KODING C++

Nur Wachid

Koding C++ dengan Qt

Turahe (Nur Wachid)

Buku ini dijual di http://leanpub.com/koding-cpp-qt

Versi ini diterbitkan pada 2018-08-10

Ini adalah sebuah buku Leanpub. Leanpub memberdayakan penulis dan penerbit dengan proses Lean Publishing. Lean Publishing adalah model penerbitan ebook dalam-proses menggunakan piranti ringan dan sejumlah iterasi untuk memperoleh masukan dari pembaca, menerapkan pivot hingga Anda dapat mewujudkan komposisi buku yang pas dan menarik.

Juga Oleh Turahe (Nur Wachid)

HTML CSS

Saya persembahkan buku ini bagi para peminat bahasa C/C++ yang ingin tahu lebih dalam, karena sesungguhnya ilmu itu milik Allah semata.

Contents

1. Mukadimah

1.1 Pengenalan Bahasa C++

Bahasa C merupakan bahasa pemrograman tingkat menengah. Pada tahun 1972 bahasa C pertama kali dirancang oleh Dennis M Ritchie[1] di Bell aboratories. Kemudian tahun 1978 Dennis dan Brian W. Kernighan[2]mempublikasikan bahasa C melalui The C Programming Language[3] sehingga bahasa C dikenal banyak orang. Selanjutnya pada tahun 1989, akhirnya bahasa C distandardisasi ANSI[4] (American National Standard Institute) sehingga bahasa C menjadi bahasa pemrograman standar hingga saat ini dan bisa dibuat kompilernya pada beberapa platform yang berbeda-beda.

Bahasa C dikatakan sebagai bahasa pemrograman terstruktur, fungsional karena strukturnya menggunakan fungsi-fungsi sebagai program-program bagian (subroutine/ module). Fungsifungsi selain fungsi utama disebut subroutine/ module dan ditulis setelah fungsi utama (main) atau diletakkan pada file pustaka (library). Jika fungsi-fungsi diletakkan pada file pustaka dan akan dipakai disuatu program, maka nama file header-nya harus dilibatkan dalam program menggunakan preprocessor directive `#include`.

Kemudian bahasa C dikembangkan oleh Bjarne Stroustrup at Bell Labs menjadi bahasa C++. Pada bulan Oktober 1985 munculah buku *The C++ Programming Language* yang membahas tentang bahasa pemrograman itu langsung dari penciptanya sendiri.

Bahasa C++ mengalami dua tahap evolusi.

- Pertama, dirilis oleh AT&T Laboratories[5], dinamakan cfront[6]. C++ versi ini hanya berupa kompiler yang menterjemahkan bahasa C++ menjadi bahasa C untuk dieksekusi

[1] https://id.wikipedia.org/wiki/Dennis_Ritchie
[2] https://id.wikipedia.org/wiki/Brian_Kernighan
[3] https://id.wikipedia.org/wiki/The_C_Programming_Language
[4] https://id.wikipedia.org/wiki/ANSI_C
[5] https://id.wikipedia.org/wiki/Bell_Laboratories
[6] https://en.wikipedia.org/wiki/Cfront

- Kedua, Borland International Inc[7]. mengembangkan kompiler C++ menjadi sebuah kompiler yang mampu mengubah C++ langsung menjadi bahasa mesin (assembly). Tahun1990, C++ mulai diarahkan ke pengembangan Pemrograman Berorientasi Obyek[8].

Beberapa keunggulan C++ dibandingkan dengan bahasa C adalah sebagai berikut.

Object-oriented programming
: Bahasa pemrograman ini sangat mendukung pemrograman berorientasi obyek yang melihat permasalahan secara obyek dan bukan prosedural.

Portability
: Kita dapat mengkompilasi C++ kode yang sama di hampir semua jenis komputer dan sistem operasi tanpa membuat perubahan apapun. C++ adalah bahasa pemrograman yang paling sering digunakan di dunia.

Brevity
: Karena bahasa C++ merupakan bahasa tingkat tinggi, maka bahasa yang ditulis dengan bahasa C++ termasuk ringkas dan pendek dibandingkan bahasa-bahasa sejamannya pada waktu itu.Bahasa C++ termasuk bahasa pemrograman tua yang sudah mendukung berbagai macam kata kunci yang mampu menyingkat proses penulisan kode program.

Modular programming
: Tubuh program pada bahasa C++ dapat terdiri dari beberapa file source code yang disusun secara terpisah dan kemudian dihubungkan secara bersama-sama. Kemampuan ini jelas menghemat waktu karena tidak perlu mengkompilasi ulang aplikasi yang lengkap ketika membuat satu perubahan, tetapi hanya file yang berisi perubahan itu saja. Selain itu, karakteristik ini memungkinkan kita untuk menghubungkan kode C++ dengan kode yang dibuat oleh bahasa lain, seperti bahasa Assembly dan C dan dapat digunakan kembali (reuseable).

C Compatibility
: C++ sangat backward compatible dengan bahasa C, sehingga aplikasi / kode program yang ditulis dengan bahasa C dapat digabungkan dengan bahasa C++ dengan sangat mudah, bahkan hampir tidak perlu mengubah kodenya.

Speed
: Kode yang dihasilkan dari kompilasi C++ sangat efisien, karena C++ mendukung prinsip dualitas bahwa dia mendukung bahasa tingkat tinggi dan

[7] https://en.wikipedia.org/wiki/Borland
[8] https://id.wikipedia.org/wiki/Pemrograman_berorientasi_objek

bahasa tingkat rendah sehingga dapat mengurangi ukuran hasil kompiliasi dari bahasa itu.

1.2 Pengantar Qt Creator

Qt Creator merupakan cross-platform C++ integrated development environment yang merupakan bagian dari Qt SDK. Qt Creator mempunyai debugger dalam bentuk visual dan layout GUI serta tempat perancangan form. Teks editornya mempunyai fasilitas syntax highlighting dan autocompletion. Qt Creator menggunakan compiler C++ dari kumpulan compiler GNU pada Linux dan FreeBSD. Pada Windows Qt Creator dapat menggunakan MinGW[9] atau MSVC[10] yang sudah build-in di dalam install.

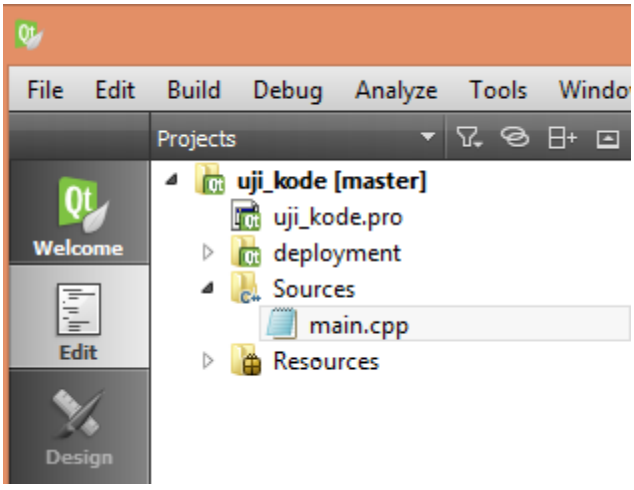
Project Qt Creator menggunakan format cross platform project (.pro) untuk mengizinkan tim developer untuk share project yang mempunyai platform-platform yang berbeda-beda dan menggunakan common tool untuk implementasi dan debugging program. Sebuah project dapat meliputi:

file-file yang digroup secara bersama-sama, langkah-langkah build program, form-form dan file-file resource, dan pengaturan untuk menjalankan aplikasi.

Projek dapat dibuat secara manual atau diimport dari file projek yang sudah ada. Jika projeknya dibuat secara manual, maka sebuah file-file akan digenerate oleh Qt Creator, tergantung dari tipe file yang dimiliki. Seperti Jika filenya adalah sebuah GUI application, maka Qt Creator men-generate sebuah file kosong yang berekstensikan `.ui` yang akan imodifikasikan melalui Qt Designer. Qt Creator diinterrasikan dengan sistem cross-platform untuk mem-build secara automatis: qmake dan CMake. Projek yang tidak menggunakan qmake atau CMake dapat diimport-kan, dan Qt Creator dapat meng-ignore sistem build.

[9] minGW adalah salah satu aplikasi yng digunakan untuk mengkompile bahasa C agar dapat dipahami oleh bahasa mesin (asembler) pada komputer. Aplikasi ini dapat di unduh secara gratis.

[10] https://id.wikipedia.org/wiki/Microsoft_Visual_Studio_Express

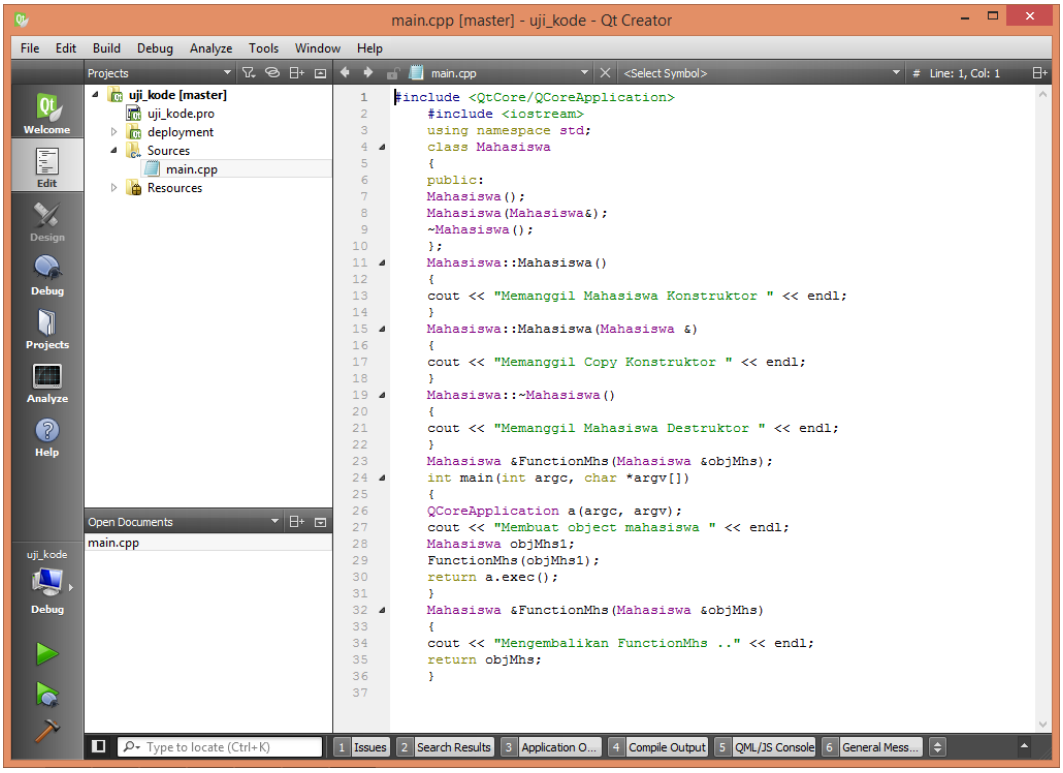


Gambar Projek pada Qt Creator

Editor Qt Creator mempunyai sebuah code editor yang telah terintegrasi dengan Qt Designer untuk mendesain dan membangun aplikasi GUI dari Qt widgets. Karena Qt Creator adalah sebuah Integrated Development Enviroment (IDE), maka Qt Creator memisahkan antara text editor untuk build dan editor untuk menjalankan (run) aplikasi-aplikasi. Qt Creator bukan hanya bisa membaca text file biasa, akan tetapi juga bisa membaca file C++ dan bahasa QML.

Keunggulan Code Editor Qt Creator

- Dapat menulis code dengan format yang benar.
- Mengantisipasikan apa yang akan programer tulis dan code yang komplit.
- Menampilkan baris-baris yang error dan pesan-pesan warning.
- Memandu programer secara semantik untuk menulis classes, functions, dan symbols.
- Menyediakan fasilitas bantuan context-sensitive pada classes, functions, dan symbols.
- Me-rename symbol-symbol dengan langkah intelligent, sehingga simbol-simbol yang lain dengan nama yang sama tidak ter-rename.
- Menampilkan lokasi function, class yang dideklarasikan atau yang dipanggil



Gambar Code Editor

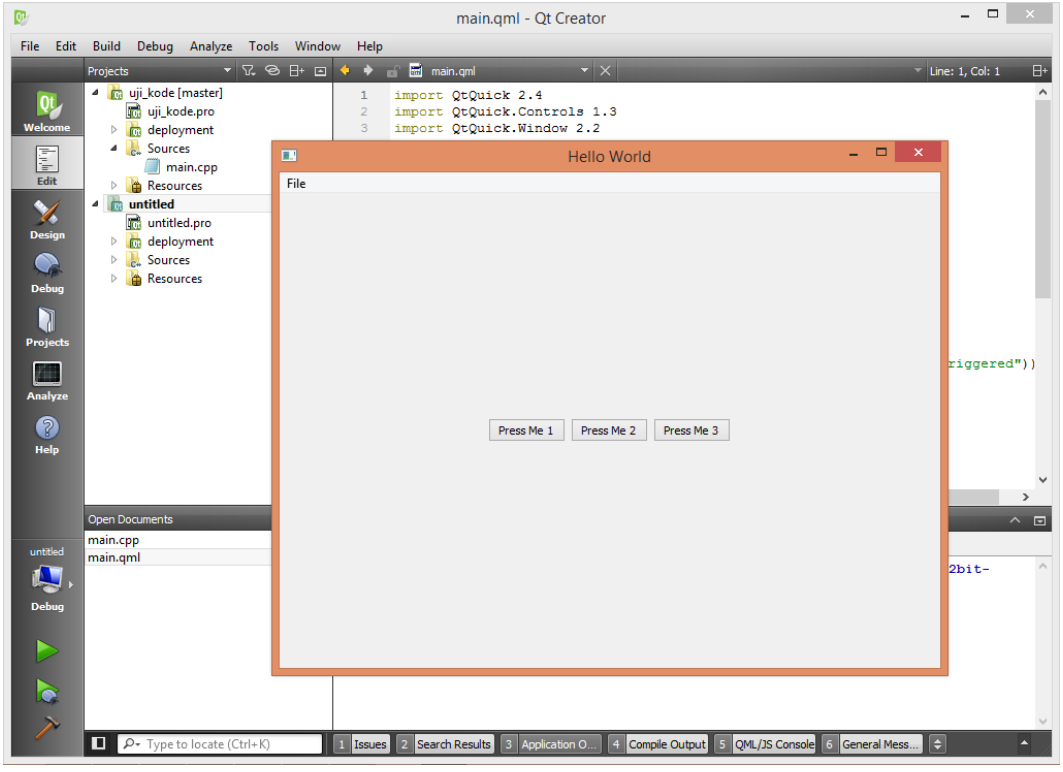
a) UI Designer

Qt Creator menyajikan dua buah editor visual: Qt Designer dan Qt Quick Designer. Qt Designer merupakan sebuah tool untuk mendesain dan membangun aplikasi GUI dari Qt widgets. Widgets dan forms yang dibentuk dengan Qt Designer terintegrasi dengan code program, Qt signals dan mekanisme slots, sehingga kita dengan mudah memberikan nilai-nilai dan properti-properti pada pada elemen-elemen grafik. Semua properti-properti yang diatur pada Qt Designer dapat diubah secara dinamik melalui/di dalam code.

Qt Quick Designer digunakan untuk membangun secara mudah animasi-animasi dengan menggunakan sebuah bahasa pemograman yang dikenal dengan Qt Meta-Object Language (QML)[11]. Dalam QML, sebuah user interface dispesifikasikan

[11] https://en.wikipedia.org/wiki/QML

sebagai sebuah pohon (tree) dari objects dengan properti-properti. Kamu menggunaan teks editor visual untuk menciptakan items, screens, dan aplikasi, serta mendefinisikan perubahan action-acton pada komponennya. Dapat digunakan Qt atau JavaScript untuk mengimplementasikan logik aplikasi.



Gambar UI Designers

b) Bahasa yang di dukung

Anda dapat menggunakan code editor menulis code dalam Qt C++ atau bahasa pemograman QML, javascript bahkan HTML5. Syntax highlighting juga disajikan untuk banyak bahasa pemograman yang lain.

c) Platform

Qt Creator men-support untuk membangun dan menjalankan aplikasi-aplikasi Qt untuk desktop environments (Seperti Windows, Linux, FreeBSD dan Mac OS) Selain itu juga bisa dijalankan pada mobile devices (seperti Android, windows 8 dan iOS). Ketika sebuah aplikasi dibangun untuk mobile device yang bisa mengkoneksi ke Personal Computer (PC), maka Qt Creator men-generate sebuah package instalasi, menginstall package tersebut pada device, dan meneksekusikannya.

d) Tools

Qt Creator diintegrasikan dengan kumpulan tool-tool yang bermanfaat dan membantu, seperti version control systems dan Qt Simulator. Qt Creator menggunakan command line client version control system untuk mengakses repositories (Git[12], [subversion]https://subversion.apache.org/), Perfoce[13], CVS[14], Mercurial[15]).

e) Debuggers

Qt Creator tidak mempunyai debugger. Qt Creator mempunyai plugin debugger yang bekerja sebagai interface antara Qt Creator core dan external native debuggers
Debuggers adalah:

- GNU Symbolic Debugger (gdb)
- Microsoft Console Debugger (CDB)
- internal Java Script debugger

Dapat menghubungkan mobile devices dengan PC dan memproses debug yang berjalan pada devices.

[12] https://git-scm.com/
[13] https://www.perforce.com
[14] www.nongnu.org/cvs/
[15] https://www.mercurial-scm.org

1.3 Mengenal macam - macam Teknologi User Interface (UI) pada QT

Interaksi antara pengguna dengan logic software dinamakan User Interface disingkat dengan UI. UI ini berwujud bisa sebuah window, bisa tombol, bisa sebuah textarea dan lain sebagainya. Inilah komponen User Interface. Sebagai seorang programmer (pembuat program aplikasi), terlebih programmer yang menggunakan Qt, maka anda akan disediakan beragam UI yang bisa anda pilih sesuai kebutuhan. Ada QtWitget, QtQuick dan QtWebKit. Ketiganya dapat anda pilih sesuai kebutuhan sebagai UI progam anda. Ingin tahu bedanya? Mari kita simak.

Sama seperti Visual Studio yang menyediakan beragam Amazing User Interface tingkat tinggi sampai pengguna bingung memilihnya, Qt menyediakan tiga UI yang dapat kita gunakan. Anda pun bisa menggabungkan ketiganya. :)

Qt Creator, sebuah Editor Qt adalah contoh dari perpaduan multiple teknologi UI ini. Coba amati Qt Creator yang selalu anda gunakan tersebut.

Qt Creator menggunakan teknologi QtWidget sebagai User Interface pada menu dan kotak dialog nya. Coba perhatikan kembali Welcome Screen dari Qt Creator, lihat, tampilannya berbeda bukan, bahkan button nya sangat berbeda dan tidak biasa, ini karena UI untuk Welcome Screen menggunakan teknologi QtQuick. Lalu dimana letak penggunaan QtWebkit?

Yup, perhatikan Help Documentation nya, wow, ini seperti halaman web yang menyatu dengan softwarenya bukan? Ya, teknologi QtWebKit digunakan dalam pembangunan Help Documentation ini.

a) Teknologi User Interface QtQuick

QtQuick merupakan salah satu Teknologi UI dari Qt yang menggunakan QML dan JavaScript sebagai penyusun UI nya.Mirip seperti XAML yang dikembangkan Microsoft , QML merupakan teknologi binding dari Qt yang memfasilitasi pengguna dengan visual canvas dan rendering engine nya. Teknologi UI ini sangat cocok sekali untuk Hardware Acceleration seperti OpenGL pada VGA driver kita.

Jangan salah, bila anda menggunakan QtQuick versi 2, maka memang butuh OpenGL yang disuport oleh VGA anda. OpenGL sekarang begitu terkenal, banyak games – games yang mulai menargetkan OpenGL karena begitu flexibel dan mudah. Tapi sayangnya, OpenGL ini tidak terdapat pada VGA driver bawaan OS.

Jadi jangan heran, saat anda install ulang komputer (bahkan Windows 8.1 sekalipun) anda akan menemukan bahwa tidak ada OpenGL pada VGA driver anda. Cara terbaik adalah periksa Motherboard anda dan cari driver VGA yang sesuai dengan Motherboard anda. Biasanya gratis.

Animation, Transition, Visual Effect, Shader Effect dan lain – lain merupakan fasilitas yang dapat anda kembangkan saat menggunakan QtQuick sebagai User Interface (UI) aplikasi anda.

b) Teknologi User Interface QtWidget

QtWidget merupakan tradisional User Interface element yang biasanya terdapat dalam dekstop environment. Bila anda pengguna linux, maka UI ini merupakan bagian dari KDE. Tapi jangan salah, QtWidget sangat dinamis untuk Windows dan Mac OS. Sehingga bila anda menggunakan QtWidget sebagai UI anda maka tampilannya mirip sekali dengan UI pada OS anda.

Bila anda pengguna Windows 8.1 seperti saya dengan Amazing Flat Designnya, maka QtWidget ini menyesuaikan dengan UI OS.

Semua standar komponen untuk aplikasi seperti button, textarea, menu dan lain – lain terdapat pada QtWidget ini. Sehingga sangat cocok sekali untuk anda yang gemar membuat aplikasi tradisional standar.

Bila kita membuat aplikasi dengan QtWidget, maka saat memulai project, akan muncul pemilihan Base Class, ada tiga yaitu Class QWidget, Class QMainWindow, dan Class Qdialog.

Perbedaan dari ketiga Base Class di atas adalah berikut ini:

- `*QWidget` merupakan base class untuk semua GUI element pada QtWidget User Interface. Coba lah explorasi dan bedakan ketiganya :)
- `*QDialog` merupakan sebuah window yang biasanya digunakan untuk mengejutkan pengguna, seperti saat window dialog muncul ketika pengguna harus memasukan input dengan benar atau hal – hal yang lain, tampilan dari Qdialog tidak berbeda dengan Qwidget, anda bisa menggunakan salah satu.
- `*QMainWindow`, nah, ini adalah sebuah class yang sangat unik, karena menggunakan feature *built in* yang sangat populer seperti status bar, toolbar, dan menu bar. Cobalah membuat applikasi QtWidget dengan QmainWindow sebagai base class nya, pasti secara otomatis akan ditampilan menubar , toolbar, dan statusbar.

c) Teknologi User Interface QtWebkit

Tahukah anda bahwa web programing adalah kegiatan yang paling berkembang di dunia saat ini? Tahukah anda bahwa web koding seperti html, css, js sangat populer dan mudah digunakan dan mulai merambah ke teknologi desktop seperti Html5 dan Css3?

Lalu kenapa tidak digunakan dalam pemrograman desktop? Ya, dengan menggunakan User Interface QtWebkit ini anda bisa membuat sebuah desktop application dengan menggunakan koding web. Unik bukan?

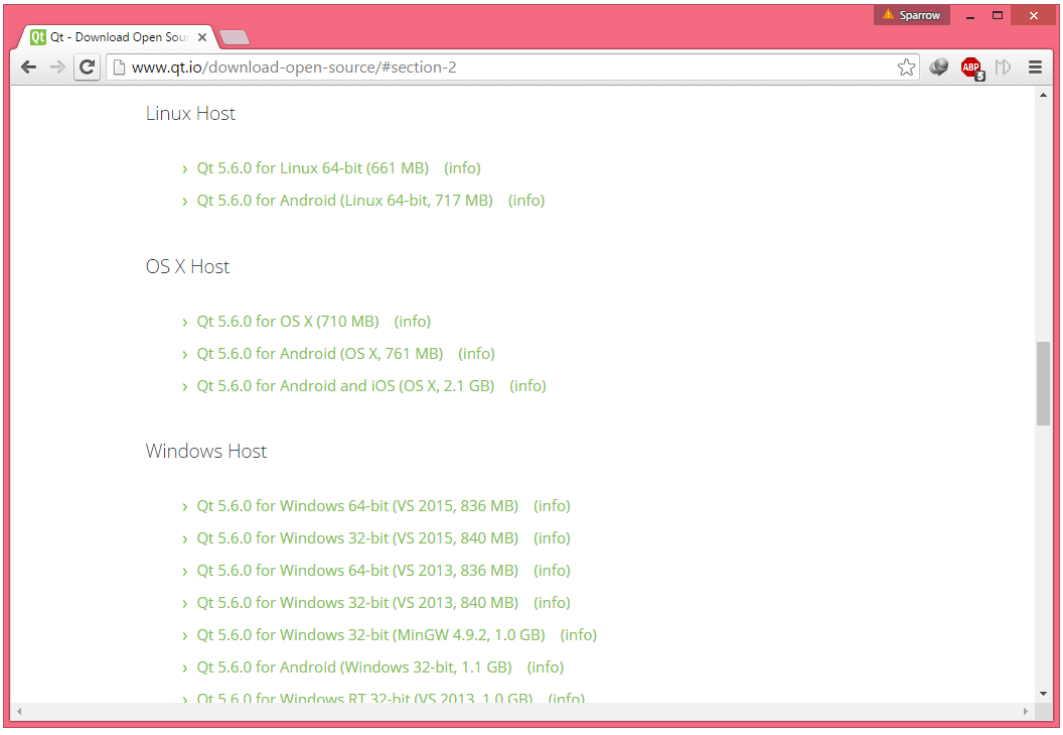
Teknologi QtWebkit menampilkan web content melalui QML, sedangkan C++ API digunakan untuk interaksi dengan web content tersebut.

Perlu diperhatikan bersama bahwa pemilihan teknologi adalah biasa, so, tetaplah berkreasi, berikut kita kutipkan beberapa perbandingan antar tiga teknologi UI dari Qt Help Documentation.

1.4 Install Qt Creator

Pada tutorial ini kita akan menginstall Qt pada ubuntu 14.04 atau Windows 8 dengan menggunakan versi terbaru dari Qt Creator yang dapat di unduh di halaman[16] Qt cCreator.

[16] http://www.qt.io/download



Halaman web Downlaod Qt Creator

A. Install Qt Creator di Ubuntu 14.04

1. Download

Kunjungi website Qt untuk mendowload Qt Crator sesui dengan versi sisem operasi yang di gunakan baik itu 64-bit atau 32 bit. atau juga dapat di download dengan menggunakan command line di linux dengan mengetikan.

Contoh:

```
1 wget http://download.qt.io/official_releases/online_installers/qt-unifi\
2 ed-linux-x86-online.run
```

jika menggunakan sistem operasi beberbasis 64 bit

```
wget http://download.qt.io/official_releases/qt/5.6/5.6.0/qt-opensource\
-linux-x64-5.6.0.run
```

2. Install

Atur permisi, jalankan installer dan ikuti perintah berikut ini untuk mnginstall Qt Creator secara lengkap.

```
chmod +x qt-opensource-linux-x64-5.6.0.run
./qt-opensource-linux-x64-5.6.0.run
```

3. Install g++

Buka terminal untuk menginstal g++:

```
sudo apt-get install build-essential
```

4. K0nfigurasi Kompiler

Buka Qt Creator klik tool > Options. Klik build & run dan pilih tab Kit. Konfigurasikan kompiler jka belum terdeteksi secara otomatis.

5. Install Pustaka OpenGL

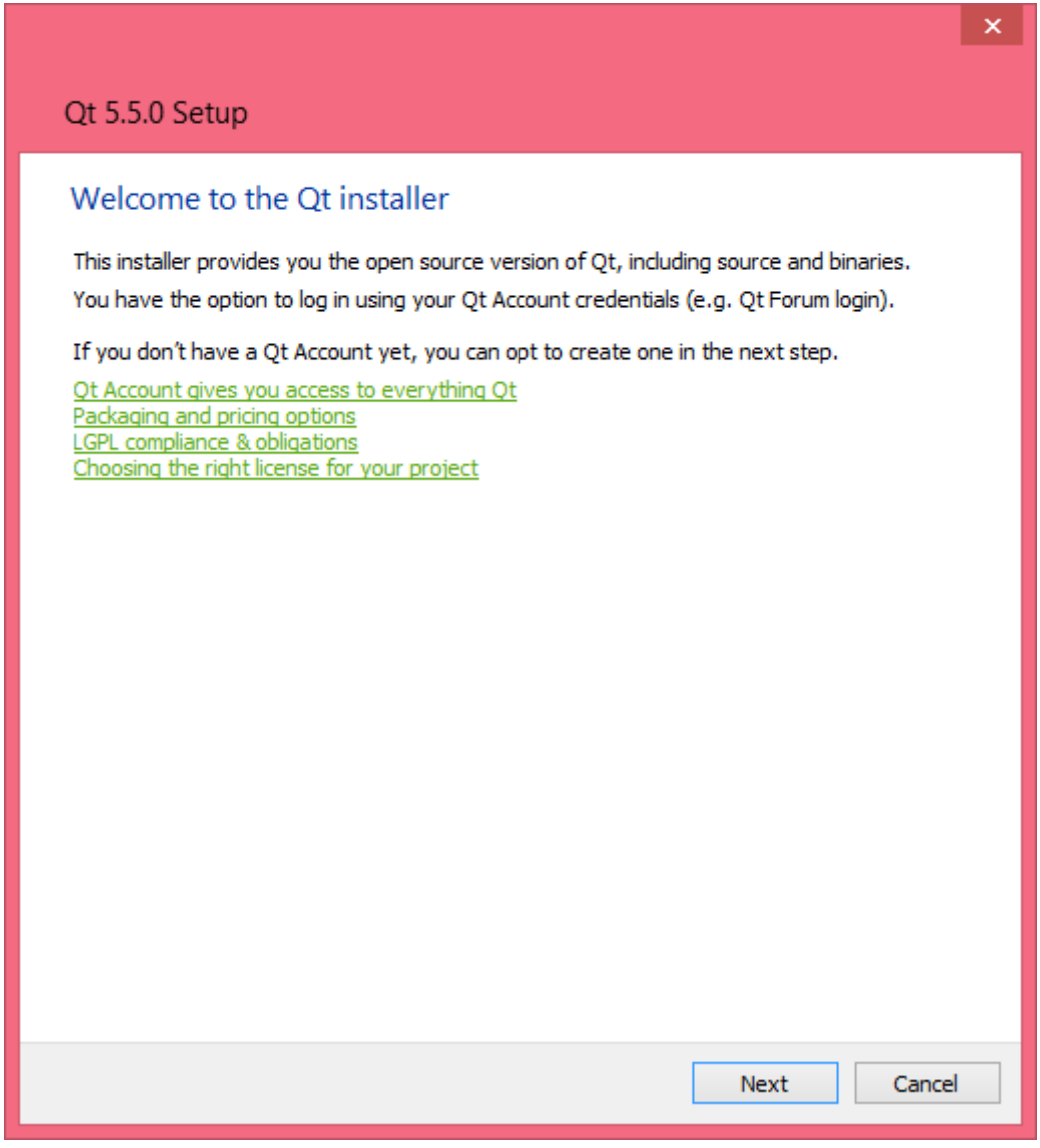
Jalankan Perintah berikut ini mengintall Pustaka OpenGL:

```
sudo apt-get install mesa-common-dev
```

B. Install Qt di Windows

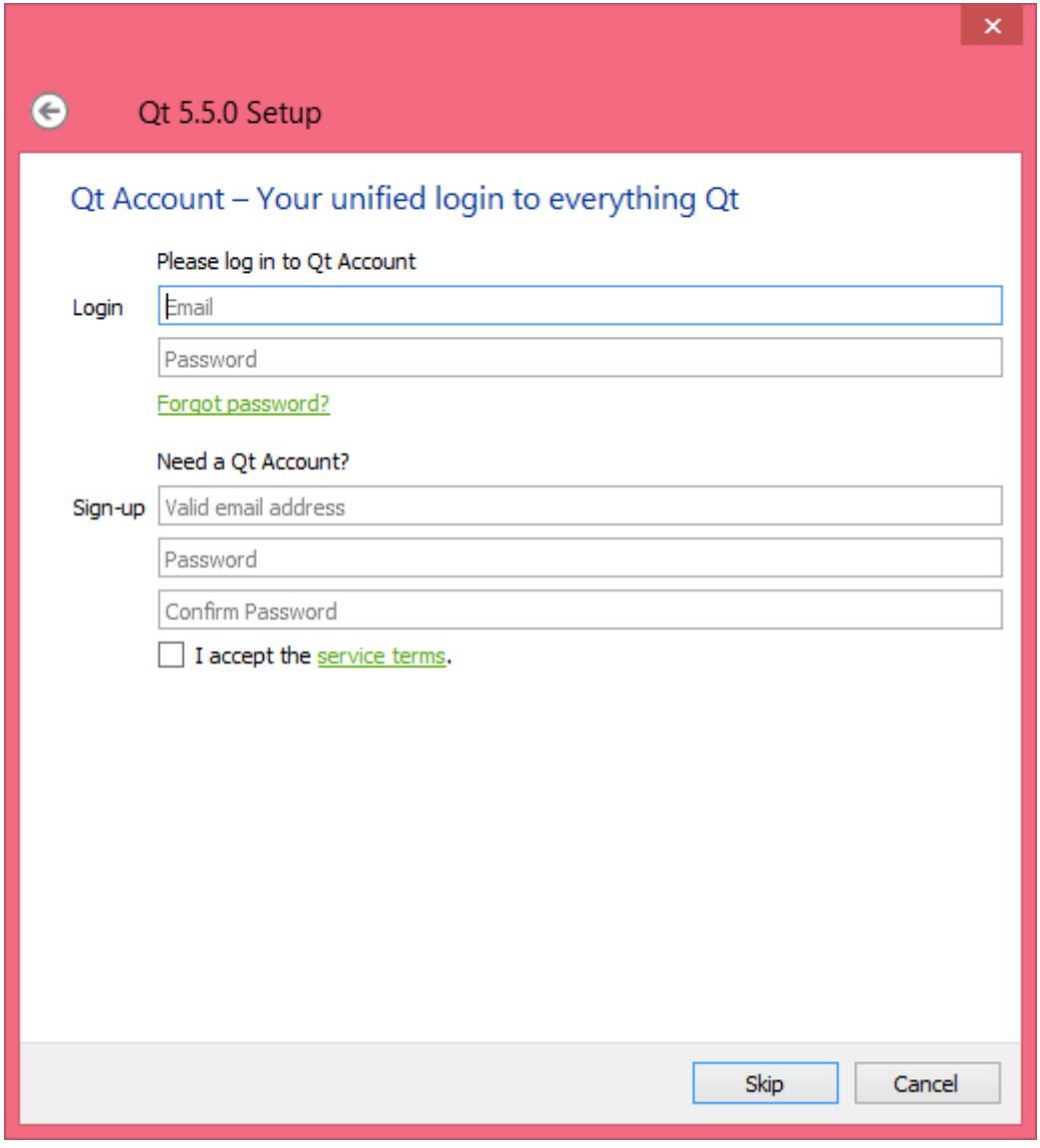
Anda bisa mendownload di halaman downlod Qt creator dan memlilih versi dari Aplikasi yang Anda butuhkan baik 64 bit atau 32 bit. Sesuikan dengan sistem operasi yang Anda miliki.

1. Jika Anda telah mendownlaod Qt creator maka qt-opensource-windows-x86-mingw492-5.5.0.exe. Disin penulis menggunakan versi 32 bit jika sistem oeprasi Anda 64 bit maka gunakanlah 64 bit walaupun dapat menggunakan versi 32 bit.

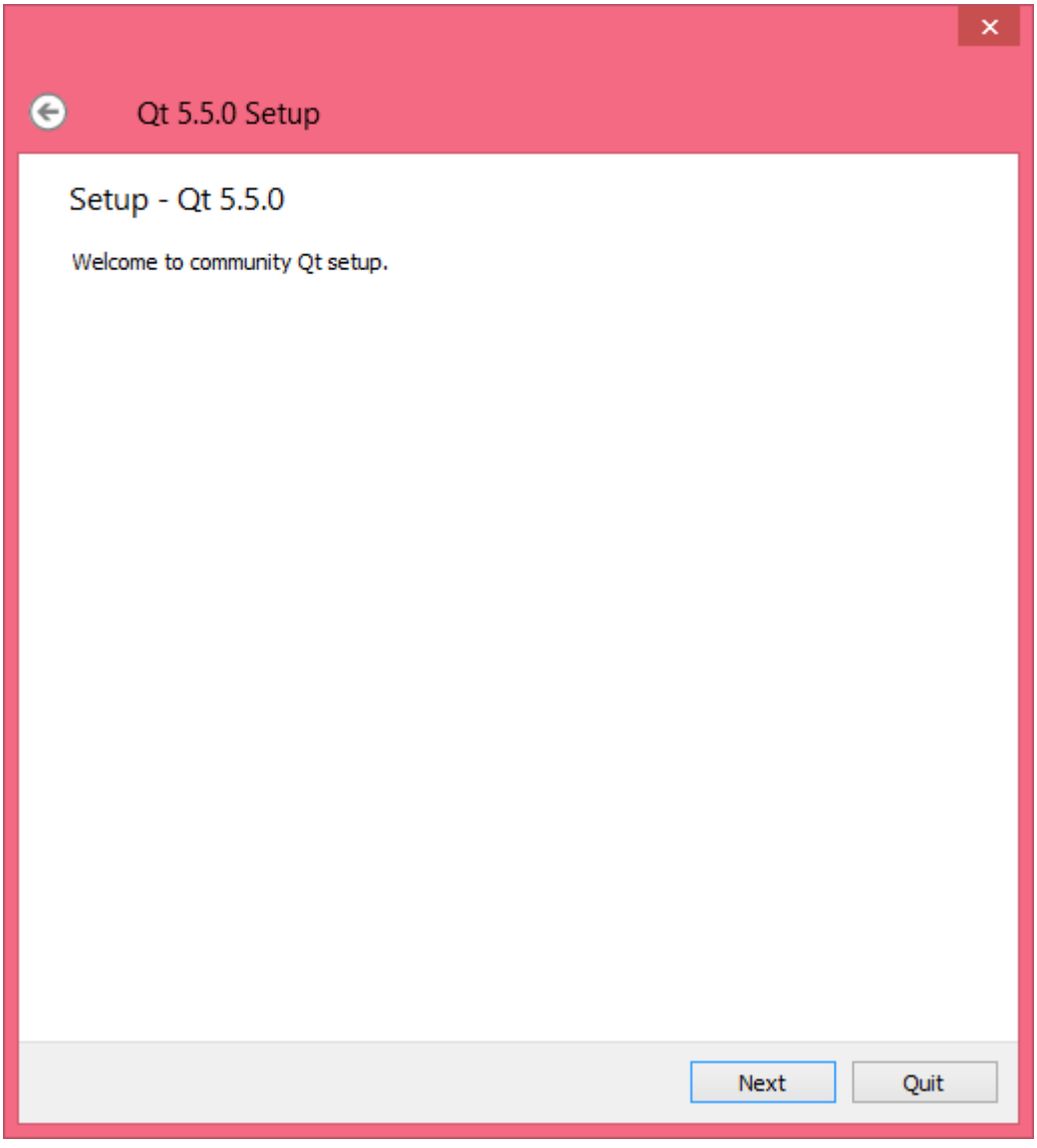


2. Pilih **Next** dan akan muncul halaman Qt Acount jika anda tidak ingin men-

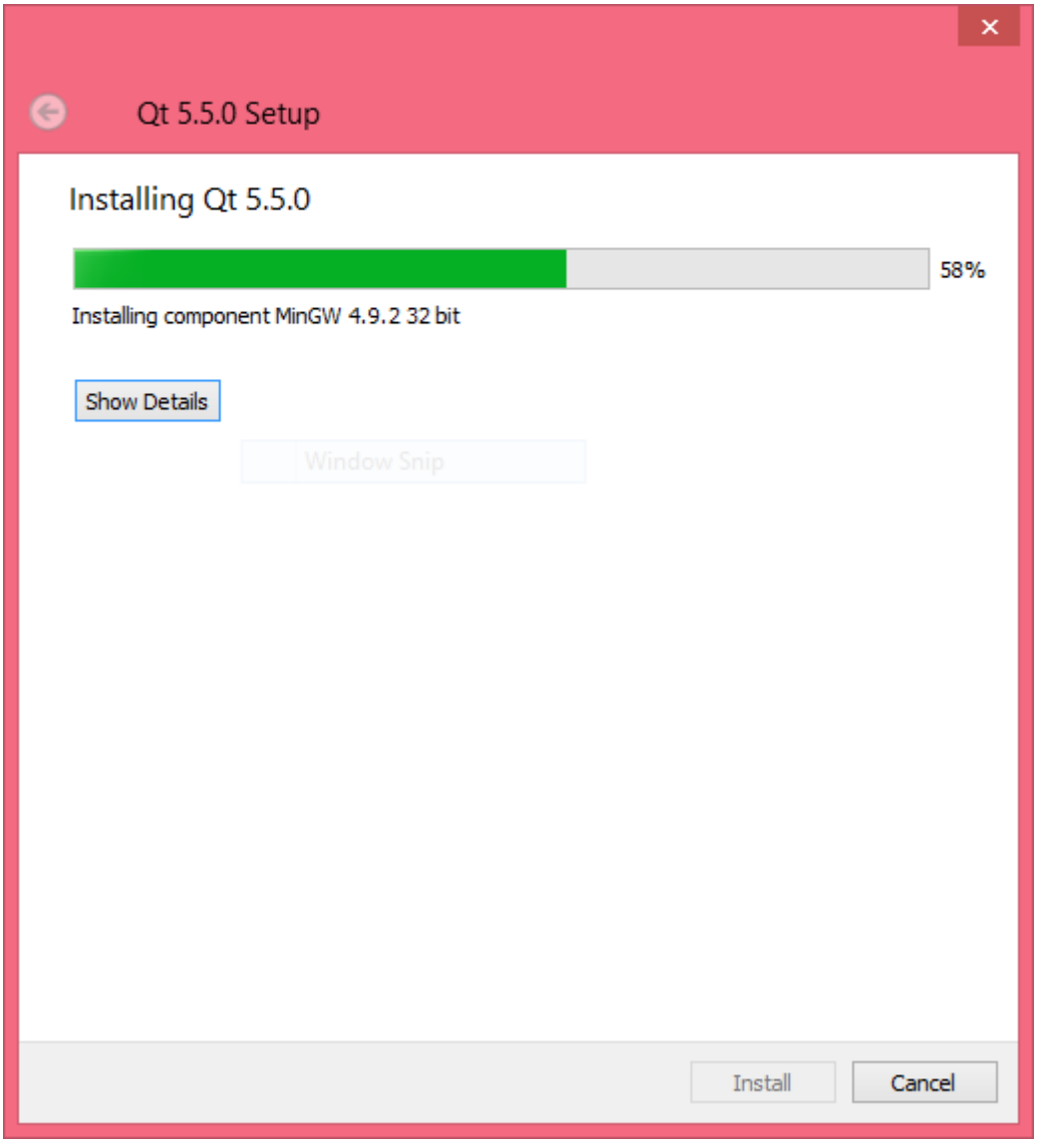
datarkan diri dapat di lewati dengan memilih **skip**.



3. Masuk ke halaman Setup terus **next** saja.

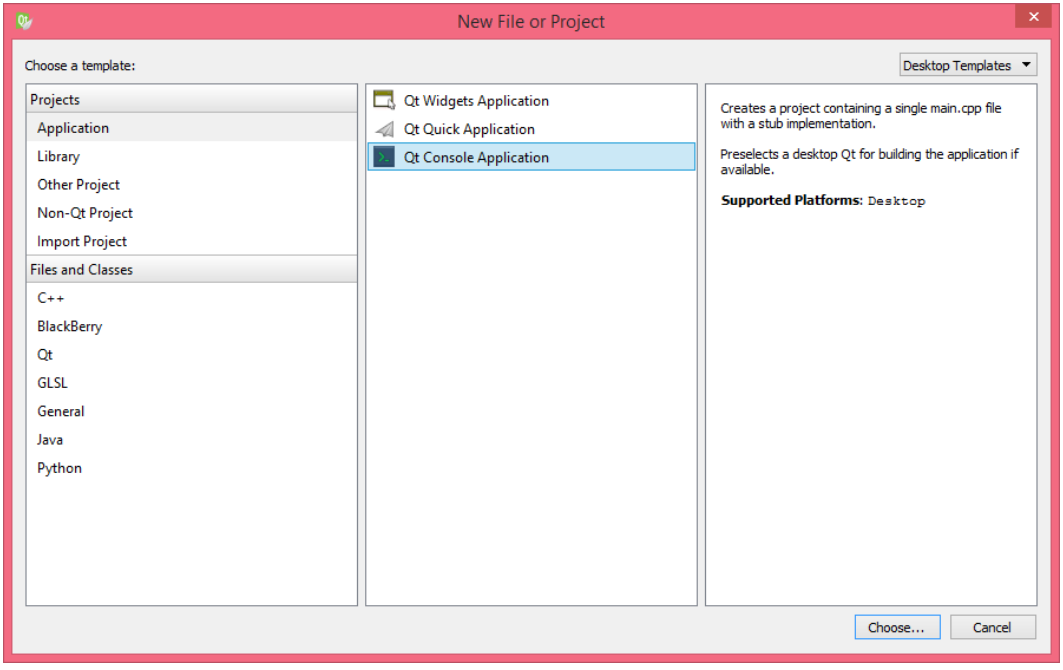


4. Installer akan menginstall aplikasi sampai selesai apabila telah selesai maka klik finish untuk mengakhiri proses pemasangan aplikasi.

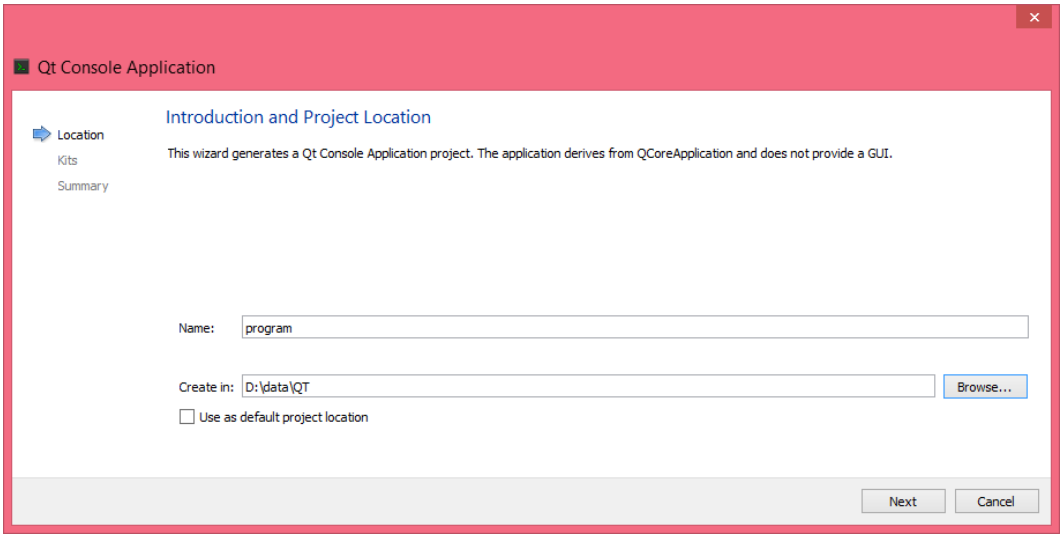

Qt 5.5.0 Setup
Installing Qt 5.5.0
58%
Installing component MinGW 4.9.2 32 bit
Show Details
Window Snip
Install
Cancel

1.5 Program Console Pertama dengan Qt Creator

1. Untuk mencoba membuat aplikasi dengan Qt Creator maka kita perlu dengan membuat menu file > new Project dan pilih project aplication > Qt console aplication



2. kemudian beri nama dengan Program yang akan kita buat dan direktori tempat aplikasi yang kita buat.

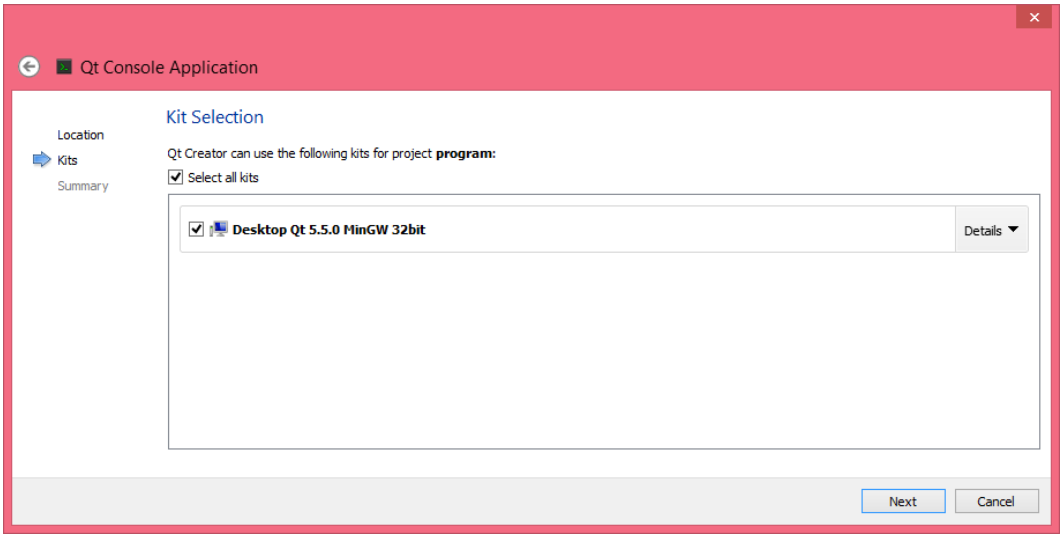


3. Klik Next, kemudian pilih compiler yang akan kita gunakan. Disini penulis menggunakan MinGW sebagai compilernya.

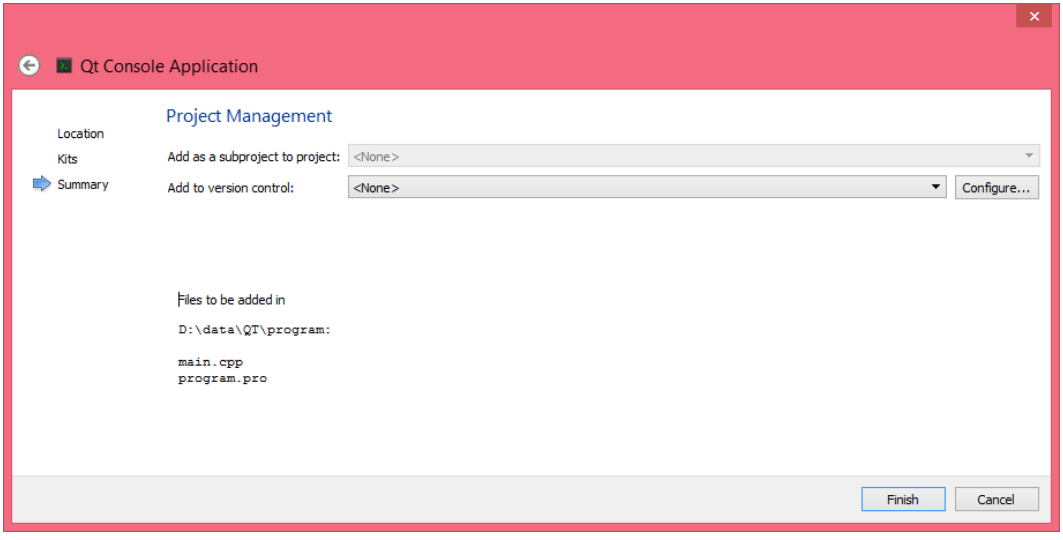
Tips

Simulator atau compiler yang lengkap teridir dari

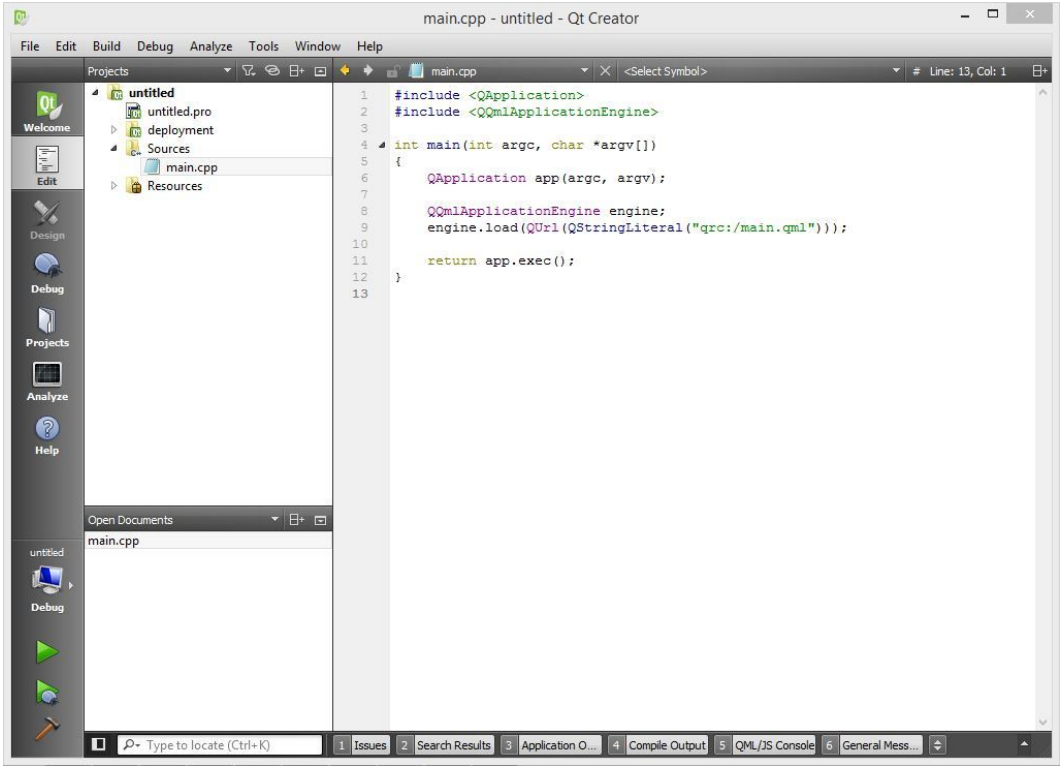
- Qt Simulator MingGW 4.4
- Qt Simulator VS 2008, 2010, 2011, 2012 2013, 2014
- Android SDK dan NDK



1. Kemudian pilih jenis sub version yang akan kita gunakan, jika Anda tidak mengunakan sub version maka pilih none pada add to subversion.



1. Apabila di lakukan dengan benar maka akan muncul Qt Editor sebagai berikut ini.



1.6 Struktur Program C++

Program Bahasa C/C++ tidak mengenal aturan penulisan di kolom/baris tertentu, jadi bisa dimulai dari kolom/baris manapun. Namun demikian, untuk mempermudah pembacaan program dan untukkeperluan dokumentasi, sebaiknya penulisan program di bahasa C/C++ diatur sedemikian rupa sehingga mudah dan enak dibaca.

Berikut adalah struktur dasar program yang dibuat dengan bahasa C++:

```
#include <header>
using namespace std;
int main(int argc, char *argv[])
{
        deklarasi variabel;
        deklarasi konstanta;
        perintah âperintah;
//komentar
        return 0;
}
```

Penjelasan :

1. #include <header>

`#include` adalah salah satu pengarah preprocessor directive yang tersedia pada C++. Preprocessor selalu dijalankan terlebih dahulu pada saat proses kompilasi terjadi. Bentuk umumnya:

```
# include <nama_file>
```

Bagian tersebut tidak diakhiri dengan tanda semicolon, karena bentuk tersebut bukanlah suatu bentuk pernyataan, tetapi merupakan preprocessor directive. Baris tersebut menginstruksikan kepada kompiler untuk menyisipkan file lain dalam hal ini file yang berakhiran .h (file header) yaitu file yang berisi C++ standard library. Pada C++ ekstensi .h tidak dituliskan.

Beberap contoh pengikutsertaan berkas adalah:

- `#include <iostream>` : diperlukan pada program yang melibatkan objek `cout` dan `cin`
- `#include <conio>`: diperlukan bila melibatkan `clrscr()`, yaitu perintah untuk membersihkan layar dan fungsi `getch()` untuk menerima sembarang input keyboard dari user.
- `#include <iomanip>` : diperlukan bila melibatkan `setw()` yang bermanfaat untuk mengatur lebar dari suatu tampilan data.
- `#include <math>` : diperlukan pada program yang menggunakan operasi `sqrt()` yang bermanfaat untuk operasi matematika kuadrat.

2. using namespace std;

Semua elemen standard C++ library dinyatakan dalam apa yang disebut namespace, namespace tersebut bernama std. Jadi artinya untuk mengakses semua fungsionalitas std kita menuliskan bahwa kita menggunakan namespace std.

3. int main ()

Program C++ terdiri dari satu atau lebih fungsi, dan di antara salah satunya harus ada fungsi main dan hanya boleh ada satu main pada tiap program C++. Setiap program C++ akan dan pasti akan memulai eksekusi programnya pada fungsi main ini, meskipun main bukan fungsi yang pertama ditulis di program. Melihat bentuk seperti itu dapat kita ambil kesimpulan bahwa batang tubuh program utama berada didalam fungsi main(). Berarti dalam setiap pembuatan program utama, maka dapat dipastikan seorangpemrogram menggunakan minimal sebuah fungsi.

Tanda { dan pada akhir program terdapat tanda }. Tanda { harus ada pada setiap awal dari sebuah fungsi dan tentu saja harus diakhiri dengan tanda }. Tanda ini digunakan untuk menunjukkan cakupan(scope) dari sebuah fungsi,dimana untukmenunjukkan fungsi ini dimulai danberakhir.

4. Komentar

Komentar tidak pernah dicompile oleh compiler. Dalam C++ terdapat 2 jenis komentar, yaitu:

1. /* Komentar anda diletakkan di dalam ini bisa mengapit lebih dari satu baris */
2. // Komentar anda diletakkan disini (hanya bisa sebaris)

Programmer sering sekali memasukkan komentar di dalam code agar program lebih mudah dibaca. Komentar juga membantu orang lain untuk membaca dan mengerti isi dari code. Komentar tidak menyebabkan komputer melakukan suatu instruksi ketika program dijalankan.

5. Tanda Semicolon (;)

Tanda semicolon " ; " digunakan untuk mengakhiri sebuah pernyataan. Setiap pernyataan harus diakhiri dengan sebuah tanda semicolon.

9. return 0

Pernyataan return menyebabkan fungsi utama untuk menyelesaikan kegiatannya lalu mengembalikanhasil dari fungsi utama. Kode kembalian biasanya angka 0 atau 1. Jika angka yang dikembalikan 0 berartiprogram berakhir dengan tidak ada error, sedangkan jika 1 maka program berakhir dengan error.

Contoh 1. Structur program C++

Untuk lebih jelasnya silahkan coba ketik program berikut pada project baru.

```
1  #include <QtCore/QCoreApplication>
2  #include <iostream>
3
4  using namespace std;
5
6  int main (int argc, char *argv [])
7  {
8          QCoreApplication a (argc, argv);
9          cout<<"Hello World"<<endl;
10         cout<<"Selamat Belajar C/C++ ";
11         cout<<"enter my World";
12         return a.exec ();
13 }
```

Kemudian jalankan dengan menekan tombol Run (CTRL + R)

```
Hello world
Selamat belajar C/C++ enter my world
```

Tampilan Hello World diakhiri dengan tanda enter baru kemudian dilanjutkan dengan tulisan berikutnya yaitu Selamat Belajar C/C++ enter my World. Artinya perintah `endl` merupakan perintah untuk memberi tanda enter. Sedangkan untuk tulisan Selamat Belajar C/C++ dan tulisan enter my World yang pada source code terpisah dengan perintah `cout`, pada tampilan hasil program tetap sama dan tidak ada enter diantaranya. Hal ini karena tidak ada perintah untuk menampilkan enter

diantara kedua kalimat tersebut. Penulisan pada kode tidak akan mempengaruhi hasil output program.

2. Tipe Data, Identifier, Operator dan Control Statement

2.1 Tipe Data dan Identifier

Program adalah kumpulan instruksi yang disusun sedemikian rupa sehingga mempunyai urutan nalar yang tepat untuk menyelesaikan suatu persoalan. Instruksi-instruksi yang digunakan dalam pemrograman mengacu pada suatu bahasa pemrograman tertentu, pada buku ini menggunakan bahasa pemrograman C++, sehingga penulisan program pada buku ini mengikuti tata bahasa C++.

Segala sesuatu yang diproses oleh program adalah data. Dalam hal ini data adalah elemen-elemen yang digunakan untuk menjelaskan segala sesuatu yang mempunyai besaran (ukuran/ nilai), seperti misalnya **umur** besarannya bisa berupa biangan desimal **42.5** (maksudnya 42½ tahun), **golongan** seorang karyawan besarannya bisa berupa sebuah karakter A (maksudnya goongan A) dan sebagainya. Bahasa C++ menyimpan besaran-besaran tersebut di memori utama untuk dikelola oleh program, sehingga perlu dilakukan pengaturan pemakaian memori, oleh karena itu dalam bahasa pemrograman selalu terdapat istilah-istilah yang bernama **Tipe Data**, **Variabel** dan **Konstanta**.

Identifier (pengenal) adalah suatu nama yang digunakan program untuk merujuk ke suatu lokasi memori tertentu agar nilai pada lokasi tersebut dapat diakses. Alamat lokasi memori sebenarnya berupa angka angka heksadesimal[1], namun pada bahasa pemrograman setingkat C++ (middle level programming language) dan di atasnya, telah mengubahnya dalam bentuk identifier (pengenal) yaitu berupa suatu huruf atau kata (label) sehingga kita tidak perlu mengetahu alamat yang sesungguhnya dan dengan identifier (label) akan lebih mudah untuk diingat.

1

2.2 Tipe Data Bahasa C++

Data yang dapat dikelola oleh program bisa bermacam-macam, seperti misalnya bilangan bulat (*integer*), bilangan dengan desimal (*floating point*), huruf (*character*), dan sebagainya. Oleh sebab itu ketika kita akan memakai suatu lokasi memori tertentu untuk menyimpan nilai diperlukan 2 hal, yaitu `identifier` sebagai pengenal (label) lokasi memori yang digunakan dan `tipe data`, yaitu besaran yang menentukan ukuran memori yang dialokasikan. Sekali suatu identifier sudah dialokasikan dengan tipe data tertentu besarnya ruang yang digunakan tidak bisa diubah. Bahasa C++ mengenal tipe-tipe data berikut ini :

Tipe Data	Ukuran	Jangkauan Nilai Yang dapat Ditampung
bool	1 byte	True or false
unsigned short int	2 bytes	0 to 65,535
short int	2 bytes	−32,768 to 32,767
unsigned long int	4 bytes	0 to 4,294,967,295
long int	4 bytes	−2,147,483,648 to 2,147,483,647
int (16 bit)	2 bytes	−32,768 to 32,767
int (32 bit)	4 bytes	−2,147,483,648 to 2,147,483,647
unsigned int (16 bit)	2 bytes	0 to 65,535
unsigned int (32 bit)	4 bytes	0 to 4,294,967,295
char	1 byte	256 character values
float	4 bytes	1.2e−38 to 3.4e38
double	8 bytes	2.2e−308 to 1.8e308

2.3 Variabel dan Konstanta

Nilai yang tersimpan di memori dan dikenal melalui identifier tersebut terdiri dari variabel dan konstanta. Perbedaan diantara keduanya adalah bahwa variabel (sesuai dengan namanya) nilainya dapat diubah-ubah pada saat program dieksekusi, sedangkan konstanta nilainya tidak dapat diubah (`konstan = tetap`).

Sebelum suatu variabel atau konstanta dapat digunakan, tempat pada memori harus dipesan terlebih dahulu, mekanisme ini dinamalan deklarasi. Deklarasi dilakukan dengan cara menuliskan tipe data (ukuran memori yang dibutuhkan) dan

diikuti dengan nama pengenal (nama variabel), jika dikehendaki bisa juga suatu variabel langsung diinisialisasi dengan suatu nilai. Pengenal (identifier) bisa terdiri dari sebuah huruf atau kombinasi antara huruf dengan angka dengan syarat.

- Harus diawali dengan huruf
- Tidak boleh memakai karakter khusus kecuali $ dan garis bawah (_)
- Tidak boleh sama dengan kata kunci yang digunakan pada C++
- Bersifat case sensitif (huruf besar dan kecil dibedakan)

Walaupun demikian, sebaiknya memberikan nama pengenal variabel sesuai dengan isi dari variabel tersebut, sebab walaupun nama variabel "**c21i8k**" untuk menyimpan nama mahasiswa adalah valid (diperbolehkan), namun akan lebih mudah dimengerti jika identifier yang dipilih adalah "**nama**".

Konstanta mirip dengan variabel, hanya saja nilainya konstan, tidak dapat diubah-ubah. Untuk dapat membuat konstanta diperlukan inisialisasi ketika konstanta dibuat dan setelah itu nilainya tidak dapat diubah. C++ mempunyai 2 macam konstanta, yaitu konstanta literal dan konstanta simbolik. Berikut ini adalah contoh deklarasi variabel:

```
1 int harga;
```

Yang dimaksud dengan konstanta literal adalah suatu nilai yang ditulis pada kode program. Sebagai contoh misalnya :

```
1 int usiaku = 42;
```

Nilai 42 tidak dapat menerima nilai lain dan nilai tersebut bersifat tetap. Perhatikan dalam hal ini identifier "usiaku" adalah variabel (bukan konstanta), yang dinamakan konstanta literal adalah nilai "42" tersebut.

Konstanta simbolik adalah konstanta yang direpresentasikan dengan suatu nama, sama seperti variabel, namun berbeda dengan variabel setelah suatu konstanta diinisialisasi dengan suatu nilai maka nilainya tidak dapat diubah. Ada 2 cara untuk mendeklarasikan konstanta simbolik, yaitu dengan menggunakan preprocessor directive `#define` dan yang kedua adalah dengan memakai kata kunci `const`. Berikut ini contoh mendeklarasikan dan menginisialisasi konstanta :

```
1 #define kapasitas 15
```

Perhatikan bahwa `kapasitas` tidak mempunyai tipe data tertentu (int, char dsb.). Preprosessor akan melakukan substitusi berupa teks, setiap ada akses terhadap kata `kapasitas`, akan digantikan dengan teks 15. Karena preprosesor bekerja sebelum kompiler, kompiler tidak mengenal konstanta `kapasitas`, yang dikenal hanyalah bilangan 15.

TIPS

Walaupun dengan memakai preprocessor directive `#define` tampak mudah, namun sebaiknya cara ini tidak digunakan, karena sudah dinyatakan usang pada standard C++ .

Cara yang kedua untuk menginisialisasi sebuah konstanta adalah dengan memakai kata kunci const seperti berikut :

```
1 const int usiaku = 42;
```

Contoh diatas adalah mendeklarasikan konstanta simbolik bernama usiaku bertipe int dan diinisialisasi dengan nilai 42. Setelah baris ini simbol (identifier) bernama usiaku tidak dapat diubah-ubah nilainya. Keuntungan pembuatan konstanta dengan cara ini adalah lebih mudah dipelihara dan mencegah adanya kesalahan dan yang paling penting adalah bahwa konstanta ini mempunyai tipe data dan kompiler dapat mengharuskan konstanta ini diperlakukan sebagai tipe data tersebut.

Contoh 1. Tipe data dan Identifier.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 1, kemudian tulis kode berikut.

```
#include <iostream>
int main(int argc, char *argv[])
{
    using namespace std;
    QCoreApplication a(argc, argv);

    int panjang, lebar;

    panjang = 15; //<-- nilai diubah menjadi 15
    lebar = 12; //<-- nilai diubah menjadi 12
    cout << "Panjang = " << panjang << endl;

    cout << "Lebar = " << lebar << endl;
    return a.exec();
}
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, outputnya adalah sebagai berikut.

```
Panjang                =15
lebar                 =12
```

Keterangan:

- Pada program di atas variabel panjang dan lebar dideklarasikan bertipe int.
- Kemudian variabel panjang diberi nilai 15 (integer) dan lebar diberi nilai 12 (integer), tampak bahwa nilai dari variabel tersebut dapat diubah.
- Pada baris berikutnya nilai dari variabel dapat diakses untuk dicetak ke layar.

2.4 Statement

Dalam bahasa C++, sebuah statement mengontrol urutan pengerjaan eksekusi, mengevaluasi ekspresi atau tidak mengejakan apapun (*null statement*). Semua statement C++ diakhiri dengan titik koma (;), sebagai contoh misalnya :

```
1  x = a + b;
```

Pernyataan tersebut bukanlah suatu pernyataan persamaan aljabar dalam matematika yang artinya x sama dengan a + b, melainkan memberi nilai x dengan hasil penjumlahan a dengan b. Pada statement ini terjadi 2 urutan pengerjaan, yaitu pertama menambahkan a dengan b, kemudian yang kedua memberikan hasil perhitungan tersebut ke variabel x dengan operator pengerjaan (=). Walaupun pada pernyataan tersebut terdapat 2 pekerjaan, namun merupakan sebuah statement dan oleh karena itu diakhiri hanya dengan sebuah titik koma (;) saja. Hasil penjumlahan a dengan b ini disebut ekspresi, sedangkan sama dengan (=) dan plus (+) dinamakan operator yang akan dibahas berikut ini.

CATATAN

Operator pengerjaan "=" akan mengambil nilai apapun yang ada disebelah kanannya kenudian memberikannya kepada apapun yang berada di sebelah kirinya. C++ mengenal juga operator pembanding "==" yang mempunyai arti berbeda dengan operator sama dengan "=", akan dibahas lebih detail pada sub bab berikut ini.

2.5 Operator dan Ekspresi

Operator adalah suatu simbol yang digunakan untuk melakukan suatu operasi. Operator mempunyai beberapa kategori, antara lain : Aritmatika, Pengerjaan, Hubungan dan Logika. Operator Aritmatika adalah operator yang digunakan untuk melakukan operasi aritmatika seperti misalnya penjumlahan, pengurangan, perkalian dan pembagian. Simbol untuk operator aritmatika ini adalah : +, -, *, / dan %. Berikut ini adalah operator-operator yang dikenal pada bahasa pemrograman C++.

Kategori	Operator	Arah Proses	Jenjang
Kurung, indeks larik dan elemen struktur data	() [] . ->	Kiri - Kanan	1
Operator Unary	! ~ - ++ –	Kanan – Kiri	2
Operator Aritmatika Perkalian, Pembagian dan Sisa Pembagian	* / %	Kiri – Kanan	3
Operator aritmatika Pertambahan dan Pengurangan	+ -	Kiri – Kanan	4
Operator Bitwise Pergeseran Bit	<< >>	Kiri – Kanan	5
Operator Hubungan	< <= > >=	Kiri – Kanan	6
Operator Hubungan Kesamaan dan Ketidaksamaan	== !=	Kiri – Kanan	7
Operator Bitwise AND	&	Kiri – Kanan	8
Operator Bitwise XOR	^	Kiri – Kanan	9
Operator Bitwise OR	|	Kiri – Kanan	10
Operator Kondisi AND	&&	Kiri – Kanan	11
Operator Kondisi OR	||	Kiri – Kanan	12
Operator Ternary ?	|	Kanan – Kiri	13
Operator Pengerjaan Aritmatika	= += -= *= /= %=	Kanan – Kiri	14
Operator Pengerjaan Bitwise	&= ^= |= <<= >>=	Kanan – Kiri	15
Operator Koma	,	Kiri – Kanan	16

Ekspresi adalah suatu peryataan yang menghasilkan suatu nilai, bisa berasal dari sebuah variabel maupun kumpulan variabel-variabel yang dioperasikan dengan suatu operator, jadi hasil akhir dari suatu ekspresi adalah suatu nilai yang mempunyai besaran dan tipe data tertentu. Pernyataan berikut ini yang disebut ekspresi adalah 15, 12 dan "panjang * lebar" yang menghasilkan nilai 15, 12 dan 180:

```
1  panjang = 15;
2  lebar = 12;
3  luas = panjang * lebar ;
```

Keterangan :

- Pada baris pertama dan kedua di atas digunakan hanya sebuah operator "= " (yaitu jenjang ke 14), arah proses dari kanan ke kiri, sehingga yang dilakukan :
- Ekspresi : `15`, diberikan kepada variabel `panjang` (dibaca dari kanan ke kiri).
- Ekspresi : `12`, diberikan kepada variabel `lebar` (dibaca dari kanan ke kiri).
- Pada baris ketiga terdapat 2 operator, yaitu operator "= " (jenjang ke 14) dan "*" operator "= " (yaitu jenjang ke 3). Jenjang menunjukkan operator yang akan dikerjakan terlebih dahulu, jika dalam sebuah ungkapan terdapat lebih dari satu jenis operator. Jenjang nomor 1 adalah jenjang yang paling tinggi, maka pada pernyataan di atas yang akan dikerjakan terlebih dahulu adalah orator "*" baru kemudian operator "=", sehingga yang dilakukan: - Ekspresi : `panjang * lebar` , berarti `panjang` dikalikan `lebar` (dibaca dari kiri ke kanan), menghasilkan nilai integer `180`. - Berikutnya operator "=" mengoperasikan hasil ekspresi tersebut, yaitu nilai integer `180` diberikan kepada variabel `luas` (dibaca dari kanan ke kiri).

TIPS

Operator "(" dan ")" dapat dipakai untuk merubah jenjang suatu ekspresi menjadi jenjang tertinggi, sehingga akan diproses terlebih dahulu.

a) Operator Unary

Operator unary adalah operator yang hanya menggunakan sebuah operand saja, operator unary yang dipakai pada kebanyakan bahasa pemrograman adalah operator unary minus (-). Operator unary ditulis sebelum operand, operator unary "-" berbeda dengan operator aritmatika "-" yang membutuhkan dua operand. Dalam bahasa C++ disediakan bermacam-macam operator unary.

Operator	Arti
-	Unary minus
++	Peningkatan dengan nilai penambahan 1
–	Penurunan dengan nilai pengurangan 1
!	Unary not
~	Operator unary komplemen satu (bitwise NOT)

b) Operator Unary Minus

Operator ini dipakai untuk memberi nilai minus suatu nilai numerik (bukan pengurangan). Misalnya ungkapan : `A + - B * C` akan diartikan `A + (-B) * C`. Operator unary "-" ditulis di depan operand.

c) Operator Unary ++ dan –

Operator unary "++" dan "–" merupakan operator khusus yang ada di bahasa C. Operator "++" akan menambahkan nilai 1 ke pengenal yang menggunakannya sedangkan operator "–" akan mengurangi dengan nilai numerik 1. Operator unary tersebut jika dituliskan sebelum operand disebut *pre increment* sedangkan jika ditulis setelah operand disebut *post increment*. Perhatikan perbedaannya pada contoh dibawah ini :

Post Increment	**Pre Increment**
x = 5; a = x++;	x = 5; a = ++x;
Hasil: x = 6 dan a = 5	**Hasil**: x = 6 dan a = 6

d) Operator Pengerjaan

Operator pengerjaan atau disebut assignment operator, digunakan untuk menempatkan nilai dari suatu ekspresi ke suatu pengenal. Operator yang umum dipakai pada bahasa pemrograman adalah operator pengerjaan "=". Selain operator pengerjaan "=", bahasa C++ menyediakan beberapa operator pengerjaan yang lain seperti tabel di bawah ini.

Operator	**Contoh**	**Maksud/ Ekuivalen dengan**
=	a = b + c	Mengerjakan b+c ke a
+=	a += 1	a = a + 1
-=	a -= b	a = a – b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b

Tabel berikut ini memberikan contoh pemakaian operator-operator di atas, misalnya variabel a dan b bernilai 10.

Statement	Ekuivalen dengan	Hasil Ungkapan
a += 3	a = a + 3	a = 10 + 3 = 13
a -= 2	a = a - 2	a = 10 − 2 = 8
a *= b/2	a = a * (b/2)	a = 10 * (10/2) = 50
a /= b − 8	a = a / (j − 8)	a = 10 / (10-8) = 5

Dari contoh di atas terlihat bahwa operator pengerjaan mempunyai jenjang yang lebih rendah dibanding operator aritmatika, sehingga operator aritmatika dikerjakan terlebih dahulu.

C++ mengijinkan operator pengerjaan ditulis lebih dari satu kali pada sebuah statement, misalnya :

```
1  x = y = a * b;
```

Dalam hal ini yang dikerjakan adalah a dikalikan b terlebih dahulu meudian hasilnya diberikan kepada variabel y dan hasil ekspresi y = a * b diberikan kepada variabel x. sehingga misalnya a bernilai 8 dan b bernilai 7, maka baik variabel x maupun y keduanya bernilai 15.

e) Operator Hubungan

Operator hubungan (*relational operator*) digunakan untuk menunjukkan hubungan antara dua buah operand, hasil dari operator ini adalah True atau False.

Operator	Jenjang	Arti
<	6	Lebih kecil dari
<=	6	Lebih kecil atau sama dengan
>	6	Lebih besar dari
>=	6	Lebih besar atau sama dengan
==	7	Sama dengan
!=	7	Tidak sama dengan

Berikut ini contoh hasil ekspresi jika a bernilai 5, b bernilai 7 dan c bernilai ‘a’

Ungkapan Hubungan	Hasil	Nilai
a == 5	Benar	1
a == b	Salah	0
b < 7	Salah	0
a <= 7	Benar	1
(a+b) != 35	Benar	1
c != 'A'	Benar	1
c <= 'z'	Benar	1

f) Operator Logika

Jika operator hubungan membandingkan hubungan antara dua buah operand, maka operator logika (*logical operator*) digunakan untuk menggabungkan logika hasil dari operator-operator hubungan. Operator logika menggabungkan **dua buah** nilai logika. Nilai logika adalah nilai benar (True) atau salah (False).

Operator	Jenjang	Arti
&&	11	Logika DAN (AND)
||	12	Logika ATAU (OR)

Selain dua operator logika ini, operator unary "!" (logika NOT) dapat digunakan untuk operasi logika.

x	y	x && y	x || y	!x
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE

Contoh : Misalnya A bernilai 5, B bernilai 7 dan C bernilai 'a' maka ungkapan dibawah ini mempunyai hasil akhir benar (True).

```
1  A < B || B == 7 && C > 'z'
```

Hasil akhir benar (True) dari ekspresi logika tersebut didapat dari langkah-langkah sebagai berikut:

1. Jenjang operator hubungan lebih tinggi dibandingkan dengan operator logika,

jadi operator hubungan dikerjakan terlebih dahulu.
2. Operator logika "&&" mempunyai jenjang lebih tinggi dari operator "||", sehingga operator "&&" dikerjakan terlebih dahulu.
3. Bagian yang paling akhir dikerjakan adalah operator "||", sehingga hasil akhir logika bernilai logika benar atau True.

2.6 Control Statement

Aliran program tidak selalu berjalan secara sekuensial berurutan dari atas ke bawah, kadang-kadang diperlukan **percabangan** atau **perulangan** atau kombinasi dari keduanya. Semua bahasa pemrograman mempunyai struktur kendali (*control statement*) demikian juga bahasa C++. Struktur kendali merupakan pengatur aliran program, mempunyai rangkaian perintah yang harus ditulis untuk memenuhi beberapa keadaan, yaitu:

- Mengulang suatu perintah jika suatu kondisi dipenuhi.
- Melanjutkan sebuah pernyataan bila kondisi terpenuhi.
- Memilih sebuah pilihan dari beberapa alternatif bila kondisi terpenuhi.

A. Percabangan

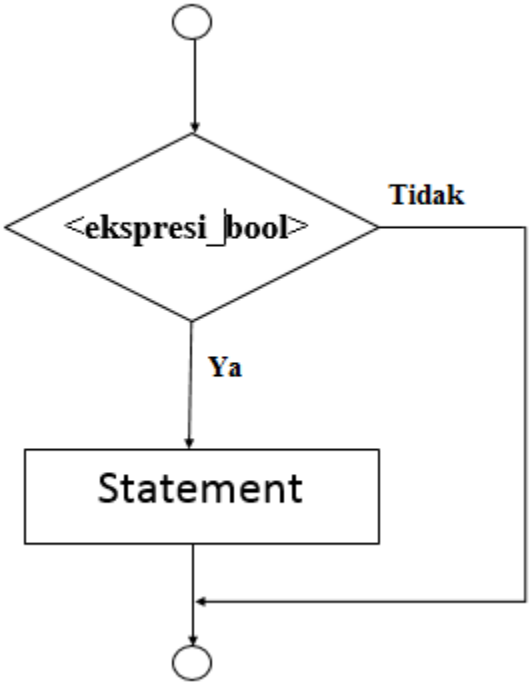
Adalah perintah yang memungkinkan pemilihan atas perintah yang akan dijalankan sesuai dengan kondisi tertentu. Ada tiga macam perintah percabangan dalam C++, yaitu `if`, `if … else`, dan `switch`. Dengan percabangan, suatu baris program akan dikerjakan jika suatu kondisi dipenuhi (benar) atau tidak (else), jadi tidak semua baris program akan dieksekusi.

1. Percabangan dengan if

Sintaks penulisannya sebagai berikut:

```
1  if (<ekspresi_boolean>)
2  {
3  <statements>
4  }
```

Flowchart untuk statment ini adalah :



2. Percabangan dengan if .. else

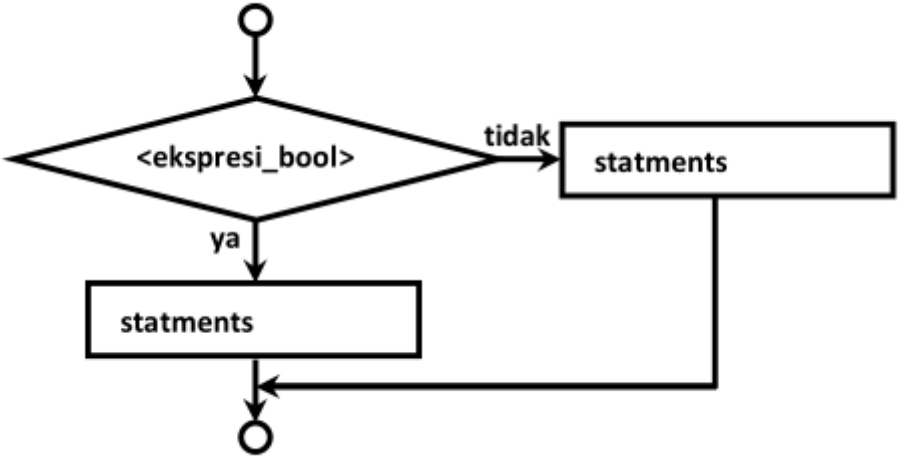
Sintaks penulisannya sebagai berikut :

```
 if (<ekspresi_boolean>)
{
        <dijalankan jika ekspresi_boolean benar>
}
 else
{
        < dijalankan jika ekspresi_boolean salah>
}
```

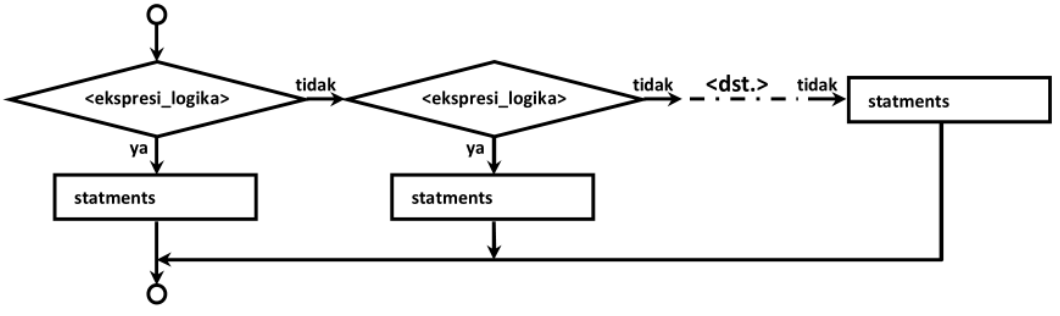
Flowchart untuk statment ini adalah :

CATATAN

Di dalam `if()` maupun di dalam else bisa diisi dengan perintah `if()` lagi. Bentuk `if()` dalam `if()` ini sering disebut dengan `nested if` (if bersarang).



Flowchart untuk statment if bersarang ini adalah :

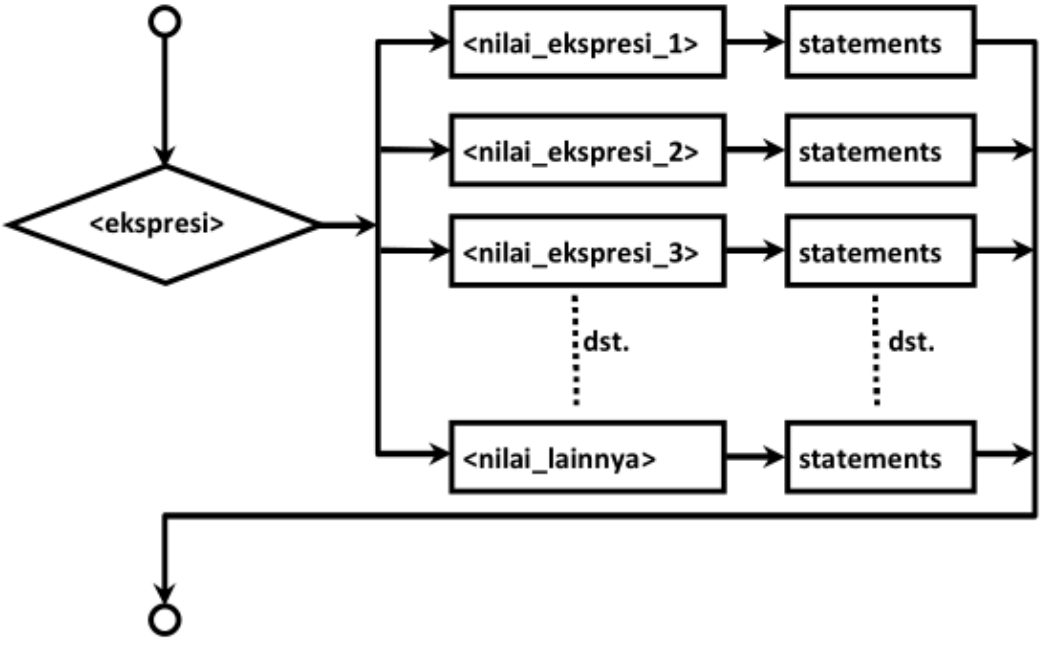


3. Percabangan dengan switch

Perintah ini digunakan sebagai alternatif pengganti dari statment `if … else` dengan `else` lebih dari satu. Dengan perintah ini percabangan dapat diarahkan pada beberapa alternatif pilihan berdasarkan nilai ekspresi. Berbeda dengan `if`, `switch` tidak dapat medeteksi *operator pembanding* (>, <, dsb.), karena ekspresi degan operator ini menghasilkan nilai *boolean*, melainkan hanya dapat mengalihkan alur program ke suatu nilai yang sama, pada statement ini ekspresi yang diminta harus menghasilkan bilangan *bulat*.

```
switch (<ekspresi>)
{
case <konst_1>: <pernyataan_1>;
        break;
case <konst_2>: <pernyataan_2>;
        break;
case <konst_n>: <pernyataan_n>;
        break;
default : <pernyataan_default>;
        }
```

Perintah `switch` akan membaca nilai dari `<ekspresi>` kemudian membandingkan hasilnya dengan konstanta-konstanta (`<konst_1>`, `<konst_2>`, `<konst_n>`) yang berada di case. Pembandingan akan dimulai dari `<konst_1>` sampai konstanta `<konst_n>`. Jika hasil dari kondisi sama dengan nilai konstanta tertentu, misalnya `<konst_1>`, maka pernyataan 1 akan dijalankan sampai ditemukan `break`. Pernyataan `break` akan membawa proses keluar dari perintah `switch`. Jika hasil dari kondisi tidak ada yang sama dengan konstanta-konstanta yang diberikan, maka pernyataan pada *default* yang akan dijalankan.



Flowchart untuk statement ini adalah :
Contoh.2 Tipe data dan Identifier.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 2, kemudian tulis kode berikut.

```
1       #include <QtCore/QCoreApplication>
2       #include <iostream>
3       int main(int argc, char *argv[])
4       {
5           using namespace std;
6           QCoreApplication a(argc, argv);
7
8           int hari = 6;
9
10          switch(hari){
11              case 1 : cout << "Senin" << endl;
```

```
12              break;
13              case 2 : cout << "Selasa" << endl;
14              break;
15              case 3 : cout << "Rabu" << endl;
16              break;
17              case 4 : cout << "Kamis" << endl;
18              break;
19              case 5 : cout << "Jumat" << endl;
20              break;
21              case 6 : cout << "Sabtu" << endl;
22              break;
23              case 7 : cout << "Minggu" << endl;
24              break;
25              default: cout << "Tidak ada..." << endl;
26              }
27              return a.exec();
28      }
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, outputnya adalah sebagai berikut.

```
sabtu
```

Keterangan Program :

- Pada program di atas variabel `hari` dideklarasikan bertipe `int` dan diinisialisasi dengan nilai 6.
- Kemudian pada bagian ekspresi di dalam `switch` di isi variabel `hari`, hasil ekspresi tersebut di evaluasi, karena menghasilkan nilai 6, maka yang dicetak ke layar adalah "**Sabtu**".

B. Perulangan

Perulangan digunakan untuk mengulang suatu perintah sebanyak yang diinginkan tanpa harus menulis ulang. C++ mengenal tiga jenis perintah perulangan,

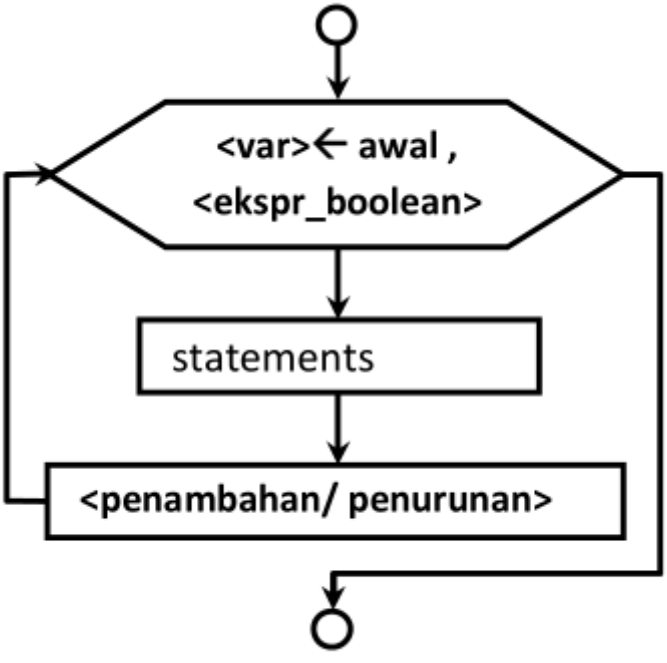
yaitu `for`, `while` dan `do ..while`.

1. Perulangan dengan for

Digunakan untuk mengulangi perintah dengan jumlah perulangan yang sudah diketahui. Pada statement for ini perlu dituliskan suatu kondisi untuk diuji yang berupa ekspresi boolean, nilai awal dan perintah yang dipakai untuk penghitung (counter). Nilai variabel penghitung akan secara otomatis bertambah atau berkurang tiap kali sebuah perulangan dilaksanakan tergantung perintah yang ditulis pada argumen ini.

Bentuk umum penulisannya sebagai berikut :

```
1  for(<nilai_awal>; <ekspresi_boolean>; <penambahan/penurunan>)
2  {
3  <statmemnts>
4  }
```

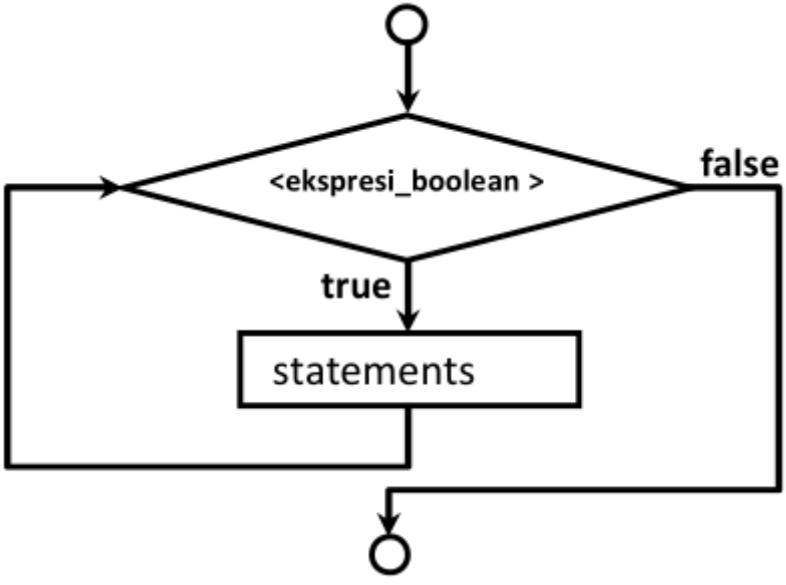


2. Perulangan dengan while

Perintah ini digunakan untuk mengulangi suatu perintah sampai kondisi tertentu. Perulangan akan terus berjalan selama kondisi masih bernilai benar.

Sintaks penulisannya sebagai berikut :

```
for(<nilai_awal>; <ekspresi_boolean>; <penambahan/penurunan>)
{
<statmemnts>
}
```

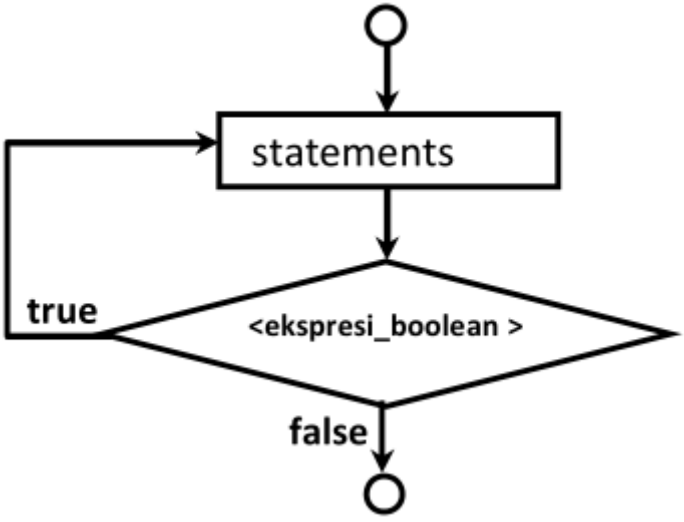


3. Perulangan dengan do ... while

Proses perulangan akan berjalan jika kondisi yang diperiksa di `while` masih bernilai benar dan perulangan akan dihentikan jika kondisinya sudah bernilai salah.

Sintaks penulisannya sebagai berikut :

```
do
{
        <statements>
}
while(<expresi_boolean>)
```



Perbedaan antara perintah `while` dengan `do … while` adalah terletak dari kondisi yang diperiksa. Pada perintah `while`, kondisi yang diperiksa terletak diawal perulangan, sehingga sebelum masuk ke dalam perulangan `while` kondisi harus bernilai benar. Sedangkan pada perintah `do … while`, kondisi diperiksa di akhir perulangan. Ini berarti bahwa paling sedikit sebuah perulangan akan dilakukan oleh perintah `do … while`, karena untuk masuk ke dalam perulangan tidak ada kondisi yang harus dipenuhi.

4. Kata kunci continue dan break

Kata kunci `break` digunakan untuk keluar dari suatu blok programn sebelum ekspresi *boloean* yang ada pada statement tersebut menghentikan, sedangkan kata kunci `continue` dugunakan untuk mengabaikan baris perintah suatu perintah di bawahnya dan melanjutkan ke perulangan selanjutnya.

Sintaks penulisan `break` dan `continue` adalah sebagai berikut :

```
while(<expresi_boolean1>)
{
        <statements>
if(<expresi_boolean2>)
        continue;
        <statements>
}
```

3. Array dan String

3.1 Array

Array adalah suatu tipe data terstuktur yang berupa sejumlah data sejenis (bertipe data sama) yang jumlahnya tetap dan diberi suatu nama tertentu. Elemen-elemen array tersusun secara sekuensial di dalam memori sehingga memiliki alamat yang berdekatan. Array dapat berupa array 1 dimensi, 2 dimensi, bahkan n-dimensi. Elemen-elemen array bertipe data sama tapi bisa bernilai sama atau berbeda-beda. Array digunakan untuk menyimpan data-data yang diinputkan masing-masing kedalam memory komputer. Jadi jumlah datanya banyak namun satu jenis.

Array dapat digunakan untuk menyimpan data yang cukup banyak namun memiliki tipe yang sama. Bagaimana array melakukan penyimpanan datanya di memory komputer? Ilustrasi array satu dimensi pada memory komputer adalah sebagai berikut:

Array menyimpan data secara berurutan pada memory komputer. Sekali array dideklarasikan (dibuat), maka akan dialokasikan sejumlah tempat di memory komputer yang selalu letaknya berdekatan (bersebelahan). Array memiliki indeks dan nilai data itu sendiri. Sedangkan jarak antar elemen pada array disesuaikan dengan lebar data untuk masing-masing tipe data array. Misalnya pada tipe data integer, maka jarak antar elemennya bernilai 2 s/d 4 byte. Indeks array pada C++ selalu dimulai dari indeks ke 0, dan seterusnya indeks ke-1, 2, 3, dan lain-lain.

A. Array 1 Dimensi

Elemen-elemen array dapat diakses oleh program menggunakan suatu indeks tertentu. Pengaksesan elemen array dapat dilakukan berurutan atau random berdasarkan indeks tertentu secara langsung. Pengisian dan pengambilan nilai pada indeks tertentu dapat dilakukan dengan mengeset nilai atau menampilkan nilai pada indeks yang dimaksud.

1. Deklarasi Array satu Dimensi

Bentuk umum deklarasi array satu dimensi:

```
1  tipe_data nama_var_array;
```

Dimana:
`tipe_data` : menyatakan jenis tipe data elemen larik (int, char, float, dll)
`nama_var_array` : menyatakan nama variabel yang dipakai.
`ukuran` : menunjukkan jumlah maksimal elemen larik.
Contoh:

```
1  char huruf[9];
2  int umur[10];
3  int kondisi[2] = {0,1};
4  int arr_dinamis[] = {1,2,3};
```

Artinya:

`char huruf[9]`
berarti akan memesan tempat di memori komputer sebanyak 9 tempat dengan indeks dari 0-8, dimana semua elemennya bertipe data karakter semuanya. Kalau satu karakter berukuran 1 byte, berarti membutuhkan memori sebesar 9 byte.

`int umur[10]`
berarti akan memesan tempat di memori komputer sebanyak 10 tempat dengan indeks dari 0-9, dimana semua elemennya bertipe data integer semuanya. Kalau satu integer berukuran 4 bytes, berarti membutuhkan memori sebesar 4 x 10 = 20 bytes.

int kondisi[2]
berarti akan memesan tempat di memori komputer sebanyak 2 tempat dengan indeks 0-1, dimana semua elemennya bertipe data integer semuanya. Dan pada contoh di atas isi elemen-elemennya yang sebanyak 2 buah diisi sekaligus (diinisialisasi) yaitu pada elemen kondisi[0] bernilai 0, dan elemen kondisi[1] bernilai 1.

int arr_dinamis[]
berarti mendeklarasikan array dengan ukuran maksimum array tidak diketahui, namun ukuran tersebut diketahui berdasarkan inisialisasi yaitu sebanyak 3 elemen, yang isinya 1,2, dan 3. Ingat bahwa array dinamis tidak bisa dibuat tanpa inisialisasi.

Tanda `[]` disebut juga "elemen yang ke- ". Misalnya "`kondisi[0]`" berarti elemen yang ke nol. Array yang sudah dipesan, misalnya 10 tempat tidak harus diisi semuanya, bisa saja hanya diisi 5 elemen saja, baik secara berurutan maupun tidak. Namun pada kondisi yang tidak sepenuhnya terisi tersebut, tempat pemesanan di memori tetap sebanyak 10 tempat, jadi tempat yang tidak terisi tetap akan terpesan dan dibiarkan kosong.

Contoh 1. Contoh Input dan Output Array
Buatlah project baru dan tulis kode berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        int nilai[5], x;
        cout<<"Memasukkan nilai"<<endl;
        for(x=0;x<5;x++)
        {
                cout<<"Nilai Angka ke - "<<x+1<< ": ";
                cin>>nilai[x];
                }
                cout<<endl;
```

```
17                  cout<<"Membaca nilai :\n";
18                  for(x=0;x<5;x++)
19                  {
20                          cout<<"Nilai Angka : "<<nilai[x]<<endl;
21                  }
22          return a.exec();
23  }
```

Hasil:

```
Memasukan nilai
Nilai Angka ke - 1 : 1
Nilai Angka ke - 2 : 2
Nilai Angka ke - 3 : 3
Nilai Angka ke - 4 : 4
Nilai Angka ke - 5 : 5

Membaca nilai:
Nilai Angka : 1
Nilai Angka : 2
Nilai Angka : 3
Nilai Angka : 4
Nilai Angka : 5
```

Keterangan:

- Pada program diatas, kita membuat sebuah variabel array bernama `nilai` yang berisi `5` elemen bertipe `integer`. Kemudian untuk memasukkan nilai ke masing-masing elemen, digunakan perintah perulangan untuk mengakses indeksnya yang dimulai dari indeks ke `0`. Perulangan dilakukan dari indeks ke `0` sampai dengan indeks ke `4` (dalam hal ini `x < 5`). Mengapa sampai dengan indeks ke `4`? Hal ini karena `5` elemen array yang kita deklarasikan dimulai dari indeks ke `0`. Terdapat `5` elemen array, berarti indeks ke `0`, `1`, `2`, `3`, dan `4`.
- Setelah kita masukkan nilai ke masing-masing elemen, maka kita hanya perlu membaca datanya lagi, yaitu dengan melakukan perulangan kembali dengan

cara mengakses indeks elemen-elemennya seperti pada saat kita memasukkan elemen-elemen tersebut kedalam *array*. Perulangan untuk membaca isi elemen array juga diulang dari 0 sampai 4, yang artinya juga 5 elemen. Pada masing-masing perulangan tersebut, ditampilkan isi elemen ke layar dengan perintah cout<<.

Contoh 2. Contoh Manipulasi Array
Buatlah project baru dan tulis kode berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        int bil[7],i;
        cout<<"elemen ke-1 ? "; cin>>bil[0];
        bil[1] = 5;
        bil[2] = bil[1] + 20;
        for(i=4;i<7;i++)
        bil[i] = i*10;
        bil[3] = bil[bil[1]];
        for(i=0;i<7;i++)
        cout<<"bil["<<i<<"] = "<<bil[i]<<" dan alamatnya: "<<&bil[i]<<"\n";
        return a.exec();
}
```

Hasil:

```
elemen ke-1 ? 1
bil[0] = 1 dan alamatnya: 0x28fe68
bil[1] = 5 dan alamatnya: 0x28fe6c
bil[2] = 25 dan alamatnya: 0x28fe70
bil[3] = 50 dan alamatnya: 0x28fe74
```

```
bil[4] = 40 dan alamatnya: 0x28fe78
bil[5] = 50 dan alamatnya: 0x28fe7c
bil[6] = 60 dan alamatnya: 0x28fe80
```

Keterangan:

- Program diatas memasukkan nilai-nilai integer kedalam array bernama bil yang berisi `7` elemen (dari indeks `0-6`).
- Dalam array satu dimensi, suatu elemen array dapat diisi dengan isi elemen array pada indeks tertentu seperti pada contoh `bil[2] = bil[1] + 20;`. Pada contoh diatas, `bil[2]` diisi dengan `bil[1]` yang berisi `25` ditambah dengan `20`, yaitu `55`.
- Pada program `bil[3] = bil[bil[1]]`, artinya bilangan elemen ke-3 diisi dengan elemen array yang ke – `bil[1]`. Bilangan elemen ke-1, bernilai `5`, yang berarti `bil[3] = bil[5]`. `Bil[5]` bernilai `50`, berarti `bil[3] = 50` juga.
- Terlihat bahwa jarak antar elemen array `bil` berjarak `4 bytes`.
- Cara untuk menampilkan alamat *array* adalah dengan menggunakan operator `&`.

TIPS

Dalam bahasa C++, tidak terdapat *error handling* terhadap batasan nilai indeks, apakah indeks tersebut berada di dalam indeks array yang sudah didefinisikan atau belum. Hal ini merupakan tanggung jawab programmer. Sehingga jika programmer mengakses indeks yang salah, maka nilai yang dihasilkan akan berbeda atau rusak karena mengakses alamat memori yang tidak sesuai.

Contoh 3. Penanganan Batas Indeks Elemen Array
Buatlah program beikut ini:

```
#include <QtCore/QCoreApplication>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        int angka[5];
        angka[0] = 0; //batas bawah array
        angka[4] = 4; //batas atas array
        angka[5] = 5; //indeks melebihi batas
        //program akan HANG
        cout<<angka[5];
        return a.exec();
}
```

Hasil dan Keterangan:

- Progarm akan HANG-UP. Hal ini terjadi karena compiler tidak bertanggung-jawab dengan pengaksesan indeks array yang melebihi batas yang dipesankan di memory.
- Mengapa kompiler tidak menampilkan error pada saat kompilasi? Hal ini karena secara sintaks, program diatas tidaklah memiliki error penulisan. Error yang terjadi pada program diatas adalah runtime error, yaitu error yang terjadi / yang bisa dideteksi saat program sudah berjalan!

Inisialisasi Array Satu Dimensi

Array satu dimensi dapat diisi secara langsung ditulis pada program. Pengisian data seperti itu sering disebut dengan inisialisasi data array. Cara menginisialisasi data pada array adalah dengan menuliskannya secara langsung pada source code program. Berikut contohnya:

```
// An array of 5 integers, all elements initialized to 0
int IntegerArray[5] = {0};
```

Pada contoh diatas, semua elemen array bertipe integer yang berjumlah 5 buah tersebut diisi dengan nilai 0 semuanya. Cara lain menginisialisasi array satu dimensi adalah sebagai berikut:

```
// An array of 5 integers initialized to zero
int IntegerArray[5] = { 0, 0, 0, 0, 0 };
```

Nah, bagaimana jika kita ingin menginisialisasi elemen terakhirnya saja? Kita tidak bisa melakukannya secara langsung. Yang harus dilakukan adalah dengan menginisialisasinya satu-persatu seperti contoh berikut:

```
// An array of 5 integers initialized to zero
int IntegerArray[5] = { 0, 0, 0, 0, 6 };
```

Pada contoh diatas, elemen terakhir diinilisasi dengan nilai 6. Kita tidak bisa langsung mengisi dengan cara `int IntegerArray[5] = {6}`, karena jika di isi dengan cara demikian, maka isi elemen indeks ke-0 bernilai 6, sedangkan elemen lainnya bernilai 0.

Contoh 4. Inisialisasi Array dengan nilai 0

Buatlah program berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        int bil[7] = {0}; //inisialisasi 0
        for(int i=0;i<7;i++){
        cout<<"Elemen ke-"<<i<<": "<<bil[i]<<"\n";
        }
        return a.exec();
}
```

Hasil:

```
Elemen ke-0: 0
Elemen ke-1: 1
Elemen ke-2: 2
Elemen ke-3: 3
Elemen ke-4: 4
Elemen ke-5: 5
Elemen ke-6: 6
```

Keterangan

Pada program diatas elemen array bernama bil yang dipesan sebanyak 7 elemen, di inisialisasi dengan nilai 0. Setelah di inisialisasi dengan nilai 0, maka semua elemen array tersebut juga akan berisi dengan nilai 0. Hal ini dibuktikan dengan cara perulangan semua elemen array dan ditampilkan dengan `cout`.

Contoh 5. Inisialisasi Array dua nilai elemen pertama

Buatlah program berikut ini:

```
#include <QtCore/QCoreApplication>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        int bil[7] = {2,5}; //inisialisasi dua elemen pertama
        for(int i=0;i<7;i++){
                cout<<"Elemen ke-"<<i<<": "<<bil[i]<<"\n";
        }
        return a.exec();
}
```

Hasil:

```
Elemen ke-0: 2
Elemen ke-1: 5
Elemen ke-2: 0
Elemen ke-3: 0
Elemen ke-4: 0
Elemen ke-5: 0
Elemen ke-6: 0
```

Keterangan

Inisialisasi elemen array dapat dilakukan hanya pada dua elemen pertama saja, hal ini dilakukan dengan cara memberikan dua nilai pertama, selanjutnya semua elemen lainnya yang tidak di inisialisasi secara otomatis bernilai 0.

TIPS

Untuk semua array pada C++, inisialisasi satu buah elemen saja pada array akan membuat semua elemen array lainnya berisi nilai 0.

Contoh:

```
1  int angka[100] = {1};
```

Maka hasilnya adalah:

```
1  angka[0] = 1,
2  angka[1] s/d angka[99] = 0
```

Pada array satu dimensi, kita tidak dapat melakukan inisialisasi pada array melebihi batas jumlah elemen array yang dipesan.

Pada array satu dimensi, kita juga dapat membuat array 1 dimensi tanpa menyebutkan jumlah elemen array yang dipesan. Namun perlu di ingat bahwa semua elemen harus di inisialisai terlebih dahulu.

Contoh:

```
int data[5] = {1,2,3,4,5,6}; //error
int data2[] = {10,20}; //terpesan 2 tempat dimemory
```

Contoh 6. Tanpa inisialisasi, array langsung ditampilkan
Tulislah program berikut ini:

```
#include <QtCore/QCoreApplication>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        char h[5];
        for(int i=0;i<5;i++){
                cout<<"Elemen ke "<<i<<" = "<<h[i]<<endl;
        }
        return a.exec();
}
```

Hasil:

```
Elemen ke-0: 2
Elemen ke-1: 5
Elemen ke-2: 0
Elemen ke-3: 0
Elemen ke-4: v
```

Keterangan

Pada program C++, elemen array yang sudah dipesan dimemory pasti sudah berisi data. Namun nilai datanya bersifat acak. Sehingga jika kita mendeklarasikan sebuah elemen array tanpa di inisialisasi, maka nilai masing-masing elemen akan bersifat acak juga seperti pada hasil program diatas. Untuk itulah inisialisasi elemen array sangatlah penting.

TIPS

Inisialisasi pada elemen array yang dideklarsikan **SANGATLAH PENTING** untuk menghindari nilai **ACAK**!

Contoh 7. Penggunaan tipe data enum pada Array satu dimensi
Buatlah program berikut:

```
1  #include <QtCore/QCoreApplication>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(int argc, char *argv[])
7  {
8          QCoreApplication a(argc, argv);
9          enum Hari { Minggu, Senin, Selasa, Rabu, Kamis, Jumat, Sabtu };
10         int ArrayHari[7] = { 10, 20, 30, 40, 50, 60, 70 };
11         cout << "Nilai Hari Selasa adalah: " << ArrayHari[Selasa];
12         return a.exec();
13 }
```

Hasil:

```
Nilai hari selasa adalah = 30
```

Keterangan:

Pada program diatas, kita membuat sebuah tipe data enum bernama Hari yang memiliki 7 elemen. Masing-masing elemen `enum` sama saja seperti indeks array yaitu `0-6`. Kemudian kita membuat sebuah array bernama `ArrayHari` yang berisi 7 elemen juga dan berisi nilai `10-70`. Karena kita memanggil `ArrayHari[Selasa]` berarti sama artinya dengan `ArrayHari[2]`. Mengapa 2? Karena indeks Selasa adalah 2. Sehingga muncullah output berupa 30, karena 30 berada pada indeks ke-2 dari `ArrayHari`.

Arti dari program diatas menunjukkan kita dapat mengakses indeks *array* dengan menggunakan `tipe data enum`, karena tipe data `enum` pada kenyataannya

akan dikonversikan kedalam nilai `integer`, mulai dari `0`.

Pengalamatan dan Pengkopian Array 1 Dimensi

Array tidak bisa disalin begitu saja antara array satu yang ada nilainya ke array lain yang kosong. Hal ini dikerenakan array bukanlah tipe data primitif biasa. Array merupakan tipe data referensi dimana data yang berada didalam elemen array berjumlah lebih dari satu buah dan diakses dengan menggunakan alamat memory. Compiler C++ akan mencatat alamat pertama dari indeks pertama array yang kita deklarasikan.

Contoh:

```
1  int data[5] = {1,2,3,4,5};
```

Maka variabel array data tersebut akan dicatat alamat elemen `data[0]` pada memory. Jika kita mengakses elemen keduanya, yaitu `data[1]`, maka compiler akan melakukan kalkulasi untuk mendapatkan alamat `data[1]`, yaitu dengan cara menambahkan alamat `data[0]` dengan lebar tipe data array yang kita deklarasikan. Pada contoh diatas, kita membuat array bertipe integer. Karena integer berukuran `4` byte, maka jika `data[0]` beralamat di alamat `1000`, maka `data[1]` beralamat di `1000 + 4 = 1004` dan seterusnya.

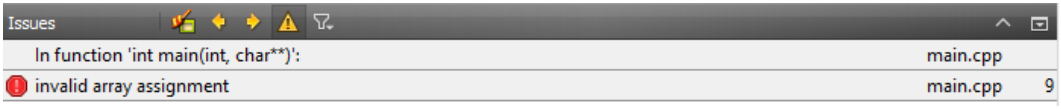
Lalu bagaimana cara mengkopikan isi elemen array dari satu variabel ke variable array 1 dimensi lainnya? Kita harus menggunakan cara manual, yaitu mengkopikan masing-masing elemennya satu persatu dengan perulangan manual sesuai dengan jumlah elemen array yang dibuat.

Contoh 8. Percobaan Penyalinan Array 1 dimensi

Buatlah program berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        int A[6]={1,2,3,4,5,6};
        int B[6];
        B = A;
        return a.exec();
}
```

Hasil:



Keterangan:

Program tidak bisa dijalankan karena terdapat **error**, bahwa array tidak bisa dilakukan operasi assigment. Artinya kita tidak bisa mengkopi antar array begitu saja.

Contoh 9. Penyalinan Array 1 dimensi dengan Perulangan

Buatlah program berikut ini:

```
#include <QtCore/QCoreApplication>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        int A[6]={1,2,3,4,5,6};
        int B[6];
```

```
11          for(int i=0;i<6;i++){
12                  B[i]=A[i];
13          }
14          for(int j=0;j<6;j++){
15                  cout<<B[j]<<endl;
16          }
17          return a.exec();
18  }
```

Hasil:

```
1
2
3
4
5
6
```

Keterangan:

- Cara penyalinan array adalah dengan melakukan perulangan sebanyak elemen array yang akan disalin dan menyalinnya secara manual satu-persatu pada indeks yang sama.
- Kemudian ditampilkan sesuai dengan indeksnya. Elemen array yang dikopikan masih tetap memiliki array yang asli. Untuk menghapusnya, maka harus dilakukan secara manual.

Array Multi Dimensi

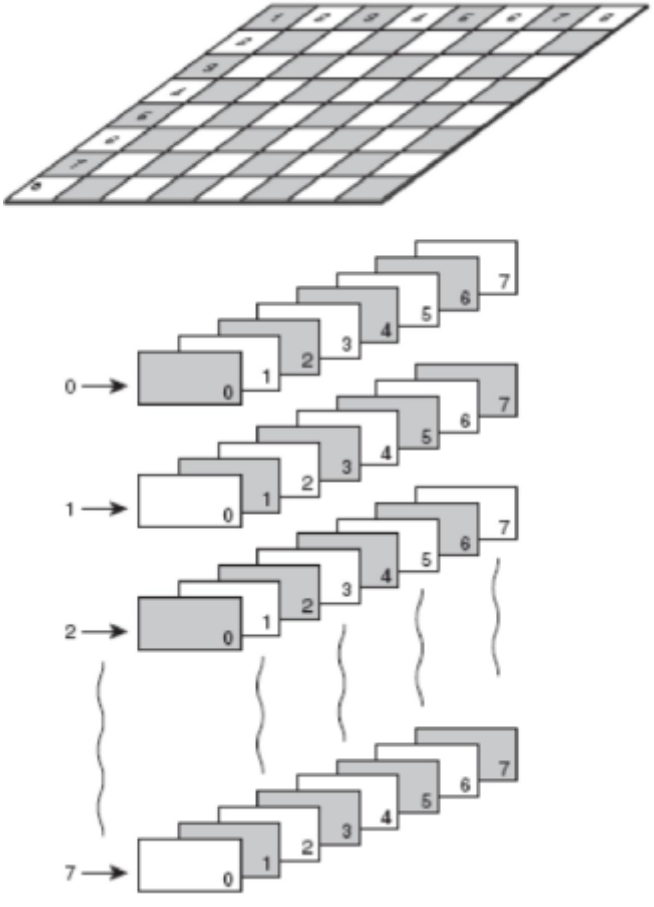
Array multi dimensi berarti array yang kita deklasaikan dapat dikembangkan ke array dimensi 2 dan seteruanya. Array multi dimensi merupakan topik yang menarik dalam matematika. Setiap dimensi dalam array direpresentasikan sebagai sub bagian dalam array. Oleh karena itu, array dua dimensi array memiliki dua sub bagian, sebuah array tiga-dimensi memiliki tiga sub bagian dan sebagainya. Sebuah

contoh bentuk nyata yang baik dari array dua dimensi adalah sebuah papan catur. Satu dimensinya merupakan delapan baris, sedangkan dimensi lainnya merupakan delapan kolom.

Contoh deklarasi array dua dimensi yang menggambarkan papan catur adalah:

```
1  int papan[8][8];
```

yang digambarkan dalam bentuk:



Array dua dimensi sering kali digambarkan/dianalogikan sebagai sebuah matriks atau bentuk grid. Jika array berdimensi satu hanya terdiri dari 1 baris dan banyak

kolom, array berdimensi dua terdiri dari banyak baris dan banyak kolom yang bertipe sama. Ilustrasi array dua dimensi dapat dilihat sebagai berikut.

Berikut adalah gambar array berdimensi (baris x kolom = 3 x 4)

	0	1	2	3
0	5	20	1	11
1	4	7	67	-9
2	9	0	45	3

Deklarasi Array Dua Dimensi

```
tipe_data nama_var_array[batas_baris][batas_kolom];
```

Contoh:

```
int matriks[3][4];
int matriks2[3][4] = { {5,20,1,11}, {4,7,67,-9}, {9,0,45,3} };
```

Array dua dimensi dapat mewakili bentuk suatu matriks, contoh matriks:

$$X = \begin{pmatrix} 8 & 5 & 9 & 8 \\ 8 & 2 & 1 & 0 \end{pmatrix}$$

selanjutnya dapat dideklarasikan sebagai berikut:

```
int x[2][4];
```

atau diklarasikan dengan langsung menginisialisasi nilai elemen-elemen-nya sebagai berikut:

```
int x[2][4]= {{8, 5, 9, 8},{8, 2, 1, 0}}
```

Selanjutnya larik dua dimensi x dapat digambarkan sebagai berikut:

```
x[0][0]=8 x[0][1]=5 x[0][2]=9 x[0][3]=8
x[1][0]=8 x[1][1]=2 x[1][2]=1 x[1][3]=0
```

Array dua dimensi dapat digunakan untuk menampung tipe data numerik atau non numerik.

Berikut adalah berbagai bentuk pembuatan array dua dimensi dengan tipe data numerik ataupun non numerik.

Array dua dimensi bertipe data numerik

```
int matriks[3][5] = {{5,12,17,10,7},
{15,6,25,2,19},
{4,9,20,22,11}};
```

Jika data array integer yang diinputkan kurang dari deklarasi

```
int matriks[3][5] = {{5,12,17,10,7},
{15,6,25,2,19},
{4,9 }}; //kurang 3 angka
```

Maka tiga data yang kurang akan diisi dengan angka 0

Array 2 dimensi dapat juga digunakan untuk menyimpan data karakter (character). Pendeklarasian array 2 dimensi character adalah sebagai berikut:

```
char matriks[3][5] = {{'A','B','C','D','E'},
{'F','G','H','I','J'},
{'K','L','M','N','O'}};
char matriks[3][5] = {"ABCDE",
"FGHIJ",
"KLMNO"};
```

Akan ditampilkan sebagai:

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O

Array 2 dimensi juga dapat dideklarasikan sebagai berikut:

```
char matriks[5][12] = {"Jakarta",
"Bandung",
"Surabaya",
"Semarang",
"Yogyakarta"};
```

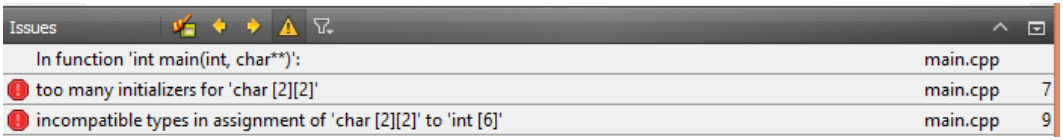
Array diatas akan ditampilkan sebagai:

J	a	k	a	r	t	a	\0			
B	a	n	d	u	n	g	\0			
S	u	r	a	b	y	a	\0			
S	e	m	a	r	a	n	g	\0		
Y	o	g	y	a	k	r	t	a	\0	

Jika jumlah nilai character lebih banyak daripada deklarasi

```
1 char matriks2[2][2] = {'a','b','c','d','e'};
```

Akan terjadi ERROR!



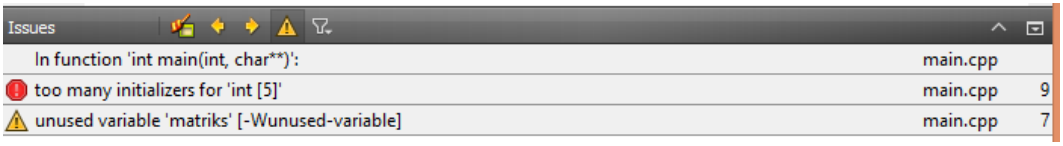
Jika data array character yang diinputkan kurang dari deklarasi

```
1 char matriks[3][5] = {{‘a’,’b’,’c’,’d’,’e’},
2 {‘f’,’g’,’h’,’i’,’j’},
3 {‘k’,’l’ }}; //kurang 3 karakter
```

Maka tiga data yang kurang akan diisi dengan karakter NULL atau ‘\0’
Jika data array integer yang diinputkan lebih dari deklarasi

```
1 int matriks[3][5] = {{5,12,17,10,7},
2 {15,6,25,2,19},
3 {4,9,20,22,11,14,19 }}; //lebih 2 angka
```

Matriks yang jumlah datanya lebih akan menyebabkan ERROR



Array 2 dimensi juga dapat dideklarasikan secara dinamis. Dinamis bisa dilakukan pada baris array 2 dimensi. Namun kita tidak bisa mendeklarasikan array 2 dimensi secara dinamis pada kolom. Contoh pendeklarasian baris dinamis adalah :

```
int matriks[][5] = {{5,12,17,10,7},
{15,6,25,2,19},
{4,9,20,22,11}};
```

Akan ditampilkan sebagai:

5	12	17	10	7
15	6	25	2	19
4	9	20	22	11

Contoh matriks dengan deklarasi baris dinamis lainnya:

```
int matriks[][5] = {5,12,17,10,7,
15,6,25,2,19,
4,9,20,22,11,77,88,99};
```

Pada contoh diatas, jika kita hitung jumlah datanya adalah 18 buah, padahal jika kita bagi per lima kolom, maka data 18 akan lebih 3 buah (18/5 = 3). Sehingga secara otomatis terdapat 3 baris dan sisa 3 buah data berikutnya akan membuat baris baru. Array dua dimensi tersebut akan ditampilkan sebagai:

5	12	17	10	7
15	6	25	2	19
4	9	20	22	11
77	88	99	0	0

Pengaksesan Array 2 Dimensi

Pengaksesan elemen-elemen array 2 dimensi dilakukan dengan cara perulangan. Perulangan yang dilakukan harus disesuaikan dengan jumlah dimensinya. Maka array 2 dimensi berarti perulangan yang dilakukan harus dua kali. Terdapat outer loop yang digunakan untuk mengakses baris array 2 dimensi, dan inner loop yang digunakan untuk mengakses kolom array 2 dimensi.

Contoh 10. Deklarasi dan Menampilkan Array 2 Dimensi

Buatlah program berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        int matriks[3][5] = {{5,12,17,10,7},
        {15,6,25,2,19},
        {4,9,1,5,2}};
        for(int i=0;i<3;i++){
                for(int j=0;j<5;j++){
                        cout<<matriks[i][j]<<"\t";
                }
                cout<<endl;
        }
        return a.exec();
}
```

Hasil:

```
5       12      17      10      7
```

```
15      6       25      2       19
4       9       1       5       2
```

Keterangan:

Program diatas mendeklarasikan sebuah variabel array 2 dimensi bernama matriks berukuran 3 baris dan 5 kolom. Kemudian matriks tersebut langsung diinisialisasi dengan data integer sejumlah 15 data. Setelah diinisialisasi kemudian dilakukan pengaksesan terhadap array 2 dimensi tersebut dengan cara melakukan dua buah perulangan. Perulangan pertama disebut outer loop yang digunakan untuk mengakses indeks baris variabel matriks, sedangkan perulangan kedua disebut inner loop yang digunakan untuk mengakses indeks kolom variabel matriks. Kemudian untuk menampilkan data nya digunakan perintah cout dan untuk setiap data elemen array diberikan karakter tab yang digunakan untuk memberi jarak antar output data. Karakter tab pada bahasa C menggunakan escape character '\t'.

Contoh 11. Penyalinan Array 2 Dimensi ke Array 2 Dimensi lainnya

Misalkan terdapat array 2 dimensi sebagai berikut matriks[3][5]

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

Buatlah program berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        int matriks[3][5]={{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15}};
        int matrikshasil[3][5];
        for(int i=0;i<3;i++){
                for(int j=0;j<5;j++){
                        matrikshasil[i][j] = matriks[i][j];
                }
        }
        for(int i=0;i<3;i++){
                for(int j=0;j<5;j++){
                        cout<<matrikshasil[i][j]<<"\t";
                }
        cout<<endl;
        }

        return a.exec();
}
```

Hasil:

```
1       2       3       4       5
6       7       8       9       10
11       12       13       14       15
```

Keterangan:

Program diatas menyalin data dari matriks 2 dimensi ke matriks 2 dimensi lainnya dengan menggunakan perulangan bertingkat. Perulangan bertingkat memiliki

2 buah loop, yang pertama (*outer loop*) digunakan untuk mengakses baris matriks, dan inner loop digunakan untuk mengakses kolom matriks. Kemudian untuk masing-masing elemen matriks dimasukkan kedalam variabel array matrikshasil tepat pada baris dan kolom yang sesuai.

Contoh 12. Penyalinan array 2 dimensi ke dalam array 1 dimensi.

Buatlah program berikut ini:

```
#include <QtCore/QCoreApplication>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        int matriks[3][5]={{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15}};
        int matrikshasil[15];
        int counter=-1;
        for(int i=0;i<3;i++){
                for(int j=0;j<5;j++){
                        counter++;
                        matrikshasil[counter] = matriks[i][j];
                }
        }
        for(int i=0;i<15;i++){
                cout<<matrikshasil[i]<<endl;
        }

        return a.exec();
}
```

Hasil:

```
1
2
3
```

```
4
5
6
7
8
9
10
11
12
13
14
15
```

Keterangan:

- Untuk menyalin array 2 dimensi ke 1 dimensi, maka harus diperlukan sebuah array 1 dimensi baru yang berukuran total sesuai dengan hasil perkalian antara ukuran baris matriks dua dimensi dikalikan kolomnya. Misal array 2 dimensi berukuran 3 x 5, maka harus dibuat array 1 dimensi berukuran minimal 15.
- Kemudian untuk mengkopikan dari array 2 dimensi matriks ke array 1 dimensi matrikshasil, harus dilakukan perulangan sesuai dengan baris dan kolom matriks. Indeks array matrikshasil diperoleh dari penambahan nilai counter yang diinisialisasi dari -1, dan berjalan mulai dari 0 sampai dengan 14.

Cara Pengaksesan Array dapat dilakukan dengan 2 cara:

1. Pengaksesan Baris demi Baris

Cara ini menelusuri elemen array dua dimensi per dimulai dari baris pertama lalu kekanan sesuai dengan jumlah kolomnya. Setelah eleman dalam baris tersebut habis, maka penelusuran akan berganti baris ke baris berikutnya dan demikian seterusnya. Cara ini membutuhkan 2 buah loop, dimana outer loop digunakan untuk mengakses indeks baris, dan inner loop digunakan untuk mengakses indeks kolom.

Berikut adalah contohnya:

```
#include <QtCore/QCoreApplication>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        int A[2][3]={{1,2,3},{4,5,6}};
        for(int baris=0;baris<2;baris++){
                for(int kolom=0;kolom<3;kolom++){
                        cout<<A[baris][kolom]<<"\t";
                }
        cout<<endl;
        }
        return a.exec();
}
```

Hasil:

```
1       2       3
4       5       6
```

2. Pengaksesan Kolom demi Kolom

Cara ini menelusuri elemen array dua dimensi per dimulai dari kolom pertama lalu kebawah sesuai dengan jumlah barisnya. Setelah eleman dalam kolom tersebut habis, maka penelusuran akan berganti kolom ke kolom berikutnya dan demikian seterusnya. Cara ini membutuhkan 2 buah loop, dimana outer loop digunakan untuk mengakses indeks kolom, dan inner loop digunakan untuk mengakses indeks baris.

Contoh:

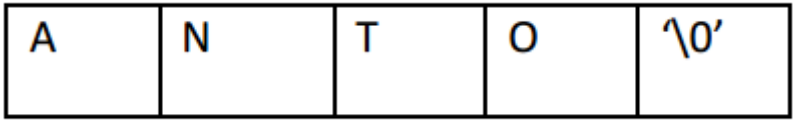
```
#include <QtCore/QCoreApplication>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        int A[2][3]={{1,2,3},{4,5,6}};
        for(int kolom=0;kolom<3;kolom++){
                for(int baris=0;baris<2;baris++){
                        cout<<A[baris][kolom]<<"\t";
                }
        cout<<endl;
        }
        return a.exec();
}
```

Hasil:

```
1       4
2       5
3       6
```

String

String adalah kumpulan dari nilai-nilai karakter yang berurutan dalam bentuk satu dimensi, nilai string ini haruslah ditulis didalam tanda petik dua (“) misalnya: “ini string”. Suatu nilai string disimpan di memori dengan diakhiri oleh nilai ‘\0’(null), misalnya nilai string “ANTO” disimpan dimemori dalam bentuk



Dengan mengetahui nilai string diakhiri oleh nilai '\0', maka akhir nilai dari suatu string dapat dideteksi.

Untuk mendeklarasikan sebuah string terdapat dua cara:

1. Menggunakan array of character yang sering disebut C-style string
2. Menggunakan tipe data string pada C++

Array of Character

Cara menggunakan array of character sama seperti mendeklarasikan variabel bertipe array namun bertipe data character seperti yang sudah dijelaskan pada bagian-bagian sebelumnya. Array of character memiliki sifat-sifat array lainnya yaitu bersifat statis dan letaknya berurutan di dalam memory komputer. String berjenis array of character selalu dibuat dengan menggunakan array satu dimensi yang dapat terdiri dari karakter-karakter yang ditulis dengan menggunakan tanda petik tunggal (') atau satu kesatuan string yang ditulis dengan tanda petik ganda (").

Contoh pendeklarasian array of character

```
1 char nama[6]; //tanpa inisialisasi
2 char nama2[6] = "anton"; //langsung diinisialisasi
3 char nama2[6] = {'a','n','t','o','n'}; //langsung diinisialisasi
```

Pada sebuah string, terdapat karakter `\0` yang dapat digunakan untuk mengetahui kapan berakhirnya suatu string. Berikut adalah contoh penggunaaanya.

Contoh 13. Penggunaan karakter \0

Buatlah program berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        char string[100]="String";
        int K;
        for(K=0;string[K]!='\0';K++)
        cout<<string[K];
        return a.exec();
}
```

Hasil:

```
String
```

Keterangan:

Program diatas dapat mengetahui kapan berakhirnya suatu string, dalam arti kita dapat mengetahui panjang suatu string dengan melakukan perulangan untuk setiap karakter yang ada pada array sampai ditemukannya katakter '\0'.

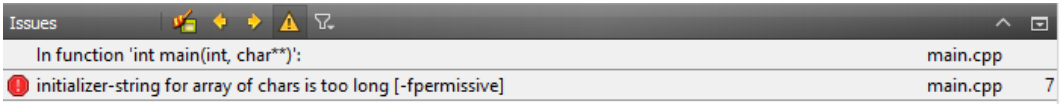
Contoh 14. String tanpa karakter \0

Buatlah program berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        char string[6]="String";
        int K;
        for(K=0;string[K]!='\0';K++)
        cout<<string[K];
```

```
11          return a.exec();
12  }
```

Hasil:



Keterangan:

Terlihat bahwa kita tidak bisa membuat array fa character tepat sesuai dengan jumlah karakter yang kita inisialisasikan. Jika dilihat kata "String" berjumlah 6 huruf, sedangkan kita sudah mendeklarasikan variabel `string[6]` namun ternyata jumlah elemennya masih dianggap terlalu sedikit. Hal ini terjadi karena minimal kita harus mengalokasikan sejumlah 7 buah elemen. Elemen ke-7 digunakan untuk menyimpan tanda akhir string atau karakter \0 tersebut.

Contoh 15. Mengisi Array of Character

Buatlah program berikut:

```
1   #include <QtCore/QCoreApplication>
2   #include <iostream>
3
4   using namespace std;
5
6   int main(int argc, char *argv[])
7   {
8           QCoreApplication a(argc, argv);
9           char buffer[50] = {'\0'};
10          cout << "Isi data string: ";
11          cin >> buffer;
12          cout << "Hasil data string: " << buffer << endl;
13          return a.exec();
14  }
```

Hasil:

```
Isi data string: Baehaki
hasil data string: Baehaki
```

Keterangan:

- Pada program diatas kita mendeklarasikan variabel array of string bernama buffer yang berukuran 6 elemen. Variabel buffer diatas merupakan variabel berjenis string C-style yang diinisialisasi dengan karakter \0 atau karakter NULL.
- Problem lainnya adalah jika kita menginputkan data string yang mengandung spasi, maka cin hanya akan membaca data string sebelum spasi saja.

Contoh:

```
Isi data string: Ahmad Baehaki
hasil data string: Ahmad
```

Contoh 16. Pengisian variabel array of character dengan maksimum jumlah karakter.

Tulislah program berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        char buffer[50] = {'\0'};
        cout << "Isi data string: ";
        cin.get(buffer,49); //ambil sebanyak 50 karakter atau diakhiri tanda e\
nter
        cout << "Hasil data string: " << buffer << endl;
        return a.exec();
}
```

Hasil:

```
Isi data string: Ahmad Baehaki
hasil data string: Ahmad Baehaki
```

Keterangan:

Pada contoh program diatas, kita menggunakan perintah `cin.get(buffer,49)`. Perintah diatas "memaksa" agar perintah `cin` mengambil semua data inputan ke dalam variabel buffer sampai sejumlah 49 karakter. Jika karakter yang diinputkan lebih dari 50 karakter, maka otomatis karakter yang disimpan kedalam variabel buffer hanyalah berjumlah 50 karakter pertama saja.

Fungsi-fungsi String

Bahasa C++ menggunakan fungsi-fungsi pustaka yang disediakan untuk meng-operasikan suatu nilai string yang dimasukkan dalam file header string.h. Beberapa fungsi string yang terdapat pada header string.h adalah sebagai berikut:

1 `strlen()`

Berfungsi untuk menentukkan panjang suatu nilai string.
Bentuk umum: `int strlen(<identifier string>);`

1 `length()`

Berfungsi untuk menentukan panjang suatu nilai tipe data class string
Bentuk umum method: `<nama_var_string>.length();`
Contoh 17. Penggunaan fungsi strlen()
Buatlah program berikut ini:

```
#include <QtCore/QCoreApplication>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        char data[100];
        cout<<"Masukkan kalimat apapun yang anda sukai (max 100 huruf): ";
        cin.get(data,99);
        cout<<"panjang huruf adalah: "<<strlen(data)<<" karakter";
        return a.exec();
}
```

Hasil:

```
Masukkan kalimat apapun yang anda sukai (max 100 huruf): Nur Wachid
panjang huruf adalah: 10 karakter
```

Keterangan:

Fungsi strlen menerima satu parameter yang hanya bertipe array of character. Fungsi ini tidak bisa menerima parameter berupa tipe data C++ string.

Contoh 17. Penggunaan fungsi length pada tipe data string C++

Buatlah program berikut ini:

```
#include <QtCore/QCoreApplication>
#include <iostream>
#include <string>

using namespace std;

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        string data2;
        cout<<"Masukkan kalimat apapun yang anda sukai (max 100 huruf): ";
        getline(cin,data2);
        cout<<"panjang huruf adalah: "<<data2.length()<<" karakter";
        return a.exec();
}
```

Hasil:

```
Masukkan kalimat apapun yang anda sukai (max 100 huruf): Nur Wachid
panjang huruf adalah: 10 karakter
```

Keterangan

- Program diatas tidak menggunakan array of character, melainkan menggunakan tipe data C++ `class string`. Tipe data ini spesial karena berupa tipe data object oriented. Untuk menggunakan tipe data ini kita harus menginclude-kan `#include <string>` pada bagian *preprocessor directive*.
- Kemudian untuk mengakses panjang karakternya digunakan method (fungsi) dari object string bernama `length()`. Fungsi `length` sama dengan fungsi `strlen` yaitu mengambil jumlah karakter dalam string tersebut.

strcpy() dan strncpy()

Dalam bahasa C++, untuk menyalin nilai suatu string tidak dapat langsung menuliskannya seperti halnya kompiler lain, sehingga proses menyalin atau menger-

jakan suatu nilai string ke variabel string yang lain diperlukan suatu fungsi pustaka yang bernama `strcpy()`.

1. Bentuk umum: `void strcpy(<stringhasil>,<stringsumber>);`
2. Bentuk umum: `void strncpy(<stringhasil>,<stringsumber>);`

Contoh 18. Penggunaan fungsi strcpy()
Buatlah program berikut ini:

```
#include <QtCore/QCoreApplication>
#include <iostream>
#include <string>

using namespace std;

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        char data[100] = {'\0'};
        char data2[]="STRING";
        strcpy(data,data2);
        cout<<"string pertama: "<<data<<"\n";
        cout<<"string kedua : "<<data2;
        return a.exec();
}
```

Hasil:

```
String pertama : STRING
String Kedua : STRING
```

Keterangan:

- Program diatas digunakan untuk mengkopikan nilai dari array of character data ke data2 dengan menggunakan perintah `strcpy`. Hal itu terbukti dengan hasil akhir dimana string pertama dan kedua bernilai sama, yaitu “STRING”.

- Jika variabel sesumber lebih besar daripada variabel hasil, `trcpy ()` akan error karena melebihi buffer. Untuk melindungi hal ini, digunakan fungsi `strncpy ()` Fungsi ini dapat memberikan parameter jumlah maksimum karakter untuk penyalinan. `strncpy ()` akan menyalin sampai karakter null pertama atau jumlah maksimum. Contoh Error:

```
- char data[5] = {'\0'}; - char data2[]="STRING";
```

- Hal ini terjadi karena data2 berjumlah 6 karakter, sedangkan data berjumlah 5 karakter. Jadi ketika data2 dikopikan ke data, maka akan terjadi error karena tempatnya kurang.

Contoh 19. Penggunaan fungsi strncpy()
Buatlah program beriku:

```
#include <QtCore/QCoreApplication>
#include <iostream>
#include <string>

using namespace std;

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        char data[6] = {'\0'};
        char data2[]="STRINGKU";
        strncpy(data,data2,5);
        cout<<"string pertama: "<<data<<"\n";
        cout<<"string kedua : "<<data2;
        return a.exec();
}
```

Hasil:

```
String pertama : STRING
```

```
String Kedua : STRINGKU
```

Keterangan:

- Fungsi `strncpy` dapat digunakan untuk menyalin dari satu array of character ke array of character lainnya dengan memberikan penanda batas maksimal penyalinan. Pada contoh diatas, string "STRINGKU" hendak disalin ke variabel data yang hanya berisi 6 elemen. Karena fungsi `strncpy` hanya dibatasi menyalin 5 karakter saja, maka yang tersalin adalah `STRIN` saja.
- Karakter ke-6 pada variabel data digunakan untuk menyimpan karakter NULL atau \0.

strcat()

Bentuk umum: `strcat(<string hasil>, <string sumber>);`

String dalam C++ tidak bisa digabungkan begitu saja dengan menggunakan operator + seperti pada bahasa pemrograman Pascal. Jika dipaksakan menggunakan operator + akan ditampilkan pesan kesalahan sebagai berikut ini.

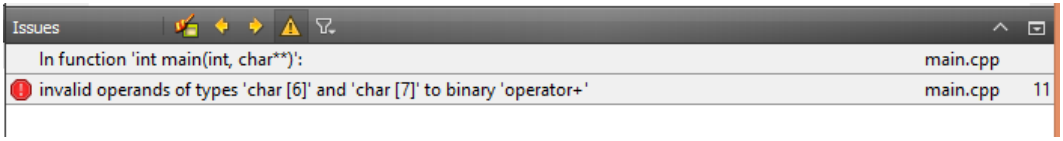
Contoh 20. Penggunaan fungsi strcat()

Buatlah program beriku:

```
1  #include <QtCore/QCoreApplication>
2  #include <iostream>
3  #include <string>
4
5  using namespace std;
6
7  int main(int argc, char *argv[])
8
9  {
10         QCoreApplication a(argc, argv);
11         char str[6] = "anton";
12         char str2[7];
13         char str3[14];
14         str3 = str + str2;
```

```
15          return a.exec();
16  }
```

Hasil:



Keterangan:

Opearator + tidak bisa digunakan untuk menggabungkan dua buah string. Untuk menggabungkan dua string, digunakan fungsi `strcat()`.

Contoh 21. Penggunaan fungsi strcat()

Buatlah program berikut:

```
1   #include <QtCore/QCoreApplication>
2   #include <iostream>
3   #include <string>
4
5   using namespace std;
6
7   int main(int argc, char *argv[])
8   {
9           QCoreApplication a(argc, argv);
10          char string1[100]="Kami kelompok ";
11          char string2[]=" belajar Qt C++";
12          strcat(string1,string2);
13          cout<<"Jadi gabungannya adalah: "<<string1;
14          return a.exec();
15  }
```

Hasil:

```
Jadi gabungannya adalah: Kami kelompok  belajar Qt C++
```

Keterangan:

- Program diatas menggunakan fungsi `strcat` dimana fungsi tersebut akan menggabungkan dua buah string. Parameter string pertama juga digunakan untuk menampung string gabungan kedua string tersebut. Sehingga pada akhirnya variabel string1 lah yang ditampilkan ke layar.
- Variabel `string1` diberi ukuran 100 karena jika tidak diberi ukuran elemen maka `string1` tidak bisa memperbesar ukurannya di memory komputer sehingga akan menyebabkan program **HANG**.

TIPS

Beberapa fungsi yang `#include string.h` dan dapat digunakan untuk memanipulasi *array of character* adalah:

strrev()

Bentuk umum: `strrev(string)`

Digunakan untuk membalik susunan string, misal: anton menjadi notna

strlwr()

Bentuk umum: `strlwr(string)`

Digunakan untuk mengubah string menjadi huruf kecil semua

strupr

Bentuk umum: `strupr(string)`

Digunakan untuk mengubah string menjadi huruf besar semua

strchr()

Bentuk umum: `strchr(stringsumber,karakter yang dicari)`

Dalam bahasa C++ disediakan suatu fungsi pustaka yaitu `strchr()` untuk mencari nilai suatu karakter yang ada di suatu string. Hasil dari fungsi ini adalah alamat letak dari karakter pertama di nilai string yang sama dengan karakter yang dicari.

strcmp()

Bentuk umum: `strcmp(string1,string2);`

Untuk membandingkan dua nilai string tidak bisa menggunakan operator hubungan, karena operator tersebut tidak untuk operasi string. Untuk membandingkan dua nilai string kita gunakan fungsi pustaka `strcmp()` dengan hasil sebagai berikut:

- Hasil < 0, Jika string1 < string2
- Hasil = 0, Jika string1 = string2
- Hasil > 0, Jika string1 > string2

Fungsi mengubah string menjadi numerik dan sebaliknya

Pada bahasa C++ tipe data *array of character* bisa dikonversi menjadi `numerik` dan sebaliknya numerik bisa dikonversi menjadi `array of character`. Caranya adalah `#include <stdlib>`. Fungsi-fungsi konversi dari string ke numerik adalah:

```
atoi() //untuk mengubah string menjadi int
atof() //untuk mengubah string menjadi float
atol() //untuk mengubah string menjadi long int
```

Sedangkan kebalikannya, fungsi untuk mengubah numerik menjadi string adalah:

```
itoa() //untuk mengubah int menjadi string
ltoa() //untuk mengubah long int menjadi string
ultoa() //untuk mengubah unsigned long menjadi string
```

Fungsi diatas menerima parameter `<var numerik>`, `<var array of character>`, dan `<basis bilangan>`

Class string pada C++

C++ library standar memiliki kelas string yang membuat bekerja dengan string lebih mudah dengan menyediakan satu set encapsulasi dari data, dan fungsi untuk memanipulasi data string. Kelas ini dikenal dengan `std::` string yang dapat menangani rincian alokasi memori dan membuat kopi string, atau menempatkan mereka di memory dengan lebih mudah.

Contoh 22. Pembuatan variabel string C++, penyalinan string, dan penggabungan string

Buatlah program berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
#include <string>
using namespace std;
int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        string str1("Ini string C++");
        cout<<"Isi str1 = "<<str1<<endl;
        //salin isi str1 ke str2
        string str2;
        str2 = str1;
        cout<<"Isi str2 = "<<str2<<endl;
        //ubah str2
        str2 = "Hallo, ";
        //buat strhasil dan isi dgn gabungan dari str1 dan str2
        string strhasil;
        strhasil = str2 + str1;
        cout<<strhasil;
        return a.exec();
}
```

Hasil:

```
Isi str1 : Ini string C++
Isi str1 : Ini string C++
Isi str1 : Ini string C++
Halo, Ini string C++
```

Keterangan:

- Tanpa perlu dipelajari lebih dalam, kita dapat melihat bahwa class string pada C++ jelas jauh lebih cepat penggunaannya dan mudah dalam pembuatan serta penyalinan seperti semudah mengoperasikan variabel bertipe integer saja. Demikian pula, *concatenating* (penggabungan) dua string dapat dilakukan

dengan hanya menambahkan mereka, sama juga seperti kita akan melakukan penjumlahan dengan integer apapun.

- Syarat untuk dapat menggunakan class string adalah harus mengincludekan `#include <string>`, seperti yang dapat dilihat pada kode program diatas.

TIPS

Class string memiliki beberapa fitur / manfaat, yaitu:

- Mengurangi kesulitan dalam upaya penciptaan dan memanipulasi string
- Meningkatkan stabilitas aplikasi yang sedang diprogram dalam pengelolaan dan alokasi memori internal
- Mudah dalam menyalin, memotong, menemukan, dan penghapusan string
- Memberikan kesempatan pada programmer untuk lebih fokus pada pengembangan aplikasi daripada kesulitan dalam manipulasi string

Contoh 23. Penggunaan class string untuk manipulasi data
Buatlah program berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
#include <string>
using namespace std;
int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        //buat var nama dgn C-style string
        char nama[10] = "Halooooooo";
        //buat var string dan diisi nilai dari var nama
        string nama_copy(nama);
        cout<<nama_copy<<endl;
        //buat var nama2 dan kopikan isinya ke nama2_copy melalui konstruktor
        string nama2 = "saya belajar";
```

```
15          string nama2_copy(nama2);
16          cout<<nama2_copy<<endl;
17          //buat var nama35 yg diisi nilai dari nama tapi hanya 5 huruf saja
18          string nama35(nama,5);
19          cout<<nama35<<endl;
20          //buat var ulang yg diisi huruf 'C' sebanyak 10 buah
21          string ulang(10,'a');
22          cout<<ulang;
23          return a.exec();
24  }
```

Hasil:

```
Halooooooo
saya belajar
C
```

Keterangan:
Dapat dilihat langsung pada baris komentar program diatas.
Contoh 24. Penggabungan string dengan menggunakan class string
Buatlah program berikut:

```
1   #include <QtCore/QCoreApplication>
2   #include <iostream>
3   #include <string>
4   using namespace std;
5   int main(int argc, char *argv[])
6   {
7           QCoreApplication a(argc, argv);
8           string satu("Percobaan 1 ");
9           string dua("Percobaan 2 ");
10          satu += dua;
11          cout<<satu<<endl;
12          string tampung = "Percobaan tampung";
13          satu.append("Percobaan 3 ");
```

```
14          satu.append(tampung);
15          cout<<satu;
16          return a.exec();
17  }
```

Hasil:

```
Percobaan 1                Percobaan 2
Percobaan 1                Percobaan 2         Percobaan 3         Percebaan tampung
```

Keterangan:

- Pada program diatas, terdapat dua buah variabel bertipe string, yaitu `satu` dan `dua`. Tipe data string tidak mendukung penggabungan string dengan mudah yaitu dengan menggunakan operator +. Pada contoh diatas, variabel satu ditambah isinya dengan variabel dua dan disimpan kembali pada variabel satu. Sehingga variabel satu berisi string gabungan "Percobaan 1 Percobaan 2".
- Kemudian dibuat suatu variabel tampung yang kemudian juga digabungkan kedalam variabel satu. Cara penggabungan (concatenation) string dapat dilakukan juga dengan cara kedua, yaitu dengan menggunakan `method append`. `Method append` ini dimiliki oleh semua variabel bertipe class string dan dapat langsung digunakan dengan memasukkan parameter bertipe string juga.

Contoh 25. Pengaksesan isi nilai class string
Buatlah program berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
#include <string>

using namespace std;

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        string satu("Indonesia Raya");
        for(size_t i=0;i<satu.length();i++){
                cout<<satu[i]<<endl;
        }
        cout<<endl;
        cout<<"C-syltle: "<<satu.c_str();
        return a.exec();
}
```

Hasil:

```
I
n
d
o
n
e
s
i
a

R
a
y
a
C-style: Indonesia Raya
```

Keterangan:

- Variabel string yang bertipe class string juga memiliki sifat yang sama dengan variabel string dengan model C-string style. Keduanya merupakan gabungan dari karakter-karakter yang berbentuk array berdimensi satu. Sehingga jika kita memiliki variabel string satu seperti pada program, kita dapat mengakses semua elemen-elemen karakter penyusun string tersebut dengan menggunakan perulangan dan kemudian kita akses indeks dari masing-masing elemen array characternya.
- Pada bagian kedua, kita juga bisa mengkonversi dari tipe data class string menjadi tipe data array of character atau tipe data C-style string dengan menggunakan method dari class string, yaitu `c_str()`.

Contoh 26. Menemukan substring pada sebuah string besar
Tulislah program berikut ini:

```
#include <QtCore/QCoreApplication>
#include <iostream>
#include <string>

using namespace std;

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        string strSample ("Kata pak Hari, \"hari ini matahari cerah sekali!\""\
);
        cout << "Contoh string adalah: " << endl;
        cout << strSample << endl << endl;
        // Temukan kata "hari"
        size_t nOffset = strSample.find ("hari", 0);
        // Cek apakah ketemu?
        if (nOffset != string::npos)
        cout << "Ketemu pertama kata \"hari\" pada offset " << nOffset;
        else
        cout << "Substring tidak ditemukan" << endl;
        cout << endl << endl;
```

```
22          cout << "Mencari semua kata substring \"hari\"" << endl;
23          size_t nSubstringOffset = strSample.find ("hari", 0);
24          while (nSubstringOffset != string::npos)
25          {
26          cout << "Kata \"hari\" ada di offset " << nSubstringOffset << endl;
27          // Pencarian dilanjutkan ke karakter berikutnya dst
28          size_t nSearchOffset = nSubstringOffset + 1;
29          nSubstringOffset = strSample.find ("hari", nSearchOffset);
30          }
31          cout << endl;
32          cout << "Mencari semua karakter 'a'" << endl;
33          const char chCharToSearch = 'a';
34          size_t nCharacterOffset = strSample.find (chCharToSearch, 0);
35          while (nCharacterOffset != string::npos)
36          {
37          cout << "'" << chCharToSearch << "' ditemukan";
38          cout << " pada posisi " << nCharacterOffset << endl;
39          //pencarian dilanjutkan
40          size_t nCharSearchOffset = nCharacterOffset + 1;
41          nCharacterOffset = strSample.find(chCharToSearch,nCharSearchOffset);
42          }
43          return a.exec();
44  }
```

Hasil:

```
Contoh string adalah:
Kata pak Hari, "hari ini matahari cerah sekali!"

Ketemu pertama kata "hari" pada offset 16

Mencari semua kata substring "hari"
Kata "hari" ada di offset 16
Kata "hari" ada di offset 29

Mencari semua karakter 'a'
```

```
'a' ditemukan pada posisi 1
'a' ditemukan pada posisi 3
'a' ditemukan pada posisi 6
'a' ditemukan pada posisi 10
'a' ditemukan pada posisi 17
'a' ditemukan pada posisi 26
'a' ditemukan pada posisi 28
'a' ditemukan pada posisi 30
'a' ditemukan pada posisi 37
'a' ditemukan pada posisi 43
```

Keterangan:

- Program diatas membuat sebuah variabel string bernama strSample yang diisi dengan kalimat : "Kata pak Hari, "hari ini matahari cerah sekali!"". Kemudian program akan mencari kata "hari" yang pertama ditemukan pada kalimat tersebut dengan menggunakan method find(`<kata yang dicari>`,`<posisi indeks dimulainya pencarian>`). Method ini bersifat case-sensitive sehingga kata "Hari" dengan "hari" berbeda. Pencarian dimulai dari huruf pertama, sehingga kata "hari" ditemukan pada huruf ke 16, bukan ke-9, karena karakter ke-9 kata "Hari" menggunakan huruf besar.
- Pencarian berikutnya adalah pencarian semua kata "hari". Karena kata "hari" ada lebih dari satu buah, maka pencarian harus diloop, karena method find membutuhkan indeks mulainya pencarian. Untuk setiap kata "hari" yang ditemukan, kemudian ditampilkan posisi indeksnya ke layar.
- Selain dapat menerima parameter berupa substring, method find juga dapat menerima parameter berupa character dengan proses pencarian yang sama dengan proses pencarian dengan parameter substring.

Contoh 27. Membalik kata / kalimat.
Tulislah program berikut ini:

```
#include <QtCore/QCoreApplication>
#include <iostream>
#include <string>
#include <algorithm>

using namespace std;

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        string strSample ("String ini akan dibalik!");
        cout << "String asli: " << endl;
        cout << strSample << endl << endl;
        reverse (strSample.begin (), strSample.end ());
        cout << "Setelah dibalik: " << endl;
        cout << strSample;
        return a.exec();
}
```

Hasil:

```
String asli:
String ini akan dibalik!

Setelah dibalik:
!kilabid naka ini gnirtS
```

Keterangan:

Untuk membalik kalimat bertipe string, kita harus menggunakan library header algoritm, sehingga kita harus mengincludekan library tersebut `#include <algorithm>`. Setelah itu untuk menggunakannya kita gunakan perintah *reverse*(`<indeks string pertam>`,`<indeks string terakhir>`). Perintah reverse tersebut akan benar-benar mengganti string asli menjadi terbalik, sehingga variable string kita akan berubah berisi kalimat yang sudah terbalik.

Contoh 28. Konversi huruf besar dan kecil.
Tulislah program berikut ini:

```
#include <QtCore/QCoreApplication>
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        cout << "Masukkan sebuah string: " << endl;
        string strInput;
        getline (cin, strInput);
        transform(strInput.begin(),strInput.end(),strInput.begin(),(int(*)(int\
))toupper);
        cout << "Hasil konversi ke huruf besar: " << endl;
        cout << strInput << endl << endl;
        transform
        (strInput.begin(),strInput.end(),strInput.begin(),(int(*)(int))tolower\
);
        cout << "Hasil konversi ke huruf kecil: " << endl;
        cout << strInput << endl << endl;
        return a.exec();
}
```

Hasil:

```
Masukkan sebuah string:
Ini KoK tulisaNya AlaY BaNGet yA!
Hasil konversi ke huruf besar:
INI KOK TULISANYA ALAY BANGET YA!

Hasil konversi ke huruf kecil:
```

```
    ini kok tulisanya alay banget ya!
```

Keterangan:

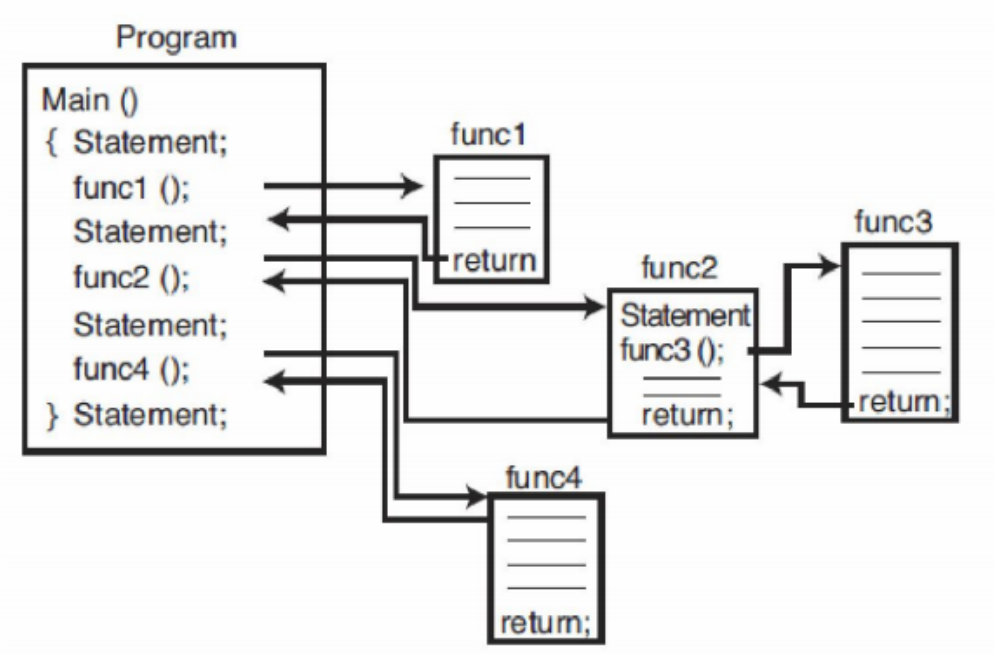
Program diatas menunjukkan function transform pada library algoritm dapat digunakan untuk mengkonversi string dari besar ke kecil dan dari kecil ke besar.

4. Fungsi

Fungsi (Function) adalah sekumpulan program yang diberi nama, sehingga dengan demikian jika program itu diperlukan dapat dipanggil kembali. Walaupun Pemrograman Berorientasi Objek telah menggeser perhatian dari fungsi ini, namun fungsi tetap saja merupakan bagian paling inti dalam suatu program. Fungsi global bisa berada di luar kelas maupun objek.

Fungsi dapat melakukan manipulasi terhadap data dan dapat mengembalikan suatu nilai. Semua program yang ditulis dengan bahasa C++ paling tidak mempunyai sebuah fungsi, yaitu `main()`, fungsi ini akan dipanggil secara otomatis ketika program dieksekusi, sedangkan fungsi yang lain baru akan bekerja ketia fungsi tersebut dipanggil. Karena fungsi ini bukan merupakan bagian dari objek, maka fungsi ini dipanggil secara global, dapat diakses dari manapun dalam program yang ditulis

Setiap fungsi diberi nama, dan ketika dalam suatu program dijumpai nama tersebut, maka eksekusi program akan dialihkan ke tubuh (isi) fungsi tersebut, setelah selesai, yaitu ditandai dengan statemen `return` atau tanda kurung kurawal tutup, maka akan kembali ke progam utama melanjutkan ke baris program berikutnya. Peristiwa ini dinamakan pemanggilan fungsi, berikut ini adalah ilustrasi mengenai pemanggilan fugsi :



Fungsi yang baik mengerjakan sebuah pekerjaan yang spesifik, mudah dipahami dan mudah dikenali berdasarkan nama fungsi tersebut. Pekerjaan yang kompleks seharusnya dipecah-pecah menjadi beberapa fungsi yang nantinya dapat dipanggil ketika diperlukan.

Fungsi terdiri dari 2 macam, yaitu fungsi yang dibuat sendiri (*user-defined*) dan fungsi standard (*built-in*).Fungsi standard merupakan bagian dari paket kompiler yang kita pakai yang sudah tersedia untuk digunakan, sedangkan fungsi yang dibuat sendiri adalah fungsi yang kita tulis sebelum dapat dipergunakan.

Konsep Dasar Fungsi

Fungsi sebenarnya mirip dengan prosedur (pada bhs. Pascal), dan kedua hal ini disebut sebagai `Subrutin`. Kedua jenis subrutin ini (fungsi dan prosedur) memiliki kegunaan yang sama, yaitu melakukan tugas tertentu. Perbedaannya fungsi selalu mengembalikan suatu nilai setelah dipanggil sedangkan prosedur tidak.

Kita memerlukan subrutin, karena dalam program yang besar akan lebih baik jika tugas tertentu dilakukan oleh subrutin tertentu, dengan demikian program akan menjadi lebih mudah dibaca dan dipelihara.

Catatan :

Fungsi bisa dikatakan sebagai bentuk lain dari instruksi yang dapat memberikan sebuah nilai apabila diberi masukan yang dibutuhkan. Masukan tersebut dikenal dengan istiah Parameter.

Fungsi-fungsi merupakan elemen utama dari program bahasa C++. Program dari bahasa C++ dibentuk dari kumpulan fungsi, mulai dari fungsi utama dengan nama `main()`, fungsi-fungsi pustaka (standar) dan fungsi-fungsi yang dibuat sendiri oleh pemrogram (UDF = *User Defined Function*). Fungsi-fungsi banyak digunakan dengan dua alasan utama, yaitu:

1. Fungsi-fungsi menjadikan program C++ mempunyai struktur yang jelas. Dengan memisahkan langkah–langkah detail ke satu atau lebih fungsi–fungsi, maka fungsi utama (`main()`) akan menjadi lebih pendek, jelas dan mudah dimengerti. Hal seperti ini menunjukan suatu struktur program yang baik.
2. Fungsi-fungsi dapat digunakan untuk menghindari penulisan program yang sama ditulis secara berulang-ulang. Selanjutnya bagian program yang membutuhkan langkah-langkah yang sama tidak perlu selalu dituliskan, melainkan cukup memanggil fungsi-fungsi tersebut.

Suatu fungsi harus diberi nama supaya dapat dipanggil dari bagian program yang membutuhkannya. Tugas yang dilakukan oleh suatu fungsi dapat berupa tugas input/output, penyeleksian atau tugas-tugas perhitungan dan sebagainya.

Mendefinisikan Fungsi

Secara umum, fungsi terdiri dari dua komponen yaitu definisi fungsi dan tubuh fungsi. Isi dari definisi fungsi adalah : tipe dari fungsi, nama dari suatu fungsi dan paramter-parameter yang digunakan. Tubuh dari fungsi berisikan statemen-statemen yang akan melakukan tugas yang diberikan oleh fungsi tersebut. Tubuh suatu fungsi diawali dengan tanda kurung kurawal buka dan diakhiri dengan tanda kurung kurawal tutup. Beikut ini adalah bentuk umum dari suatu fungsi:

```
<tipe> <nama_fungsi>([<paramter1>, <paramter2> ,...])
{
<tubuh fungsi>
[return <ekspresi>]
}
```

Definisi fungsi ditulis sebelum dituliskan tubuh fungsi dan tidak diakhiri dengan tanda titik koma. Tipe dari definisi fungsi sesuai dengan tipe data dari nilai yang dikembalikan jika fungsi itu mempunyai statment `return`, jika tidak terdapat statement `return` tipe ini diberi tipe `void`. Nama suatu fungsi dibentuk sendiri oleh pemrogram sesuai dengan syarat penamaan identifier yang telah dibahas pada bab 2 dan nama fungsi yang baik mencerminkan pekerjaan dari fungsi tersebut. Parameter suatu fungsi dapat dituliskan dengan dipisahkan oleh tanda koma, bisa mempunyai beberapa parameter namun dapat juga tidak mempunyai parameter sama sekali. Parameter dibutuhkan jika dalam tubuh fungsi memerlukan nilai dari luar fungsi. Parameter ini dinamakan parameter formal. Berikut ini adalah contoh cara mendefinisikan fungsi.

```
int terbesar(int bil1, int bil2)
{
int hasil;
if (bil1>bil2)
kembali = bil1;
else
kembali = bil2;
return kembali;
}
```

Deklarasi Fungsi (Prototype)

Suatu fungsi harus dideklarasikan sebelum digunakan, jika suatu fungsi tidak dideklarasikan maka fungsi tersebut tidak akan bisa dipanggil. Deklarasi tersebut akan memberitahukan kepada kompiler mengenai nama fungsi, tipe data kembalian dan parameter dari fungsi, sedangkan definisi dari fungsi memberitahukan kepada kompiler mengenai cara kerja fungsi. Deklarasi fungsi ini dinamakan prototipe (*prototype*).

Ada tiga cara mendeklarasikan fungsi (membuat *prototype*), yaitu :

- Menuliskan prototipe ke dalam sebuah file, kemudian menggunakan directive `#include` untuk menyertakannya.
- Tuliskan prototype di dalam file yang memakai fungsi tersebut.
- Definisikan fungsi di file yang memakai fungsi tersebut di posisi sebelum pemanggilnya, dengan demikian definisi fungsi ini bertidak sebagai prototype itu sendiri.

Meskipun kita dapat mendefiniskan fungsi sebelum digunakan, sehingga bisa menghindari pembuatan prototype, namun cara ini merupakan cara yang tidak baik karena tiga alasan. Pertama, menampilkan fungsi dalam sebuah file dengan urutan tertentu adalah tidak baik, karena akan menyulitkan ketika terjadi perubahan program.

Kedua, ada kemungkinan fungsi pertama memerlukan pemanggilan fungsi kedua, tetapi ada juga kemungkinan fungsi kedua memanggil fungsi yang pertama. Pada kasus semacam ini tidak mungkin menempatkan definisi fungsi pada urutan yang benar tanpa membuat prototype.

Ketiga, penggunaan prototype merupakan teknik penelusuran kesalahan yang baik dan handal. Ketika suatu prototype mendeklarasikan fungsi dengan parameter tertentu dan nilai kembalian tertentu, maka kompiler akan menjaga konsistensinya dengan definisi fungsi tanpa harus menunggu program dijalankan.

Compiler C++ dapat memeriksa tipe data melalui parameter-parameter (actual parameter) yang dikirimkan dari program yang menggunakannya, dengan terlebih dahulu menyebutkan prototype fungsi tersebut. Jika terjadi kesalahan perbedaan antara tipe-tipe data parameter nyata yang dikirim dengan tipe-tipe data parameter formalnya, maka dapat diketahui melalui ketidakcocokan antara compiler untuk tipe data tersebut.

Prototype fungsi standard berada di file-file judulnya, dalam fungsi pustaka sebagai contoh, fungsi pustaka `printf()`, prototypenya berada di dalam file judul `stdio.h`. Pencantuman prototype fungsi dapat menggunakan preprocessor directive `#include`.

Contoh 1. Membuat Fungsi yang mengembalikan nilai.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 1, kemudian tulis kode berikut.

```
#include <QtCore/QCoreApplication>
#include <iostream>
int absolut(int bil);
int main(int argc, char *argv[])
{
  using namespace std;
  QCoreApplication a(argc, argv);
  int bilangan = -10;
    cout << "Bilangan : " << bilangan << endl;
    cout << "Dimutlakkan menjadi : " << absolut(bilangan) << endl;
  return a.exec();
  }
  int absolut(int bil){
    if(bil<0)
    return - bil;
    else
    return bil;
}
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, outputnya adalah sebagai berikut.

```
Bilangan : -10
Dimutlakkan menjadi : 10
```

Analisa Program :

- Pada program diatas baris ketiga tertulis : `int absolut(int bil);` inilah yang disebut sebagai prototype, ditulis sebelum fungsi yang memakainya, yaitu `main()`.
- Pada tubuh pogram, terdapat pemanggilan fungsi :

```
cout << "Dimutlakkan menjadi : " << absolut(bilangan) << endl;
```

Tampak pada hasil eksekusi bahwa nama fungsi tersebut digantikan dengan nilai 10, yaitu nilai kembalian fungsi, ini menunjukkan bahwa fungsi `absolut()` mengembalikan nilai.

- Di bawah fungsi `main()` terdapat sebuah blok program dengan nama `absolut()`, inilah yang
  dinamakan definisi fungsi.

Catatan :

Nama parameter pada prototype tidak harus sama dengan nama parameter pada definisi fungsi, oleh karena itu prototype tersebut di atas boleh juga dituliskan seperti berikut :

```
int absolut(int x);
```

Nama parameter pada prototype tidak harus disebutkan, oleh karena itu prototype tersebut di atas boleh juga dituliskan seperti berikut :

```
int absolut(int);
```

Hasil Balik Fungsi

Suatu fungsi dalam menyelesaikan tugasnya, dapat hanya melakukan suatu tugas tanpa memberikan suatu nilai kembalian atau melakukan suatu tugas yang kemudian memberikan suatu nilai kembalian. Fungsi yang hanya menampilkan hasil di layar merupakan suatu fungsi yang hanya melakukan tugasnya saja tanpa memberikan hasil balik. Untuk membuat fungsi yang tidak mempunyai nilai kembalian digunakan tipe data void untuk tipe fungsi tersebut dan pada tubuh definisi fungsi tidak ada satmenet return.

Contoh 2 Membuat Fungsi yang tidak mengembalikan nilai.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama contoh 2, kemudian tulis kode berikut.

```
#include <QtCore/QCoreApplication>
#include <iostream>
void hello(int kali);
int main(int argc, char *argv[])
{
QCoreApplication a(argc, argv);
hello(3);
return a.exec();
}
void hello(int kali){
using namespace std;
for(int x=0;x<kali;x++)
cout << "Hello World!" << endl;
}
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, outputnya adalah sebagai berikut.

```
Hello World!
Hello World!
Hello World!
```

Keterangan Program :

- Pada program diatas baris ketiga tertulis : `void hello(int kali);` tampak tipe dari fungsi ini adalah void, berarti tidak mengembalikan nilai.
- Pada tubuh pogram, terdapat pemanggilan fungsi :

`hello(3);`

Tampak pada hasil eksekusi bahwa nama fungsi ini dieksekusi bukan diakses nilainya (dicetak dengan cout), ini menunjukkan bahwa fungsi `hello()` tersebut tidak mengembalikan nilai.

- Di bawah fungsi `main()` terdapat sebuah blok program dengan nama `hello()`, pada tubuh fungsi ini tidak ada perintah `return` sama sekali, karena memang tidak mengembalikan nilai.

Jika suatu fungsi memberikan nilai kembalian, maka nilai kembalian yang diberikan oleh fungsi dapat dilakukan oleh statemen return yang diikuti oleh nilai hasil baliknya. Contoh fungsi yang mengembalikan nilai adalah seperti contoh pada Lab1 di atas.

Ruang Lingkup Variabel

Variabel-variabel memiliki ruang lingkup yang berbeda-beda, sesuai dengan ruang lingkup variabel, jenis-jenis variable dapat dibagi menjadi tiga:

1. Variable Lokal

Variable lokal merupakan variable yang hanya berlaku untuk pernyataan di dalam satu blok statemen saja, tidak dapat dipergunakan oleh blok lain, pendeklarasianya variabel lokal berada di dalam blok statement (dalam kurung kurawal) yang bersangkutan. Variabel lokal akan dihapus dari memori jika proses sudah meninggalkan blok statemen letak variable lokalnya.

Contoh 3. Variabel Lokal.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 2, kemudian tulis kode berikut.

```
1   #include <QtCore/QCoreApplication>
2   #include <iostream>
3   float kali(float a, float b); /*prototype fungsi*/
4   int main(int argc, char *argv[])
5   {
6   using namespace std;
7   QCoreApplication a(argc, argv);
8   float hasil;
9   hasil = kali(4,7);
10  cout << "Hasil = " << hasil << endl;
```

```
11  return a.exec();
12  }
13  float kali(float a, float b)
14  {
15  float c;
16  c = a * b;
17  return c;
18  }
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, outputnya adalah sebagai berikut.

```
Hasil = 28
```

Analisa Program:

- Variable a, b dan c merupakan variabel lokal pada fungsi `kali()`. Variabel ini tidak dikenal pada fungsi utama sehingga variabel ini tidak dapat digunakan pada fungsi `main()` di atas, sebaliknya variabel hasil adalah variabel yang sifatnya lokal pada fungsi `main()`, sehingga tidak dikenal pada fungsi `kali()`.
- Jika variabel a atau b atau c dibaca pada fungsi `main()` maka akan terjadi kesalahan, yaitu bahwa variabel-variabel tersebut tidak dikenal (tidak dideklarasikan), demikian juga jika variabel hasil diakses di dalam fungsi `kali()`, maka variabel tersebut juga tidak akan dikenal.
- Variabel lokal sifat kerjanya hanya sekali. Jadi ketika fungsi `kali()` selesai dieksekusi, maka variabel a, b dan c dibebaskan dari memori, ketika fungsi ini dipanggil kembali di waktu lain, maka akan terjadi deklarasi (pemesanan tempat) lagi dan dianggap sebagai variabel baru.

2. Variable Global

Sesuai dengan namanya, variable global maksudnya adalah suatu variable yang dapat dikenali oleh semua bagian dari program, tidak hanya terbatas pada satu blok

statemen saja. Supaya menjadi variabel global, maka variabel global ini dideklarasikan di luar suatu blok ataupun di luar fungsi-fungsi yang mengguanakanya.

Contoh 4. Variabel Global.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 2, kemudian tulis kode berikut.

```
#include <QtCore/QCoreApplication>
#include <iostream>
void kali(float a, float b); /*prototype fungsi*/
float hasil; /*variabel global*/
int main(int argc, char *argv[])
{
  using namespace std;
  QCoreApplication a(argc, argv);
  kali(4,7);
  cout << "Variabel global hasil = " << hasil << endl;
  return a.exec();
  }
  void kali(float a, float b)
  {
  hasil = a * b;
}
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, outputnya adalah sebagai berikut.

```
Variabel global hasil = 28
```

Analisa Program:

- Variabel hasil dideklarasikan di luar blok program (di luar kurung kurawal), maka variabel hasil merupakan variabel global yang dikenal di blok manapun.

- Ketika variabel hasil mengalami manipulasi di dalam fungsi `kali()`, maka sebenarnya yang diubah adalah variabel hasil yang sama, sehingga ketika ditampilkan dengan cout variabel ini menghasilkan nilai perkalian antara a dan b seperti apa yang dilakukan di dalam fungsi `kali()`.
- Perlu diperhatikan bahwa variabel hasil bersifat global bagi fungsi `main()` maupun fungsi `kali()` karena deklarasi variabel hasil tersebut diletakkan di atas kedua fungsi-fungsi tersebut. Jadi letak deklarasi suatu vaiabel yang diluar blok, menentukan cakupan sifat global variabel tersebut.

3. Variabel statik

Jika dilihat dari prinsip kerjanya, variabel statik bertentangan dengan variable lokal, variable lokal tidak lagi digunakan setelah suatu proses dalam blok selesai, namun variable static adalah jenis variabel yang masih tetap ada nilainya dan akan tetap dipertahankan nilainya walaupun sudah keluar dari proses. Sebenarnya variabel statik ini merupakan pengubah (modifer) dari variable lokal atau global, sehingga variabel statik dapat bersifat statik lokal atau statik global tergantung dari letak pendeklarasianya.

Contoh 5. Variabel Statik.

Buka Qt Creator dan buat project Qt Console Application baru dengan nama contoh 2, kemudian tulis kode berikut.

```
1  #include <QtCore/QCoreApplication>
2  #include <iostream>
3  long int kali(long int i); /*prototype*/
4  int main(int argc, char *argv[])
5  {
6  using namespace std;
7  QCoreApplication a(argc, argv);
8  int i,n;
9  long int fak;
10 n = 5;
11 /*menghitun n faktorial (5!)*/
12 if(n<=0)
13 fak=0;
14 else
```

```
15 for(i=1;i<=n;i++)
16 fak = kali(i);
17 cout << n << " Faktorial = " << fak << endl;
18 return a.exec();
19 }
20 /*---Fungsi kali---*/
21 long int kali(long int i)
22 {
23 static long int f=1;
24 f = f * i;
25 }
```

Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, outputnya adalah sebagai berikut.

```
5 Faktorial = 120
```

Analisa Program:

- Dari contoh program ini, variable `f` di fungsi `kali ()` merupakan variable lokal yang bersifat statik yang mempunyai nilai awal `1`. Pada fungsi ini nilai variabel `f` yang lama akan dikalikan dengan nilai variable `i` untuk mendapatkan nialai `f` yang baru.
- Pada fungsi utama, fungsi `kali()` akan dipanggil sebanyak n kali dengan nilai yang dikirim ke fungsi berupa nilai `1` sampai dengan nilai `n` (pada contoh ini `n = 5`), sehingga akan dihasilkan suatu niali n!.
- Supaya nilai variable `f` yang lama masih tetap dipertahankan, maka variable ini perlu dibuat menjadi variable statik. Jika variabel ini tidak bersifat static, maka setiap kali fungsi `kali()` dipanggil, nilai variable `f` akan mempunyai nilai awal 1 lagi.

Penggunaan variabel lokal lebih disarankan, karena penggunaan variabel global akan memnyebabkan dampak-dampak sebagai berikut :

1. Memboroskan memori computer karena computer masih menyimpan nilainya walaupun sudah tidak diperlukan lagi.

2. Mudah terjadi kesalahan program karena satu perubahan dapat menyebabkan perubahan menyeluruh pada program.
3. Pembuatan fungsi lebih sulit, karena harus diketahui variable global apa saja yang digunakan.
4. Pendeteksian kesalahan program lebih sulit dilakukan.

Pengiriman Parameter

Seperti contoh program-program di atas, fungsi dapat menerima nilai melalui parameter formal dan dapat mengembalikan nilai melalui statment `return`. Ketika fungsi dipanggil, fungsi tersebut akan melakukan suatu pekerjaan dan mengirimkan suatu nilai hasil suatu pekerjaan tersebut yang dinamakan nilai kembalian (`return value`). Jika kita mendeklarasikan seperti berikut:

```
1  int fungsiku();
```

Ini berarti kita mendeklarasikan fungsi bernama fungsiku yang akan mengembalikan nilai bertipe integer. Jika kita mendeklarasikan seperti berikut:

```
1  int fungsiku(int nilaiInt, float nilaiFloat);
```

Ini berarti kita mendeklarasikan fungsi bernama fungsiku yang juga akan mengembalikan nilai bertipe integer dan selain itu juga menerima 2 buah nilai yang satu bernama nilaiInt bertipe int dan yang lainnya adalah bernama nilaiFloat bertipe float. Variabel-variabel penerima nilai ini disebut parameter formal, daftar nilai-nilai yang diterima oleh fungsi ini dinamakan parameter list. Pada contoh di atas, paremeter list tersebut adalah : nilaiInt yaitu sebuah variabel bertipe int dan nilaiFloat yaitu sebuah variabel bertipe float.

Ketika kita mengirimka nilai ke dalam suatu fungsi, yaitu ketika memanggil fungsi sambil menuliskan nilai yang dikirim di dalam tanda kurung, parameter ini dinamakan parameter aktual atau argumen. Sebagai contoh misalnya :

```
1  Hasil = fungsiku(10, 12.5);
```

Tampak bahwa nilai 10 (bertipe int) dan nilai 12.5 (bertipe float) dikirim sebagai parameter aktual atau argumen, tipe-tipe data dari parameter aktual ini harus sesuai dengan tipe-tipe data yang dideklarasikan pada parameter formal. Pada contoh ini

nilai 10 dikirim ke parameter formal pertama dan nilai 12.5 dikirim ke parameter formal kedua dan keduanya sudah sesuai dengan tipe data yang dideklarasikan pada fungsi `fungsiku()`.

Pengiriman parameter ke suatu fungsi dapat dilakukan dengan dua cara, yaitu yang disebut pengiriman secara nilai (by value) atau pengiriman secara acuan (by reference). Pada pengiriman secara nilai, yang dikirimkan adalah nilai (value) dari parameter tersebut, jadi pada waktu memanggil fungsi, parameter dapat langsung diisi suatu nilai tidak harus menggunakan suatu variabel, sedangkan pengiriman secara acuan yang dikirimkan adalah alamat dari variabel yang menyimpan nilai yang dikirmkan tersebut.

Hasil dari suatu fungsi dapat diperoleh dari nilai kembaliannya (return) atau dengan variabel global. Seperti contoh pada Contoh 4, hasil proses dari suatu fungsi tersebut dapat diperoleh karena variabel yang dipakai dalam fungsi bersifat global. Selain dengan cara tersebut di atas, hasil dari suatu fungsi dapat juga diperoleh dari parameter aktual yang dikirimkan ke parameter formal, karena parameter formal seolah-olah akan mengirimkan kembali nilai hasil proses dalam fungsi. Pengiriman parameter yang seolah-olah akan mengirimkan kembali nilai hasil proses dalam fungsi ini dinamakan pengiriman parameter secara acuan (pass by reference). Lebih jauh mengenai pengiriman parameter secara acuan ini akan dibahas pada Bab 5 yaitu mengenai Pointer dan References.

Parameter Default

Pada pembahasan sebelumnya, sudah dijelaskan bahwa untuk setiap parameter formal yang telah dideklarasikan pada prototype, harus mendapatkan nilai yang dikirim pada saat pemanggilan fungsi melalui parameter aktual bahkan tipe data dari parameter aktual tersebut harus sesuai dengan tipe data yang dideklarasikan pada parameter formal.

Sebenarnya dengan memberikan nilai default yang dinamakan default parameter, suatu parameter formal bisa mempunyai suatu nilai default ketika tidak ada nilai yang diterima dari parameter aktual. Misalnya deklarasi prototype seperti berikut :

```
1  int fungsiku(int nilaiInt = 10);
```

Ini berarti, `fungsiku()` akan mengembalikan suatu nilai bertipe int dan menerima nilai parameter bertipe int, jika tidak ada nilai yang diterima maka akan

digunakan nilai default yaitu 10. Karena nama parameter tidak diwajibkan pada prototype, maka prototype tersebut juga boleh ditulis :

```
int fungsiku(int = 10);
```

Pemakaian parameter default ini tidak mengubah definisi fungsi, header dari definisi fungsi tersebut tetap seperti berikut:

```
int fungsiku(int x);
```

Jika pemanggilan fungsi `fungsiku()` tidak disertai parameter aktual maka kompiler akan memberikan nilai default 10 pada x. Seperti sudah dijelaskan pada contph 1, nama dari default parameter tidak harus sama dengan nama pada header definisi fungsi, nilai default dikerjakan berdasarkan posisi parameter bukan nama parameter.

Semua parameter fungsi dapat diberikan nilai default, dengan syarat jika tidak ada nilai default untuk parameter di kanannya maka parameter tersebut tidak boleh diberikan nilai default. Misalnya jika prototype suatu fungsi adalah sepoerti berikut:

```
int fungsiku(int a, int b, int c);
```

Berarti kita hanya boleh memberikan nilai default untuk b jika kita telah memberikan nilai default untuk c. Nilai default untuk a hanya boleh diberikan jika kita telah memberikan nilai default untuk b dan c.

Contoh 6. Default Parameter.

Buka Qt Creator dan buat project Qt Console Application baru dengan nama contoh 2, kemudian tulis kode berikut.

```
#include <QtCore/QCoreApplication>
#include <iostream>

int volume(int,int=1,int=1); /*prototype*/

int main(int argc, char *argv[])
{
        using namespace std;
        QCoreApplication a(argc, argv);
        int panjang,lebar,tinggi;
        panjang = 10;
```

```
12          lebar = 15;
13          tinggi = 25;
14
15  /*--- menghitung volume ---*/
16          cout << "Volume 1 --> " << volume(panjang,lebar,tinggi)<< endl;
17          cout << "Volume 2 --> " << volume(panjang,lebar)<< endl;
18          cout << "Volume 3 --> " << volume(panjang)<< endl;
19          return a.exec();
20  }
21
22  /*---Fungsi volume---*/
23  int volume(int p, int l, int t)
24  {
25          return p * l * t;
26  }
```

Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, outputnya adalah sebagai berikut.

```
Volume 1 -> 3750 Volume 2 -> 150 Volume 3 -> 10
```

Analisa Program:

- Dari contoh program ini, Volume 1 dihasilkan dari 10 x 15 x 25 karena semua parameter formal menerima nilai, maka hasilnya 3750.
- Dari contoh program ini, Volume 1 dihasilkan dari 10 x 15 x 1 karena parameter formal ketiga tidak menerima nilai, maka hasilnya 150.
- Dari contoh program ini, Volume 1 dihasilkan dari 10 x 1 x 1 karena parameter formal kedua dan ketiga tidak menerima nilai, maka hasilnya 10.

5. Pointer dan References

Agenda

Pada chapter ini kita akan membahas beberapa topik yang berhubungan dengan pointer dan reference yaitu:

- Penggunaan Pointer.
- Pointer dan Array.
- Mengalokasikan memory dengan keyword 'new'.
- Penggunaan Reference.

Apa itu Pointer?

Pointer adalah variabel yang dapat menyimpan alamat memory. Untuk dapat memahami pointer lebih jauh anda perlu mengenal sedikit tentang memory komputer.

Memory Komputer

Memory Komputer dibagi menjadi beberapa lokasi memory yang berurutan dan mempunyai nomor tertentu. Setiap variabel akan disimpan di lokasi yang unik dalam memory yang disebut alamat memory (memory address). Contoh pada gambar dibawah ini menunjukan variabel dengan nama umur yang bertipe unsigned long.

Setiap lokasi dalam memory dapat menyimpan data dengan ukuran 1 byte (8 bit), untuk menympan data bertipe unsigned long dibutuhkan memory dengan ukuran 4 bytes (32 bit). Dari contoh diatas byte pertama dari variabel umur disimpan pada alamat memory 102, maka alamat memory dari variabel umur adalah 102.

Mengambil Alamat Memory dari Variabel

Tiap komputer mempunyai skema yang berbeda untuk penomoran memory, sebagai programmer anda tidak perlu tahu skema alamat dalam memory untuk menyimpan variabel karena kompiler akan melakukan pekerjaan tersebut untuk

anda. Jika anda ingin mengetahui pada alamat memory yang mana variabel anda disimpan maka anda dapat menggunakan operator address-of (&).

Contoh 1. Menampilkan alamat memory menggunakan address-of operator.

Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 1, kemudian tulis kode berikut.

```
#include <QtCore/QCoreApplication>
#include <iostream>

int main(int argc, char *argv[])
{
        using namespace std;
        QCoreApplication a(argc, argv);

        unsigned short bil1 =20;
        ulong bil2 = 200000;
        long bil3 = -670000;

        cout << "bil1 = " << bil1 << " address = " << &bil1 << endl;
        cout << "bil2 = " << bil2 << " address = " << &bil2 << endl;
        cout << "bil3 = " << bil3 << " address = " << &bil3 << endl;
        return a.exec();
}
```

Kemudian jalankan kode diatas dengan menekan tombol <kbd>Ctrl+R</kbd>, outputnya adalah sebagai berikut.

```
bil1 = 20 address = 0x28fe96 bil2 = 200000 address = 0x28fe90 bil3 = -670000 address = 0x28fe8c
```

Analisa Program :

- Pada program diatas operator address of (&) digunakan untuk mengetahui alamat memory tempat variabel bil disimpan.

- Ketika anda mendeklarasikan variabel dengan tipe tertentu maka compiler akan menentukan ukuran dari memory yang diperlukan untuk menyimpan data dan secara otomatis menetapkan alamat memory dimana variabel tersebut akan disimpan.

Menyimpan Alamat Variabel pada Pointer

Setiap variabel mempunyai alamat, bahkan jika anda tidak tau secara spesifik alamat memory dari variabel tersebut, anda tetap dapat menyimpan alamat variabel kedalam pointer. Sebagai contoh untuk mendeklarasikan pointer yang menunjuk ke variabel tertentu yang bertipe integer, anda dapat menuliskannya sebagai berikut.

```
1  int *pBil = 0;
```

Statement diatas bertujuan untuk membuat pointer variabel yang menunjuk ke alamat variabel bertipe integer. Tanda bintang (*) digunakan untuk mendeklarasikan variabel pointer.

Pada contoh diatas pemberian nilai 0 berarti anda mendeklarasikan null pointer, setiap pointer ketika dideklarasikan harus diinisialiasi nilainya. Jika anda belum tahu alamat yang akan ditunjuk oleh pointer maka anda dapat memberi nilai 0. Pointer yang tidak diinisialisasi disebut dengan wild pointer karena bisa menunjuk ke alamat manapun, wild pointer harus dihindari karena sangat berbahaya.

TIPS

Selalu lakukan inisialisasi ketika membuat pointer.

Memberi Nama Pointer

Karena pointer juga merupakan variabel maka aturan penamaan pointer juga sama dengan aturan penamaan variabel biasa. Kesepakatan tidak tertulis programmer dalam pemberian nama pointer adalah diawali dengan huruf p misal (pBil, pUmur).

Contoh dibawah ini adalah cara deklarasi dan inisialiasi pointer.

```
int *pBil = 0; //membuat variabel pointer dan inisialisasi null
int bil = 12; //deklarasi variabel
pBil = &bil; //menunjuk ke alamat variabel bil
```

Pada baris yang ketiga dapat anda lihat bahwa pointer pBil menunjuk ke alamat dari variabel bil, tanda address-of (&) digunakan untuk mengambil alamat memory dari variabel bil. Anda dapat menuliskan statement diatas dengan lebih singkat sebagai berikut:

```
int bil = 12; //deklarasi variable
int *pBil = &bil; //menunjuk ke alamat variabel bil
```

Mengambil Nilai dari Variabel

Mengambil nilai dari variabel dengan menggunakan pointer disebut dengan `indirection` karena anda secara tidak langsung mengakses nilai dari variabel melalui `pointer`. Sebagai contoh anda dapat mengakses nilai dari variabel bil diatas menggunakan pointer `pBil`.

Operator *indirection ()** disebut juga dengan operator *dereferensi*, ketika pointer di dereferensi maka nilai dari variabel yang alamatnya ditunjuk oleh pointer dapat diambil.

```
int number = *pBil; //mengambil nilai variabel yg alamatnya disimpan pa\
dapointer pBil
```

Pada kode ditas dapat dilihat bahwa nilai dari `*pBil` akan sama dengan nilai bil, karena pointer `pBil` mereferensi ke alamat dimana variabel bil disimpan, maka number akan bernilai 12.

```
*pBil = 20; //nilai dari variabel bil juga akan berubah menjadi 20
```

Pada kode diatas nilai dari variabel bil akan berubah menjadi 20, karena variabel bil direferensi oleh pointer `pBil`.

Contoh 2. Memanipulasi data menggunakan Pointer

Buka Qt Creator, buat project Qt Console Application dengan nama Contoh 2. Kemudian tulis kode berikut.

```
#include <QtCore/QCoreApplication>
#include <iostream>
int main(int argc, char *argv[])
{
        using namespace std;
        QCoreApplication a(argc, argv);

        ushort umur;
        ushort *pUmur = 0;
        umur = 17;

        cout << "Umur : " << umur << endl;
        pUmur = &umur;
        cout << "pUmur : " << *pUmur << endl;
        cout << "Merubah nilai pUmur.." << endl;

        *pUmur = 28;
        cout << "Umur : " << umur << endl;
        cout << "pUmur : " << *pUmur << endl;
        cout << "Merubah nilai umur.." << endl;

        umur = 30;
        cout << "Umur : " << umur << endl;
        cout << "pUmur : " << *pUmur << endl;
        return a.exec();
}
```

Tekan <kbd>Ctrl+R</kbd> untuk menjalankan kode diatas, outputnya adalah sebagai berikut.

```
Umur : 17
pUmur : 17
Merubah nilai pUmur..
Umur : 28
pUmur : 28
```

```
Merubah nilai umur..
Umur : 30
pUmur : 30
```

Analisa Program:

- Pada program diatas pointer `pUmur` mereferensi/menunjuk ke alamat dimana nilai variabel umur disimpan.
- Untuk mengakses nilai dari variabel umur lewat pointer dapat menggunakan dereference operator (*).
- Ketika nilai dereference pointer `*pUmur` diubah menjadi 28, maka akan mempengaruhi nilai pada variabel umur yang akan menjadi 28 juga.
- Ketika nilai variabel umur diubah menjadi 30, dan anda mengakses nilainya dengan menggunakan pointer `*pUmur` maka nilainya juga akan berubah menjadi 30.

Mengganti alamat yang direferensi oleh Pointer

Anda juga dapat mengganti alamat variabel yang direferensi oleh pointer tertentu tanpa harus mengetahui nilai dari variabel tersebut.

Contoh 3. Mengganti alamat yang di referensi oleh pointer

Buat project Qt Console Application baru, beri nama Contoh 3, kemudian tulis kode berikut

```
#include <QtCore/QCoreApplication>
#include <iostream>
int main(int argc, char *argv[])
{
        using namespace std;
        QCoreApplication a(argc, argv);
        ushort umur1 = 17, umur2 = 28;
        ushort *pUmur = &umur1; //ganti referensi
        cout << "umur1 : " << umur1 << " alamat : " << &umur1 << endl;
        cout << "pUmur : " << *pUmur << " alamat : " << pUmur << endl;
        pUmur = &umur2;
        cout << "umur2 : " << umur2 << " alamat : " << &umur2 << endl;
        cout << "pUmur : " << *pUmur << " alamat : " << pUmur << endl;
        return a.exec();
}
```

Tekan Ctrl+R untuk menjalankan program diatas, outputnya adalah sebagai berikut.

```
umur1 : 17 alamat : 0x28fe92
pUmur : 17 alamat : 0x28fe92
umur2 : 28 alamat : 0x28fe90
pUmur : 28 alamat : 0x28fe90
```

Analisa:

- Pada program diatas dapat dilihat bahwa pertama kali pointer pUmur mereferensi pada alamat variabel `umur1`, sehingga ketika dicetak nilai dari `*pUmur` sama dengan nilai variabel `umur1`.
- Anda dapat mengganti referensi dari `pUmur` yang tadinya menunjuk ke alamat variabel `umur1` menjadi menunjuk ke alamat variabel `umur2`, sehingga ketika `*pUmur` dicetak menghasilkan nilai yang sama dengan variabel `umur2`.

Pointer dan Array

Pada C++ nama dari array adalah konstan pointer yang menunjuk pada elemen pertama dari array, misal untuk deklarasi array berikut

```
int Numbers[5];
```

Numbers adalah pointer yang menunjuk alamat `&Numbers[0]` yang merupakan alamat dari elemen pertama array diatas.

Anda dapat menggunakan nama array sebagai konstan pointer, misalnya Numbers+3 adalah cara penulisan untuk mengakses pointer yang menunjuk ke `Numbers[3]`.

Contoh 4. Pointer dan Array

Buat project Qt Console Application dengan nama Contoh 4, kemudian tulis kode berikut.

```
#include <QtCore/QCoreApplication>
#include <iostream>
int main(int argc, char *argv[])
{
        using namespace std;
        QCoreApplication a(argc, argv);

        const int ARRAY_LENGTH = 5;
        int numbers[ARRAY_LENGTH] = {100,200,222,111,777};

        //mengakses alamat pertama dari array (numbers[0])
        cout << "Alamat numbers[0] : " << numbers << endl;

        //mengakses nilai dari elemen pertama array (numbers[0])

        cout << "Nilai numbers[0] : " << *numbers << endl;

        //mengakses alamat numbers[4]
        cout << "Alamat numbers[4] : " << numbers+4 << endl;

        //mengakses nilai dari numbers[4]
        cout << "Nilai numbers[4] : " << *(numbers+4) << endl;
```

```
23          const int *pNumber = numbers;
24
25          //menggunakan pointer untuk mencetak semua elemen array
26          for(int i=0; i<ARRAY_LENGTH; i++)
27          {
28                  cout << "numbers[" << i << "] = " << *(pNumber+i) << endl;
29          }
30
31          return a.exec();
32  }
```

Tekan Ctrl+R untuk menjalankan program, outputnya adalah sebagai berikut.

```
Alamat numbers[0] : 0x28fe78
Nilai numbers[0] : 100
Alamat numbers[4] : 0x28fe88
Nilai numbers[4] : 777
numbers[0] = 100
numbers[1] = 200
numbers[2] = 222
numbers[3] = 111
numbers[4] = 777
```

Keterangan:

Nama dari array numbers merupakan konstan pointer yang menunjuk alamat element pertama pada array (`numbers[0]`), jadi jika anda ingin mengetahui nilai dari elemen pertama array anda dapat menggunakan dereference operator `*numbers`.

Anda dapat menggunakan nama array `numbers+4` untuk menunjuk ke alamat elemen `numbers[4]`, untuk menampilkan nilai `numbers[4]` anda dapat menuliskan `*(numbers+4)`.

Kapan kita menggunakan pointer?

Setelah kita mempelajari cara penggunaan pointer sekarang kita akan melihat kapan pointer biasa digunakan dalam pemrograman.

- Pengaturan data pada free store / heap memory.
- Mengakses class member dan data function.
- Passing variabel dengan reference pada function.

Mengalokasikan tempat dengan keyword ‘new’

Anda dapat mengalokasikan memory pada free store / heap memory dengan menggunakan keyword ‘new’ diikuti dengan tipe data dari objek yang akan anda simpan sehingga compiler dapat mengetahui berapa banyak memory yang dibutuhkan untuk menyimpan data tersebut. Contoh penggunaan keyword ‘new’ dapat dilihat pada kode berikut:

```
1  //mengalokasikan memory di heap untuk menyimpan data integer
2  int *pBil = new int;
3  //nilai 19 akan disimpan di heap yg sudah dialokasikan
4  *pBil = 19;
```

Membersihkan memory dengan keyword ‘delete’

Ketika anda sudah selesai menggunakan objek yang ada di memory, anda harus mengosongkan kembali memory tersebut agar dapat digunakan kembali. Anda dapat menggunakan keyword ‘delete’ untuk mengembalikan memory yang anda gunakan ke heap / free store.

Penting untuk anda ketahui bahwa memory yang dialokasikan menggunakan keyword ‘new’ tidak akan dibersihkan secara otomatis, maka sebagai programmer anda harus disiplin untuk membebaskan memory yang sudah tidak digunakan.

Ketika anda menghapus memory maka pointer tetap menunjuk ke alamat memory yang sudah anda hapus, agar tidak terjadi kesalahan setelah menghapus memory anda disarankan untuk memberi nilai null (0) pada pointer.

Contoh 5. Mengalokasikan, menggunakan, dan mendelete Pointer

Buat project Qt Console Application dengan nama Contoh 5, kemudian tulis kode berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
int main(int argc, char *argv[])
{
        using namespace std;
        QCoreApplication a(argc, argv);
        int bil = 20;

        //pointer yang menunjuk ke alamat lokal
        int *pBil = &bil;
        cout << "bil : " << bil << endl;
        cout << "pBil : " << *pBil << endl;

        //mengalokasikan memory di heap untuk menyimpan data integer
        int *pHeap = new int;

        //nilai 19 akan disimpan di heap yg sudah dialokasikan
        *pHeap = 19;
        cout << "Nilai pHeap : " << *pHeap << endl;
        delete pHeap;

        pHeap = 0; //null pointer

        //mengalokasikan memory
        pHeap = new int;
        *pHeap = 100;
        cout << "Nilai pHeap : " << *pHeap << endl;
        delete pHeap;
        return a.exec();
}
```

Tekan <kbd>Ctrl+R</kbd> untuk menjalankan program, outputnya adalah sebagai berikut.

```
bil : 20
pBil : 20
Nilai pHeap : 19
Nilai pHeap : 100
```

Analisa:

- `pHeap` adalah pointer yang menunjuk ke alamat memory yang sudah di-alokasikan dengan keyword 'new', anda dapat menyimpan nilai kedalam memory yang dialokasikan dengan `*pHeap=19`
- Setelah selesai digunakan anda harus membersihkan memory dengan menggunakan keyword 'delete', jangan lupa menginisialisasi pointer dengan null (0) agar tidak terus menunjuk ke alamat memory yang sudah dihapus.

TIPS

Setelah menghapus objek di memory dengan keyword delete anda harus menginisialisasi pointer yang sudah tidak digunakan dengan nilai null (0).

Membuat objek pada heap

Selain tipe data primitive (int, float, byte, dll) anda juga dapat menyimpan data bertipe class kedalam free store / heap, misal jika anda ingin membuat objek bertipe class Mahasiswa anda dapat mendeklarasikan pointer untuk class tersebut dan mengalokasikan memory di heap untuk menyimpan objek tersebut. Sintaks penulisanya sama dengan sebelumnya.

```
1  Mahasiswa *mhs = new Mahasiswa;
```

Ketika anda menggunakan keyword 'new' untuk membuat pointer yang menunjuk ke objek maka otomatis default konstruktor dari class tersebut akan dipanggil.

Ketika anda menghapus pointer yang menunjuk ke objek dengan keyword 'delete', maka destruktor akan dipanggil, ini akan memberi kesempatan bagi programmer untuk membersihkan heap memory dari variabel yang sudah tidak digunakan.

Contoh 6. Membuat dan menghapus objek dari Heap

Buat project Qt Console Application dengan nama Contoh 6, kemudian tulis kode berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Mahasiswa
{
        public:
        Mahasiswa();
        ~Mahasiswa();
        private:
        float ipk;
};


Mahasiswa::Mahasiswa()
{
        cout << "Konstruktor dipanggil.." << endl;
        ipk=3.5;
}

Mahasiswa::~Mahasiswa()
{
        cout << "Destruktor dipanggil.." << endl;
}

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        cout << "Deklarasi object tanpa pointer " << endl;
```

```
29          Mahasiswa mhs1;
30          cout << "Mengalokasikan heap memory untuk menyimpan objek " << endl;
31          Mahasiswa *mhs2 = new Mahasiswa;
32          cout << "Delete objek di memory " << endl;
33          delete mhs2;
34          mhs2 = 0; //null pointer
35          return a.exec();
36  }
```

Tekan Ctrl+R untuk menjalankan program, outputnya adalah sebagai berikut.

```
Deklarasi object tanpa pointer
Konstruktor dipanggil..
Mengalokasikan heap memory untuk menyimpan objek
Konstruktor dipanggil..
Delete objek di memory
Destruktor dipanggil..
```

Analisa :

- Pada program diatas kita membuat class Mahasiswa yang mempunyai objek konstruktor dan destruktor.
- Pertama kali kita mendeklarasikan object mhs1 pada local variable (stack), pembuatan object ini menyebabkan konstruktor dipanggil.
- Kemudian dibuat pointer yang menunjuk ke objek di heap dengan nama mhs2, ketika objek mhs2 dibuat, objek konstruktor dipanggil. Ketika anda menghapus objek di heap menggunakan delete maka objek destruktor akan dipanggil.
- Objek desktruktor untuk mhs1 akan dipanggil ketika fungsi main berakhir.

Menggunakan const Pointer

Anda dapat menggunakan keyword ‘const’ pada pointer dengan menuliskannya sebelum atau sesudah tipe data, atau keduanya. Contoh deklarasi const pointer dapat dilihat pada kode dibawah ini:

```
1 const int * pBil1;
2 int * const pBil2;
3 const int * const pBil3;
```

Tiga statement diatas memiliki pengertian yang berbeda, yaitu:

- Statement pertama : `pBil1` adalah pointer yang menunjuk ke konstan integer, jadi nilai yang ditunjuk oleh pointer tidak dapat diubah.
- Statement kedua : `pBil2` adalah konstan pointer yang menunjuk ke variabel integer, nilai variabel integer dapat diubah namun pBil2 tidak dapat menunjuk ke variabel lain.
- Statement ketiga : `pBil3` adalah konstan pointer yang menunjuk ke konstan variabel bertipe integer, nilai variabel tidak dapat diubah dan pointer `pBil3` tidak dapat menunjuk ke variabel lain.

TIPS

Lihat letak penulisan keyword const, jika sebelum tipe data maka nilai konstan, jika setelah tipe data maka alamat pointer yang konstan.

Apa itu Reference

Pada pembahasan sebelumnya kita membahas penggunaan pointer untuk mengakses objek secara tidak langsung (indirect). Fungsi reference mirip seperti pointer namun dengan penulisan yang relatif lebih mudah.

Reference adalah alias, ketika anda membuat reference anda menginisialisasi dengan nama dari objek yg dijadikan target. Reference adalah alternatif nama dari objek target, jika anda merubah reference maka objek target juga akan berubah.

Cara penulisan reference adalah menambahkan operator (&) didepan nama variabel, contohnya :

```
1 int &rBil = intBil;
```

Statement diatas dapat diartikan “rBil adalah referensi dari variabel intBil”, reference berbeda dengan variabel biasa karena reference harus diinisialisasi ketika dibuat.

Contoh 7. Membuat dan Menggunakan Reference.

Buat project Qt Console Application dengan nama Contoh 7, kemudian tulis kode berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        int bil1 = 18;
        int &rBil = bil1;
        cout << "Nilai bil1 : " << bil1 << endl;
        cout << "Nilai &rBil : " << rBil << endl;

        bil1 = 19;
        cout << "Nilai bil1 : " << bil1 << endl;
        cout << "Nilai &rBil : " << rBil << endl;

        rBil = 33;
        cout << "Nilai bil1 : " << bil1 << endl;
        cout << "Nilai &rBil : " << rBil << endl;
        cout << "Menampilkan alamat memory :" << endl;
        cout << "&bil1 : " << &bil1 << endl;
        cout << "&rBil : " << &rBil << endl;
        return a.exec();
}
```

Tekan Ctrl+R untuk menjalankan program, outputnya adalah sebagai berikut.

```
Nilai bil1 : 18
Nilai &rBil : 18
Nilai bil1 : 19
Nilai &rBil : 19
```

```
Nilai bil1 : 33
Nilai &rBil : 33
Menampilkan alamat memory :
&bil1 : 0x28fe90
&rBil : 0x28fe90
```

Analisa:

- Pertama kita mendeklarasikan referensi `rBil=bil1`, maka ketika dicetak nilai `rBil` sama dengan nilai variabel `bil1` karena `rBil` merupakan reference / alias dari `bil1`.
- Ketika variabel `bil1` nilainya dirubah menjadi `19`, maka otomatis nilai dari `rBil` juga berubah menjadi `19`.
- Demikian pula ketika `rBil` nilainya dirubah menjadi `33`, maka nilai dari `bil1` juga ikut berubah.
- Anda juga dapat menampilkan alamat memory dari variabel dan variabel reference dengan menambahkan keyword (&) didepan variabel.

Re-assign Reference Variable

Variabel reference tidak dapat di re-assign (ditetapkan ulang). Agar lebih jelas perhatikan contoh dibawah ini:

Contoh 8. Re-assign Reference Value

Buat project Qt Console Application dengan nama Contoh 8, kemudian tulis kode berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>

int main(int argc, char *argv[])
{
        using namespace std;
        QCoreApplication a(argc, argv);

        int bil = 14;
        int &rBil = bil;
        cout << "rBil : " << rBil << endl;

        int bil2 = 19;
        rBil = bil2; //tebak hasilnya !
        cout << "rBil : " << rBil << endl;
        cout << "bil : " << bil << endl;
        return a.exec();
        }
```

Tekan Ctrl+R untuk menjalankan program, outputnya adalah sebagai berikut.

```
rBil : 14
rBil : 19
bil : 19
```

Analisa:
Variabel reference `rBil` yang sudah diinisialisasi dengan `bil1` coba di re-assign dengan `bil2` dan gagal, karena `rBil=bil2` tidak menjadikan referensinya berubah tetapi nilai `bil2` yang mengganti nilai `rBil` dan `bil1`.

Passing function argument dengan reference

Pada chapter sebelumnya tentang `function`, kita sudah membahas beberapa keterbatasan dari `function` diantaranya, argument hanya dapat di-*passing by value*, dan return statement hanya dapat mengembalikan satu nilai saja.

Passing reference value pada `function` dapat mengatasi masalah diatas. Contoh dibawah ini akan menunjukan perbedaan penggunaan passing by value dan passing by reference (dengan pointer dan variaabel reference).

Contoh 9. Passing by Value.

Buat project Qt Console Application dengan nama Contoh 9, kemudian tulis kode berikut:

```
1  #include <QtCore/QCoreApplication>
2  #include <iostream>
3
4  using namespace std;
5
6  void Tukar(int x,int y);
7
8  int main(int argc, char *argv[])
9  {
10         QCoreApplication a(argc, argv);
11         int x=12, y=21;
12         cout << "Pada main, sebelum ditukar x=" << x << ", y=" << y << endl;
13         Tukar(x,y);
14         cout << "Pada main, setelah ditukar x=" << x << ", y=" << y << endl;
15         return a.exec();
16 }
17
18 void Tukar(int x,int y)
19 {
20         int tampung;
21         cout << "Pada fungsi, sebelum ditukar, x=" << x << ", y=" << y << endl;
22         tampung = x;
23         x=y;
24         y=tampung;
```

```
25          cout << "Pada fungsi, Setelah ditukar, x=" << x << ", y=" << y << endl;
26  }
```

Tekan Ctrl+R untuk menjalankan program, outputnya adalah sebagai berikut.

```
Pada main, sebelum ditukar x=12, y=21
Pada fungsi, sebelum ditukar, x=12, y=21
Pada fungsi, Setelah ditukar, x=21, y=12
Pada main, setelah ditukar x=12, y=21
```

Keterangan:

Pada kode diatas dapat dilihat bahwa *passing by value* ke fungsi `Tukar()` tidak akan mempengaruhi variabel `x` dan `y` yang ada pada fungsi main, dan hanya berpengaruh pada scope fungsi `Tukar()`. saja.

Contoh 10. Passing by reference dengan pointer

Buat project Qt Console Application dengan nama Contoh 10, kemudian tulis kode berikut:

```
1   #include <QtCore/QCoreApplication>
2   #include <iostream>
3
4   using namespace std;
5
6   void Tukar(int *x, int *y);
7
8   int main(int argc, char *argv[])
9   {
10          QCoreApplication a(argc, argv);
11          int x=12, y=21;
12          cout << "main func, x=" << x << ", y=" << y << endl;
13          Tukar(&x,&y);
14          cout << "main func, x=" << x << ", y=" << y << endl;
15          return a.exec();
16  }
```

```
17
18  void Tukar(int *x, int *y)
19  {
20          int tampung;
21          cout << "Pada fungsi, sebelum ditukar x=" << *x << ",y=" << *y << endl;
22          tampung = *x;
23          *x = *y;
24          *y = tampung;
25          cout << "Pada fungsi, sesudah ditukar x=" << *x << ",y=" << *y << endl;
26  }
```

Tekan Ctrl+R untuk menjalankan program, outputnya adalah sebagai berikut.

```
main func, x=12, y=21
Pada fungsi, sebelum ditukar x=12,y=21
Pada fungsi, sesudah ditukar x=21,y=12
main func, x=21, y=11
```

Analisa:

Pada kode diatas kita melakukan passing by reference untuk passing parameter ke fungsi `Tukar()` menggunakan pointer, dapat anda lihat bahwa setelah fungsi `Tukar()` dijalankan variabel `x` dan `y` di main function nilainya sudah berhasil ditukar.

Contoh 11. Menjalankan fungsi Tukar() dengan reference

Buat project Qt Console Application dengan nama Contoh 11, kemudian tulis kode berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>

using namespace std;

void Tukar(int &x, int &y);

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);

        int x=12, y=21;
        cout << "main func, sebelum ditukar x=" << x << ", y=" << y << endl;
        Tukar(x,y);
        cout << "main func, setelah ditukar x=" << x << ", y=" << y << endl;
        return a.exec();
}

void Tukar(int &x, int &y)
{
        int tampung;
        cout << "Pada function, sebelum ditukar x=" << x << ", y=" << y << end\
l;
        tampung = x;
        x = y;
        y = tampung;
        cout << "Sesudah function, sebelum ditukar x=" << x << ", y=" << y << \
endl;
}
```

Tekan Ctrl+R untuk menjalankan program, outputnya adalah sebagai berikut.

```
main func, sebelum ditukar x=12, y=21
Pada function, sebelum ditukar x=12, y=21
Sesudah function, sebelum ditukar x=21, y=12
```

```
    main func, setelah ditukar x=21, y=11
```

Analisa:

Pada kode diatas kita juga berhasil menukar nilai x dan y menggunakan fungsi tukar sama dengan kode sebelumnya. Ini karena passing parameter menggunakan variabel reference.

Function yang mengembalikan beberapa nilai

Seperti yang sudah kita bahas sebelumnya bahwa salah satu keterbatasan dari function adalah hanya dapat mengembalikan satu nilai saja. Bagaimana jika anda ingin mengembalikan lebih dari satu nilai pada function? Untuk memecahkan masalah tersebut anda dapat menggunakan function pass by reference. Karena function pass by reference dapat memanipulasi objek asli. Agar lebih jelas coba kerjakan contoh dibawah ini.

Contoh 12. Mengembalikan beberapa nilai dengan pointer

Buat project Qt Console Application dengan nama Contoh 12, kemudian tulis kode berikut:

```
1  #include <QtCore/QCoreApplication>
2  #include <iostream>
3
4  using namespace std;
5
6  int Hitung(int number, int *pLuasPersegi, int *pVolumeKubus);
7
8  int main(int argc, char *argv[])
9  {
10         QCoreApplication a(argc, argv);
11         int number, pLuasPersegi, pVolumeKubus;
12         short error;
13         cout << "Masukan number : ";
14         cin >> number;
```

```
15          error = Hitung(number,&pLuasPersegi,&pVolumeKubus);
16          if(!error)
17          {
18                  cout << "Number : " << number << endl;
19                  cout << "pLuasPersegi : " << pLuasPersegi << endl;
20                  cout << "pVolumeKubus : " << pVolumeKubus << endl;
21          }
22          else
23                  cout << "Terjadi Error !! ";
24          return a.exec();
25          }
26          int Hitung(int number, int *pLuasPersegi, int *pVolumeKubus)
27          {
28          short status;
29          if(number > 0)
30          {
31                  *pLuasPersegi = number * number;
32                  *pVolumeKubus = number * number * number;
33                  status = 0;
34          }
35          else
36          {
37                  status = 1;
38          }
39          return status;
40  }
```

Tekan Ctrl+R untuk menjalankan program, outputnya adalah sebagai berikut.

```
Masukan number : 5
Number : 5
pLuasPersegi : 25
pVolumeKubus : 125
```

Analisa:

- Inputan untuk variabel number harus lebih besar dari 0, jika tidak program akan menghasilkan pesan error.
- Dapat dilihat bahwa function `Hitung()` mengembalikan 3 nilai yaitu : nilai kembalian dari `function` itu sendiri yang bertipe `integer`, `pLuasPersegi`, dan `pVolumeKubus` yang merupakan parameter bertipe pointer.
- `pLuasPersegi` dan `pVolumeKubus` nilainya dapat bukan karena nilai kembalian dari function, tapi karena parameter by reference dari function yang berupa pointer, sehingga ketika nilai `pLuasPersegi` dan `pVolumeKubus` diubah di dalam function nilai variabel asli di main function juga berubah.

Contoh 13. Mengembalikan beberapa nilai dengan reference variabel

Buat project Qt Console Application dengan nama Contoh 13, kemudian tulis kode berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>

using namespace std;

enum ERR_STATUS {SUCCESS, ERROR};
ERR_STATUS Hitung(int,int &,int &);

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        ERR_STATUS status;
        int number, rLuasPersegi, rVolumeKubus;
        cout << "Masukan number : ";
        cin >> number;
        status = Hitung(number,rLuasPersegi,rVolumeKubus);
        if(status==SUCCESS)
        {
                cout << "Number : " << number << endl;
                cout << "pLuasPersegi : " << rLuasPersegi << endl;
                cout << "pVolumeKubus : " << rVolumeKubus << endl;
```

```
22          }
23          else
24                  cout << "Terjadi Error !!";
25          return a.exec();
26          }
27          ERR_STATUS Hitung(int number, int &rLuasPersegi, int &rVolumeKubus)
28          {
29          ERR_STATUS status;
30          if(number > 0)
31          {
32                  rLuasPersegi = number * number;
33                  rVolumeKubus = number * number * number;
34                  status = SUCCESS;
35                  }
36          else
37                  status = ERROR;
38          return status;
39  }
```

Tekan Ctrl+R untuk menjalankan program, outputnya adalah sebagai berikut.

```
Masukan number : 5
Number : 5
pLuasPersegi : 25
pVolumeKubus : 125
```

Analisa:

- Hasil program diatas sama dengan Contoh 12 sebelumnya, namun perbedaannya adalah program diatas menggunakan parameter reference pada function `Hitung()` sehingga ketika variabel `rLuasPersegi` dan `rVolumeKubus` pada function diubah nilainya maka variabel di function main juga ikut berubah.
- Keyword enum digunakan untuk membuat objek enumerasi untuk mempermudah pembacaan program.

Passing By Reference untuk Efisiensi

Setiap kali anda melakukan passing objek by value, copy dari objek tersebut akan dibuat kembali. Untuk tipe data objek yang besar (struct atau class yang dibuat sendri oleh user) ini akan menurunkan performa dari program. Untuk melakukan passing parameter objek melalui function disarankan menggunakan reference pada objek.

Contoh 14. Passing Object By Value

Buat project Qt Console Application dengan nama Contoh 14, kemudian tulis kode berikut:

```
1  #include <QtCore/QCoreApplication>
2  #include <iostream>
3  using namespace std;
4  class Mahasiswa
5  {
6          public:
7          Mahasiswa();
8          Mahasiswa(Mahasiswa&);
9          ~Mahasiswa();
10 };
11
12 Mahasiswa::Mahasiswa()
13 {
14         cout << "Memanggil Mahasiswa Konstruktor " << endl;
15 }
16
17 Mahasiswa::Mahasiswa(Mahasiswa &)
18 {
19         cout << "Memanggil Copy Konstruktor " << endl;
20 }
21
22 Mahasiswa::~Mahasiswa()
23 {
24         cout << "Memanggil Mahasiswa Destruktor " << endl;
25 }
26
```

```
27 Mahasiswa FunctionMhs(Mahasiswa objMhs);
28 int main(int argc, char *argv[])
29 {
30         QCoreApplication a(argc, argv);
31         cout << "Membuat object mahasiswa " << endl;
32         Mahasiswa objMhs1;
33         FunctionMhs(objMhs1);
34         return a.exec();
35         }
36         Mahasiswa FunctionMhs(Mahasiswa objMhs)
37         {
38                 cout << "Mengembalikan FunctionMhs .." << endl;
39         return objMhs;
40 }
```

Tekan Ctrl+R untuk menjalankan program, outputnya adalah sebagai berikut.

```
Membuat object mahasiswa
Memanggil Mahasiswa Konstruktor
Memanggil Copy Konstruktor
Mengembalikan FunctionMhs ..
Memanggil Copy Konstruktor
Memanggil Mahasiswa Destruktor
Memanggil Mahasiswa Destruktor
```

Analisa:

- Dapat kita lihat diatas bahwa *passing object by value* tidak efisien karena setiap kali function dipanggil dan mengembalikan nilai harus melakukan copy terhadap objek objMhs1.
- Hal ini dapat dilihat dari output yang dihasilkan, copy konstruktor dipanggil sebanyak 2 kali, saat pemanggilan function dan pengembalian nilai function.
- Cara yang lebih efisien akan dibahas pada contoh program selanjutnya.

Contoh 15. Passing Object By Reference

Buat project Qt Console Application dengan nama Contoh 15, kemudian tulis kode berikut:

```
#include <QtCore/QCoreApplication>

#include <iostream>

using namespace std;

class Mahasiswa
{
        public:
        Mahasiswa();
        Mahasiswa(Mahasiswa&);
        ~Mahasiswa();
};

Mahasiswa::Mahasiswa()
{
        cout << "Memanggil Mahasiswa Konstruktor " << endl;
}

Mahasiswa::Mahasiswa(Mahasiswa &)
{
        cout << "Memanggil Copy Konstruktor " << endl;
}

Mahasiswa::~Mahasiswa()
{
        cout << "Memanggil Mahasiswa Destruktor " << endl;
}

Mahasiswa &FunctionMhs(Mahasiswa &objMhs);
int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
```

```
34          cout << "Membuat object mahasiswa " << endl;
35          Mahasiswa objMhs1;
36          FunctionMhs(objMhs1);
37          return a.exec();
38  }
39
40  Mahasiswa &FunctionMhs(Mahasiswa &objMhs)
41  {
42          cout << "Mengembalikan FunctionMhs .." << endl;
43          return objMhs;
44  }
```

Tekan Ctrl+R untuk menjalankan program, outputnya adalah sebagai berikut.

```
Membuat object mahasiswa
Memanggil Mahasiswa Konstruktor
Mengembalikan FunctionMhs ..
```

Analisa:

- Dengan menambahkan reference pada function dan parameter yang dikirimkan, performa aplikasi anda dapat lebih efektif karena objek tidak perlu dicopy ketika function dijalankan dan saat function tersebut mengembalikan nilai.
- Output yang dihasilkan lebih sedikit karena tidak perlu memanggil copy objek konstruktor.

6. Class dan Object

Pemrograman Berorientasi Obyek

Bahasa C++ memang berbeda dengan bahasa C. Bahasa C++ memiliki keunggulan dan perubahan yang besar dibandingkan dengan bahasa C. Salah satu perubahan mendasarnya adalah bahasa C++ dibuat untuk mendukung pemrograman berorientasi obyek.

Program adalah kumpulan instruksi yang disusun dengan urutan nalar yang tepat untuk menyelesaikan suatu persoalan. Dalam pembuatan program, pemrogram mempunyai cara pandang terhadap eksekusi sebuah program yang disebut sebagai paradigma pemrograman. Sebagai contoh dalam paradigma pemrograman berorientasi objek (OOP), pemrogram bisa melihat bahwa sebuah program adalah kumpulan objek yang saling berinteraksi, sedangkan dalam paradigma pemrograman terstruktur, pemrogram melihat bahwa sebuah program adalah suatu urutan instruksi yang dieksekusi secara berurutan.

Pemrograman Berorientasi Objek merupakan suatu paradigma pemrograman yang sudah sangat populer, meskipun metodologi pemrograman ini lebih baru dibandingkan dengan metodologi pemrograman terstruktur, tidak berarti pemrograman terstruktur harus ditinggalkan, karena sebenarnya secara internal pemrograman berorientasi objek juga dibangun dengan teknik pemrograman terstruktur, selain itu logika pemanipulasian objek kadang-kadang diekspresikan dengan pemrograman terstruktur juga. Pemrograman berorientasi obyek memiliki kelebihan, yaitu:

- Membuat suatu representasi teknis sedekat mungkin dengan pandangan konseptual dari dunia nyata.
- membuat kerangka analisis dan spesifikasi yang stabil.
- Memudahkan pengembangan dan perubahan programmer.

Tanpa kita sadari, dunia ini penuh dengan obyek. Obyek yang ada misalnya: sepeda, matahari, rumah, orang, anjing, topi, meja, dan masih banyak lagi. Ciri khas dari masing-masing obyek yang ada adalah dapat dilihat dan dapat digunakan. Setiap obyek pada dunia nyata juga memiliki ciri khas yang membedakannya dengan

obyek lain terutama yang berbeda jenis. Sebagai contoh, obyek meja tulis, meja makan, dan meja belajar memiliki ciri yang sama, yaitu memiliki berat, memiliki kaki meja, memiliki warna meja, dan lain-lain. Kemudian obyek-obyek meja tersebut juga dapat menerima dan dilakukan operasi/kegiatan terhadapnya, misalnya kegiatan mengubah warna meja, kegiatan memotong atau menambah kaki meja dan lain-lain. Jadi sebuah obyek memiliki sifat yang melekat padanya dan memiliki hal yang dapat dikenakan atau dilakukannya. Antara obyek meja dengan obyek mobil memiliki perbedaan yang sangat besar.

Pemrograman berorientasi objek adalah suatu cara yang dipakai untuk mengorganisasikan program kedalam suatu komponen logis (kelas), yang pada saat akan digunakan harus diinstansiasi menjadi sebuah objek (yaitu sebuah instan/ instance dari sebuah kelas). Sebuah kelas mempunyai anggota (data) dan metoda (fungsi yang bekerja untuk data tersebut), dengan kata lain Pemrograman Berorientasi Objek mengemas data (variabel / data member) dan prosedur (fungsi / function member / method) dalam sebuah objek sehingga kode program menjadi lebih fleksibel dan mudah dipelihara. Dalam Pemrograman Berorientasi Objek, objek yang dibuat melakukan suatu proses terhadap masukan tertentu dan mengeluarkan hasil tertentu dan pemakai tidak melihat bagaimana cara objek tersebut melakukan proses (karena program dibungkus di dalam objek).

Kelas

Kelas adalah Blue Print dari objek, yaitu prototype yang mendefinisikan variabel-variabel dan metodametoda (sub program) secara umum. Untuk dapat digunakan, suatu kelas harus diinstansiasi menjadi objek. Setiap objek yang diinstansiasi dari kelas yang sama akan mempunyai sifat dan tingkah laku yang sama.

Secara umum, ada dua bagian utama pada sebuah class C++, yaitu class declaration dan class body. Deklarasi kelas mendefinisikan nama class dan attributnya, sedangkan class body mendeklarasikan variabel dan method.

Object

Objek adalah suatu pengenal (identifier) yang menyatukan atribut (sering juga disebut property atau state) dengan tingkah laku (yang disebut behaviour atau

method). Penyatuan State dan Behaviour ini pada konsep Pemrograman Berorientasi Objek disebut dengan istilah enkapsulasi (encapsulation). Object mempunyai dua karakteristik, yaitu :

- Memiliki attribut, sebagai status (keadaan), yang kemudain disebut state.
- Memiliki tingkah laku, yang disebut behaviour.

Contoh:
Object Sepeda

- Object Sepeda memiliki attribut (state) : pedal, roda, jeruji, warna, stang, jumlah roda.
- Object Sepeda memiliki tingkah laku (behaviour): kecepatan menaik, kecepatan menurun, memberhentikan, menjalankan, mengganti gigi.

Class dan Object

Sering ada pertanyaan, class dan obyek duluan yang mana? Padahal kenyataannya: class adalah blueprint/prototype saja. Class merupakan definisi tentang state dan behaviour suatu objek. Bisa juga disebut class adalah kumpulan object yang memiliki atribut dan service yang sama. Sedangkan object adalah "barang nyata" dari sebuah class.

Contoh class: manusia sedangkan object-nya adalah kita, misalnya: Anton, Rudi, dan Amir.

Class memiliki sifat pewarisan, yang berarti sifat dari satu class dapat diturunkan ke class lain. Contoh pewarisan adalah class dosen memiliki semua atribut/state dan services dari class manusia. Dari contoh di atas, dapat dikatakan bahwa class dosen mewarisi semua atribut dan service dari class manusia. Class manusia adalah class induk, class dosen adalah class anak.

Pembuatan Class pada C++

Class pada C++ bisa dianggap sebagai tipe data baru. Selain tipe data yang sudah dibawakan oleh C++, kita juga dapat membuat tipe data baru. Jika pada bahasa C/C++ tipe data baru dibuat menggunakan struct, pada C++ tipe data baru bisa dibuat dengan menggunakan class. Konsep ini merupakan konsep berorientasi obyek.

Struktur sederhana sebuah class pada C++:

```
class <namaclass>{
//bagian member variabel / property class
<tipedat <namavariabel>;
//bagian member function / method
<tipedat <namafunction>(<parameter>){
//isi program dalam fungsi
}
};
```

Contoh class:

```
class Kucing{
//bagian member variabel yang bersifat private
private:
int umur;
int berat;
string jenis_kelamin;
string nama_kucing;
//bagian member function/method yang bersifat public
public:
void Bersuara();
int tampilkanUmur();
}
```

TIPS

- Untuk membuat nama class, biasakanlah menggunakan huruf besar. Contohnya: Kucing, Rumah, Handphone, dan lain-lain.
- Untuk membuat nama variabel biasakanlah menggunakan nama yang mewakili property yang dimiliki dan melekat pada nama kelasnya. Pada contoh diatas, class Kucing memiliki berat, umur, dan nama yang melekat erat padanya. Setiap kucing yang ada didunia ini pada umumnya memiliki berat, umur, dan nama yang berbeda-beda satu sama lain. Untuk menamai member variabel, jika nama variabel hanya terdiri dari satu kata gunakanlah huruf kecil, sedangkan jika terdiri dari lebih dari satu kata, gunakan huruf kecil pada huruf kata pertama, sedangkan untuk kata selanjutnya gunakan huruf besar pada huruf-huruf pertamanya. Contoh: string namaKucing, float ipkMahasiswa, int berat, dan lain-lain.
- Untuk membuat nama member function, gunakanlah cara penulisan yang tepat untuk menggambarkan secara benar setiap nama function yang ada. Biasakanlah memberi nama function sesuai dengan behaviour yang memang dikerjakannya. Contohnya: `void cariData(string judul)`, atau `int ambilNilai()` dan lain-lain.

Mendefinisikan Obyek

Setelah kita selesai membuat class baru, maka kita bisa menggunakan class tersebut adalah dengan menginisialisasinya (dengan membuat sebuah atau beberapa obyek) dari class tersebut. Membuat obyek bisa dianggap seperti membuat variabel yang bertipe class yang kita buat. Contoh:

```
1 Kucing kucingku;
2 Orang anton;
3 Mobil kijang;
```

Class dan obyek adalah berbeda. Class merupakan template dari member variabel dan member function yang dapat dibuat wujud nyatanya dalam sebuah obyek. Pada contoh diatas, Kucing adalah class. Kucing tidak bisa langsung digunakan

dalam program. Untuk bisa menggunakan Kucing, yang harus dilakukan adalah membuat obyeknya, yaitu kucingku. Pada dunia nyata kucingku bisa disebut sesuai dengan nama kucing yang kia pelihara. Jadi contoh diatas mungkin bisa diubah menjadi Kucing katty. Dimana katty adalah obyek dari class Kucing yang dapat digunakan dalam program.

Mengakses Member Variabel

Setelah kita membuat obyek seperti:

```
1 Kucing katty;
```

Cara mengakses member variabel adalah dengan menggunakan tanda titik (.). Contoh jika kita hendak mengisi data berat badan katty dengan nilai 8 kg, maka yang harus dilakukan adalah:

```
1 Katty.berat = 8;
```

Demikian juga dengan nama, umur, dan jenis kelamin.

```
1 Katty.nama = “Katty”;
2 Katty.jenis_kelamin = “jantan”;
3 Katty.umur = 2;
```

Mengakses Member Function/Method

Sedangkan cara untuk mengakses member function dari suatu class adalah dengan juga menggunakan tanda titik pada obyeknya. Contoh:

```
1 katty.Bersuara();
2 katty.tampilkanUmur();
```

Contoh 1. Pembuatan class Sepeda
Buatlah project baru dan tulis kode berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>

using namespace std;

class Sepeda{
        private:
        int kecepatan;
        int gigi;
        string merk;
        public:
        void ubahKecepatan(int kec);
        void ubahGigi(int g);
        void setMerk(string m);
        void tampilSepeda();
};

void Sepeda::ubahKecepatan(int kec){
        this->kecepatan = kec;
}

void Sepeda::ubahGigi(int g){
        this->gigi = g;
}

void Sepeda::setMerk(string m){
        this->merk = m;
}

void Sepeda::tampilSepeda(){
        cout <<"Kecepatan: "<<this->kecepatan<<endl<<
        "Merk: "<<this->merk<<endl<<
        "Gigi: "<<this->gigi;
}

int main(int argc, char *argv[])
```

```
37 {
38         QCoreApplication a(argc, argv);
39         Sepeda objSpd;
40         objSpd.ubahGigi(2);
41         objSpd.ubahKecepatan(30);
42         objSpd.setMerk("Federal");
43         objSpd.tampilSepeda();
44         return a.exec();
45 }
```

Hasil:

```
Kecepatan: 30 Merk: Federal
```

Keterangan:

- Program diatas membuat sebuah class bernama Sepeda. Di dalam class Sepeda, terdapat dua bagian, bagian pertama berisi semua member variabel yang bersifat private, yaitu kecepatan, gigi, dan merk. Pada bagian kedua terdapat member function yang hanya beisi judul method saja sedangkan implementasinya diletakkan diluar class Sepeda.
- Diluar kelas Sepeda, kita mendefinisikan semua implementasi method dari semua member function yang sudah kita definisikan diatas. Untuk mengakses member function dari luar kelasnya, digunakan tanda `::` setelah nama class. Implementasi method bisa menggunakan cara lain yang akan dijelaskan dibagian-bagian berikutnya.
- Pada function main, kita membuat obyek dari class Sepeda yang bernama objSpd dan kemudian kita akses semua member functionnya.
- Sebelum menampilkan hasil kita isi terlebih dahulu kecepatan, gigi, dan merk dari Sepeda yang kita buat.
- Keyword this mengacu pada class itu sendiri (class Sepeda) dan merupakan variabel pointer. Keyword tersebut digunakan untuk mengakses semua member variabel dan member method class Sepeda.

TIPS

Keyword `this` pada class Sepeda merupakan kata kunci untuk mengakses class yang didefinisikan (kelas dirinya sendiri). Tanda `->` merupakan tanda bahwa `this` merupakan obyek pointer. Cara lain untuk mengakses kelas itu sendiri adalah dengan menggunakan `<namakelas>::` diikuti nama method / variabel member .

Contoh :

```
1  void Sepeda::ubahKecepatan(int kec){
2  Sepeda::kecepatan = kec;
3  }
```

Contoh 2. Pembuatan obyek Sepeda.
Buatlah project baru dan tulis kode berikut:

```
1  #include <QtCore/QCoreApplication>
2  #include <iostream>
3  using namespace std;
4  class Sepeda{
5          private:
6          int kecepatan;
7          int gigi;
8          string merk;
9          public:
10         void ubahKecepatan(int kec);
11         void ubahGigi(int g);
12         void setMerk(string m);
13         void tampilSepeda();
14 };
15
16 void Sepeda::ubahKecepatan(int kec){
17         this->kecepatan = kec;
18 }
19
20 void Sepeda::ubahGigi(int g){
```

```
21          this->gigi = g;
22  }
23
24  void Sepeda::setMerk(string m){
25          this->merk = m;
26  }
27
28  void Sepeda::tampilSepeda(){
29          cout <<"Kecepatan: "<<this->kecepatan<<endl<<
30          "Merk: "<<this->merk<<endl<<
31          "Gigi: "<<this->gigi;
32  }
33  int main(int argc, char *argv[])
34  {
35          QCoreApplication a(argc, argv);
36          cout<<"Sepeda pertama:\n";
37          Sepeda objSpd;
38          objSpd.ubahGigi(2);
39          objSpd.ubahKecepatan(30);
40          objSpd.setMerk("Federal");
41          objSpd.tampilSepeda();
42          cout<<"\nSepeda kedua:\n";
43          Sepeda objSpd2;
44          objSpd2.ubahGigi(1);
45          objSpd2.ubahKecepatan(45);
46          objSpd2.setMerk("Polygon");
47          objSpd2.tampilSepeda();
48          return a.exec();
49  }
```

Hasil:

```
Sepeda pertama:
Kecepatan: 30
Merk: Federal
```

```
Gigi: 2
Sepeda kedua:
Kecepatan: 45
Merk: Polygon
```

Keterangan:

- Program diatas merupakan pengembangan dari program sebelumnya dimana kita membuat satu lagi variabel `objSpd2`.
- Terlihat bahwa masing-masing obyek sepeda yang terbuat memiliki data yang berbeda-beda satu sama lain.
- Artinya class hanyalah merupakan template / blueprint saja, dimana data-data dan tingkah laku dari kelas haruslah dilakukan oleh obyeknya. Jadi obyek adalah bentuk nyata dari sebuah kelas yang memiliki data dan method yang berbeda-beda satu sama lain.

Contoh 3. Pembuatan Obyek Array Sepeda
Buatlah program beikut ini:

```
1  #include <QtCore/QCoreApplication>
2
3  #include <iostream>
4
5  using namespace std;
6
7  class Sepeda{
8          private:
9          int kecepatan;
10         int gigi;
11         string merk;
12         public:
13         void ubahKecepatan(int kec);
14         void ubahGigi(int g);
15         void setMerk(string m);
16         void tampilSepeda();
```

```
17  };
18
19  void Sepeda::ubahKecepatan(int kec){
20          this->kecepatan = kec;
21  }
22
23  void Sepeda::ubahGigi(int g){
24          this->gigi = g;
25  }
26
27  void Sepeda::setMerk(string m){
28          this->merk = m;
29
30  }
31  void Sepeda::tampilSepeda(){
32          cout <<"Kecepatan: "<<this->kecepatan<<endl<<
33          "Merk: "<<this->merk<<endl<<
34          "Gigi: "<<this->gigi;
35  }
36
37  int main(int argc, char *argv[])
38  {
39          QCoreApplication a(argc, argv);
40          cout<<"Sepeda pertama:\n";
41          Sepeda objSpdArray[5];
42
43          for(int i=0;i<5;i++){
44                  objSpdArray[i].setMerk("Merk-"+i);
45                  objSpdArray[i].ubahGigi(i+10);
46                  objSpdArray[i].ubahKecepatan(i+30);
47          }
48
49          for(int i=0;i<5;i++){
50                  cout<<"Tampilan Sepeda ke-"<<(i+1)<<endl;
51                  objSpdArray[i].tampilSepeda();
52                  cout<<endl;
```

```
53  }
54
55  return a.exec();
56
57  }
```

Hasil:

```
Sepeda pertama:
Tampilan Sepeda ke-1
Kecepatan: 30
Merk: Merk-
Gigi: 10
Tampilan Sepeda ke-2
Kecepatan: 31
Merk: erk-
Gigi: 11
Tampilan Sepeda ke-3
Kecepatan: 32
Merk: rk-
Gigi: 12
Tampilan Sepeda ke-4
Kecepatan: 33
Merk: k-
Gigi: 13
Tampilan Sepeda ke-5
Kecepatan: 34
Merk: -
Gigi: 14
```

Keterangan:

- Program diatas merupakan pengembangan lagi dari Contoh 1.
- Program diatas membuat obyek dari class Sepeda dalam bentuk Array 1 dimensi yang bertipe Sepeda.

- Array yang bertipe class Sepeda tersebut tetap memiliki indeks dari 0 sampai dengan n-1
- Masing-masing obyek elemen array `objSpdArray` berisi data-data yang berbeda-beda satu sama lainnya.

TIPS

Kita juga dapat melakukan assigment / penugasan terhadap obyek ke obyek lain. Contoh kita memiliki class Sepeda dan kita membuat obyek spd1 dan spd2.

```
1  Sepeda spd1,spd2;
2  spd1.setMerk("X");
3  spd1.ubahKecepatan(50);
4  spd1.ubahGigi(4);
```

maka bisa dilakukan:

```
1  spd2 = spd1;
```

Jika `spd2` ditampilkan, dengan `sp2.tampilSepeda()`, maka nilai yang ditampilkan akan sama persis dengan nilai `spd1`.

Class yang kita definisikan memiliki member method. Semua member method tersebut dapat kita gunakan. Apa yang kita buat dalam member method akan membuat kompiler mendaftarkan semua method yang kita buat kedalam memory sehingga hanya method yang kita daftarkan saja yang bisa kita akses dari class kita.

Hak Akses Member Variabel dan Method Variabel

Pada pemrograman berorientasi obyek, terdapat konsep penting yang bernama *enkapsulasi*. Konsep tersebut berarti kita "membungkus" semua member variabel dan member method kedalam suatu class termasuk hak akses terhadap mereka. Apa arti hak akses? Hak akses adalah bagaimana class yang terenkapsulasi tersebut

"menyembunyikan" hal-hal yang tidak perlu / tidak boleh dilihat dari luar class. Dengan adanya hak akses tersebut semua data dan method akan terlindungi dan tidak termodifikasi. Kita dapat analogikan dengan kasus nyata sebuah benda, misalnya AC. AC merupakan alat elektronik yang rumit dan mampu mendinginkan ruangan. Jika seseorang memasang AC maka AC akan melindungi dirinya dengan hak akses. Kita sebagai orang awam tentang AC hanya diperbolehkan mengakses yang diperbolehkan saja untuk mengantisipasi hal-hal yang tidak diinginkan seperti misalnya AC akan rusak. Kita hanya diberikan tombol-tombol sederhana dan mungkin remote untuk mengatur semua tentang AC, kita tidak bisa mengakses hardwarenya, kabel didalamnya, PCB nya, kondensatornya dan lain-lain. Yang bisa melakukan itu adalah para ahli AC. Dengan demikian AC sudah berusaha melindungi dirinya dari tangan-tangan orang awam yang memang tidak berhak.

Demikian pula pada class, class juga memiliki dua bagian: member variabel dan member method. Kedua bagian ini berbeda fungsinya. Member variabel digunakan untuk menyimpan sifat-sifat yang dimiliki dan melekat pada class sedangkan method digunakan untuk melakukan operasi/kegiatan terhadap class tersebut. Jika kita bandingan dengan AC, maka method bisa dibilang sebagai remote/tombol. Sehingga untuk mengakses data-data yang ada pada class kita sangat direkomendasikan untuk menggunakan method bukan mengaksesnya secara langsung.

Class pada C++ memiliki cara melindungi dirinya yaitu dengan menggunakan keyword private, protected dan public. Keyword private atau protected biasanya digunakan pada semua variabel member sedangkan keyword protected atau public digunakan pada semua variabel method. Dengan menggunakan keyword private, maka bagian private tersebut tidak akan bisa diakses dari luar class, harus dari dalam class tersebut atau berada dalam method class tersebut, sedangkan jika public maka bisa akses dari luar class.

Contoh 4. Perbedaan private dan public pada member variabel

Buatlah program berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>

using namespace std;

class Sepeda{
        private:
        int kecepatan;
        int gigi;
        string merk;
        public:
        int pkecepatan;
        int pgigi;
        string pmerk;
};

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        cout<<"Pengaksesan public:\n";
        Sepeda s;
        s.pgigi = 3;
        s.pkecepatan = 30;
        s.pmerk = "Polygon";
        cout<<"Gigi: "<<s.pgigi<<endl;
        cout<<"Kecepatan: "<<s.pkecepatan<<endl;
        cout<<"Merk: "<<s.pmerk<<endl;
        return a.exec();
}
```

Hasil:

```
Pengaksesan public:
Gigi: 3
Kecepatan: 30
Merk: Polygon
```

Keterangan:

Semua variabel member yang bersifat public dapat diakses dan diisi dengan baik dari luar class, dalam hal ini adalah function `int main()`. Function `int main()` berada diluar class Sepeda Akan terjadi hal yang berbeda jika kita mengakses semua variabel member yang bersifat private.

Ubahlah program pada Contoh 5 diatas menjadi seperti berikut ini:

```
#include <QtCore/QCoreApplication>

#include <iostream>

using namespace std;

class Sepeda{
        private:
        int kecepatan;
        int gigi;
        string merk;
        public:
        int pkecepatan;
        int pgigi;
        string pmerk;
};

int main(int argc, char *argv[])
{
        QCoreApplication a(argc, argv);
        cout<<"Pengaksesan public:\n";
        Sepeda s;
        s.gigi = 3;
        s.kecepatan = 30;
        s.merk = "Federal";
```

```
26          cout<<"Gigi: "<<s.gigi<<endl;
27          cout<<"Kecepatan: "<<s.kecepatan<<endl;
28          cout<<"Merk: "<<s.merk<<endl;
29          return a.exec();
30  }
```

Hasil:

Keterangan::

Akan terjadi compile time error, karena kita mengakses variabel member yang bersifat *private*. Berarti class Sepeda sudah bisa menerapkan fungsi enkapsulasi dan melindungi data-datanya dari pengaksesan langsung.

Member Function / Member Method

Seperti yang sudah dijelaskan, member method merupakan bagian yang harus dideklarasikan sebagai bagian public. Salah satu kegunaan member function adalah mengakses semua member variabel dan tetap mendukung enkapsulasi. Cara untuk membuat member method adalah dengan mendeklarasikannya pada bagian public, sedangkan implementasi kodingnya berada diluar kelas. Berikut adalah contohya.

```
class Sepeda{
        private:
        int kecepatan;
        int gigi;
        string merk;
        public:
        void setKecepatan(int k);
        void setGigi(int g);
        void setMerk(string m);
};
```

Di dalam pemrograman beriorientasi obyek pada umumnya member function m\
inimal selalu mewakili semua member variabelnya. Misal kita memiliki 1 \
buah member variabel bernama umur, maka minimal kita akan memiliki satu\
 buah member function, misalnya bernama ubahUmur(int u).

Contoh 5. Member function dan implementasinya.

Buatlah program berikut ini:

```cpp
#include <QtCore/QCoreApplication>
#include <iostream>

using namespace std;

//pembuatan class Sepeda
class Sepeda{
        private:

        //daftar member variabel
        int kecepatan;
        int gigi;
        string merk;
        public:
```

```
37
38          //daftar member function
39          void ubahKecepatan(int kec);
40          void ubahGigi(int g);
41          void setMerk(string m);
42          void tampilSepeda();
43  };
44
45  //implementasi member function berada diluar class Sepeda
46  //function ubahKecepatan menerima input jumlah kecepatan
47  //mengubah kecepatan Sepeda
48
49  void Sepeda::ubahKecepatan(int kec){
50          this->kecepatan = kec;
51  }
52
53  //function ubahGigi menerima input jumlah gigi
54  //mengubah gigi Sepeda
55  void Sepeda::ubahGigi(int g){
56          this->gigi = g;
57  }
58  //function setMerk menerima input string merk
59  //mengisi merk Sepeda
60  void Sepeda::setMerk(string m){
61          this->merk = m;
62  }
63  //function tampilSepeda tidak menerima input
64  //fungsinya hanya untuk menampilkan informasi obyek Sepeda
65  void Sepeda::tampilSepeda(){
66          cout <<"Kecepatan: "<<this->kecepatan<<endl<<
67          "Merk: "<<this->merk<<endl<<
68          "Gigi: "<<this->gigi;
69  }
70
71  //function main
72  int main(int argc, char *argv[])
```

```
73 {
74         QCoreApplication a(argc, argv);
75         Sepeda objSpd;
76         objSpd.ubahGigi(2);
77         objSpd.ubahKecepatan(30);
78         objSpd.setMerk("Federal");
79         objSpd.tampilSepeda();
80         return a.exec();
81
82 }
```

Hasil:

```
Kecepatan: 30 Merk: Federal
```

Keterangan:

- Semua member variabel yang dimiliki tidak diakses secara langsung dari function main, tapi melalui method-methodnya.
- Pada program diatas setiap member variabel memiliki minimal satu buah method member
- Terdapat satu buah method tambahan yang berfungsi untuk menampilkan semua informasi mengenai sepeda
- Setiap method member dapat menerima input dan mengeluarkan output.
- Kata kunci `this->` pada method member berfungsi untuk mengakses semua member variabel yang terdapat pada class Sepeda yang biasanya bersifat private.
- Implementasi method member berada diluar class Sepeda dan dimulai dengan nama classnya kemudian diikuti tanda :: yang artinya mengakses member method.

Accessor dan Mutator Method

Pada pemrograman berorientasi obyek dengan C++, kita memiliki method member. Tujuan dari method member selain memberi tingkah laku dari class tersebut

adalah melakukan akses terhadap semua member variabel yang bersifat private agar tetap bisa diakses dari luar class.

Member method yang berkaitan dengan member variabel ada 2 jenis, yaitu member method yang berfungsi untuk mengeset / mengisi nilai member variabel dan member method yang berfungsi untuk mengambil nilai member variabel.

Accessor method

method ini berfungsi untuk mengambil nilai dari sebuah member variabel. Asesor method biasanya dinamai :

```
<tipedataMemberVariabel> get<NamaMemberVariabel>();
```

Contoh:

```
int getUmur();
```

Mutator method

method ini berfungsi untuk mengisi / mengeset nilai kepada sebuah member variabel

Mutator method biasanya dinamai :

```
void set<NamaMemberVariabel>(<tipedataMemberVariabel> <namavariabel>);
```

Contoh:

```
void setUmur(int u);
```

Contoh 6. Penggunaan accessor dan mutator method

Tulislah program berikut ini:

```
#include <QtCore/QCoreApplication>
#include <iostream>

using namespace std;

class Kucing{
        private:
        int umur;
        float berat;
        string nama;
        public:

        //asesor method
        int getUmur();
        float getBerat();
        string getNama();

        //mutator method
        void setUmur(int u);
        void setBerat(float b);
        void setNama(string s);

        //method tambahan
        void berlari();
};
//implementasi
int Kucing::getUmur(){
        return this->umur;
}

float Kucing::getBerat(){
        return this->berat;
}

string Kucing::getNama(){
        return this->nama;
```

```
37  }
38
39  void Kucing::setUmur(int u){
40          this->umur = u;
41  }
42
43  void Kucing::setBerat(float b){
44          this->berat = b;
45  }
46
47  void Kucing::setNama(string s){
48          this->nama = s;
49  }
50
51  void Kucing::berlari(){
52          cout<<"Kucing "<<this->getNama()<<" sedang berlari!";
53  }
54
55  int main(int argc, char *argv[])
56  {
57          QCoreApplication a(argc, argv);
58          Kucing mycat;
59          mycat.setNama("Katty");;
60          mycat.setBerat(4);
61          mycat.setUmur(2);
62          cout<<"Kucingku bernama "<<mycat.getNama()<<", dia berbobot "<<
63          mycat.getBerat()<<" kg dan sudah berumur "<<mycat.getUmur()
64                  <<" tahun sekarang."<<endl;
65          mycat.berlari();
66          return a.exec();
67  }
```

Hasil:

Kucingku bernama Katty, dia berbobot 4 kg dan sudah berumur 2 tahun sekarang.

Keterangan:

- Program diatas memperlihatkan bagaimana setiap member variabel memiliki tepat dua buah method member, dimana setiap method member yang satu berfungsi sebagai asesor method dan yang lain berfungsi sebagai mutator method.
- Terdapat sebuah method tambahan yaitu berlari yang hendak menggambarkan bahwa selain asesor dan mutator kita masih diperbolehkan membuat method lainnya.
- Asesor method mengambil data member variabel sehingga dibuat fungsi berupa function non void, sedangkan mutator method mengeset data member variabel sehingga dibuat fungsi berupa function void yang menerima parameter yang sesuai dengan tipe data member variabelnya.

Constructor dan Destructor

Kita dapat mendeklarasikan variabel biasa dan kemudian melakukan inisialisasi terhadap variabel tersebut dengan mudah. Contoh:

```
1  int umur = 5;
```

Inisialisasi variabel berfungsi untuk mengisi suatu nilai awal terhadap suatu variabel yang kita deklarasikan. Variabel tersebut masih bisa kita ubah-ubah lagi nilainya dikemudian waktu. Nah bagaimana untuk menginisialisasi variabel member pada suatu class? Caranya dengan membuat method yang berjenis constructor method. Sedangkan untuk mendealokasi dan melakukan finalisasi sebuah class kita gunakan destructor method. Constructor berfungsi untuk menginisialisasi obyek dari class dan mempersiapkan ruang memory, sedangkan destructor menghapus dan membersihkan obyek ketika sudah tidak terpakai dan membebaskan memory yang tadinya terpakai.

Constructor method merupakan method yang namanya sama dengan nama classnya dan bersifat public tapi tidak berjenis void ataupun non void. Constructor dapat menerima parameter namun tidak bisa mengembalikan nilai apapun.

Desktruktor method merupakan method kebalikan dari constructor yang juga bernama sama dengan nama classnya namun diawali dengan tanda ~. Destructor tidak boleh memiliki parameter apapun.

Contoh jika kita memiliki class bernama Sepeda, maka kita dapat membuat constructor dengan nama Sepeda() juga. Sedangkan destructor method sama dengan constructor namun diawali dengan tanda ~ didepannya. Contoh:

```
class Sepeda{
        private:
        //member variabel

        public:
        //konstruktor
        Sepeda();

        //destruktor
        ~Sepeda();
};
```

Default Constructor

Pada bahasa C++ semua class yang telah dibuat PASTI memiliki constructor walaupun tidak kita buat. Compiler bahasa C++ pasti membuatnya walau secara implisit. Constructor yang bernama sama dengan nama classnya dan tidak berparameter disebut default constructor. Secara default pasti semua class ada default constructornya. Kapan kita menggunakan constructor? Setiap kali kita membuat obyek baru (melakukan instansiasi obyek), maka kita memanggil constructor default.

Contoh:

```
Sepeda sepedaku;
```

Berarti kita memanggil default konstruktor bernama Sepeda() tanpa parameter apapun. Jika kita membuat konsruktor dengan menggunakan parameter seperti misalnya:

```
Sepeda(string merk, int berat);
```

Maka pada saat instansiasi kita menggunakan cara sebagai berikut:

```
Sepeda sepedaku(“Federal”,2);
```

Arti instasiasi diatas adalah kita memanggil konstruktor yang berparameter dua buah, string dan integer.

Contoh 7. Menggunakan Constructor dan Destructor.

Buatlah program berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>

using namespace std;

class Kucing{
        private:
        int umur;
        float berat;
        string nama;
        public:

        //konstruktor
        Kucing(int umur);

        //desktruktor
        ~Kucing();

        //asesor method
        int getUmur();
        float getBerat();
        string getNama();

        //mutator method
        void setUmur(int u);
        void setBerat(float b);
```

```
27          void setNama(string s);
28  };
29
30  //implementasi konstruktor dan desktruktor
31  Kucing::Kucing(int u){
32          this->umur = u;
33  }
34  Kucing::~Kucing(){
35  }
36
37  //implementasi function
38  int Kucing::getUmur(){
39          return this->umur;
40  }
41
42  float Kucing::getBerat(){
43          return this->berat;
44  }
45
46  string Kucing::getNama(){
47          return this->nama;
48  }
49
50  void Kucing::setUmur(int u){
51          this->umur = u;
52  }
53
54  void Kucing::setBerat(float b){
55          this->berat = b;
56  }
57
58  void Kucing::setNama(string s){
59          this->nama = s;
60  }
61
62  int main(int argc, char *argv[])
```

```
63 {
64         QCoreApplication a(argc, argv);
65         Kucing mycat(2);
66         mycat.setNama("Katty");;
67         mycat.setBerat(4);
68         cout<<"Kucingku bernama "<<mycat.getNama()<<", dia berbobot "<<
69         mycat.getBerat()<<" kg dan sudah berumur "<<mycat.getUmur()
70         <<" tahun sekarang."<<endl;
71         mycat.setUmur(7);
72         cout<<"Lima tahun telah berlalu, sekarang kucingku sudah berumur:
73         "<<mycat.getUmur()<<" tahun";
74         return a.exec();
75 }
```

Hasil:

Kucingku bernama Katty, dia berbobot 4 kg dan sudah berumur 2 tahun sekarang.

Keterangan:

- Program diatas menunjukkan pemakaian konstruktor dan desktruktor. Constructor digunakan untuk menginisialisasi umur kucing pada saat awal pertama obyek dibuat, kemudian pada akhirnya kita juga tetap dapat mengubah umur kucing dibagian akhir program.
- Destruktor yang kita buat merupakan default desktruktor dimana desktruktor tidak boleh memiliki parameter apapun.

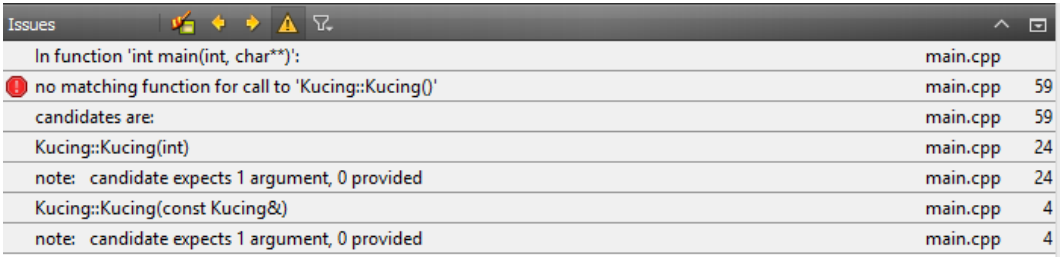
TIPS

Jika kita sudah membuat konstruktor yang memiliki parameter pada class kita, maka secara otomatis default constructor yang dibuat oleh compiler tidak ada lagi, sehingga ketika kita melakukan instansiasi pada class Kucing diatas tanpa parameter pasti akan error.

Contoh, tambahkan satu baris berikut ini pada bagian akhir kode pada Contoh 7 sebelum `return a.exec()`.

```
Kucing kucingku2;
```

Ketika dilakukan kompilasi akan menghasilkan error sebagai berikut:



Issues		
In function 'int main(int, char**)':	main.cpp	
no matching function for call to 'Kucing::Kucing()'	main.cpp	59
candidates are:	main.cpp	59
Kucing::Kucing(int)	main.cpp	24
note: candidate expects 1 argument, 0 provided	main.cpp	24
Kucing::Kucing(const Kucing&)	main.cpp	4
note: candidate expects 1 argument, 0 provided	main.cpp	4

Error diatas mengatakan bahwa class Kucing tidak memiliki function yang bernama `Kucing::Kucing()`, yang artinya method constructor defaultnya sudah hilang. Agar kita dapat menggunakan baris `Kucing kucingku2;` maka kita harus menambah method constructor lagi yang tidak berparameter.

Contoh 8. Percobaan Menambah Constructor Method.

Buatlah program berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Kucing{
        private:
        int umur;
        float berat;
        string nama;
        public:

        //konstruktor
        Kucing(int umur);
        Kucing();

        //desktruktor
        ~Kucing();

        //asesor method
```

```
19          int getUmur();
20
21          float getBerat();
22          string getNama();
23
24          //mutator method
25          void setUmur(int u);
26          void setBerat(float b);
27          void setNama(string s);
28  };
29
30  //implementasi konstruktor dan desktruktor
31  Kucing::Kucing(int u){
32          this->umur = u;
33  }
34  Kucing::Kucing(){
35  }
36
37  Kucing::~Kucing(){
38          cout<<"Obyek sudah dihancurkan!";
39  }
40  //implementasi function
41  int Kucing::getUmur(){
42          return this->umur;
43  }
44
45  float Kucing::getBerat(){
46          return this->berat;
47  }
48
49  string Kucing::getNama(){
50          return this->nama;
51  }
52
53  void Kucing::setUmur(int u){
54          this->umur = u;
```

```
55  }
56
57  void Kucing::setBerat(float b){
58          this->berat = b;
59  }
60
61  void Kucing::setNama(string s){
62          this->nama = s;
63  }
64
65  int main(int argc, char *argv[])
66  {
67          QCoreApplication a(argc, argv);
68          Kucing mycat(2);
69          mycat.setNama("Katty");;
70          mycat.setBerat(4);
71          cout<<"Kucingku bernama "<<mycat.getNama()<<", dia berbobot "<<
72          mycat.getBerat()<<" kg dan sudah berumur "<<mycat.getUmur()
73          <<" tahun sekarang."<<endl;
74          mycat.setUmur(7);
75          cout<<"Lima tahun telah berlalu, sekarang kucingku sudah berumur:
76          "<<mycat.getUmur()<<" tahun"<<endl;
77          Kucing kucingku2;
78          kucingku2.setNama("Frizky");
79          cout<<"Nama kucing keduaku: "<<kucingku2.getNama();
80          return a.exec();
81  }
```

Hasil:

```
1  Kucingku bernama Katty, dia berbobot 4 kg dan sudah berumur 2 tahun sek\
2  arang.
3  Lima tahun telah berlalu, sekarang kucingku sudah berumur:7 tahun
```

Keterangan:

- Pada program C++, kita dapat membuat konstruktor method lebih dari satu,

asal tidak sama. Konsep diatas dinamakan dengan polymorfisme (OVERLOADING) yang akan dibahas lebih lanjut dibab-bab berikutnya.

- Dengan mendefinisikan konstruktor tanpa parameter maka kita dapat menginstansiasi obyek dengan cara biasa, seperti pada contoh Kucing kucingku2;

Constructor Dengan nilai Default

Constructor dapat memiliki nilai default sehingga jika konstruktor yang dipanggil tidak diisi nilai, maka nilai-nilai lainnya akan tetap diinisialisasi dengan nilai defaultnya. Hal ini diperlukan untuk mempermudah menginisialisasi data variabel member. Penggunaan nilai default ini juga memungkinkan kita untuk tidak memasukkan semua parameter pada pemanggilan konstruktor.

Contoh 9. Penggunaan Constructor dengan Nilai Default

Buatlah program berikut:

```
1  #include <QtCore/QCoreApplication>
2  #include <iostream>
3
4  using namespace std;
5
6  class Buku{
7          private:
8          int jmlhal;
9          string pengarang;
10         string judul;
11         public:
12         Buku(string pengarang="unknown", string judul="unknown",int jmlhal=1){
13                 Buku::jmlhal = jmlhal;
14                 Buku::pengarang = pengarang;
15                 Buku::judul = judul;
16         }
17
18         void tampilInfo(){
19                 cout<<"Judul: "<<Buku::judul<<endl;
20                 cout<<"Pengarang: "<<Buku::pengarang<<endl;
```

```
21                  cout<<"Jumlah halaman: "<<Buku::jmlhal<<endl;
22          }
23  };
24
25  int main(int argc, char *argv[])
26  {
27          QCoreApplication a(argc, argv);
28          Buku b1;
29          Buku b2("Antonius");
30          Buku b3("Robert","Membuat aplikasi C++");
31          Buku b4("Walter","Pemrograman C",100);
32                  b1.tampilInfo();
33          b2.tampilInfo();
34          b3.tampilInfo();
35          b4.tampilInfo();
36          return a.exec();
37  }
```

Hasil:

```
Judul: unknown
Pengarang: unknown
Jumlah halaman: 1
Judul: unknown
Pengarang: Antonius
Jumlah halaman: 1
Judul: Membuat aplikasi C++
Pengarang: Robert
Jumlah halaman: 1
Judul: Pemrograman C
Pengarang: Walter
Jumlah halaman: 100
```

Keterangan:

- Program diatas menunukkan bahwa kita dapat membuat konstruktor dengan

nilai default, yaitu dengan menggunakan parameter dan langsung diinisialisasi dengan menggunakan tanda sama dengan (=).

- Pada pemanggilan konstruktor, terlihat bahwa jika konstruktor tidak diisi parameter apapun maka ketika data ditampilkan semua isi member variabel sesuai dengan nilai defaultnya.
- Pada pemanggilan konstruktor kedua, yaitu dengan satu parameter string, maka string tersebut mengacu pada parameter pertama, yaitu Pengarang, sehingga judul dan jumlah halaman berisi nilai default.
- Pada pemanggilan konstruktor ketiga, yaitu dengan dua parameter string, maka kedua parameter itu mengisi pengarang dan judulnya (hal ini sesuai dengan urutan penempatan pada pendefinisian method konstruktor pada program), sedangkan variabel member lain berisi default
- Pada pemanggilan ketiga, ketiga parameter diisi sehingga semua nilai default berubah.

TIPS

Kita juga dapat memberi nilai default dengan cara lain, perhatikan contoh berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Contoh{
        private:
        int x;
        int y;
        int z;
        public:
        Contoh():x(0),y(2),z(4){
        }
                void tampilInfo(){
        cout<<"x="<<x<<" y="<<y<<" z="<<z;
        }
};
```

```
16
17 int main(int argc, char *argv[])
18 {
19         QCoreApplication a(argc, argv);
20         Contoh aa;
21         aa.tampilInfo();
22         return a.exec();
23 }
```

Hasil:

```
x=0 y=2 z=4
```

Keterangan:
Terlihat bahwa kita bisa menginisialisasi isi dari variabel member yang kita miliki dengan cara menuliskannya pada bagian header method member seperti pada contoh diatas. Dan ketika class diinstasiasi maka otomatis konstruktor dipanggil dan semua nilai variabel member telah diinisialisasi seperti yang sudah dituliskan.

const member method

Kita menggunakan kata kunci const untuk membuat suatu identifier konstanta. Konstanta berarti suatu variabel yang tidak bisa diganti / diubah nilainya pada saat program berjalan (runtime). Konstanta juga dapat digunakan pada method member Dengan memberikan kata kunci const setelah nama method, maka method tersebut juga tidak akan bisa diubah nilainya pada saat class dijalankan. Kegunaan method const adalah pada asesor method. Mengapa? Karena pada asesor method kita menggunakan method tersebut untuk mengambil nilai dari member variabel, bukan untuk mengubah nilainya. Sedangkan pada mutator method, method tersebut tidak boleh dibuat const method karena method tersebut digunakan khusus untuk mengubah nilai dari member function. Sehingga cara yang tepat untuk mendeklarasikan asesor method adalah dengan cara memberi kata kunci const pada akhir nama method tersebut. Contoh:

```
//mutator
void setUmur(int u);
//asesor
int getUmur() const;
```

Contoh 10. Penggunaan const method.
Buatlah program berikut ini:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Kucing{
        private:
        int umur;
        float berat;
        string nama;
        public:
        //konstruktor
        Kucing(int umur);
        Kucing();
        //desktruktor
        ~Kucing();
        //asesor method
        int getUmur() const;
        float getBerat() const;
        string getNama() const;
        //mutator method
        void setUmur(int u);
        void setBerat(float b);
        void setNama(string s);
};

//implementasi konstruktor dan desktruktor
Kucing::Kucing(int u){
        this->umur = u;
}

```

```
30 Kucing::Kucing(){
31 }
32
33 Kucing::~Kucing(){
34         cout<<"Obyek sudah dihancurkan!";
35 }
36
37 //implementasi function
38 int Kucing::getUmur() const{
39         return this->umur;
40 }
41
42 float Kucing::getBerat() const{
43         return this->berat;
44 }
45
46 string Kucing::getNama() const{
47         return this->nama;
48 }
49
50 void Kucing::setUmur(int u){
51         this->umur = u;
52 }
53
54 void Kucing::setBerat(float b){
55         this->berat = b;
56 }
57
58 void Kucing::setNama(string s){
59         this->nama = s;
60 }
61
62 int main(int argc, char *argv[])
63 {
64         QCoreApplication a(argc, argv);
65         Kucing mycat(2);
```

```
66          mycat.setNama("Katty");;
67          mycat.setBerat(4);
68          cout<<"Kucingku bernama "<<mycat.getNama()<<", dia berbobot "<<
69          mycat.getBerat()<<" kg dan sudah berumur "<<mycat.getUmur()
70          <<" tahun sekarang."<<endl;
71          mycat.setUmur(7);
72          cout<<"Lima tahun telah berlalu, sekarang kucingku sudah berumur:
73          "<<mycat.getUmur()<<" tahun"<<endl;
74          Kucing kucingku2;
75          kucingku2.setNama("Frizky");
76          cout<<"Nama kucing keduaku: "<<kucingku2.getNama();
77          return a.exec();
78  }
```

Hasil:

> Kucingku bernama Katty, dia berbobot 4 kg dan sudah berumur 2 tahun sekarang.
>
> Lima tahun telah berlalu, sekarang kucingku sudah berumur: 7 tahun

Keterangan:

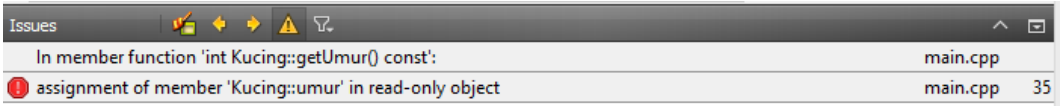
- Program diatas hasilnya sama dengan program sebelumnya karena kita hanya mengubah bagian asesor method dengan cara menambah kata const dibelakangnya. Bagian implementasi method tersebut juga harus disesuaikan.
- Dengan cara ini method asesor tersebut sudah bersifat read-only. Ubahlah bagian method `void getUmur() const`; Kita coba tambahkan baris program berikut sebelum return:

  this->umur = 5.

Kode lengkapnya adalah:

```
int Kucing::getUmur() const{
this->umur = 5;
return this->umur;
}
```

Jika kita kompilasi program diatas, maka akan terjadi error sebagai berikut:



Mengapa hal ini terjadi? Karena method `getUmur` sudah dibuat menjadi *konstan*, yang artinya *readonly*. Di dalam *method read-only* kita tidak diperbolehkan melakukan operasi *assigment* atau *pemberian nilai*. Namun jika kita buang kata kunci `const`, maka method `getUmur` ini tetap dapat diubah nilainya. Dengan demikian kata kunci const benar-benar mampu mengamankan method dari hal yang tidak diinginkan, karena pada dasarnya method *asesor* memang tidak boleh mengubah nilai, hanya boleh membaca/mengambil nilai saja.

Mendefinisikan Method Member

Selama ini kita mendefinisikan method member pada luar class. Selain cara diatas, kita juga bisa mendefinsikan method di dalam class itu sendiri secara langsung. Hal tersebut dinamakan inline implementation. Contoh inline implementation adalah:

```
class Manusia{
        private:
        string nama;
        public:
void setNama(string n){
        this->nama = n;
        }
        string getNama() const{
        return this->nama;
        }
}
```

Pada contoh diatas terlihat bahwa pada class Manusia implementasi kode method setNama dan getNama langsung dituliskan didalam program tersebut. Hal itu disebut inline implementation.

Contoh 11. Inline Implementation.

Buatlah program berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>

using namespace std;

class Manusia{
        private:
        string nama;
        char jenis_kelamin;
        public:
        //konstruktor

        Manusia(){
        }
        Manusia(string nama){
        this->nama = nama;
        }

        //desktruktor
        ~Manusia(){
        }

        //accessor method
        string getNama() const{
        return this->nama;
        }

        char getJenis_Kelamin() const{
        return this->jenis_kelamin;
        }
```

```
31
32          //mutator method
33          void setNama(string n){
34          this->nama = n;
35          }
36
37          void setJenis_Kelamin(char jk){
38          this->jenis_kelamin = jk;
39          }
40
41          //method lain
42          void tampilSemua(){
43          cout<<"Nama: "<<this->getNama()<<", "<<
44          "Jenis Kelamin: "<<this->getJenis_Kelamin()<<endl;
45          }
46  };
47
48  int main(int argc, char *argv[])
49  {
50          QCoreApplication a(argc, argv);
51          Manusia suami;
52          suami.setNama("Susanto");
53          suami.setJenis_Kelamin('L');
54          Manusia istri("Susanti");
55          istri.setJenis_Kelamin('P');
56          Manusia anak("Rudi");
57          anak.setJenis_Kelamin('L');
58          suami.tampilSemua();
59          istri.tampilSemua();
60          anak.tampilSemua();
61          return a.exec();
62  }
```

Hasil:

```
Nama: Susanto, Jenis Kelamin: L
Nama: Susanti, Jenis Kelamin: P
Nama: Rudi, Jenis Kelamin: L
```

Keterangan:

- Program diatas hanya menjelaskan bagaimana kita dapat mengimplementasikan method member langsung didalam tubuh class, tidak diluar class.
- Hal seperti ini biasa dilakukan pada bahasa pemrograman berorientasi obyek lain seperti misalnya Java.
- Inline implementation tidak berbeda dengan non-inline implementation.

Class yang bertipe Class lain

Sangatlah mungkin kita membentuk class yang kompleks. Di dalam class tersebut member variabelnya dapat bertipe class lainnya. Contohnya adalah kita membuat class Mobil yang tentunya memiliki variabel member berupa class Roda, class Jok Mobil, class Mesin dan lain-lain. Contoh lain adalah class Garis yang terdiri dari class Titik. Class Bujursangkar juga dapat terdiri dari class Garis, dimana class Garis juga terdiri dari class Titik. Class dapat menjadi solusi yang baik untuk membuat tipe data baru yang memiliki member variabel dan member method yang tentunya sangat berguna.

Contoh 12. Class Mobil dan Class Roda.

Buatlah program berikut:

```
#include <QtCore/QCoreApplication>
#include <iostream>
using namespace std;
class Roda{
        private:
        string merk_roda;
        int diameter;
        public:
        Roda(string merk,int diamtr){
        this->diameter = diamtr;
        this->merk_roda = merk;
        }
        Roda(){
        }
        string getMerk(){
        return this->merk_roda;
        }
        int getDiameter(){
        return this->diameter;
        }
        void setMerkRoda(string m){
        this->merk_roda = m;
        }
        void setDiameter(int d){
        this->diameter = d;
        }
        };
        class Mobil{
        private:
        string merk_mobil;
        Roda roda_depan1;
        Roda roda_depan2;
        Roda roda_belakang1;
        Roda roda_belakang2;
        public:
        //konstruktor
```

```
37          Mobil(){
38          }
39          Mobil(string merk, Roda roda[4]){
40          this->merk_mobil = merk;
41          this->roda_depan1 = roda[0];
42          this->roda_depan2 = roda[1];
43          this->roda_belakang1 = roda[2];
44          this->roda_belakang2 = roda[3];
45          }
46          Mobil(Roda r1, Roda r2, Roda r3, Roda r4){
47          this->roda_depan1 = r1;
48          this->roda_depan2 = r2;
49          this->roda_belakang1 = r3;
50          this->roda_belakang2 = r4;
51          }
52          //desktruktor
53          ~Mobil(){
54          }
55          //accessor method
56          string getMerkMobil() const{
57          return this->merk_mobil;
58                  }
59          //mutator method
60          void setMerkMobil(string m){
61          this->merk_mobil = m;
62          }
63          void setRoda(Roda rd[4]){
64          this->roda_depan1 = rd[0];
65          this->roda_depan2 = rd[1];
66          this->roda_belakang1 = rd[2];
67          this->roda_belakang2 = rd[3];
68          }
69          //method lain
70          void tampilRoda(){
71          cout<<"Roda depan1:"<<endl;
72          cout<<"Merk: "<<this->roda_depan1.getMerk()<<endl;
```

```
73          cout<<"Diameter: "<<this->roda_depan1.getDiameter()<<endl;
74          cout<<"Roda depan2:"<<endl;
75          cout<<"Merk: "<<this->roda_depan2.getMerk()<<endl;
76          cout<<"Diameter: "<<this->roda_depan2.getDiameter()<<endl;
77          cout<<"Roda belakang1:"<<endl;
78          cout<<"Merk: "<<this->roda_belakang1.getMerk()<<endl;
79          cout<<"Diameter: "<<this->roda_belakang1.getDiameter()<<endl;
80          cout<<"Roda belakang2:"<<endl;
81          cout<<"Merk: "<<this->roda_belakang2.getMerk()<<endl;
82          cout<<"Diameter: "<<this->roda_belakang2.getDiameter()<<endl;
83          }
84          void tampilSemua(){
85          cout<<"Merk: "<<this->getMerkMobil()<<endl;
86          this->tampilRoda();
87          }
88  };
89  int main(int argc, char *argv[])
90  {
91          QCoreApplication a(argc, argv);
92          Roda r1("Bridgestone",40);
93          Roda r2("Bridgestone",40);
94          Roda r3("Bridgestone",40);
95          Roda r4("Bridgestone",40);
96          Mobil m1(r1,r2,r3,r4);
97          m1.setMerkMobil("Innova");
98          m1.tampilSemua();
99          return a.exec();
100 }
```

Hasil:

```
Merk: Innova
Roda depan1:
Merk: Bridgestone
Diameter: 40
```

```
Roda depan2:
Merk: Bridgestone
Diameter: 40
Roda belakang1:
Merk: Bridgestone
Diameter: 40
Roda belakang2:
Merk: Bridgestone
Diameter: 40
```

Keterangan:

- Program diatas mendemonstrasikan kepada kita bahwa kita dapat membuat class yang memiliki variabel member yang bertipe class lain.
- Cara mendeklarasikan variabel member bertipe class sama seperti cara mendefinisikan variabel member bertipe data biasa
- Variabel member yang bertipe data class akan memiliki sifat-sifat class tersebut.
- Pada contoh program diatas, class Mobil memiliki variabel member bertipe class Roda, maka variabel member `roda_depan1`, `roda_depan2`, `roda_belakang1`, dan `roda_belakang2` akan memiliki sifat-sifat class Roda, dimana kita dapa mengakses semua variabel member class Roda dan juga method member class Roda.
- Cara mengakses variabel member dan method member class Roda sama seperti biasa, yaitu dengan menggunakan tanda titik (.). Namun perlu diingat bahwa kita tidak dapat langsung mengakses variabel member class Roda karena variabel member tersebut bersifat private. Yang dapat kita lakukan adalah mengakses method member yang menenkapsulasi variabel member class Roda. Pada contoh diatas kita mengakses method `getMerk()` dan `getDiameter()`.

Contoh 12. Contoh Class Titik dan Garis.
Buatlah program berikut ini:

```
#include <QtCore/QCoreApplication>
#include <iostream>
#include <math.h>

using namespace std;

class Titik{
        private:
        int x;
        int y;
        public:

        Titik(int x,int y){
        this->x = x;
        this->y = y;
        }

        Titik(){
        }

        int getX(){
        return this->x;
        }

        int getY(){
        return this->y;
        }

        void setX(int x){
        this->x=x;
        }

        void setY(int y){
        this->y=y;
        }

```

```
37          //method tambahan
38          void printPoint(){
39          cout<<"("<<this->x<<" , "<<this->y<<")"<<endl;
40          }
41          bool isOrigin(){
42          return (this->x == 0 && this->y == 0);
43          }
44          };
45          class Garis{
46          private:
47          Titik p1;
48          Titik p2;
49          public:
50          Garis(int x1,int y1,int x2,int y2) {
51          p1.setX(x1);
52          p1.setY(y1);
53          p2.setX(x2);
54          p2.setY(y2);
55          }
56          Garis(){
57          }
58          Garis(Titik t1,Titik t2){
59          p1=t1;
60          p2=t2;
61          }
62          void setPoint1(Titik p1){
63          this->p1 = p1;
64          }
65          void setPoint2(Titik p2){
66          this->p2 = p2;
67          }
68          void setPoints(Titik p1,Titik p2){
69          this->p1 = p1;
70          this->p2 = p2;
71          }
72          Titik getPoint1(){
```

```
73          return this->p1;
74          }
75          Titik getPoint2(){
76          return this->p2;
77          }
78          void printLine(){
79          cout<<"awal: ";
80          this->p1.printPoint();
81                  cout<<"akhir: ";
82          this->p2.printPoint();
83          }
84          //Hitung panjang garis
85          float getLength(){
86          return sqrt((this->p1.getX() - this->p2.getX())*(this->p1.getX() - thi\
87  s-
88          >p2.getX()) + (this->p1.getY() - this->p2.getY())*(this->p1.getY() - t\
89  his-
90          >p2.getY()));
91          }
92  };
93
94  int main(int argc, char *argv[])
95  {
96          QCoreApplication a(argc, argv);
97          Titik A(1,1);
98          cout<<"A ";
99          A.printPoint();
100         Titik B(5,1);
101         cout<<"B ";
102         B.printPoint();
103         Titik C(5,6);
104         cout<<"C ";
105         C.printPoint();
106         Titik D(1,6);
107         cout<<"D ";
108         D.printPoint();
```

```
109         Garis ab(A,B);
110         cout<<"ab ";
111         ab.printLine();
112         cout<<"Panjang garis ab: "<<ab.getLength()<<endl;
113         Garis bc(B,C);
114         cout<<"bc ";
115         bc.printLine();
116         cout<<"Panjang garis bc: "<<bc.getLength()<<endl;
117         Garis cd(D,C);
118         cout<<"cd ";
119         cd.printLine();
120         cout<<"Panjang garis cd: "<<cd.getLength()<<endl;
121         Garis da(D,A);
122         cout<<"da ";
123         da.printLine();
124         cout<<"Panjang garis da: "<<da.getLength()<<endl;
125         return a.exec();
126 }
```

Hasil:

```
A (1 , 1)
B (5 , 1)
C (5 , 6)
D (1 , 6)
ab awal: (1 , 1)
akhir: (5 , 1)
Panjang garis ab: 4
bc awal: (5 , 1)
akhir: (5 , 6)
Panjang garis bc: 5
cd awal: (1 , 6)
akhir: (5 , 6)
Panjang garis cd: 4
da awal: (1 , 6)
akhir: (1 , 1)
```

```
Panjang garis da: 5
```

Keterangan:

- Program diatas juga menunjukkan contoh lain dari suatu class yang memiliki member variabel yang bertipe class lain. Pada contoh diatas class Garis memiliki variabel member yang berasal dari class Titik. Sehingga dari obyek Garis kita dapat mengakses semua method member class Titik.
- Dengan menggunakan rumus matematis perhitungan jarak antara dua buah koordinat (titik), maka kita bisa menghitung panjang garis. Untuk perhitungan dibutuhkan function sqrt yang berarti akar kuadrat, sehingga kita harus memasukan `header math.h`
- Method `isOrigin` pada class Titik digunakan untuk mengetahui apakah suatu koordinat berada di titik `0,0` atau tidak. Kita dapat menambahkan method lain yang sesuai kebutuhan kita.
- Di dalam kelas Garis kita memiliki beberapa konstruktor, ada yang tidak berparameter, ada yang berparameter 2 Titik dan berparameter 4 koordinat. Semuanya itu digunakan untuk tujuan yang sama, yaitu menciptakan obyek Titik pada member variabel class Garis, karena Garis pada dasarnya adalah terdiri dari 2 buah Titik.

TIPS

Pada bahasa C++ kita tidak dapat memanggil konstruktor dari dalam konstruktor lain yang berada dalam satu class.

Contoh:

```
class Halaman{
        private:
        int nohal;
        int jenishal;
        //1 -> halaman biasa, 2 -> halaman header
        public:

        Halaman(){
        }

        Halaman(int nohal){
        this->nohal = nohal;
        }

        Halaman(int nohal,int jenishal){
        Halaman(nohal); //memanggil konstruktor Halaman diatasnya!
        this->jenishal = jenishal;
        }
};
```

Akan menghasilkan error!