

  cout<<"\n\nMasukkan Bilangan yang akan Dicari = ";
  cin>>Bilangan;

  //Melakukan Pencarian
  i=0;
  do
  {
    if(Nilai[i]==Bilangan)
    ketemu = true;
    i++;
  }
  while(!(ketemu));

  if(ketemu)
  cout<<"Bilangan "<<Bilangan<<" ditemukan";
  else
  cout<<"Bilangan "<<Bilangan<<" tidak ditemukan";
  getch();
}
```

2. Binary Search

Binary search adalah metode pencarian suatu data atau elemen di dalam suatu array dengan kondisi data dalam keadaan terurut. Proses pencarian binary search hanya dapat dilakukan pada kumpulan data yang sudah diurutkan terlebih dahulu (menaik atau menurun). Prinsip dari binary search terhadap N elemen dapat dijelaskan seperti berikut:

1. Tentukan posisi awal = 0 dan posisi akhir = *N*-1.
2. Hitung posisi tengah = (posisi awal + posisi akhir)/2.
3. Bandingkan data yang dicari dengan elemen posisi tengah.

- Jika sama maka catat posisi dan cetak kemudian berhenti.
- Jika lebih besar maka akan dilakukan pencarian kembali ke bagian kanan dengan posisi awal = posisi tengah + 1 dan posisi akhir tetap kemudian ulangi mulai poin 2.
- Jika lebih kecil maka akan dilakukan pencarian kembali ke bagian kiri dengan nilai posisi awal tetap dan nilai posisi akhir = posisi tengah – 1 kemudian ulangi mulai poin 2.

Misalkan kita mempunyai sederetan data dalam array nilai sebanyak 10 elemen dan akan dilakukan pencarian data 87 terhadap array.

Nilai[0,9] = 12, 45, 23, 87, 90, 55, 15, 25, 40, 21

Urutkan elemen array secara menaik, sehingga diperoleh:

Nilai[0,9] = 12, 15, 21, 23, 25, 40, 45, 55, 87, 90

Data yang akan dicari = 87 (Bilangan)

Tentukan nilai awal=0, akhir=N-1=9

Hitung tengah =(9+0)/2=4

Bandingkan Bilangan < Nilai[tengah] → 87=25→false

Bandingkan Bilangan < Nilai[tengah] → 87<25→false

Bandingkan Bilangan < Nilai[tengah] → 87>25→true maka pencarian dilakukan ke sebelah kanan dengan nilai awal = tengah + 1 = 5

Karena awal masih lebih kecil dari akhir maka ulangi kembali mulai menghitung tengah

Hitung tengah = (9+5)/2 = 7

Bandingkan Bilangan < Nilai[tengah] → 87=55→false

Bandingkan Bilangan < Nilai[tengah] → 87<55→false

Bandingkan Bilangan < Nilai[tengah] → 87>55→true maka pencarian dilakukan ke sebelah kanan dengan nilai awal = tengah + 1 = 8

Karena awal masih lebih kecil dari akhir maka ulangi kembali mulai menghitung tengah

Hitung tengah = (9+8)/2 = 8

Bandingkan Bilangan < Nilai[tengah] → 87=87→true

Karena sudah ditemukan hasilnya maka proses pencarian berhenti.

Berikut adalah program Lat_Searching_02 untuk pencarian dengan metode binary search.

```
/* Program Pencarian dengan Binary Search
  Nama File : Lat_Searching_02 */

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

void main()
{
  //Deklarasi Variabel
  int Nilai[20];
  int i, j, N;
  int temp, awal, akhir, tengah, Bilangan;

  //Proses Penginputan data
  cout<<"Banyak Bilangan : ";
  cin>>N;
  for(i=0; i<N; i++)
  {
    cout<<"Elemen ke-"<<i<<" = ";
    cin>>Nilai[i];
  }
  cout<<"\nElemen Sebelum diurut = ";
  for(i=0; i<N; i++)
    cout<<setw(3)<<Nilai[i];

  //Proses Pengurutan Data
  for(i=0; i<N-1; i++)
  {
    for(j=i+1; j<N; i++)
    {
      if (Nilai[i] > Nilai[j])
      {
        temp = Nilai[i];
        Nilai[i] = Nilai[j];
        Nilai[j] = temp;
      }
    }
  }
  cout<<"\nElemen Setelah diurut = ";
  for(i=0; i<N; i++)
    cout<<setw(3)<<Nilai[i];
    cout<<"\nindeks Elemen = ";
  for(i=0; i<N; i++)
    cout<<setw(3)<<i;
    cout<<"\nMasukkan data yang akan anda cari: ";
    cin<<Bilangan;

  //Proses Pencarian Data
  awal = 0;
  akhir = N-1;
  do
  {
    tengah = (akhir + awal)/2;
    if(Bilangan < Nilai[tengah])
      akhir = tengah - 1;
    else
      awal = tengah + 1;
  }
  while((akhir >= awal) && (Nilai[tengah] !=Bilangan));
  if(Nilai[tengah] == Bilangan)
  {
    cout<<"\nData "<<Bilangan<<" ada dalam array";
    cout<<" pada posisi "<<tengah;
  }
  else
    cout<<"\nData "<<Bilangan<<" tidak ada dalam array\n";
  getch();
}
```

3. Interpolation Search

Proses pencarian data ini hampir sama dengan proses pencarian binary search. Pencarian ini juga dilakukan pada kumpulan data yang sudah urut (menaik atau menurun). Akan tetapi jika pada binary search kita membagi data dengan (akhir+awal)/2 tiap prosesnya, maka pada interpolation search kita akan membagi data menurut rumus sebagai berikut:

Posisi = (kunci – data[low]/data[high] – data[low]) * (high – low) + low

```
/* Program Pencarian metode Interpolasi Search
  Nama File : Lat_Searching_03 */

#include<stdio.h>
#include<conio.h>
#include<iostream.h>
#include<iomanip.h>
void main()
{
  //deklarasi variable
  int Nilai[20]
  int i,j,Bilangan,N;
  int temp,awal,akhir,tengah'
  //proses penginputan data
  cout<<"Banyak Data : ";
  cin>>N;
  for(i=0; i<N; i++)
  {
    cout<<"elemen ke-"<<i<<" : ";
    cin>>Nilai[i];
  }

  cout<<"\nData sebelum diurut : ";
  for(i=0; i<N; i++)
    cout<<setw(3)<<Nilai[i];

  //proses pengurutan data
  for(i=0; i<N-1; i++)
  {
    for(j=i+1; j<N; j++)
    {
      if (Nilai[i]>Nilai[j])
      {
        temp=Nilai[i];
        Nilai[i]=Nilai[j];
        Nilai[j]=temp;
      }
    }
  }

  cout<<"\nData setelah diurut : ";
  for(i=0; i<N; i++)
    cout<<setw(3)<<Nilai[i];
```

```
  cout<<"\nData setelah diurut : ";
  for(i=0; i<N; i++)
    cout<<setw(3)<<Nilai[i];

  //Input data yang akan dicari
  cout<<"\nMasukkan data yang akan anda cari : ";
  cin>>Bilangan;

  //proses pencarian data
  akhir = N-1;
  awal = 0;
  do
  {
    tengah = ((Bilangan - Nilai(awal)) / (Nilai(akhir) - Nilai(awal))) * (akhir-awal) +
awal;
    if (Nilai(tengah) > Bilangan)
      akhir = tengah-1;
    else if (Nilai(tengah) < Bilangan)
      awal = tengah + 1;
  }
  while(Bilangan>=Nilai(awal))&&(Nilai(tengah)!=Bilangan);
  if (Nilai(tengah) == Bilangan)
    cout<<"\nData "<<Bilangan<<" ada dalam array pada posisi "<<tengah;
  else
    cout<<"\nData "<<Bilangan<<" tidak ada dalam array";
  getch();
 }
```

Thank You

ukrida.ac.id

UKRIDA
Universitas Kristen Krida Wacana

Sorting Data

CYNTHIA HAYAT S.KOM., M.MSI

KRIDA WACANA CHRISTIAN UNIVERSITY
Faculty of Engineering and Computer Science
Departement of Information System

UKRIDA
Universitas Kristen Krida Wacana

Pengertian Sorting pada C++. Salah satu bagian penting dari struktur data adalah proses pengurutan data. Data terkadang akan berada dalam bentuk yang tidak berpola ataupun dengan pola tertentu yang tidak kita inginkan. Namun dalam penggunaannya, kita akan selalu ingin menggunakan data tersebut dalam bentuk yang rapi atau berpola sesuai dengan yang kita inginkan. Maka dari itu proses sorting adalah proses yang sangat penting dalam struktur data. Proses pengurutan banyak ditemukan dalam pemrosesan komputer. Data yang sudah terurut memiliki beberapa keuntungan. Selain mempercepat pencarian, data yang sudah terurut juga dapat dengan mudah menentukan Nilai terbesar atau terkecil.

Pengurutan data memang sangat relevan dan merupakan aktivitas yang sangat penting berkaitan dengan pemrosesan data. Bahkan pengurutan data telah banyak dilakukan dengan bantuan alat. Adanya kebutuhan terhadap peroses pengurutan memunculkan bermacam-macam metode pengurutan yang bertujuan untuk memperoleh metode pengurutan yang optimal.

Data yang diurut dapat berupa data bertipe data dasar atau tipe data bentukan. Jika data bertipe bentukan (*structure*), maka harus disebutkan berdasarkan field apa data tersebut akan diurutkan.

Sama halnya dengan pencarian, pengurutan juga dibedakan menjadi dua kelompok, yaitu:

1. Pengurutan Internal, yaitu pengurutan terhadap sekumpulan data yang disimpan di dalam memori komputere. Umumnya struktur internal yang dipakai untuk pengurutan ini adalah array, sehingga pengurutan internal disebut dengan pengurutan array.
2. Pengurutan Eksternal, yaitu pengurutan data yang disimpan di dalam memori sekunder. Biasanya data dengan berjumlah besar sehingga tidak mampu dimuat semuanya dalam memori komputer. Struktur eksternal yang dipakai adalah arsip (*file*), maka pengurutan ini juga sering disebut dengan pengurutan arsip.

2. Bubble Sort

Bubble Sort adalah metode yang membandingkan elemen yang sekarang dengan elemen-elemen berikutnya. Pembandingan elemen dapat dimulai dari awal atau mulai dari paling akhir. Apabila elemen yang sekarang lebih besar (untuk urut menaik) atau lebih kecil (untuk urut menaik) dari elemen berikutnya, maka posisinya ditukar, tetapi jika tidak maka posisinya tetap.

Contoh : Misalkan kita mempunyai array Nilai sebanyak 8 elemen akan diurutkan secara menaik dengan metode Bubble Sort: 25, 71, 30, 45, 20, 15, 6, 50. Urutan langkah pengurutannya yang dimulai dari belakang seperti berikut dan programnya dapat dilihat pada program Lat_Sorting_01a.

Langkah -1:

25	72	30	45	20	15	6	50
25	72	30	45	20	6	15	50
25	72	30	45	6	20	15	50
25	72	30	6	45	20	15	50
25	72	6	30	45	20	15	50
25	6	72	30	45	20	15	50
6	25	72	30	45	20	15	50

Langkah -2:

6	25	72	30	45	20	15	50
6	25	72	30	45	15	20	50
6	25	72	30	15	45	20	50
6	25	72	15	30	45	20	50
6	25	15	72	30	45	20	50
6	15	25	72	30	45	20	50

Langkah -3:

6	15	25	72	30	45	20	50
6	15	25	72	30	20	45	50
6	15	25	72	20	30	45	50
6	15	25	20	72	30	45	50
6	15	20	25	72	30	45	50

Langkah -4:

6	15	20	25	72	30	45	50
6	15	20	25	72	30	45	50
6	15	20	25	30	72	45	50

Langkah -5:

6	15	20	25	30	72	45	50
6	15	20	25	30	45	72	50

Langkah -6:

6	15	20	25	30	45	59	72

```
/* Program Pengurutan Metode Bubble Sort
   Pengurutan Secara Menaik
   Nama File : Lat_Sorting_01a */

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

void main()
{
  int Nilai[20];
  int i, j, k, N;
  int temp;
  bool tukar;
  cout<<"Masukkan Banyak Bilangan : ";
  cin>>N;
  for(i=0; i<N; i++)
  {
    cout<<"Elemen ke-"<<i<<" : ";
    cin>>Nilai[i];
  }
}

//Proses Cetak Sebelum diurutkan
cout<<"\nData sebelum diurut : ";
for(i=0; i<N; i++)
  cout<<setw(3)<<Nilai[i];
```

```
//Proses Pengurutan
i=0;
tukar = true;
while ((i<=N-2) && (tukar))
{
  tukar = false;
  for(j=N-1; j>=i+1; j--)
  {
    if(Nilai[j] < Nilai[j-1])
    {
      {
        temp = Nilai[j];
        Nilai[j] = Nilai[j-1];
        Nilai[j-1] = temp;
        tukar = true;
        cout<<"\nUntuk j = " <<j<<" : ";
        for(k=0; k<N; k++)
          cout<<set(3)<<Nilai[k];
      }
    }
    i++;
  }

  //Proses Cetak Setelah diurutkan
  cout<<"\nData Setelah di urut : ";
  for(i=0; i<N; i++)
    cout<<setw(3)<<Nilai[i];
  getch();
}
```

3. Quick Sort

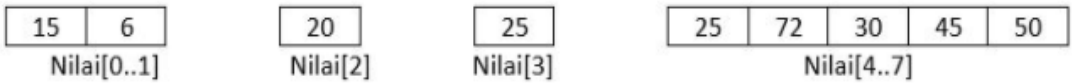
Quick Sort merupakan metode tercepat dalam proses pengurutan data dengan menggunakan prinsip rekursif. Metode ini menggunakan strategi "pecah-belah" dengan mekanisme berikut ini.

Misalkan kita mempunyai array Nilai[k..l]. Array dipartisi menjadi dua bagian array kiri Nilai[k..m] dan array kanan Nilai[m+1..l]. Dasar mempartisi array menjadi dua adalah dengan mengambil elemen pertama sebagai elemen pivot. Letakkan semua elemen array yang lebih kecil dari pivot ke sebelah pivot dan semua elemen array yang lebih besar dari pivot ke sebelah kanan pivot. Elemen-elemen yang di sebelah kiri elemen pivot merupakan elemen-elemen array Nilai[k..m] sedangkan elemen-elemen array Nilai[m+2..l] adalah semua elemen yang lebih besar dari pivot. Lakukan hal yang sama seperti di atas terhadap array Nilai[k..m] dan Nilai[m+1..l] hingga tidak dapat dipartisi lagi.

Contoh: Misalkan kita mempunyai array Nilai sebanyak 8 elemen akan diurutkan secara menaik dengan metode Maximum Sort: 25, 72, 30, 45, 20, 15, 6, 50. Urutan langkah pengurutannya seperti berikut dan programnya dapat dilihat pada program Lat_Sorting_02a.

1. Ambil elemen pertama sebagai elemen pivot, letakkan semua elemen array yang lebih kecil dari pivot ke sebelah kiri elemen pivot dan letakkan semua elemen array yang lebih besar dari pivot ke sebelah kanan elemen pivot.

2. Gunakan array Nilai[0..2]. Ambil elemen pertama sebagai elemen pivot, letakkan semua elemen array yang lebih kecil dari pivot ke sebelah kiri elemen pivot dan letakkan semua elemen array yang lebih besar dari pivot ke sebelah kanan elemen pivot.



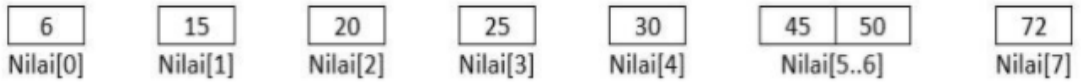
Perhatikan array Nilai[2] tidak dapat lagi dipartisi maka berhenti sampai disana.

3. Gunakan array Nilai[0..1]. Ambil elemen pertama sebagai elemen pivot, letakkan semua elemen array yang lebih kecil dari pivot ke sebelah kiri elemen pivot dan letakkan semua elemen array yang lebih besar dari pivot ke sebelah kanan elemen pivot.

Perhatikan array Nilai[0] dan Nilai[1] tidak dapat lagi dipartisi maka berhenti sampai di sana.

4. Gunakan array Nilai[4..7]. Ambil elemen pertama sebagai elemen pivot, letakkan semua elemen array yang lebih kecil dari pivot ke sebelah kiri elemen pivot dan letakkan semua elemen array yang lebih besar dari pivot ke sebelah kanan elemen pivot.

5. Gunakan array Nilai[4..6]. Ambil elemen pertama sebagai elemen pivot, letakkan semua elemen array yang lebih kecil dari pivot ke sebelah kiri elemen pivot dan letakkan semua elemen array yang lebih besar dari pivot ke sebelah kanan elemen pivot.



6. Gunakan array Nilai[5..6]. Ambil elemen pertama sebagai elemen pivot, letakkan semua elemen array yang lebih kecil dari pivot ke sebelah kiri elemen pivot dan letakkan semua elemen array yang lebih besar dari pivot ke sebelah kanan elemen pivot.

Karena semua elemen array sudah tidak dapat dipartisi maka proses pengurutan berakhir dan hasilnya diperoleh seperti berikut (gabungkan mulai Nilai[0] hingga Nilai[7]):

Untuk melakukan proses pengurutan data secara menurun dengan metode Quick Sort dilakukan dengan meletakkan semua elemen array yang lebih kecil dari pivot ke sebelah kanan pivot dan semua elemen array yang lebih besar dari pivot ke sebelah kiri pivot.

```
/* Program Pengurutan Metode Quick Sort
   Pengurutan Secara Menaik
   Nama File : Lat_Sorting_02a */

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

void Cetak(int data[], int n)
{
  int i;
  for(i=0; i<n; i++)
    cout<<setw(3)<<data[i];
  cout<<"\n";
}

int Partisi(int data[], int p, int r)
{
  int x, i, j, temp;
  x = data[p];
  i=p;
  j=r;
  while(1)
  {
    while(data[j]>x)
      j--;
    while(data[i]<x)
      i++;
    if(i<j)
    {
      temp = data[i];
      data[i] = data[j];
      data[j] = temp;
    }
    else
      return j;
  }
}

void Quick_Sort(int data[], int p, int r)
{
  int q;
  if(p<r)
  {
    q=Partisi(data, p, r+1);
    Quick_Sort(data, p, 1);
    Quick_Sort(data, q+1, r);
  }
}

void main()
{
  int Nilai[20];
  int i, N;
  cout<<"Masukkan Banyak Bilangan : ";
  cin>>N;
  for(i=0; i<N; i++)
  {
    cout<<"Elemen ke-"<<i<<" : ";
    cin>>Nilai[i];
  }
  cout<<"\nData Sebelum di urut : ";
  Cetak(Nilai, N);
  cout<<endl;
  Quick_Sort(Nilai, 0, N-1);
  cout<<"\nData Setelah di urut : ";
  Cetak(Nilai,N);
  getch();
}
```

4. Metode Maximum/Minimum Sort

Metode Maximum/Minimum Sort dilakukan berdasarkan pemilihan elemen maksimum/minimum, maka metode ini disebut juga dengan metode pemilihan/seleksi (*Selection Sort*).

4.1 Metode Maximum Sort

Metode ini disebut dengan metode Maximum karena didasarkan pada pemilihan elemen maksimum sebagai dasar pengurutan. Konsepnya adalah memilih elemen maksimum kemudian mempertukarkan elemen maksimum tersebut dengan elemen paling akhir untuk urut menaik dan elemen pertama untuk urut menurun. Selanjutnya elemen paling akhir/awal tersebut di-"isolasi", artinya elemen tersebut tidak disertakan lagi untuk tahapan berikutnya. Proses yang sama dilakukan kembali terhadap elemen array yang tersisa, yaitu memilih elemen maksimum kemudian mempertukarkan elemen maksimum tersebut dengan elemen paling akhir/awal dari array yang tersisa tadi. Kemudian diisolasi kembali. Demikian seterusnya hingga semua elemen terurut.

Contoh: Misalkan kita mempunyai array Nilai sebanyak 8 elemen akan diurutkan secara menaik dengan metode Maximum Sort: 25, 72, 30, 45, 20, 15, 6, 50. Urutan langkah pengurutannya seperti berikut dan programnya dapat dilihat pada program Lat_Sorting_04a.

Langkah -1: Cari elemen maksimum di dalam array Nilai[0..7] → Nilai[1]=72 kemudian tukarkan dengan elemen array paling akhir Nilai[7] sehingga diperoleh array:

25	50	30	45	20	15	6	72

Langkah -2: Cari elemen maksimum di dalam array yang tersisa Nilai[1..7] → Nilai[1]=50 kemudian tukarkan dengan elemen paling akhir array yang tersisa Nilai[6] sehingga diperoleh array:

25	6	30	45	20	15	50	72

Langkah -3: Cari elemen maksimum di dalam array yang tersisa Nilai[2..7] → Nilai[3]=45 kemudian tukarkan dengan elemen paling akhir array yang tersisa Nilai[5] sehingga diperoleh array:

25	6	30	15	20	45	50	72

Langkah -4: Cari elemen maksimum di dalam array yang tersisa Nilai[3..7] → Nilai[2]=30 kemudian tukarkan dengan elemen paling akhir array yang tersisa Nilai[4] sehingga diperoleh array:

25	6	20	15	30	45	50	72

Langkah -5: Cari elemen maksimum di dalam array yang tersisa Nilai[4..7] → Nilai[0]=25 kemudian tukarkan dengan elemen paling akhir array yang tersisa Nilai[3] sehingga diperoleh array:

15	6	20	25	30	45	50	72

Langkah -6: Cari elemen maksimum di dalam array yang tersisa Nilai[5..7] → Nilai[2]=20 kemudian tukarkan dengan elemen paling akhir array yang tersisa Nilai[2], sehingga diperoleh array:

15	6	20	25	30	45	50	72

Langkah -7: Cari elemen maksimum di dalam array yang tersisa Nilai[6..7] → Nilai[0]=15 kemudian tukarkan dengan elemen paling akhir array yang tersisa Nilai[1], sehingga diperoleh array:

6	15	20	25	30	45	50	72

Selesai. Dan array telah terurut secara menarik. Program mengurutkan secara menaik dengan metode Maximum Sort dapat dilihat pada program Lat_Sorting_04a.

```
/* Program Pengurutan Metode Maximum Sort
   Pengurutan Secara Menaik
   Nama File : Lat_Sorting_04a */

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

void main()
{
  int Nilai[20];
  int i, j, N, l;
  int temp, U, Imaks;
  cout<<"Masukkan Banyak Bilangan : ";
  cin>>N;
  for(i=0; i<N; i++)
  {
    cout<<"Elemen ke-"<<i<<" : ";
    cin>>Nilai[i];
  }

  //Proses Cetak Sebelum diurutkan
  cout<<"\nData sebelum diurut : ";
  for(i=0; i<N; i++)
    cout<<setw(3)<<Nilai[i];

  //Proses Pengurutan
  U=N-1;
  for(i=0; i<=N-2; i++)
  {
    Imaks = 0;
    for(j=1; j<=U; j++)
    {
      if(Nilai[j] > Nilai[Imaks])
      Imaks = j;
    }
    temp = Nilai[U];
    Nilai[U] = Nilai[Imaks];
    Nilai[Imaks] = temp;
    U--;
    cout<<endl;
    for(l=0; l<N; l++)
      cout<<setw(3)<<Nilai[l];
  }
  cout<<"\nData Setelah di urut : ";
  for(i=0; i<N; i++)
    cout<<setw(3)<<Nilai[i];
  getch();
}
```

4.2. Metode Minimum Sort

Metode ini disebut dengan metode minimum karena didasarkan pada pemilihan elemen minimum sebagai dasar pengurutan. Konsepnya adalah memilih elemen minimum kemudian mempertukarkan elemen minimum tersebut dengan elemen paling akhir untuk urut menaik dan elemen pertama untuk urut menurun. Selanjutnya elemen paling akhir/pertama tersebut di "isolasi" artinya elemen tersebut tidak disertakan lagi untuk tahapan berikutnya. Proses yang sama dilakukan kembali terhadap elemen array yang tersisa, yaitu memilih elemen minimum kemudian mempertukarkan elemen minimum tersebut dengan elemen paling akhir/pertama dari array yang tersisa tadi. Kemudian diisolasi kembali. Demikian seterusnya hingga semua elemen terurut.

Contoh: Misalkan kita mempunyai array Nilai sebanyak 8 elemen akan diurutkan secara menaik dengan metode Minimum Sort: 25, 72, 30, 45, 20, 15, 6, 50. Urutan langkah pengurutannya seperti berikut dan programnya dapat dilihat pada program Lat_Sorting_05a.

Langkah -1: Cari elemen minimum di dalam array Nilai[0..7] → Nilai[6]=6 kemudian tukarkan dengan elemen pertama array Nilai[0]=25 sehingga diperoleh array:

6	72	30	45	20	15	25	

Langkah -2: Cari elemen minimum di dalam array Nilai[1..7] → Nilai[5]=15 kemudian tukarkan dengan elemen pertama array yang tersisa Nilai[1]=72 sehingga diperoleh array:

6	15	30	45	20	72	25	

Langkah -3: Cari elemen minimum di dalam array Nilai[2..7] → Nilai[4]=2- kemudian tukarkan dengan elemen pertama array yang tersisa Nilai[2]=30 sehingga diperoleh array:

6	15	20	45	30	72	25	

Langkah -4: Cari elemen minimum di dalam array Nilai[3..7] → Nilai[6]=25 kemudian tukarkan dengan elemen pertama array yang tersisa Nilai[3]=45 sehingga diperoleh array:

6	15	20	25	30	72	45	

Langkah -5: Cari elemen minimum di dalam array Nilai[4..7] → Nilai[4]=30 kemudian tukarkan dengan elemen pertama array yang tersisa Nilai[4]=30 sehingga diperoleh array:

6	15	20	25	30	72	45	

Langkah -6: Cari elemen minimum di dalam array Nilai[5..7] → Nilai[6]=45 kemudian tukarkan dengan elemen pertama array yang tersisa Nilai[5]=72 sehingga diperoleh array:

6	15	20	25	30	45	72	

Langkah -7: Cari elemen minimum di dalam array Nilai[6..7] → Nilai[7]=50 kemudian tukarkan dengan elemen pertama array yang tersisa Nilai[6]=72 sehingga diperoleh array:

6	15	20	25	30	45	50	

```
/* Program Pengurutan Metode Minimum Sort
   Pengurutan Secara Menaik
   Nama File : Lat_Sorting_05a */

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

void main()
{
  int Nilai[20];
  int i, j, N, l;
  int temp, Imin;
  cout<<"Masukkan Banyak Bilangan : ";
  cin>>N;
  for(i=0; i<N; i++)
  {
    cout<<"Elemen ke-"<<i<<" : ";
    cin>>Nilai[i];
  }

  //Proses Cetak Sebelum diurutkan
  cout<<"\nData sebelum diurut : ";
  for(i=0; i<N; i++)
    cout<<setw(3)<<Nilai[i];

  //Proses Pengurutan
  for(i=0; i<=N-2; i++)
  {
    Imin = i;
    for(j=i+1; j<N; j++)
    {
      if(Nilai[j] < Nilai[Imin]);
      Imin = j;
    }
  temp = Nilai[i];
  Nilai[i] = Nilai[Imin];
  Nilai[Imin] = temp;
  cout<<endl;
  for(l=0; l<N; l++)
    cout<<setw(3)<<Nilai(l);
  }
  cout<<"\nData Setelah di urut : ";
  for(i=0; i<N; i++)
    cout<<setw(3)<<Nilai[i];
  getch();
}
```

7. Metode Insertion Sort

Metode Insertion Sort merupakan metode pengurutan dengan cara menyisipkan elemen array pada posisi yang tepat. Pencarian yang tepat dilakukan dengan melakukan pencarian beruntun di dalam array. Selama pencarian posisi yang tepat dilakukan pergeseran elemen array. Algoritma pengurutan ini tepat untuk persoalan menyisipkan elemen baru ke dalam array yang sudah terurut. Misalnya dalam permainan kartu, kartu yang dicabut biasanya disisipkan oleh pemain pada posisi yang tepat sehingga penambahan kartu tersebut membuat semua kartu tetap terurut.

Misalkan kita memiliki suatu array dengan N maka pengurutan secara menaik dengan metode Insertion Sort sebagai berikut:

Langkah -1: elemen pertama Nilai[0] diasumsikan telah sesuai tempatnya.

Langkah -2: ambil elemen ke dua (Nilai[1]), cari lokasi yang tepat pada Nilai[0..0] untuk Nilai Nilai[1]. Lakukan pergeseran ke kanan jika Nilai[0..1] lebih besar (untuk urut menaik) atau lebih kecil (untuk urut menurun) dari Nilai[1]. Misalnya posisi yang tepat adalah j, maka sisipkanlah Nilai[1] pada Nilai[j].

Langkah -3: ambil elemen ke tiga (Nilai[2]), cari lokasi yang tepat pada Nilai[0..1] untuk Nilai[2]. Lakukan pergeseran ke kanan jika Nilai[0..2] lebih besar (untuk urut menaik) atau lebih kecil (untuk urut menurun) dari Nilai[2]. Misalnya posisi yang tepat adalah j, maka sisipkanlah Nilai[2] pada Nilai[j].

Langkah -4: ambil elemen ke empat (Nilai[3]), cari lokasi yang tepat pada Nilai[0..3] untuk Nilai Nilai[3]. Lakukan pergeseran ke kanan jika Nilai[0..2] lebih besar (untuk urut menaik) atau lebih kecil (untuk urut menurun) dari Nilai[3]. Misalnya posisi yang tepat adalah j, maka sisipkanlah Nilai[3] pada Nilai[j].

...

Langkah –N: ambil elemen ke N (Nilai[N]), cari lokasi yang tepat pada Nilai[0..N-1] untuk Nilai Nilai[N]. Lakukan pergeseran ke kanan jika Nilai[0..N-1] lebih besar (untuk urut menaik) atau lebih kecil (untuk urut menurun) dari Nilai[N]. Misalnya posisi yang tepat adalah j, maka sisipkanlah Nilai[N] pada Nilai[j].

Contoh: Misalkan kita mempunyai array Nilai sebanyak 8 elemen akan diurutkan secara menaik dengan metode Insertion Sort: 25, 72, 30, 45, 20, 15, 6, 50. Urutan langkah pengurutannya seperti berikut:

Langkah -1: Nilai[0] diasumsikan telah terurut.

25	72	30	45	20	15	6	

Langkah -2: cari lokasi yang tepat untuk Nilai[1] pada Nilai[0..0]. Dalam hal ini, ternyata 72 tidak lebih besasr dari 25 maka tidak terjadi pergeseran, sehingga diperoleh array seperti berikut:

25	72	30	45	20	15	6	

Langkah -3: cari lokasi yang tepat untuk Nilai[2] pada Nilai[0..1]. Dalam hal ini, ternyata lokasi yang tepat adalah 1, maka Nilai[1] digeser ke kanan sehingga Nilai[2] di posisi ke 1, sehingga diperoleh array sebagai berikut:

25	30	72	45	20	15	6	

Langkah -4: cari lokasi yang tepat untuk Nilai[3] pada Nilai[0..2]. Dalam hal ini, ternyata lokasi yang tepat adalah 2, maka Nilai[2] digeser ke kanan sehingga Nilai[3] di posisi ke 3, sehingga diperoleh array seperti berikut:

25	30	45	72	20	15	6	

Langkah -5: cari lokasi yang tepat untuk Nilai[4] pada Nilai[0..3]. Dalam hal ini, ternyata lokasi yang tepat adalah 0, maka Nilai[0], Nilai[1], Nilai[2], Nilai[3] digeser masing-masing satu posisi ke kanan sehingga Nilai[4] di posisi ke 0, sehingga array seperti berikut:

20	25	30	45	72	15	6	

Langkah -6: cari lokasi yang tepat untuk Nilai[5] pada Nilai[0..4]. Dalam hal ini, ternyata lokasi yang tepat adalah 0, maka Nilai[0], Nilai[1], Nilai[2], Nilai[3] dan Nilai[4] digeser masing-masing satu posisi ke kanan sehingga Nilai[6] di posisi ke 0, diperoleh array seperti berikut:

15	20	25	30	45	72	6	

Langkah -7: cari lokasi yang tepat untuk Nilai[6] pada Nilai[0..5]. Dalam hal ini, ternyata lokasi yang tepat adalah 0, maka Nilai[0], Nilai[1], Nilai[2], Nilai[3], Nilai[4] dan Nilai[5] digeser masing-masing satu posisi ke kanan sehingga Nilai[6] di posisi ke 0, diperoleh array seperti berikut:

6	15	20	25	30	45	72	

Langkah -8: cari lokasi yang tepat untuk Nilai[7] pada Nilai[0..6]. Dalam hal ini, ternyata lokasi yang tepat adalah 6, maka Nilai[6] digeser satu posisi ke kanan sehingga Nilai[7] di posisi ke 6, diperoleh array seperti berikut:

6	15	20	25	30	45	50	72

Selesai. Dan array telah terurut secara menaik. Program untuk mengurutkan elemen array secara menaik dengan metode Insertion Sort dapat dilihat pada program Lat_Sorting_07a.

```
/* Program Pengurutan Metode Insertion Sort
   Pengurutan Secara Menaik
   Nama File : Lat_Sorting_07a */

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

void main()
{
  int Nilai[20];
  int i, j, k, N;
  int temp;
  cout<<"Masukkan Banyak Bilangan : ";
  cin>>N;
  for(i=0; i<N; i++)
  {
    cout<<"Elemen ke-"<<i<<" : ";
    cin>>Nilai[i];
  }

  //Proses Cetak sebelum diurutkan
  cout<<"\nData sebelum diurut : ";
  for(i=0; i<N; i++)
    cout<<setw(3)<<Nilai[i];

  //Proses Pengurutan
  for(i=1; i<N; i++)
  {
    temp = Nilai[i];
    j=i-1;
    while((temp <= Nilai(j)) && (j>=1))
    {
      Nilai[j+1] = Nilai[j];
      j--;
    }
    if(temp >= Nilai[j])
      Nilai[j+1] = temp;
    else
    {
      Nilai[j+1] = Nilai[j];
      Nilai[j] =  temp;
    }
    cout<<endl;
    for(k=0; k<N; k++)
      cout<<setw(3)<<Nilai[k];
  }

  //Proses Cetak Setelah diurutkan
  cout<<"\nData Setelah di urut : ";
  for(i=0; i<N; i++)
    cout<<setw(3)<<Nilai[i];
  getch();
}
```

Thank You

ukrida.ac.id

UKRIDA
Universitas Kristen Krida Wacana