

B E L A J A R

C + +

dengan Qt Creator



Code less.
Create more.
Deploy everywhere.



© 2016 Nur Wachid
All rights reserved.

Buku ini boleh digandakan atau didistribusikan dengan seizin penulis atau mencantumkan *credit* kepada website penulis atau email penulis. Tetapi pendistribusian buku ini tidak boleh untuk kepentingan komersial.

Penulis: Nur Wachid

Website: <http://turahe.github.io>

Email: wachid (at) outlook dot com

0858 7984 5219

Tentang Penulis

Nur Wachid



Belajar C++ dengan Qt Creator

Daftar Isi

| | |
|--|------------|
| Tentang Penulis | iii |
| Prakata | xix |
| | |
| I Dasar | 1 |
| 1 Mukadimah | 3 |
| 1.1 Pengenalan Bahasa C++ | 4 |
| 1.2 Pengantar Qt Creator | 6 |
| 1.2.1 UI Designer | 8 |
| 1.2.2 Bahasa yang di dukung | 11 |
| 1.2.3 Platform | 11 |
| 1.2.4 Tools | 11 |
| 1.2.5 Debuggers | 12 |
| 1.3 Mengenal macam - macam Teknologi User Interface (UI) pada QT | 12 |
| 1.3.1 Teknologi User Interface QtQuick | 13 |
| 1.3.2 Teknologi User Interface QWidget | 14 |
| 1.3.3 Teknologi User Interface QtWebkit | 15 |
| 1.4 Install Qt Creator | 16 |
| 1.4.1 Install Qt Creator di Ubuntu 14.04 | 16 |
| 1.4.2 Install Qt di Windows | 18 |



| | | |
|----------|---|-----------|
| 1.5 | Program Console Pertama dengan Qt Creator | 20 |
| 1.6 | Struktur Program C++ | 23 |
| 2 | Tipe Data, Identifier, Operator dan Control Statement | 27 |
| 2.1 | Tipe Data dan Identifier | 28 |
| 2.2 | Tipe Data Bahasa C++ | 29 |
| 2.3 | Variabel dan Konstanta | 29 |
| 2.4 | Statement | 33 |
| 2.5 | Operator dan Ekspresi | 33 |
| 2.5.1 | Operator Unary | 36 |
| 2.5.2 | Operator Unary Minus | 36 |
| 2.5.3 | Operator Unary ++ dan – | 37 |
| 2.5.4 | Operator Penggerjaan | 37 |
| 2.5.5 | Operator Hubungan | 38 |
| 2.5.6 | Operator Logika | 39 |
| 2.6 | Control Statement | 40 |
| 2.6.1 | Percabangan | 41 |
| 2.6.2 | Perulangan | 47 |
| 3 | Array dan String | 53 |
| 3.1 | Array | 54 |
| 3.1.1 | Array 1 Dimensi | 55 |
| 3.1.2 | Inisialisasi Array Satu Dimensi | 61 |
| 3.1.3 | Pengalamatan dan Pengkopian Array 1 Dimensi | 67 |
| 3.1.4 | Array Multi Dimensi | 70 |
| 3.1.5 | Cara Pengaksesan Array dapat dilakukan dengan 2 cara: | 81 |
| 3.2 | String | 83 |
| 3.2.1 | Array of Character | 83 |
| 3.3 | Fungsi-fungsi String | 88 |
| 3.3.1 | strcpy() dan strncpy() | 90 |
| 3.3.2 | strcat() | 93 |
| 3.3.3 | Fungsi mengubah string menjadi numerik dan sebaliknya | 96 |
| 3.3.4 | Class string pada C++ | 96 |

| | |
|--|------------|
| 4 Fungsi | 109 |
| 4.1 Konsep Dasar Fungsi | 111 |
| 4.2 Mendefinisikan Fungsi | 112 |
| 4.3 Deklarasi Fungsi (Prototype) | 113 |
| 4.4 Hasil Balik Fungsi | 116 |
| 4.5 Ruang Lingkup Variabel | 118 |
| 4.5.1 Variable Lokal | 118 |
| 4.5.2 Variable Global | 120 |
| 4.5.3 Variabel statik | 122 |
| 4.6 Pengiriman Parameter | 124 |
| 4.7 Parameter Default | 125 |
| 5 Pointer dan References | 129 |
| 5.1 Apa itu Pointer? | 130 |
| 5.2 Memory Komputer | 130 |
| 5.2.1 Mengambil Alamat Memory dari Variabel | 130 |
| 5.2.2 Menyimpan Alamat Variabel pada Pointer | 132 |
| 5.2.3 Memberi Nama Pointer | 133 |
| 5.2.4 Mengambil Nilai dari Variabel | 133 |
| 5.2.5 Mengganti alamat yang direferensi oleh Pointer | 136 |
| 5.3 Pointer dan Array | 137 |
| 5.3.1 Kapan kita menggunakan pointer? | 139 |
| 5.3.2 Membuat objek pada heap | 142 |
| 5.3.3 Menggunakan const Pointer | 145 |
| 5.4 Apa itu Reference | 145 |
| 5.4.1 Re-assign Reference Variable | 148 |
| 5.4.2 Passing function argument dengan reference | 149 |
| 5.4.3 Function yang mengembalikan beberapa nilai | 153 |
| 5.4.4 Passing By Reference untuk Efisiensi | 157 |
| II Objek Oriented Programming (OOP) | 163 |
| 6 Class dan Object | 165 |



| | | |
|----------|---|------------|
| 6.1 | Pemrograman Berorientasi Obyek | 166 |
| 6.2 | Kelas | 168 |
| 6.3 | Object | 168 |
| 6.4 | Class dan Object | 169 |
| 6.5 | Pembuatan Class pada C++ | 169 |
| 6.6 | Mendefinisikan Obyek | 171 |
| 6.7 | Mengakses Member Variabel | 171 |
| 6.8 | Mengakses Member Function/Method | 172 |
| 6.9 | Hak Akses Member Variabel dan Method Variabel | 180 |
| 6.10 | Member Function / Member Method | 184 |
| 6.11 | Accessor dan Mutator Method | 187 |
| 6.11.1 | Accessor method | 188 |
| 6.11.2 | Mutator method | 188 |
| 6.12 | Constructor dan Destructor | 191 |
| 6.12.1 | Default Constructor | 192 |
| 6.12.2 | Constructor Dengan nilai Default | 199 |
| 6.12.3 | const member method | 202 |
| 6.13 | Mendefinisikan Method Member | 206 |
| 6.14 | Class yang bertipe Class lain | 209 |
| 7 | Inheritance | 221 |
| 7.1 | Pewarisan (Inheritance) | 222 |
| 7.2 | Penulisan Penurunan | 222 |
| 7.3 | Jenis Akses Penurunan Kelas | 225 |
| 7.4 | Warisan | 227 |
| 7.4.1 | Tiap Kelas Mempunyai Konstruktor | 228 |
| 7.4.2 | Konstruktor Kelas Turunan Pasti Memanggil Konstruktor Kelas Dasar | 230 |
| 7.5 | Mengganti Metode Kelas Dasar Pada Kelas Turunan (Overriding) | 237 |
| 7.6 | Memanggil Metode Kelas Dasar | 241 |
| 7.7 | Penyembunyian Metode Kelas Dasar | 243 |
| 7.8 | Metode Virtual | 245 |
| 7.9 | Pemotongan (Slicing) | 249 |
| 7.10 | Memakai static_cast | 251 |



| | |
|---|------------|
| 8 Operator Types dan Operator Overloading | 255 |
| 8.1 Operator pada C++ | 256 |
| 8.1.1 Unary Operator | 257 |
| 8.2 Conversion Operator | 263 |
| 8.2.1 Binary Operator | 266 |
| 8.3 Addition-Assignment Operator | 269 |
| 8.4 Comparison Operator | 272 |
| 8.5 Overloading Operator <, >, <=, >= | 277 |
| 8.6 Subscript Operator | 280 |
| 8.7 Function operator() | 283 |
| 9 Polymorphism | 285 |
| 9.1 Problema Pewarisan Tunggal (Single Inheritance) | 287 |
| 9.2 Peletakan ke atas (Pecolating Upward) | 292 |
| 9.3 Konversi ke bawah (Casting Down) | 293 |
| 9.4 Menambahkan ke Dua Daftar | 296 |
| 9.5 Pewarisan Ganda (Multiple Inheritance) | 297 |
| 9.6 Komponen Objek Multi Inheritance | 302 |
| 9.7 Konstruktor Kelas Banyak Turunan (Multiple Inheritance) | 303 |
| 9.8 Problem Ambiguitas | 305 |
| 9.9 Penurunan dari Kelas Dasar Bersama | 306 |
| 9.10 Penurunan Virtual (Virtual Inheritance) | 311 |
| 9.11 Masalah Pada Multiple Inheritance | 314 |
| 10 Casting dan Database | 317 |
| 10.1 Mengenal Casting | 318 |
| 10.2 Data Type Casting (Conversion) | 319 |
| 10.2.1 Implicit conversion | 319 |
| 10.2.2 Explicit conversion | 320 |
| 10.3 Casting Operator pada C++ | 324 |
| 10.3.1 Penggunaan static_cast | 324 |
| 10.3.2 Penggunaan dynamic_cast | 326 |
| 10.3.3 Penggunaan reinterpret_cast | 330 |
| 10.3.4 Penggunaan const_cast | 332 |



| | |
|---|------------|
| 10.4 Pemrograman Basisdata dengan QtConsole Application | 334 |
| 10.4.1 Koneksi Qt dengan Basisdata | 335 |
| 10.4.2 Koneksi Qt dengan MySQL dan menampilkan datanya . . | 335 |
| 10.4.3 Koneksi Qt dengan SQLite | 340 |
| 10.4.4 Membaca data pada tabel SQLITE | 344 |
| 10.4.5 Menambah data pada tabel SQLITE | 346 |
| 10.4.6 Mengedit data pada tabel SQLITE | 350 |
| 10.4.7 Menghapus data pada tabel SQLITE | 352 |
| III Interface | 369 |
| 11 GUI | 371 |
| 11.1 Pemrograman aplikasi GUI | 371 |
| 11.1.1 Menyiapkan Proyek | 372 |
| 11.1.2 Membuat aplikasi | 375 |
| IV Widget | 377 |
| 12 File, Stream, dan XML | 379 |
| 12.1 Bekerja dengan Paths | 380 |
| 12.2 Bekerja dengan Files | 384 |
| 12.3 Bekerja dengan Stream | 385 |
| 12.3.1 Text Stream | 386 |
| 12.3.2 Data Stream | 388 |
| 12.4 XML | 392 |
| 12.5 DOM | 393 |
| 12.5.1 Membuat File XML | 393 |
| 12.5.2 Membaca XML file | 395 |
| 12.5.3 Memodifikasi File XML | 398 |
| 12.6 QXMLStream Reader | 403 |
| 13 Qt Webkit | 409 |



| | |
|---|------------|
| 13.1 Qt Webkit | 410 |
| 13.2 My Web Browser version 2 | 413 |
| 13.2.1 Starting the Project | 413 |
| 13.2.2 Working on UI | 416 |
| 13.3 Writing Code for the Slots | 418 |
| 13.3.1 Running the Code | 420 |
| 13.3.2 Source Code | 420 |
| V Library | 425 |
| 14 Library | 427 |
| 14.1 Qt Library | 428 |
| 14.2 Menurunkan objek dari class QObject | 429 |
| 14.3 Automatic Memory Management dengan QObject | 429 |
| 14.4 Menggunakan Qt String | 435 |
| 14.5 Collection dan Iterator | 441 |
| 14.5.1 Menggunakan QList | 441 |
| 14.5.2 QList::const_iterator | 444 |
| 14.6 Menambahkan Data pada List | 446 |
| 14.7 Tipe List yang Lain | 447 |
| 14.8 Special List | 448 |
| 14.9 Stack dan Queue | 450 |
| 14.10 Mapping | 452 |
| VI Lampiran | 455 |
| A | 457 |



Listings

| | | |
|------|---|----|
| 1.1 | Struktur Program C++ | 23 |
| 1.2 | Contoh struktur program C++ | 25 |
| 2.1 | Tipe data dan Identifier | 32 |
| 2.2 | Tipe data dan Identifier | 45 |
| 3.1 | Input dan Output Array | 56 |
| 3.2 | Manipulasi Array | 59 |
| 3.3 | Penanganan Batas Indeks Elemen Array | 60 |
| 3.4 | Inisialisasi Array dengan nilai \0 | 62 |
| 3.5 | Inisialisasi Array dua nilai elemen pertama | 63 |
| 3.6 | Tanpa inisialisasi array langsung ditampilkan | 65 |
| 3.7 | Penggunaan tipe data enum pada Array satu dimensi | 66 |
| 3.8 | Percobaan Penyalinan Array 1 dimensi | 68 |
| 3.9 | Penyalinan Array 1 dimensi dengan Perulangan | 69 |
| 3.10 | Deklarasi dan Menampilkan Array 2 Dimensi | 76 |
| 3.11 | Penyalinan Array 2 Dimensi ke Array 2 Dimensi lainnya | 77 |
| 3.12 | Penyalinan array 2 dimensi ke dalam array 1 dimensi | 79 |
| 3.13 | Pengaksesan Baris demi Baris | 81 |
| 3.14 | Pengaksesan Kolom demi Kolom | 82 |
| 3.15 | Penggunaan karakter \0 | 84 |
| 3.16 | String tanpa karakter \0 | 85 |
| 3.17 | Mengisi Array of Character | 86 |
| 3.18 | Pengisian variabel array of character dengan maksimum jumlah karakter | 87 |



| | |
|--|-----|
| 3.19 Penggunaan fungsi strlen() | 88 |
| 3.20 Penggunaan fungsi length pada tipe data string C++ | 89 |
| 3.21 Penggunaan fungsi strcpy() | 90 |
| 3.22 Penggunaan fungsi strncpy() | 92 |
| 3.23 Penggunaan fungsi strcat() | 93 |
| 3.24 Penggunaan fungsi strcat() | 94 |
| 3.25 Pembuatan variabel string C++ penyalinan string dan penggabungan string | 97 |
| 3.26 Penggunaan class string untuk manipulasi data | 99 |
| 3.27 Penggabungan string dengan menggunakan class string | 100 |
| 3.28 Pengaksesan isi nilai class string | 101 |
| 3.29 Menemukan substring pada sebuah string besar | 103 |
| 3.30 Membalik kata / kalimat | 106 |
| 3.31 Konversi huruf besar dan kecil | 107 |
| 4.1 Membuat Fungsi yang mengembalikan nilai | 115 |
| 4.2 Membuat Fungsi yang tidak mengembalikan nilai | 117 |
| 4.3 Variabel Lokal | 119 |
| 4.4 Variabel Global | 120 |
| 4.5 Variabel Statik | 122 |
| 4.6 Default Parameter | 127 |
| 5.1 Menampilkan alamat memory menggunakan address-of operator | 131 |
| 5.2 Memanipulasi data menggunakan Pointer | 134 |
| 5.3 Mengganti alamat yang di referensi oleh pointer | 136 |
| 5.4 Pointer dan Array | 138 |
| 5.5 Mengalokasikan menggunakan dan mendelete Pointer | 141 |
| 5.6 Membuat dan menghapus objek dari Heap | 143 |
| 5.7 Membuat dan Menggunakan Reference | 146 |
| 5.8 Re-assign Reference Value | 148 |
| 5.9 Passing by Value | 149 |
| 5.10 Passing by reference dengan pointer | 151 |
| 5.11 Menjalankan fungsi Tukar() dengan reference | 152 |
| 5.12 Mengembalikan beberapa nilai dengan pointer | 154 |
| 5.13 Mengembalikan beberapa nilai dengan reference variabel | 156 |
| 5.14 Passing Object By Value | 158 |



| | | |
|------|--|-----|
| 5.15 | Passing Object By Reference | 160 |
| 6.1 | Pembuatan class Sepeda | 172 |
| 6.2 | Pembuatan obyek Sepeda | 175 |
| 6.3 | Pembuatan Obyek Array Sepeda | 177 |
| 6.4 | Public pada member variabel | 181 |
| 6.5 | Privat pada member variabel | 183 |
| 6.6 | Member function dan implementasinya | 185 |
| 6.7 | Penggunaan accessor dan mutator method | 188 |
| 6.8 | Menggunakan Constructor dan Destructor | 193 |
| 6.9 | Percobaan Menambah Constructor Method | 196 |
| 6.10 | Penggunaan Constructor dengan Nilai Default | 199 |
| 6.11 | Memberi nilai default pada constructor | 201 |
| 6.12 | Penggunaan const method | 203 |
| 6.13 | Implementasi inline | 207 |
| 6.14 | Class Mobil dan Roda | 209 |
| 6.15 | Class titik dan Garis | 214 |
| 7.1 | Pewarisan (Inheritance) | 223 |
| 7.2 | Jenis Akses Public Pada Penurunan | 226 |
| 7.3 | Konstruktor default | 229 |
| 7.4 | Konstruktor default kelas turunan | 231 |
| 7.5 | Konstruktor default kelas turunan memanggil konstruktor kelas dasar | 232 |
| 7.6 | Konstruktor kelas turunan harus memanggil salah satu konstruktor kelas dasar | 235 |
| 7.7 | Melakukan Overriding | 238 |
| 7.8 | Mengakses metode kelas dasar | 241 |
| 7.9 | Penyembunyian metode kelas dasar | 243 |
| 7.10 | Menghilangkan comment pada metode hallo | 244 |
| 7.11 | Metode virtual dan non virtual | 246 |
| 7.12 | Menghapus methode getluas | 247 |
| 7.13 | Metode virtual dan non virtual | 249 |
| 7.14 | Memakai static_cast | 252 |
| 8.1 | Menggunakan Increment Operator (Notasi prefix) | 258 |
| 8.2 | Menggunakan Operator Increment (notasi postfix) | 261 |



| | |
|--|-----|
| 8.3 Conversion Operator untuk konversi class Kalender ke integer | 264 |
| 8.4 Menggunakan Binary Addition Operator | 267 |
| 8.5 Menggunakan Addition Assignment Operator dan Subtraction Assignment Operator | 269 |
| 8.6 Overloading Comparison Operator (Equality dan Inequality) | 272 |
| 8.7 Menggunakan Operator Overloading | 277 |
| 8.8 Subscript Operator untuk Dynamic Array | 281 |
| 8.9 Menggunakan operator() untuk membuat function object | 283 |
| 9.1 Meletakkan metode kelas turunan di kelas dasar | 289 |
| 9.2 Melakukan Down Casting | 294 |
| 9.3 Multiple Inheritance | 297 |
| 9.4 Konstruktor Kelas Multiple Inheritance | 303 |
| 9.5 Penurunan pada umumnya (Common Base Class) | 308 |
| 9.6 Penurunan pada umumnya (Common Base Class) | 311 |
| 10.1 Explicit Casting pada Tipe Data Numerik | 321 |
| 10.2 Contoh Dynamic Casting | 328 |
| 10.3 Percobaan koneksi MySQL dengan QtConsole | 337 |
| 10.4 Percobaan koneksi SQLite dengan QtConsole | 342 |
| 10.5 Membaca data pada Sqlite | 345 |
| 10.6 Menambahkan data pada SQLite | 347 |
| 10.7 Membaca data pada SQLite | 349 |
| 10.8 menggunakan class QSqlQuery dan method query | 350 |
| 10.9 Mengedit data pada SQLite | 351 |
| 10.10 menghapus data adalah dengan menggunakan class QSqlQuery dan method query | 353 |
| 10.11 Menghapus data pada SQLite | 354 |
| 10.12 Pembuatan manipulasi data pada SQLite dengan menggunakan menu | 356 |
| 12.1 Menampilkan daftar drives dari root directories. | 381 |
| 12.2 Memeriksa apakah file ada dan bisa diakses | 384 |
| 12.3 Menggunakan Stream untuk membaca file | 386 |
| 12.4 file pro untuk membuka library Qt GUI | 388 |
| 12.5 Menggunakan Data Stream | 389 |
| 12.6 Membuat Nodes untuk membuat simple XML Document | 393 |



| | |
|---|-----|
| 12.7 Contoh Membaca DOM dari dokumen XML | 396 |
| 12.8 Main program Modifikasi data dokumen XML | 399 |
| 12.9 Menggunakan QXMLStream Reader untuk membaca XML | 404 |
| 12.10 Membuat dokumen XML dengan QXMLStreamWriter | 406 |
| 14.1 Alokasi memory dinamis tanpa QObject | 430 |
| 14.2 Alokasi memory dinamis dengan QObject | 433 |
| 14.3 Cek apakah nilai QString Null atau Empty | 436 |
| 14.4 Menggunakan Fungsi Left Mid Right | 437 |
| 14.5 Menggabungkan String | 438 |
| 14.6 Membalik String | 439 |
| 14.7 Menggunakan QList | 441 |
| 14.8 Menggunakan object Iterator | 443 |
| 14.9 Menggunakan Iterator untuk memodifikasi data di list | 445 |
| 14.10 Beberapa cara menambahkan data ke list | 446 |
| 14.11 Menggunakan QStringList | 448 |
| 14.12 Menggunakan Stack dan Queue | 450 |
| 14.13 Menggunakan QMap | 452 |



Prakata

Bahasa pemrograman C/C++ merupakan bahasa yang popular didalam pengajaran pada computer sains maupun pada kalangan programmer yang mengembangkan system software maupun aplikasi.

Bahasa C/C++ sifatnya portable, karena compilernya tersedia hampir pada semua arsitektur computer maupun system operasi, sehingga investasi waktu dan tenaga yang anda lakukan dalam mempelajari bahasa pemrograman ini memiliki nilai strategis yang sangat menjanjikan.

Bahasa C/C++ merupakan bahasa yang sangat ketat dalam pemakaian type data maupun penulisannya yang case sensitif, hal ini berarti programmer di tuntut kedisiplinannya dalam penulisan program.

Sesuatu fasilitas yang tersedia dalam C/C++ yang tidak ditemukan pada bahasa pemrograman lainnya adalah pointer, dengan pemanfaatan pointer programmer dapat melakukan manipulasi memori secara langsung.

Dewasa ini beberapa bahasa yang memiliki syntax penulisan yang menyerupai C/C++ adalah Java, Javascript dan PHP, yang artinya bahwa kemampuan pemrograman dengan C/C++ akan mempermudah anda untuk mempelajari bahasa modern seperti Java maupun C# (dibaca C sharp).

Akhirnya penulis mengucapkan selamat belajar dan semoga buku ini dapat memberi manfaat yang sebesarnya dalam pembelajaran mata kuliah C/C++ Programming.



Struktur buku ini

- Bab 1. Mukadimah** Pada bab ini akan di perkenalkan mengenai bahasa C++, mengenal Qt Creator, teknologi User Interface yang digunakan, Instalasi dan Struktur program C++ pada Qt Creator secara umum
- Bab 2. tipe data, identifier, Operator dan Control Flow** pada bab ini akan dibahas tuntas mengenai apa itu tipe data C++, Variabel dan konsanta, Statement, Operator dan Control Statement.
- Bab 3. Array dan String** Pada bab ini akan di perkenalkan macam-macam array dari array 1 dimensi sampai dengan array multidimesi dan String berupa penggunaan string di dalam bahasa C++.
- Bab 4. Fungsi** Pada bab ini akan mempelari tentang konsep dasar fungsi cara mendefinisikan fungsi dan medeklarasikanya, Hasil balik fungsi serta mempelajari ruang lingkup variabel dan pengiriman parameter.
- Bab 5. Pointer dan Array** Pada bab ini akan di perkenalkan mengenai pointer di dalam komputer dan cara penggunaan dalam bahasa C++ kemudian tentang Array dan sedikit tentang reference.
- Bab 6. Class dan Objek** Pada bab ini akan di paparkan secara gamblang mengenai pemrograman berorientasi objek dengan memperkenalkan struktur dasar dari OOP yakni Class dan objek.
- Bab 7. Inheritance** Pada bab ini akan di perkenalakan tentang Inheritance yakni tentang pemakaian kode yang telah ada untuk di gunakan kembali.
- Bab 8. Operator dan Operator Overloading** Pada bab awal kita sudah mempelajari berbagai macam operator (+, -, /, >, <) yang dapat digunakan pada tipe data yang sudah ada di C++ seperti int, float, bool, dll. Namun jika anda ingin menggunakan operator tersebut pada tipe data yang Anda definisikan sendiri seperti tipe data Class, maka anda dapat menggunakan keyword operator.
- Bab 9. Polymorphism** Pada bab ini akan mempelajari tentang Polymorphism



yakni suatu konsep untuk merelasikan diantara kelas-kelas C++ melalui overriding metode-metode virtual, sehingga dengan demikian satu tipe kelas dapat dikonversikan menjadi kelas lain. Aspek penting pertama dalam Pewarisan (*Inheritance*) adalah pengorganisasian kelas yang mengijinkan kelas lain berbagi program dan data (*code reuse*) sehingga program tidak harus dibangun ulang dari awal. Aspek penting kedua dari Pewarisan (*Inheritance*) adalah pengelompokan fitur-fitur kelas yang serupa kedalam sebuah kelas dasar (*base class*) dan kemudian membuat kelas lain dengan cara menurunkan kelas tersebut sehingga bentuk utama/ pokok dari kelas-kelas turunan menjadi serupa dengan kelas dasar sedemikian rupa sehingga **pointer** atau referensi bertipe kelas dasar dapat menerima nilai berbagai macam bentuk objek bertipe kelas turunannya (berubah tipe kelas) dan dapat mengeksekusi fitur-fitur yang serupa tersebut.

Bab 10. Casting dan Database Casting merupakan mekanisme dimana programmer dapat secara permanen atau temporary mengubah interpretasi compiler terhadap suatu obyek. Perubahan ini tidak benar-benar terjadi, namun hanya cara pandang compiler saja yang diubah. Casting diimplementasikan dalam bentuk “casting operator”. Mengapa butuh casting? Dalam dunia pemrograman yang semuanya jelas (strong type language) dan jika kita hanya menggunakan satu bahasa pemrograman saja, seperti C++, maka kita tidak membutuhkan operator casting. Namun kenyataannya pada dunia nyata yang kita hadapi, banyak bahasa pemrograman, banyak vendor-vendor berbeda-beda sehingga kode / modul yg dihasilkan jd berbeda-beda. Hal ini menyebabkan compiler-compiler bahasa pemrograman tertentu, termasuk C++ juga harus diubah interpretasinya dengan cara lain sehingga mampu melakukan kompilasi dan menghasilkan hasil yang kompatibel.

Bab 11. GUI Pada bab ini akan memamparkan bagaimana membuat sebuah aplikasi GUI sederhana dengan mudah menggunakan Qt Creator . Aplikasi sederhana dengan fungsi dasar tombol dan label (untuk gambar). Jika tombol pertama di klik, maka gambar A muncul. Jika tombol kedua di klik, maka gambar B muncul. Dengan contoh dasar tersebut di harapkan mamou memahami dasar-dasar dari sebuah pemrograman GUI di Qt Creator yakni



signal dan slot.

Bab 12. File, Stream dan XML Pada bab ini kita akan membahas tentang beberapa class khusus pada Qt Framework yang digunakan untuk bekerja dengan File dan dokumen XML.

Bab 13.

Bab 14. Library Pada bab ini akan mempelajari mengenai Qt SDK menyediakan beberapa class library yang dapat anda gunakan untuk mempercepat pembuatan program, misalnya library untuk membuat GUI (Graphical User Interface), network programming, dan library untuk bekerja dengan XML.



Bagian I

Dasar



BAB 1

Mukadimah

Agenda

Pada chapter ini kita akan membahas mengenai bahasa C dan Qt Creator seperti berikut ini

Contents

| | | |
|-------|--|----|
| 1.1 | Pengenalan Bahasa C++ | 4 |
| 1.2 | Pengantar Qt Creator | 6 |
| 1.2.1 | UI Designer | 8 |
| 1.2.2 | Bahasa yang di dukung | 11 |
| 1.2.3 | Platform | 11 |
| 1.2.4 | Tools | 11 |
| 1.2.5 | Debuggers | 12 |
| 1.3 | Mengenal macam - macam Teknologi User Interface (UI) pada QT | 12 |
| 1.3.1 | Teknologi User Interface QtQuick | 13 |
| 1.3.2 | Teknologi User Interface QWidget | 14 |



| | | |
|------------|--|-----------|
| 1.3.3 | Teknologi User Interface QtWebkit | 15 |
| 1.4 | Install Qt Creator | 16 |
| 1.4.1 | Install Qt Creator di Ubuntu 14.04 | 16 |
| 1.4.2 | Install Qt di Windows | 18 |
| 1.5 | Program Console Pertama dengan Qt Creator | 20 |
| 1.6 | Struktur Program C++ | 23 |

1.1 Pengenalan Bahasa C++

Bahasa C merupakan bahasa pemrograman tingkat menengah. Pada tahun 1972 bahasa C pertama kali dirancang oleh [Dennis M Ritchie](#)¹Dennis M Ritchie di Bell laboratories. Kemudian tahun 1978 Dennis dan [Brian W. Kernighan](#)²Brian W. Kernighan 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++ Pro-*

¹https://id.wikipedia.org/wiki/Dennis_Ritchie

²https://id.wikipedia.org/wiki/Brian_Kernighan

³https://id.wikipedia.org/wiki/The_C_Programming_Language

⁴https://id.wikipedia.org/wiki/ANSI_C



gramming 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
- Kedua, [Borland International Inc](#)⁷. mengembangkan kompiler C++ menjadi sebuah kompiler yang mampu mengubah C++ langsung menjadi bahasa mesin (assembly). Tahun 1990, 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 sejagannya pada waktu itu. Bahasa C++ termasuk bahasa pemrograman tua yang sudah mendukung berbagai macam kata kunci yang mampu menyingkat proses penulisan kode program.

⁵Bell Laboratories (juga dikenal dengan nama Bell Labs dan sebelumnya dengan nama AT&T Bell Laboratories dan Bell Telephone Laboratories) adalah bagian dari organisasi riset dan pengembangan dari Alcatel-Lucent dan sebelumnya dari United States Bell System.

⁶<https://en.wikipedia.org/wiki/Cfront>

⁷Borland Software Corporation adalah sebuah perusahaan perangkat lunak komputer yang berkantor pusat di Austin, Texas. Perusahaan ini didirikan pada tahun 1983 oleh Niels Jensen, Ole Henriksen, Mogens Glad dan Philippe Kahn.

⁸OOP (Object Oriented Programming) adalah suatu metode pemrograman yang berorientasi kepada objek. Tujuan dari OOP diciptakan adalah untuk mempermudah pengembangan program dengan cara mengikuti model yang telah ada di kehidupan sehari-hari.



Modular programming⁹ 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 kodennya.

Speed Kode yang dihasilkan dari kompilasi C++ sangat efisien, karena C++ mendukung prinsip dualitas bahwa dia mendukung bahasa tingkat tinggi dan bahasa tingkat rendah sehingga dapat mengurangi ukuran hasil kompilasi 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 MinGW¹⁰ atau MSVC¹¹ yang su-

¹⁰ minGW adalah salah satu aplikasi yg digunakan untuk mengkompile bahasa C agar dapat dipahami oleh bahasa mesin (asembler) pada komputer. Aplikasi ini dapat di unduh secara gratis.

¹¹ sebuah perangkat lunak lengkap (suite) yang dapat digunakan untuk melakukan pengembangan aplikasi, baik itu aplikasi bisnis, aplikasi personal, ataupun komponen aplikasinya, dalam bentuk aplikasi console, aplikasi Windows, ataupun aplikasi Web. Visual Studio mencakup kompiler, SDK, Integrated Development Environment (IDE), dan dokumentasi (umumnya berupa MSDN Library). Kompiler yang dimasukkan ke dalam paket Visual Studio antara lain Visual C++, Visu-



dah build-in di dalam install.

Project Qt Creator seperti pada gambar ?? 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 generate 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 berekstensi .ui yang akan imodifikasi melalui Qt Designer. Qt Creator diintervasikan 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.

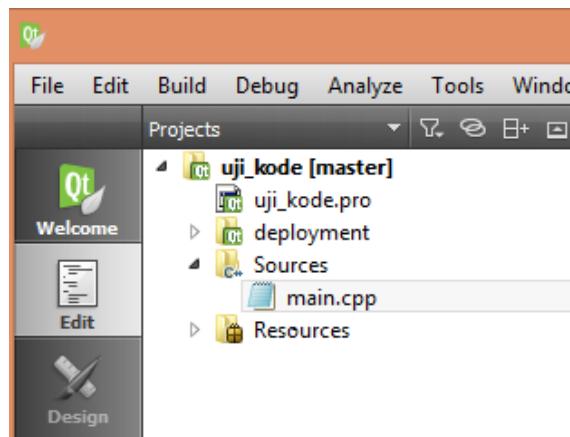
Editor Qt Creator seperti pada gambar ?? mempunyai sebuah code editor yang telah terintegrasi dengan Qt Designer untuk mendesain dan membangun aplikasi GUI dari Qt widgets. Karena Qt Creator adalah sebuah Integrated Development Environment (IDE), maka Qt Creator memisahkan antara text editor untuk build 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.

al C#, Visual Basic, Visual Basic .NET, Visual InterDev, Visual J++, Visual J#, Visual FoxPro, dan Visual SourceSafe.





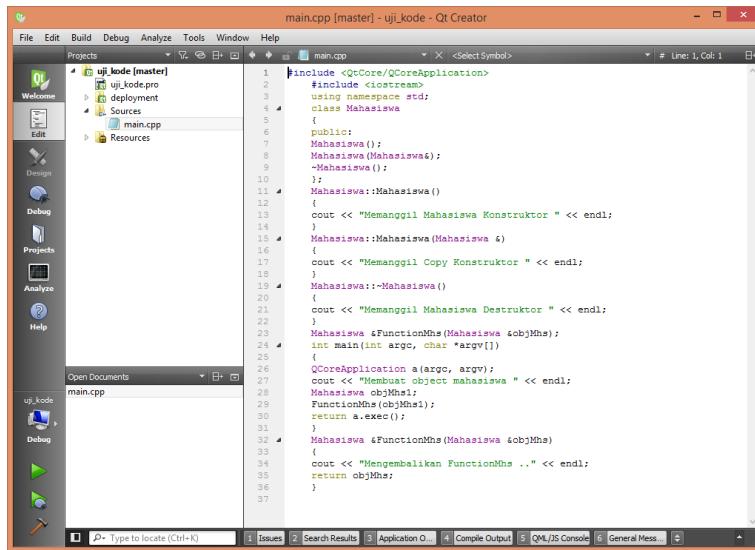
Gambar 1.1: Projek pada Qt Creator

- 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

1.2.1 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



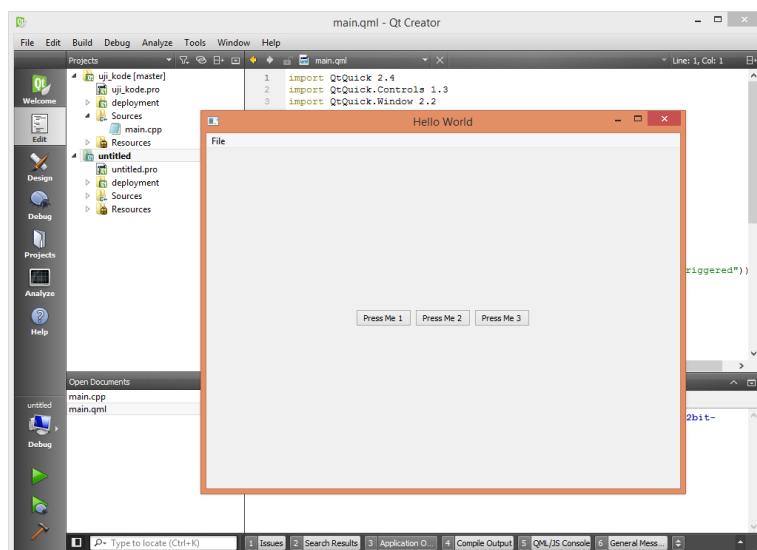


Gambar 1.2: Code Editor

elemen-elemen grafik. Semua properti-properti yang diatur pada Qt Designer dapat diubah secara dinamik melalui/di dalam code.

Qt Quick Designer seperti gambar 1.3 digunakan untuk membangun secara mudah animasi-animasi dengan menggunakan sebuah bahasa pemrograman yang dikenal dengan [Qt Meta-Object Language \(QML\)](#)¹². Dalam QML, sebuah user interface dispesifikasikan sebagai sebuah pohon (tree) dari objects dengan properti-properti. anda bisa menggunakan teks editor visual untuk menciptakan items, screens, dan aplikasi, serta mendefinisikan perubahan action-action pada komponennya. Dapat digunakan Qt atau JavaScript untuk mengimplementasikan logika aplikasi.

¹²<https://en.wikipedia.org/wiki/QML>



Gambar 1.3: Gambar UI Designers



1.2.2 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.

1.2.3 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 meng-koneksi ke Personal Computer (PC), maka Qt Creator men-generate sebuah package instalasi, menginstall package tersebut pada device, dan meneksekusikannya.

1.2.4 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 ([Git¹³](#), [subversion¹⁴](#), [Perforce¹⁵](#), [CVS¹⁶](#), [Mercurial¹⁷](#)).

¹³Git adalah tools yang berfungsi sebagai Version Control System (VCS) dan kalau diartikan ke bahasa kita artinya sebuah sistem pelacak perubahan pada file.

¹⁴Subversion (SVN) adalah salah satu version control system popular yang ada. SVN mula-mula diciptakan sebagai pengganti CVS, software yang popular sebelumnya namun memiliki banyak kelemahan.

¹⁵<https://www.perforce.com>

¹⁶Computer vision syndrome (CVS) adalah suatu keadaan yang terjadi karena terlalu lama memfokuskan mata pada layar komputer. Umumnya penderita akan mengeluhkan nyeri kepala, pusing, penglihatan kabur, nyeri leher, mata merah, penglihatan ganda, sulit memfokuskan mata, bahkan kelelahan.

¹⁷Mercurial adalah salah satu software revision control yang dikembangkan dengan menggunakan Python, dan tersedia untuk beberapa platform seperti Linux, Windows dan MacOS. Tidak



1.2.5 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.

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. ☺

seperti software revision control lain yang sudah lama dipakai seperti CVS dan SVN, Mercurial menggunakan distributed revision control. Dengan demikian, tidak ada satu server pusat yang berisi repository source-code (koleksi kode program). Setiap orang dapat mengambil source code dari sebuah repository, kemudian membuat repository-nya sendiri.



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 QWidget 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.

1.3.1 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 disupport 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 seba-



gai User Interface (UI) aplikasi anda.

1.3.2 Teknologi User Interface QWidget

QWidget 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, QWidget sangat dinamis untuk Windows dan Mac OS. Sehingga bila anda menggunakan QWidget 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 QWidget ini menyesuaikan dengan UI OS.

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

Bila kita membuat aplikasi dengan QWidget, 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 aplikasi QtWidget dengan QMainWindow sebagai base class nya, pasti secara otomatis akan ditampilkan



menubar , toolbar, dan statusbar.

1.3.3 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 Html5¹⁸ dan CSS3¹⁹?

Lalu kenapa tidak digunakan dalam pemrograman desktop? Ya, dengan menggunakan User Interface QtWebkit²⁰ ini anda bisa membuat sebuah desktop²¹ 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,

¹⁸HTML5 adalah sebuah bahasa markah untuk menstrukturkan dan menampilkan isi dari World Wide Web, sebuah teknologi inti dari Internet. HTML5 adalah revisi kelima dari HTML dan hingga bulan Juni 2011 masih dalam pengembangan.

¹⁹CSS3 adalah Cascading Style Sheet versi ke 3, yaitu pengatur dan pengendali tampilan sebuah halaman blog/ web. CSS3 melakukan penataan terhadap komponen HTML maupun XHTML pada halaman web sehingga menghasilkan tampilan yang ramah dimata atau retina friendly.

²⁰WebKit adalah sebuah Mesin Layout yang didesain agar browser dapat merender halaman web. Webkit adalah komponen dasar dari penjelajah web Apple Safari dan Google Chrome. Pada bulan juli 2012, berdasarkan StatCounter - webkit mendapatkan lebih dari 40 % kue di pasar penjelajah web. Webkit menjadi dasar utama pada browser default di iOS, Android, Tablet Blackberry, dan sistem operasi webOS. Webkit menyediakan sekumpulan kelas untuk menampilkan isi pada jendela dan menerapkannya pada fitur penjelajah web, misalnya : mengikuti tautan ketika di-klik oleh pengguna, mengatur daftar kembali-maju, dan rekaman halaman yang baru saja dikunjungi.

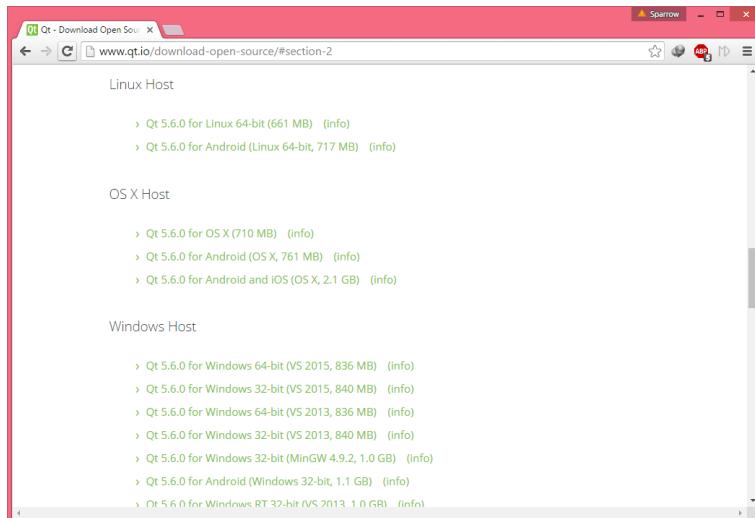
²¹Komputer meja (bahasa Inggris: desktop computer atau cukup desktop saja) adalah komputer pribadi yang ditujukan untuk penggunaan secara umum di satu lokasi yang berlawanan dengan komputer jinjing atau komputer portabel. Periferal-periferal komputer meja seperti tampilan komputer, CPU, dan papan ketik terpisah satu sama lain dan relatif berukuran besar (juga berlawanan dengan periferal pada komputer jinjing yang terintegrasi dan berukuran kecil). Komputer jenis ini dirancang untuk diletakkan dan digunakan di atas meja di rumah atau kantor. Komputer meja merupakan komputer yang paling terjangkau dan paling umum digunakan.



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 [halaman²²](https://www.qt.io/download-open-source/#section-2) Qt cCreator.



Gambar 1.4: Halaman web Downlaod Qt Creator

1.4.1 Install Qt Creator di Ubuntu 14.04

1. Download Kunjungi website Qt untuk mendownload Qt Crator sesui dengan versi sistem operasi yang digunakan baik itu 64-bit atau 32 bit. atau juga dapat

²²<https://www.qt.io/download-open-source/#section-2>



di download dengan menggunakan command line di linux dengan mengetikan.

Contoh:

```
wget http://download.qt.io/official_releases/
    online_installers/qt-unified-linux-x86-online.run
```

jika menggunakan sistem operasi beberapabasis 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. KOnfigurasi 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:

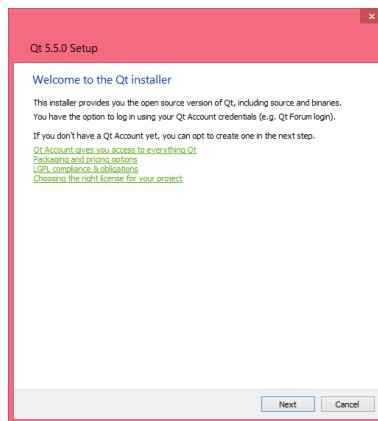
```
sudo apt-get install mesa-common-dev
```



1.4.2 Install Qt di Windows

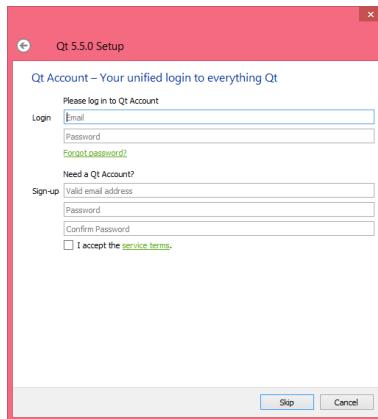
Anda dapat mendownload Qt creator di halaman websitenya seperti gambar 1.4 dan memilih versi dari Aplikasi yang Anda butuhkan baik 64 bit atau 32 bit. Sesuaikan dengan sistem operasi yang Anda miliki.

Langkah 1 Jika Anda telah mendownload Qt creator maka *qt-opensource-windows-x86-mingw492-5.5.0.exe*. Disini penulis menggunakan versi 32 bit jika sistem operasi Anda 64 bit maka gunakanlah 64 bit walaupun dapat menggunakan versi 32 bit.

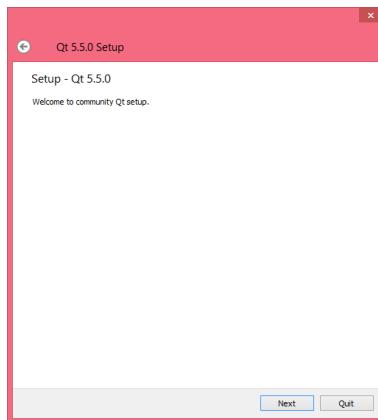


Langkah 2 Pilih **Next** dan akan muncul halaman Qt Account jika anda tidak ingin mendatarkan diri dapat di lewati dengan memilih **skip**.



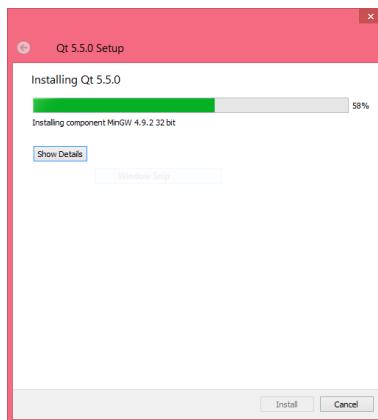


Langkah 3 Masuk ke halaman Setup terus **next** saja.



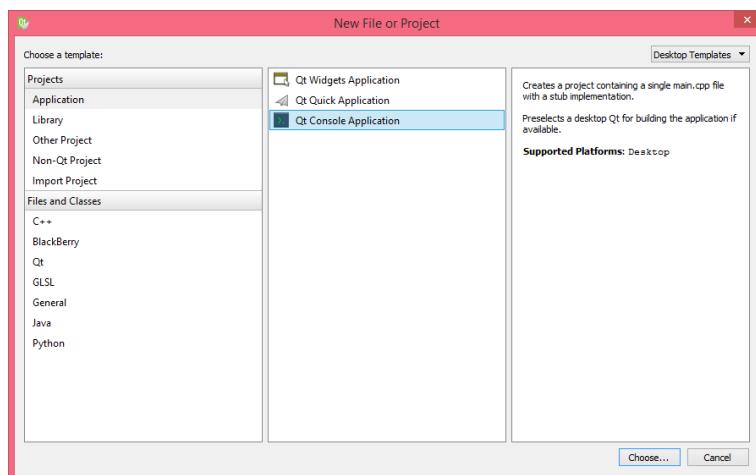
Langkah 4 Installer akan menginstall aplikasi sampai selesai apabila telah selesai maka klik finish untuk mengakhiri proses pemasangan aplikasi.



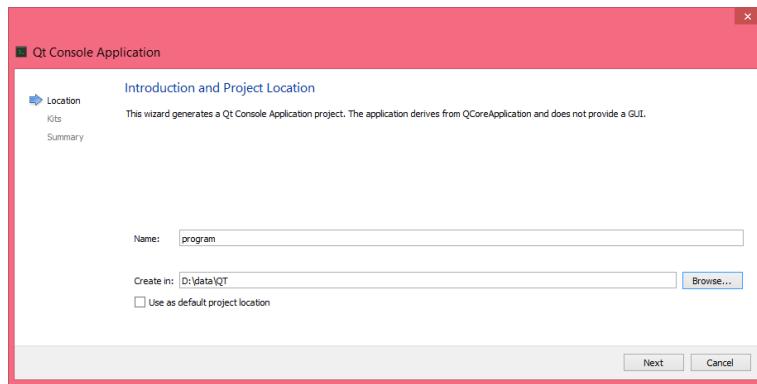


1.5 Program Console Pertama dengan Qt Creator

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



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



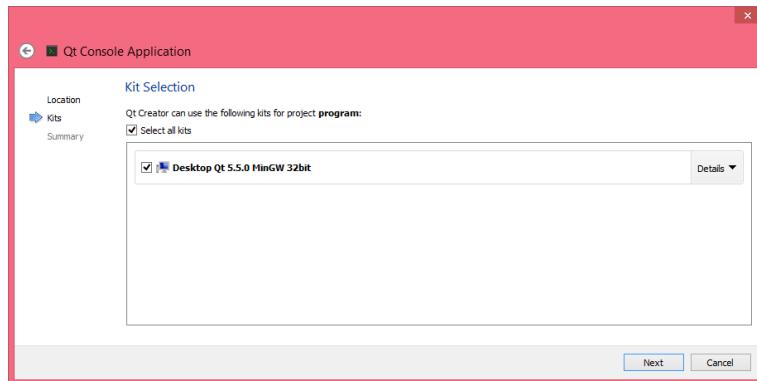
Klik Next, kemudian pilih compiler yang akan kita gunakan. Disini penulis menggunakan MinGW sebagai compilernya.



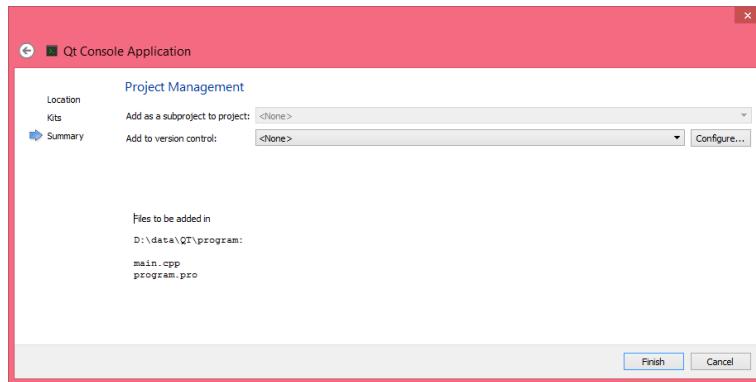
Tips

Simulator atau compiler yang lengkap terdiri dari

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

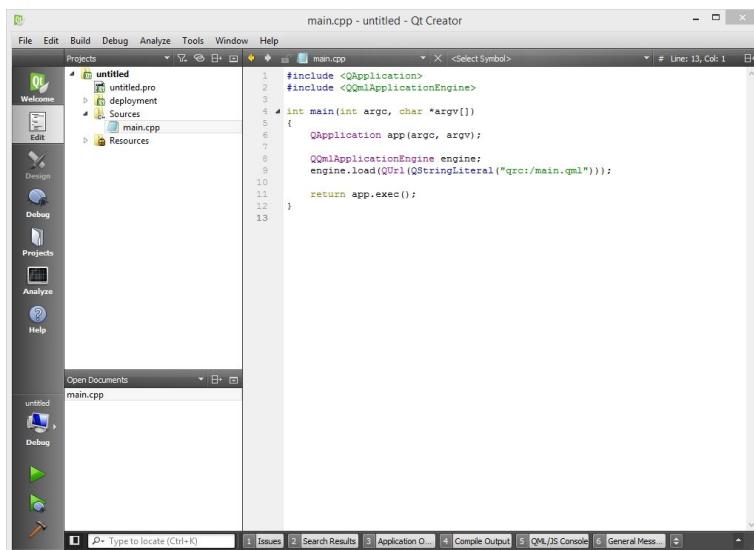


4. Kemudian pilih jenis sub version yang akan kita gunakan, jika Anda tidak menggunakan sub version maka pilih none pada add to subversion.



Gambar 1.5: Langkah 1 Setup Qt Creator

5. 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 untuk keperluan 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++:

Listing 1.1: Struktur Program C++

```
1 #include <header>
2 using namespace std;
3 int main(int argc, char *argv[])
4 {
5     deklarasi variabel;
6     deklarasi konstanta;
7     perintah perintah;
8     //komentar
9     return 0;
10 }
```

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 seorang pemrogram 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 untuk menunjukkan fungsi ini dimulai dan berakhir.



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

```
/* Komentar anda diletakkan  
di dalam ini bisa mengapit  
lebih dari satu baris */  
  
// 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 mengembalikan hasil dari fungsi utama. Kode kembalian biasanya angka 0 atau 1. Jika angka yang dikembalikan 0 berarti program berakhir dengan tidak ada error, sedangkan jika 1 maka program berakhir dengan error.

Contoh Structur program C++

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

Listing 1.2: Contoh struktur program C++

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



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

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.



BAB 2

Tipe Data, Identifier, Operator dan Control Statement

Agenda

Pada bab ini kita akan membahas beberapa topik yang berhubungan dengan tipe data, Identifier dan kontrol statement yaitu:

Contents

| | | |
|-------|------------------------------------|----|
| 2.1 | Tipe Data dan Identifier | 28 |
| 2.2 | Tipe Data Bahasa C++ | 29 |
| 2.3 | Variabel dan Konstanta | 29 |
| 2.4 | Statement | 33 |
| 2.5 | Operator dan Ekspresi | 33 |
| 2.5.1 | Operator Unary | 36 |
| 2.5.2 | Operator Unary Minus | 36 |
| 2.5.3 | Operator Unary ++ dan - | 37 |
| 2.5.4 | Operator Penggeraan | 37 |
| 2.5.5 | Operator Hubungan | 38 |
| 2.5.6 | Operator Logika | 39 |



| | |
|--|-----------|
| 2.6 Control Statement | 40 |
| 2.6.1 Percabangan | 41 |
| 2.6.2 Perulangan | 47 |

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** besarnya bisa berupa biangan desimal **42.5** (maksudnya $42\frac{1}{2}$ tahun), **golongan** seorang karyawan besarnya 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, 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 mengetahui alamat yang sesungguhnya dan dengan identifier (label) akan lebih mudah untuk diingat.



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 tabel 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 dinamakan 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:

```
int harga;
```

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

```
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 :

```
#define kapasitas 15
```

Perhatikan bahwa `kapasitas` tidak mempunyai tipe data tertentu (int, char dsb.). Preprocessor 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 :

```
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 Tipe data dan Identifier.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 2.1, kemudian tulis kode berikut.

Listing 2.1: Tipe data dan Identifier

```
1 #include <iostream>
2 int main(int argc, char *argv[])
3 {
4     using namespace std;
5     QCoreApplication a(argc, argv);
6     int panjang, lebar;
7     panjang = 15; //<-- nilai diubah menjadi 15
8     lebar = 12; //<-- nilai diubah menjadi 12
9     cout << "Panjang = " << panjang << endl;
10
11
12    cout << "Lebar = " << lebar << endl;
13    return a.exec();
14 }
```

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 :

`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 kenujian 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, Penggeraan, 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 Penggerjaan Aritmatika | = += -= *= /= %= | Kanan - Kiri | 14 |



| Kategori | Operator | Arah Proses | Jenjang |
|-----------------------------|--------------------------|-----------------|---------|
| Operator Penggeraan 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:

```
panjang = 15 ;
lebar = 12 ;
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.

2.5.1 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 |
|-----------------|---|
| <code>-</code> | Unary minus |
| <code>++</code> | Peningkatan dengan nilai penambahan 1 |
| <code>-</code> | Penurunan dengan nilai pengurangan 1 |
| <code>!</code> | Unary not |
| <code>~</code> | Operator unary komplemen satu (bitwise NOT) |

2.5.2 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`



- * C. Operator unary “-” ditulis di depan operand.

2.5.3 Operator Unary ++ dan -

Operator unary “++” dan “-” merupakan operator khusus yang ada di bahasa C. Operator “++” akan menambahkan nilai 1 ke pengenal yang menggunakan-nya 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 |
|---------------------|---------------------|
| $x = 5;$ | $x = 5;$ |
| $a = x++;$ | $a = ++x;$ |
| ----- | ----- |
| Hasil: | Hasil: |
| $x = 6$ dan $a = 5$ | $x = 6$ dan $a = 6$ |

2.5.4 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 |
| += | $a += 1$ | $a = a + 1$ |
| -= | $a -= b$ | $a = a - b$ |
| * | $a *= b$ | $a = a * b$ |



| Operator | Contoh | Maksud/ Ekuivalen dengan |
|----------|-----------|--------------------------|
| $/=$ | $a /= b$ | $a = a / b$ |
| $\%=$ | $a \%= b$ | $a = a \% b$ |

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 penggerjaan mempunyai jenjang yang lebih rendah dibanding operator aritmatika, sehingga operator aritmatika dikerjakan terlebih dahulu.

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

`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.

2.5.5 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 |

2.5.6 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).

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.

2.6.1 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:

```
if (<ekspresi_boolean>)
{
<statements>
}
```

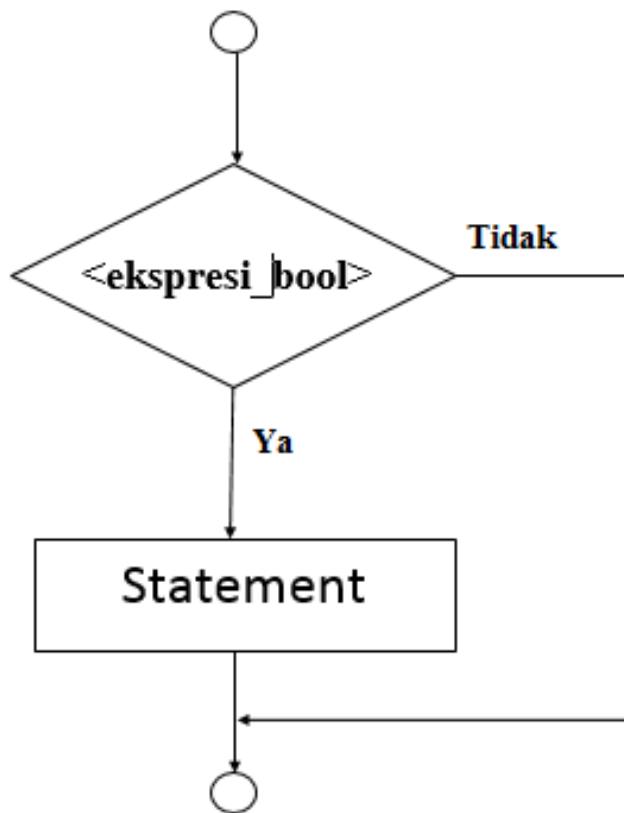
Flowchart untuk statement ini adalah :

2. Percabangan dengan if .. else

Sintaks penulisannya sebagai berikut :

```
1  if (<ekspresi_boolean>)
2  {
3      <dijalankan jika ekspresi_boolean benar>
4  }
5  else
```

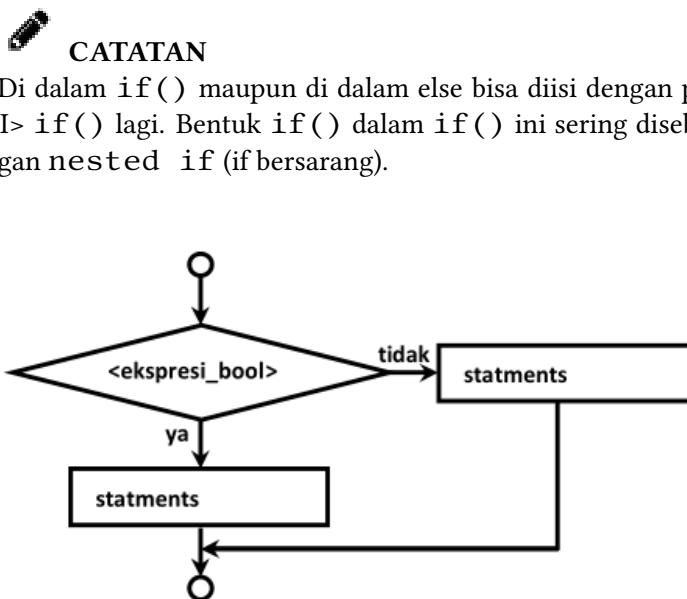




Gambar 2.1: flowchart percabangan if

```
6 {  
7 < dijalankan jika ekspresi_boolean salah>  
8 }
```

Flowchart untuk statement ini adalah :

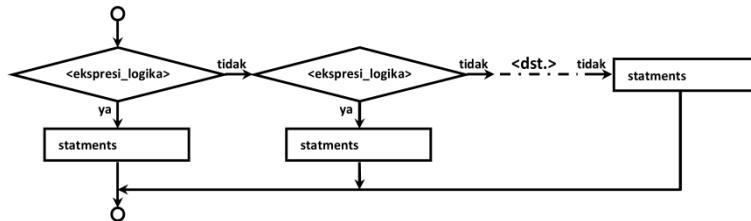


Gambar 2.2: flowchart percabangan if..else

Flowchart untuk statement if bersarang ini adalah :

3. Percabangan dengan switch

Perintah ini digunakan sebagai alternatif pengganti dari statement `if . . . else` dengan `else` lebih dari satu. Dengan perintah ini percabangan dapat diarahkan pada beberapa alternatif pilihan berdasarkan nilai ekspresi. Berbeda



Gambar 2.3: flowchart percabangan if bersarang

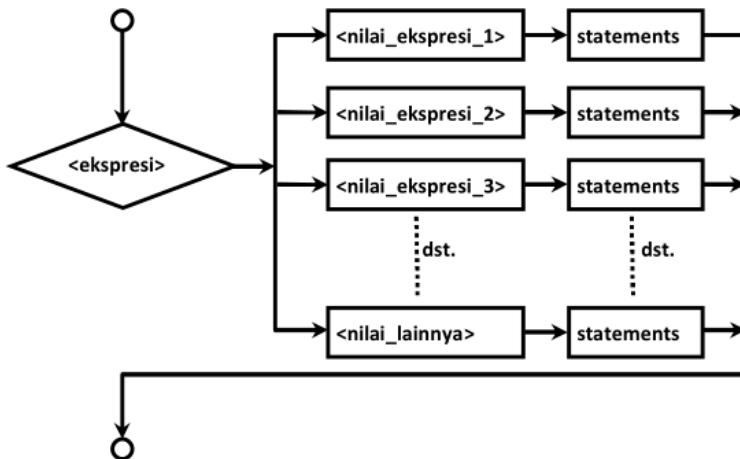
dengan **if**, **switch** tidak dapat medeteksi *operator pembanding* ($>$, $<$, dsb.), karena ekspresi dengan operator ini menghasilkan nilai *boolean*, melainkan hanya dapat mengalihkan alur program ke suatu nilai yang sama, pada statement ini ekspresi yang diminta harus menghasilkan bilangan *bulat*.

```

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.





Gambar 2.4: Percabangan dengan switch

Flowchart untuk statement ini adalah :

Contoh Tipe data dan Identifier.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 2.2, kemudian tulis kode berikut.

Listing 2.2: Tipe data dan Identifier

```

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     int hari = 6;
8     switch(hari){
  
```



```
9 case 1 : cout << "Senin" << endl;
10 break;
11 case 2 : cout << "Selasa" << endl;
12 break;
13 case 3 : cout << "Rabu" << endl;
14 break;
15 case 4 : cout << "Kamis" << endl;
16 break;
17 case 5 : cout << "Jumat" << endl;
18 break;
19 case 6 : cout << "Sabtu" << endl;
20 break;
21 case 7 : cout << "Minggu" << endl;
22 break;
23 default: cout << "Tidak ada..." << endl;
24 }
25 return a.exec();
26 }
```

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**”.



2.6.2 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 :

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

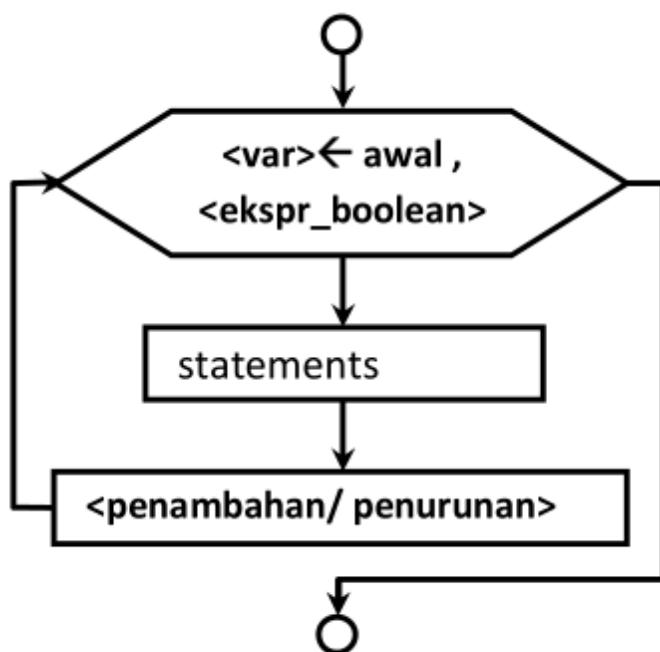
2. Perulangan dengan while

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

Sintaks penulisannya sebagai berikut :

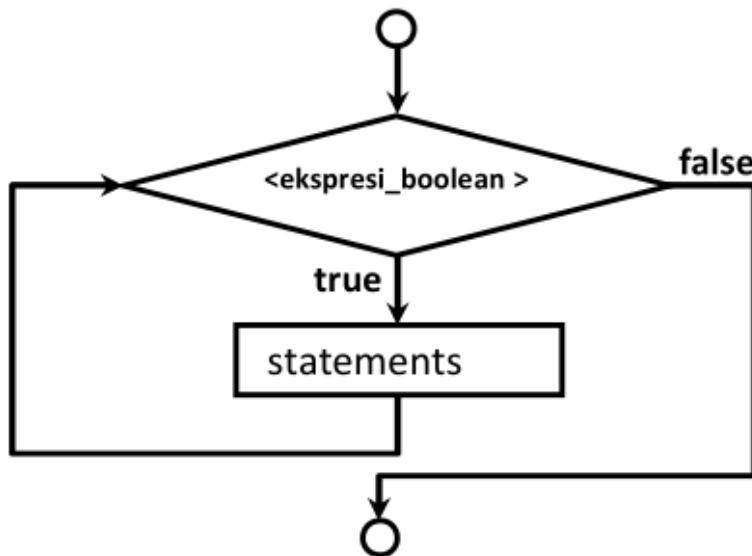
```
for(<nilai_awal>; <ekspresi_boolean>; <penambahan/
    penurunan>)
{
<statmemnts>
```





Gambar 2.5: flowchart perulangan dengan for

{



Gambar 2.6: flowchart perulangan dengan while

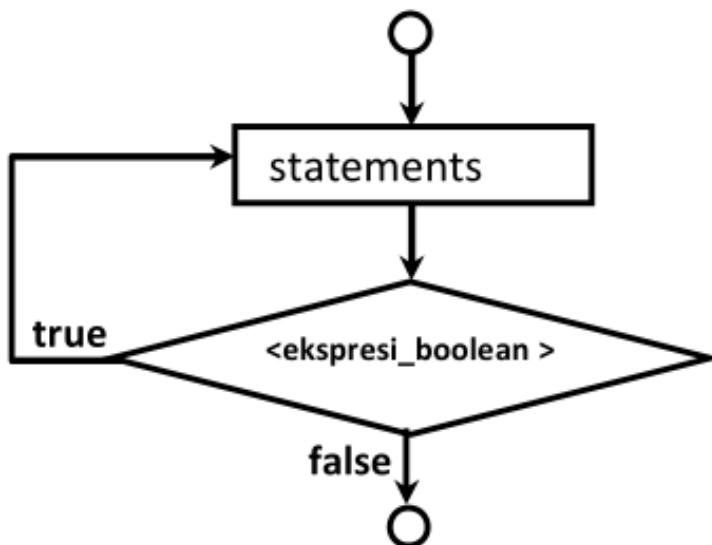
3. Perulangan dengan do ... while

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(<ekspresi_boolean>)
```





Gambar 2.7: flowchart perulangan dengan do..

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 program sebelum ekspresi *boolean* yang ada pada statement tersebut menghentikan, sedangkan kata kunci `continue` digunakan 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>
}
```



BAB 3

Array dan String

Agenda

Pada bab ini kita akan mempelajari mengenai Array dan String seperti berikut ini

Contents

| | | |
|------------|---|-----------|
| 3.1 | Array | 54 |
| 3.1.1 | Array 1 Dimensi | 55 |
| 3.1.2 | Inisialisasi Array Satu Dimensi | 61 |
| 3.1.3 | Pengalamatan dan Pengkopian Array 1 Dimensi | 67 |
| 3.1.4 | Array Multi Dimensi | 70 |
| 3.1.5 | Cara Pengaksesan Array dapat dilakukan dengan 2 cara: | 81 |
| 3.2 | String | 83 |
| 3.2.1 | Array of Character | 83 |
| 3.3 | Fungsi-fungsi String | 88 |
| 3.3.1 | strcpy() dan strncpy() | 90 |

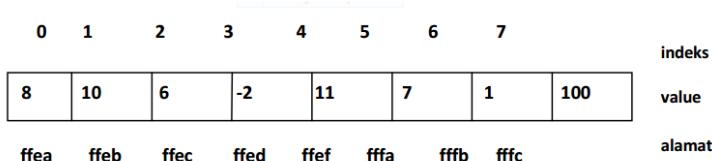


| | | |
|-------|---|----|
| 3.3.2 | strcat() | 93 |
| 3.3.3 | Fungsi mengubah string menjadi numerik dan sebaliknya | 96 |
| 3.3.4 | Class string pada C++ | 96 |

3.1 Array

Array adalah suatu tipe data terstruktur yang berupa sejumlah data sejenis (ber-tipe data sama) yang jumlahnya tetap dan diberi suatu nama tertentu. Elemen-elemen array tersusun secara sekuelensial 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 seperti gambar 3.1.



Gambar 3.1: Ilustrasi array satu dimensi pada memory komputer

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.

3.1.1 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.

Deklarasi Array satu Dimensi

Bentuk umum deklarasi array satu dimensi:

```
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:

```
char huruf[9];
int umur[10];
int kondisi[2] = {0,1};
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 $4 \times 10 = 20$ 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 Input dan Output Array

Buatlah project baru dan tulis kode berikut:

Listing 3.1: Input dan Output Array

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



```
2 #include <iostream>
3 using namespace std;
4 int main(int argc, char *argv[])
5 {
6     QApplication a(argc, argv);
7     int nilai[5], x;
8     cout<<"Masukkan nilai"<<endl;
9     for(x=0;x<5;x++)
10    {
11        cout<<"Nilai Angka ke - "<<x+1<< ":" ;
12        cin>>nilai[x];
13    }
14    cout<<endl;
15    cout<<"Membaca nilai :\n";
16    for(x=0;x<5;x++)
17    {
18        cout<<"Nilai Angka : "<<nilai[x]<<endl;
19    }
20    return a.exec();
21 }
```

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 Manipulasi Array

Buatlah project baru dan tulis kode berikut:

Listing 3.2: Manipulasi Array

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 int main(int argc, char *argv[])
5 {
6     QCoreApplication a(argc, argv);
7     int bil[7], i;
8     cout<<"elemen ke-1 ? "; cin>>bil[0];
9     bil[1] = 5;
10    bil[2] = bil[1] + 20;
11    for(i=4;i<7;i++)
12        bil[i] = i*10;
13    bil[3] = bil[bil[1]];
14    for(i=0;i<7;i++)
15        cout<<"bil["<<i<<"] = "<<bil[i]<<" dan alamatnya:
16        "<<&bil[i]<<"\n";
17    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 Penanganan Batas Indeks Elemen Array

Buatlah program beikut ini:

Listing 3.3: Penanganan Batas Indeks Elemen Array

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



```
3 using namespace std;
4 int main(int argc, char *argv[])
5 {
6     QCoreApplication a(argc, argv);
7     int angka[5];
8     angka[0] = 0; //batas bawah array
9     angka[4] = 4; //batas atas array
10    angka[5] = 5; //indeks melebihi batas
11    //program akan HANG
12    cout<<angka[5];
13    return a.exec();
14 }
```

Hasil dan Keterangan:

- Progarm akan HANG-UP. Hal ini terjadi karena compiler tidak bertanggungjawab 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!

3.1.2 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 diinisialisasi 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 Inisialisasi Array dengan nilai \0

Buatlah program berikut:

Listing 3.4: Inisialisasi Array dengan nilai \0

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 int main(int argc, char *argv[])
5 {
6     QCoreApplication a(argc, argv);
7     int bil[7] = {0}; //inisialisasi 0
8     for(int i=0;i<7;i++){
9         cout<<"Elemen ke-"<<i<<": "<<bil[i]<<"\n";
10    }
11    return a.exec();
```



12 }

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 Inisialisasi Array dua nilai elemen pertama

Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 3.1, kemudian tulis kode berikut.

Listing 3.5: Inisialisasi Array dua nilai elemen pertama

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 int main(int argc, char *argv[])
5 {
6     QCoreApplication a(argc, argv);
7     int bil[7] = {2,5}; //inisialisasi dua elemen
8     pertama
9     for(int i=0;i<7;i++){
    cout<<"Elemen ke- "<<i<<": "<<bil[i]<<"\n";
```



```
10     }
11     return a.exec();
12 }
```

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 diinisialisasi 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:

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

Maka hasilnya adalah:

```
angka[0] = 1,
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 inisialisasi terlebih dahulu.

Contoh:

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

Contoh Tanpa inisialisasi, array langsung ditampilkan

Tulislah program berikut ini:

Listing 3.6: Tanpa inisialisasi array langsung ditampilkan

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 int main(int argc, char *argv[])
5 {
6     QCoreApplication a(argc, argv);
7     char h[5];
8     for(int i=0;i<5;i++){
9         cout<<"Elemen ke "<<i<<" = "<<h[i]<<endl;
10    }
11    return a.exec();
12 }
```

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 diinisialisasi, 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 Penggunaan tipe data enum pada Array satu dimensi

Buatlah program berikut:

Listing 3.7: Penggunaan tipe data enum pada Array satu dimensi

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

3.1.3 Pengalaman dan Pengkopian Array 1 Dimensi

Array tidak bisa disalin begitu saja antara array satu yang ada nilainya ke array lain yang kosong. Hal ini dikenakan 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:

```
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 Percobaan Penyalinan Array 1 dimensi

Buatlah program berikut:

Listing 3.8: Percobaan Penyalinan Array 1 dimensi

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 int main(int argc, char *argv[])
5 {
6     QCoreApplication a(argc, argv);
7     int A[6]={1,2,3,4,5,6};
8     int B[6];
9     B = A;
10    return a.exec();
11 }
```

Hasil:



Keterangan:

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

Contoh Penyalinan Array 1 dimensi dengan Perulangan

Buatlah program berikut ini:

Listing 3.9: Penyalinan Array 1 dimensi dengan Perulangan

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 int main(int argc, char *argv[])
5 {
6     QCoreApplication a(argc, argv);
7     int A[6]={1,2,3,4,5,6};
8     int B[6];
9     for(int i=0;i<6;i++){
10         B[i]=A[i];
11     }
12     for(int j=0;j<6;j++){
13         cout<<B[j]<<endl;
14     }
15     return a.exec();
16 }
```

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 dikopian masih tetap memiliki array yang asli. Untuk menghapusnya, maka harus dilakukan secara manual.

3.1.4 Array Multi Dimensi

Array multi dimensi berarti array yang kita deklasaiakan dapat dikembangkan ke array dimensi 2 dan seterusnya. 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:

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

yang digambarkan dalam bentuk gambar 3.1.4:

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.

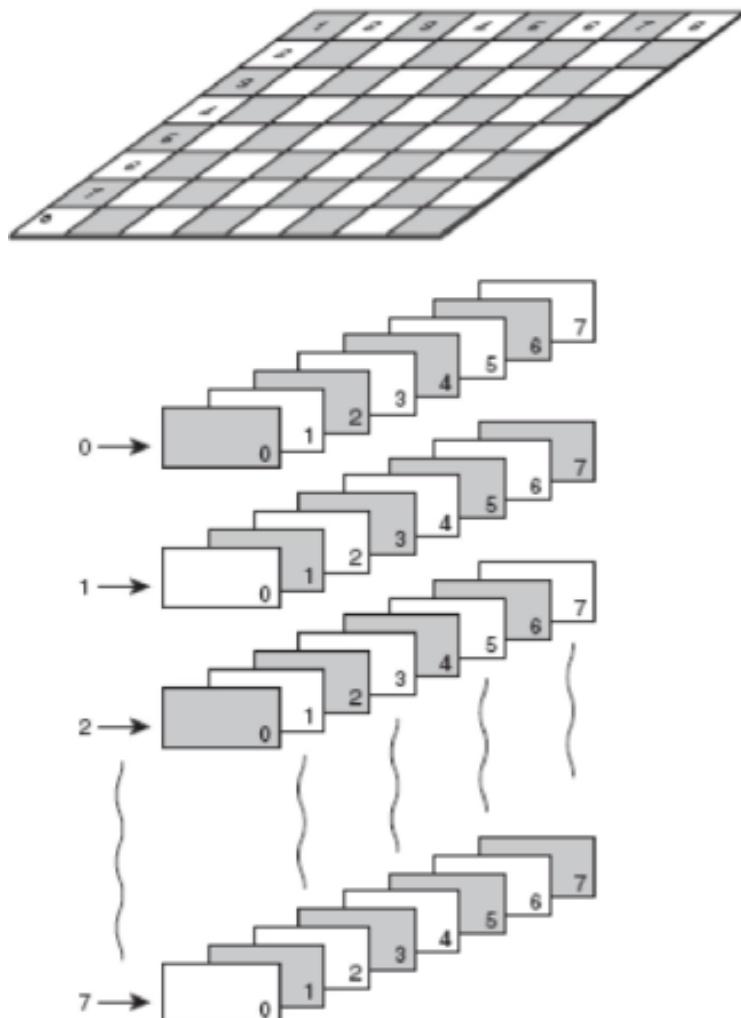
Berikut adalah gambar array berdimensi (baris x kolom = 3 x 4)

Deklarasi Array Dua Dimensi

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

Contoh:





Gambar 3.2: Contoh deklarasi array dua dimensi yang menggambarkan papan catur

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

Gambar 3.3: array dimensi 3 x 4

```
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:

$$x = \begin{bmatrix} 8 & 5 & 9 & 6 \\ 8 & 2 & 1 & 0 \end{bmatrix}$$

selanjutnya dapat dideklarasikan sebagai berikut:

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

atau diklarasikan dengan langsung menginisialisasi nilai elemen-elemennya 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 matiks[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 matiks[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 matiks[3][5] = {{'A','B','C','D','E'},  
'F','G','H','I','J'},  
'K','L','M','N','O'}};  
char matiks[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 | a | y | a | \0 | | | |
| S | e | m | a | r | a | n | g | \0 | | | |
| Y | o | g | y | a | k | a | r | t | a | \0 | |

Jika jumlah nilai character lebih banyak daripada deklarasi

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



Akan terjadi ERROR!



Jika data array character yang diinputkan kurang dari deklarasi

```
char matriks[3][5] = {{'a', 'b', 'c', 'd', 'e'},
{'f', 'g', 'h', 'i', 'j'},
{'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

```
int matriks[3][5] = {{5,12,17,10,7},
{15,6,25,2,19},
{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[][] = {{5,12,17,10,7},
{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[][] = {5,12,17,10,7,
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 Deklarasi dan Menampilkan Array 2 Dimensi

Buatlah program berikut:

Listing 3.10: Deklarasi dan Menampilkan Array 2 Dimensi

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 int main(int argc, char *argv[])
5 {
6     QCoreApplication a(argc, argv);
7     int matriks[3][5] = {{5,12,17,10,7},
8     {15,6,25,2,19},
9     {4,9,1,5,2}};
10    for(int i=0;i<3;i++){
11        for(int j=0;j<5;j++){
12            cout<<matriks[i][j]<<"\t";
13        }
14        cout<<endl;
15    }
16    return a.exec();
17 }
```

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 ''.

Contoh 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:

Listing 3.11: Penyalinan Array 2 Dimensi ke Array 2 Dimensi lainnya

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 int main(int argc, char *argv[])
5 {
6     QApplication a(argc, argv);
```



```
7 int matriks
8 [3][5]={ {1,2,3,4,5}, {6,7,8,9,10}, {11,12,13,14,15} };
9
10 int matrikhasil[3][5];
11 for(int i=0;i<3;i++){
12     for(int j=0;j<5;j++){
13         matrikhasil[i][j] = matriks[i][j];
14     }
15 }
16 for(int i=0;i<3;i++){
17     for(int j=0;j<5;j++){
18         cout<<matrikhasil[i][j]<<"\t";
19     }
20     cout<<endl;
21 }
22 return a.exec();
23 }
```

Hasil:

| | | | | |
|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 |
| 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 16 |

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 matrikhasil tepat pada baris dan kolom yang sesuai.



Contoh Penyalinan array 2 dimensi ke dalam array 1 dimensi.

Buatlah program berikut ini:

Listing 3.12: Penyalinan array 2 dimensi ke dalam array 1 dimensi

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 int main(int argc, char *argv[])
5 {
6     QCoreApplication a(argc, argv);
7     int matriks
8         [3][5]={ {1,2,3,4,5}, {6,7,8,9,10}, {11,12,13,14,15}};

9     int matrikshasil[15];
10    int counter=-1;
11    for(int i=0;i<3;i++){
12        for(int j=0;j<5;j++){
13            counter++;
14            matrikshasil[counter] = matriks[i][j];
15        }
16    }
17    for(int i=0;i<15;i++){
18        cout<<matrikshasil[i]<<endl;
19    }
20    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 3×5 , 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.



3.1.5 Cara Pengaksesan Array dapat dilakukan dengan 2 cara:

Pengaksesan Baris demi Baris

Cara ini menelusuri elemen array dua dimensi per dimulai dari baris pertama lalu kekanan sesuai dengan jumlah kolomnya. Setelah elemen 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:

Listing 3.13: Pengaksesan Baris demi Baris

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 int main(int argc, char *argv[])
5 {
6     QCoreApplication a(argc, argv);
7     int A[2][3]={{1,2,3},{4,5,6}};
8     for(int baris=0;baris<2;baris++){
9         for(int kolom=0;kolom<3;kolom++){
10             cout<<A[baris][kolom]<<"\t";
11         }
12         cout<<endl;
13     }
14     return a.exec();
15 }
```

Hasil:

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |



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:

Listing 3.14: Pengaksesan Kolom demi Kolom

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 int main(int argc, char *argv[])
5 {
6     QCoreApplication a(argc, argv);
7     int A[2][3]={{1,2,3},{4,5,6}};
8     for(int kolom=0;kolom<3;kolom++){
9         for(int baris=0;baris<2;baris++){
10             cout<<A[baris][kolom]<<"\t";
11         }
12         cout<<endl;
13     }
14     return a.exec();
15 }
```

Hasil:

| | |
|---|---|
| 1 | 4 |
| 2 | 5 |
| 3 | 6 |



3.2 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

| | | | | |
|---|---|---|---|----|
| A | N | T | 0 | \0 |
|---|---|---|---|----|

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++

3.2.1 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

```
char nama[6]; //tanpa inisialisasi
char nama2[6] = "anton"; //langsung diinisialisasi
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 penggunaannya.

Contoh Penggunaan karakter \0

Buatlah program berikut:

Listing 3.15: Penggunaan karakter \0

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 int main(int argc, char *argv[])
5 {
6     QCoreApplication a(argc, argv);
7     char string[100] = "String";
8     int K;
9     for(K=0; string[K] != '\0'; K++)
10         cout << string[K];
11     return a.exec();
12 }
```

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 String tanpa karakter \0

Buatlah program berikut:

Listing 3.16: String tanpa karakter \0

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 int main(int argc, char *argv[])
5 {
6     QCoreApplication a(argc, argv);
7     char string[6] = "String";
8     int K;
9     for(K=0; string[K] != '\0'; K++)
10         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 Mengisi Array of Character

Buatlah program berikut:



Listing 3.17: Mengisi Array of Character

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

Hasil:

```
Isi data string: Wachid
hasil data string: Wachid
```

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: Nur Wachid
hasil data string: Wachid
```



Contoh Pengisian variabel array of character dengan maksimum jumlah karakter.

Tulislah program berikut:

Listing 3.18: Pengisian variabel array of character dengan maksimum jumlah karakter

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 int main(int argc, char *argv[])
5 {
6     QCoreApplication a(argc, argv);
7     char buffer[50] = {'\0'};
8     cout << "Isi data string: ";
9     cin.get(buffer,49); //ambil sebanyak 50 karakter
10    atau diakhiri tanda enter
11    cout << "Hasil data string: " << buffer << endl;
12    return a.exec();
13 }
```

Hasil:

```
Isi data string: Nur Wachid
hasil data string: Nur Wachid
```

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.



3.3 Fungsi-fungsi String

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

strlen()

Berfungsi untuk menentukan panjang suatu nilai string.

Bentuk umum: `int strlen(<identifier string>);`

length()

Berfungsi untuk menentukan panjang suatu nilai tipe data class string

Bentuk umum method: `<nama_var_string>.length();`

Contoh Penggunaan fungsi strlen()

Buatlah program berikut ini:

Listing 3.19: Penggunaan fungsi strlen()

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 int main(int argc, char *argv[])
5 {
6     QCoreApplication a(argc, argv);
7     char data[100];
8     cout<<"Masukkan kalimat apapun yang anda sukai (max
9         100 huruf): ";
10    cin.get(data, 99);
11    cout<<"panjang huruf adalah: "<<strlen(data)<<""
12        karakter";
```



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

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 Penggunaan fungsi length pada tipe data string C++

Buatlah program berikut ini:

Listing 3.20: Penggunaan fungsi length pada tipe data string C++

```
1 #include <QtCore/QCoreApplication>  
2 #include <iostream>  
3 #include <string>  
4 using namespace std;  
5 int main(int argc, char *argv[]){  
6     QCoreApplication a(argc, argv);  
7     string data2;  
8     cout<<"Masukkan kalimat apapun yang anda sukai (max  
9         100 huruf): ";  
10    getline(cin,data2);  
11    cout<<"panjang huruf adalah: "<<data2.length()<<"  
12         karakter";  
13    return a.exec();  
14 }
```



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.

3.3.1 `strcpy()` dan `strncpy()`

Dalam bahasa C++, untuk menyalin nilai suatu string tidak dapat langsung menuliskannya seperti halnya kompiler lain, sehingga proses menyalin atau mengerjakan suatu nilai string ke variabel string yang lain diperlukan suatu fungsi pustaka yang bernama `strcpy()`.

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

Contoh Penggunaan fungsi `strcpy()`

Buatlah program berikut ini:

Listing 3.21: Penggunaan fungsi `strcpy()`

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



```
2 #include <iostream>
3 #include <string>
4 using namespace std;
5 int main(int argc, char *argv[])
6 {
7     QApplication a(argc, argv);
8     char data[100] = {'\0'};
9     char data2[]="STRING";
10    strcpy(data,data2);
11    cout<<"string pertama: "<<data<<"\n";
12    cout<<"string kedua : "<<data2;
13    return a.exec();
14 }
```

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, `strcpy ()` 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 dikopian ke data, maka akan terjadi error karena tempatnya kurang.

Contoh Penggunaan fungsi strncpy()

Buatlah program berikut:

Listing 3.22: Penggunaan fungsi strncpy()

```
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     char data[6] = {'\0'};
9     char data2[]="STRINGKU";
10    strncpy(data,data2,5);
11    cout<<"string pertama: "<<data<<"\n";
12    cout<<"string kedua : "<<data2;
13    return a.exec();
14 }
```

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 `strn-`



cpy 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.

3.3.2 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 Penggunaan fungsi strcat()

Buatlah program berikut:

Listing 3.23: Penggunaan fungsi strcat()

```
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     char str[6] = "anton";
9     char str2[7];
10    char str3[14];
11    str3 = str + str2;
12    return a.exec();
13 }
```

Hasil:



Belajar C++ dengan Qt Creator

**Keterangan:**

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

Contoh Penggunaan fungsi `strcat()`

Buatlah program berikut:

Listing 3.24: Penggunaan fungsi `strcat()`

```
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     char string1[100] = "Kami kelompok ";
9     char string2[] = " belajar Qt C++";
10    strcat(string1, string2);
11    cout << "Jadi gabungannya adalah: " << string1;
12    return a.exec();
13 }
```

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(string sumber, 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`

3.3.3 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>`

3.3.4 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 Pembuatan variabel string C++, penyalinan string, dan penggabungan string

Buatlah program berikut:

Listing 3.25: Pembuatan variabel string C++ penyalinan string dan penggabungan string

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 #include <string>
4 using namespace std;
5 int main(int argc, char *argv[])
6 {
7     QApplication a(argc, argv);
8     string str1("Ini string C++");
9     cout<<"Isi str1 = "<<str1<<endl;
10    //salin isi str1 ke str2
11    string str2;
12    str2 = str1;
13    cout<<"Isi str2 = "<<str2<<endl;
14    //ubah str2
15    str2 = "Hallo, ";
16    //buat strhasil dan isi dgn gabungan dari str1 dan
17    //str2
18    string strhasil;
19    strhasil = str2 + str1;
20    cout<<strhasil;
21    return a.exec();
22 }
```

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 menginclude `#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 Penggunaan class string untuk manipulasi data

Buatlah program berikut:



Listing 3.26: Penggunaan class string untuk manipulasi data

```
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     //buat var nama dgn C-style string
9     char nama[10] = "Haloooooooo";
10    //buat var string dan diisi nilai dari var nama
11    string nama_copy(nama);
12    cout<<nama_copy<<endl;
13    //buat var nama2 dan kopikan isinya ke nama2_copy
14    //melalui konstruktor
15    string nama2 = "saya belajar";
16    string nama2_copy(nama2);
17    cout<<nama2_copy<<endl;
18    //buat var nama35 yg diisi nilai dari nama tapi
19    //hanya 5 huruf saja
20    string nama35(nama,5);
21    cout<<nama35<<endl;
22    //buat var ulang yg diisi huruf 'C' sebanyak 10 buah
23    string ulang(10, 'a');
24    cout<<ulang;
25    return a.exec();
26 }
```

Hasil:

```
Haloooooooo
saya belajar
C
```

Keterangan:

Dapat dilihat langsung pada baris komentar program diatas.

Contoh Penggabungan string dengan menggunakan class string

Buatlah program berikut:

Listing 3.27: Penggabungan string dengan menggunakan class string

```
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 3 |
| Percobaan 1 | Percobaan 2 | |

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 digabungan 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 Pengaksesan isi nilai class string

Buatlah program berikut:

Listing 3.28: Pengaksesan isi nilai class string

```
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("Indonesia Raya");
9     for(size_t i=0;i<satu.length();i++){
10         cout<<satu[i]<<endl;
11     }
12     cout<<endl;
13     cout<<"C-sytyle: "<<satu.c_str();
14     return a.exec();
15 }
```

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 Menemukan substring pada sebuah string besar

Tulislah program berikut ini:

Listing 3.29: Menemukan substring pada sebuah string besar

```
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 strSample ("Kata pak Hari, \"hari ini
9         matahari cerah sekali!\"");
10    cout << "Contoh string adalah: " << endl;
11    cout << strSample << endl << endl;
12    // Temukan kata "hari"
13    size_t nOffset = strSample.find ("hari", 0);
14    // Cek apakah ketemu?
15    if (nOffset != string::npos)
16        cout << "Ketemu pertama kata \"hari\" pada offset
17            " << nOffset;
18    else
19        cout << "Substring tidak ditemukan" << endl;
20        cout << endl << endl;
21        cout << "Mencari semua kata substring \"hari\"" <<
22            endl;
23        size_t nSubstringOffset = strSample.find ("hari",
24            0);
25        while (nSubstringOffset != string::npos)
26        {
27            cout << "Kata \"hari\" ada di offset " <<
28            nSubstringOffset << endl;
```



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

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 Membalik kata / kalimat.

Tulislah program berikut ini:

Listing 3.30: Membalik kata / kalimat

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 #include <string>
4 #include <algorithm>
5 using namespace std;
6 int main(int argc, char *argv[])
7 {
8     QCoreApplication a(argc, argv);
9     string strSample ("String ini akan dibalik!");
10    cout << "String asli: " << endl;
11    cout << strSample << endl << endl;
12    reverse (strSample.begin (), strSample.end ());
13    cout << "Setelah dibalik: " << endl;
14    cout << strSample;
15    return a.exec();
16 }
```

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 Konversi huruf besar dan kecil.

Tulislah program berikut ini:

Listing 3.31: Konversi huruf besar dan kecil

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 #include <string>
4 #include <algorithm>
5 using namespace std;
6 int main(int argc, char *argv[])
7 {
8     QCoreApplication a(argc, argv);
9     cout << "Masukkan sebuah string: " << endl;
10    string strInput;
11    getline (cin, strInput);
12    transform(strInput.begin(),strInput.end(),strInput.
13              begin(),(int(*)(int))toupper);
14    cout << "Hasil konversi ke huruf besar: " << endl;
15    cout << strInput << endl << endl;
16    transform
17    (strInput.begin(),strInput.end(),strInput.begin(),(
18        int(*)(int))tolower);
19    cout << "Hasil konversi ke huruf kecil: " << endl;
20    cout << strInput << endl << endl;
```



```
19     return a.exec();  
20 }
```

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.



BAB 4

Fungsi

Agenda

Pada Bab ini kita akan mempelajari tentang funtion bahasa C++ seperti berikut ini

Contents

| | |
|---|------------|
| 4.1 Konsep Dasar Fungsi | 111 |
| 4.2 Mendefinisikan Fungsi | 112 |
| 4.3 Deklarasi Fungsi (Prototype) | 113 |
| 4.4 Hasil Balik Fungsi | 116 |
| 4.5 Ruang Lingkup Variabel | 118 |
| 4.5.1 Variable Lokal | 118 |
| 4.5.2 Variable Global | 120 |
| 4.5.3 Variabel statik | 122 |
| 4.6 Pengiriman Parameter | 124 |
| 4.7 Parameter Default | 125 |



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.

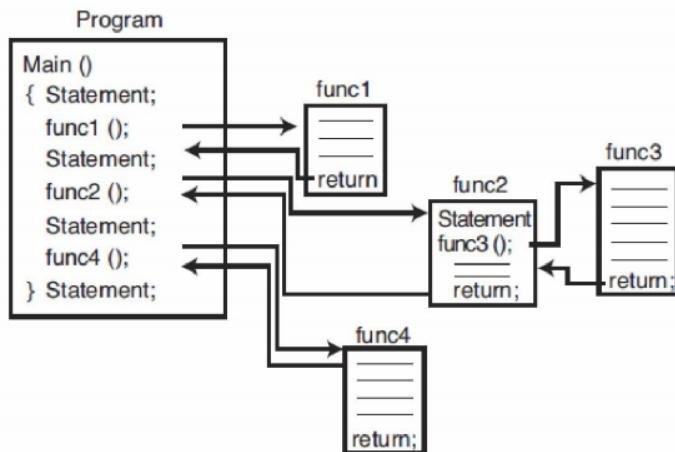
Fungsi dapat melakukan manipulasi terhadap data dan dapat mengembalikan suatu nilai. Semua program yang ditulis dengan bahasa C++ paling tidak mempunyai sebuah fungsi, yaitu `main()`, fungsi ini akan dipanggil secara otomatis ketika program dieksekusi, sedangkan fungsi yang lain baru akan bekerja ketika 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 program utama melanjutkan ke baris program berikutnya. Peristiwa ini dinamakan pemanggilan fungsi, berikut ini adalah ilustrasi mengenai pemanggilan fungsi seperti gambar 4.

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.





Gambar 4.1: Ilustrasi penanganan fungsi

4.1 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 istilah 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 menunjukkan 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.

4.2 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 parameter-parameter yang digunakan. Tubuh dari fungsi berisikan statement-statement 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 statement `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 perkerjaan 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;  
}
```

4.3 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 bertidat 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 Membuat Fungsi yang mengembalikan nilai.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 4.1, kemudian tulis kode berikut.

Listing 4.1: Membuat Fungsi yang mengembalikan nilai

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 int absolut(int bil);
4 int main(int argc, char *argv[])
5 {
6     using namespace std;
7     QCoreApplication a(argc, argv);
8     int bilangan = -10;
9     cout << "Bilangan : " << bilangan << endl;
10    cout << "Dimutlakkan menjadi : " << absolut(
11        bilangan) << endl;
12    return a.exec();
13 }
14 int absolut(int bil){
15     if(bil<0)
16         return - bil;
17     else
18         return bil;
19 }
```

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);
```

4.4 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 Membuat Fungsi yang tidak mengembalikan nilai.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama contoh 4.2, kemudian tulis kode berikut.

Listing 4.2: Membuat Fungsi yang tidak mengembalikan nilai

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 void hello(int kali);
4 int main(int argc, char *argv[])
5 {
6     QCoreApplication a(argc, argv);
7     hello(3);
8     return a.exec();
9 }
10 void hello(int kali){
11     using namespace std;
12     for(int x=0;x<kali;x++)
13         cout << "Hello World!" << endl;
14 }
```

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 4.1 di atas.

4.5 Ruang Lingkup Variabel

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

4.5.1 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 Variabel Lokal.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 4.3, kemudian tulis kode berikut.

Listing 4.3: Variabel Lokal

```
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.

4.5.2 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 menggunakannya.

Contoh Variabel Global.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 4.4, kemudian tulis kode berikut.

Listing 4.4: Variabel Global

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 void kali(float a, float b); /*prototype fungsi*/
4 float hasil; /*variabel global*/
```



```
5 int main(int argc, char *argv[])
6 {
7     using namespace std;
8     QCoreApplication a(argc, argv);
9     kali(4,7);
10    cout << "Variabel global hasil = " << hasil <<
11        endl;
12    return a.exec();
13 }
14 void kali(float a, float b)
15 {
16     hasil = a * b;
17 }
```

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 variabel yang diluar blok, menentukan cakupan sifat global variabel tersebut.



4.5.3 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 Variabel Statik.

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

Listing 4.5: Variabel Statik

```
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     QApplication 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 ?? 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 menyebabkan 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. Pendekripsi kesalahan program lebih sulit dilakukan.

4.6 Pengiriman Parameter

Seperti contoh program-program di atas, fungsi dapat menerima nilai melalui parameter formal dan dapat mengembalikan nilai melalui statement `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:

```
int fungsiku();
```

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

```
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, parameter list tersebut adalah : `nilaiInt` yaitu sebuah variabel bertipe `int` dan `nilaiFloat` yaitu sebuah variabel bertipe `float`.

Ketika kita mengirimkan 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 :

```
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 dikirimkan tersebut.

Hasil dari suatu fungsi dapat diperoleh dari nilai kembalinya (return) atau dengan variabel global. Seperti contoh pada Contoh 4.4, 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.

4.7 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 :

```
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 contoh 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 seperi 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 Default Parameter.

Buka Qt Creator dan buat project Qt Console Application baru dengan nama contoh 4.6, kemudian tulis kode berikut.

Listing 4.6: Default Parameter

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 int volume(int,int=1,int=1); /*prototype*/
4 int main(int argc, char *argv[])
5 {
6     using namespace std;
7     QCoreApplication a(argc, argv);
8     int panjang,lebar,tinggi;
9     panjang = 10;
10    lebar = 15;
11    tinggi = 25;
12    /*menghitung volume*/
13    cout << "Volume 1 --> " << volume(panjang,lebar,
14        tinggi)<< endl;
15    cout << "Volume 2 --> " << volume(panjang,lebar)<<
16        endl;
17    cout << "Volume 3 --> " << volume(panjang)<< endl;
18    return a.exec();
19 }
20 /*---Fungsi volume---*/
21 int volume(int p, int l, int t)
22 {
23     return p * l * t;
24 }
```

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 4.6, Volume 1 dihasilkan dari $10 \times 15 \times 25$ karena semua parameter formal menerima nilai, maka hasilnya 3750.
- Dari contoh 4.6, Volume 1 dihasilkan dari $10 \times 15 \times 1$ karena parameter formal ketiga tidak menerima nilai, maka hasilnya 150.
- Dari contoh 4.6, Volume 1 dihasilkan dari $10 \times 1 \times 1$ karena parameter formal kedua dan ketiga tidak menerima nilai, maka hasilnya 10.



BAB 5

Pointer dan References

Agenda

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

Contents

| | | |
|-------|--|-----|
| 5.1 | Apa itu Pointer? | 130 |
| 5.2 | Memory Komputer | 130 |
| 5.2.1 | Mengambil Alamat Memory dari Variabel | 130 |
| 5.2.2 | Menyimpan Alamat Variabel pada Pointer | 132 |
| 5.2.3 | Memberi Nama Pointer | 133 |
| 5.2.4 | Mengambil Nilai dari Variabel | 133 |
| 5.2.5 | Mengganti alamat yang direferensi oleh Pointer | 136 |
| 5.3 | Pointer dan Array | 137 |
| 5.3.1 | Kapan kita menggunakan pointer? | 139 |
| 5.3.2 | Membuat objek pada heap | 142 |
| 5.3.3 | Menggunakan const Pointer | 145 |



| | |
|--|------------|
| 5.4 Apa itu Reference | 145 |
| 5.4.1 Re-assign Reference Variable | 148 |
| 5.4.2 Passing function argument dengan reference | 149 |
| 5.4.3 Function yang mengembalikan beberapa nilai | 153 |
| 5.4.4 Passing By Reference untuk Efisiensi | 157 |

5.1 Apa itu Pointer?

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

5.2 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 menyimpan 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.

5.2.1 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 Menampilkan alamat memory menggunakan address-of operator.

Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 5.1, kemudian tulis kode berikut.

Listing 5.1: Menampilkan alamat memory menggunakan address-of operator

```
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     unsigned short bil1 = 20;
8     ulong bil2 = 200000;
9     long bil3 = -670000;
10    cout << "bil1 = " << bil1 << " address = " << &bil1
11        << endl;
12    cout << "bil2 = " << bil2 << " address = " << &bil2
13        << endl;
14    cout << "bil3 = " << bil3 << " address = " << &bil3
15        << endl;
16    return a.exec();
17 }
```

Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, 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.

5.2.2 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.

```
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 diinisialisasi 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.

5.2.3 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 inisialisasi 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
```

5.2.4 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 *derefensi*, ketika pointer di dereferensi maka nilai dari variabel yang alamatnya ditunjuk oleh pointer dapat diambil.

```
int number = *pBil; //mengambil nilai variabel yg  
alamatnya disimpan pada pointer pBil
```

Pada kode diatas 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 Memanipulasi data menggunakan Pointer

Buka Qt Creator, buat project Qt Console Application dengan nama Contoh 5.2. Kemudian tulis kode berikut.

Listing 5.2: Memanipulasi data menggunakan Pointer

```
1 #include <QtCore/QCoreApplication>  
2 #include <iostream>  
3 int main(int argc, char *argv[]){  
4     using namespace std;  
5     QCoreApplication a(argc, argv);  
6     ushort umur;  
7     ushort *pUmur = 0;  
8     umur = 17;  
9     cout << "Umur : " << umur << endl;  
10    pUmur = &umur;  
11    cout << "pUmur : " << *pUmur << endl;
```



```
13 cout << "Merubah nilai pUmur.." << endl;
14 *pUmur = 28;
15 cout << "Umur : " << umur << endl;
16 cout << "pUmur : " << *pUmur << endl;
17 cout << "Merubah nilai umur.." << endl;
18 umur = 30;
19 cout << "Umur : " << umur << endl;
20 cout << "pUmur : " << *pUmur << endl;
21 return a.exec();
22 }
```

Tekan Ctrl+R 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.

5.2.5 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 Mengganti alamat yang di referensi oleh pointer

Buat project Qt Console Application baru, beri nama Contoh 5.3, kemudian tulis kode berikut

Listing 5.3: Mengganti alamat yang di referensi oleh pointer

```
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     ushort umur1 = 17, umur2 = 28;
8     ushort *pUmur = &umur1; //ganti referensi
9     cout << "umur1 : " << umur1 << " alamat : " << &
10    umur1 << endl;
11    cout << "pUmur : " << *pUmur << " alamat : " <<
12    pUmur << endl;
13    pUmur = &umur2;
14    cout << "umur2 : " << umur2 << " alamat : " << &
15    umur2 << endl;
16    cout << "pUmur : " << *pUmur << " alamat : " <<
17    pUmur << endl;
18    return a.exec();
19 }
```



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.

5.3 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 Pointer dan Array

Buat project Qt Console Application dengan nama Contoh 5.4, kemudian tulis kode berikut.

Listing 5.4: Pointer dan Array

```
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     const int ARRAY_LENGTH = 5;
8     int numbers[ARRAY_LENGTH] = {100, 200, 222, 111, 777};
9     //mengakses alamat pertama dari array (numbers[0])
10    cout << "Alamat numbers[0] : " << numbers << endl;
11    //mengakses nilai dari elemen pertama array (numbers
12      [0])
13    cout << "Nilai numbers[0] : " << *numbers << endl;
14    //mengakses alamat numbers[4]
15    cout << "Alamat numbers[4] : " << numbers+4 << endl;
16    //mengakses nilai dari numbers[4]
17    cout << "Nilai numbers[4] : " << *(numbers+4) <<
18      endl;
19    const int *pNumber = numbers;
20    //menggunakan pointer untuk mencetak semua elemen
21      array
22    for(int i=0; i<ARRAY_LENGTH; i++)
23    {
24        cout << "numbers[" << i << "] = " << *(pNumber+i)
25        << endl;
26    }
27    return a.exec();
28 }
```



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).

5.3.1 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.



1. 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:

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

2. 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 Mengalokasikan, menggunakan, dan mendelete Pointer

Buat project Qt Console Application dengan nama Contoh 5.5, kemudian tulis kode berikut:



Listing 5.5: Mengalokasikan menggunakan dan mendelete Pointer

```
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     int bil = 20;
8     //pointer yang menunjuk ke alamat lokal
9     int *pBil = &bil;
10    cout << "bil : " << bil << endl;
11    cout << "pBil : " << *pBil << endl;
12    //mengalokasikan memory di heap untuk menyimpan data
13    //integer
14    int *pHeap = new int;
15    //nilai 19 akan disimpan di heap yg sudah
16    //dialokasikan
17    *pHeap = 19;
18    cout << "Nilai pHeap : " << *pHeap << endl;
19    delete pHeap;
20    pHeap = 0; //null pointer
21    //mengalokasikan memory
22    pHeap = new int;
23    *pHeap = 100;
24    cout << "Nilai pHeap : " << *pHeap << endl;
25    delete pHeap;
26    return a.exec();
27 }
```

Tekan Ctrl+R 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 dialokasikan 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).

5.3.2 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 penulisannya sama dengan sebelumnya.

```
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 Membuat dan menghapus objek dari Heap

Buat project Qt Console Application dengan nama Contoh 5.6, kemudian tulis kode berikut:

Listing 5.6: Membuat dan menghapus objek dari Heap

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class Mahasiswa
5 {
6 public:
7 Mahasiswa();
8 ~Mahasiswa();
9 private:
10 float ipk;
11 };
12 Mahasiswa::Mahasiswa()
13 {
14 cout << "Konstruktor dipanggil.." << endl;
15 ipk=3.5;
16 }
17 Mahasiswa::~Mahasiswa()
18 {
19 cout << "Destruktor dipanggil.." << endl;
20 }
21 int main(int argc, char *argv[])
22 {
```



```
23 QCOREAPPLICATION a(argc, argv);
24 cout << "Deklarasi object tanpa pointer " << endl;
25 Mahasiswa mhs1;
26 cout << "Mengalokasikan heap memory untuk menyimpan
     objek " << endl;
27 Mahasiswa *mhs2 = new Mahasiswa;
28 cout << "Delete objek di memory " << endl;
29 delete mhs2;
30 mhs2 = 0; //null pointer
31 return a.exec();
32 }
```

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 destruktur.
- 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 destruktur akan dipanggil.
- Objek desktruktur untuk mhs1 akan dipanggil ketika fungsi main berakhir.



5.3.3 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:

```
const int * pBil1;
int * const pBil2;
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.

5.4 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 :

```
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 Membuat dan Menggunakan Reference.

Buat project Qt Console Application dengan nama Contoh 5.7, kemudian tulis kode berikut:

Listing 5.7: Membuat dan Menggunakan Reference

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 int main(int argc, char *argv[])
5 {
6     QCoreApplication a(argc, argv);
7     int bil1 = 18;
8     int &rBil = bil1;
9     cout << "Nilai bil1 : " << bil1 << endl;
10    cout << "Nilai &rBil : " << rBil << endl;
11    bil1 = 19;
12    cout << "Nilai bil1 : " << bil1 << endl;
13    cout << "Nilai &rBil : " << rBil << endl;
14    rBil = 33;
```



```
15 cout << "Nilai bil1 : " << bil1 << endl;
16 cout << "Nilai &rBil : " << rBil << endl;
17 cout << "Menampilkan alamat memory :" << endl;
18 cout << "&bil1 : " << &bil1 << endl;
19 cout << "&rBil : " << &rBil << endl;
20 return a.exec();
21 }
```

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.



5.4.1 Re-assign Reference Variable

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

Contoh Re-assign Reference Value

Buat project Qt Console Application dengan nama Contoh 5.8, kemudian tulis kode berikut:

Listing 5.8: Re-assign Reference Value

```
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     int bil = 14;
8     int &rBil = bil;
9     cout << "rBil : " << rBil << endl;
10    int bil2 = 19;
11    rBil = bil2; //tebak hasilnya !
12    cout << "rBil : " << rBil << endl;
13    cout << "bil : " << bil << endl;
14    return a.exec();
15 }
```

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`.

5.4.2 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 Passing by Value.

Buat project Qt Console Application dengan nama Contoh 5.9, kemudian tulis kode berikut:

Listing 5.9: Passing by Value

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 void Tukar(int x,int y);
5 int main(int argc, char *argv[])
6 {
7     QApplication a(argc, argv);
8     int x=12, y=21;
9     cout << "Pada main, sebelum ditukar x=" << x << ", y "
10    =" << y << endl;
11    Tukar(x,y);
```



```
11 cout << "Pada main, setelah ditukar x=" << x << ", y  
12     =" << y << endl;  
13 return a.exec();  
14 }  
15 void Tukar(int x,int y)  
16 {  
17     int tampung;  
18     cout << "Pada fungsi, sebelum ditukar, x=" << x << "  
19         , y=" << y << endl;  
20     tampung = x;  
21     x=y;  
22     y=tampung;  
23     cout << "Pada fungsi, Setelah ditukar, x=" << x << "  
24         , y=" << y << endl;  
25 }
```

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 Passing by reference dengan pointer

Buat project Qt Console Application dengan nama Contoh 5.10, kemudian tulis kode berikut:



Listing 5.10: Passing by reference dengan pointer

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

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 Menjalankan fungsi Tukar() dengan reference

Buat project Qt Console Application dengan nama Contoh 5.11, kemudian tulis kode berikut:

Listing 5.11: Menjalankan fungsi Tukar() dengan reference

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



```
19  x = y;  
20  y = tampung;  
21  cout << "Sesudah function, sebelum ditukar x=" << x  
     << ", y=" << y << endl;  
22 }
```

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.

5.4.3 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 Mengembalikan beberapa nilai dengan pointer

Buat project Qt Console Application dengan nama Contoh 5.12, kemudian tulis kode berikut:



Listing 5.12: Mengembalikan beberapa nilai dengan pointer

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 int Hitung(int number, int *pLuasPersegi, int *
5             pVolumeKubus);
6 int main(int argc, char *argv[])
7 {
8     QCoreApplication a(argc, argv);
9     int number, pLuasPersegi, pVolumeKubus;
10    short error;
11    cout << "Masukan number : ";
12    cin >> number;
13    error = Hitung(number,&pLuasPersegi,&pVolumeKubus)
14        ;
15    if(!error)
16    {
17        cout << "Number : " << number << endl;
18        cout << "pLuasPersegi : " << pLuasPersegi <<
19            endl;
20        cout << "pVolumeKubus : " << pVolumeKubus <<
21            endl;
22    }
23    else
24        cout << "Terjadi Error !! ";
25    return a.exec();
26 }
27 int Hitung(int number, int *pLuasPersegi, int *
28             pVolumeKubus)
29 {
30     short status;
31     if(number > 0)
32     {
```



```
28     *pLuasPersegi = number * number;
29     *pVolumeKubus = number * number * number;
30     status = 0;
31 }
32 else
33 {
34     status = 1;
35 }
36 return status;
37 }
```

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, pLuasPerseg i, dan pVolumeKubus yang merupakan parameter bertipe pointer.
- pLuasPerseg i dan pVolumeKubus nilainya dapat bukan karena nilai kembalian dari function, tapi karena parameter by reference dari function yang berupa pointer, sehingga ketika nilai pLuasPerseg i dan pVolumeKubus diubah di dalam function nilai variabel asli di main function juga berubah.



Contoh Mengembalikan beberapa nilai dengan reference variabel

Buat project Qt Console Application dengan nama Contoh 5.13, kemudian tulis kode berikut:

Listing 5.13: Mengembalikan beberapa nilai dengan reference variabel

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 enum ERR_STATUS {SUCCESS, ERROR};
5 ERR_STATUS Hitung(int, int &, int &);
6 int main(int argc, char *argv[])
7 {
8     QCoreApplication a(argc, argv);
9     ERR_STATUS status;
10    int number, rLuasPersegi, rVolumeKubus;
11    cout << "Masukan number : ";
12    cin >> number;
13    status = Hitung(number, rLuasPersegi, rVolumeKubus);
14    if(status==SUCCESS)
15    {
16        cout << "Number : " << number << endl;
17        cout << "pLuasPersegi : " << rLuasPersegi <<
18            endl;
19        cout << "pVolumeKubus : " << rVolumeKubus <<
20            endl;
21    }
22    else
23        cout << "Terjadi Error !!";
24    return a.exec();
25 }
26 ERR_STATUS Hitung(int number, int &rLuasPersegi,
27                     int &rVolumeKubus)
28 {
```



```
26     ERR_STATUS status;
27     if(number > 0)
28     {
29         rLuasPersegi = number * number;
30         rVolumeKubus = number * number * number;
31         status = SUCCESS;
32     }
33     else
34         status = ERROR;
35     return status;
36 }
```

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.

5.4.4 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



sendiri oleh user) ini akan menurunkan performa dari program. Untuk melakukan passing parameter objek melalui function disarankan menggunakan reference pada objek.

Contoh Passing Object By Value

Buat project Qt Console Application dengan nama Contoh 5.14, kemudian tulis kode berikut:

Listing 5.14: Passing Object By Value

```
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 Mahasiswa::Mahasiswa()
12 {
13     cout << "Memanggil Mahasiswa Konstruktor " << endl
14     ;
15 }
16 Mahasiswa::Mahasiswa(Mahasiswa &)
17 {
18     cout << "Memanggil Copy Konstruktor " << endl;
19 }
20 Mahasiswa::~Mahasiswa()
21 {
22     cout << "Memanggil Mahasiswa Destruktor " << endl;
23 }
```



```
23 Mahasiswa FunctionMhs(Mahasiswa objMhs);
24 int main(int argc, char *argv[])
25 {
26     QApplication a(argc, argv);
27     cout << "Membuat object mahasiswa " << endl;
28     Mahasiswa objMhs1;
29     FunctionMhs(objMhs1);
30     return a.exec();
31 }
32 Mahasiswa FunctionMhs(Mahasiswa objMhs)
33 {
34     cout << "Mengembalikan FunctionMhs .." << endl
35     ;
36     return objMhs;
}
```

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 fun-



ction.

- Cara yang lebih efisien akan dibahas pada contoh program selanjutnya.

Contoh Passing Object By Reference

Buat project Qt Console Application dengan nama Contoh 5.15, kemudian tulis kode berikut:

Listing 5.15: Passing Object By Reference

```
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    Mahasiswa::Mahasiswa()
12    {
13        cout << "Memanggil Mahasiswa Konstruktor " <<
14            endl;
15    }
16    Mahasiswa::Mahasiswa(Mahasiswa &)
17    {
18        cout << "Memanggil Copy Konstruktor " << endl;
19    }
20    Mahasiswa::~Mahasiswa()
21    {
22        cout << "Memanggil Mahasiswa Destruktor " << endl;
23    }
24    Mahasiswa &FunctionMhs(Mahasiswa &objMhs);
```



```
24 int main(int argc, char *argv[])
25 {
26     QCoreApplication a(argc, argv);
27     cout << "Membuat object mahasiswa " << endl;
28     Mahasiswa objMhs1;
29     FunctionMhs(objMhs1);
30     return a.exec();
31 }
32     Mahasiswa &FunctionMhs(Mahasiswa &objMhs)
33 {
34     cout << "Mengembalikan FunctionMhs .." << endl;
35     return objMhs;
36 }
```

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.



Bagian II

Objek Oriented Programming (OOP)



BAB 6

Class dan Object

Agenda

Pada bab ini kita akan mempelajari konsep dari Pemrograman Berorientasi Objek tentang class dan objek seperti berikut ini.

Contents

| | | |
|------|---|-----|
| 6.1 | Pemrograman Berorientasi Obyek | 166 |
| 6.2 | Kelas | 168 |
| 6.3 | Object | 168 |
| 6.4 | Class dan Object | 169 |
| 6.5 | Pembuatan Class pada C++ | 169 |
| 6.6 | Mendefinisikan Obyek | 171 |
| 6.7 | Mengakses Member Variabel | 171 |
| 6.8 | Mengakses Member Function/Method | 172 |
| 6.9 | Hak Akses Member Variabel dan Method Variabel | 180 |
| 6.10 | Member Function / Member Method | 184 |
| 6.11 | Accessor dan Mutator Method | 187 |



| | |
|---|------------|
| 6.11.1 Accessor method | 188 |
| 6.11.2 Mutator method | 188 |
| 6.12 Constructor dan Destructor | 191 |
| 6.12.1 Default Constructor | 192 |
| 6.12.2 Constructor Dengan nilai Default | 199 |
| 6.12.3 const member method | 202 |
| 6.13 Mendefinisikan Method Member | 206 |
| 6.14 Class yang bertipe Class lain | 209 |

6.1 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 pemanipulasi objek kadang-kadang diekspre-



sikan 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).



6.2 Kelas

Kelas adalah Blue Print dari objek, yaitu prototype yang mendefinisikan variabel-variabel dan metoda/metoda (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.

6.3 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.



6.4 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.

6.5 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.

6.6 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:

Kucing kucingku;

Orang anton;

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.

6.7 Mengakses Member Variabel

Setelah kita membuat obyek seperti:



```
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:

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

Demikian juga dengan nama, umur, dan jenis kelamin.

```
Katty.nama = "Katty";
Katty.jenis_kelamin = "jantan";
Katty.umur = 2;
```

6.8 Mengakses Member Function/Method

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

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

Contoh Pembuatan class Sepeda

Buatlah project baru dan tulis kode berikut:

Listing 6.1: Pembuatan class Sepeda

```
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 void Sepeda::ubahKecepatan(int kec){
16 this->kecepatan = kec;
17 }
18 void Sepeda::ubahGigi(int g){
19 this->gigi = g;
20 }
21 void Sepeda::setMerk(string m){
22 this->merk = m;
23 }
24 void Sepeda::tampilSepeda(){
25 cout << "Kecepatan: " << this->kecepatan << endl <<
26 "Merk: " << this->merk << endl <<
27 "Gigi: " << this->gigi;
28 }
29 int main(int argc, char *argv[])
{
30 QCoreApplication a(argc, argv);
31 Sepeda objSpd;
32 objSpd.ubahGigi(2);
33 objSpd.ubahKecepatan(30);
35 objSpd.setMerk("Federal");
36 objSpd.tampilSepeda();
37 return a.exec();
38 }
```

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 memiliki 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 :

```
void Sepeda::ubahKecepatan(int kec){  
    Sepeda::kecepatan = kec;
```



}

Contoh Pembuatan obyek Sepeda.

Buatlah project baru dan tulis kode berikut:

Listing 6.2: Pembuatan obyek Sepeda

```
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 void Sepeda::ubahKecepatan(int kec){
16     this->kecepatan = kec;
17 }
18 void Sepeda::ubahGigi(int g){
19     this->gigi = g;
20 }
21 void Sepeda::setMerk(string m){
22     this->merk = m;
23 }
24 void Sepeda::tampilSepeda(){
25     cout << "Kecepatan: " << this->kecepatan << endl <<
26     "Merk: " << this->merk << endl <<
```



```
27 "Gigi: "<<this->gigi;  
28 }  
29 int main(int argc, char *argv[])  
30 {  
31 QCOREAPPLICATION a(argc, argv);  
32 cout<<"Sepeda pertama:\n";  
33 Sepeda objSpd;  
34 objSpd.ubahGigi(2);  
35 objSpd.ubahKecepatan(30);  
36 objSpd.setMerk("Federal");  
37 objSpd.tampilSepeda();  
38 cout<<"\nSepeda kedua:\n";  
39 Sepeda objSpd2;  
40 objSpd2.ubahGigi(1);  
41 objSpd2.ubahKecepatan(45);  
42 objSpd2.setMerk("Polygon");  
43 objSpd2.tampilSepeda();  
44 return a.exec();  
45 }
```

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 Pembuatan Obyek Array Sepeda

Buatlah program beikut ini:

Listing 6.3: Pembuatan Obyek Array Sepeda

```
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 void Sepeda::ubahKecepatan(int kec){
16     this->kecepatan = kec;
17 }
18 void Sepeda::ubahGigi(int g){
19     this->gigi = g;
20 }
21 void Sepeda::setMerk(string m){
```



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

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 6.3.
- 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 obj SpdArray 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.

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

maka bisa dilakukan:

```
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.

6.9 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 bandingkan 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 Perbedaan private dan public pada member variabel

1. Buatlah program berikut:

Listing 6.4: Public pada member variabel

```
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 int pkecepatan;
11 int pgigi;
12 string pmerk;
13 };
14 int main(int argc, char *argv[])
15 {
16 QApplication a(argc, argv);
17 cout<<"Pengaksesan public:\n";
18 Sepeda s;
19 s.pgigi = 3;
20 s.pkecepatan = 30;
21 s.pmerk = "Polygon";
22 cout<<"Gigi: "<<s.pgigi<<endl;
23 cout<<"Kecepatan: "<<s.pkecepatan<<endl;
24 cout<<"Merk: "<<s.pmerk<<endl;
25 return a.exec();
26 }
```

2. Hasil:

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

3. 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.

4. Ubahlah program pada Contoh 6.4 diatas menjadi seperti berikut ini:

Listing 6.5: Privat pada member variabel

```
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    int pkecepatan;
11    int pgigi;
12    string pmerk;
13 };
14 int main(int argc, char *argv[])
15 {
16     QCoreApplication a(argc, argv);
17     cout<<"Pengaksesan public:\n";
18     Sepeda s;
19     s.gigi = 3;
20     s.kecepatan = 30;
21     s.merk = "Federal";
22     cout<<"Gigi: "<<s.gigi<<endl;
23     cout<<"Kecepatan: "<<s.kecepatan<<endl;
24     cout<<"Merk: "<<s.merk<<endl;
25     return a.exec();
26 }
```

5. Hasil:



| In function 'int main(int, char*)': | |
|---|-------------|
| ① 'int Sepeda::gigi' is private | main.cpp 7 |
| ① within this context | main.cpp 19 |
| ① 'int Sepeda::kecepatan' is private | main.cpp 6 |
| ① within this context | main.cpp 20 |
| ① 'std::string Sepeda::merk' is private | main.cpp 8 |
| ① within this context | main.cpp 21 |
| ① 'int Sepeda::gigi' is private | main.cpp 7 |
| ① within this context | main.cpp 22 |
| ① 'int Sepeda::kecepatan' is private | main.cpp 6 |
| ① within this context | main.cpp 23 |
| ① 'std::string Sepeda::merk' is private | main.cpp 8 |
| ① within this context | main.cpp 24 |

6. 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.

6.10 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 contohnya.

```
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 berorientasi obyek pada umumnya member function minimal 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 Member function dan implementasinya.

Buatlah program berikut ini:

Listing 6.6: Member function dan implementasinya

```
1 #include <QtCore/QCoreApplication>  
2 #include <iostream>  
3 using namespace std;  
4 //pembuatan class Sepeda  
5 class Sepeda{  
6 private:  
7 //daftar member variabel  
8 int kecepatan;  
9 int gigi;  
10 string merk;  
11 public:  
12 //daftar member function  
13 void ubahKecepatan(int kec);  
14 void ubahGigi(int g);  
15 void setMerk(string m);  
16 void tampilSepeda();  
17 };  
18 //implementasi member function berada diluar class  
// Sepeda  
19 //function ubahKecepatan menerima input jumlah  
// kecepatan
```



```
20 //mengubah kecepatan Sepeda
21 void Sepeda::ubahKecepatan(int kec){
22     this->kecepatan = kec;
23 }
24 //function ubahGigi menerima input jumlah gigi
25 //mengubah gigi Sepeda
26 void Sepeda::ubahGigi(int g){
27     this->gigi = g;
28 }
29 //function setMerk menerima input string merk
30 //mengisi merk Sepeda
31 void Sepeda::setMerk(string m){
32     this->merk = m;
33 }
34 //function tampilSepeda tidak menerima input
35 //fungsinya hanya untuk menampilkan informasi obyek
36     Sepeda
37 void Sepeda::tampilSepeda(){
38     cout << "Kecepatan: " << this->kecepatan << endl <<
39     "Merk: " << this->merk << endl <<
40     "Gigi: " << this->gigi;
41 }
42 //function main
43 int main(int argc, char *argv[])
44 {
45     QCoreApplication a(argc, argv);
46     Sepeda objSpd;
47     objSpd.ubahGigi(2);
48     objSpd.ubahKecepatan(30);
49     objSpd.setMerk("Federal");
50     objSpd.tampilSepeda();
51     return a.exec();
52 }
```



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.

6.11 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.



6.11.1 Accessor method

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

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

Contoh:

```
int getUmur();
```

6.11.2 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 Penggunaan accessor dan mutator method

Tulislah program berikut ini:

Listing 6.7: Penggunaan accessor dan mutator method

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class Kucing{
5 private:
```



```
6 int umur;
7 float berat;
8 string nama;
9 public:
10 //asesor method
11 int getUmur();
12 float getBerat();
13 string getNama();
14 //mutator method
15 void setUmur(int u);
16 void setBerat(float b);
17 void setNama(string s);
18 //method tambahan
19 void berlari();
20 };
21 //implementasi
22 int Kucing::getUmur(){
23 return this->umur;
24 }
25 float Kucing::getBerat(){
26 return this->berat;
27 }
28 string Kucing::getNama(){
29 return this->nama;
30 }
31 void Kucing::setUmur(int u){
32 this->umur = u;
33 }
34 void Kucing::setBerat(float b){
35 this->berat = b;
36 }
37 void Kucing::setNama(string s){
38 this->nama = s;
39 }
```



```
40 void Kucing::berlari(){
41     cout<<"Kucing "<<this->getNama()<<" sedang berlari!";
42 }
43 int main(int argc, char *argv[])
44 {
45     QCoreApplication a(argc, argv);
46     Kucing mycat;
47     mycat.setNama("Katty");
48     mycat.setBerat(4);
49     mycat.setUmur(2);
50     cout<<"Kucingku bernama "<<mycat.getNama()<<", dia
      berbobot "<-
51     mycat.getBerat()<<" kg dan sudah berumur "<<mycat.
      getUmur()
52     <<" tahun sekarang."<<endl;
53     mycat.berlari();
54     return a.exec();
55 }
```

Hasil:

```
1 Kucingku bernama Katty,
2 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.

6.12 Constructor dan Destructor

Kita dapat mendeklarasikan variabel biasa dan kemudian melakukan inisialisasi terhadap variabel tersebut dengan mudah. Contoh:

```
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.

Desstruktor method merupakan method kebalikan dari constructor yang juga bernama sama dengan nama classnya namun diawali dengan tanda ~. Desstruktor 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();  
//destruktur  
~Sepeda();  
};
```

6.12.1 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 konsrukтор 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 Menggunakan Constructor dan Destructor.

Buatlah program berikut:

Listing 6.8: Menggunakan Constructor dan Destructor

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class Kucing{
5 private:
6     int umur;
7     float berat;
8     string nama;
9 public:
10    //konstruktor
11    Kucing(int umur);
12    //desktruktor
13    ~Kucing();
14    //asesor method
15    int getUmur();
16    float getBerat();
17    string getNama();
18    //mutator method
19    void setUmur(int u);
20    void setBerat(float b);
21    void setNama(string s);
22    };
23    //implementasi konstruktor dan desktruktor
24    Kucing::Kucing(int u){
25        this->umur = u;
26    }
27    Kucing::~Kucing(){
28    }
29    //implementasi function
```



```
30 int Kucing::getUmur(){
31     return this->umur;
32 }
33 float Kucing::getBerat(){
34     return this->berat;
35 }
36 string Kucing::getNama(){
37     return this->nama;
38 }
39 void Kucing::setUmur(int u){
40     this->umur = u;
41 }
42 void Kucing::setBerat(float b){
43     this->berat = b;
44 }
45 void Kucing::setNama(string s){
46     this->nama = s;
47 }
48 int main(int argc, char *argv[])
49 {
50     QCoreApplication a(argc, argv);
51     Kucing mycat(2);
52     mycat.setNama("Katty");
53     mycat.setBerat(4);
54     cout<<"Kucingku bernama "<<mycat.getNama()<<", dia
55         berbobot "<<
56     mycat.getBerat()<<" kg dan sudah berumur "<<mycat.
57         getUmur()
58     <<" tahun sekarang."<<endl;
59     mycat.setUmur(7);
60     cout<<"Lima tahun telah berlalu, sekarang kucingku
61         sudah berumur:
62     "<<mycat.getUmur()<<" tahun";
63     return a.exec();
64 }
```



```
61 }
```

Hasil:

```
1 Kucingku bernama Katty, dia berbobot 4 kg dan  
    sudah  
2 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 6.1 sebelum `return a.exec();`.

```
Kucing kucingku2;
```

Ketika dilakukan kompilasi akan menghasilkan error sebagai berikut:

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 Percobaan Menambah Constructor Method.

Buatlah program berikut:

Listing 6.9: Percobaan Menambah Constructor Method

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class Kucing{
5 private:
6     int umur;
7     float berat;
8     string nama;
9 public:
10 //konstruktor
11 Kucing(int umur);
12 Kucing();
13 //desktruktor
14 ~Kucing();
15 //asesor method
16 int getUmur();
17 float getBerat();
18 string getNama();
19 //mutator method
20     void setUmur(int u);
21 void setBerat(float b);
22 void setNama(string s);
```



```
23 } ;
24 //implementasi konstruktor dan desktruktor
25 Kucing::Kucing(int u){
26     this->umur = u;
27 }
28 Kucing::Kucing(){
29 }
30 Kucing::~Kucing(){
31     cout<<"Obyek sudah dihancurkan!" ;
32 }
33 //implementasi function
34 int Kucing::getUmur(){
35     return this->umur;
36 }
37 float Kucing::getBerat(){
38     return this->berat;
39 }
40 string Kucing::getNama(){
41     return this->nama;
42 }
43 void Kucing::setUmur(int u){
44     this->umur = u;
45 }
46 void Kucing::setBerat(float b){
47     this->berat = b;
48 }
49 void Kucing::setNama(string s){
50     this->nama = s;
51 }
52 int main(int argc, char *argv[])
53 {
54     QApplication a(argc, argv);
55     Kucing mycat(2);
56     mycat.setNama("Katty");;
```



```
57 mycat.setBerat(4);
58 cout<<"Kucingku bernama "<<mycat.getNama()<<", dia
      berbobot "<<
59 mycat.getBerat()<<" kg dan sudah berumur "<<mycat.
      getUmur()
60 <<" tahun sekarang."<<endl;
61 mycat.setUmur(7);
62 cout<<"Lima tahun telah berlalu, sekarang kucingku
      sudah berumur:
63 "<<mycat.getUmur()<<" tahun"<<endl;
64 Kucing kucingku2;
65 kucingku2.setNama("Frizky");
66 cout<<"Nama kucing keduaku: "<<kucingku2.getNama();
67 return a.exec();
68 }
```

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:

- 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;



6.12.2 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 Penggunaan Constructor dengan Nilai Default

Buatlah program berikut:

Listing 6.10: Penggunaan Constructor dengan Nilai Default

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class Buku{
5 private:
6     int jmlhal;
7     string pengarang;
8     string judul;
9 public:
10    Buku(string pengarang="unknown", string judul="unknown"
11          ,int jmlhal=1){
12        Buku::jmlhal = jmlhal;
13        Buku::pengarang = pengarang;
14        Buku::judul = judul;
15    }
16    void tampilInfo(){
17        cout<<"Judul: "<<Buku::judul<<endl;
18        cout<<"Pengarang: "<<Buku::pengarang<<endl;
19        cout<<"Jumlah halaman: "<<Buku::jmlhal<<endl;
20    }
21 }
```



```
20 } ;  
21 int main(int argc, char *argv[])  
22 {  
23     QCoreApplication a(argc, argv);  
24     Buku b1;  
25     Buku b2("Antonius");  
26     Buku b3("Robert", "Membuat aplikasi C++");  
27     Buku b4("Walter", "Pemrograman C", 100);  
28         b1.tampilInfo();  
29     b2.tampilInfo();  
30     b3.tampilInfo();  
31     b4.tampilInfo();  
32     return a.exec();  
33 }
```

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 menunjukkan 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:

Listing 6.11: Memberi nilai default pada constructor

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class Contoh{
5 private:
6     int x;
7     int y;
8     int z;
9 public:
10    Contoh():x(0),y(2),z(4){
11    }
12    void tampilInfo(){
13        cout<<"x="<<x<<" y="<<y<<" z="<<z;
14    }
```



```
15  } ;  
16 int main(int argc, char *argv[])  
17 {  
18     QCoreApplication a(argc, argv);  
19     Contoh aa;  
20     aa.tampilInfo();  
21     return a.exec();  
22 }
```

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 diinstansiasi maka otomatis konstruktor dipanggil dan semua nilai variabel member telah diinisialisasi seperti yang sudah dituliskan.

6.12.3 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 Penggunaan const method.

Buatlah program berikut ini:

Listing 6.12: Penggunaan const method

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class Kucing{
5 private:
6     int umur;
7     float berat;
8     string nama;
9 public:
10    //konstruktor
11    Kucing(int umur);
12    Kucing();
13    //desktruktor
14    ~Kucing();
15    //asesor method
16    int getUmur() const;
17    float getBerat() const;
18    string getNama() const;
19    //mutator method
20    void setUmur(int u);
21    void setBerat(float b);
22    void setNama(string s);
23 };
24 //implementasi konstruktor dan desktruktor
```



```
25 Kucing::Kucing(int u){  
26     this->umur = u;  
27 }  
28 Kucing::Kucing(){  
29 }  
30 Kucing::~Kucing(){  
31     cout<<"Obyek sudah dihancurkan!";  
32 }  
33 //implementasi function  
34 int Kucing::getUmur() const{  
35     return this->umur;  
36 }  
37 float Kucing::getBerat() const{  
38     return this->berat;  
39 }  
40 string Kucing::getNama() const{  
41     return this->nama;  
42 }  
43 void Kucing::setUmur(int u){  
44     this->umur = u;  
45 }  
46 void Kucing::setBerat(float b){  
47     this->berat = b;  
48 }  
49 void Kucing::setNama(string s){  
50     this->nama = s;  
51 }  
52 int main(int argc, char *argv[]){  
53 {  
54     QCOREAPPLICATION a(argc, argv);  
55     Kucing mycat(2);  
56     mycat.setNama("Katty");  
57     mycat.setBerat(4);
```



```
58 cout<<"Kucingku bernama "<<mycat.getNama()<<", dia  
59 berbobot "<<  
60 mycat.getBerat()<<" kg dan sudah berumur "<<mycat.  
61 getUmur()  
62 <<" tahun sekarang."<<endl;  
63 mycat.setUmur(7);  
64 cout<<"Lima tahun telah berlalu, sekarang kucingku  
65 sudah berumur:  
66 "<<mycat.getUmur()<<" tahun"<<endl;  
67 Kucing kucingku2;  
68 kucingku2.setNama("Frizky");  
69 cout<<"Nama kucing keduaku: "<<kucingku2.getNama();  
70 return a.exec();  
71 }
```

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:

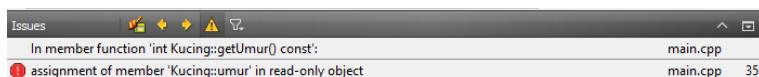
`this->umur = 5.` Kode lengkapnya adalah:

```
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.

6.13 Mendefinisikan Method Member

Selama ini kita mendefinisikan method member pada luar class. Selain cara diatas, kita juga bisa mendefinisikan 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 Inline Implementation.

Buatlah program berikut:

Listing 6.13: Implementasi inline

```
1 #include <QtCore/QCoreApplication>  
2 #include <iostream>  
3 using namespace std;  
4 class Manusia{  
5 private:  
6     string nama;  
7     char jenis_kelamin;  
8 public:  
9     //konstruktor  
10    Manusia(){  
11    }  
12    Manusia(string nama){  
13        this->nama = nama;  
14    }  
15    //desktruktor  
16    ~Manusia(){  
17    }  
18    //accessor method  
19    string getNama() const{  
20        return this->nama;  
21    }
```



```
22 char getJenis_Kelamin() const{
23     return this->jenis_kelamin;
24 }
25 //mutator method
26 void setNama(string n){
27     this->nama = n;
28 }
29 void setJenis_Kelamin(char jk){
30     this->jenis_kelamin = jk;
31 }
32 //method lain
33     void tampilSemua(){
34     cout<<"Nama: "<<this->getNama()<<", "<<
35     "Jenis Kelamin: "<<this->getJenis_Kelamin()<<endl;
36 }
37 };
38 int main(int argc, char *argv[])
39 {
40     QCoreApplication a(argc, argv);
41     Manusia suami;
42     suami.setNama("Susanto");
43     suami.setJenis_Kelamin('L');
44     Manusia istri("Susanti");
45     istri.setJenis_Kelamin('P');
46     Manusia anak("Rudi");
47     anak.setJenis_Kelamin('L');
48     suami.tampilSemua();
49     istri.tampilSemua();
50     anak.tampilSemua();
51     return a.exec();
52 }
```

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.

6.14 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 Class Mobil dan Class Roda.

Buatlah program berikut:

Listing 6.14: Class Mobil dan Roda

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class Roda{
```



```
5 private:
6 string merk_roda;
7 int diameter;
8 public:
9 Roda(string merk,int diamtr){
10 this->diameter = diamtr;
11 this->merk_roda = merk;
12 }
13 Roda(){
14 }
15 string getMerk(){
16 return this->merk_roda;
17 }
18 int getDiameter(){
19 return this->diameter;
20 }
21 void setMerkRoda(string m){
22 this->merk_roda = m;
23 }
24 void setDiameter(int d){
25 this->diameter = d;
26 }
27 };
28 class Mobil{
29 private:
30 string merk_mobil;
31 Roda roda_depan1;
32 Roda roda_depan2;
33 Roda roda_belakang1;
34 Roda roda_belakang2;
35 public:
36 //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
```

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 dapat 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 Class Titik dan Garis.

Buatlah program berikut ini:

Listing 6.15: Class titik dan Garis

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 #include <math.h>
4 using namespace std;
5 class Titik{
6 private:
7     int x;
8     int y;
9 public:
10    Titik(int x, int y){
11        this->x = x;
12        this->y = y;
13    }
14    Titik(){
15    }
16    int getX(){
17        return this->x;
18    }
19    int getY(){
20        return this->y;
21    }
22    void setX(int x){
23        this->x=x;
24    }
25    void setY(int y){
26        this->y=y;
```



```
27 }
28 //method tambahan
29 void printPoint(){
30 cout<<"("<<this->x<<" , " <<this->y<<") "<<endl;
31 }
32 bool isOrigin(){
33 return (this->x == 0 && this->y == 0);
34 }
35 };
36 class Garis{
37 private:
38 Titik p1;
39 Titik p2;
40 public:
41 Garis(int x1,int y1,int x2,int y2) {
42 p1.setX(x1);
43 p1.setY(y1);
44 p2.setX(x2);
45 p2.setY(y2);
46 }
47 Garis(){
48 }
49 Garis(Titik t1,Titik t2){
50 p1=t1;
51 p2=t2;
52 }
53 void setPoint1(Titik p1){
54 this->p1 = p1;
55 }
56 void setPoint2(Titik p2){
57 this->p2 = p2;
58 }
59 void setPoints(Titik p1,Titik p2){
60 this->p1 = p1;
```



```
61 this->p2 = p2;
62 }
63 Titik getPoint1(){
64 return this->p1;
65 }
66 Titik getPoint2(){
67 return this->p2;
68 }
69 void printLine(){
70 cout<<"awal: ";
71 this->p1.printPoint();
72 cout<<"akhir: ";
73 this->p2.printPoint();
74 }
75 //Hitung panjang garis
76 float getLength(){
77 return sqrt((this->p1.getX() - this->p2.getX())*(this
    ->p1.getX() - this-
    >p2.getX()) + (this->p1.getY() - this->p2.getY())*(
        this->p1.getY() - this-
        >p2.getY()));
78 }
79 };
80 }
81 };
82 int main(int argc, char *argv[])
83 {
84 QApplication a(argc, argv);
85 Titik A(1,1);
86 cout<<"A ";
87 A.printPoint();
88 Titik B(5,1);
89 cout<<"B ";
90 B.printPoint();
91 Titik C(5,6);
92 cout<<"C ";
```



```
93 C.printPoint();
94 Titik D(1,6);
95 cout<<"D ";
96 D.printPoint();
97 Garis ab(A,B);
98 cout<<"ab ";
99 ab.printLine();
100 cout<<"Panjang garis ab: "<<ab.getLength()<<endl;
101 Garis bc(B,C);
102 cout<<"bc ";
103 bc.printLine();
104 cout<<"Panjang garis bc: "<<bc.getLength()<<endl;
105 Garis cd(D,C);
106 cout<<"cd ";
107 cd.printLine();
108 cout<<"Panjang garis cd: "<<cd.getLength()<<endl;
109 Garis da(D,A);
110 cout<<"da ";
111 da.printLine();
112 cout<<"Panjang garis da: "<<da.getLength()<<endl;
113 return a.exec();
114 }
```

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!



BAB 7

Inheritance

Agenda

Pada bab ini kita akan mempelajari konsep dari Pemrograman Berorientasi Objek tentang Pewarisan (Inheritance) seperti berikut ini.

Contents

| | | |
|-------|---|-----|
| 7.1 | Pewarisan (Inheritance) | 222 |
| 7.2 | Penulisan Penurunan | 222 |
| 7.3 | Jenis Akses Penurunan Kelas | 225 |
| 7.4 | Warisan | 227 |
| 7.4.1 | Tiap Kelas Mempunyai Konstruktor | 228 |
| 7.4.2 | Konstruktor Kelas Turunan Pasti Memanggil Konstruktor Kelas Dasar | 230 |
| 7.5 | Mengganti Metode Kelas Dasar Pada Kelas Turunan (Overriding) | 237 |
| 7.6 | Memanggil Metode Kelas Dasar | 241 |
| 7.7 | Penyembunyian Metode Kelas Dasar | 243 |



| | |
|------------------------------------|-----|
| 7.8 Metode Virtual | 245 |
| 7.9 Pemotongan (Slicing) | 249 |
| 7.10 Memakai static_cast | 251 |

7.1 Pewarisan (Inheritance)

Pemrograman Berorientasi Objek mempunyai fitur penting yang memudahkan pemrogram dalam membuat program yaitu pewarisan (inheritance). Aspek penting pewarisan dalam pemrograman berorientasi objek adalah pemakaian kode program yang sudah ada (code reuse), yang akan dibahas pada bab ini dan polimorfisme (polymorphism) yang akan dibahas pada bab berikutnya. Kelas yang sudah ada dapat digunakan lagi untuk dikembangkan menjadi kelas yang baru, dalam hal ini kelas yang sudah ada dinamakan kelas dasar (base class) sedang kelas baru yang akan dibuat dinamakan kelas turunan (derived class). Dengan demikian, semua anggota kelas dasar yang tidak bersifat privat akan diwarisi oleh kelas turunannya dan pemrogram tinggal menambahkan anggota-anggota baru untuk menambahkan fungsionalitas kelas tersebut.

7.2 Penulisan Penurunan

Untuk membuat kelas turunan dapat dilakukan dengan cara sama seperti mendeklarasikan kelas biasa dengan menambahkan titik dua (:) setelah nama kelas dan diikuti dengan jenis penurunan (public dsb.) dan nama kelas dasar yang akan diturunkan, bentuk umum adalah seperti berikut:

```
class <kelas_turunan> : <jenis_penurunan> <kelas_dasar>
```

Sebagai contoh misalnya akan dibuat kelas turunan Silinder dari kelas dasar Lingkaran, maka dapat dituliskan :



```
class Silinder : public Lingkaran
```

Mengenai jenis akses public ini akan dibahas nanti, sekarang kita akan memakai public. Kelas dasar yang akan diturunkan harus sudah dideklarasikan lebih dahulu, jika tidak maka kita akan menjumpai pesan kesalahan kompiler.

Contoh Pewarisan (Inheritance).

Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 7.1, kemudian tulis kode berikut.

Listing 7.1: Pewarisan (Inheritance)

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class Lingkaran{
5 public:
6     //Konstruktor
7     Lingkaran(float radius = 0){
8         Lingkaran::radius = radius;
9     }
10    //Destruktor
11    ~Lingkaran(){}
12    protected :
13    float radius;
14    public :
15    float getLuas(){
16        return 3.14 * radius * radius;
17    }
18    };
19 class Silinder : public Lingkaran{
20 public:
21     //Konstruktor
```



```
22 Silinder(float radius, float tinggi){  
23     Silinder::radius = radius; //<-- warisan  
24     Silinder::tinggi = tinggi; //<-- anggota baru  
25 }  
26 //Destruktor  
27 ~Silinder(){}
28 private:  
29     float tinggi; //<-- anggota baru  
30 public:  
31     float getVolume(){ //<-- anggota baru  
32     //getLuas() adalah warisan  
33     return getLuas() * tinggi;  
34 }  
35 };  
36 int main(int argc, char *argv[])
37 {
38     QCOREAPPLICATION a(argc, argv);
39     Silinder drum(50, 125);
40     cout << "Volume drum = " << drum.getVolume() << " Cm2"
        << endl;
41     return a.exec();
42 }
```

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

```
Volume drum = 981250 Cm2
```

Keterangan Program :

- Pada program diatas sudah dibuat kelas Lingkaran, dengan variabel anggota `radius` bertipe `float` dan fungsi anggota `getLuas()` yang mengembalikan nilai `float`.
- Kelas kedua adalah `Silinder` yang merupakan turunan dari kelas `Ling-`



karan, oleh karena itu deklarasi dituliskan : `class Silinder : Lingkaran,` dengan demikian kelas `Silinder` akan mewarisi anggota kelas `Lingkaran` yang tidak private, yaitu : variabel anggota `radius` bertipe `float` dan fungsi anggota `getLuas()` yang mengembalikan nilai `float`.

- Tampak pada kelas `Silinder` ditambahkan variabel anggota tinggi bertipe `float` dan fungsi anggota `getVolume()` yang mengembalikan nilai `float`. Ini memberikan contoh penambahan fungsionalitas dari kelas yang sudah ada.
- Pada fungsi anggota `getVolume()` pada kelas `Silinder`, nilai kembalian dirumuskan `return getLuas() * tinggi`, ini menjelaskan bahwa fungsi `getLuas()` tersebut sekarang juga menjadi milik kelas `Silinder` (*mendapat warisan*).
- Pada hasil eksekusi, tampak bahwa fungsi `getVolume()` menghitung $(3.14 * 50 * 50) * 125$ dan menghasilkan nilai 981250. Ini berarti perhitungan `getLuas()` memakai fungsi anggota milik kelas `Lingkaran` yang diwariskan kepada kelas `Silinder`.

7.3 Jenis Akses Penurunan Kelas

Deklarasi kelas `Silinder` di atas adalah : `class Silinder : public Lingkaran,` ini berarti semua anggota yang bersifat `public` dan `protected` dari kelas `Lingkaran` akan diwariskan kepada kelas `Silinder` dan pada kelas `Silinder` anggota-anggota warisan tersebut akan tetap mempunyai jenis akses seperti itu. Namun jika modifier akses `public` dihilangkan maka berarti pewarisan memakai jenis akses `private`, sebab secara default C++ memakai jenis akses `private` jika modifier akses tidak dituliskan. Jika ini terjadi, maka akan terjadi perubahan modifier akses terhadap anggota-anggota warisan tersebut di dalam kelas `Silinder`, yaitu semua anggota yang diwariskan (baik berjenis `public` maupun `protected`) akan berubah menjadi `private` di dalam kelas `Silinder`.



Contoh Jenis Akses Public Pada Penurunan.

Buka project Contoh 7.1 di atas, kemudian tambahkan (edit) kode berikut pada fungsi main() : cout << "Milik Base Class --> " << drum.getLuas() << endl;

Listing 7.2: Jenis Akses Public Pada Penurunan

```
1 int main(int argc, char *argv[])
2 {
3     QApplication a(argc, argv);
4     Silinder drum(50,125);
5     cout << "Milik Base Class --> " << drum.getLuas() <<
6         endl;
7     cout << "Volume drum = " << drum.getVolume() << " Cm2"
8         << endl;
9     return a.exec();
10 }
```

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

```
Volume drum = 981250 Cm2
Milik Base Class --> 7850
```

Keterangan Program :

Tampak pada program dapat mengakses metode warisan kelas Lingkaran dari dalam program utama (main()). Ini menunjukkan bahwa metode tersebut diwariskan ke kelas Silinder dan jenis aksesnya masih tetap sama yaitu public.

Kemudian hapuslah jenis akses penurunan public (atau gantilah dengan private) pada deklarasi kelas Silinder yang tadinya:

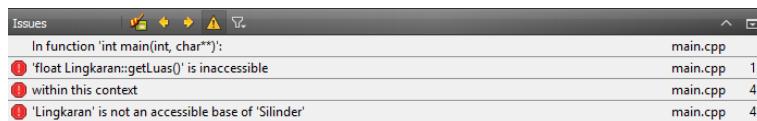
```
class Silinder : public Lingkaran
```

Sehingga menjadi:



```
class Silinder : Lingkaran
```

Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka tidak akan ada output karena terjadi kesalahan kompilasi sebagai berikut.



Keterangan Program :

Tampak pada program metode warisan kelas Lingkaran bernama getLuas() tidak dapat diakses dari dalam program utama (main()). Hal ini disebabkan karena metode tersebut ketika diwariskan ke kelas Silinder jenis aksesnya berubah menjadi private, yang berarti diwariskan akan tetapi hanya dapat diakses dari dalam kelas Silinder, akibatnya ketika akan diakses dari program utama (main()), terjadi kesalahan kompilasi seperti di atas.

Berikut ini adalah perubahan jenis akses anggota dari kelas dasar ke kelas turunan berdasarkan jenis akses penurunan:



TIPS

Pada umumnya jenis akses penurunan adalah *public*, oleh karena itu biasakan menuliskan jenis akses public ketika akan menurunkan suatu kelas.

7.4 Warisan

Anggota-anggota kelas (member variable dan member function) mempunyai jenis akses private, protected dan public, jika tidak dituliskan pada deklarasi anggota tersebut maka akan digunakan jenis akses private. Dalam hal pewarisan (inheritance) pada pemrograman berorientasi objek, seperti sudah dijelaskan di atas bahwa anggota yang diwariskan adalah anggota dengan jenis akses public



atau protected. Jenis akses public pada suatu anggota artinya bahwa anggota tersebut dapat diakses dari manapun dan akan diwariskan jika kelas tersebut diturunkan, sedangkan jenis akses protected berarti anggota tersebut diwariskan kepada kelas turunannya dan hanya bisa diakses dari dalam kelas turunan tersebut. Berikut ini adalah tabel yang menjelaskan jenis akses dan aksesibilitas suatu anggota:

| Aksesibilitas | public | protected | private |
|--------------------------------------|--------|-----------|---------|
| Dari dalam kelas itu sendiri | Ya | Ya | Ya |
| Dari kelas beda turunan | Ya | Ya | Tidak |
| Dari kelas beda tetapi bukan turunan | Ya | Tidak | Tidak |

Perlu diketahui bahwa konstruktor dan destruktur tidak diwariskan. Hal ini bisa dimaklumi, sebab konstruktor bekerja spesifik untuk kelas tersebut. Pada C++ jelas bahwa nama konstruktor sama dengan nama kelasnya, karena nama kelas turunan tidak mungkin sama dengan nama kelas dasar, maka tidak mungkin konstruktor kelas dasar juga merupakan konstruktor kelas turunan.

Namun demikian, pada konteks pewarisan, perlu diketahui bahwa itu tidak berarti konstruktor kelas dasar dapat diabaikan, sebab bagaimanapun juga dalam pembentukan objek konstruktor suatu kelas pasti bekerja (oleh karena itu diberi nama “konstruktor” yang artinya pembentuk). Berikut ini hal-hal yang perlu diperhatikan pada pewarisan mengenai konstruktor:

7.4.1 Tiap Kelas Mempunyai Konstruktor

Tidak ada kelas yang tidak mempunyai konstruktor. Adalah benar bahwa secara eksplisit kita bisa menuliskan sebuah kelas tanpa mendeklarasikan konstruktor sama sekali, namun itu tidak berarti bahwa kelas tersebut tidak mempunyai konstruktor, sebab sebenarnya yang dieksekusi oleh komputer bukan kode program yang kita tulis tersebut, melainkan hasil kompilasi dari kode program tersebut.



Pada waktu dikompilasi, kompiler akan menambahkan konstruktor tanpa parameter yang tidak melakukan apa-apa seperti berikut:

```
<nama_kelas>(){}  
 
```

Contoh Konstruktor default.

1. Buka Qt Creator, buat project Qt Console Application dengan nama Contoh 7.3. Kemudian tulis kode berikut.

Listing 7.3: Konstruktor default

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class Kelasku{
5 public:
6     Kelasku(){} //<-- konstruktor ini boleh tidak
    ditulis
7 public:
8     void hai(){
9         cout << "Hai, apa khabar...?" << endl;
10    }
11 };
12 int main(int argc, char *argv[])
13 {
14     QApplication a(argc, argv);
15     Kelasku test;
16     test.hai();
17     return a.exec();
18 }
```

2. Tekan Ctrl+R untuk menjalankan kode di atas, outputnya adalah sebagai berikut.



Hai, apa khabar...?

3. Kemudian hapuslah konstruktor Kelasku() {}, kemudian tekan Ctrl+R untuk menjalankan kode di atas, outputnya adalah sebagai berikut.

Hai, apa khabar...?

Keterangan Program:

Pada contoh program ini tampak bahwa ada konstruktor maupun tidak ada konstruktor program di atas tetap bisa dijalankan dan tidak ada perbedaan sama sekali. Hal ini disebabkan oleh karena jika suatu kelas tidak mempunyai konstruktor, maka secara otomatis kompiler akan menambahkan konstruktor default (yaitu konstruktor tanpa parameter dan tanpa program apapun) pada hasil kompilasi, jadi pada contoh program di atas hasil kompilasi dengan atau tanpa konstruktor adalah tetap sama.

7.4.2 Konstruktor Kelas Turunan Pasti Memanggil Konstruktor Kelas Dasar

Seperti dijelaskan di atas bahwa tidak ada kelas yang tidak mempunyai konstruktor, demikian juga dengan kelas turunan. Pada waktu ada pembentukan suatu objek dari suatu kelas turunan, secara otomatis ada terlebih dahulu pembentukan objek kelas dasarnya karena harus ada anggota-anggota yang diwariskan, dengan demikian bisa dimengerti bahwa konstruktor kelas turunan pasti memanggil konstruktor kelas dasarnya.

Sama seperti pada penulisan kelas biasa, pada kelas turunan juga bisa tidak dituliskan konstruktor secara eksplisit dan pada kasus ini pada saat kompilasi kompiler akan menambahkan konstruktor kosong tanpa parameter. Namun bentuk konstruktor kosong pada kelas turunan berbeda dengan konstruktor kosong kelas biasa, karena pada konstruktor kosong kelas turunan akan memanggil konstruktor kosong kelas dasarnya.



Contoh Konstruktor default kelas turunan.

1. Buka Qt Creator, buka project Qt Console Application dengan nama Contoh 7.4 tadi. Kemudian ubah isi konstruktor kelas Kelasku dan tambahkan kelas Turunan berikut.

Listing 7.4: Konstruktor default kelas turunan

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class Kelasku{
5 public:
6 Kelasku(){
7 cout << "Konstruktor Kelas Dasar dijalankan..." <<
8 endl;
9 }
10 public:
11 void hai(){
12 cout << "Hai, apa khabar...?" << endl;
13 }
14 class Turunan : public Kelasku{
15 };
16 int main(int argc, char *argv[])
17 {
18 QCoreApplication a(argc, argv);
19 Turunan test; //<-- membuat objek dari kelas
20 Turunan saja
21 return a.exec();
22 }
```

2. Tekan Ctrl+R untuk menjalankan kode di atas, outputnya adalah sebagai berikut.



Konstruktor Kelas Dasar dijalankan...

Keterangan Program:

- Pada contoh program ini tampak bahwa tidak ada objek yang dibuat dari kelas dasar, namun jika dilihat hasil eksekusinya, konstruktor yang dijalankan adalah konstruktor kelas dasar. Ini berarti bahwa ada pemanggilan konstruktor kelas dasar, yaitu pada waktu pembentukan objek kelas Turunan.
- Berdasarkan kenyataan bahwa tiap kelas pasti punya konstruktor, maka ini berarti kelas Turunan juga mempunyai konstruktor namun konstruktor tersebut di dalamnya ada pemanggilan konstruktor kelas dasarnya.
- Dari percobaan ini, bisa disimpulkan bahwa pada saat dikompilasi, karena kelas turunan secara eksplisit tidak dituliskan konstruktor, maka kompiler akan menambahkan konstruktor kosong tanpa parameter yang memanggil konstruktor kelas dasarnya seperti berikut :

```
Turunan(): Kelasku(){} //--- Konstruktor default  
kelas  
Turunan
```

Karakteristik konstruktor kelas turunan ini penting untuk dipahami, karena kadang-kadang kita lupa bahwa pada pembuatan kelas turunan pasti di dalamnya ada pemanggilan konstruktor kelas dasarnya.

Contoh Konstruktor default kelas turunan memanggil konstruktor kelas dasar.

1. Buka Qt Creator, buka project Qt Console Application dengan nama Contoh 7.5 tadi. Kemudian ubah isi konstruktor kelas Kelasku seperti berikut.

Listing 7.5: Konstruktor default kelas turunan memanggil konstruktor kelas dasar

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



```

2 #include <iostream>
3 using namespace std;
4 class Kelasku{
5 public:
6     Kelasku(string kata){
7         cout << "Konstruktor Kelas Dasar" << endl;
8         cout << "Mengucapkan : " << kata << endl;
9     }
10    public:
11    void hai(){
12        cout << "Hai, apa khabar...?" << endl;
13    }
14 };
15 class Turunan : public Kelasku{
16 };
17 int main(int argc, char *argv[])
18 {
19     QCoreApplication a(argc, argv);
20     Turunan test; //<-- membuat objek dari kelas
21     Turunan saja
22     return a.exec();
23 }
```

2. Tekan Ctrl+R untuk menjalankan kode di atas, tidak akan ada output karena ada kesalahan dengan pesan kesalahan kompilasi sebagai berikut.



The screenshot shows the Qt Creator interface with the 'Issues' tab selected. It displays a list of compilation errors:

- In constructor 'Turunan::Turunan()':
no matching function for call to 'Kelasku::Kelasku()'
candidates are:
 Kelasku::Kelasku(std::string)
 note: candidate expects 1 argument, 0 provided
 Kelasku::Kelasku(const Kelasku&)
 note: candidate expects 1 argument, 0 provided
- In function 'int main(int, char*)':
synthesized method 'Turunan::Turunan()' first required here

On the right side of the issues list, there is a column showing the file name 'main.cpp' and line numbers: 15, 15, 6, 6, 4, 4, 20.

Keterangan Program:



Seperti pada percobaan contoh 7.4, kelas Turunan tidak mempunyai konstruktor secara eksplisit, sehingga dibuatkan konstruktor oleh kompiler berupa :

```
Turunan():Kelasku(){} //<-- Konstruktor default kelas  
Turunan
```

Namun masalahnya sekarang pada kelas dasar tidak mempunyai konstruktor tanpa parameter seperti itu, sehingga kompiler manampilkan pesan :

```
no matching function call to 'Kelasku::Kelasku()'
```

Maksud pesan ini adalah bahwa pada kelas Kelasku tidak terdapat konstruktor Kelasku() dengan tanpa parameter. Ini juga membuktikan bahwa konstruktor default mempunyai bentuk seperti yang sudah dijelaskan pada Keterangan percobaan contoh 7.4.

Dengan demikian jelas bahwa konstruktor kelas dasar pasti dipanggil oleh konstruktor default kelas turunan. Akan tetapi bagaimanakah jika pada kelas turunan mempunyai konstruktor sendiri? Bisakah kita membuat konstruktor sendiri pada kelas turunan tanpa memanggil konstruktor kelas dasar? Tentu saja secara eksplisit bisa kita menuliskan konstruktor pada kelas turunan tanpa memanggil konstruktor kelas dasar, namun tetap saja kompiler nantinya akan menambahkan pemanggilan konstruktor default kelas dasar (tanpa parameter) jika pada konstruktor kelas turunan tidak memanggil salah satu konstruktor kelas dasarnya. Untuk meneliti mengenai hal ini, tambahkan konstruktor yang ditulis secara eksplisit pada kelas Turunan seperti berikut.

Contoh Konstruktor kelas turunan harus memanggil salah satu konstruktor kelas dasar.

1. Buka Qt Creator, buka project Qt Console Application dengan nama Contoh 7.6 tadi. Kemudian tambahkan konstruktor pada kelas Turunan seperti berikut.



Listing 7.6: Konstruktor kelas turunan harus memanggil salah satu konstruktor kelas dasar

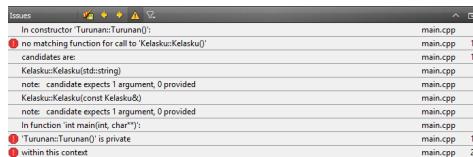
```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class Kelasku{
5 public:
6     Kelasku(string kata){
7         cout << "Konstruktor Kelas Dasar" << endl;
8         cout << "Mengucapkan : " << kata << endl;
9     }
10    public:
11    void hai(){
12        cout << "Hai, apa khabar...?" << endl;
13    }
14 };
15 class Turunan : public Kelasku{
16     Turunan(){} //--- membuat konstruktor pada kelas
17     turunan
18 };
19 int main(int argc, char *argv[])
20 {
21     QApplication a(argc, argv);
22     Turunan test; //--- membuat objek dari kelas
23     Turunan saja
24     return a.exec();
25 }
```

2. Tekan Ctrl+R untuk menjalankan kode di atas, tidak akan ada output karena ada kesalahan dengan pesan kesalahan kompilasi sebagai berikut.

Keterangan Program:

Pada program ini mencoba untuk menghindari pemanggilan konstruktor kelas dasar dengan cara membuat sendiri konstruktor pada kelas Turunan yang





tidak memanggil konstruktor kelas dasar. Namun tetap saja kompiler memberikan pesan kesalahan yang sama, yaitu :

no maching function call to 'Kelasku:::Kelasku()'

Padahal jelas pada konstruktor yang ditulis pada kelas Turunan sama sekali tidak pernah memanggil konstruktor tanpa kelas dasar parameter tersebut. Percobaan ini membuktikan bahwa bagaimanapun juga pada kelas turunan, konstruktor kelas dasar pasti dipanggil, dan jika secara eksplisit tidak dituliskan pemanggilan konstruktor kelas dasar, maka kompiler akan menambahkan pemanggilan konstruktor default (konstruktor tanpa parameter) pada kelas dasar.

3. Sekarang ubahlah konstruktor kelas Turunan, agar secara eksplisit memanggil konstruktor yang ada pada kelas dasar seperti berikut :

```
class Turunan : public Kelasku{\ public: //  
    memanggil konstruktor kelas
```

dasar dng sebuah parameter Turunan():Kelasku("Hallo.. :-"){} };

Tekan Ctrl+R untuk menjalankan kode di atas, tidak akan ada output karena ada kesalahan dengan pesan kesalahan kompilasi sebagai berikut.

Konstruktor Kelas Dasar
Mengucapkan : Hallo.. :-)

Keterangan Program:

- Pada program ini sekarang kelas Turunan mempunyai konstruktor yang memanggil konstruktor kelas dasar. Karena pada kelas dasar hanya mempunyai sebuah konstruktor dengan satu parameter bertipe string yang tan-

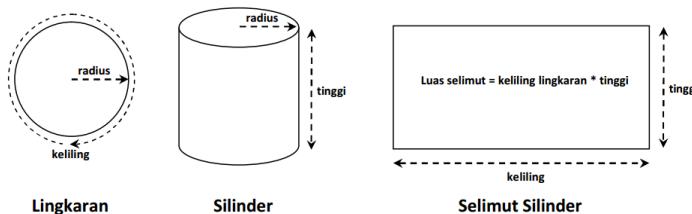


pa nilai default, maka yang dipanggil adalah konstruktor dengan sebuah parameter bertipe string.

- Dengan demikian perlu selalu diingat, bahwa konstruktor kelas turunan harus memanggil salah satu konstruktor kelas dasar, jika tidak dilakukan maka kompiler akan menambahkan pemanggilan konstruktor default kelas dasar.

7.5 Mengganti Metode Kelas Dasar Pada Kelas Turunan (Overriding)

Ada kalanya kelas turunan mempunyai implementasi lain untuk nama metode yang sama dengan kelas dasarnya. Sebagai contoh misalnya, kelas Lingkaran mempunyai implementasi (rumus) menghitung luas pada metode `getLuas()` adalah : $3.14 * \text{radius} * \text{radius}$, sedangkan kelas turunannya, misalkan Silinder mempunyai luas pada metode `getLuas()` yang terdiri dari dua luas tutup dan luas selimut dengan rumus $2 * (3.14 * \text{radius} * \text{radius}) + (2 * 3.14 * \text{radius}) * \text{tinggi}$ seperti gambar ilustrasi berikut:



Gambar 7.1: Ilustrasi metode class dasar pada class turunan dalam objek lingkaran, silinder dan persegi

Dengan demikian kelas turunan Silinder harus membuat implementasi yang berbeda untuk metode `getLuas()` supaya hasil dari metode tersebut



sesuai dengan objek Silinder. Membuat metode yang sama dengan metode milik kelas dasarnya dinamakan Overriding (override artinya mengesampingkan, atau boleh juga dikatakan menimpa) dengan demikian metode yang dipanggil untuk objek dari kelas Silinder adalah metode yang baru yang ditulis pada kelas Silinder.

Suatu metode bisa dikatakan override dari metode kelas dasarnya jika menuhi 2 syarat, yaitu *nama metode* dan *signature* dari metode tersebut sama. Signature artinya daftar parameter yang ada pada metode, yaitu banyaknya parameter maupun tipe data dari masing-masing parameter tersebut.

Contoh Melakukan Overriding.

1. Buka project Qt Console Application projek Contoh 7.7 yang tadi sudah dibuat, kemudian ubah kode menjadi seperti berikut

Listing 7.7: Melakukan Overriding

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class Lingkaran{
5 public:
6     //Konstruktor
7     Lingkaran(float radius = 0){
8         Lingkaran::radius = radius;
9     }
10    //Destruktor
11    ~Lingkaran(){}
12    protected :
13        float radius;
14    public :
15        float getLuas(){
16            return 3.14 * radius * radius;
17        }
```



```
18  };
19 class Silinder : public Lingkaran{
20 public:
21 //Konstruktor
22 Silinder(float radius, float tinggi){
23 Silinder::radius = radius; //<-- warisan
24 Silinder::tinggi = tinggi; //<-- anggota baru
25 }
26 //Destruktor
27 ~Silinder(){}
28 private:
29 float tinggi; //<-- anggota baru
30 public:
31 float getLuas(){ //<-- Overriding
32 float luasTutup = 3.14 * this->radius * this-
   radius;
33 float luasSelimut = 2 * 3.14 * this->radius * this-
   ->tinggi;
34 float luassilinder = 2 * luasTutup + luasSelimut;
35 return luassilinder;
36 }
37 float getVolume(){ //<-- anggota baru
38 //getLuas() sebenarnya sudah ditimpak
39 return getLuas() * tinggi;
40 }
41 };
42 int main(int argc, char *argv[])
43 {
44 QCoreApplication a(argc, argv);
45 Silinder drum(50,125);
46 cout << "Luas Silinder = " << drum.getLuas() << "
   Cm2" << endl;
47 cout << "Milik Base Class --> " << drum.getLuas()
   << endl;
```



```
48 cout << "Volume drum = " << drum.getVolume() << "
      Cm3" << endl;
49 return a.exec();
50 }
```

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

```
Luas Silinder = 54950 Cm2
Milik Base Class --> 54950
Volume drum = 6.86875e+006 Cm3
```

Bandingkan dengan hasil keluaran pada program di contoh 7.2 seperti berikut:

```
Volume drum = 981250 Cm2
Milik Base Class -> 7850
```

Keterangan:

- Pada program ini, pembuatan objek dilakukan dengan memberikan nilai radius=50 dan tinggi = 125, ini tampak pada program utama (main()) : Silinder drum(50, 125), sama dengan ketika membuat objek pada contoh 7.1. Namun tampak pada hasil eksekusi, Luas Silinder adalah 54950 Cm2 karena menggunakan rumus baru, tetapi pada keluaran Milik Base Class nilainya juga sama yaitu 54950, dan akibatnya volume drum yang seharusnya 981250 Cm3 menjadi 6.86875e+006 Cm3. Ini semua terjadi karena pada kelas Silinder tidak lagi memakai rumus getLuas() milik kelas dasarnya. Pada satu sisi getLuas() pada kelas Silinder sudah benar, sesuai dengan yang diharapkan, namun ketika metode getVolume() menghitung volume memakai metode getLuas(), ternyata sudah berubah menjadi rumus luas Silinder yang akibatnya perhitungan volume menjadi sangat besar (salah).
- Walaupun pada kasus ini menimbulkan masalah, tetapi percobaan ini memperlihatkan adanya overriding terhadap metode milik kelas dasar.



7.6 Memanggil Metode Kelas Dasar

Pada kasus contoh 7.7 di atas sebenarnya metode `getLuas()` masih dibutuhkan pada kelas Silinder untuk menghitung volume (`getVolume()`), bahkan sebenarnya untuk menghitung luas tutup, yang sebenarnya juga luas lingkaran, masih membutuhkan metode `getLuas()` milik kelas dasar Lingkaran. Namun karena adanya kebutuhan yang berbeda pada kelas Silinder untuk menghitung luasnya maka dilakukan overriding terhadap metode `getLuas()`. Untuk mengatasi hal ini diperlukan suatu cara untuk tetap dapat mengakses anggota milik kelas dasar, yaitu dengan cara menyebutkan nama kelas dasar kemudian diikuti dengan dua titik dua (::) dan anggota yang akan diakses.

<kelas_dasar>::<anggot

Contoh Mengakses metode kelas dasar.

1. Buka project Qt Console Application projek Contoh 7.8 yang baru saja dibuat, kemudian ubah kode pada metode : `getLuas()`, `getVolume()` dan program utama. Berikut ini adalah potongan program yang mengalami perubahan saja:

Listing 7.8: Mengakses metode kelas dasar

```
1 public:
2     float getLuas(){ //<-- Overriding
3         float luasSelimut = 2 * 3.14 * this->radius * this
4             ->tinggi;
5         float luassilinder = 2 * Lingkaran::getLuas() +
6             luasSelimut;
7         return luassilinder;
8     }
9     float getVolume(){ //<-- anggota baru
10        //getLuas() milik kelas dasar
11        return Lingkaran::getLuas() * tinggi;
```



```
10    }
11  };
12 int main(int argc, char *argv[])
13 {
14     QCoreApplication a(argc, argv);
15     Silinder drum(50,125);
16     cout << "Luas Silinder = " << drum.getLuas() << "
17         Cm2" << endl;
18     cout << "Milik Base Class --> " << drum.Lingkaran
19         ::getLuas() << endl;
20     cout << "Volume drum = " << drum.getVolume() << "
21         Cm3" << endl;
22     return a.exec();
23 }
```

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

```
Luas Silinder = 54950 Cm2
Milik Base Class --> 54950
Volume drum = 981250 Cm3
```

Keterangan:

- Sekarang keluaran “Luas Silinder” dengan “Milik Base Class” berbeda, yaitu 54950 sedangkan “Milik Base Class” yang menghitung luas lingkaran adalah 7850. Walaupun keduanya sama-sama memanggil metode `getLuas()`, namun metode yang dipanggil pada “Milik Base Class” adalah metode milik kelas Lingkaran, yaitu dengan cara penulisan nama metode ditambahkan nama kelas dasar dan dua titik dua (::) didepannya seperti berikut:

`drum.Lingkaran::getLuas()`

- Pada waktu menghitung `getLuas()` pada kelas Silinder, bisa memanfaatkan metode `getLuas()` milik kelas dasar dengan cara memanggil me-



tode milik kelas dasar, sehingga dengan demikian metode `getLuas()` ini seperti tampak pada kode program di atas, bisa menjadi lebih ringkas.

- Demikian juga pada metode `getVolume()`, sekarang tidak ada kesalahan seperti tadi, karena rumus `getLuas()` yang digunakan sudah benar, yaitu metode `getLuas()` milik kelas dasar.
- Dari percobaan contoh 7.7 ini tampak cara melakukan overriding dan cara memanggil anggota milik kelas dasar.

7.7 Penyembunyian Metode Kelas Dasar

Ketika terjadi overriding terhadap suatu metode kelas dasar, maka semua metode milik kelas dasar yang bernama sama, yaitu metode-metode yang dioverloading pada kelas dasar, akan disembunyikan (tidak diwariskan) kepada kelas turunan.

Contoh Penyembunyian metode kelas dasar.

1. Jalankan Qt Console Application projek, buat projek bernama Contoh 7.9 seperti berikut:

Listing 7.9: Penyembunyian metode kelas dasar

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class Dasar{
5 public:
6     void hallo() const{
7         cout << "Hallo" << endl;
8     }
9     void hallo(string kata) const{
10        cout << "Hallo " << kata << endl;
11    }
```



```
12 void hallo (string kata, string nama) const{
13     cout << "Hallo " << kata << " nama saya " << nama
14         << endl;
15 }
16 class Turunan : public Dasar{
17 public:
18 void hallo() const{ //<-- override terhadap salah
19     satu metode
20     cout << "Ni hao ma" << endl;
21 }
22 int main(int argc, char *argv[])
23 {
24     QCoreApplication a(argc, argv);
25     Turunan agus;
26     agus.hallo();
27     //agus.hallo("apa kahabar");
28     //agus.hallo("apa kahabar", "Agus");
29     return a.exec();
30 }
```

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

Ni hao ma

3. Hilangkan tanda comment pada pemanggilan metode hallo pada dua baris di progam utama contoh 7.10 menjadi seperti berikut:

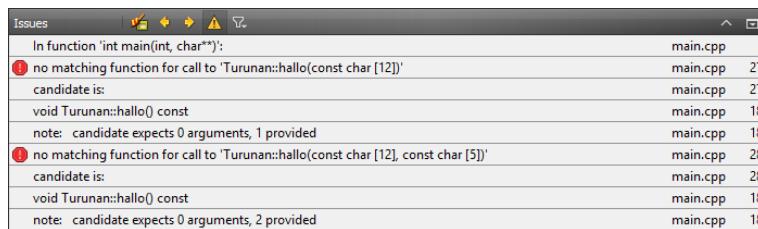
Listing 7.10: Menghilangkan comment pada metode hallo

```
1 int main(int argc, char *argv[])
2 {
3     QCoreApplication a(argc, argv);
4     Turunan agus;
```



```
5 agus.hallo();  
6 agus.hallo("apa kahabar");  
7 agus.hallo("apa kahabar", "Agus");  
8 return a.exec();  
9 }
```

4. Tekan Ctrl+R untuk menjalankan program diatas, maka tidak akan ada output dan menuliskan pesan kesalahan saat kompilasi seperti berikut.



Keterangan:

Tampak pada hasil kompilasi kedua tidak dijalankan, pesan kesalahan adalah tidak ditemukannya metode dengan sebuah parameter `char[12]` (`string`) dan metode dengan dua buah parameter kedua-duanya bertipe `char[12]` (`string`). Padahal jelas terlihat pada kelas Dasar terdapat kedua metode tersebut dan jenis aksesnya adalah `public`, bahkan cara penurunannya pada kelas Dasar juga menggunakan jenis akses `public`, seharusnya dengan cara demikian metode-metode tersebut diwariskan kepada kelas Turunan. Percobaan ini menunjukkan bahwa jika terjadi overriding terhadap suatu metode kelas dasar, maka semua metode milik kelas dasar yang bernama sama, yaitu metode-metode yang dioverloading pada kelas dasar, akan disembunyikan (tidak diwariskan) kepada kelas turunan.

7.8 Metode Virtual

Metode Virtual adalah metode yang seharusnya dioverride oleh kelas turunannya, dengan tujuan jika ada variabel pointer bertipe kelas dasarnya yang ber-



si objek bertipe kelas turunan tersebut, maka metode yang menanggapi adalah metode milik kelas turunan hasil override tersebut. Ini sebenarnya bagian yang sangat penting yang diperlukan pada polimorfisme yang akan dibahas pada bab berikutnya. Jika suatu metode pada kelas dasar tidak virtual, kemudian pada kelas turunan melakukan override terhadap metode tersebut, maka jika ada suatu variabel pointer bertipe kelas dasar yang berisi objek bertipe kelas turunan memanggil metode tersebut, yang menanggapi adalah metode milik kelas dasar. Percobaan berikut ini merupakan penyederhanaan dari kelas Lingkaran sebagai kelas dasar dan kelas Silinder sebagai kelas turunan untuk memahami metode virtual.

Contoh Metode virtual dan non virtual.

1. Jalankan Qt Console Application projek, buat projek bernama Contoh 7.11 seperti berikut:

Listing 7.11: Metode virtual dan non virtual

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class Lingkaran{
5 public:
6     virtual float getLuas(){
7         cout << "Luas Lingkaran" << endl;
8         return 0;
9     }
10 };
11 class Silinder : public Lingkaran{
12 public:
13     float getLuas(){
14         cout << "Luas Silinder" << endl;
15         return 0;
16 }
```



```
17 } ;  
18 int main(int argc, char *argv[])  
19 {  
20 QCOREAPPLICATION a(argc, argv);  
21 Lingkaran *objek1; //<-- variabel pointer bertipe  
kelas dasar  
22 objek1 = new Silinder(); //<-- objek bertipe kelas  
turunan  
23 objek1->getLuas(); //<-- memanggil metode yang  
dioverride  
24 return a.exec();  
25 }
```

2. Tekan Ctrl+R untuk menjalankan program diatas, hasil keluaran adalah seperti berikut.

Luas Silinder

3. Sekarang hapuslah kata kunci pada contoh 7.12 virtual metode getLuas() pada kelas Lingkaran seperti berikut.

Listing 7.12: Menghapus methode getluas

```
1 class Lingkaran{  
2 public:  
3 float getLuas(){  
4 cout << "Luas Lingkaran" << endl;  
5 return 0;  
6 }  
7 };
```

4. Tekan Ctrl+R untuk menjalankan program diatas, hasil keluaran adalah seperti berikut.

Luas Lingkaran



Keterangan:

- Tampak pada program utama, variabel pointer bertipe kelas dasar (Lingkaran) digunakan untuk menunjuk objek bertipe kelas turunan (Silinder). Ketika dipanggil metode `getLuas()` yang ada di kelas dasar maupun turunan, maka metode mana yang menanggapi tergantung apakah metode tersebut bersifat virtual pada kelas dasar atau tidak.
- Pada percobaan pertama, metode `getLuas()` pada kelas Lingkaran dibuat virtual, ketika program utama dijalankan, tampak bahwa yang menanggapi adalah metode milik kelas turunan (metode yang ada pada objek), yaitu mencetak “Luas Silinder”.
- Pada percobaan kedua, tidak ada perubahan kode program sama sekali kecuali menghilangkan kata kunci virtual pada metode `getLuas()` pada kelas Lingkaran, hasilnya tampak bahwa yang menanggapi adalah metode milik kelas dasar (metode yang ada pada kelas Lingkaran), yaitu mencetak “Luas Lingkaran”.
- Percobaan pertama, yaitu membuat metode virtual, adalah yang diperlukan pada proses polimorfisme yang akan dibahas pada bab berikutnya.

**Catatan**

- Kata kunci virtual tidak membuat metode tersebut harus dioverride.
- Pada kelas turunan metode virtual yang dioverride otomatis virtual walaupun tidak ditulis, namun sebaiknya untuk kemudahan perawatan sebaiknya kata kunci virtual ditulis.
- Variabel pointer bertipe kelas dasar jika digunakan untuk menunjuk objek bertipe kelas turunan, untuk mengakses anggota kelas turunannya, ia hanya bisa memanggil metode virtual yang dioverride oleh kelas turunannya. Dengan kata lain, variabel tersebut tidak bisa memanggil metode kelas turunan yang bukan merupakan override dari metode virtual kelas dasar.



7.9 Pemotongan (Slicing)

Perlu diperhatikan bahwa kemampuan untuk memanggil metode kelas turunan dari variabel bertipe kelas dasar hanya berlaku untuk variabel pointer dan variabel referensi, sedangkan variabel nilai (value variable) tidak dapat mengakses metode virtual seperti itu.

Ketika kita membuat objek bertipe kelas turunan, sebenarnya sebelumnya sudah dibuat objek bertipe kelas dasarnya, seperti sudah dibahas secara tidak langsung pada bagian yang membicarakan konstruktor di atas. Jadi ada bagian yang merupakan anggota kelas dasar dan ada bagian yang merupakan anggota kelas turunan. Sebagai contoh misalnya berikut ini ilustrasi mengenai objek bertipe Silinder yang merupakan turunan dari kelas Lingkaran yang dibahas pada contoh 7.10 di atas.



Gambar 7.2: Bagan pemotongan antara lingkaran dan silinder

Pada konversi dari suatu variabel ke variabel lain, bisa terjadi Pemotongan (Slicing). Supaya lebih jelas lakukan percobaan berikut ini.

Contoh Metode virtual dan non virtual.

Jalankan Qt Console Application projek, buka projek bernama contoh 7.13 yang dibuat tadi, kemudian ubah kode program pada bagian program utama seperti berikut:



Listing 7.13: Metode virtual dan non virtual

```
1 int main(int argc, char *argv[])
2 {
3     QCOREAPPLICATION a(argc, argv);
4     Silinder* s = new Silinder(); //<-- objek bertipe
5         Silinder
6     Lingkaran* objek1 = s; //<-- variabel pointer bertipe
7         kelas dasar
8     Lingkaran& objek2 = *s; //<-- variabel referensi
9         bertipe kelas dasar
10    Lingkaran objek3 = *s; //<-- variabel nilai bertipe
11        kelas dasar
12    cout << "Pointer : " ;
13    objek1->getLuas(); //<-- memanggil dai variabel
14        pointer
15    cout << "Referensi : " ;
16    objek2.getLuas(); //<-- memanggil dai variabel
17        referensi
18    cout << "Nilai : " ;
19    objek3.getLuas(); //<-- memanggil dai variabel nilai
20    return a.exec();
21 }
```

Tekan Ctrl+R untuk menjalankan program diatas, hasil keluaran adalah seperti berikut.

```
Pointer : Luas Silinder
Referensi : Luas Silinder
Nilai : Luas Lingkaran
```

Keterangan:

- Tampak pada hasil percobaan, pointer dan referensi memanggil metode virtual, sehingga yang dieksekusi adalah metode `getLuas()` milik Silinder. Ini tampak pada 2 baris pertama hasil keluaran di atas. Sedangkan



pada baris ke 3 metode yang dieksekusi adalah metode milik kelas dasar (Lingkaran) itu sendiri.

- Variabel objek3 bertipe kelas dasar (Lingkaran), maka ketika menerima objek bertipe kelas turunan (Silinder) kompiler memotong (slices down) objek Silinder menjadi bentuk kelas dasar (Lingkaran) saja. Oleh karena itu ketika dipanggil metode `getLuas()` maka yang ada hanya anggota-anggota kelas dasar (Lingkaran). Potongan objek bertipe kelas turunan (Silinder) hilang, inilah efek dari variabel nilai yang diberi nilai objek kelas turunannya, efek ini disebut Pemotongan (Slicing) karena bagian kelas turunan (Silinder) dipotong keluar ketika dikonversikan menjadi kelas dasar (Lingkaran).

7.10 Memakai static_cast

Seperti tertulis pada catatan di atas, pointer bertipe kelas dasar yang menunjuk objek bertipe kelas turunan tidak bisa memanggil metode-metode kelas turunan yang bukan merupakan override dari metode virtual kelas dasar. Namun dengan mekanisme casting hal ini bisa dilakukan.

Casting adalah mekanisme yang digunakan untuk mengubah tipe variabel dari tipe data ke tipe data yang lain. `static_cast` adalah mekanisme yang dapat digunakan untuk mengkonversikan pointer ke tipe pointer lain yang mempunyai hubungan kekerabatan (inheritance) dan melakukan konversi secara eksplisit tipe data standar. `static_cast` pada saat kompilasi melakukan pemeriksaan untuk memastikan bahwa pointer yang akan di-“cast” mempunyai hubungan yang benar. Dengan `static_cast` sebuah pointer dapat di-“up-casted” menjadi tipe kelas dasar, atau bisa pula di-“down-casted” menjadi bertipe kelas turunan. Jika suatu variabel bertipe kelas dasar diberi nilai objek bertipe kelas turunan sebagai berikut:

```
Kelas_dasar* variabel1 = new Kelas_turunan();
```



Maka tipe data variabel tersebut bisa di-“down-casted” menjadi tipe data kelas turunan dengan cara:

```
Kelas_turunan* variabel2 = static_cast<Kelas_turunan  
*(variabel1);
```

Contoh Memakai static_cast.

Jalankan Qt Console Application projek, buka projek bernama contoh 7.14 di atas, ubah pada bagian program utama (main()) menjadi seperti berikut:

Listing 7.14: Memakai static_cast

```
1 int main(int argc, char *argv[])
2 {
3     QCOREAPPLICATION a(argc, argv);
4     Lingkaran* objek1; //<-- variabel pointer bertipe
5         kelas dasar
6     objek1 = new Silinder(); //<-- objek bertipe kelas
7         turunan
8     objek1->getLuas(); //<-- memanggil metode yang
9         dioverride
7     static_cast<Silinder*>(objek1)->getVolume(); //<--
10        casting mjd Silinder
11     return a.exec();
12 }
```

Tekan Ctrl+R untuk menjalankan program diatas, hasil keluaran adalah sepeti berikut.

```
Luas Silinder
Volume Silinder
```

Keterangan:



- Tampak pada program utama, variabel pointer bertipe kelas dasar (Lingkaran), yaitu K, di-“down-casted” menjadi tipe data kelas turunan (Silinder). Ketika dipanggil metode `getVolume()` yang sebenarnya tidak ada di kelas dasar tetapi hanya ada di kelas turunan, tampak bahwa metode `getVolume()` milik kelas Silinder menanggapinya.
- Percobaan ini menunjukkan bahwa tipe data pointer dapat dikonversikan menjadi tipe data pointer lain.



Catatan

- Casting terhadap pointer hanya bisa dilakukan terhadap tipe-tipe yang mempunyai hubungan kekerabatan (inheritance).
- Dengan casting kita bisa memaksa variabel pointer bertipe kelas dasar yang digunakan untuk menunjuk objek bertipe kelas turunan untuk memanggil metode yang bukan merupakan override dari metode virtual kelas dasar, namun cara ini tidak cukup aman jika objek tersebut bukan dari kelas turunan.



BAB 8

Operator Types dan Operator Overloading

Agenda

Pada chapter ini kita akan membahas beberapa topik tentang penggunaan Operator Types dan Operator Overloading, adapun topik yang akan dibahas adalah

Contents

| | | |
|------------|--|------------|
| 8.1 | Operator pada C++ | 256 |
| 8.1.1 | Unary Operator | 257 |
| 8.2 | Conversion Operator | 263 |
| 8.2.1 | Binary Operator | 266 |
| 8.3 | Addition-Assignment Operator | 269 |
| 8.4 | Comparison Operator | 272 |
| 8.5 | Overloading Operator <, >, <=, >= | 277 |
| 8.6 | Subscript Operator | 280 |
| 8.7 | Function operator() | 283 |



8.1 Operator pada C++

Pada bab awal kita sudah mempelajari berbagai macam operator (+, -, /, >, <) yang dapat digunakan pada tipe data yang sudah ada di C++ seperti int, float, bool, dll. Namun jika anda ingin menggunakan operator tersebut pada tipe data yang anda definisikan sendiri seperti tipe data Class, maka anda dapat menggunakan keyword operator .

```
return_type operator operator_symbol (...parameter  
list...);
```

Penggunaan keyword operator sebenarnya mirip dengan penggunaan fungsi , hanya anda dapat menggunakan operator symbol seperti (+, -, >,< , =, dll). Mungkin anda bertanya kenapa harus menggunakan keyword operator jika anda dapat menggunakan fungsi ? Ilustrasi dibawah ini akan menunjukan kenapa kita membutuhkan operator.

```
CKataString strKata1("Hello");  
CKataString strKata2("World");
```

Jika anda menginginkan untuk menggabungkan kedua kata tersebut anda dapat membuat function Concatenate seperti berikut:

```
CKataString strGabung;  
strGabung = strKata1.Concatenate(strKata2);
```

Selain cara seperti diatas akan lebih natural jika anda menulis kode sebagai berikut:

```
CKataString strGabung;  
strGabung = strKata1 + strKata2;
```

Walaupun hasil dari kedua cara penulisan diatas sama, namun penggunaan operator + untuk menggabungkan string akan lebih intuitif dan mudah dipahami.

Ada 2 macam operator yang terdapat di C++ yaitu unary dan binary.



8.1.1 Unary Operator

Unary operator hanya mempunyai single operand saja, cara penulisan unary operator adalah sebagai berikut.

```
return_type operator operator_type (parameter_type)  
{  
// ... implementation  
}
```

Tipe dari unary operator yang dapat digunakan adalah

| Operator | Name |
|----------|---------------------|
| ++ | Increment |
| - | Decrement |
| * | Pointer Dereference |
| -> | Member Selection |
| ! | Logika NOT |
| & | Address-of |
| ~ | One's Complement |
| + | Unary Plus |
| - | Unary Negation |

Pada Labs1 akan ditunjukkan penggunaan operator increment. Pada contoh dibawah ini kita akan membuat Class Kalender yang mempunyai tiga class member yang merepresentasikan hari, bulan, dan tahun (tipe integer), anda dapat menggunakan operator ++ untuk menambahkan hari.

Contoh Menggunakan Increment Operator (Notasi prefix).

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama contoh 8.1, kemudian tulis kode berikut.



Listing 8.1: Menggunakan Increment Operator (Notasi prefix)

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class Kalender
5 {
6 private:
7     int _hari;
8     int _bulan;
9     int _tahun;
10    void AddHari(int hari)
11    {
12        _hari += hari;
13        if(_hari > 30)
14        {
15            AddBulan(_hari/30);
16            _hari %= 30;
17        }
18    }
19    void AddBulan(int bulan)
20    {
21        _bulan += bulan;
22        if(_bulan > 12)
23        {
24            AddTahun(_bulan/12);
25            _bulan %= 12;
26        }
27    }
28    void AddTahun(int tahun)
29    {
30        _tahun += tahun;
31    }
32 public:
```



```
33 Kalender(int hari,int bulan, int tahun):_hari(hari
34     ),_bulan(bulan),_tahun(tahun){ }
35 Kalender &operator ++ ()
36 {
37     AddHari(1);
38     return *this;
39 }
40 void TampilData()
41 {
42     cout << _hari << " / " << _bulan << " / " <<
43         _tahun << endl;
44 }
45 };
46 int main(int argc, char *argv[])
47 {
48     QCoreApplication a(argc, argv);
49     Kalender objKal(23,10,2010);
50     cout << "Membuat object kalender dan memberi
51         inisialisasi" << endl;
52     objKal.TampilData();
53     cout << endl;
54     ++objKal;
55     cout << "Tanggal setelah notasi prefik dijalankan
56         " << endl;
57     objKal.TampilData();
58     return a.exec();
59 }
```

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



```
Membuat object kalender dan memberi inisialisasi  
23 / 10 / 2010
```

```
Tanggal setelah notasi prefik dijalankan  
24 / 10 / 2010
```

Keterangan:

- Class Kalender berisi tiga member variabel yaitu `_hari`, `_bulan`, dan `_tahun` yang merepresentasikan waktu tertentu.
- operator `++` digunakan untuk menambahkan 1 hari kedalam objek Kalender, dengan menggunakan operator `++` penulisan menjadi lebih intuitif dan mudah dipahami, misal untuk menambahkan 1 hari kedalam objek Kalender anda dapat menuliskan `++objKal`.
- Untuk memanggil operator `++` pada program diatas digunakan notasi prefix (tanda `++` dituliskan sebelum nama objek).
- Karena kode diatas menggunakan notasi prefix maka pada operator `++` akan mengakses objek by reference.

Ada perbedaan penulisan notasi prefix dan postfix, sebagai contoh anda dapat melihat kode dibawah ini.

```
int bil1 = 22;  
int bil2 = bil1++;  
//mengcopy nilai lama dari bil1  
cout << "bil2 : " << bil2;  
//nilai bil1 setelah di increment  
cout << "bil1 : " << bil1;
```

Nilai `bil2` adalah 22, karena yang dimasukan kedalam `bil2` adalah nilai lama dari `bil1`, baru setelah itu `bil1` di increment.

Untuk contoh dibawah ini kita akan mencoba menggunakan notasi `postfix`, dengan notasi `postfix` yang dilakukan adalah menduplikat objek yang diinputkan, melakukan increment dan mengembalikan objek tersebut by value.



Contoh Menggunakan Operator Increment (notasi postfix).

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama contoh 8.2, kemudian tulis kode berikut.

Listing 8.2: Menggunakan Operator Increment (notasi postfix)

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class Kalender
5 {
6     private:
7     int _hari;
8     int _bulan;
9     int _tahun;
10    void AddHari(int hari)
11    {
12        _hari += hari;
13        if(_hari > 30)
14        {
15            AddBulan(_hari/30);
16            _hari %= 30;
17        }
18    }
19    void AddBulan(int bulan)
20    {
21        _bulan += bulan;
22        if(_bulan > 12)
23        {
24            AddTahun(_bulan/12);
25            _bulan %= 12;
26        }
27    }
28    void AddTahun(int tahun)
```



```
29  {
30  _tahun += tahun;
31 }
32 public:
33 Kalender(int hari,int bulan, int tahun):_hari(hari
34     ),_bulan(bulan),_tahun(tahun){ }
35 Kalender operator ++ (int)
36 {
37     Kalender objKal(_hari,_bulan,_tahun);
38     AddHari(1);
39     return objKal;
40 }
41 void TampilData()
42 {
43     cout << _hari << " / " << _bulan << " / " <<
44     _tahun << endl;
45 }
46 };
47 int main(int argc, char *argv[])
48 {
49     QCoreApplication a(argc, argv);
50     Kalender objKal(23,10,2010);
51     cout << "Membuat object kalender dan memberi
52         inisialisasi" << endl;
53     objKal.TampilData();
54     cout << endl;
55     cout << "Menggunakan notasi postfix" << endl;
56     //menggunakan notasi postfix
57     Kalender objLama(objKal++);
58     cout << "objLama : ";
59     objLama.TampilData();
60     cout << "objKal : ";
61     objKal.TampilData();
62     return a.exec();
```



```
60  
61 }
```

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

```
Membuat object kalender dan memberi inisialisasi  
23 / 10 / 2010
```

```
Menggunakan notasi postfix  
objLama : 23 / 10 / 2010  
objKal : 24 / 10 / 2010
```

Keterangan:

- Output yang dihasilkan akan sama dengan kode sebelumnya, hanya saja penulisan operator ++ menggunakan notasi `postfix`.
- Karena menggunakan `postfix` maka yang dilakukan pada operator ++ adalah mengkopi objek yang lama, menambahkan data, kemudian mengembalikan objek tersebut by value.

8.2 Conversion Operator

Bagaimana jika anda menginginkan statement `int bil = Kalender(23,10,2010)` memiliki arti? Untuk itu anda perlu mengkonversi dari objek Kalender menjadi tipe data integer, dengan demikian anda akan dengan mudah mengirimkan data Kalender ke module lain yang hanya menerima parameter bertipe integer.

Anda dapat melakukan konversi diatas dengan menggunakan conversion operator yang mempunyai syntax sebagai berikut:

```
operator conversion_type();
```

Jadi jika anda menghendaki mengkonversi tipe Kalender menjadi int anda dapat menggunakan operator berikut.



```
operator int()
{
// implementation
return intValue;
}
```

Contoh 8.3 dibawah ini akan menunjukan bagaimana penggunaan conversion operator untuk mengonversi tipe Kalender menjadi int.

Contoh Conversion Operator untuk konversi class Kalender ke integer.

Buka Qt Creator dan buat project Qt Console Application baru dengan nama contoh 8.3, kemudian tulis kode berikut.

Listing 8.3: Conversion Operator untuk konversi class Kalender ke integer

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class Kalender
5 {
6 private:
7     int _hari;
8     int _bulan;
9     int _tahun;
10 public:
11     Kalender(int hari,int bulan,int tahun) : _hari(hari),
12             _bulan(bulan),
13             _tahun(tahun) {}
14     operator int()
15     {
16         return ((_tahun*1000) + (_bulan*100) + _hari);
17     }
18     void TampilData()
```



```
18 {
19     cout << _hari << " / " << _bulan << " / " << _tahun <<
20     endl;
21 }
22 int main(int argc, char *argv[])
23 {
24     QCOREAPPLICATION a(argc, argv);
25     Kalender objKal(23,10,2010);
26     cout << "Inisialisasi data : " << endl;
27     objKal.TampilData();
28     cout << endl; cout << endl;
29     int nData = objKal;
30     cout << "Integer yang sesuai dengan data " << nData <<
31     endl;
32     return a.exec();
33 }
```

Hasilnya

```
Inisialisasi data :
23 / 10 / 2010

Integer yang sesuai dengan data 2011023
```

Keterangan:

- Pada operator `int()`, variabel `_tahun`, `_bulan`, dan `_hari` dikalikan dengan bilangan tertentu sehingga menghasilkan kembalian berupa `int`.
- Statement `int nData = objKal` akan menjalankan operator `int` dan mengembalikan nilai integer dari objek `Kalender`.
- Dengan menggunakan operator `int` akan lebih mudah membandingkan dua objek `Kalender`, karena objek tersebut dapat mengembalikan satu nilai



integer.

8.2.1 Binary Operator

Operator yang mengoperasikan dua operand disebut dengan binary operator, cara penulisan binary operator sama dengan penulisan operator yang sebelumnya.

```
return_type operator_type (parameter);
```

Ada beberapa macam binary operator yang dapat digunakan pada C++, diantaranya :

| Operator | Name |
|----------|--------------------------|
| + | Addition |
| += | Addition/Assignment |
| - | Subtraction |
| -= | Subtraction/Assignment |
| < | Less Than |
| > | Greater Than |
| <= | Less than or equal to |
| >= | Greater than or equal to |
| == | Equal to |
| != | Inequality |

Contoh program dibawah ini menggunakan operator Addition (+) untuk menambahkan hari pada objek kalender, anda dapat menambahkan beberapa hari kedepan, misal 5 atau 10 hari dari tanggal sekarang.

Contoh Menggunakan Binary Addition Operator.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama contoh 8.4, kemudian tulis kode berikut.



Listing 8.4: Menggunakan Binary Addition Operator

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class Kalender
5 {
6 private:
7     int _hari;
8     int _bulan;
9     int _tahun;
10    void TambahHari(int hari)
11    {
12        _hari += hari;
13        if(_hari>30)
14        {
15            TambahBulan(_hari/30);
16            _hari %= 30;
17        }
18    }
19    void TambahBulan(int bulan)
20    {
21        _bulan += bulan;
22        if(_bulan>12)
23        {
24            TambahTahun(_bulan/12);
25            _bulan %= 12;
26        }
27    }
28    void TambahTahun(int tahun)
29    {
30        _tahun += tahun;
31    }
32 public:
```



```
33 Kalender(int hari, int bulan, int tahun) : _hari(
34     hari), _bulan(bulan),
35     _tahun(tahun) {}
36 Kalender operator + (int hari)
37 {
38     Kalender objKal(_hari,_bulan,_tahun);
39     objKal.TambahHari(hari);
40     return objKal;
41 }
42 void TampilTanggal()
43 {
44     cout << _hari << " / " << _bulan << " / " <<
45     _tahun << endl;
46 }
47 };
48 int main(int argc, char *argv[])
49 {
50     QCoreApplication a(argc, argv);
51     Kalender objKal(23,10,2011);
52     cout << "Inisialisasi Data " << endl;
53     objKal.TampilTanggal();
54     cout << "Menambahkan 25 hari kedepan " << endl;
55     Kalender objKalBaru(objKal + 25);
56     cout << "Hasil setelah ditambahkan 25 hari " <<
57         endl;
58     objKalBaru.TampilTanggal();
59 //cara penulisan yang lain
60     objKal=objKal+20;
61     objKal.TampilTanggal();
62     return a.exec();
63 }
```

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



Inisialisasi Data

```
23 / 10 / 2011 Menambahkan 25 hari kedepan  
Hasil setelah ditambahkan 25 hari 18 / 11 / 2011  
13 / 11 / 2011
```

Keterangan:

- Dengan menggunakan operator + anda dapat menambahkan hari pada objek Kalender, jumlah hari yang ditambahkan tergantung dari nilai yang diinputkan pada parameter.
- Anda dapat menambah hari dengan menggunakan operator + pada objek Kalender, misal: objKal = objKal + 25 atau dengan membuat objek baru untuk menampung nilai hasil penambahan Kalender objKal-Baru(objKal+20)

8.3 Addition-Assigment Operator

Dengan menggunakan Addition-Assigment operator anda dapat menuliskan sintaks `a += b`, yang sama artinya dengan `a = a + b`. Pada contoh program dibawah ini operator *Addition-Assigment* akan digunakan untuk menambahkan hari pada objek Kalender.

Contoh Menggunakan Addition Assigment Operator dan Substraction Assigment Operator.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama contoh 8.5, kemudian tulis kode berikut.

Listing 8.5: Menggunakan Addition Assigment Operator dan Substraction Assigment Operator

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



```
2 #include <iostream>
3 using namespace std;
4 class Kalender
5 {
6 private:
7 int _hari;
8 int _bulan;
9 int _tahun;
10 void TambahHari(int hari)
11 {
12 _hari += hari;
13 if(_hari>30)
14 {
15 TambahBulan(_hari/30);
16 _hari %= 30;
17 }
18 }
19 void TambahBulan(int bulan)
20 {
21 _bulan += bulan;
22 if(_bulan>12)
23 {
24 TambahTahun(_bulan/12);
25 _bulan %= 12;
26 }
27 }
28 void TambahTahun(int tahun)
29 {
30 _tahun += tahun;
31 }
32 public:
33 Kalender(int hari, int bulan, int tahun) : _hari(
34     hari), _bulan(bulan),
35     _tahun(tahun) {}
```



```
35 void operator += (int hari)
36 {
37     TambahHari(hari);
38 }
39 void TampilTanggal()
40 {
41     cout << _hari << "/" << _bulan << "/" << _tahun
        << endl;
42 }
43 };
44 int main(int argc, char *argv[])
45 {
46     QCoreApplication a(argc, argv);
47     Kalender objKal(23,10,2011);
48     cout << "Inisialisasi Data" << endl;
49     objKal.TampilTanggal();
50     cout << "Menambahkan 25 hari kedepan" << endl;
51     objKal+=25;
52     cout << "Hasil setelah ditambahkan 25 hari" <<
        endl;
53     objKal.TampilTanggal();
54     return a.exec();
55 }
```

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

```
Inisialisasi Data
23/10/2011
Menambahkan 25 hari kedepan
Hasil setelah ditambahkan 25 hari
18/11/2011
```

Keterangan:



Pada kode diatas anda dapat menggunakan operator Addition-Assignment pada objek Kalender, misal anda ingin menambahkan 25 hari pada objek Kalender, anda dapat menuliskan kode `objKal += 25;`

8.4 Comparison Operator

Pada kasus tertentu dimana anda ingin membandingkan dua objek bertipe Kalender ada dapat menggunakan comparison operator.

```
if (objKal1 == objKal2)
{
    // Do something
}
else
{
    // Do something else
}
```

Anda dapat menggunakan equality operator (`==`) atau inequality operator (`!=`). Anda juga dapat membuat lebih dari satu equality atau inequality operator yang mempunyai return value atau parameter yang berbeda, ini disebut dengan overloading operator.

Contoh Overloading Comparison Operator (Equality dan Inequality).

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama contoh 8.6, kemudian tulis kode berikut.

Listing 8.6: Overloading Comparison Operator (Equality dan Inequality)

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class Kalender
```



```
5  {
6  private:
7  int _hari;
8  int _bulan;
9  int _tahun;
10 void TambahHari(int hari);
11 void TambahBulan(int bulan);
12 void TambahTahun(int tahun);
13 public:
14 Kalender(int hari,int bulan, int tahun) : _hari(
15     hari), _bulan(bulan),
16     _tahun(tahun) {}
17 void TampilTanggal()
18 {
19     cout << _hari << " / " << _bulan << " / " <<
20     _tahun << endl;
21 }
22 operator int();
23 bool operator == (const Kalender &objKal);
24 bool operator != (const Kalender &objKal);
25 bool operator != (int tglNumber);
26 Kalender::operator int()
27 {
28     return ((_tahun*10000)+(_bulan*100)+_hari);
29 }
30 bool Kalender::operator ==(const Kalender &objKal)
31 {
32     return ((objKal._hari==_hari) && (objKal._bulan==
33         _bulan) &&
34     (objKal._tahun==_tahun));
35 bool Kalender::operator ==(int tglNumber)
```



```
36  {
37  return tglNumber == (int)*this;
38 }
39 bool Kalender::operator !=(const Kalender &objKal)
40 {
41 return !(this->operator ==(objKal));
42 }
43 bool Kalender::operator !=(int tglNumber)
44 {
45 return !(this->operator ==(tglNumber));
46 }
47 void Kalender::TambahHari(int hari)
48 {
49 _hari += hari;
50 if(_hari>30)
51 {
52 TambahBulan(_hari/30);
53 _hari %= 30;
54 }
55 }
56 void Kalender::TambahBulan(int bulan)
57 {
58 _bulan += bulan;
59 if(_bulan>12)
60 {
61 TambahTahun(_bulan/12);
62 _bulan %= 12;
63 }
64 }
65 void Kalender::TambahTahun(int tahun)
66 {
67 _tahun += tahun;
68 }
69 int main(int argc, char *argv[])
```



```
70  {
71  QCoreApplication a(argc, argv);
72  Kalender objKal1(23,10,2010);
73  cout << "Inisialisasi Kalender 1" << endl;
74  objKal1.TampilTanggal();
75  cout << endl;
76  Kalender objKal2(23,10,2011);
77  cout << "Inisialisasi Kalender 2" << endl;
78  objKal2.TampilTanggal();
79  cout << endl;
80 //menggunakan operator tidak sama dengan
81 if(objKal1 != objKal2)
82 cout << "kalender1 dan kalender2 tidak sama !" <<
     endl;
83 Kalender objKal3(23,10,2010);
84 cout << "Inisialisasi Kalender 3" << endl;
85 objKal3.TampilTanggal();
86 cout << endl;
87 //menggunakan operator sama dengan
88 if(objKal1==objKal3)
89 cout << "kalender1 dan kalender3 sama !" << endl;
90 int intKal3 = objKal3;
91 cout << "nilai integer yang ekuivalen dengan
     objKal3 adalah " << intKal3 << endl;
92 //menggunakan overloading operator sama dengan
     untuk membandingkan integer
93 if(objKal3 == intKal3)
94 cout << "Nilai integer dari objKal3 dan intKal3
     sama" << endl;
95 //menggunakan overloading operator tidak sama
     dengan untuk membandingkan integer
96 if(objKal2 != intKal3)
97 cout << "Nilai integer dari objKal2 dan intKal3
     tidak sama" << endl;
```



```
98 return a.exec();  
99 }
```

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

```
Inisialisasi Kalender 1  
23 / 10 / 2010
```

```
Inisialisasi Kalender 2  
23 / 10 / 2011
```

```
kalender1 dan kalender2 tidak sama !
```

```
Inisialisasi Kalender 3  
23 / 10 / 2010
```

```
kalender1 dan kalender3 sama !
```

```
nilai integer yang ekuivalen dengan objKal3 adalah 20101023  
Nilai integer dari objKal3 dan intKal3 sama  
Nilai integer dari objKal2 dan intKal3 tidak sama
```

Keterangan:

- Pada kode diatas terdapat 4 comparison operator yang berbeda, walaupun masing-masing ada 2 *inequality* dan *equality* operator namun parameter-nya berbeda ini disebut sebagai *overloading operator*.
- Pada operator equality (==) yang pertama membandingkan dua objek Kalender dengan cara membandingkan member variabel hari, bulan, tahun pada masing-masing objek yang dibandingkan, sedangkan operator == yang kedua membandingkan objek Kalender yang terlebih dahulu sudah dikonversi menjadi int.
- Anda dapat melihat bahwa kedua operator comparison diatas sama-sama dapat membandingkan isi dari 2 objek Kalender, baik dengan cara mem-



bandingkan member variabel maupun membandingkan nilai int (hasil konversi dari objek Kalender).

8.5 Overloading Operator <, >, <=, >=

Seperti pada contoh sebelumnya anda juga dapat menggunakan operator <, >, <=, >= untuk membandingkan objek Kalender. Agar mudah untuk dibandingkan maka objek Kalender dikonversi terlebih dahulu menjadi tipe int.

Contoh Menggunakan Operator <, >, <=, >=.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama contoh 8.7, kemudian tulis kode berikut.

Listing 8.7: Menggunakan Operator Overloading

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class Kalender
5 {
6 private:
7     int _hari;
8     int _bulan;
9     int _tahun;
10    void TambahHari(int hari);
11    void TambahBulan(int bulan);
12    void TambahTahun(int tahun);
13 public:
14    Kalender(int hari,int bulan, int tahun) : _hari(
15        hari), _bulan(bulan),
16        _tahun(tahun) {}
17    void TampilTanggal()
```



```
17  {
18  cout << _hari << " / " << _bulan << " / " <<
19    _tahun << endl;
20  }
21  operator int() const;
22  bool operator < (const Kalender &objKal) const;
23  bool operator <= (const Kalender &objKal) const;
24  bool operator > (const Kalender &objKal) const;
25  bool operator >= (const Kalender &objKal) const;
26  };
27  Kalender::operator int() const
28  {
29  return ((_tahun*10000)+(_bulan*100)+_hari);
30  }
31  bool Kalender::operator <(const Kalender &objKal)
32      const
33  {
34  return (this->operator int() < objKal.operator int
35      ());
36  }
37  bool Kalender::operator >(const Kalender &objKal)
38      const
39  {
40  return (this->operator int() > objKal.operator int
41      ());
42  }
43  bool Kalender::operator <=(const Kalender &objKal)
44      const
45  {
46  return (this->operator int() <= objKal.operator
47      int());
48  }
49  bool Kalender::operator >=(const Kalender &objKal)
50      const
```



```
43  {
44  return (this->operator int() >= objKal.operator
45    int());
46 }
47 {
48 QCoreApplication a(argc, argv);
49 Kalender objKal1(23,10,2010);
50 Kalender objKal2(16,10,1980);
51 Kalender objKal3(23,10,2010);
52 cout << "objKal1 berisi : " << endl;
53 objKal1.TampilTanggal();
54 cout << endl;
55 cout << "objKal2 berisi : " << endl;
56 objKal2.TampilTanggal();
57 cout << endl;
58 cout << "objKal3 berisi : " << endl;
59 objKal3.TampilTanggal();
60 cout << endl;
61 //menggunakan operator <
62 cout << "objKal1 < objKal2 = ";
63 cout << ((objKal1 < objKal2) ? "true" : "false")
64   << endl;
65 //menggunakan operator >
66 cout << "objKal1 > objKal2 = ";
67 cout << ((objKal1 > objKal2) ? "true" : "false")
68   << endl;
69 //menggunakan operator <=
70 cout << "objKal1 <= objKal3 = ";
71 cout << ((objKal1 <= objKal3) ? "true" : "false")
72   << endl;
73 //menggunakan operator >=
74 cout << "objKal1 >= objKal3 = ";
```



```
72 cout << ((objKal1 >= objKal3) ? "true" : "false")
    << endl;
73 return a.exec();
74 }
```

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

```
objKal1 berisi :
23 / 10 / 2010

objKal2 berisi :
16 / 10 / 1980

objKal3 berisi :
23 / 10 / 2010

objKal1 < objKal2 = false
objKal1 > objKal2 = true
objKal1 <= objKal3 = true
objKal1 >= objKal3 = true
```

Keterangan

- Pada program diatas penggunaan operator <, >, <=, >= digunakan untuk membandingkan dua objek Kalender yang berbeda.
- Untuk mempermudah membandingkan dua objek Kalender maka objek Kalender tersebut dikonversi terlebih dahulu menjadi `int`.

8.6 Subscript Operator

Subscript operator dapat digunakan jika anda ingin mengakses class seperti ketika anda mengakses array, anda dapat menambahkan operator [] pada objek



yang anda buat untuk mengakses nilai dengan index tertentu dari objek. Contoh dibawah ini akan menjelaskan penggunaan subscript operator untuk membuat array yang dinamis.

Contoh Subscript Operator untuk Dynamic Array.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama contoh 8.8, kemudian tulis kode berikut.

Listing 8.8: Subscript Operator untuk Dynamic Array

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class COpArray
5 {
6 private:
7     int *_myArray;
8     int _numElement;
9 public:
10    COpArray(int numElement);
11    ~COpArray();
12    int& operator[](int index);
13 };
14    int& COpArray::operator [](int index)
15 {
16     return _myArray[index];
17 }
18    COpArray::COpArray(int numElement)
19 {
20     _myArray = new int[numElement];
21     _numElement = numElement;
22 }
23    COpArray::~COpArray()
```



```
24  {
25  delete [] _myArray;
26 }
27 int main(int argc, char *argv[])
28 {
29 QCoreApplication a(argc, argv);
30 COpArray arrOp(5);
31 arrOp[0]=23;
32 arrOp[1]=16;
33 arrOp[2]=9;
34 arrOp[3]=20;
35 arrOp[4]=55;
36 cout << "The content array are : " << "{";
37 for(int i=0;i<5;++i)
38 {
39 cout << arrOp[i] << " ";
40 }
41 cout << "}" << endl;
42 return a.exec();
43 }
```

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

```
The content array are : {23 16 9 20 55 }
```

Keterangan

Dengan menggunakan subscript operator anda dapat mengakses member variable yang bertipe array pada class dengan menggunakan array-like syntax (nama objek diikuti dengan tanda []), misal: namaObjek[index].



8.7 Function operator()

Function operator digunakan jika anda ingin membuat objek bekerja seperti function. Untuk lebih jelasnya penggunaan function `operator()` anda dapat mencoba program dibawah ini.

Contoh Menggunakan operator() untuk membuat function object.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama contoh 8.9, kemudian tulis kode berikut.

Listing 8.9: Menggunakan operator() untuk membuat function object

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class CTampil
5 {
6 public:
7     void operator()(string msg) const
8     {
9         cout << msg << endl;
10    }
11 };
12 int main(int argc, char *argv[])
13 {
14     QCoreApplication a(argc, argv);
15     CTampil objTampil;
16     //penulisan ekuivalen dengan objTampil.operator_()
17     //("Hello Function Operator !");
18     objTampil("Hello Function Operator !");
19     return a.exec();
20 }
```



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

```
Hello Function Operator !
```

Analisis

- Dengan membuat function operator maka anda dapat menggunakan objek seperti ketika anda menggunakan function.
- Pada program diatas objek objTampil dapat dipanggil seperti function.



BAB 9

Polymorphism

Agenda

Pada chapter ini kita akan membahas beberapa topik tentang teknis polimorfisme, adapun topik yang akan dibahas adalah

Contents

| | | |
|------|---|-----|
| 9.1 | Problema Pewarisan Tunggal (Single Inheritance) | 287 |
| 9.2 | Peletakan ke atas (Pecolating Upward) | 292 |
| 9.3 | Konversi ke bawah (Casting Down) | 293 |
| 9.4 | Menambahkan ke Dua Daftar | 296 |
| 9.5 | Pewarisan Ganda (Multiple Inheritance) | 297 |
| 9.6 | Komponen Objek Multi Inheritance | 302 |
| 9.7 | Konstruktor Kelas Banyak Turunan (Multiple Inheritance) | 303 |
| 9.8 | Problem Ambiguitas | 305 |
| 9.9 | Penurunan dari Kelas Dasar Bersama | 306 |
| 9.10 | Penurunan Virtual (Virtual Inheritance) | 311 |



9.11 Masalah Pada Multiple Inheritance 314

Secara teknis polimorfisme merupakan suatu konsep untuk merelasikan diantara kelas-kelas C++ melalui overriding metode-metode virtual, sehingga dengan demikian satu tipe kelas dapat konversikan menjadi kelas lain. Aspek penting pertama dalam Pewarisan (*Inheritance*) adalah pengorganisasian kelas yang mengijinkan kelas lain berbagi program dan data (*code reuse*) sehingga program tidak harus dibangun ulang dari awal. Aspek penting kedua dari Pewarisan (*Inheritance*) adalah pengelompokan fitur-fitur kelas yang serupa kedalam sebuah kelas dasar (*base class*) dan kemudian membuat kelas lain dengan cara menurunkan kelas tersebut sehingga bentuk utama/ pokok dari kelas-kelas turunan menjadi serupa dengan kelas dasar sedemikian rupa sehingga pointer atau referensi bertipe kelas dasar dapat menerima nilai berbagai macam bentuk objek bertipe kelas turunannya (berubah tipe kelas) dan dapat mengeksekusi fitur-fitur yang serupa tersebut. Inilah yang disebut dengan polimorfisme.

Berbeda dengan tipe data standard (int, float, double, bool, dsb.) yang jenisnya terbatas dan sudah pasti dikenal dalam pemrograman C++, kelas adalah tipe data terstruktur yang bisa dibuat sendiri oleh programer, sehingga tidak terbatas ada berapa macam kelas dalam C++. Namun di sisi lain, dalam pemrograman kita pasti perlu untuk dapat berhubungan antara kelas satu dengan yang lain, sehingga diperlukan suatu kerangka yang dapat memberikan arahan mengenai bentuk dan fitur dari suatu kelas sehingga dengan demikian ada suatu pointer atau referensi yang dapat menerima berbagai macam bentuk yang dinamakan polimorfisme.

Mekanisme untuk dapat terjadi polimorfisme adalah pewarisan (*inheritance*), oleh karena itu polimorfisme hanya bisa terjadi diantara kelas-kelas yang mempunyai hubungan kekerabatan, tepatnya pointer atau referensi bertipe kelas dasar hanya dapat menerima berbagai macam bentuk objek yang bertipe kelas-kelas turunannya, sehingga pointer atau referensi tersebut dapat mengeksekusi fitur-fitur yang serupa dengannya. Contoh sederhana dari polimorfisme adalah seperti sudah dibahas pada bab 7 yaitu mengenai metode virtual pada percobaan contoh 7.10.



Berikut ini akan dibahas secara lebih mendalam mengenai berbagai aspek polimorfisme dari problem pada `single inheritance`, `multiple inheritance` hingga `abstract data type`.

9.1 Problema Pewarisan Tunggal (Single Inheritance)

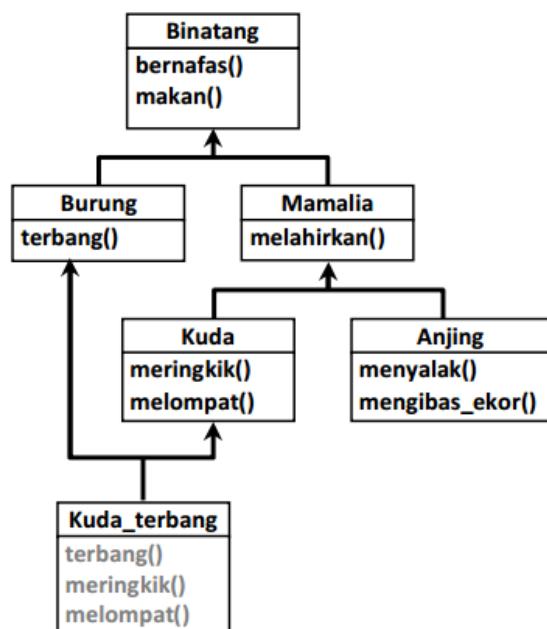
Supaya terlihat sederhana, untuk menjelaskan masalah ini marilah kita menggunakan ilustrasi mengenai kelas-kelas dengan perumpamaan kelas binatang dan turunan-turunanya. Misalnya kelas Binatang mempunai hierarki keturuan **Mamalia** dan **Burung**, kelas Burung mempunyai member function `terbang()`, sedangkan kelas Mamalia sudah diturunkan menjadi beberapa kelas diantaranya Kuda dan Anjing. Kelas Kuda mempunyai member function `meringkik()` dan `melompat()`. Suatu saat terpikir untuk membuat kelas `Kuda_terbang` yang merupakan kombinasi antara kelas Burung dan kelas Kuda, karena Kuda `terbang` bisa `terbang()`, `meringkik()` dan `melompat()`.

Dengan single inheritance hal ini tidak bisa dilakukan dengan mudah, karena hanya bisa menurunkan dari salah satu kelas yang sudah ada, kita bisa membuat `Kuda_terbang` adalah turunan Burung yang bisa `tebang()` tetapi akibatnya tidak bisa `meringkik()` dan `melompat()`, sebaliknya jika dibuat sebagai turunan dari Kuda akan bisa `meringkik()` dan `melompat()` tetapi tidak bisa `terbang()`.

Untuk memaksakan hal ini bisa dilakukan dengan membuat metode `terbang()` di dalam kelas `Kuda_terbang` dan kelas ini diturunkan dari kelas Kuda. Ini akan dapat dilakukan, akan tetapi harga yang harus dibayar sekarang adalah kita mempunyai metode `terbang()` dalam dua kelas yang berbeda (Burung dan `Kuda_terbang`), jika ada perubahan di salah satu metode `terbang()`, maka harus selalu diingat untuk merubah yang lainnya, ini sangat berisiko akan adanya ketidakkonsistenan program.

Belum lagi nanti akan timbul masalah *polimorfisme* ketika akan dibuat daftar objek Kuda atau daftar objek Burung, kalau `Kuda_terbang` dapat dima-





Gambar 9.1: Problema Pewarisan Tunggal (Single Inheritance)

sukkan ke dalam dafatar Kuda maka ia tidak akan bisa dimasukkan kedalam dafatar Burung. Akibatnya akan timbul ide untuk mengubah metode `melompat()` pada Kuda menjadi `berpindah()` kemudian melakukan override dalam kelas `Kuda_terbang` yang melakukan pekerjaan `terbang()` dan pada turunan Kuda lainnya melakukan override `berpindah()` yang melakukan pekerjaan `melompat()`. Kemudian karena seharusnya `Kuda_terbang` tetap dapat melompat kita membatasi jika jarak dekat melompat jika jarak jauh terbang seperti berikut:

```
Kuda_terbang::berpindah(long jarak)
{
    if (jarak > sangat_jauh)
        terbang(jarak);
    else
        melompat(jarak);
}
```

Tapi ini menjadikan terbatas, bagaimana jika nanti ternyata `Kuda_terbang` bisa `melompat()` lebih jauh atau `terbang()` lebih dekat? Ini akan menjadi masalah.

Solusi lainnya untuk membahas keterbatasan *single inheritance* ini adalah membuat metode `terbang()` pada kelas Kuda dan pada kelas ini tidak melakukan `terbang()`, baru nanti kalau berupa objek `Kuda_terbang`, barulah `terbang()` yang sesungguhnya dikerjakan. Marilah kita melakukan percobaan berikut ini.

Contoh Meletakkan metode kelas turunan di kelas dasar.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 9.1, kemudian tulis kode berikut.

Listing 9.1: Meletakkan metode kelas turunan di kelas dasar

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



```
2 #include <iostream>
3 using namespace std;
4 class Kuda{
5 public:
6 void melompat(){ cout << "Lompat!" << endl;}
7 virtual void terbang(){cout <<"kuda tidak bisa
8     terbang" << endl;}
9 class Kuda_terbang : public Kuda{
10 public:
11 void terbang() {cout << "terbang..." << endl;}
12 }
13 int main(int argc, char *argv[])
14 {
15 QCoreApplication a(argc, argv);
16 Kuda* kandang[5];
17 Kuda* kudanya;
18 int pilih;
19 for(int nomor=0; nomor<5; nomor++){
20 cout << "Pilih (0) Kuda atau (1) Kuda terbang : ";
21 cin >> pilih;
22 if(pilih==0)
23 kudanya = new Kuda();
24 else
25 kudanya = new Kuda_terbang();
26 kandang[nomor] = kudanya;
27 }
28 cout << endl;
29 for(int nomor=0; nomor<5; nomor++){
30 kandang[nomor]->terbang();
31 delete kandang[nomor];
32 }
33 return a.exec();
34 }
```



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

```
Pilih (0) Kuda atau (1) Kuda terbang : 0
Pilih (0) Kuda atau (1) Kuda terbang : 0
Pilih (0) Kuda atau (1) Kuda terbang : 1
Pilih (0) Kuda atau (1) Kuda terbang : 1
Pilih (0) Kuda atau (1) Kuda terbang : 0

kuda tidak bisa terbang
kuda tidak bisa terbang
terbang...
terbang...
kuda tidak bisa terbang
```

Analisa Program :

- Pada program diatas ada dua kelas, yaitu Kuda sebagai kelas dasar dan Kuda_terbang sebagai kelas turunan, Kuda mempunyai metode terbang() yang sebenarnya diada-adakan supaya bisa terjadi proses polimorfisme dengan baik, akibatnya metode terbang() ini tidak melakukan terbang yang sesungguhnya. Kelas Kuda_terbang melakukan override terhadap metode virutual terbang(), pada metode ini melakukan terbang yang sesungguhnya.
- Pada program utama disediakan program untuk menentukan 5 jenis kuda, yang ditampung dalam pointer kudanya. Pointer ini bisa berpolimorfisme kaena memenuhi syarat bahwa dia bertipe kelas dasar (Kuda) dan memanggil metode virutual terbang().
- Pada percobaan eksekusi di atas dipilih 0, 0, 1, 1, 0 yang berarti array kandang berisi :

```
Kandang[0] <-- berisi objek Kuda
Kandang[1] <-- berisi objek Kuda
Kandang[2] <-- berisi objek Kuda_terbang
```



```
Kandang[3] <-- berisi objek Kuda_terbang  
Kandang[4] <-- berisi objek Kuda
```

Sehingga hasil keluaran berikutnya:

```
kuda tidak bisa terbang  
kuda tidak bisa terbang  
terbang...  
terbang...  
kuda tidak bisa terbang
```

- Dari percobaan ini tampak *polimorfisme* bisa berhasil dengan baik. Namun seperti sudah dijelaskan di atas, harga yang harus dibayar adalah sekarang ada metode `terbang()` pada kelas `Kuda_terbang` dan `Burung`, sehingga jika ada perubahan di salah satu metode `terbang()`, maka harus selalu diingat untuk merubah yang lainnya, ini sangat berisiko akan adanya ketidakkonsistenan program.



Catatan:

Pada percobaan ini kelas telah disederhanakan untuk fokus pada pokok masalah yang hendak dijelaskan. Konstruktor, destruktur dan sebagainya dihilangkan supaya program tampak sederhana untuk membahas masalah *single inheritance*. Tidak direkomendasikan untuk menulis program yang seperti ini.

9.2 Peletakan ke atas (Pecolating Upward)

Meletakkan fungsi yang diperlukan pada kelas yang berada pada hierarki di atasnya, seperti percobaan di atas, adalah solusi yang biasa digunakan untuk keperluan polimorfisme dan juga dalam hal ini mengatasi single inheritance dan beraikibat menghasilkan banyak fungsi-fungsi “yang diletakkan di atas” (“Percolating up”) ke dalam kelas dasar. Kemudian kelas dasar tersebut akan menjadi sangat berbahaya karena menjadi global namespace untuk semua fungsi yang mungkin



diperlukan kelas turunan yang berpotensi merusak tatanan tipe kelas C++ dan menciptakan kelas dasar yang berukuran besar dan tidak efisien.

Sebenarnya kejadian ini terjadi karena keinginan untuk menaruh fungsionalitas bersama ke hierarki di atasnya tanpa mengubah antarmuka (interface) dari tiap kelas. Artinya jika ada dua kelas yang memakai kelas dasar yang sama (misalnya Burung dan Kuda sama-sama bersal dari kelas dasar Binatang) dan keduanya sama-sama mempunyai sebuah fungsi yang sama (misalnya Burung dan Kuda sama-sama bisa makan), maka akan timbul ide untuk memindahkan fungsi tersebut ke hierarki di atasnya dan membuat metode virtual pada kelas dasar.

9.3 Konversi ke bawah (Casting Down)

Alternatif lain untuk mengatasi single inheritance adalah tetap membuat metode `terbang()` dalam kelas `Kuda_terbang` dan metode ini hanya dipanggil jika pointer menunjuk objek bertipe `Kuda_terbang`. Untuk keperluan ini diperlukan untuk dapat mendeteksi objek tipe apa yang sedang ditunjuk oleh pointer yang dikenal sebagai Runtime Type Identification (RTTI).



Catatan:

Hati-hati dalam menggunakan RTTI dalam program. Kebutuhan untuk memakai RTTI bisa menjadi pertanda adanya desain hierarki pewarisan yang kurang baik. Untuk itu sebaiknya gunakan metode virtual, template atau multiple inheritance daripada memakai RTTI.

Pada percobaan Contoh 9.1 di atas, kita menunjuk baik Kuda maupun `Kuda_terbang` dengan array Kuda (kandang). Semuanya dimasukkan sebagai Kuda. Dengan RTTI, kita akan memeriksa tiap-tiap elemen array apakah objek yang ditunjuk berupa Kuda atau sebenarnya `Kuda_terbang`.

Pada percobaan ini kita tidak melakukan “percolating upward”, yaitu menuliskan metode `terbang()` ke dalam kelas Kuda melainkan melakukan “do-



wn casting” untuk memanggil metode `terbang()`. Untuk memanggil metode `terbang()` tersebut harus dipastikan bahwa pointer sedang menunjuk objek bertipe Kuda_terbang, bukan Kuda. C++ mendukung “down casting” (RTTI) memakai operator `dynamic_cast`. Cara kerja `dynamic_cast` adalah demikian, jika kita memiliki pointer bertipe kelas dasar seperti Kuda dan digunakan untuk menunjuk objek bertipe kelas turunan misalnya Kuda_Terbang, maka pointer tersebut bisa langsung dipergunakan secara polimorfisme. Kemudian jika kita ingin mengambil objek Kuda_terbang yang sudah ditunjuk oleh pointer bertipe Kuda itu dapat dilakukan dengan cara membuat pointer bertipe Kuda_terbang kemudian gunakan operator `dynamic_cast` untuk mengkonversikannya. Pada saat eksekusi (runtime), pointer dasar akan diperiksa, jika sesuai maka pointer Kuda_terbang bekerja dengan baik, jika tidak sesuai maka sebenarnya bukan objek Kuda_terbang yang ditunjuk melainkan pointer kosong (null). Lakukan percobaan berikut ini.

Contoh Melakukan Down Casting.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 9.2, kemudian tulis kode berikut.

Listing 9.2: Melakukan Down Casting

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class Kuda{
5 public:
6     virtual void melompat(){ cout << "Lompat!" << endl
7         ;}
8 };
9 class Kuda_terbang : public Kuda{
10 public:
11     virtual void terbang() {cout << "terbang..." <<
12         endl;}
```



```
11  };
12 int main(int argc, char *argv[])
13 {
14 QCOREAPPLICATION a(argc, argv);
15 Kuda * kandang[5];
16 Kuda * kudanya;
17 int pilih;
18 for(int nomor=0; nomor<5; nomor++){
19 cout << "Pilih (0) Kuda atau (1) Kuda terbang : ";
20 cin >> pilih;
21 if(pilih==0)
22 kudanya = new Kuda();
23 else
24 kudanya = new Kuda_terbang();
25 kandang[nomor] = kudanya;
26 }
27 cout << endl;
28 for(int nomor=0; nomor<5; nomor++){
29 Kuda_terbang * pKterb = dynamic_cast<Kuda_terbang*>
(kandang[nomor]);
30 if (pKterb != NULL)
31 pKterb->terbang();
32 else
33 cout << "Kuda biasa " << endl;
34 delete kandang[nomor];
35 }
36 return a.exec();
37 }
```

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



```
Pilih (0) Kuda atau (1) Kuda terbang : 1  
Pilih (0) Kuda atau (1) Kuda terbang : 0  
Pilih (0) Kuda atau (1) Kuda terbang : 1  
Pilih (0) Kuda atau (1) Kuda terbang : 0  
Pilih (0) Kuda atau (1) Kuda terbang : 1
```

terbang...

Kuda biasa

terbang...

Kuda biasa

Analisa Program :

- Jalan keluar single inheritance ini berjalan dengan baik, namun tidak di-rekomendasikan untuk memakai cara ini. Hasilnya hanya berupa tambal-sulam saja, sebenarnya metode `terbang()` tidak ada di kelas Kuda dan tidak dipanggil dari sana. Ketika dipanggil dengan pointer `Kuda_terbang` dilakukan casting secara eksplisit, maka harus dipastikan bahwa pointer `pKterb` benar-bena berisi objek bertipe `Kuda_terbang` sebelum memanggil metode `terbang()`.
- Pemakain down casting semacam ini merupakan pertanda bahwa desain yang dibuat kurang baik, program semacam ini merusak fungsi *virtual polimorfisme* karena ini dilakukan casting objek menjadi tipe yang sesungguhnya (bukan polimorfisme).

9.4 Menambahkan ke Dua Daftar

Masalah lain yang dihadapi oleh karena memakai solusi di atas adalah bahwa kita telah mendeklarasikan `Kuda_terbang` bertipe `Kuda`, sehingga kita tidak bisa menambahkan objek `Kuda_terbang` ke dalam daftar objek `Burung`. Dalam hal ini bisa saja kita melakukan pemindahan metode `terbang()` ke hierarki atasnya yaitu kedalam kelas `Kuda` atau melakukan down casting pada pointer, tetapi tetap saja kita tidak mendapatkan fungsionalitasnya secara penuh.



Satu solusi terakhir untuk mengatasi masalah single inheritance ini adalah memindahkemana semua fungsi `terbang()`, `meringkik()` dan `melompat()` ke kelas dasar dari Burung dan Kuda, yaitu kelas Binatang. Dengan demikian kita bisa mendaftar objek-objek Burung, Kuda maupun objek-objek `Kuda_terbang` dalam satu kesatuan daftar Binatang. Namun akibatnya kelas dasar mempunyai semua karakteristik kelas turunannya sehingga ukuran kelas dasar menjadi sangat besar dan ini tidak diinginkan.

9.5 Pewarisan Ganda (Multiple Inheritance)

Dengan C++ dimungkinkan untuk menurunkan kelas baru yang berasal dari lebih dari satu kelas dasar, yang dinamakan pewarisan ganda (multiple inheritance). Penurunan lebih dari satu kelas dasar dilakukan dengan cara menuliskan kelas dasar berikutnya dipisahkan dengan tanda koma (,) seperti berikut:

```
class KelasTurunan : public KelasDasar1, public  
KelasDasar2{}
```

Percobaan berikut ini memberikan ilustrasi cara deklarasi kelas `Kuda_terbang` yang mewarisi kelas dasar Kuda dan Burung, kemudian program menambahkan objek `Kuda_terbang` ini di kedua jenis daftar objek tersebut.

Contoh Multiple Inheritance.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 9.3, kemudian tulis kode berikut.

Listing 9.3: Multiple Inheritance

```
1 #include <QtCore/QCoreApplication>  
2 #include <iostream>  
3 using namespace std;  
4 class Kuda{  
5 public:
```



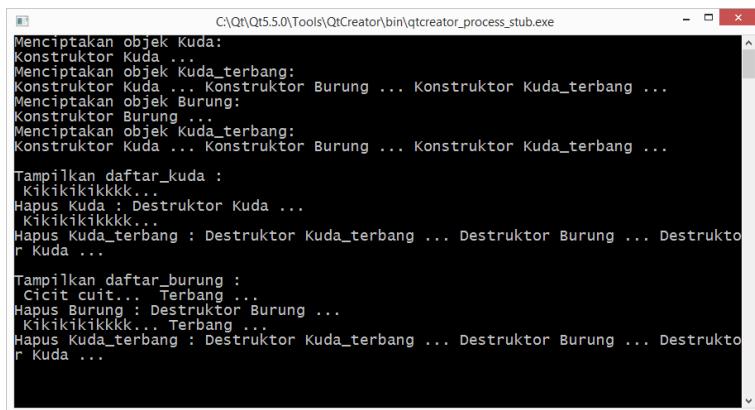
```
6 Kuda() {cout << "Konstruktor Kuda . . . ";}
7 virtual ~Kuda() {cout << "Destruktor Kuda . . . \n"
8 ;}
9 virtual void meringkik() const {cout << "
10 Kikikikikkkk...";}
11 };
12 class Burung{
13 public:
14 Burung() { cout << "Konstruktor Burung . . . ";}
15 virtual ~Burung() { cout << "Destruktor Burung . . .
16 "; }
17 virtual void berkicau() const { cout << " Cicit
18 cuit...";}
19 virtual void terbang() const { cout << " Terbang
20 . . . ";}
21 };
22 class Kuda_terbang : public Kuda, public Burung{
23 public:
24 void berkicau() const { meringkik();}
25 Kuda_terbang() { cout << "Konstruktor Kuda_terbang
26 . . . ";}
27 ~Kuda_terbang() { cout << "Destruktor Kuda_terbang
28 . . . ";}
29 };
30 int main(int argc, char *argv[])
31 {
32 QApplication a(argc, argv);
33 Kuda* daftar_kuda[2]; //<-- kumpulan Kuda
34 Burung* daftar_burung[2]; //<-- kumpulan Burung
35 cout << "Menciptakan objek Kuda:" << endl;
36 daftar_kuda[0] = new Kuda(); //<-- objek Kuda
37 cout << "\nMenciptakan objek Kuda_terbang:" <<
38 endl;
```



```
31 daftar_kuda[1] = new Kuda_terbang(); //<-- objek
     Kuda_terbang
32 cout << "\nMenciptakan objek Burung:" << endl;
33 daftar_burung[0] = new Burung(); //<-- objek
     Burung
34 cout << "\nMenciptakan objek Kuda_terbang:" <<
     endl;
35 daftar_burung[1] = new Kuda_terbang(); //<-- objek
     Kuda_terbang
36 cout << "\n\nTampilkan daftar_kuda :" << endl;
37 daftar_kuda[0]->meringkik(); //<-- berisi objek
     Kuda
38 cout << "\nHapus Kuda : ";
39 delete daftar_kuda[0];
40 daftar_kuda[1]->meringkik(); //<-- berisi objek
     Kuda_terbang
41 cout << "\nHapus Kuda_terbang : ";
42 delete daftar_kuda[1];
43 cout << "\nTampilkan daftar_burung :" << endl;
44 daftar_burung[0]->berkicau(); //<-- berisi objek
     Burung
45 daftar_burung[0]->terbang(); //<-- berisi objek
     Burung
46 cout << "\nHapus Burung : ";
47 delete daftar_burung[0];
48 cout << endl;
49 daftar_burung[1]->berkicau(); //<-- berisi objek
     Kuda_terbang
50 daftar_burung[1]->terbang(); //<-- berisi objek
     Kuda_terbang
51 cout << "\nHapus Kuda_terbang : ";
52 delete daftar_burung[1];
53 return a.exec();
54 }
```



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



```
Menciptakan objek Kuda: Konstruktor Kuda ...
Menciptakan objek Kuda_terbang: Konstruktor Kuda ... Konstruktor Burung ... Konstruktor Kuda_terbang ...
Menciptakan objek Burung: Konstruktor Burung ...
Menciptakan objek Kuda_terbang: Konstruktor Kuda ... Konstruktor Burung ... Konstruktor Kuda_terbang ...

Tampilkan daftar_kuda :
Kikikikiikkk...
Hapus Kuda : Destruktor Kuda ...
Kikikikiikkk...
Hapus Kuda_terbang : Destruktor Kuda_terbang ... Destruktor Burung ... Destruktor Kuda ...

Tampilkan daftar_burung :
Cicit cuit... Terbang ...
Hapus Burung : Destruktor Burung ...
Kikikikiikkk... Terbang ...
Hapus Kuda_terbang : Destruktor Kuda_terbang ... Destruktor Burung ... Destruktor Kuda ...
```

Analisa Program :

- Pada kelas Kuda_terbang tampak penggunaan multiple inheritance, yaitu pada deklarasikelas Kuda_terbang yang merupakan keturunan dari Kuda dan Burung :

```
class Kuda_terbang : public Kuda, public Burung
```

- Kemudian kelas ini melakukan override terhadap metode berkicau(), metode berkicau() pada Kuda_terbang ini mengerjakan pemanggilan metode meringkik(), yaitu metode warisan dari kelas Kuda:

```
void berkicau() const { meringkik(); }
```

- Tampak pada percobaan Contoh 9.3 ini ketika diciptakan objek bertipe Kuda maka yang bekerja adalah konstruktor Kuda, demikian juga ketika diciptakan objek Burung yang bekerja adalah konstruktor Burung, namun ketika diciptakan Kuda_terbang yang bekerja tiga konstruktor sekaligus, yaitu : konstruktor Kuda, konstruktor Burung dan konstruktor Kuda_terbang. Ini memperlihatkan bahwa kelas Kuda_terbang



merupakan turunan dari kelas Kuda sekaligus turunan kelas Burung. Dengan kata lain objek Kuda_terbang adalah objek yang di dalamnya terkandung bagian objek Kuda dan bagian objek Burung.

- Kebalikannya saat dihapus, destruktor yang dijalankan : destruktor Kuda_terbang, destruktor Burung baru kemudian destruktor Kuda. Ini adalah akibat adanya destruktor yang selalu virtual.
- Pada saat ditampilkan `daftar_kuda`, tampak pada program sebenarnya yang pertama berisi objek Kuda sedangkan yang kedua berisi objek Kuda_terbang :

```
daftar_kuda[0] = new Kuda(); //<-- objek Kuda  
daftar_kuda[1] = new Kuda_terbang(); //<-- objek  
Kuda_terbang
```

- Demikian juga pada `daftar_burung`, yang pertama berisi objek Burung sedangkan yang kedua berisi Kuda_terbang :

```
daftar_burung[0] = new Burung(); //<-- objek  
Burung  
daftar_burung[1] = new Kuda_terbang(); //<-- objek  
Kuda_terbang
```

- Ini menunjukkan bahwa baik `daftar_kuda[]` maupun `daftar_burung[]` dapat berpolimorfisme dengan sempurna berkat adanya multiple inheritance.
- Pada waktu `daftar_burung` berisi objek Kuda_terbang, ketika dipanggil metode berkicau berikut :

```
daftar_burung[1]->berkicau(); //<-- berisi objek  
Kuda_terbang
```

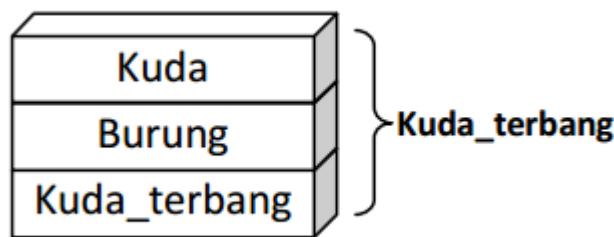
- Maka yang menanggapi adalah metode hasil override di dalam kelas Kuda_terbang yaitu pemanggilan metode `meringkik()`, sehingga keluarannya adalah :



" Kikikikikk... "

9.6 Komponen Objek Multi Inheritance

Ketika objek Kuda_terbang diciptakan di memori, kedua kelas dasarnya juga tercipta sebagai bagian pembentuk objek tersebut. Gambar di bawah ini menggambarkan sebuah objek bertipe Kuda_terbang termasuk fitur-fitur baru yang ditambahkan pada kelas Kuda_terbang maupun fitur-fitur warisan kelaskelas dasarnya.



Gambar 9.2: Komponen Objek Multi Inheritance

Ada beberapa hal yang perlu diperhatikan pada objek yang merupakan turunan dari beberapa kelas dasar. Sebagai contoh misalnya, apa yang terjadi jika kedua kelas dasar tersebut memiliki metode virtual atau data yang sama? Bagaimana cara menginisialisasi konstruktor kelas dasarnya? Bagaimana jika kedua kelas dasar yang diturunkan merupakan keturunan dari suatu kelas dasar yang sama? Berikut ini akan kita bicarakan mengenai isu-isu tersebut.



9.7 Konstruktor Kelas Banyak Turunan (Multiple Inheritance)

Seperti pada umumnya, kelas turunan pasti memanggil konstruktor kelas dasarnya, demikian juga konstruktor kelas banyak turunan (multiple inheritance) juga harus memanggil konstruktor-konstruktor kelas dasarnya. Jika bentuk konstruktor default (konstruktor yang diciptakan secara otomatis oleh kompiler jika kelas turunan tidak menuliskan konstruktor secara eksplisit) pada single inheritance yaitu:

```
KelasTurunan():KelasDasar(){} //<-- Konstruktor  
default kelas Turunan tunggal
```

Maka konstruktor default kelas banyak turunan (mutiple inheritance) adalah :

```
KelasTurunan():KelasDasar1(),KelasDasar2(){}  
//<-- Konstruktor default kelas Turunan
```

Jadi pada waktu membuat kelas banyak turunan, khususnya jika kelas dasarnya tidak mempunyai konstruktor tanpa parameter, maka konstruktor kelas turunan tersebut harus memanggil salah satu dari konstruktor kelas-kelas dasarnya, karena kalau tidak kompiler akan memanggil konstruktor default kelas dasar yang sebenarnya tidak ada sehingga menimbulkan kesalahan kompilasi.

Supaya bisa fokus pada pokok permasalahan, yaitu konstruktor, percobaan berikut ini menghilangkan berbagai macam anggota yang lain supaya terlihat sederhana dan mudah dipahami.

Contoh Konstruktor Kelas Multiple Inheritance.

Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 9.4, kemudian tulis kode berikut.

Listing 9.4: Konstruktor Kelas Multiple Inheritance

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



```
2 #include <iostream>
3 using namespace std;
4 class Kuda{
5 public:
6     Kuda(string nama){
7         cout << "Konstruktor Kuda, nama = " << nama << endl;
8     }
9 };
10 class Burung{
11 public:
12     Burung(string warna){
13         cout << "Konstruktor Burung, warna = " << warna <<
14             endl;
15     }
16 };
17 class Kuda_terbang : public Kuda, public Burung{
18 public:
19     Kuda_terbang():Kuda("Gondrong"),Burung("Merah"){
20         cout << "Konstruktor Kuda_terbang";
21     }
22     int main(int argc, char *argv[])
23     {
24         QApplication a(argc, argv);
25         Kuda_terbang* gondrong = new Kuda_terbang();
26         return a.exec();
27     }
```

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

```
Konstruktor Kuda, nama = Gondrong
Konstruktor Burung, warna = Merah
```



Analisa Program :

- Kelas Kuda hanya mempunyai sebuah konstruktor dengan sebuah parameter bertipe `string`, demikian juga kelas Burung hanya mempunyai sebuah konstruktor dengan sebuah parameter bertipe `string`.
- Pada kelas `Kuda_terbang` tampak penggunaan multiple inheritance, yaitu pada deklarasi kelas `Kuda_terbang` yang merupakan keturunan dari Kuda dan Burung :

```
class Kuda_terbang : public Kuda, public Burung
```

- Oleh karena itu konstruktor kelas `Kuda_terbang` ini harus memanggil secara eksplisit konstruktor-konstruktor kelas dasarnya seperti berikut:
`Kuda_terbang() :Kuda("Gondrong"),Burung("Merah")`
- Tampak pada hasil output, instansiasi kelas `Kuda_terbang` menjalankan konstruktor Kuda dengan satu parameter dan konstruktor Burung dengan satu parameter, baru kemudian menjalankan konstruktornya sendiri.

9.8 Problem Ambiguitas

Pada kelas multi inheritance yang beberapa kelas dasarnya mempunyai metode virtual yang sama akan menimbulkan masalah ketika objek kelas turunan akan memanggil metode tersebut, sebab kelas tersebut mendapatkan warisan dari keduaanya, sehingga ketika metode tersebut dipanggil akan menimbulkan masalah ambiguitas bagi kompiler.

Sebagai contoh misalnya kelas Kuda dan kelas Burung sama-sama mempunyai metode `virtual makan()` dan jika kita memanggil metode dari objek `Kuda_terbang` yang merupakan multi inheritance dari kedua kelas tersebut di atas.

```
pKterbang->makan();
```



Maka kompile akan menampilkan pesan kesalahan :

Member is ambiguous: 'Kuda::makan; and 'Burung::makan'

Untuk memecahkan masalah ini dengan pemanggilan secara eksplisit bisa dilakukan dengan cara menyebutkan nama kelas dasar pemilik metode virtual yang akan dieksekusi, misalnya:

```
pKterbang->Kuda::makan();
```

Kapan saja kita ingin mengeksekusi fungsi anggota atau variabel anggota kelas mana yang akan diakses, kita dapat melakukan dengan cara tersebut di atas. Jika kelas Kuda_terbang akan melakukan overriding fungsi tersebut, maka pemanggilan tersebut bisa dilakukan di dalam fungsi anggota Kuda_terbang:

```
virtual void makan() const { Kuda::makan() }
```

Dengan cara ini kita bisa menyembunyikan masalah ambiguitas ini dari pemakai kelas Kuda_terbang dengan cara mengenkapsulasi di dalam kelas Kuda_terbang mengenai kelas dasar mana yang diambil warisan fungsi makan()-nya. Namun pemakai tetap saja bebas untuk dapat memanggil waisan lainnya dengan menulis :

```
pKterbang->Burung::makan();
```

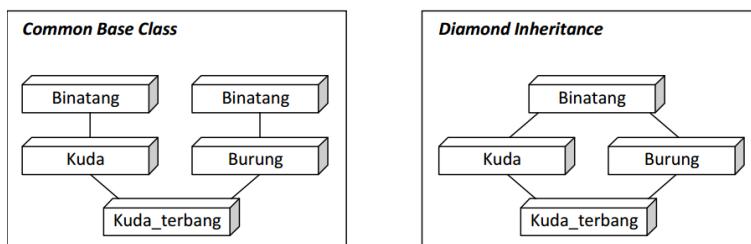
9.9 Penurunan dari Kelas Dasar Bersama

Jika dua kelas dasar yang dijadikan multiple inheritance merupakan turunan dari kelas dasar yang sama, maka ketika diciptakan objek dari kelas turunan akan bisa terbentuk objek dengan 2 kemungkinan, yaitu:

1. Tercipta 2 objek kelas dasar (**Common Base Class**)
2. Tercipta 1 Objek kelas dasar (**Diamond Inheritance**)



Sebagai ilustrasi seperti contoh sebelumnya, Kuda dan Burung merupakan turunan dari sebuah kelas dasar yang sama yaitu kelas Binatang, sedangkan Kuda_terbang merupakan hasil *multiple inheritance* dari kelas Kuda dan Burung. Pada saat *instansiasi* kelas Kuda_terbang terdapat 2 kemungkinan bentuk kelas yang terjadi:



Gambar 9.3: Penurunan dari class dasar yang sama

Common base class adalah penurunan seperti biasa yang terjadi, yaitu ketika kelas paling bawah (Kuda_terbang) melakukan instansiasi, secara otomatis kelas-kelas dasarnya (Kuda dan Burung) juga terbentuk, dan ketika kedua kelas dasar tersebut terbentuk (Kuda dan Burung), masing-masing memanggil secara terpisah konstruktor kelas dasar paling atas (Binatang), sehingga terbentuklah objek kelas dasar paling atas untuk masing-masing (sendiri-sendiri). Sedangkan Diamond inheritance terjadi ketika dilakukan penurunan secara virtual, yaitu konstruktor kelas turunan paling bawah (Kuda_terbang) memanggil secara langsung konstruktor kelas dasar paling atas (Binatang), sehingga pemanggilan oleh kedua kelas dasarnya (Kuda dan Burung) terhadap konstruktor kelas dasar paling atas (Binatang) diabaikan. Hal ini bisa terjadi jika penurunan dilakukan secara virtual yang akan dibahas nanti.

Untuk penurunan pada umumnya (Common base class) ada kemungkinan kelas Kuda maupun Burung masing-masing sudah melakukan overriding terhadap metode milik kelas dasarnya (Binatang). Sebagai contoh misalnya kelas Binatang mempunyai member variabel umur dan member function getUmur(). Jika kelas turunan paling bawah (Kuda_terbang) memanggil metode getUmur() tersebut, maka akan timbul ambiguitas lagi, yaitu metode yang dipanggil me-



rupakan getUmur() Kuda atau getUmur() Burung? Pemecahan problem ini sama dengan pada kelas multi inheritance yang yang kedua kelas dasarnya mempunyai metode virtual yang sama, yaitu bisa dilakukan dengan cara melakukan override terhadap metode getUmur() kemudian di dalamnya memilih metode yang akan merespon, misalnya dipilih getUmur() dari Kuda seperti berikut:

```
virtual int getUmur() const { return Kuda::getUmur(); }
```

Agar lebih jelas, lakukan pecobaan berikut ini.

Contoh Penurunan pada umumnya (Common Base Class).

Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 9.5, kemudian tulis kode berikut.

Listing 9.5: Penurunan pada umumnya (Common Base Class)

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class Binatang{
5 public:
6     Binatang(int umur=5):umur(5){cout << "Konstruktor
    Binatang\n";}
7     ~Binatang(){cout << "Destruktor Binatang\n";}
8     protected:
9     int umur;
10    public:
11    virtual int const getUmur(){
12        cout << "dari Binatang..." ;
13        return umur;
14    }
15    };
16    class Kuda : public Binatang{
17    public:
```



```
18 Kuda(){cout << "Konstruktor Kuda\n";}
19 ~Kuda(){cout << "Destruktor Kuda\n";}
20 virtual int const getUmur(){
21 cout << "dari Kuda...";
22 return umur;
23 }
24
25 };
26 class Burung : public Binatang{
27 public:
28 Burung(){cout << "Konstruktor Burung\n";}
29 ~Burung(){cout << "Destruktor Burung\n";}
30 virtual int const getUmur(){
31 cout << "dari Burung...";
32 return umur;
33 }
34 };
35 class Kuda_terbang : public Kuda, public Burung{
36 public:
37 Kuda_terbang(){cout << "Konstruktor Kuda_terbang\n";}
38 ~Kuda_terbang(){cout << "Destruktor Kuda_terbang\n";}
39 virtual int const getUmur(){
40 cout << "dari Kuda_terbang...";
41 return Kuda::getUmur();
42 }
43 };
44 int main(int argc, char *argv[])
45 {
46 QApplication a(argc, argv);
47 Kuda_terbang* gondrong = new Kuda_terbang();
48 cout << "Umur : " << gondrong->getUmur() << endl;
49 delete gondrong;
50 return a.exec();
51 }
```



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

A> {linenos=off} A> Konstruktor Binatang A> Konstruktor Kuda A> Konstruktor Binatang A> Konstruktor Burung A> Konstruktor Kuda_terbang A> dari Kuda_terbang...dari Kuda...Umur : 5

Analisa Program :

- Pada program Utama hanya menciptakan objek bertipe Kuda_terbang. Tampak pada hasil keluaran, dilihat dari konstruktornya maka bisa disimpulkan bahwa terbentuk objek Binatang, objek Kuda, objek Binatang, objek Burung baru kemudian objek Kuda_terbang. Ini menunjukkan bahwa ada 2 objek Binatang yang masing-masing merupakan bagian Kuda dan Burung.
- Oleh karena Kuda_terbang adalah turunan dari Kuda dan Burung, maka jika ia memanggil metode getUmur() yang merupakan warisan dari mereka, akan terjadi ambiguitas, oleh karena itu pada kelas Kuda_terbang dilakukan overriding seperti berikut :

```
virtual int const getUmur(){
    cout << "dari Kuda_terbang..." ;
    return Kuda::getUmur();
}
```

- Tampak pada hasil pemanggilan metode tersebut pada kelas Kuda_terbang: dari Kuda_terbang...dari Kuda...Umur : 5 Ini menunjukkan bahwa metode yang dipanggil adalah metode hasil override pada kelas Kuda_terbang yang di dalamnya sudah mengatasi problem ambiguitas dengan memastikan yang dipanggil adalah getUmur() dari kelas Kuda (yaitu : Kuda::getUmur()).
- Sekali lagi pada saat objek dihapus (delete), destruktor yang dijalankan adalah untuk objek Kuda_terbang, Burung, Binatang, Kuda dan Binatang ini menunjukkan bahwa ada 2 objek Binatang tadi.



9.10 Penurunan Virtual (Virtual Inheritance)

Berbeda dengan penurunan biasa (*Common Base Class*), penurunan virtual pada kedua kelas dasar (Kuda dan Burung) menyebabkan kelas turunan multi inheritance Kuda_terbang dapat langsung memanggil konstruktor kelas dasar paling atas (Binatang), sehingga terbentuklah objek dengan bentuk Diamond Inheritance seperti gambar tadi. Supaya lebih jelas, lakukan pecobaan berikut ini.

Contoh Penurunan pada umumnya (Common Base Class).

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 9.6, kemudian tulis kode berikut.

Listing 9.6: Penurunan pada umumnya (Common Base Class)

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class Binatang{
5 public:
6     Binatang(int umur=5):umur(5){cout << "Konstruktor
    Binatang\n";}
7     ~Binatang(){cout << "Destruktor Binatang\n";}
8     protected:
9     int umur;
10    public:
11        virtual int const getUmur(){
12            cout << "dari Binatang...";
13            return umur;
14        }
15    };
16    class Kuda : virtual public Binatang{
17    public:
18        Kuda(){cout << "Konstruktor Kuda\n";}
```



```
19 ~Kuda(){cout << "Destruktor Kuda\n";}
20 };
21 class Burung : virtual public Binatang{
22 public:
23 Burung(){cout << "Konstruktor Burung\n";}
24 ~Burung(){cout << "Destruktor Burung\n";}
25 };
26 class Kuda_terbang : public Kuda, public Burung{
27 public:
28 Kuda_terbang():Kuda(),Burung(),Binatang(){
29 cout << "Konstruktor Kuda_terbang\n";
30 ~Kuda_terbang(){cout << "Destruktor Kuda_terbang\n"
31 " ";}
32 int main(int argc, char *argv[])
33 {
34 QApplication a(argc, argv);
35 Kuda_terbang* gondrong = new Kuda_terbang();
36 cout << "Umur : " << gondrong->getUmur() << endl;
37 delete gondrong;
38 return a.exec();
39 }
```

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

```
Konstruktor Binatang
Konstruktor Kuda
Konstruktor Burung
Konstruktor Kuda_terbang dari
BinatangUmur : 5
```

Analisa Program :

- Pada program Utama, sama seperti percobaan Contoh 9.5, hanya mencip-



takan objek bertipe Kuda_terbang. Tampak pada hasil keluaran, dilihat dari konstruktornya maka bisa disimpulkan bahwa terbentuk objek Binatang, objek Kuda, objek Binatang, objek Burung baru kemudian objek Kuda_terbang. Ini menunjukkan bahwa ada 2 hanya 1 objek Binatang yang masing-masing merupakan bagian Kuda, dan Burung dan Kuda_terbang.

- Walaupun Kuda_terbang adalah turunan dari Kuda dan Burung, namun saat instansiasi kelas Kuda_terbang memanggil langsung konstruktor Binatang seperti berikut:

```
Kuda_terbang() : Kuda(), Burung(), Binatang()
```

Memang kelas Kuda_terbang bukan keturunan langsung kelas Binatang, namun hal ini bisa dimungkinkan karena kelas dasar dari Kuda_terbang, yaitu Kuda dan Binatang melakukan penurunan secara virutal :

```
class Kuda : virtual public Binatang{}  
class Burung : virtual public Binatang{}
```

sehingga dimungkinkan konstruktor Kuda_terbang() memanggil langsung konstruktor Binatang(). Konsekuensinya maka jika ia memanggil metode getUmur() yang merupakan warisan dari Binatang, metode-metode warisan yang ada pada Kuda dan Burung akan diabaikan (kecuali mereka melakukan override) dan di-“By pass” langsung ke kelas Binatang. Oleh karena itu pada kelas Kuda_terbang tidak lagi terjadi ambigu seperti tadi. - Tampak pada hasil pemanggilan metode getUmur() tersebut pada kelas Kuda_terbang: dari Binatang... Umur : 5 Ini menunjukkan bahwa metode yang dipanggil adalah metode warisan dari kelas Binatang secara langsung. - Sekali lagi pada saat objek dihapus (delete), destruktor yang dijalankan adalah untuk objek Kuda_terbang, Burung, Binatang, Kuda dan Binatang ini menunjukkan bahwa ada 2 hanya ada 1 objek Binatang. - Jika dibayangkan, maka “silsilah kekerabatan” penurunan virtual ini adalah seperti gambar Daimond Inheritance di atas.



Catatan:



- Untuk memastikan kelas turunan hanya mempunyai sebuah kelas dasar bersama, deklarasikan kelas-kelas turunan secara virtual dari kelas dasar. Contoh:

```
class Binatang //<-- common base class (
    kelas dasar bersama)
class Kuda : virtual public Binatang //<-- 
    penurunan secara virtual
class Burung : virtual public Binatang // 
    <-- penurunan secara virtual
class Kuda_terbang : public Kuda, public
    Burung
```

- Jika penurunan dilakukan secara virtual seperti di atas, secara otomatis kelas Kuda_terbang akan memanggil konstruktor Binatang() walaupun tidak ditulis secara eksplisit, contoh konstruktor Kuda_terbang() tanpa memanggil konstruktor Binatang() berikut ini akan menghasilkan hasil yang sama:

```
Kuda_terbang(){//<-- pemanggilan Kuda(),
    Burung() dan Binatang() implisit
cout << "Konstruktor Kuda_terbang\n";}
```

9.11 Masalah Pada Multiple Inheritance

Meskipun multiple inheritance menawarkan banyak keuntungan dibanding penurunan tunggal, banyak pemrogram C++ menghindari penggunaan multiple inheritance. Mereka mengatakan bahwa multiple inheritance membuat pelacak kesalahan menjadi lebih sulit, dengan mengembangkan kelas multiple inheritance hirarki semakin rumit dan semakin berisiko dibandingkan penurunan tunggal dan bahwa hampir semua yang bisa dilakukan dengan multiple inheritance juga dapat dilakukan dengan single inheritance. Bahasa pemrograman lain seperti Java dan C# tidak mendukung multiple inheritance dengan alasan yang



sama. Oleh karena itu, jika bisa dilakukan dengan cara single inheritance jangan memakai multiple inheritance.



Catatan

- Pakailah multiple inheritance jika suatu kelas baru memerlukan fitur-fitur yang ada di dua kelas yang berbeda.
- Pakailah penurunan secara virtual jika ada lebih dari satu kelas turunan namun hanya diperlukan sebuah instan dari kelas dasar.
- Pasti terjadi pemanggilan konstruktor kelas dasar bersama (shared base class) dari kelas turunan paling bawah (multiple inheritance) ketika memakai kelas dasar virtual.



BAB 10

Casting dan Database

Agenda

Pada chapter ini kita akan membahas beberapa topik tentang Casting dan Database, adapun topik yang akan dibahas adalah

Contents

| | |
|--|------------|
| 10.1 Mengenal Casting | 318 |
| 10.2 Data Type Casting (Conversion) | 319 |
| 10.2.1 Implicit conversion | 319 |
| 10.2.2 Explicit conversion | 320 |
| 10.3 Casting Operator pada C++ | 324 |
| 10.3.1 Penggunaan static_cast | 324 |
| 10.3.2 Penggunaan dynamic_cast | 326 |
| 10.3.3 Penggunaan reinterpret_cast | 330 |
| 10.3.4 Penggunaan const_cast | 332 |
| 10.4 Pemrograman Basisdata dengan QtConsole Application | 334 |
| 10.4.1 Koneksi Qt dengan Basisdata | 335 |



| | |
|--|-----|
| 10.4.2 Koneksi Qt dengan MySQL dan menampilkan datanya | 335 |
| 10.4.3 Koneksi Qt dengan SQLite | 340 |
| 10.4.4 Membaca data pada tabel SQLITE | 344 |
| 10.4.5 Menambah data pada tabel SQLITE | 346 |
| 10.4.6 Mengedit data pada tabel SQLITE | 350 |
| 10.4.7 Menghapus data pada tabel SQLITE | 352 |

10.1 Mengenal Casting

Casting merupakan mekanisme dimana programmer dapat secara permanen atau temporary mengubah interpretasi compiler terhadap suatu obyek. Perubahan ini tidak benar-benar terjadi, namun hanya cara pandang compiler saja yang diubah. Casting diimplementasikan dalam bentuk “casting operator”. Mengapa butuh casting? Dalam dunia pemrograman yang semuanya jelas (strong type language) dan jika kita hanya menggunakan satu bahasa pemrograman saja, seperti C++, maka kita tidak membutuhkan operator casting. Namun kenyataannya pada dunia nyata yang kita hadapi, banyak bahasa pemrograman, banyak vendor-vendor berbeda-beda sehingga kode / modul yg dihasilkan jd berbeda-beda. Hal ini menyebabkan compiler-compiler bahasa pemrograman tertentu, termasuk C++ juga harus diubah interpretasinya dengan cara lain sehingga mampu melakukan kompilasi dan menghasilkan hasil yang kompatibel.

Contoh nyatanya terdapat pada bahasa C dan C++. Pada bahasa C tidak terdapat tipe data bool (boolean) sehingga kita harus menggunakan kata kunci typedef.

```
typedef unsigned int BOOL;  
BOOL mybool = 0;  
BOOL isTrue(){  
    return mybool;  
}
```



Pada contoh diatas maka kita harus membuat tipe data baru bertipe unsigned int yang kita definisikan sebagai BOOL. Nah setelah kita mendefinisikan tipe data baru BOOL pada C, bagaimana jika kemudian dikembangkan dengan bahasa C++? Kita harus melakukan konversi BOOL pada bahasa C ke bahasa C++, dimana bahasa C++ sudah ada tipe data bool yang tentunya berbeda “persepsi” dengan BOOL dari bahasa C. Maka dari itu dilakukan casting. Berikut adalah contoh castingnya:

```
bool hasilku = (bool) isTrue(); // C-Style cast
```

10.2 Data Type Casting (Conversion)

Mengubah sebuah ekspresi dari tipe yang diberikan dalam jenis lain dikenal sebagai tipe-casting. Konversi tersebut ada dua jenis:

10.2.1 Implicit conversion

Jenis ini tidak membutuhkan operator khusus. Tipe data yang jangkauannya besar biasanya dapat dikonversi ke tipe data yang jarak jangkauannya lebih kecil secara otomatis dengan cara pemotongan nilai otomatis. Kemudian variabel yang tipe datanya berjarak jangkauan besar juga dapat menerima data dari variabel yang bertipe data berjarak jangkauan kecil.

Contoh:

```
short a=2000;  
int b;  
b=a;
```

Pada contoh diatas, tipe data `short` yang berjarak jangkauan kecil dapat ditampung oleh tipe data `int` yang jarak jangkauannya lebih besar, walaupun tipe datanya berbeda. Hal ini karena keduanya sama-sama data numerik dan



memang termasuk dalam konversi implisit. *Implisit conversion* juga dapat diterapkan pada *constructor* sebuah *class*, sehingga ketika kita memanggil konstruktor tersebut, maka konversi akan dilakukan. Contoh:

```
class A {  
};  
class B {  
public:  
    B (A a)  
    {  
    }  
};
```

Proses instansiasi adalah:

```
A a;  
B b=a;
```

Pada contoh diatas, terlihat bahwa ketika kita membuat obyek class B, maka otomatis konstruktor class B dijalankan dan menerima parameter obyek A. Sehingga kita bisa memasukkan parameter bertipe class A kedalam class B.

10.2.2 Explicit conversion

Konversi tipe ini harus dituliskan pada kode program dengan menggunakan tanda kurung.

Sintaks:

(<type>) <value>

Contoh:

```
short a=2000;  
int b;  
b = (int) a; // c-like cast notation  
b = int (a); // functional notation
```



Cara pertama dengan menegunakan *C-cast like notation*. Pada contoh diatas, kita memaksa / mengcasting variabel a yang bertipe `short` agar diperlakukan seperti tipe data `integer` dengan cara memberi tanda kurung (`int`) sebelum variabel a. Dengan demikian variabel a akan dianggap / diperlakukan oleh kompiler menjadi tipe data integer dan bisa di assign ke dalam variabel b yang bertipe integer.

Cara kedua adalah menggunakan *functional notation* dimana kita bisa memaksa variabel a yang bertipe `short` diperlakukan menjadi `integer` dengan cara menuliskan kata `int` kemudian diberi tanda kurung pada variabel a sehingga variabel a bisa di assign ke dalam variabel b yang bertipe `integer`.

Contoh Explicit Casting pada Tipe Data Numerik.

Tulislah kode berikut:

Listing 10.1: Explicit Casting pada Tipe Data Numerik

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 int main(int argc, char *argv[])
5 {
6     QCoreApplication a(argc, argv);
7     int aa=5;
8     int bb=2;
9     float x = 5.0;
10    float y = 2.0;
11    int c=aa/bb;
12    cout<<"1. c="<<c<<endl;
13    c = (float) aa/bb;
14    cout<<"2. c="<<c<<endl;
15    c = (float) aa / (float) bb;
16    cout<<"3. c="<<c<<endl;
17    c = x / y;
```



```
18 cout<<"4. c="<<c<<endl;
19 c = (int) x / (int) y;
20 cout<<"5. c="<<c<<endl;
21 c = (float) x/y;
22 cout<<"6. c="<<c<<endl;
23 float d=aa/bb;
24 cout<<"7. d="<<d<<endl;
25 d = (float) aa/bb;
26 cout<<"8. d="<<d<<endl;
27 d = (float) aa / (float) bb;
28 cout<<"9. d="<<d<<endl;
29 d = x / y;
30 cout<<"10. d="<<d<<endl;
31 d = (int) x/ (int) y;
32 cout<<"11. d="<<d<<endl;
33 d = (int) x/y;
34 cout<<"12. d="<<d<<endl;
35 return a.exec();
36 }
```

Hasil:

```
1. c=2
2. c=2
3. c=2
4. c=2
5. c=2
6. c=2
7. d=2
8. d=2.5
9. d=2.5
10. d=2.5
11. d=2
12. d=2.5
```



Keterangan:

Mari kita analisa perbaris.

- Integer dibagi integer dan dimasukkan ke dalam variabel integer akan menjadi integer, walaupun ada koma dibelakangnya, yaitu 0.5 tapi akan dipotong, karena tipe data integer tidak mampu menerima koma, sehingga akan dipotong.
- Walaupun kita casting ke dalam float, namun kenyataannya tidak berhasil, karena tipe data integer lebih kecil lebar datanya daripada tipe data float, sehingga ketika kita casting dalam float, datanya sudah terlanjur terpotong, sehingga tidak ada perubahan
- Perintah ketiga sama Keterangan:nya dengan no 2.
- C bertipe integer, sehingga walaupun x dan y float dan ada koma, namun ketika disimpan didalam integer akan terpotong menjadi bilangan bulat
- Sudah jelas, karena x dicasting ke integer dan y juga, berarti semua menjadi integer
- Keterangan: sama dengan alasan no 2
- D bertipe float namun karena aa dan bb bertipe integer, maka hasilnya integer juga dan nilainya sudah terpotong, sehingga ketika disimpan kedalam tipe float sudah terlanjur terpotong nilainya.
- Variabel aa dan bb yang bertipe integer dipaksa menjadi float, dan karena disimpan dalam variabel d yang bertipe float, maka hasilnya float
- Variabel aa dan bb yang bertipe integer dipaksa menjadi float masing-masing sehingga akan menjadi float, dan disimpan didalam variabel float d.
- Sudah jelas, float dibagi float dan disimpan dalam tipe float sehingga sudah benar tampil sebagai float
- Variabel x dan y yang bertipe float dipaksa menjadi tipe data yang lebih



kecil, yaitu integer, kemudian baru disimpan dalam tipe data integer sehingga hasil akhirnya integer.

- Variabel yang dicasting hanya variabel x sehingga variabel y dan d masih float. Tipe data x (integer) dibagi float maka akan tetap menjadi float, karena float tipe datanya lebih besar lebar datanya daripada integer.



TIPS

Untuk melakukan casting yang benar, maka casting akan valid jika kita mengcasting tipe data yang berukuran lebih besar menjadi tipe data yang berukuran lebih kecil. Misalnya float dicast menjadi int.

10.3 Casting Operator pada C++

C++ memiliki cara pengcastingan yang baru, yaitu:

- static_cast
- dynamic_cast
- reinterpret_cast
- const_cast

Bentuk umum untuk semuanya adalah:

```
tipe_tujuan hasil = tipe_casting <tipe_tujuan> (  
    obyek_yg_mau_dicasting);
```

10.3.1 Penggunaan static_cast

Static_cast adalah mekanisme yang dapat digunakan untuk mengkonversi pointer diantara tipe data/class terkait dan melakukan konversi tipe data tersebut



secara eksplisit untuk tipe data standar yang jika tidak dilakukan konversi akan terjadi secara otomatis (implisit). Dengan menggunakan konsep pointer, `static_cast` menerapkan pengecekan casting pada saat compile-time dengan melakukan pemeriksaan untuk memastikan bahwa pointer dicasting ke tipe yang benar/sesuai. Hal ini merupakan perbaikan dari casting yang berjenis C-style, dimana memungkinkan casting ke obyek yang tidak ada relasinya sama sekali. Dengan menggunakan `static_cast`, pointer bisa dicasting ke class induknya atau dapat down-case menjadi class turunannya. Berikut contohnya:

```
CInduk* pInduk = new CTurunan(); // membuat obyek
                                Cturunan dari Cinduk (polymorfisme)
CTurunan* pTurunan = static_cast<CTurunan*>(pInduk);
                                // mengcasting pInduk menjadi Cturunan, valid!
// CTdkAdaHubungan merupakan class yang tidak ada
// hubungannya dengan CInduk melalui inheritance
CTdkAdaHubungan* pTdkHub = static_cast<CTdkAdaHubungan
                                *>(pInduk); // Error
// karena casting tidak diperbolehkan, tidak ada
// hubungan class!
```

Keterangan:

- Pada contoh diatas terlihat bahwa class anak dapat dibuat dari class induk karena ada hubungan pewarisan. Konsep ini merupakan konsep polimorfisme. Kemudian untuk memastikan agar tipe data pInduk benar/valid untuk dimasukkan ke pTurunan yang merupakan obyek CTurunan, maka dilakukan casting dengan `static_cast`.
- Kemudian pada bagian kedua, terlihat bahwa jika kita membuat class yang tidak ada hubungan apapun dengan class Induk maka kita tidak bisa melakukan `static_cast`.
- Dengan demikian, `static_cast` digunakan untuk menyakinkan validitas suatu obyek pointer bahwa obyek tersebut ada hubungan dengan obyek yang dicastingnya. Pengcastingan dilakukan dengan mengubah class Induk menjadi class Anak, bukan sebaliknya.



Bagi para programmer C yang beralih ke C++, static casting sangat mirip dengan C-style casting dan sangat disarankan untuk mengganti C-style casting karena static casting lebih aman dan tampak tertulis dengan jelas. Kita dapat melakukan `static_cast` pada tipe data biasa agar programmer dapat melihat

secara eksplisit tipe data yang dicastingnya. Sintaks umumnya adalah:

```
static_cast<<type>>(<value>);
```

Berikut contohnya:

```
double myphi = 3.14;  
int angka = static_cast<int>(myphi);
```

10.3.2 Penggunaan `dynamic_cast`

Fungsi `dynamic_cast` merupakan kebalikan dari `static_cast`, hal ini karena proses pengcastingan terjadi saat runtime. Casing jenis ini sangat tepat untuk digunakan pada class yang memiliki sifat polimorfisme. Casting ini dapat digunakan untuk melakukan casting secara aman pada pointer superclass menjadi sebuah pointer subclass dalam sebuah hierarki kelas. Jika ternyata casting invalid karena tipe obyek yang dicasting tidak setipe dengan class supernya, maka casting akan gagal. Agar lebih aman, penggunaan `dynamic_cast` sebaiknya digunakan dalam blok `try...catch`.

Bentuk umumnya adalah:

```
tipe_tujuan* pTujuan = dynamic_cast <tipe_class*> (  
    pSumber);  
if (pTujuan) // apakah proses casting sukses?  
pTujuan->CallFunc ();
```

Contoh penggunaan:



```
CInduk* pInduk = new CTurunan();
// Melakukan down casting
CTurunan* pTurunan = dynamic_cast <CTurunan*> (pInduk)
;
if (pTurunan) // cek apakah sukses?
pTurunan->CallFungsiClassTurunan();
```

Keterangan:

Pada contoh diatas, kita membuat obyek class Turunan dari class Induk, kemudian kita melakukan casting ke class Turunan untuk memastikan validitas obyeknya, kemudian karena sifat pengecekan compiler bersifat runtime, maka kita bisa memeriksa apakah proses castingnya telah berjalan dengan sukses atau tidak.

Dynamic_cast juga dapat digunakan untuk referensi pointer. Caranya dengan menggunakan tanda &.

Casting ini tidak boleh menghasilkan kembalian NULL. Sintaksnya:

```
<type> subclass = dynamic_cast<<type> &>( ref_obj );
```

Contoh:

```
class CBase { };
class CDerived: public CBase { };
CBase b; CBase* pb;
CDerived d; CDerived* pd;
pb = dynamic_cast<CBase*>(&d); // ok: derived-to-base
pd = dynamic_cast<CDerived*>(&b); // wrong: base-to-
derived
```

Contoh Dynamic Casting.

Buatlah program berikut:



Listing 10.2: Contoh Dynamic Casting

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 using namespace std;
4 class CAnimal
5 {
6 public:
7     virtual void Bersuara () = 0;
8 };
9 class CDog : public CAnimal
10 {
11 public:
12     void KibasEkor () {
13         cout << "Dog: Kibas-kibas ekor!" << endl;
14     }
15     void Bersuara () {
16         cout << "Dog: Guk-guk!" << endl;
17     }
18 };
19 class CCat : public CAnimal
20 {
21 public:
22     void TangkapTikus () {
23         cout << "Cat: tikus tertangkap!" << endl;
24     }
25     void Bersuara () {
26         cout << "Cat: Meong!" << endl;
27     }
28 };
29 void TentukanTipeClass (CAnimal* pAnimal);
30 int main(int argc, char *argv[])
31 {
32     QCoreApplication a(argc, argv);
```



```
33 // pAnimal1 berupa obyek Dog
34 CAnimal* pAnimal1 = new CDog ();
35 // pAnimal2 berupa obyek Cat
36 CAnimal* pAnimal2 = new CCat ();
37 cout << "Penggunaan dynamic_cast untuk menentukan
38 jenis Animal 1" << endl;
39 cout << "Penggunaan dynamic_cast untuk menentukan
40 jenis Animal 2" << endl;
41 TentukanTipeClass(pAnimal1);
42 cout << "Verifikasi tipe: Animal 1 Besuara!" << endl;
43 pAnimal1->Bersuara ();
44 cout << "Verifikasi tipe: Animal 2 Besuara!" << endl;
45 pAnimal2->Bersuara ();
46 return a.exec();
47 }
48 void TentukanTipeClass (CAnimal* pAnimal)
49 {
50     CDog* pDog = dynamic_cast <CDog*>(pAnimal);
51     if (pDog)
52     {
53         cout << "Binatang Dog!" << endl;
54         // panggil fungsi dog
55         pDog->KibasEkor();
56     }
57     CCat* pCat = dynamic_cast <CCat*>(pAnimal);
58     if (pCat)
59     {
60         cout << "Binatang kucing!" << endl;
61         pCat->TangkapTikus();
62     }
63 }
```



Hasil:

```
Penggunaan dynamic_cast untuk menentukan jenis Animal 1  
Binatang Dog!  
Dog: Kibas-kibas ekor!  
Penggunaan dynamic_cast untuk menentukan jenis Animal 2  
Binatang kucing!  
Cat: tikus tertangkap!  
Verifikasi tipe: Animal 1 Besuara!  
Dog: Guk-guk!  
Verifikasi tipe: Animal 2 Besuara!  
Cat: Meong!
```

Keterangan:

Kelas abstrak Canimal diturunkan pada dua class, yaitu CDog dan CCat sehingga memiliki function Berbicara() dimana Dog menggunakannya untuk menggonggong dan Cat menggunakannya untuk mengeong. Fungsi yang digunakan adalah dynamic_cast yang akan menentukan mana method Berbicara() yang diimplementasikan. Dog akan mengimplementasikan menggonggong, sedangkan Cat akan mengimplementasikan mengeong. Setelah mengetahui fungsi mana yang diimplementasikan, method TentukanTipe juga dapat menggunakan pointernya untuk memanggil method pada class turunannya sesuai dengan jenis classnya. Untuk class Dog memanggil method KibasEkor(), sedangkan class Cat memanggil method TangkapTikus().

10.3.3 Penggunaan reinterpret_cast

Penggunaan casting ini benar-benar tidak memungkinkan programmer untuk mengcasting dari satu obyek ke jenis obyek lain, terlepas dari apakah jenis obyeknya berhubungan atau tidak. Casting ini tidak boleh digunakan untuk melakukan down casting pada hierarki kelas atau untuk menghapus const volatile. Sintaks:

```
reinterpret_cast<<type>>( <val> ) ;
```



Contoh:

```
reinterpret_cast<int*>(100);
```

Reinterpret_cast pada class menggunakan syntax sebagai berikut:

```
CInduk * pInduk = new CInduk ();
CTdkAdaHubungan * pTdkHubung = reinterpret_cast<
    CTdkAdaHubungan*>(pInduk);
// program diatas bisa dikompile tapi sangat tidak
// disarankan karena class CtdkAdaHubungan bukanlah
// turunan dari Cinduk.
```

Casting model ini sebenarnya memaksa compiler untuk menerima situasi dimana pada static_cast tidak diijinkan. Casting model ini biasanya ditemukan pada pemrograman aplikasi tingkat rendah tertentu (seperti driver) dimana harus dilakukan konversi ke tipe sederhana dimana API dapat menerimanya.

```
CSomeClass* pObject = new CSomeClass ();
// harus dikirimkan dalam bentuk byte (unsigned char)
unsigned char* pBytes = reinterpret_cast <unsigned
    char*>(pObject);
```

Contoh lain:

```
class A {};
class B {};
A * a = new A;
B * b = reinterpret_cast<B*>(a);
```

**TIPS**

Sebisa mungkin reinterpret_cast tidak digunakan jika tidak terpaksa karena tidak aman.



10.3.4 Penggunaan `const_cast`

`Const_cast` digunakan untuk menghilangkan sifat `const-ness` atau sifat `volatile`-an dari sebuah variabel. `Const_cast` harus digunakan dengan tepat. Salah satu contoh penggunaan yang sah dari `const_cast` adalah untuk menghilangkan sifat `const-an` dari sebuah pointer agar dapat lulus menjadi fungsi ketika kita yakin fungsi tersebut tidak akan memodifikasi variabel tetapi fungsi itu didesain untuk tidak menentukan inputan sebagai konstanta.

Sintaks:

```
const_cast<<type>>(<value>);
```

Contoh:

```
void func(char *);  
const char *x = "abcd";  
func(const_cast<char *>(x));
```

`const_cast` juga memungkinkan kita untuk menonaktifkan method `const` pada suatu objek. Mengapa diperlukan? Karena kadang-kadang programmer melupakan penggunaan `const` pada method yang seharusnya berjenis `const` method. Contoh:

```
ContohClass  
{  
public:  
// ...  
void tampilanAnggota (); // method ini berjenis const  
};  
void tampilanData (const ContohClass& mData)  
{  
mData.tampilanAnggota (); // compile error, karena "  
call to a non-const member using  
a const reference"  
}
```



Kita dapat menggunakan `const_cast` untuk mengubah `a` adalah:

```
void tampilanData (const ContohClass& mData)
{
ContohClass& refData = const_cast <ContohClass&>(mData
    );
refData.tampilanAnggota(); // OK!
}
```

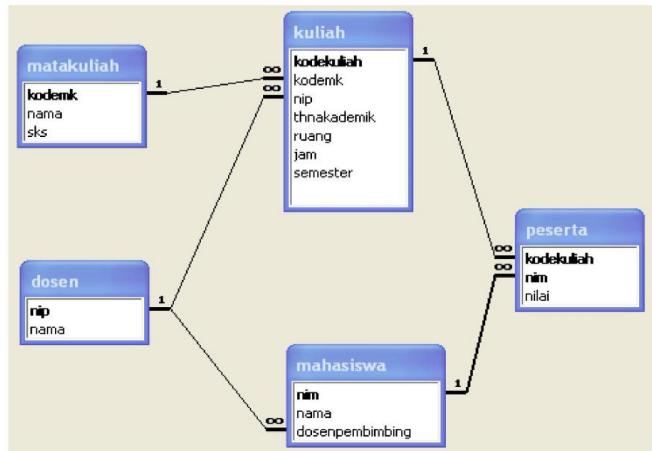
Kita juga dapat menggunakan pointer:

```
void tampilanData (const ContohClass* pData)
{
// pData->DisplayMembers(); Error: attempt to invoke a
// non-const function!
CSomeClass* pCastedData = const_cast <CSomeClass*>(
    pData);
pCastedData->DisplayMembers(); // Allowed!
}
```

Contoh lain:

```
// const_cast
#include <iostream>
using namespace std;
void print (char * str)
{
cout << str << endl;
}
int main () {
const char * c = "sample text";
print ( const_cast<char *> (c) );
return 0;
}
```





Gambar 10.1: Pemrograman Basis data Qt Creator

10.4 Pemrograman Basisdata dengan QtConsole Application

Pada bagian kedua ini kita akan berkenalan dengan bagaimana mengakses basisdata dengan menggunakan Qt. Basis data adalah suatu kumpulan tabel-tabel yang berisi data-data yang saling berelasi satu sama lain secara logika. Basis data tersusun dari tabel, sedangkan tabel tersusun dari baris record-record yang memiliki atribut (kolom) dan lainnya. Gambaran basis data pada Qt Creator seperti gambar 10.1

Pada contoh diatas, terdapat sebuah basisdata perkuliahan, dimana terdapat 5 buah tabel. Tabel yang ada memiliki field (atribut/kolom). Field pada masing-masing tabel sangat berasosiasi dengan tabelnya, artinya kodemk, nama, dan sks pada tabel matkuliah sangat spesifik menggambarkan tabel matakuliah, demikian juga yang lainnya.

Kita tidak akan membahas lebih lanjut tentang basisdata. Pada basis data kita juga mengenal SQL (Structured Query Language). SQL merupakan bahasa



untuk mengakses dan memanipulasi basis data terutama isi record-record pada tabel.

10.4.1 Koneksi Qt dengan Basisdata

Untuk melakukan koneksi pada basis data, Qt menyediakan dukungan pada beberapa basis data yang terkenal, misalnya [Sqlite](#), [Oracle](#), [MySQL](#), Db2¹, ODBC², dan [Postgresql](#). Secara default basis data yang didukung adalah Sqlite dan ODBC saja, sedangkan untuk basis data lainnya harus menggunakan driver yang biasanya harus didownload pada situsnya secara langsung.

Pada Qt kita dapat membuat aplikasi console yang terkoneksi dengan basis data. Koneksi terhadap kedua database tersebut tidak perlu melakukan konfigurasi dan mendownload driver tertentu. Pada tulisan ini kita akan menggunakan basis data Sqlite.

10.4.2 Koneksi Qt dengan MySQL dan menampilkan datanya

[MySQL](#) merupakan database yang sudah disupport oleh Qt. Untuk membuat database MySQL.

Untuk melakukan koneksi QtConsole dengan MySQL, maka lakukan Contoh berikut:

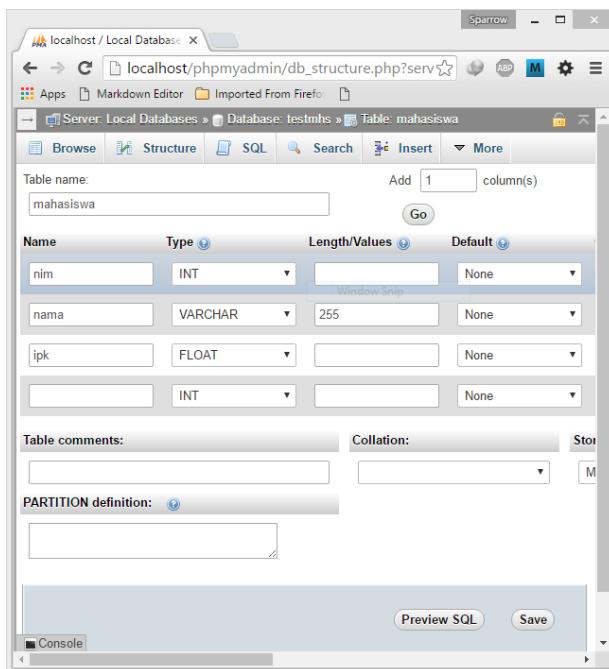
¹IBM DB2 adalah produk database server yang dikembangkan oleh IBM. Produk-produk ini mendukung sistem manajemen basis data relasional (relational DBMS), namun belakangan ini sudah mendukung pula sistem manajemen basis data berbasis object-relational (object-relational DBMS) dan juga non-relational seperti XML.

²Open Database Connectivity (ODBC) adalah Application Programming interface (API) database yang khusus digunakan untuk mengakses database relasional. ODBC terdapat dalam setiap komputer yang menggunakan sistem operasi windows, karena ODBC merupakan bagian dari Windows Open System Architecture (WOSA). Dalam ODBC disediakan berbagai Application Programming Interface (API) yang berguna untuk menyediaan dan memberikan stiktar bagi berbagai kegiatan pemrograman. Keuntungan utama menggunakan ODBC ini adalah fleksibilitas, fleksibel disini artinya pengubahan jenis database yang dipergunakan oleh sebuah aplikasi tidak akan mempengaruhi kode program aplikasi tersebut.



Contoh Percobaan koneksi MySQL dengan QtConsole

1. Buatlah sebuah database pada MySQL dengan nama: testmhs
2. Gunakan gunakan PHPmyAdmin untuk membuat tabel database.



3. Isilah datanya sebagai berikut:

| NIM | Nama | IPK |
|--------|--------|-----|
| 011041 | Wachid | 3.6 |
| 012042 | Arif | 3.6 |
| 011012 | Eko | 3.4 |

Hasil:



| + Options | | |
|-----------|--------|-----|
| nim | nama | ipk |
| 11041 | Wachid | 3.6 |
| 12042 | Arif | 3.6 |
| 11012 | Eko | 3.4 |

4. Tulis kode berikut ini:

Listing 10.3: Percobaan koneksi MySQL dengan QtConsole

```
1 #include <QtCore/QCoreApplication>
2 #include <QtSql/QtSql>
3 #include <QtDebug>
4 int main(int argc, char *argv[])
5 {
6     QCoreApplication a(argc, argv);
7     qDebug() << QSqlDatabase::drivers();
8     QSqlDatabase db = QSqlDatabase::addDatabase("QMYSQL");
9     db.setDatabaseName( "testmhs" );
10    db.setHostName("localhost");
11    db.setUserName("root");
12    db.setPassword("");
13    if( !db.open() )
14    {
15        qDebug() << db.lastError();
16        qFatal( "Failed to connect." );
17    } else {
18        qDebug() << "Koneksi sukses";
```



```
19 QSqlQuery query(db);
20 query.exec("SELECT * FROM mahasiswa");
21 while (query.next()) {
22     qDebug() << query.value(0).toString();
23     qDebug() << query.value(1).toString();
24     qDebug() << query.value(2).toString();
25 }
26 }
27 return a.exec();
28 }
```

5. Pada file project yang berekstensi .pro, tambahkan linking ke library sql sebagai berikut:

```
1 #-----
2 #
3 # Project created by QtCreator 2016-01-03T19:58:33
4 #
5 #-----
6 QT += core
7 QT += gui
8 Qt += sql
9 TARGET = databases
10 CONFIG += console
11 CONFIG -= app_bundle
12 TEMPLATE = app
13 SOURCES += main.cpp
```

6. Kemudian run dan hasilnya adalah sebagai berikut:



```
( "QSQLITE", "QMYSQL", "QODBC3", "QODBC")  
Koneksi sukses  
"11041"  
"Wachid"  
"3.6"  
"12042"  
"Arif"  
"3.6"  
"11012"  
"Eko"  
"3.4"
```

Keterangan:

- Program diatas dapat menampilkan driver database yang terinstall dan dapat dikenali oleh system Qt, yaitu QSQLITE, QMYSQL, QODBC3, dan QODBC. Berarti sistem QT dapat mendukung basisdata Sqlite, MySQL, dan ODBC dari Microsoft.
- Kemudian langkah pertama yang harus dilakukan adalah membuat QSqlDatabase yang akan meload basis data yang dipilih beserta dengan drivernya. Setelah itu kita harus menentukan nama database yang akan diakses, user, password, dan host lokasi MySQL server.
- Kemudian akan diperiksa apakah database yang terpilih dapat dibuka atau tidak dengan method open(). Jika berhasil maka akan dilanjutkan, jika tidak maka akan ditampilkan error yang terjadi dengan menggunakan method dari database, yaitu lastError().
- Langkah berikutnya adalah melakukan query dengan menggunakan method query(). Setelah perintah SQL dijalankan maka record-record yang dihasilkan dari perintah select tersebut akan diloop satu persatu dengan method next() dari query, dan ditampilkan hasilnya dilayar.



10.4.3 Koneksi Qt dengan SQLite

SQLite merupakan database yang sudah disupport secara native oleh Qt. Untuk membuat database SQLite, kita membutuhkan tool yang dapat digunakan untuk mengelola databasenya dengan mudah, silahkan gunakan sqliteadmin³ yang berbasis desktop.

Yang perlu diperhatikan ketika kita membuat koneksi dengan basisdata SQLite adalah:

1. Jika basis data SQLite akan dibuat langsung dari program, maka file database hasil pembuatan tersebut akan berada di folder simulator pada project kita.
2. Jika kita sudah memiliki file database SQLite, maka file tersebut harus diletakkan (dikopikan) ke folder simulator atau simulator/debug
3. File SQLite yang dibuat harus berjenis SQLite 3 agar bisa diakses.

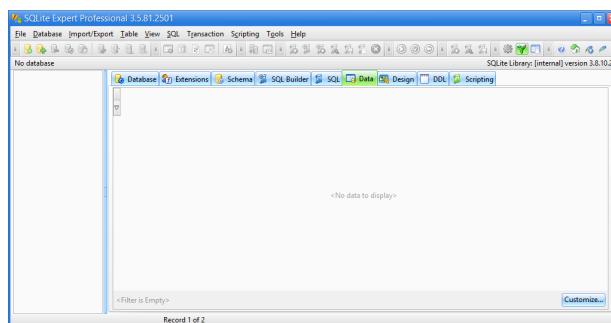
Untuk melakukan koneksi QtConsole dengan SQLite, maka lakukan Contoh berikut:

Contoh Percobaan koneksi SQLite dengan QtConsole

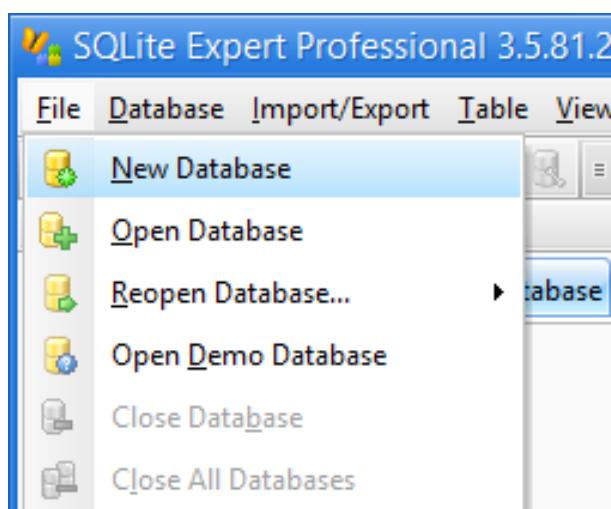
1. Buatlah sebuah database pada Sqlite dengan nama: testmhs.db, ingat harus berjenis SQLite3.

³<http://www.sqliteexpert.com/>





2. Gunakan Sqlite Expert untuk membuatnya
3. Pilih menu File > New Database



4. Simpan dengan nama testmhs.db (pilih tipe sqlite 3 database), simpan pada folder project yang akan dibuat.
5. Kemudian buat tabel baru dengan klik menu table > New Table, kemudian isikan data berikut:

Nama tabel: mahasiswa



Field:

- NIM tipe VARCHAR(8), primary key, not null, unique
- Nama tipe VARCHAR(30), not null
- IPK tipe FLOAT

6. Kemudian isi data sebagai berikut:

| NIM | Nama | IPK |
|--------|--------|-----|
| 011041 | Wachid | 3.6 |
| 012042 | Arif | 3.6 |
| 011012 | Eko | 3.4 |

7. Setelah itu buatlah project baru pada QtConsole application, dan tulislah kode program berikut:

Listing 10.4: Percobaan koneksi SQLite dengan QtConsole

```
1 #include <QtCore/QCoreApplication>
2 #include <QDebug>
3 #include <QtSql/QtSql>
4     int main(int argc, char *argv[])
5     {
6         QCoreApplication a(argc, argv);
7         qDebug() << QSqlDatabase::drivers();
8         QSqlDatabase db = QSqlDatabase::addDatabase(""
9             QSQLITE");
10        db.setDatabaseName( "testmhs.db" );
11        if( !db.open() )
12        {
13            qDebug() << db.lastError();
14            qFatal( "Failed to connect." );
15        } else qDebug() << "Koneksi berhasil";
16        return a.exec();
```



16 }

8. Pada project, pilihlah file berekstensi .pro, kemudian bukalah file tersebut dan tambahkanlah bagian kode berikut:

```
1 #-----  
2 #  
3 # Project created by QtCreator 2016-01-03T19  
4 :58:33  
5 #-----  
6 QT += core  
7 QT += gui  
8 Qt += sql  
9 TARGET = databases  
10 CONFIG += console  
11 CONFIG -= app_bundle  
12 TEMPLATE = app  
13 SOURCES += main.cpp
```

9. Build dan run

```
( "QSQLITE")  
Koneksi berhasil
```

Keterangan:

- Baris pertama output adalah daftar driver yang didukung oleh Qt dan QtCreator.
- Baris kedua adalah menggambarkan bahwa koneksi terhadap database Sqlite berhasil!
- Program diatas mengharuskan adanya urutan perintah yang harus dilakukan ketika kita akan melakukan koneksi ke database Sqlite, yaitu:



- Load driver SQLITE Tambahkan bagian ini
- setDatabaseName sesuai dengan nama file Sqlite yang akan diakses
- Kemudian open koneksi dengan memanggil method open
- Jika ada error tampilkan errornya, jika tidak tampilkan bahwa koneksi berhasil

10.4.4 Membaca data pada tabel SQLITE

Cara membaca data pada tabel Sqlite adalah dengan menggunakan perintah query SQL SELECT. Sintaksnya adalah `SELECT <field, field, field> FROM <nama_tabel> WHERE <kondisi>`. Perintah diatas bisa menghasilkan record atau malah salah sekali tidak menghasilkan record apapun. Jika menghasilkan record, maka record yang dihasilkan bisa satu atau lebih dari satu.

Pada Qt cara yang digunakan untuk membaca data adalah dengan menggunakan class QSqlQuery dan method query seperti berikut ini:

```
QSqlQuery query("SELECT * FROM mahasiswa");
while (query.next()) {
    QString nim = query.value(0).toString();
    qDebug() << nim;
}
```

Kita ingat bahwa tabel mahasiswa memiliki 3 kolom: NIM, Nama, dan IPK.

Perintah diatas menggunakan QSqlQuery yang menerima parameter QString. Setelah query dijalankan maka akan dilakukan proses looping untuk mengambil data-data per baris record dengan menggunakan method next() dari query. Di dalam looping kita mengambil variabel nim pada kolom pertama (dalam hal ini digunakan indeks 0). Untuk mengambil field tertentu pada tabel, misalnya ipk, maka hanya perlu mengganti indeksnya menjadi 2 saja.



Contoh Membaca data pada Sqlite.

Tulislah program berikut:

Listing 10.5: Membaca data pada Sqlite

```
1 #include <QtCore/QCoreApplication>
2 #include <QDebug>
3 #include <QtSql/QtSql>
4 int main(int argc, char *argv[])
5 {
6     QApplication a(argc, argv);
7     qDebug() << QSqlDatabase::drivers();
8     QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE")
9         ;
10    db.setDatabaseName("testmhs.db");
11    if(!db.open())
12    {
13        qDebug() << db.lastError();
14        qFatal( "Failed to connect." );
15    } else qDebug() << "Koneksi berhasil";
16    QSqlQuery query;
17    bool cek = query.exec("select nim,nama from mahasiswa
18        order by nim desc");
19    qDebug() << cek;
20    QString nim,nama;
21    while(query.next())
22    {
23        nim = query.value(0).toString();
24        nama = query.value(1).toString();
25        qDebug() << nim << " " << nama;
26    }
27    return a.exec();
28 }
```



Hasil:

```
("QSQLITE")
Koneksi berhasil
true
"11041""Wachid"
"12042""Arif"
"11012""Eko"
```

Keterangan:

- Program diatas melakukan koneksi ke SQLite dan kemudian mengirimkan query untuk mengambil data-data dari tabel mahasiswa dengan menggunakan query SQL select.
- Untuk mendapatkan hasil dari query kita gunakan perulangan dari variabel query dan method next(). Didalam perulangan kita ambil masing-masing nilai dari tiap-tiap kolom untuk setiap recordnya dengan menggunakan query.value().toString()
- Perintah diatas berarti kita mengambil indeks sesuai dengan kolom yang diambil dari SQL, yaitu kolom 0 untuk nim, 1 untuk nama dan seterusnya.

10.4.5 Menambah data pada tabel SQLITE

Cara menambah data pada tabel Sqlite adalah dengan menggunakan perintah query SQL INSERT. Sintaksnya adalah `INSERT INTO <namatabel> (<kolom1>, <kolom2>, dst) VALUES (<nilai1>, <nilai2>, dst)`. Perintah diatas tidak menghasilkan record sama sekali, namun dapat menghasilkan berapa jumlah record yang terpengaruh (affected rows) dan juga mengembalikan nilai bool yang akan bernilai true atau false. Nilai true jika menambahkan data berhasil, nilai false jika penambahan data gagal.

Pada Qt cara yang digunakan untuk menambah data adalah dengan menggunakan class QSqlQuery dan method query seperti berikut ini:



```
QSqlQuery query;
bool hasil = query.exec("INSERT INTO mahasiswa (nim,
    nama, ipk) VALUES
('22113344', 'anton', 3.4)");
if(hasil) qDebug() << "Berhasil"; else qDebug() << "gagal";
```

Perintah diatas menggunakan QSqlQuery yang menerima parameter sql dalam tipe data QString. Setelah query dijalankan maka akan diperiksa hasil dari akibat penambahan datanya. Jika berhasil maka akan mengembalikan nilai true, sedangkan jika gagal maka akan menghasilkan nilai false.

Contoh Menambahkan data pada SQLite.

Buatlah program berikut:

Listing 10.6: Menambahkan data pada SQLite

```
1 #include <QtCore/QCoreApplication>
2 #include <QDebug>
3 #include <QtSql/QtSql>
4 int main(int argc, char *argv[])
5 {
6     QApplication a(argc, argv);
7     qDebug() << QSqlDatabase::drivers();
8     QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE")
9         ;
10    db.setDatabaseName("testmhs.db");
11    if(!db.open())
12    {
13        qDebug() << db.lastError();
14        qFatal( "Failed to connect." );
15    } else qDebug() << "Koneksi berhasil";
16    QSqlQuery query;
```



```
16 bool hasil = query.exec("insert into mahasiswa(nim,  
17     nama,ipk) values  
18     ('22334455', 'mhs baru',3.12)");  
19 if(hasil) qDebug() << "Berhasil ditambahkan"; else  
20     qDebug() << "Gagal  
21 ditambahkan";  
22 return a.exec();  
23 }
```

Hasil:

```
("QLITE")  
Koneksi berhasil  
Berhasil ditambahkan
```

Keterangan:

- Pada awalnya data pada tabel mahasiswa hanya berjumlah 3 buah data, ketika program dijalankan maka data baru bernama “mhs baru” berhasil ditambahkan dan mengubah jumlah record pada tabel sehingga menjadi 4 buah. Tampilan perubahan adalah sebagai berikut:

| NIM | Nama | IPK |
|--------|--------|-----|
| 011041 | Wachid | 3.1 |
| 012042 | Arif | 3.6 |
| 011041 | Eko | 3.4 |
| 012011 | Alif | 3.3 |
| 013021 | Ananda | 3.8 |

- Cara menambahkan record pada SQLite sangat mudah, yaitu dengan menggunakan SQL insert into yang harus disesuaikan dengan jumlah field yang ada pada tabel. Setelah itu query akan dijalankan dengan method exec dari obyek QSqlQuery.
- Hasil kembalian dari method exec ini adalah bool yang menghasilkan nilai true atau false. Jika menghasilkan nilai true maka record berhasil ditambahkan, jika false maka record tidak berhasil ditambahkan!



- Jika program diatas dieksekusi sekali lagi (diulangi) maka akan menambahkan tulisan Gagal ditambahkan. Hal ini dikarenakan kita menambahkan record yang sama persis dengan ebelumnya padahal kita sudah menset bahwa field nim bersifat primary, yang artinya tidak boleh ada data nim yang kembar. Hal inilah yang menyebabkan data Gagal ditambahkan.

```
("QSLITE")
Koneksi berhasil
gagal ditambahkan
```

- Jika kita hendak membaca data pada SQLite, maka tambahkan kode berikut:

Listing 10.7: Membaca data pada SQLite

```
1 QSqlQuery query.exec("select nim,nama,ipk from
   mahasiswa order by nim desc");
2 QString nim,nama;
3 float ipk;
4 while(query.next())
5 {
6     nim = query.value(0).toString();
7     nama = query.value(1).toString();
8     ipk = query.value(2).toFloat();
9     qDebug() << nim << " " << nama << " " << ipk;
10 }
```

- Sehingga akan dihasilkan:

```
"012011"  "Alif"  "3.3"
"013021"  "Ananda" "3.8"
"011041"  "Wachid" "3.1"
"012042"  "Arif"   "3.6"
"011012"  "Eko"    "3.4"
```



10.4.6 Mengedit data pada tabel SQLITE

Cara mengedit data pada tabel Sqlite adalah dengan menggunakan perintah query SQL UPDATE. Sintaksnya adalah UPDATE <namabel> SET <kolom1>=<nilaikolom1>, <kolom2>=<nilaikolom2>, dst WHERE <kriteria>. Perintah diatas tidak menghasilkan record sama sekali, namun dapat menghasilkan berapa jumlah record yang terpengaruh (affected rows) dan juga mengembalikan nilai bool yang akan bernilai true atau false. Nilai true jika pengeditan data berhasil, nilai false jika pengeditan data gagal.

Pada Qt cara yang digunakan untuk mengedit data adalah dengan menggunakan class QSqlQuery dan method query seperti berikut ini:

Listing 10.8: menggunakan class QSqlQuery dan method query

```
1 QSqlQuery query;
2 bool hasil = query.exec("UPDATE mahasiswa SET nama='
3 Antonius Rachmat C' WHERE
4 nim='2200259'");
5 if(hasil) qDebug() << "Berhasil"; else qDebug() <<
6 "gagal";
```

Perintah diatas menggunakan QSqlQuery yang menerima parameter sql dalam tipe data QString. Setelah query dijalankan maka akan diperiksa hasil dari akibat pengeditan datanya. Jika berhasil maka akan mengembalikan nilai true, sedangkan jika gagal maka akan menghasilkan nilai false.

Contoh Mengedit data pada SQLite.

1. Kondisi awal tabel:



| NIM | Nama | IPK |
|--------|--------|-----|
| 011041 | Wachid | 3.1 |
| 012042 | Arif | 3.6 |
| 011041 | Eko | 3.4 |
| 012011 | Alif | 3.3 |
| 013021 | Ananda | 3.8 |

Kita akan mengedit nim 013021 menjadi bernama Aryo

2. Buatlah program berikut:

Listing 10.9: Mengedit data pada SQLite

```
1 #include <QtCore/QCoreApplication>
2 #include <QDebug>
3 #include <QtSql/QtSql>
4 #include <iostream>
5 using namespace std;
6 int main(int argc, char *argv[])
7 {
8     QApplication a(argc, argv);
9     qDebug() << QSqlDatabase::drivers();
10    QSqlDatabase db = QSqlDatabase::addDatabase("QSQLITE");
11    db.setDatabaseName("testmhs.db");
12    if(!db.open())
13    {
14        qDebug() << db.lastError();
15        qFatal( "Failed to connect." );
16    } else qDebug() << "Koneksi berhasil";
17    QSqlQuery query;
18    bool hasil = query.exec("update mahasiswa set nama
19        ='Antonius Rachmat C' where
20        nim='22002529'");
21    if(hasil) qDebug() << "Berhasil diedit"; else
22        qDebug() << "Gagal diedit";
```



```
21 qDebug() << "Jumlah record yang diedit: " << query  
     .numRowsAffected();  
22 return a.exec();  
23 }
```

3. Hasil:

```
("QSLITE")  
Koneksi berhasil  
Berhasil diedit  
Jumlah record yang diedit : 1
```

4. Kondisi akhir tabel:

| | | |
|--------|--------|-----|
| 013021 | Ananda | 3.8 |
| NIM | Nama | IPK |
| 011041 | Wachid | 3.1 |
| 012042 | Arif | 3.6 |
| 011041 | Eko | 3.4 |
| 012011 | Alif | 3.3 |

Keterangan:

- Program diatas masih sama menggunakan obyek QSqlQuery dan method exec(). Hanya SQL query nya saja yang berbeda dengan Contoh sebelumnya saat penambahan data. SQL query pada saat pengeditan menggunakan SQL UPDATE SET.
- Untuk mengetahui berapa jumlah record yang terupdate digunakan method numRowsAffected() dari obyek QSqlQuery.

10.4.7 Menghapus data pada tabel SQLITE

Cara menghapus data pada tabel Sqlite adalah dengan menggunakan perintah query SQL UPDATE. Sintaksnya adalah DELETE FROM WHERE <kriteria>. Perintah diatas tidak menghasilkan record sama sekali, namun dapat menghasilkan



berapa jumlah record yang terpengaruh (affected rows) dan juga mengembalikan nilai bool yang akan bernilai true atau false. Nilai true jika penghapusan data berhasil, nilai false jika penghapusan data gagal.

Pada Qt cara yang digunakan untuk menghapus data adalah dengan menggunakan class QSqlQuery dan method query seperti berikut ini:

Listing 10.10: menghapus data adalah dengan menggunakan class QSqlQuery dan method query

```
1 QSqlQuery query;
2 bool hasil = query.exec("DELETE FROM mahasiswa WHERE
3     nim='22334455');");
4 if(hasil) qDebug() << "Berhasil"; else qDebug << "
5     gagal";
```

Perintah diatas menggunakan QSqlQuery yang menerima parameter sql dalam tipe data QString. Setelah query dijalankan maka akan diperiksa hasil dari akibat penghapusan datanya. Jika berhasil maka akan mengembalikan nilai true, sedangkan jika gagal maka akan menghasilkan nilai false.

Contoh Menghapus data pada SQLite.

1. Kondisi awal tabel:

| NIM | Nama | IPK |
|--------|--------|-----|
| 011041 | Wachid | 3.1 |
| 012042 | Arif | 3.6 |
| 011041 | Eko | 3.4 |
| 012011 | Alif | 3.3 |
| 013021 | Ananda | 3.8 |

Kita akan menghapus data “Ananda”.

2. Buatlah program sebagai berikut:



Listing 10.11: Menghapus data pada SQLite

```
1 #include <QtCore/QCoreApplication>
2 #include <QDebug>
3 #include <QtSql/QtSql>
4 #include <iostream>
5 using namespace std;
6 int main(int argc, char *argv[])
7 {
8     QCoreApplication a(argc, argv);
9     qDebug() << QSqlDatabase::drivers();
10    QSqlDatabase db = QSqlDatabase::addDatabase(""
11        "QSQLITE");
12    db.setDatabaseName("testmhs.db");
13    if(!db.open())
14    {
15        qDebug() << db.lastError();
16        qFatal( "Failed to connect." );
17    } else qDebug() << "Koneksi berhasil";
18    QSqlQuery query;
19    bool hasil = query.exec("delete from mahasiswa
20        where nim='22334455'");
21    if(hasil) qDebug() << "Berhasil dihapus"; else
22        qDebug() << "Gagal dihapus";
23    qDebug() << "Jumlah record yang dihapus: " <<
24        query.numRowsAffected();
25    return a.exec();
26 }
```

3. Hasil:

```
("QSLITE")
Koneksi berhasil
Berhasil dihapus
Jumlah record yang dihapus : 1
```



4. Kondisi akhir tabel:

| NIM | Nama | IPK |
|--------|--------|-----|
| 011041 | Wachid | 3.1 |
| 012042 | Arif | 3.6 |
| 011041 | Eko | 3.4 |
| 012011 | Alif | 3.3 |

Keterangan:

Program diatas mampu menghapus data pada suatu record tertentu dengan menggunakan perintah SQL DELETE FROM. Program diatas tidak ada perubahan dari Contoh sebelumnya kecuali bagian perintah SQL nya.

Demikianlah kita sudah berlatih sejumlah manipulasi data pada tabel SQLite dengan menggunakan QSql. Pada databse lain misalnya MySQL semua perintah –perintah yang sudah dipelajari dapat digunakan dan hanya perlu disesuaikan pada bagian koneksi pada databasenya. Pemrograman basis data pada Qt termasuk mudah.

Pada bagian berikutnya kita akan mencoba membuat program untuk manipulasi data pada tabel mahasiswa dengan menggunakan menu. Pada menu akan ditampilkan beberapa pilihan seperti:

1. Tambah data
2. Tampil data
3. Hapus data
4. Cari nim
5. Edit data
6. Exit

Penjelasan:

- Menu 1 akan digunakan untuk menambah data mahasiswa
- Menu 2 akan digunakan untuk menampilkan data semua mahasiswa



- Menu 3 akan digunakan untuk menghapus data seorang mahasiswa berdasarkan nimnya
- Menu 4 akan digunakan untuk mencari data seorang mahasiswa berdasarkan nimnya
- Menu 5 akan digunakan untuk mengedit data seorang mahasiswa berdasarkan nimnya

Cara yang digunakan adalah dengan membuat sebuah class yang akan digunakan untuk mengakses semua fungsi yang berhubungan dengan manipulasi data pada basis data SQLite. Method pada class adalah connect untuk koneksi database, sebuah konstruktor dan method untuk mengambil nama tabel serta nama database SQlite-nya. Kemudian akan dibuat fungsi-fungsi lain diluar class yang digunakan untuk melakukan fungsi-fungsi sesuai dengan 5 fungsi yang didefinisikan pada menu. Untuk lebih jelasnya silahkan dicoba pada Contoh berikut ini.

Contoh Pembuatan manipulasi data pada SQLite dengan menggunakan menu

1. Tulislah program berikut:

Listing 10.12: Pembuatan manipulasi data pada SQLite dengan menggunakan menu

```
1 #include <QtCore/QCoreApplication>
2 #include <QDebug>
3 #include <QtSql/QtSql>
4 #include <iostream>
5 #include <conio.h>
6 using namespace std;
7 class Tabel{
8 private:
9     QString namadb;
10    QString namatabel;
```



```
11 QString strquery;
12 QSqlDatabase db;
13 public:
14 //konstruktor
15 Tabel(QString namadb, QString namatabel){
16 this->namadb = namadb;
17 this->namatabel = namatabel;
18 }
19 bool connect(){
20 this->db = QSqlDatabase::addDatabase("QSQLITE");
21 this->db.setDatabaseName(this->namadb);
22 if(!this->db.open())
23 {
24 qDebug() << "No";
25 return false;
26 } else {
27 qDebug() << "Yes";
28 return true;
29 }
30 }
31 bool jalanQuery(QString query){
32 this->strquery = query;
33 bool hasil = false;
34 if(this->db.isOpen()){
35 QSqlQuery myq(this->db);
36 hasil = myq.exec(this->strquery);
37 }
38 return hasil;
39 }
40 void ambilData(QString query){
41 this->strquery = query;
42 if(this->db.isOpen()){
43 QSqlQuery myq(this->db);
44 myq.exec(this->strquery);
```



```
45 QSqlRecord rec = myq.record();
46 int cols = rec.count();
47 QString temp;
48 for( int c=0; c<cols; c++ )
49 temp += rec.fieldName(c) + ((c<cols-1)?"\t":"");
50 qDebug() << temp;
51 while( myq.next() )
52 {
53 temp = "";
54 for( int c=0; c<cols; c++ )
55 temp += myq.value(c).toString() + ((c<cols-1)?"\t"
56 :"\n");
57 qDebug() << temp;
58 }
59 }
60 QString getNamaTabel(){
61 return this->namatabel;
62 }
63 QString getNamaDb(){
64 return this->namadb;
65 }
66 QSqlDatabase getDb(){
67 return this->db;
68 }
69 };
70 void tambahData(Tabel t){
71 string nim,nama,ipk;
72 getline(cin,nim);
73 cout << "NIM: "; getline(cin,nim);
74 cout << "Nama: "; getline(cin,nama);
75 cout << "IPK: "; getline(cin,ipk);
76 QString s = "insert into "+t.getNamaTabel()+""
values
```



```
77 ( '"+nim.c_str()+" , '"+nama.c_str()+" , "+ipk.c_str()
    ());
78 bool hasil = t.jalanQuery(s);
79 if(hasil) qDebug() << "Penambahan berhasil"; else
    qDebug() << "Penambahan gagal";
80 }
81 void hapusData(Tabel t){
82     string nim;
83     getline(cin,nim);
84     cout << "NIM yang akan dihapus: "; getline(cin,nim
    );
85 QString s = "delete from "+t.getNamaTabel()+""
    where nim='"+nim.c_str()+"'";
86 bool hasil = t.jalanQuery(s);
87 if(hasil) qDebug() << "Penghapusan berhasil"; else
    qDebug() << "Penghapusan
gagal";
88 }
89 void tampilData(Tabel t){
90     QString s = "select * from "+t.getNamaTabel()+""
        order by nim asc";
91     t.ambilData(s);
92 }
93 void cariNim(Tabel t){
94     string nim;
95     getline(cin,nim);
96     cout << "NIM yang akan dicari: "; getline(cin,nim)
    ;
97     QString s = "select * from "+t.getNamaTabel()+""
        where nim='"+nim.c_str()+"'";
98     t.ambilData(s);
99 }
100 void editData(Tabel t){
101     string nim,nama,ipk;
```



```
103 getline(cin,nim);
104 cout << "NIM yang akan diedit: "; getline(cin,nim)
105 ;
106 cout << "Nama baru: "; getline(cin,nama);
107 cout << "IPK baru: "; getline(cin,ipk);
108 QString s = "update "+t.getNamaTabel()+" set
109 nama='"+nama.c_str()+"', ipk='"+ipk.c_str()+"' where
110 nim='"+nim.c_str()+"'";
111 bool hasil = t.jalanQuery(s);
112 if(hasil) qDebug() << "Pengeditan berhasil"; else
113     qDebug() << "Pengeditan gagal";
114 }
115     int main(int argc, char *argv[])
116 {
117     QCoreApplication a(argc, argv);
118     int pil;
119     Tabel t("testmhs.db", "mahasiswa");
120     t.connect();
121     do {
122         system("cls");
123         cout << "MENU" << endl;
124         cout << "1. Tambah data\n";
125         cout << "2. Tampil data\n";
126         cout << "3. Hapus data\n";
127         cout << "4. Cari nim\n";
128         cout << "5. Edit data\n";
129         cout << "6. Exit\n";
130         cout << "Pilihan : "; cin >> pil;
131         cout << endl;
132         switch (pil) {
133             case 1:
134                 tambahData(t);break;
135             case 2:
136                 tampilData(t);break;
```



```
134 case 3:  
135 hapusData(t);break;  
136 case 4:  
137 cariNim(t);break;  
138 case 5:  
139 editData(t);  
140 }  
141 cout << "Tekan sembarang tombol..."; getch();  
142 } while (pil >=1 && pil<=5);  
143 cout << "Good bye";  
144 return a.exec();  
145 }
```

Hasil:

1. Tampilan menu:

```
MENU:  
1. Tambah data  
2. Tampil data  
3. Hapus data  
4. Cari nim  
5. Edit data  
6. Exit  
Pilihan:
```

2. Menu tambah data dan tampil data:



```
MENU:  
1. Tambah data  
2. Tampil data  
3. Hapus data  
4. Cari nim  
5. Edit data  
6. Exit  
Pilihan: 1  
NIM: 013012  
Nama: Ramadhani  
Penambahan berhasil  
Tekan sembarang tombol . . .
```

```
"NIM  Nama   IPK"  
"011041  Wachid  3.1"  
"012042  Arif   3.6 "  
"011041  Eko    3.4"  
"012011  Alif   3.3"  
"013012 Ramadhani  3.8"
```

3. Menu hapus data dan tampil data:

```
MENU:  
1. Tambah data  
2. Tampil data  
3. Hapus data  
4. Cari nim  
5. Edit data  
6. Exit  
Pilihan: 3  
NIM yang akan dihapus: 011041  
Penghapusan berhasil  
Tekan sembarang tombol . . .
```



```
"NIM  Nama  IPK"  
"012042  Arif  3.6 "  
"011041  Eko  3.4"  
"012011  Alif  3.3"  
"013012 Ramadhani  3.8"
```

4. Menu edit data dan tampil data:

Mengedit 011041 menjadi bernama Nur Wachid dan IPK menjadi 3.01

```
MENU:  
1. Tambah data  
2. Tampil data  
3. Hapus data  
4. Cari nim  
5. Edit data  
6. Exit  
Pilihan: 5  
NIM yang akan diedit: 011042  
Nama baru: Arif H  
IPK: 3.01  
Tekan sembarang tombol . . .
```

```
"NIM  Nama  IPK"  
"012042  Arif H  3.6 "  
"011041  Eko  3.4"  
"012011  Alif  3.3"  
"013012 Ramadhani  3.8"
```

5. Menu cari data:



MENU:

1. Tambah data
2. Tampil data
3. Hapus data
4. Cari nim
5. Edit data
6. Exit

Pilihan: 4

NIM yang akan dicari: 011042

"NIM Nama IPK"

"012042 Arif H 3.6 "

Tekan sembarang tombol . . .

6. Menu Exit

MENU:

1. Tambah data
2. Tampil data
3. Hapus data
4. Cari nim
5. Edit data
6. Exit

Pilihan: 6

Tekan sembarang tombol . . . Good bye

Keterangan:

- Program diatas merupakan program yang cukup banyak dan kompleks. Program ini dibuat dengan prinsip perulangan. Kita akan mengulang terus menerus bagian menu 1-6 sampai pengguna menginputkan angka yang bukan diantara 1-5. Jika pengguna memasukkan angka 1 maka dipanggil menu pertama, dan seterusnya. Jika pengguna memasukkan angka 6 maka program akan menampilkan Good Bye.
- Pada awal program kita membuat sebuah class bernama Tabel yang digunakan untuk mengelola data-data pada tabel mahasiswa. Class Tabel harus



diinisialisasi terlebih dahulu pada method int main() dengan menginputkan nama database dan nama tabelnya. Setelah diinisialisai maka class Tabel harus melakukan method connect agar database SQLite terbuka (open).

- Ketika menu penambahan data dipilih, maka method tambahData akan dipanggil dan membutuhkan parameter Tabel yang sudah dibuat dan diinisialisasi terlebih dahulu sebelumnya. Setelah itu berdasarkan obyek Tabel yang sudah dibuat, kita akan menggunakaninya untuk memasukkan data.
- Pada menu tambah, program akan meminta inputan nim, nama, dan ipk kepada pengguna. Setelah pengguna menginputkan data dengan lengkap, maka method tambahData() akan dipanggil sehingga data dapat masuk ke tabel. Proses memasukkan data dilakukan dengan menggunakan query INSERT.
- Demikian pula dengan menu tampil data, method yang digunakan sama pada contoh-contoh sebelumnya, namun ditambah dengan cara membaca kolom-kolom pada tabel dan menampilkan semua recordnya satu persatu dengan perulangan.
- Pada menu hapus data, perintah yang digunakan juga hanya mengubah query SQLnya.
- Pada menu cari data, kita juga menggunakan SQL select seperti pada menampilkan data. Perbedaannya hanyalah kondisi where yang digunakan. Pada menu pencarian data, kita mencari satu buah record mahasiswa saja berdasarkan nimnya.
- Pada menu edit data, kita juga menggunakan SQL update yang dapat digunakan untuk mengubah data pada tabel. Pada menu edit ini, kita mencari terlebih dahulu nim mahasiswa yang akan diedit baru mengeditnya.



TIPS:

Untuk mengkonversi dari tipe data string menuju ke QString, digunakan `<variabel string bias.c_str()` Perintah cin tidak bisa digunakan setelah fungsi getline, karena akan membuat



inputan menjadi bertumpuk seperti pada contoh ini:

```
int main(){
    int id, age;
    string name, address;
    cout<<"Enter ID : "; cin>>id;
    cout<<"Enter Name: "; getline(cin,name);
    cout<<"Enter Address : "; getline(cin,
        address);
    cout<<"Enter Age: "; cin>>age;
}
```

Hasil:

```
Enter ID: 23
Enter Name: Enter Address : Yogyakarta
Enter Age : 45
```

Terlihat bahwa Enter Name dan Enter Address tergabung dan menjadi satu. Untuk mencegahnya kita bisa menukar posisi bahwa cin diletakkan dibawah getline, seperti berikut:

```
int main(){
    int id, age;
    string name, address;
    cout<<"Enter Name: "; getline(cin,name);
    cout<<"Enter Address : "; getline(cin,
        address);
    cout<<"Enter ID : "; cin>>id;
    cout<<"Enter Age: "; cin>>age;
}
```

Sehingga tampilan:

```
Enter Name: Yanuar Adi
Enter Address : Yogyakarta
Enter ID: 23
Enter Age: 43
```



Jika cin tetap harus didahului sebelum getline, maka bisa dilakukan dengan cara:

```
int main(){
    int id, age;
    string name, address;
    cout<<"Enter ID : "; cin>>id;
    getline(cin,name);
    cout<<"Enter Name: "; getline(cin,name);
    cout<<"Enter Address : "; getline(cin,
        address);
    cout<<"Enter Age: "; cin>>age;
}
```

Sehingga tampilan:

```
Enter ID : 23
Enter Name : Yanuar Adi
Enter Address : Yogyakarta
Enter Age : 32
Terimakasih.
```



Bagian III

Interface



BAB 11

GUI

Contents

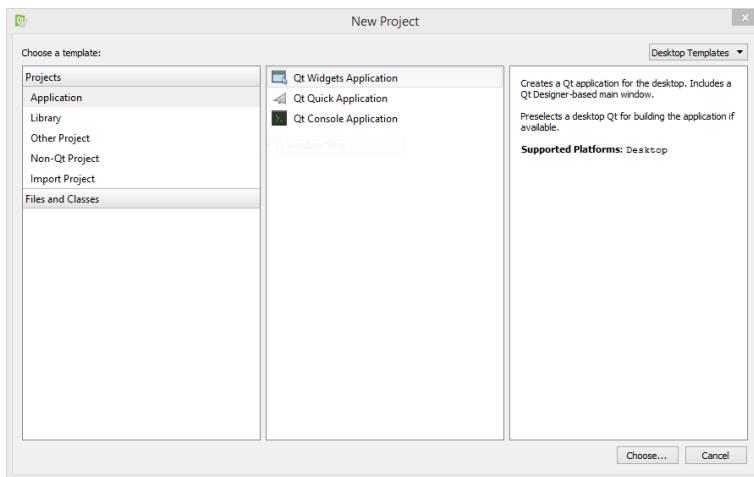
| | |
|--|------------|
| 11.1 Pemrograman aplikasi GUI | 371 |
| 11.1.1 Menyiapkan Proyek | 372 |
| 11.1.2 Membuat aplikasi | 375 |

11.1 Pemrograman aplikasi GUI

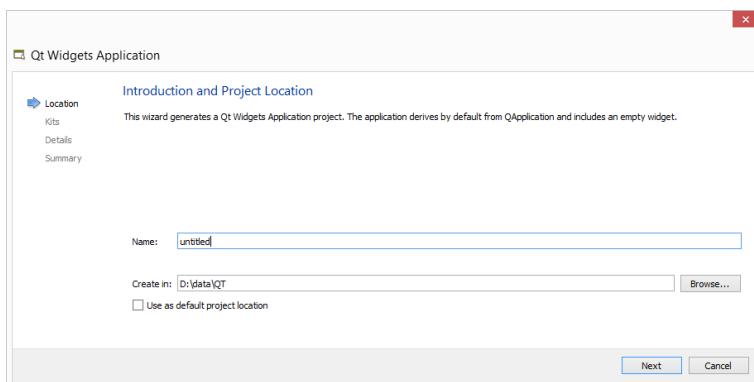
Tulisan ini ingin memamparkan bagaimana membuat sebuah aplikasi GUI sederhana dengan mudah menggunakan Qt Creator . Aplikasi sederhana dengan fungsi dasar tombol dan label (untuk gambar). Jika tombol pertama di klik, maka gamabr A muncul. Jika tombol kedua di klik, maka gambar B muncul. Dengan contoh dasar tersebut di harapakan mamou memahamai dasar-dasar dari sebuah pemrograman GUI di Qt Creator yakni signal dan slot.

11.1.1 Menyiapkan Proyek

1. Buka Qt Creator dan buat proyek baru. Pilih Application > Qt Widget Application¹.



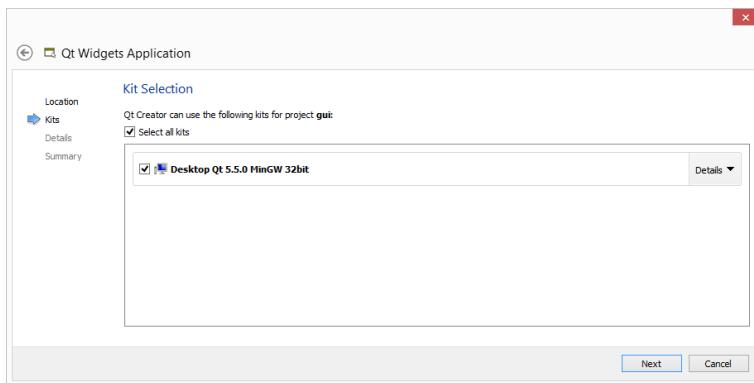
2. Beri nama Aplikasi yang akan di buat



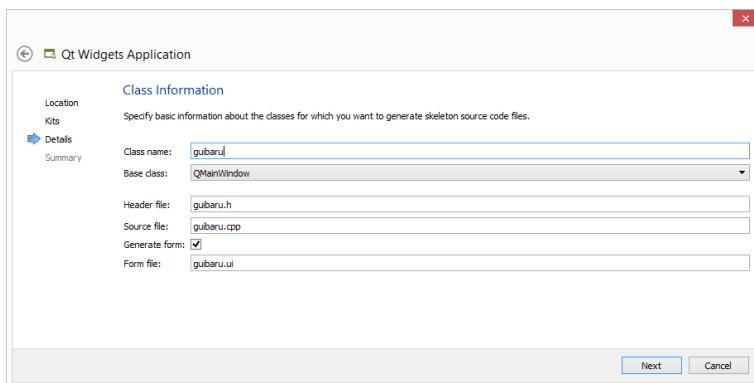
3. Pilih compiler yang di gunakan, jika mengginginkan aplikasi Cross Pla-

¹gunakan pilihan ini untuk membuat aplikasi GUI untuk proyek masa mendatang

tform, maka sebaiknya menggunakan MinGW sebagai compilernya, tapi jika ingin aplikasi full support terhadap windows maka gunakan MSC.



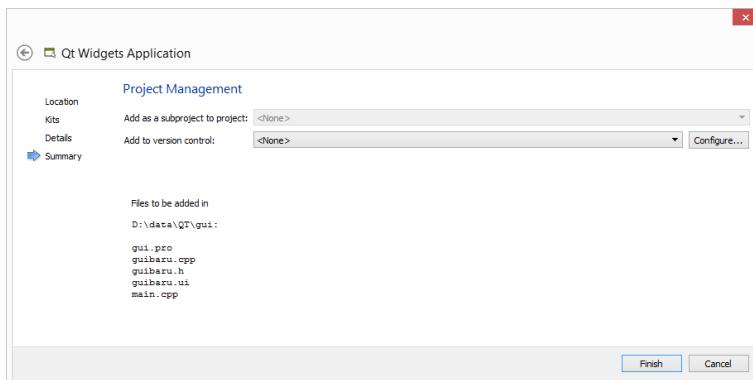
4. Pada Class Information berikan nama class yang di gunakan, disini penulis menggunakan guibaru.



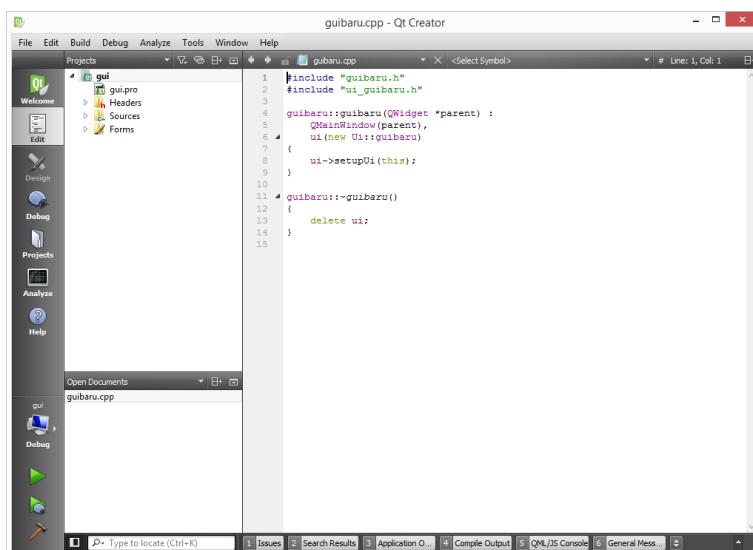
5. Kemudain langkah terakHIR adalah project management, silahkan gunakan subversion² yang Anda gunakan.

²Anda bisa menggunakan Git, subversion, mercurial, bazaar, ClearCase, Perfoce atau CVS





6. Finish, maka tampilan Qt Creator Anda akan seperti berikut ini



7. Perhatikan bahwa dalam proyek ini akan ada berkas yang terload secara otomatis oleh Qt creator yaitu

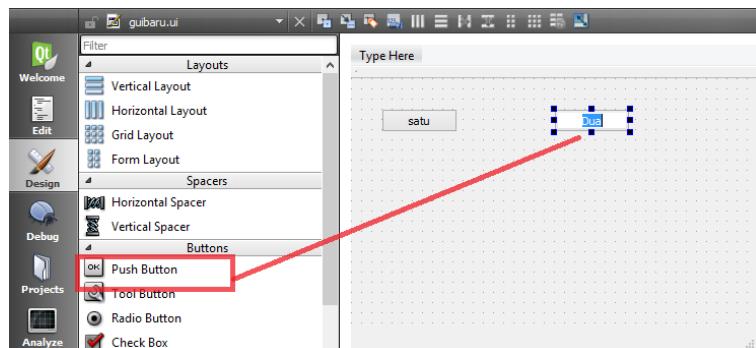
- `gui.pro` (berkas proyek Qt Creator yang bisa dipakai OS manapun)



- `guibaru.cpp` (berkas kode inti tempat signal dan slot dituliskan)
- `guibaru.h` (berkas header)
- `guibaru.ui` (berkas GUI)
- `main.cpp` (berkas berisi fungsi yang selalu ada dalam semua program C/C++ yakni `main()`)

11.1.2 Membuat aplikasi

1. Buka `guibaru.ui`. `guibaru.ui` merupakan berkas XML Qt Creator, jika di klik maka GUI builder milik Qt Creator akan muncul.
2. Drag and drop Push button ke dalam layer kemudian doble klik pada label tersebut untuk membuat nama baru.



3. Berikan nama objek untuk label ini property panel di samping kanan. di bagian atas terdapat QObject check. Tepat di bawahnya ada ObjectName, pada label PushButton ganti dengan img.

| Filter | |
|---------------|-------------------------------------|
| property | Value |
| objectName | img |
| enabled | <input checked="" type="checkbox"/> |
| geometry | [30, 30, 75 x 23] |
| sizePolicy | [Minimum, Fixed] |
| minimumSize | 0 x 0 |
| maximumSize | 16777215 x 16777215 |
| sizeIncrement | 0 x 0 |
| baseSize | 0 x 0 |
| palette | Inherited |
| font | A [MS Shell D...] |
| cursor | Arrow |
| mouseTracking | <input type="checkbox"/> |

4. Di bagian a

Bagian IV

Widget



BAB 12

File, Stream, dan XML

Agenda

Pada chapter ini kita akan membahas tentang beberapa class khusus pada Qt Framework yang digunakan untuk bekerja dengan File dan dokumen XML. Adapun materi yang akan dibahas pada HOL ini adalah:

Contents

| | |
|---|------------|
| 12.1 Bekerja dengan Paths | 380 |
| 12.2 Bekerja dengan Files | 384 |
| 12.3 Bekerja dengan Stream | 385 |
| 12.3.1 Text Stream | 386 |
| 12.3.2 Data Stream | 388 |
| 12.4 XML | 392 |
| 12.5 DOM | 393 |
| 12.5.1 Membuat File XML | 393 |
| 12.5.2 Membaca XML file | 395 |
| 12.5.3 Memodifikasi File XML | 398 |



12.1 Bekerja dengan Paths

QDir digunakan untuk bekerja dengan paths dan drives pada aplikasi Qt. QDir memiliki beberapa static method yang memudahkan anda bekerja dengan file sistem. Misal `QDir::current()` dapat digunakan untuk mengembalikan QDir dari direktori kerja anda, `QDir::home()` akan mengembalikan QDir dari home direktori pengguna, `QDir::root()` akan mengembalikan root direktori, dan `QDir::drives()` akan mengembalikan objek `QList<QFileInfo>` yang mewakili root dari semua drive yang ada. Objek `QFileInfo` menyimpan informasi tentang file dan direktori, ada beberapa method penting yang sering digunakan yaitu:

- `isDir()`, `isFile()`, dan `isSymbolicLink()` akan mengembalikan nilai true jika objek yang dicek berupa direktori, file, atau symbolic link (shortcut pada window).
- `dir()` dan `absoluteDir()` akan mengembalikan QDir yang mengandung informasi dari objek file. Method `dir()` akan mengembalikan direktori relatif dari direktori aktif, dan method `absoluteDir()` mengembalikan path direktori yang dimulai dari root.
- `exists()` akan menghasilkan nilai true jika objek tersebut ada.
- `isHidden()`, `isReadable()`, `isWritable()`, dan `isExecutable()` mengembalikan status hak akses dari file.
- `fileName()` akan mengembalikan `QString` berupa nama file tanpa path.
- `filePath()` akan mengembalikan `QString` berupa nama file beserta path, path dapat bersifat relatif terhadap direktori aktif.



- `absoluteFilePath()` akan mengembalikan `QString` berupa nama file beserta path, path diawali dari drive root.
- `completeBaseName()` and `completeSuffix()` akan mengembalikan `QString` berupa nama file dan ekstensi (suffix).

Program dibawah ini akan menunjukan cara penggunaan `QDir` untuk mengambil informasi direktori yang ada di komputer anda.

Menampilkan daftar drives dari root directories.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 12.1, kemudian tulis kode berikut.

Listing 12.1: Menampilkan daftar drives dari root directories.

```
1 #include <QtCore/QCoreApplication>
2 #include <QDebug>
3 #include <QDir>
4 #include <QFileInfo>
5 int main(int argc, char *argv[])
6 {
7     QCoreApplication a(argc, argv);
8     foreach (QFileInfo drive, QDir::drives()) {
9         qDebug() << "Drive : " << drive.absolutePath();
10        QDir dir = drive.dir();
11        dir.setFilter(QDir::Dirs);
12        foreach (QFileInfo rootDirs, dir.entryInfoList())
13            {
14                qDebug() << " " << rootDirs.fileName() ;
15            }
16    return a.exec();
17 }
```



2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.

```
Drive : "C:/"
    "MATLAB"
    "PerfLogs"
    "Program Files"
    "Program Files (x86)"
    "Qt"
    "system.sav"
    "Users"
    "Windows"
Drive : "D:/"
    "Music"
    "My Web Sites"
    "Photo"
    "Portable"
    "Referensi"
    "Software"
    "video"
    "webs"
```

Keterangan:

- Static method `QDir::drives()` akan mengembalikan collection berupa `QList<FileInfo>` yang berisi semua drive yang ada pada komputer.
- Untuk membaca semua drive beserta semua folder /direktori didalamnya anda dapat menggunakan foreach.
- Method `setFilter(QDir::Dirs)` artinya yang akan ditampilkan hanya direktori saja, tidak file atau symbolic link, untuk menampilkan semua (drive, direktori, file) anda dapat menggunakan `QDir::AllEntries`.



- Output dari program tersebut adalah daftar drive pada komputer anda berserta dengan folder yang ada didalamnya.

Dibawah ini adalah daftar kriteria yang dapat digunakan untuk melakukan filter.

- QDir::Dirs: Lists directories.
- QDir::AllDirs: Lists all directories.
- QDir::Files: Lists files.
- QDir::Drives: Lists drives.
- QDir::NoSymLinks: tidak menampilkan symbolic links.
- QDir::NoDotAndDotDot: tidak menampilkan special entries .
- QDir::AllEntries: Lists directories, files, drives, dan symbolic links.
- QDir::Readable: Lists readable files, harus dikombinasikan dengan Files atau Dirs.
- QDir::Writeable: Lists writable files. harus dikombinasikan dengan Files atau Dirs.
- QDir::Executable: Lists executable files. harus dikombinasikan dengan Files atau Dirs.
- QDir::Modified: Lists files yang sudah dimodifikasi.
- QDir::Hidden: Lists files yang sifatnya hidden.
- QDir::System: Lists system files.
- QDir::CaseSensitive: filter name harus case sensitive jika file sistem case sensitive.



12.2 Bekerja dengan Files

Anda dapat menggunakan QDir untuk mengambil informasi file dan QFileInfo untuk mengambil informasi file yang lebih lengkap, untuk proses manipulasi yang lebih jauh lagi seperti membuka, membaca, dan memodifikasi file anda harus menggunakan class QFile.

Agar lebih jelas bagaimana cara menggunakan QFile untuk membuka file, anda dapat mengerjakan Contoh dibawah ini.

Contoh Memeriksa apakah file ada dan bisa diakses.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 12.2, kemudian tulis kode berikut.

Listing 12.2: Memeriksa apakah file ada dan bisa diakses

```
1 #include <QtCore/QCoreApplication>
2 #include <QFile>
3 #include <QDebug>
4 int main(int argc, char *argv[])
5 {
6     QCoreApplication a(argc, argv);
7     QFile file("testfile.txt");
8     if(!file.exists())
9     {
10         qDebug() << "File : " << file.fileName() << "
11             tidak ditemukan";
12     }
13     if(!file.open(QIODevice::WriteOnly))
14     {
15         qDebug() << "Tidak dapat membuka file " << file.
16             fileName() << " untuk ditulis";
```



```
16 return a.exec();  
17 }  
18 qDebug("File berhasil dibuka !");  
19 file.close();  
20 return a.exec();  
21 }
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.

```
File : "testfile.txt" tidak ditemukan
```

3. Output diatas mempunyai arti bahwa anda belum membuat file dengan nama "testfile.txt", agar program diatas berhasil dijalankan buat file "testfile.txt" didalam folder Contoh build-12.2-Desktop_Qt_5_5_0_MinGW_32bit-Debug. Kemudian jalankan kembali programnya, maka akan tampil output berikut.

```
File berhasil dibuka !
```

Keterangan:

- Method `exists()` digunakan untuk mengecek apakah file yang ada atau tidak. Jika file tidak ditemukan maka program akan keluar dan menampilkan output program tidak ditemukan.
- Method `open()` digunakan untuk membuka file, permision yang digunakan pada program diatas adalah `WriteOnly`. Jika file tidak dapat dibuka maka akan keluar pesan tidak dapat membuka file.

12.3 Bekerja dengan Stream

Setelah anda membuka file akan lebih mudah untuk mengakses file tersebut menggunakan stream class. Qt hadir dengan 2 macam stream class, satu untuk teks file dan satu lagi untuk binary file. Dengan menggunakan stream untuk



mengakses file anda dapat menggunakan operator << dan >> untuk menulis dan membaca data dari file.

12.3.1 Text Stream

Untuk membuat text stream pada file, buat objek QFile dan buka file seperti biasa, disarankan jika anda menggunakan parameter QIODevice::Text dan hak akses QIODevice::ReadOnly. Agar lebih jelas bagaimana penggunaan stream buatlah contoh program pada Contoh 12.3 dibawah ini.

Menggunakan Stream untuk membaca file.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 12.3, kemudian tulis kode berikut.

Listing 12.3: Menggunakan Stream untuk membaca file

```
1 #include <QtCore/QCoreApplication>
2 #include <QDebug>
3 #include <QFile>
4 int main(int argc, char *argv[])
5 {
6     QCoreApplication a(argc, argv);
7     QFile file("D:\\sample.txt");
8     if(!file.exists())
9     {
10         qDebug() << "File " << file.fileName() << " tidak "
11             "ditemukan !";
12     }
13     if(!file.open(QIODevice::ReadOnly | QIODevice::Text))
14     {
```



```
15 qDebug() << "File " << file.fileName() << " tidak  
16     dapat diakses !";  
17 return a.exec();  
18 }  
19 QTextStream stream(&file);  
20 //membaca semua teks yang ada dalam sample.txt  
21 QString teks = stream.readAll();  
22 qDebug() << teks;  
23 //membaca teks per line  
24 while(!stream.atEnd())  
25 {  
26     QString line = stream.readLine();  
27     qDebug() << line;  
28 }  
29 file.close();  
30 return a.exec();  
31 }
```

2. Buat file teks pada alamat tertentu (pada contoh diatas di drive D:\\sample.txt), masukan sembarang teks kedalam file tersebut.
3. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.

Buku Pemrograman C++

Keterangan:

- Objek QTextStream digunakan jika anda ingin menggunakan stream untuk mengakses file text.
- Untuk membaca semua data yang ada pada file text gunakan method readAll().
- Untuk membaca file baris demi baris dapat digunakan method readLine() yang dijalankan didalam loop, method atEnd() digunakan untuk memerik-



sa apakah sudah sampai akhir file.

12.3.2 Data Stream

Pada beberapa kasus tertentu mungkin anda tidak dapat menggunakan file text untuk menyimpan data. Misalnya anda harus bekerja dengan file yang mempunyai format bukan teks, atau anda membutuhkan ukuran penyimpanan yang lebih kecil daripada menggunakan file teks. Dengan menyimpan data kedalam machine-readable seperti file biner ukuran file bisa menjadi lebih kecil dibandingkan dengan menyimpan data kedalam human-readable format seperti file teks.

Untuk membaca file biner anda dapat menggunakan objek QDataStream. Program dibawah ini menunjukkan penggunaan objek QDataStream untuk mengakses file biner.

Contoh Menggunakan Data Stream.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 12.5
2. Buka file Contoh 12.5.pro untuk menambahkan library GUI karena pada kontrol program ini digunakan class QColor.

Listing 12.4: file pro untuk membuka library Qt GUI

```
1 #-----  
2 #  
3 # Project created by QtCreator 2016-01-03T19:58:33  
4 #  
5 #-----  
6 QT += core  
7 QT += gui  
8 TARGET = Contoh 4
```



```
9 CONFIG += console  
10 CONFIG -= app_bundle  
11 TEMPLATE = app  
12 SOURCES += main.cpp
```

3. Kemudian tambahkan kode berikut pada file main.cpp.

Listing 12.5: Menggunakan Data Stream

```
1 #include <QtCore/QCoreApplication>  
2 #include <QDebug>  
3 #include <QDataStream>  
4 #include < QList >  
5 #include < QColor >  
6 #include < QFile >  
7 struct Warna  
8 {  
9     QString text;  
10    QColor color;  
11};  
12 QDataStream &operator << (QDataStream &stream,  
13                             const Warna &data)  
14 {  
15     stream << data.text << data.color;  
16     return stream;  
17 }  
18 QDataStream &operator >>(QDataStream &stream,  
19                           Warna &data)  
20 {  
21     stream >> data.text;  
22     stream >> data.color;  
23     return stream;  
24 }  
25 void saveList()  
26 {
```



```
25 QList<Warn list;
26 Warna data;
27 data.text = "Merah";
28 data.color = Qt::red;
29 list << data;
30 data.text = "Biru";
31 data.color = Qt::blue;
32 list << data;
33 data.text = "Kuning";
34 data.color = Qt::yellow;
35 list << data;
36 data.text = "Hijau";
37 data.color = Qt::green;
38 list << data;
39 QFile file( "datastream.dat" );
40 if( !file.open( QIODevice::WriteOnly ) )
41 return;
42 QDataStream stream( &file );
43 stream.setVersion( QDataStream::Qt_4_7);
44 stream << list;
45 file.close();
46 }
47 void loadList()
48 {
49 QList<Warn list;
50 QFile file( "datastream.dat" );
51 if( !file.open( QIODevice::ReadOnly ) )
52 return;
53 QDataStream stream(&file);
54 stream.setVersion(QDataStream::Qt_4_7);
55 stream >> list;
56 file.close();
57 foreach( Warna data, list )
58 qDebug() << data.text << "("
```



```
59 << data.color.red() << ","  
60 << data.color.green() << ","  
61 << data.color.blue() << ")";  
62 }  
63 int main(int argc, char *argv[]){  
64 {  
65 QCoreApplication a(argc, argv);  
66 saveList();  
67 loadList();  
68 return a.exec();  
69 }
```

4. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.

content

Keterangan:

- Pada program diatas struct Warna adalah user define type yang dibuat sendiri, struct Warna memiliki dua member variabel yaitu text yang bertipe QString dan color yang bertipe QColor.
- Untuk memasukan data biner (dengan tipe data Warna) kedalam stream buat operator <<
- Untuk mengambil data biner dari stream buat operator >>
- Method saveList() digunakan untuk membuat objek warna , memasukan objek tersebut kedalam list dan menuliskannya kedalam file datastream.dat.
- Method loadList() digunakan untuk mengambil data stream dari file datastream.dat dan menampilkan data tersebut dengan cara membaca dari list.



12.4 XML

XML adalah meta-language yang dapat digunakan untuk menyimpan data tersusunan berupa string atau teks file. Komponen dasar penyusun dokumen XML adalah tag, attribute, dan teks. Contoh dokumen XML yang sederhana ditunjukkan pada listing dibawah ini.

```
1 <document name="DocName">
2 <author name="AuthorName" />
3 Some text
4 </document>
```

Pada dokumen XML diatas document tag mengandung author tag dan teks. Tag document diawali dengan `<document>` dan diakhiri dengan closing tag `</document>`. Kedua tag document dan author memiliki attribute yang sama yaitu name.

Author tag tidak memiliki tag penutup karena tidak memiliki elemen lain didalamnya, cara penulisannya adalah `<author/>`, ini sama dengan menuliskan `<author></author>`.

Qt mendukung tiga cara untuk memanipulasi dokumen XML yaitu QStreamReader, DOM, dan SAX.

Untuk menggunakan library XML pada Qt anda harus menambahkan library XML pada Qt project file.

```
1 #-----
2 #
3 # Project created by QtCreator 2016-01-03T21:43:04
4 #
5 #-----
6 QT += core
7 QT -= gui
8 QT += xml
9 TARGET = Contoh 5
```



```
10 CONFIG += console  
11 CONFIG -= app_bundle  
12 TEMPLATE = app  
13 SOURCES += main.cpp
```

12.5 DOM

DOM (Document Object Model) bekerja dengan cara merepresentasikan semua dokumen XML menjadi bentuk tree yang mempunyai node dan menyimpanya dalam memory.

12.5.1 Membuat File XML

Pertama kita akan mulai dengan membuat file XML menggunakan DOM, adapun tahapan yang akan kita kerjakan adalah membuat node, menyambungkan node, dan terakhir menulis dokumen XML. Agar lebih jelas mari kita coba buat program dibawah ini.

Contoh Membuat Nodes untuk membuat simple XML Document.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 12.6, kemudian tulis kode berikut.

Listing 12.6: Membuat Nodes untuk membuat simple XML Document

```
1 #include <QtCore/QCoreApplication>  
2 #include <QDebug>  
3 #include <QFile>  
4 #include <QTextStream>  
5 #include <QDomDocument>  
6 #include <QDomElement>  
7 #include <QDomText>
```



```
8 int main(int argc, char *argv[])
9 {
10 QCoreApplication a(argc, argv);
11 QDomDocument dokumen;
12 QDomElement mhs = dokumen.createElement("Mahasiswa");
13     ");
14 mhs.setAttribute("Jurusan", "TI");
15 QDomElement nim = dokumen.createElement("Nim");
16 QDomElement ipk = dokumen.createElement("Ipk");
17 QDomText nimtext = dokumen.createTextNode(
18     "22002321");
19 QDomText ipktext = dokumen.createTextNode("3.5");
20 dokumen.appendChild(mhs);
21 mhs.appendChild(nim);
22 nim.appendChild(nimtext);
23 mhs.appendChild(ipk);
24 ipk.appendChild(ipktext);
25 QFile file("simple.xml");
26 if(!file.open(QIODevice::WriteOnly | QIODevice::
27             Text))
28 {
29     qDebug() << "File tidak ditemukan !";
30     a.exit(-1);
31     return a.exec();
32 }
33 QTextStream stream(&file);
34 stream << dokumen.toString();
35 qDebug() << "File XML berhasil dibuat ..";
36 file.close();
37 return a.exec();
38 }
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.



content

Hasil dari file XML “simple.xml” yang berhasil dibuat adalah :

```
1 <Mahasiswa Jurusan="TI">
2 <Nim>22002321</Nim>
3 <Ipk>3.5</Ipk>
4 </Mahasiswa>
```

Keterangan:

- Untuk membuat dokumen XML menggunakan DOM, pertama buat objek QDomDocument .
- Langkah selanjutnya adalah membuat objek QDomElement untuk membuat element Nim dan Ipk.
- Untuk menambahkan text pada element tambahkan objek QDomText.
- Setelah element dan text selesai dibuat, anda dapat memasangkan element dan teks menjadi child node dengan menggunakan method appendChild().
- Setelah dokumen XML selesai dibuat, anda dapat menuliskan dokumen tersebut ke file teks dengan menggunakan objek QTextStream, pada contoh program diatas dokumen XML disimpan dalam file “simple.xml”

12.5.2 Membaca XML file

Pada contoh sebelumnya anda telah bisa membuat dokumen XML dengan DOM, sekarang kita akan mencoba membaca dokumen XML yang sebelumnya sudah dibuat.

Pada contoh dibawah ini kita akan membaca dan memasukan file kedalam QDomDocument dan kita juga akan mempelajari bagaimana cara menemukan elemen dan teks yang terkandung dalam dokumen. Agar dokumen XML pada file dapat diload kedalam QDomDocument maka dokumen XML tersebut harus valid.



Contoh Membaca DOM dari dokumen XML.

1. Untuk dokumen XML yang akan digunakan pada contoh program ini adalah dokumen XML yang sudah kita buat sebelumnya yaitu “simple.xml”.
2. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 12.6.
3. Jalankan dahulu program tersebut dengan menekan tombol Ctrl + R, agar folder simulator dengan nama Contoh build-12.7-Desktop_Qt_5_5_0_MinGW_32bit-Debug dibuat.
4. Kopikan file “simple.xml” yang akan dibaca kedalam folder Contoh build-12.7-Desktop_Qt_5_5_0_MinGW_32bit-Debug.
5. Kemudian buka file main.cp, tulis kode untuk membaca file XML berikut:

Listing 12.7: Contoh Membaca DOM dari dokumen XML

```
1 #include <QtCore/QCoreApplication>
2 #include <QFile>
3 #include <QTextStream>
4 #include <QDomDocument>
5 #include <QDomElement>
6 #include <QDomText>
7 #include <QDebug>
8 int main(int argc, char *argv[])
9 {
10     QCoreApplication a(argc, argv);
11     QFile file("simple.xml");
12     if(!file.open(QIODevice::ReadOnly | QIODevice::Text))
13     {
14         qDebug() << "File " << file.fileName() << " tidak
15         ditemukan";
16     }
17 }
```



```
17 QDomDocument dokumen;
18 if(!dokumen.setContent(&file))
19 {
20 qDebug() << "Gagal untuk parsing ke DOM tree";
21 file.close();
22 return a.exec();
23 }
24 QDomElement dokumenElemen = dokumen.
25     documentElement();
26 QDomNode node = dokumenElemen.firstChild();
27 while(!node.isNull())
28 {
29 if(node.isElement())
30 {
31 QDomElement element = node.toElement();
32 qDebug() << "Element " << element.tagName();
33 qDebug() << "Atribut nama " << element.attribute(" 
34     nama", "tidak ada
35 attribute");
36 }
37 if(node.isText())
38 {
39 QDomText teks = node.toText();
40 qDebug() << teks.data();
41 }
42 return a.exec();
43 }
```

6. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.

content



Keterangan:

- Untuk membaca data dari file, seperti biasa gunakan objek QFile dan jalankan method open().
- Untuk mengambil data yang ada file untuk dimasukan kedalam objek QDomDocument gunakan method setContent().
- Untuk mengakses data elemen pada QDomDocument, buat objek QDomElement.
- Untuk mengambil node dari QDomElement, buat objek QDomNode. Method firstChild() digunakan untuk mengambil node awal pada QDomElement.
- Kemudian jika node yang dibaca tidak null, cek apakah node tersebut berupa elemen, jika node tersebut element lakukan konversi ke objek QDomElement dan gunakan method tagName() untuk menampilkan nama element dan method attribute() untuk menampilkan atribut jika ada.
- Cek juga apakah node bertipe teks dengan method isText(), jika teks konversi node kedalam objek QDomText kemudian tampilkan isi dari teks dengan menggunakan method toText().
- Untuk berpindah ke node selanjutnya gunakan method nextSibling().

12.5.3 Memodifikasi File XML

Pada topik sebelumnya kita sudah membahas bagaimana cara menulis dan membaca data dari dokumen XML. Pada topik kali ini akan dibahas bagaimana cara memodifikasi dokumen XML yang sudah anda load kedalam objek QDomDocument.



Contoh Modifikasi data dokumen XML.

1. Buat project console application baru dengan nama Contoh 12.9, pilih Qt Simulator sebagai simulator yang digunakan. Jalankan program sehingga akan dibuat folder Contoh build-12.9-Desktop_Qt_5_5_0_MinGW_32bit-Debug.
2. Pada folder Contoh 12.9-build-simulator buat file dengan nama “simple.xml”, kemudian tulis dokumen XML berikut.

```
1 <dokumen nama="Data Mahasiswa">
2 <Mahasiswa Jurusan="TI">
3 <Nim nama="Erick">22002321</Nim>
4 <Ipk>3.5</Ipk>
5 </Mahasiswa>
6 <Mahasiswa Jurusan="SI">
7 <Nim nama="Katon">23002333</Nim>
8 <Ipk>3.6</Ipk>
9 </Mahasiswa>
10 </dokumen>
```

3. Kemudian pada file main.cpp tambahkan kode berikut untuk melakukan modifikasi file xml yang sudah kita buat sebelumnya.

Listing 12.8: Main program Modifikasi data dokumen XML

```
1 #include <QtCore/QCoreApplication>
2 #include <QDebug>
3 #include <QFile>
4 #include <QTextStream>
5 #include <QDomDocument>
6 #include <QDomElement>
7 #include <QDomText>
8 int main(int argc, char *argv[])
9 {
10     QCoreApplication a(argc, argv);
```



```
11 QFile fileAsli("simple.xml");
12 if(!fileAsli.open(QIODevice::ReadOnly | QIODevice
13     ::Text))
14 {
15     qDebug() << "File " << fileAsli.fileName() << "
16         tidak ditemukan";
17     return a.exec();
18 }
19 QDomDocument dokumen;
20 if(!dokumen.setContent(&fileAsli))
21 {
22     qDebug() << "Gagal parsing file ke DOM tree";
23     fileAsli.close();
24     return a.exec();
25 }
26 fileAsli.close();
27 QDomElement elemenDokumen = dokumen.
28     documentElement();
29 QDomNodeList elemen = elemenDokumen.
30     elementsByTagName("Mahasiswa");
31 if(elemen.size() == 0)
32 {
33     QDomElement mhs = dokumen.createElement("Mahasiswa
34         ");
35     elemenDokumen.insertBefore(mhs,QDomNode());
36 }
37 else
38 {
39     QDomElement mhs = elemen.at(0).toElement();
40     QDomElement nama = dokumen.createElement("Nama");
41     QDomText textNama = dokumen.createTextNode("Erick
42         Kurniawan");
43     nama.appendChild(textNama);
44     mhs.appendChild(nama);
```



```
39 }
40 QFile fileModif("simplemodif.xml");
41 if(!fileModif.open(QIODevice::WriteOnly |
42     QIODevice::Text))
43 {
44     qDebug() << "Gagal untuk membaca file xml";
45     return a.exec();
46 }
47 QTextStream stream(&fileModif);
48 stream << dokumen.toString();
49 qDebug() << "File berhasil dimodifikasi dan
50     disimpan pada file simplemodif.xml";
51 fileModif.close();
52 //membaca isi dari file simplemodif.xml
53 qDebug() << "Setelah dimodifikasi isi dari
54     simplemodif.xml adalah";
55 if(!fileModif.open(QIODevice::ReadOnly | QIODevice
56     ::Text))
57 {
58     qDebug() << "Gagal membaca file xml";
59     return a.exec();
60 }
```

4. Kemudian jalankan program diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.

content

5. Untuk melihat hasil dari file xml yang sudah dimodifikasi, anda dapat membuka file “simplemodif.xml” yang ada pada folder Contoh build-12.9-Desktop_Qt_5_5_0_MinGW_32bit-Debug.



6. Isi dari file “simplemodif.xml” adalah sebagai berikut

```
1 <dokumen nama="Data Mahasiswa">
2 <Mahasiswa Jurusan="TI">
3 <Nim nama="Erick">22002321</Nim>
4 <Ipk>3.5</Ipk>
5 <Nama>Erick Kurniawan</Nama>
6 </Mahasiswa>
7 <Mahasiswa Jurusan="SI">
8 <Nim nama="Katon">23002333</Nim>
9 <Ipk>3.6</Ipk>
10 </Mahasiswa>
11 </dokumen>
```

content

Keterangan:

- Langkah pertama baca file simple.xml yang akan dimodifikasi, kemudian masukan isinya kedalam QDomDocument.
- Ambil root element dari dokumen menggunakan method documentElement() dan masukan kedalam objek bertipe QDomElement.
- Kode QDomNodeList elemen = elemenDokumen.elementsByTagName("Mahasiswa"); digunakan untuk mengambil semua element yg mempunyai tag <Mahasiswa dan memasukannya kedalam variabel elemen yang bertipe QDomNodeList.
- Kode if(elemen.size() == 0) digunakan untuk memeriksa apakah elemen dengan tag <Mahasiswa ditemukan, jika tidak ditemukan maka buat elemen <Mahasiswa, jika ditemukan ambil elemen <Mahasiswa pertama kemudian tambahkan elemen baru <Nama beserta teks di dalam elemen <Mahasiswa tersebut.
- Langkah terakhir adalah membuat QTextStream dan menyimpan dokumen XML yang sudah dimodifikasi kedalam file simplemodif.xml.



12.6 QXMLStream Reader

Selain menggunakan DOM untuk membaca dokumen XML, anda juga dapat menggunakan objek QXMLStreamReader. QXMLStreamReader adalah class parser XML tercepat dan termudah untuk digunakan, karena QXMLStreamReader parser bekerja secara incremental sehingga mempermudah pembacaan tag. QXMLStreamReader juga cocok digunakan untuk membaca file yang berukuran besar yang tidak cocok jika disimpan di memory. Beberapa token yang digunakan adalah seperti tabel ??

| Token Type | Contoh | Getter Fuctions |
|-----------------------|----------------|--|
| StartDocument | N/A | isStandaloneDocument() |
| EndDocument | N/A | isStandaloneDocument() |
| StartElement | <item> | namespaceUri() , name() , attributes() , namespaceDeclarations() |
| EndElement | </item> | namespaceUri() , name() |
| Characters | AT
 | text() , isWhitespace() , isCDA-TA() |
| Comment | <!– fix – > | text() |
| DTD | <!DOCTYPE ...> | text() , notationDeclarations() , entityDeclarations() |
| EntityReference | ™ | name() , text() |
| ProcessingInstruction | <?alert?> | processingInstructionTarget() , processingInstructionData() |
| Invalid | >&<! | error() , errorString() |

Misal anda memiliki dokumen XML sebagai berikut

```

1 <doc>
2 <quote>Einmal ist keinmal</quote>
3 </doc>

```

Setiap anda menggunakan method readNext() maka akan dibaca satu token. Untuk kode diatas kita dapat membaca pertoken.



```
StartDocument
StartElement (name() == "doc")
StartElement (name() == "quote")
Characters (text() == "Einmal ist keinmal")
EndElement (name() == "quote")
EndElement (name() == "doc")
EndDocument
```

Setelah memanggil method `readNext()`, anda dapat mengecek token yang sedang aktif dengan menggunakan `isStartElement()`, `isCharacter()`, atau menggunakan fungsi yang mirip seperti `state()`.

Contoh Menggunakan QXMLStream Reader untuk membaca XML.

1. Buat aplikasi console baru dengan nama Contoh 12.9, pilih Qt Simulator untuk menampilkan outputnya.
2. Buat file simple.xml dan buat dokumen XML sebagai berikut

```
1 <Mahasiswa>
2 <Mahasiswa>
3 <Nim nama="Erick">22002321</Nim>
4 <Ipk>3.5</Ipk>
5 </Mahasiswa>
6 <Mahasiswa>
7 <Nim nama="Katon">23002333</Nim>
8 <Ipk>3.6</Ipk>
9 </Mahasiswa>
10 </Mahasiswa>
```

3. Kemudian ketikan kode berikut pada main.cpp untuk membaca data dari dokumen XML.

Listing 12.9: Menggunakan QXMLStream Reader untuk membaca XML

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



```
2 #include <QDebug>
3 #include <QFile>
4 #include <QXmlStreamReader>
5 int main(int argc, char *argv[])
6 {
7     QCoreApplication a(argc, argv);
8     QFile file("simple.xml");
9     if(!file.open(QIODevice::ReadOnly | QIODevice::Text))
10    {
11         qDebug() << "File tidak ditemukan ";
12         return a.exec();
13    }
14    QXmlStreamReader reader;
15    reader.setDevice(&file);
16    while(!reader.atEnd())
17    {
18        reader.readNext();
19        if(reader.isStartElement())
20        {
21            qDebug() << reader.name();
22            if(reader.name() == "Nim")
23            {
24                qDebug() << "Nama Attribute :" << reader.
25                    attributes().value("nama");
26                qDebug() << "Teks : " << reader.readElementText();
27            }
28        }
29        file.close();
30        return a.exec();
31    }
```

4. Jalankan aplikasi tersebut, maka akan ditampilkan output sebagai berikut.



content**Keterangan:**

- Untuk membaca dokumen XML menggunakan QXMLStreamReader anda dapat melakukan looping dengan memeriksa apakah sudah sampai pada akhir dokumen while(!reader.atEnd()).
- Method reader.readNext(); digunakan untuk berpindah token.
- Method reader.isStartElement() digunakan untuk mengecek apakah token yg sekarang aktif adalah elemen awal.
- Kode if(reader.name() == “Nim”) digunakan untuk mengecek apakah nama elemen adalah “Nim” jika ya baca attribute dan teks pada elemen tersebut untuk ditampilkan.

Contoh Membuat dokumen XML dengan QXMLStreamWriter.

1. Buat aplikasi console dengan nama Contoh 12.10, kemudian tulis kode berikut

Listing 12.10: Membuat dokumen XML dengan QXMLStreamWriter

```
1 #include <QtCore/QCoreApplication>
2 #include <QDebug>
3 #include <QFile>
4 #include <QXmlStreamWriter>
5 int main(int argc, char *argv[])
6 {
7     QApplication a(argc, argv);
8     QFile file("simple.xml");
9     if(!file.open(QIODevice::WriteOnly | QIODevice::Text))
10    {
11        qDebug() << "File tidak ditemukan..";
12        return a.exec();
13    }
14 }
```



```
13 }
14 QXmlStreamWriter writer(&file);
15 writer.setAutoFormatting(true);
16 writer.writeStartDocument();
17 writer.writeStartElement("Books");
18 writer.writeStartElement("Book");
19 writer.writeStartElement("Author");
20 writer.writeAttribute("Name", "Erick Kurniawan");
21 writer.writeAttribute("Title", "Qt Programming");
22 writer.writeEndDocument();
23 file.close();
24 qDebug() << "File sudah berhasil dibuat !";
25 return a.exec();
26 }
```

2. Jalankan program diatas untuk menggenerate dokumen XML dengan nama simple.xml.

```
File sudah berhasil di buat !
```

3. Isi dari dokumen XML yang barusan anda buat adalah sebagai berikut.

```
1 <Books>
2 <Book>
3 <Author name="Erick Kurniawan" Title="ASP.NET 3.5"
   />
4 </Book>
5 </Books>
```

Keterangan:

- Anda dapat menggunakan QXMLStreamWriter untuk membuat dokumen XML.
- Kode writer.setAutoFormatting(true); digunakan untuk memformat secara otomatis dokumen XML yang akan dibuat, misal menambahkan line break



dan indentation pada bagian yang kosong pada elemen. Tujuan utamanya adalah memisahkan data menjadi beberapa baris sehingga membantu dalam pembacaan dokumen.

- Kode writer.writeStartDocument(); digunakan pada saat pertama kali dokument akan dibuat.
- Kode writer.writeStartElement("Books"); digunakan untuk menuliskan elemen Books.
- Kode writer.writeAttribute("Name","Erick Kurniawan"); digunakan untuk menambahkan attribute pada elemen tertentu.
- Kode writer.writeEndDocument(); digunakan untuk menutup semua start element yang sebelumnya dibuat dengan method writeStartElement().



BAB 13

Qt Webkit

Agenda

Pada Bab ini kita akan mempelajari tentang module bawaan Qt Creator yaitu Qt Webkit seperti berikut ini

Contents

| | |
|--|------------|
| 13.1 Qt Webkit | 410 |
| 13.2 My Web Browser version 2 | 413 |
| 13.2.1 Starting the Project | 413 |
| 13.2.2 Working on UI | 416 |
| 13.3 Writing Code for the Slots | 418 |
| 13.3.1 Running the Code | 420 |
| 13.3.2 Source Code | 420 |



13.1 Qt Webkit

Pada contoh ini kita akan membuat sebuah browser dengan menggunakan Qt Webkit

pertama,kita akan mencoba memuat sebuah url untuk menampilkan halaman kemudian First, we'll just try to load a url to display a web page, then start to build the more refined browser.

In my opinion, one of the most important pieces of Qt Webkit is QWebView. The Qt document also says:

Qwebview merupakan komponen widget utama dari module web browser Qtwebkit yang dapat digunakan dalam beberapa aplikasi untuk menampilkan konten pada internet

Contoh membuat browser dengan Qt Webkit dengan meload url

Sebuah website dapat di muat kedalam sebuah browser dengan menggunakan Qwebview dengan fungsi load() fungsi ini merupakan bagian core dari peting untuk menampilkan konten statis pada browser

```
QWebView *view = new QWebView(parent);  
view->load(QUrl("http://google.com/"));  
view->show();
```

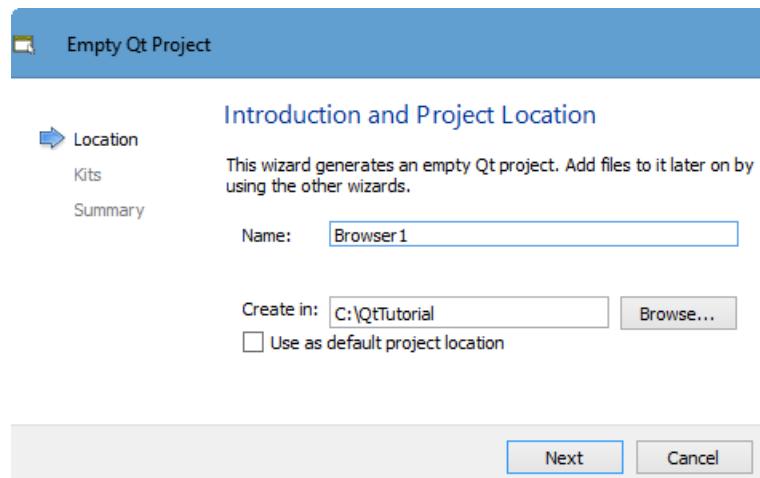
We make a QWebView object, and load a page using load(). Like all Qt widgets, to display QWebView, the show() function must be invoked.

To make it work, we need a module from webkit (actually, webkitwidgets), and we will set it in .pro file.

Though it's not necessary for this simple project, we'll use Creator.



File->New File or Project...->Other Project->Empty Qt Project->Choose...



Gambar 13.1:

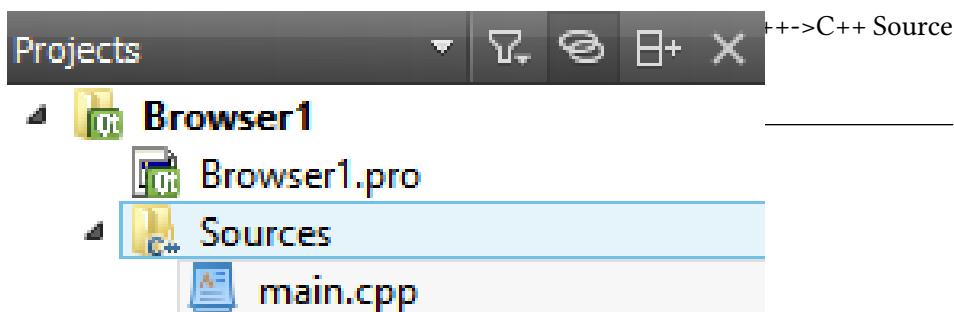
Paste the following lines to the Browser1.pro:

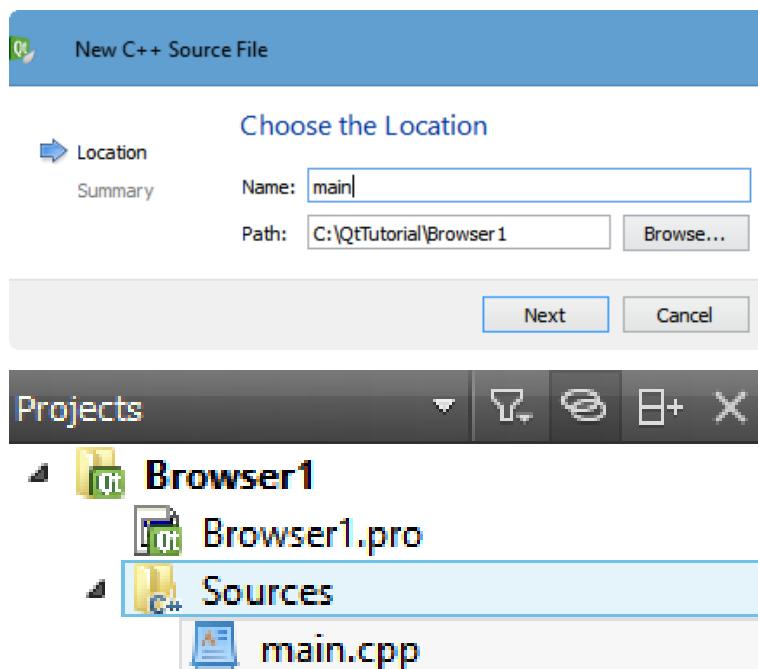
```
QT      += core gui
QT      += webkitwidgets
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = Browser1
TEMPLATE = app

SOURCES += main.cpp
```

Then we need a `main()`.





Copy the following code for the main.cpp:

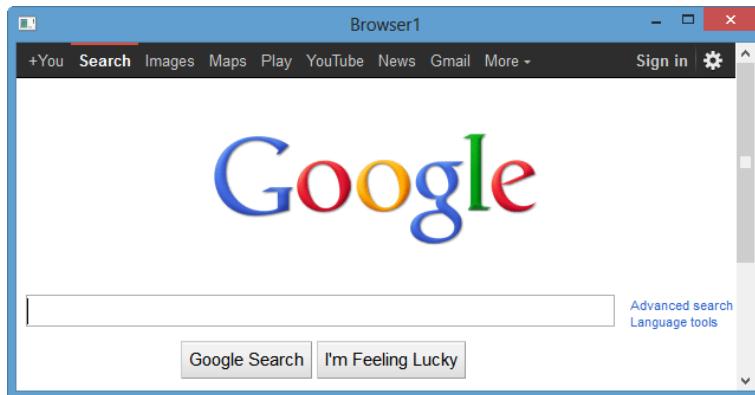
```
#include <QApplication>
#include <QWebView>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QWebView view;
    view.show();
    view.load(QUrl("http://google.com"));

    return a.exec();
}
```



Run qmake->Run, then we'll get our browser with the page loaded:



Gambar 13.2:

NOTE: The linking against the Webkit does not seem to be needed. So, I dropped the following line from the Browser1.pro file.

```
QT      += webkit
```

13.2 My Web Browser version 2

Though our browser version 1 is able to display a page, it does not have controls such as backward, forward, or refresh etc.

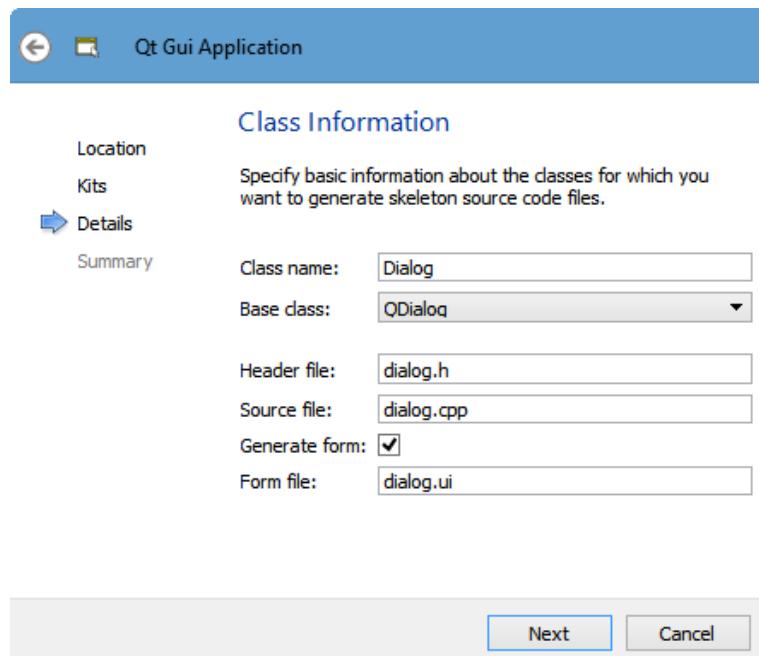
So, in the next tutorial, we'll put some UI elements so that our browser become more closer to the real one.

13.2.1 Starting the Project

Right click on the Project name and select Add new...->Applications->Qt Gui Application->Choose...



(Note) Depending on the version, we may want to select Add new...->Qt->Qt Designer Form Class->Dialog without Buttons



Gambar 13.3:

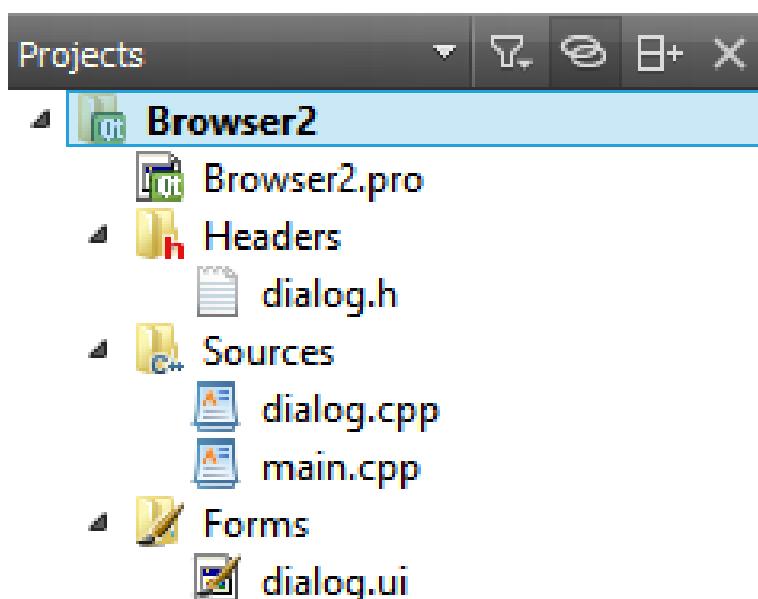
.pro File To link against Webkit, we need to add one line to the .pro file (with Qt5.3, the line will be added automatically):

```
QT      += core gui  
QT      += webkitwidgets
```

```
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
```

```
TARGET = Browser2  
TEMPLATE = app
```



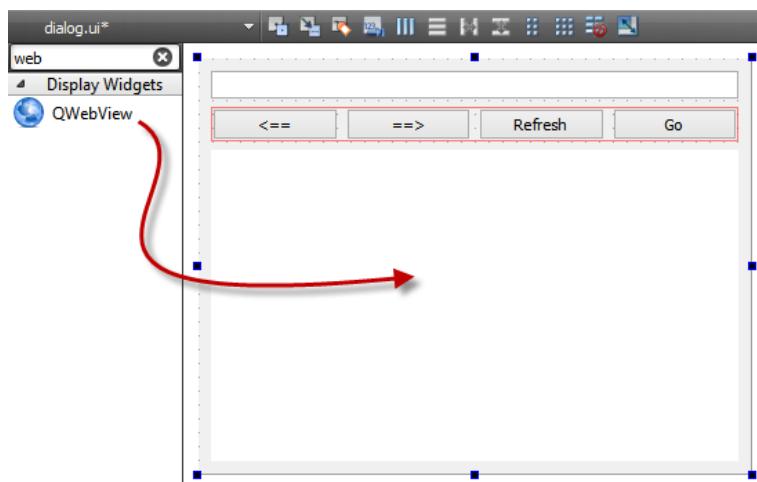


Gambar 13.4:

```
SOURCES += main.cpp\  
dialog.cpp  
  
HEADERS += dialog.h  
  
FORMS += dialog.ui
```

13.2.2 Working on UI

Let's add "urlEdit,"backButton","forwardButton","refreshButton, and "goButton". Also the key widget which is "webView" for QWebView:

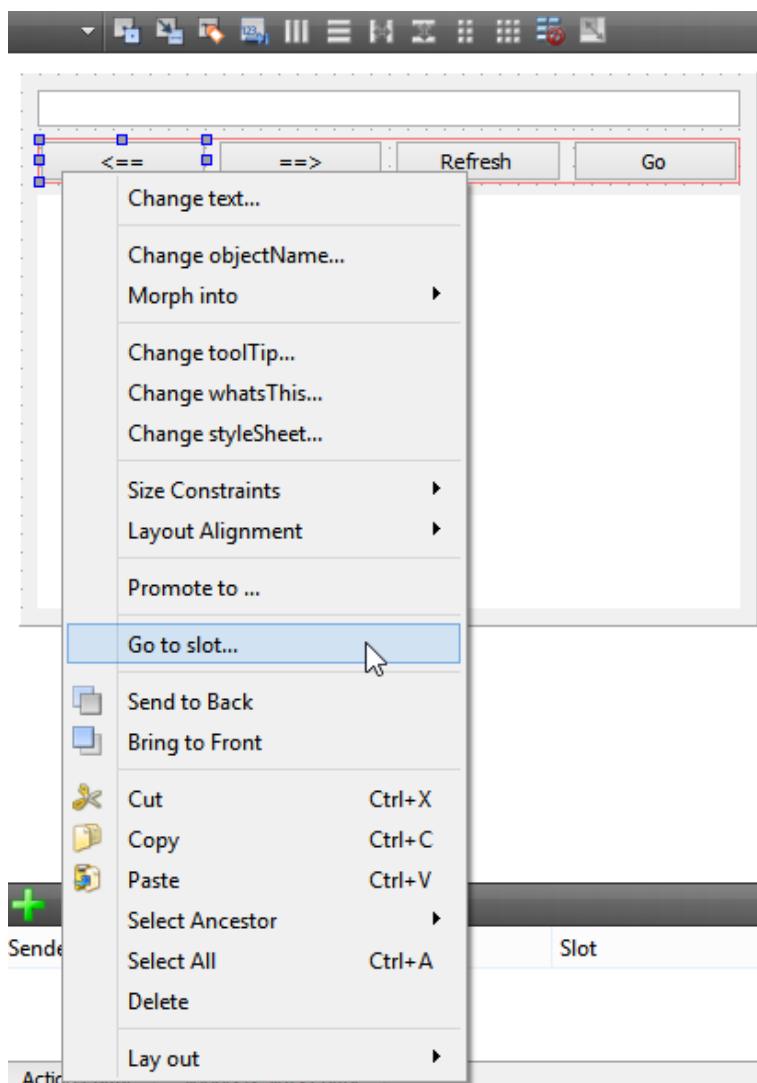


Gambar 13.5:

Slots for the buttons and urlEdit To make the Creator to write a code for us, right click on the "backButton"->GO to slot...->clicked()

Then, in the dialog.cpp, it will write a slot function for us for the "back-Button" click:





Gambar 13.6:

```
void Dialog::on_backButton_clicked()
```

} Also, the Creator will write the prototype declaration into the dialog.h as well.

We need to do the same for the other three buttons.

For the “urlEdit” button, we may want to select returnPressed(), instead.

13.3 Writing Code for the Slots

We need to tell our Browser what to do when those buttons are clicked or return key is pressed on the urlEdit.

First we need to include QWebView into the dialog.h.

Then, let's start coding for the to-do list.

Fortunately, the Creator intelligence gives us some hints on what to do:

So, the finished coding, our slots in dialog.cpp look like this:

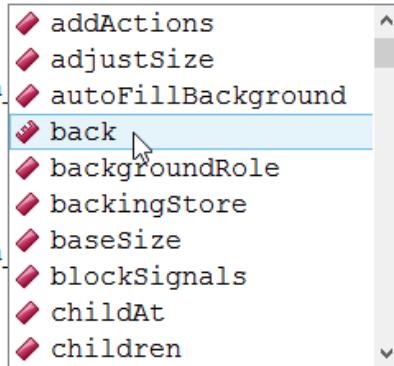
```
void Dialog::on_backButton_clicked()
{
    ui->webView->back();
}

void Dialog::on_forwardButton_clicked()
{
    ui->webView->forward();
}

void Dialog::on_refreshButton_clicked()
{
    ui->webView->reload();
}
```



```
16 void Dialog::on_backButton_clicked()
17 {
18     ui->webView->
19 }
20 void Dialog::on_
21 {
22 }
23 void Dialog::on_
24 {
25 }
26 void Dialog::on_
27 {
28 }
29 }
30
31 void Dialog::on_goButton_clicked()
32 {
33 }
34
35 void Dialog::on_urlEdit_returnPressed()
36 {
37 }
38 }
39
40 }
```

A screenshot of the Qt Creator IDE showing a code editor with C++ code. A completion dropdown menu is open over the line `ui->webView->`. The menu lists several methods starting with 'back', with the cursor hovering over it. Other listed methods include `addActions`, `adjustSize`, `autoFillBackground`, `backgroundRole`, `backingStore`, `baseSize`, `blockSignals`, `childAt`, and `children`.

Gambar 13.7:

```
void Dialog::on_goButton_clicked()
{
    // We just type the domain without "http://"
    ui->webView->load(("http://" + ui->urlEdit->text()))
    ;
}

void Dialog::on_urlEdit_returnPressed()
{
    // Same as goButton click
    on_goButton_clicked();
}
```

13.3.1 Running the Code

Let's run our code.

If we type in another site:

13.3.2 Source Code

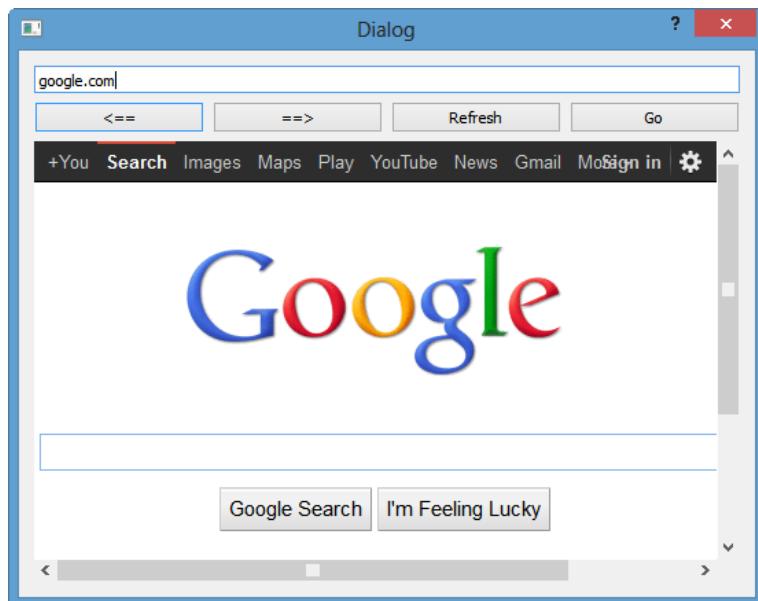
Here is the source code: Browser2.zip.

`main.cpp`.

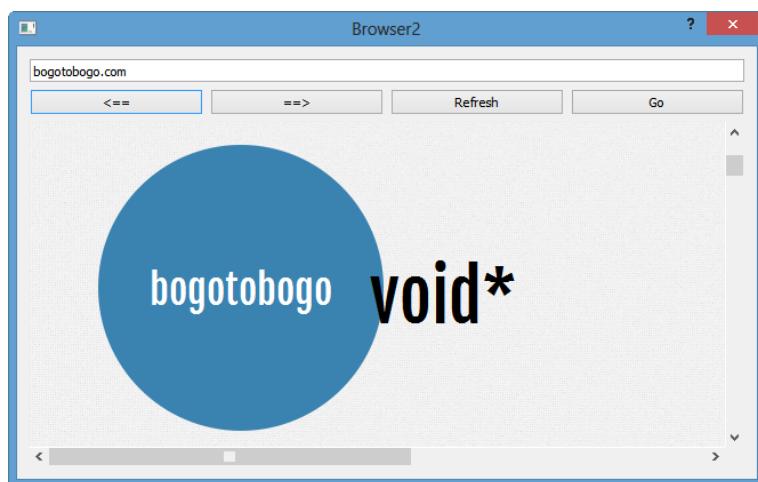
```
#include "dialog.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Dialog w;
```





Gambar 13.8:



Gambar 13.9:

```
w.setWindowTitle("Browser2");
w.show();

return a.exec();
}

dialog.h.

#ifndef DIALOG_H
#define DIALOG_H

#include <QDialog>
#include <QWebView>

namespace Ui {
class Dialog;
}

class Dialog : public QDialog
{
    Q_OBJECT

public:
    explicit Dialog(QWidget *parent = 0);
    ~Dialog();

private slots:
    void on_backButton_clicked();

    void on_forwardButton_clicked();

    void on_refreshButton_clicked();

    void on_goButton_clicked();
```



```
void on_urlEdit_returnPressed();

private:
    Ui::Dialog *ui;
};

#endif // DIALOG_H

dialog.cpp.

#include "dialog.h"
#include "ui_dialog.h"

Dialog::Dialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Dialog)
{
    ui->setupUi(this);
}

Dialog::~Dialog()
{
    delete ui;
}

void Dialog::on_backButton_clicked()
{
    ui->webView->back();
}

void Dialog::on_forwardButton_clicked()
{
    ui->webView->forward();
```



```
}

void Dialog::on_refreshButton_clicked()
{
    ui->webView->reload();
}

void Dialog::on_goButton_clicked()
{
    // We just type the domain without "http://"
    ui->webView->load(("http://" + ui->urlEdit->text()))
    ;
}

void Dialog::on_urlEdit_returnPressed()
{
    // Same as goButton click
    on_goButton_clicked();
}
```

There are lots of things to be done to make our browser better.



Bagian V

Library



BAB 14

Library

Agenda

Pada chapter ini kita akan membahas tentang beberapa class khusus yang ada pada Qt Core Module yang dapat digunakan untuk melengkapi library standar C++ yang sebelumnya sudah kita gunakan. Adapun beberapa topik yang akan kita bahas pada chapter ini adalah.

Contents

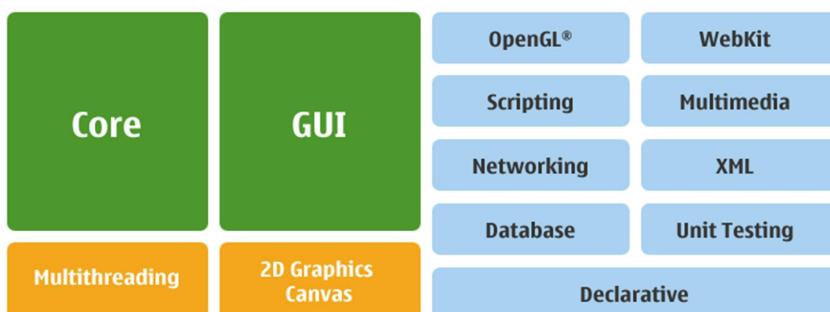
| | |
|--|------------|
| 14.1 Qt Library | 428 |
| 14.2 Menurunkan objek dari class QObject | 429 |
| 14.3 Automatic Memory Management dengan QObject | 429 |
| 14.4 Menggunakan Qt String | 435 |
| 14.5 Collection dan Iterator | 441 |
| 14.5.1 Menggunakan QList | 441 |
| 14.5.2 QList::const_iterator | 444 |
| 14.6 Menambahkan Data pada List | 446 |
| 14.7 Tipe List yang Lain | 447 |



| | |
|--------------------------------|-----|
| 14.8 Special List | 448 |
| 14.9 Stack dan Queue | 450 |
| 14.10 Mapping | 452 |

14.1 Qt Library

Qt SDK menyediakan beberapa class library yang dapat anda gunakan untuk mempercepat pembuatan program, misalnya library untuk membuat GUI (Graphical User Interface), network programming, dan library untuk bekerja dengan XML. Beberapa class library yang disediakan oleh Qt dapat dilihat pada gambar 14.1.



Gambar 14.1: Class libray yang di sediakan oleh Qt

Pada chapter ini kita akan membahas beberapa class dalam Qt Core Module yang sering digunakan seperti QObject, QString dan QStringList.

Qt Core Module adalah library yang dibutuhkan oleh setiap aplikasi Qt. Qt Core Module sendiri dapat berisi :

- Basic Data Type seperti QString dan QByteArray.
- Basic Data Structure seperti QList, QVector, dan QHash.



- Input/Output class seperti QIODevice, QTextStream, dan QFile.
- Class untuk pemrograman multithread seperti QThread.
- Class Object dan QCoreApplication (base class dari QApplication).

14.2 Menurunkan objek dari class QObject

QObject class merupakan base class dari sebagian class yang ada di Qt library. Dengan menurunkan class dari QObject maka anda dapat menggunakan fitur automatic memory management dan mekanisme signal/slot yang disediakan oleh Qt.

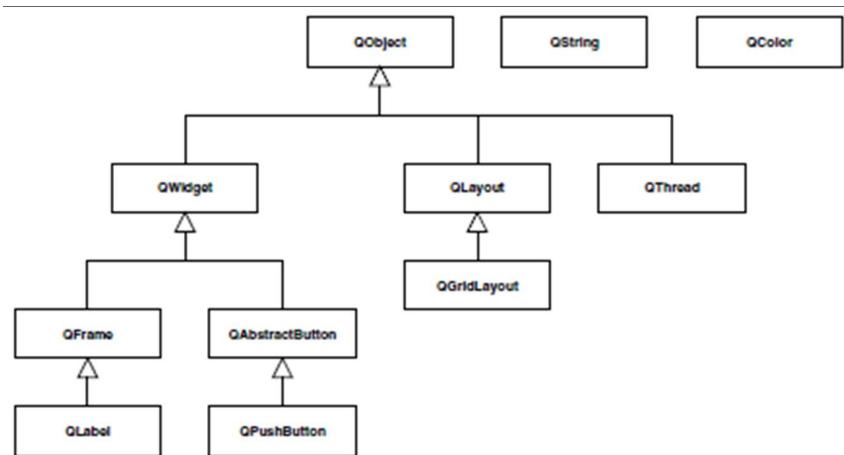
Beberapa class pada Qt yang diturunkan dari class QObject diantaranya QWidget, QLayout, dan QThread. Ada juga class yang tidak diturunkan dari class QObject seperti QString dan QColor.

Gambar 14.2 menunjukan contoh beberapa class yang diturunkan dari QObject.

14.3 Automatic Memory Management dengan QObject

Dengan menurunkan class dari QObject anda dapat melakukan automatic memory management pada program anda, sehingga kemungkinan terjadi memory leak dapat dihindari. Pada contoh dibawah ini kita akan membuat dua program yang berbeda, program pertama tidak menggunakan QObject dan program kedua menggunakan QObject.





Gambar 14.2: Turunan objek dari class QObject

Contoh Alokasi memory dinamis tanpa QObject.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 14.1, kemudian tulis kode berikut.

Listing 14.1: Alokasi memory dinamis tanpa QObject

```
1 #include <QtCore/QCoreApplication>
2 #include <iostream>
3 #include <string>
4 using namespace std;
5 class Mahasiswa
6 {
7 public:
8     Mahasiswa(const string &nim);
9     ~Mahasiswa();
10    const string &nim() const;
11    void setNim(const string &nim);
12    int getNimLength() const;
```



```
13 private:
14   string _nim;
15 };
16 Mahasiswa::Mahasiswa(const string &nim)
17 {
18   _nim = nim;
19 }
20 Mahasiswa::~Mahasiswa()
21 {
22   cout << "destroy object" << endl;
23 }
24 const string &Mahasiswa::nim() const
25 {
26   return _nim;
27 }
28 void Mahasiswa::setNim(const string &nim)
29 {
30   _nim = nim;
31 }
32 int Mahasiswa::getNimLength() const
33 {
34   return _nim.length();
35 }
36 int main(int argc, char *argv[])
37 {
38   QCoreApplication a(argc, argv);
39   Mahasiswa *objMhs1, *objMhs2, *objMhs3;
40   objMhs1 = new Mahasiswa("22002321");
41   objMhs2 = new Mahasiswa("22002322");
42   objMhs3 = new Mahasiswa("22002323");
43   cout << objMhs1->nim() << " : " << objMhs1->
44     getNimLength() << " kar" << endl;
45   objMhs1->setNim(objMhs2->nim());
46   objMhs2->setNim(objMhs3->nim());
```



```
46 cout << objMhs1->nim() << " : " << objMhs1->
     getNimLength() << " kar" << endl;
47 cout << objMhs2->nim() << " : " << objMhs2->
     getNimLength() << " kar" << endl;
48 cout << objMhs3->nim() << " : " << objMhs3->
     getNimLength() << " kar" << endl;
49 delete objMhs1;
50 delete objMhs2;
51 delete objMhs3;
52 return a.exec();
53 }
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.

```
22002321 : 8 kar
22002322 : 8 kar
22002323 : 8 kar
22002323 : 8 kar
destroy object
destroy object
destroy object
```

Keterangan:

- Class Mahasiswa yang kita buat diatas tidak diturunkan dari class QObject, sehingga kita harus menghapus memory di heap yang sudah tidak digunakan kembali secara manual untuk menghindari memory leak.
- Penanganan secara manual menuntut programmer untuk lebih teliti dan akan menyulitkan bila membangun aplikasi yang kompleks dan memiliki banyak objek.



Contoh Alokasi memory dinamis dengan QObject.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 14.2, kemudian tulis kode berikut.

Listing 14.2: Alokasi memory dinamis dengan QObject

```
1 #include <QtCore/QCoreApplication>
2 #include <QObject>
3 #include <QDebug>
4 using namespace std;
5 class Mahasiswa : QObject
6 {
7 public:
8     Mahasiswa(const QString &nim, QObject *parent=0);
9     const QString &nim() const;
10    void setNim(const QString &nim);
11    int getNimLength() const;
12 private:
13     QString _nim;
14 };
15 Mahasiswa::Mahasiswa(const QString &nim, QObject *
16                         parent)
17 {
18     _nim = nim;
19 }
20 const QString &Mahasiswa::nim() const
21 {
22     return _nim;
23 }
24 void Mahasiswa::setNim(const QString &nim)
25 {
26     _nim = nim;
27 }
28 int Mahasiswa::getNimLength() const
```



```
28  {
29      return _nim.length();
30  }
31  int main(int argc, char *argv[])
32  {
33      QCoreApplication a(argc, argv);
34      QObject parent;
35      Mahasiswa *objMhs1, *objMhs2, *objMhs3;
36      objMhs1 = new Mahasiswa("22002321", &parent);
37      objMhs2 = new Mahasiswa("22002322", &parent);
38      objMhs3 = new Mahasiswa("22002323", &parent);
39      qDebug() << objMhs1->nim() << " : " << objMhs1->
40          getNimLength() << " kar";
41      objMhs1->setNim(objMhs2->nim());
42      objMhs2->setNim(objMhs3->nim());
43      qDebug() << objMhs1->nim() << " : " << objMhs1->
44          getNimLength() << " kar";
45      qDebug() << objMhs2->nim() << " : " << objMhs2->
46          getNimLength() << " kar";
47      qDebug() << objMhs3->nim() << " : " << objMhs3->
48          getNimLength() << " kar";
49      return a.exec();
50  }
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.

```
"22002321" : 8 kar
"22002322" : 8 kar
"22002323" : 8 kar
"22002323" : 8 kar
```

Keterangan:

- Untuk menggunakan QObject anda harus menambahkan header file .



- Karena class Mahasiswa diturunkan dari class QObject, maka secara otomatis class Mahasiswa dapat menggunakan fitur automatic memory management.
- Dengan menggunakan class QObject dalam aplikasi, anda tidak perlu men-delete satu-persatu object yang anda buat, karena QObject akan secara otomatis melakukannya untuk anda.
- Method qDebug() lebih disarankan untuk menampilkan input dibandingkan dengan cout, hasil output dari qDebug() lebih kompatibel disemua platform. Dan dengan qDebug() anda tidak perlu menambahkan stl untuk menambahkan enter.

QObject akan disimpan pada stack memory sebagai parent, ketika QObject dihapus semua child dari QObject akan ikut dihapus juga.

14.4 Menggunakan Qt String

Salah satu langkah yang harus dilakukan jika anda mengembangkan aplikasi berbasis Qt adalah ganti semua STL (standar class library) C++ dengan class pada Qt (walaupun anda tetap dapat menggunakan STL). Keuntungan menggunakan class-class yang ada pada Qt adalah lebih kompatibel jika anda berpindah platform.

STL C++ yang paling sering digunakan adalah string, pada Qt anda dapat menggunakan QString. Anda dapat menggabungkan penggunaan string dan QString, namun QString akan lebih baik secara performa dan memiliki lebih banyak fitur, misal QString sudah mendukung unicode pada semua platform yang memudahkan membuat aplikasi dalam bahasa yang berbeda.

Penggunaan fungsi-fungsi yang dimiliki oleh QString akan dibahas pada contoh program dibawah ini.



Contoh Cek apakah nilai QString Null atau Empty

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 14.3, kemudian tulis kode berikut.

Listing 14.3: Cek apakah nilai QString Null atau Empty

```
1 #include <QtCore/QCoreApplication>
2 #include <QDebug>
3 int main(int argc, char *argv[])
4 {
5     QCoreApplication a(argc, argv);
6     //deklarasi string
7     QString nama = "Erick Kurniawan";
8     qDebug() << nama;
9     //cek ukuran string
10    int ukuran = nama.size();
11    qDebug() << "Ukuran string " << ukuran;
12    QString test = "";
13    //cek apakah string null
14    if(test.isNull())
15        qDebug() << "test null";
16    else
17        qDebug() << "test not null";
18    //cek apakah string empty
19    if(test.isEmpty())
20        qDebug() << "test empty";
21    else
22        qDebug() << "test not empty";
23    QString testing;
24    //cek apakah string null
25    if(testing == QString::null)
26        qDebug() << "testing null";
27    else
28        qDebug() << "testing not null";
```



```
29 return a.exec();  
30 }
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.

```
"Erick Kurniawan"  
Ukuran string 15  
test not null  
test empty  
testing null
```

Keterangan:

- Class QString memiliki method size untuk mengetahui ukuran panjang string.
- Method isNull() dapat digunakan untuk memeriksa apakah string masih belum diinisialisasi.
- Method isEmpty() dapat digunakan untuk memeriksa apakah string kosong.

Contoh Menggunakan Fungsi Left, Mid, Right.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 14.4, kemudian tulis kode berikut.

Listing 14.4: Menggunakan Fungsi Left Mid Right

```
1 #include <QtCore/QCoreApplication>  
2 #include <QDebug>  
3 int main(int argc, char *argv[]){  
4     QCoreApplication a(argc, argv);  
5     QString nama = "Erick Kurniawan";
```



```
7 //menggunakan fungsi left, mid, dan right
8 QString firstName = nama.left(5);
9 qDebug() << "firstName : " << firstName;
10QString lastName = nama.right(9);
11qDebug() << "lastName : " << lastName;
12QString midName = nama.mid(6,5);
13qDebug() << "midName : " << midName;
14return a.exec();
15}
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.

```
firstName : "Erick"
lastName : "Kurniawan"
midName : "Kurni"
```

Keterangan:

- Class QString memiliki fungsi left() untuk mengambil sejumlah karakter tertentu dari kiri.
- Fungsi right() digunakan untuk mengambil sejumlah karakter tertentu dari kanan.
- Fungsi mid() digunakan untuk mengambil sejumlah karakter tertentu dari tengah.

Contoh Menggabungkan String.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 14.5, kemudian tulis kode berikut.

Listing 14.5: Menggabungkan String

```
1 #include <QtCore/QCoreApplication>
2 #include <QDebug>
```



```
3 int main(int argc, char *argv[])
4 {
5 QCoreApplication a(argc, argv);
6 QString nama = "Slamet ";
7 nama.append("");
8 nama.append(",BA");
9 nama.prepend("Pak. ");
10 qDebug() << "Nama : " << nama;
11 nama.insert(16, ",SE");
12 qDebug() << nama;
13 return a.exec();
14 }
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.

```
Nama : "Pak. Slamet ,BA"
"Pak. Slamet ,BA ,SE"
```

Keterangan:

- Fungsi append() dapat digunakan untuk menambahkan karakter di akhir string.
- Fungsi prepend() dapat digunakan untuk menambahkan karakter di awal string.
- Fungsi insert() dapat digunakan untuk menyisipkan karakter dengan index tertentu pada string.

Contoh Membalik String.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 14.6, kemudian tulis kode berikut.



Listing 14.6: Membalik String

```
1 #include <QtCore/QCoreApplication>
2 #include <QDebug>
3 int main(int argc, char *argv[])
4 {
5     QCoreApplication a(argc, argv);
6     QString nama = "Nur Wachid";
7     QString balik;
8     for(int i=nama.length()-1;i>=0;i--)
9     {
10         balik+=nama[i];
11     }
12     qDebug() << "Balik : " << balik;
13     return a.exec();
14 }
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.

```
Balik : "dihcaW ruN"
```

Keterangan:

Karena QString adalah array of QChar anda dapat mengambil karakter dari QString menggunakan index array, misal pada contoh diatas nama[i] akan menghasilkan karakter pada index ke-i.

**TIPS**

Anda dapat menggunakan method `toStdString()` dan `fromStdString()` untuk mengkonversi dari string ke QString dan sebaliknya.



14.5 Collection dan Iterator

Qt Core Module juga memiliki class-class collection seperti : list, stack, queue, map, dan hash list. Untuk mengakses data pada object collection anda dapat menggunakan Iterator.

14.5.1 Menggunakan QList

Class QList dapat digunakan untuk membuat type safe object list untuk menyimpan data collection. Untuk mempermudah mengakses semua data pada list anda dapat menggunakan keyword foreach. Dengan QList anda dapat menambahkan data secara dinamis dan anda juga dapat menentukan tipe data ketika mendeklarasikan QList (type safe object) sehingga bila nanti objek yang anda buat diberi nilai yang tipenya berbeda dengan tipe data yang ditentukan program dapat mendeteksi kesalahan tersebut pada saat compile time. Misal jika anda medeklarasikan QList dengan tipe data QString (QList) maka anda tidak dapat memasukan nilai bertipe int kedalam list tersebut.

Contoh Menggunakan QList.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 14.7, kemudian tulis kode berikut.

Listing 14.7: Menggunakan QList

```
1 #include <QtCore/QCoreApplication>
2 #include <QDebug>
3 int main(int argc, char *argv[])
4 {
5     QCoreApplication a(argc, argv);
6     QList<QString> lstNama;
7     lstNama << "Erick" << "Anton" << "Katon" << "Budi"
8 ;
```



```
8 //mengakses data berdasarkan index tertentu
9 qDebug() << lstNama[0];
10 //akan menghasilkan error karena tipe bukan string
11 //lstNama << 12 << 13;
12 //membaca dan menampilkan semua data pada list
13 foreach (QString nama, lstNama) {
14     qDebug() << nama;
15 }
16 return a.exec();
17 }
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.

```
"Erick"
"Erick"
"Anton"
"Katon"
"Budi"
```

Keterangan:

- Pertama kita mendeklarasikan objek lstNama yang bertipe QList, kemudian objek lstNama diberi beberapa data.
- Untuk mengakses data yang ada pada lstNama anda dapat menggunakan keyword foreach Iterators

Selain menggunakan keyword foreach untuk mengakses data pada list anda juga dapat menggunakan iterator. Pada program dibawah ini ditunjukan penggunaan iterator dengan QListIterator untuk mengakses data yang ada pada QList.



Contoh Menggunakan object Iterator.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 14.8, kemudian tulis kode berikut.

Listing 14.8: Menggunakan object Iterator

```
1 #include <QtCore/QCoreApplication>
2 #include <QDebug>
3 int main(int argc, char *argv[])
4 {
5     QCoreApplication a(argc, argv);
6     QList<int> lstNumber;
7     lstNumber << 12 << 24 << 36 << 48 << 60;
8     //menggunakan iterator
9     QListIterator<int> iter(lstNumber);
10    while(iter.hasNext())
11    {
12        qDebug() << iter.next();
13    }
14    //cara lain dengan cara STL
15    QList<int>::const_iterator stlIter;
16    for(stlIter=lstNumber.begin();stlIter!=lstNumber.
17         end();++stlIter)
18    {
19        qDebug() << (*stlIter);
20    }
21    return a.exec();
22 }
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.



```
12  
24  
36  
48  
60  
12  
24  
36  
48  
60
```

Keterangan:

- QListIterator digunakan untuk mengakses data yang ada pada QList.
- Method hasNext() pada iterator digunakan untuk mendeteksi apakah masih ada data di dalam QList dan method next() pada iterator digunakan untuk berpindah data.
- Cara kedua untuk membaca data pada QList adalah dengan menggunakan const iterator

14.5.2 QList::const_iterator

Selain untuk membaca data pada list, iterator juga dapat digunakan untuk memodifikasi data di list, caranya yaitu dengan menggunakan objek QMutableListIterator. Contoh penggunaan QMutableListIterator dapat dilihat pada kode dibawah ini.

Contoh Menggunakan Iterator untuk memodifikasi data di list.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 14.9, kemudian tulis kode berikut.



Listing 14.9: Menggunakan Iterator untuk memodifikasi data di list

```
1 #include <QtCore/QCoreApplication>
2 #include <QDebug>
3 int main(int argc, char *argv[])
4 {
5     QCoreApplication a(argc, argv);
6     QList<QString> lstNama;
7     lstNama << "Erick" << "Anton" << "Katon" << "Ricky"
8     ";
9     QMutableListIterator<QString> iter(lstNama);
10    while(iter.hasNext())
11    {
12        if(iter.next().toLower().contains("rick"))
13        {
14            iter.setValue("update data..");
15        }
16        //baca data setelah diupdate
17        foreach (QString nama, lstNama) {
18            qDebug() << nama;
19        }
20        return a.exec();
21    }
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.

```
"update data.."
"Anton"
"Katon"
"update data.."
```

Keterangan:

- Dengan menggunakan QMutableListIterator anda dapat memodifikasi da-



ta yang anda akses dengan menggunakan iterator.

- Pada program diatas list akan dibaca dan akan dicari data yang mengandung kata “rick”, kemudian data yang mengandung kata tersebut akan diganti dengan kata “update data..”.
- Setelah selesai dimodifikasi dengan iterator data akan dibaca kembali menggunakan foreach, dan anda dapat melihat bahwa ada 2 data yang sudah berubah isinya.

14.6 Menambahkan Data pada List

Ada beberapa cara yang dapat digunakan untuk menambahkan data ke list. Anda dapat menambahkan data diawal, diakhir, atau ditengah list. Beberapa cara penulisan kode untuk menambahkan list adalah sebagai berikut.

Contoh Beberapa cara menambahkan data ke list

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 14.10, kemudian tulis kode berikut.

Listing 14.10: Beberapa cara menambahkan data ke list

```
1 #include <QtCore/QCoreApplication>
2 #include <QDebug>
3 int main(int argc, char *argv[])
4 {
5     QCoreApplication a(argc, argv);
6     QList<QString> lstNama;
7     //menambahkan data di akhir list
8     lstNama << "erick";
9     lstNama.append("katon");
10    //menambahkan data di awal list
11    lstNama.prepend("anton");
```



```
12 //menambahkan data pada index tertentu
13 lstNama.insert(2, "budi");
14 lstNama.insert(4, "naren");
15 foreach (QString nama, lstNama) {
16     qDebug() << nama;
17 }
18 return a.exec();
19 }
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.

```
"anton"
"erick"
"budi"
"katon"
"naren"
```

Keterangan:

- Keyword << dan method append digunakan untuk manambahkan data di akhir list.
- Method prepend digunakan untuk menambahkan data di awal list.
- Method insert digunakan untuk menambahkan data pada index tertentu di list.

14.7 Tipe List yang Lain

Selain menggunakan QList anda juga dapat menggunakan objek collection yang lain seperti QVector dan QLinkedList. Perbandingan performa dari ketiga jenis collection diatas dapat dilihat pada table berikut.



14.8 Special List

Selain untuk tipe data yang umum Qt juga menyediakan collection untuk tipe data khusus seperti QStringList.

Class QStringList diturunkan dari QList dan mempunyai banyak tambahan fungsi yang berguna untuk memanipulasi data di dalam QStringList.

Contoh Menggunakan QStringList

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 14.11, kemudian tulis kode berikut.

Listing 14.11: Menggunakan QStringList

```
1 #include <QtCore/QCoreApplication>
2 #include <QDebug>
3 #include <QStringList>
4 int main(int argc, char *argv[])
5 {
6     QCoreApplication a(argc, argv);
7     QStringList lstKota;
8     lstKota << "Jogjakarta" << "Jakarta" << "Bandung"
9         << "Semarang";
10    //menggabungkan string dengan tanda ',' sebagai
11    //pemisah
12    QString gabung = lstKota.join(",");
13    qDebug() << gabung;
14    //memecah string menjadi QStringList
15    QStringList listSplit = gabung.split(",");
16    foreach (QString kota, listSplit) {
17        qDebug() << kota;
18    }
19    //mengganti elemen dalam array
```



```
18 listSplit.replaceInStrings("a", "aaa");
19 foreach (QString kota, listSplit) {
20     qDebug() << kota;
21 }
22 return a.exec();
23 }
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.

```
"Jogjakarta, Jakarta, Bandung, Semarang"
"Jogjakarta"
"Jakarta"
"Bandung"
"Semarang"
"Jogjaaaartaaa"
"Jaaakaaaartaaa"
"Baaandung"
"Semaaraaaang"
```

Keterangan:

- Dengan menggunakan list QStringList anda akan lebih mudah untuk memanipulasi list yang bertipe string, misalnya anda ingin menggabungkan semua data di dalam list tersebut dengan karakter tertentu menjadi satu string, atau sebaliknya memecah string berdasarkan karakter tertentu kemudian datanya dimasukan kedalam list.
- Untuk memecah string berdasarkan karakter tertentu untuk dimasukan kedalam list anda dapat menggunakan method split().
- Untuk menggabungkan data yang ada di dalam list dengan karakter tertentu menjadi sebuah string anda dapat menggunakan method join().
- Pada kode diatas data pada lstKota digabungkan dengan menggunakan karakter ‘,’ menjadi string gabung.



- Pada kode diatas list lstSplit diisi data hasil pemecahan string gabung, pemecahan string berdasarkan karakter ‘,’

14.9 Stack dan Queue

Jika anda ingin menyimpan data pada collection dengan metode FIFO (first in first out) atau LIFO (last in first out) maka anda dapat menggunakan class QStack dan QQueue. Untuk FIFO anda dapat menggunakan QQueue dan untuk LIFO anda dapat menggunakan QStack.

Contoh Menggunakan Stack dan Queue.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 14.12, kemudian tulis kode berikut.

Listing 14.12: Menggunakan Stack dan Queue

```
1 #include <QtCore/QCoreApplication>
2 #include <QDebug>
3 #include <QStack>
4 #include <QQueue>
5 int main(int argc, char *argv[])
6 {
7     QApplication a(argc, argv);
8     QStack<QString> lstStack;
9     lstStack.push("17rick");
10    lstStack.push("anton");
11    lstStack.push("katon");
12    lstStack.push("budi");
13    //order LIFO
14    qDebug() << "Stack LIFO : ";
15    while(!lstStack.isEmpty())
16    {
```



```
17 qDebug() << lstStack.pop();
18 }
19 QQueue<QString> lstQueue;
20 lstQueue.enqueue("17rick");
21 lstQueue.enqueue("anton");
22 lstQueue.enqueue("katon");
23 lstQueue.enqueue("budi");
24 //order FIFO
25 qDebug() << "Queue FIFO : ";
26 while(!lstQueue.isEmpty())
27 {
28 qDebug() << lstQueue.dequeue();
29 }
30 return a.exec();
31 }
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.

```
Stack LIFO :
"budi"
"katon"
"anton"
"17rick"
Queue FIFO :
"17rick"
"anton"
"katon"
"budi"
```

Keterangan:

- Untuk menyimpan data dengan metode LIFO (Last In First Out) anda dapat menggunakan QStack.



- Untuk menambahkan data kedalam QStack digunakan method push() sedangkan untuk mengambil data dari QStack dapat menggunakan method pop().
- Untuk menyimpan data dengan metode FIFO (First In First Out) anda dapat menggunakan QQueue.
- Untuk menambahkan data kedalam QQueue anda dapat menggunakan method enqueue, dan untuk mengambil data dari QQueue anda dapat menggunakan method dequeue.

14.10 Mapping

Untuk membuat object map dan hash Qt menyediakan class QMap. Dengan QMap anda dapat membuat collection yang index-nya bukan berupa number (key-value pair).

Contoh Menggunakan QMap.

1. Buka Qt Creator dan buat project Qt Console Application baru dengan nama Contoh 14.13, kemudian tulis kode berikut.

Listing 14.13: Menggunakan QMap

```
1 #include <QtCore/QCoreApplication>
2 #include <QDebug>
3 int main(int argc, char *argv[])
4 {
5     QCoreApplication a(argc, argv);
6     QMap<QString, int> lstAge;
7     lstAge["erick"] = 29;
8     lstAge["anton"] = 29;
9     lstAge["katon"] = 42;
10    qDebug() << "erick age : " << lstAge["erick"];
```



```
11 qDebug() << "menampilkan semua data yg ada di map  
12 :";  
13 foreach (QString key, lstAge.keys()) {  
14     qDebug() << key << " : " << lstAge[key];  
15 }  
16 //menggunakan iterator  
17 qDebug() << "Mengakses data menggunakan iterator";  
18 QMap<QString, int>::ConstIterator itr;  
19 for (itr=lstAge.constBegin();itr!=lstAge.constEnd  
20      ();++itr) {  
21     qDebug() << itr.key() << " : " << itr.value();  
22 }  
23 return a.exec();  
24 }
```

2. Kemudian jalankan kode diatas dengan menekan tombol Ctrl+R, maka akan ditampilkan output sebagai berikut.

```
erick age : 29  
menampilkan semua data yg ada di map :  
"anton" : 29  
"erick" : 29  
"katon" : 42  
Mengakses data menggunakan iterator  
"anton" : 29  
"erick" : 29  
"katon" : 42
```

Keterangan:

- Dengan menggunakan QMap anda dapat membuat collection yang indexnya tidak berupa bilangan. Misal anda dapat menggunakan index yang tipenya QString, pada contoh diatas anda dapat menuliskan lstAge["erick"].
- Untuk mengambil semua data pada lstAge anda dapat menggunakan fore-



ach atau menggunakan ConstIterator.



Bagian VI

Lampiran



A



Belajar C++ dengan Qt Creator

Indeks

ANSI, 4
Array, 54
Brian W. Kernighan, 4
CMake, 7
Dennis M Ritchie, 4
DOM, 393
FreeBSD, 6
GNU, 6
GUI, 6, 388
heksadesimal, 28
identifier, 28
konstanta, 28
Linux, 6
middle level programming language, 28
Program, 28
Qcolor, 388
QDataStream, 388
Qdir, 380
QFile, 384
QFileInfo, 384
qmake, 7
QML, 7
QString, 380
Qt Designer, 7, 8
Qt Quick Designer, 8
Qt SDK, 6
Qt widgets, 7, 8
QtextStream, 387
The C Programming Language, 4
Windows, 6
XML, 393

