**BAB 1**

**Pengenalan Bahasa Pemograman**

**1.Alasan Mengapa Kita Harus Belajar Konsep Pemrograman**

Di masa yang lalu secara luas banyak yang percaya bahwa untuk memasuki zaman komputer hampir semua orang harus belajar bagaimana cara memprogram. Tetapi anggapan tersebut tidak dapat dibenarkan, karena dimasa sekarang komputer sudah memiliki perangkat lunak yang mudah digunakan dan tidak memerlukan keahlian dalam pemrograman.

Alasan lain untuk belajar pemrograman mungkin untuk berkarir sebagai ilmuwan komputer. Bagaimanapun juga, walaupun pemrograman adalah satu perlengkapan utamanya (primary tools), ilmu komputer adalah disiplin ilmu yang luas dan bermacam-macam, dimana meliputi dari persoalan engineering, seperti desain processor, persoalan yang berhubungan dengan matematik, dan performance analysis. Banyak ilmuwan komputer yang mengerjakan sedikit atau tidak sama sekali sebagai bagian dari hari kerja mereka. Jika Anda merencanakan untuk mengambil ilmu komputer sebagai yang utama ataupun tambahan, tentu saja Anda akan belajar pemrograman, tetapi untuk berkarir dibidang komputasi tersedia bagi programmer maupun nonprogrammer.

Dengan mempelajari bahasa pemograman juga dapat menambah pengetahuan anda dalam mempelajari bahasa pemograman yang tentunya akan mempermudah anda untuk mengerti makna essensial dari implementasi suatu program dan membantu bagi orang yang sudah mengenal suatu bahasa pemograman sebelumnya untuk lebih memahami sehingga lebih baik dan cepat dalam menggunakan bahasa pemograman yang diketahui.Semakin baik konsep bahasa pemograman seseorang,semakin mudah seseorang untuk mempelajari bahasa pemograman yang baru yang akan menjadi referensi bagi seseorang untuk memilih bahasa pemograman yang cocok untuk dirinya.

Terakhir alasan terbaik kenapa untuk belajar pemrograman adalah kreativitasnya dan kegiatan pemecahan masalah yang menyenangkan. Salah satu pikiran terbaik tentang pemrograman adalah Anda melihat dengan sebenarnya dan merasakan pengalaman dalam menjalankan program dari solusi yang Anda buat. Mendesain dan membangun program yang dibuat dengan baik akan menyediakan pengertian yang kuat dalam penyelesaian dan kepuasan.Jadi bagi Anda yang tertarik dengan dunia pemrograman, mulailah dari sekarang untuk belajar pemrograman, dan cobalah untuk menyelesaikan suatu permasalahan lalu diimplementasikan ke dalam program.

**2.Domain Program**

Pemograman telah menyentuh hampir seluruh titik kehidupan manusia.Namun,terdapat lima yang sangat menonjolkan eksitensi program pada suatu bidang sebagai berikut:

* *Scientific applications*

Komputer digital pertama kali muncul pada akhir tahun 1940 dan awal tahun 1950 diciptakan untuk kepentingan *scientific .*Komputer itu digunakan untuk stuktur data yang sederhana dengan data struktur berupa *array* dan matriks serta control struktur yang dipakai ialah *counting loops* dan seleksi.Bahasa pemograman yang dipakai ialah Fortran.Selanjutnya akan dipakai bahasa pemograman Algol 60 .

* *Business applications*

Komputer untuk aplikasi bisnis pertama kali dimulai pada tahun 1950 dengan bahasa pemograman tingkat tinggi pertama yaitu COBOL.Komputer digunakan untuk fasilitas menelaborasi laporan , menentukan cara tepat untuk data decimal,data karakter dan mengspesifikasikan operasi aritmatik desimal.

* *Artificial intelligence*

AI adalah aplikasi computer yang lebih banyak memakai simbol daripada komputasi numerik.Komputasi simbolik lebih nyaman dipakai dengan *linked list* daripada array .Bahasa pemograman yang dipakai pertama kali ialah LISP pada tahun 1959 dan pada awal tahun 1970 ,munculnya bahasa pemograman Prolog

* *Systems programming*

Sistem operasi dan alat yang mendukung program dari system computer disebut system software. Bahasa pemograman yang dibuat untuk *system programming* oleh IBM ialah PL/S yang nantinya akan berubah menjadi PL/I.Namun,sekarang bahasa pemograman yang dipakai ialah bahasa pemograman C dan C++.Contohnya,pada sistem operasi UNIX sendiri ditulis kebanyakan dalam bahasa C.

* *Web Software*

*The World Wide Web* didukung oleh sebuah kumpulan bahasa pemograman yang diambil dari *markup language* ,seperti HTML .Biasanya untuk pembuatan konten web yang dinamik dan fungsi ini didapatkan dari bahasa pemograman HTML,Java Script dan PHP.

**3*.Language Evaluation Criteria* ( Kriteria Evaluasi Bahasa Pemograman )**

Kriteria yang dibutuhkan untuk melakukan evaluasi pada bahasa pemograman adalah:

1. Readability: kemudahan yang program dapat dibaca dan dipahami

Faktor-faktor dari readability:

* Simplicity
* Orthogonality

Orthogonality adalah sekumpulan pembentukan data dasar yang dapat dikombinasikan dengan berbagai cara untuk membangun control dan struktur data dari suatu bahasa .

* Tipe data dan Struktur data
* Desain Syntax

1. Writability: kemudahan yang bahasa dapat digunakan untuk membuat program.

Faktor-faktor dari writability :

* Simplicity.
* Orthogonality
* Expresif
* Mendukung abstraksi.

Abstraksi adalah kemampuan untuk mendeﬁnisikan dan kemudian menggunakan struktur atau operasi yang kompleks yang mengizinkan pengabaian detail sebanyak-banyaknya.

* Tipe data dan Struktur data
* Desain Syntax

1. Reliability: sesuai dengan spesifikasi (yaitu, melakukan dengan spesifikasinya).

Faktor-faktor dari realiability :

* Type checking.
* Exception handling.
* Readability dan writability

1. Cost : Biaya akhir dari sebuah bahasa pemrograman adalah fungsi dari banyak

characteristicsa nya:

1. Biaya programmer pelatihan
2. Biaya penulisan program
3. Biaya kompilasi program
4. Biaya mengeksekusi program

**3.Pengaruh terhadap Desain Bahasa Pemograman**

* Arsitektur komputer

Bahasa dikembangkan sekitar arsitektur komputer umum, yang dikenal sebagai

arsitektur von Neumann. Desain bahasa pemrograman telah menerima efek mendalam dari arsitektur dasar komputer. Sebagian besar bahasa pemrograman yang populer dari 50 tahun terakhir telah dirancang pada masa arsitektur komputer umum, yang disebut arsitektur von Neumann, pencetusnya John von Neumann .Bahasa pemrograman yang dipakai ialah bahasa-bahasa imperatif.

Karena arsitektur von Neumann, fitur utama dari bahasa-bahasa imperatif adalah variabel, model sel memori; assignment statement, yang didasarkan pada operasi pipa (piping) dan bentuk berulang dari pengulangan, yang merupakan cara paling efisien untuk melaksanakan pengulangan pada ini arsitektur komputer ini.

Arsitektur Von Neuman

* Shared program technique :  
  -Perangkat keras sederhana yang tidak perlu tertanam untuk setiap program.

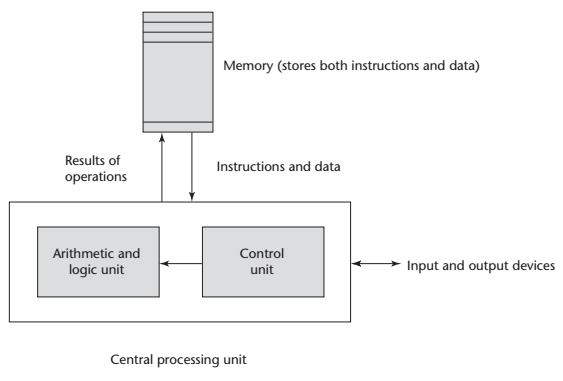
-Memori terpisah dari CPU

-Data dan Instruksi disalurkan dari memori ke CPU  
 -Instruksi kompleks untuk digunakan untuk mengontrol perangkat keras, yang

memungkinkan untuk "memprogram ulang"(“reprogrammed”) jauh lebih cepat.

* Conditional control transfer (Transfer kontrol kondisional) :

-subroutines atau blok kecil kode yang dapat melompat ke dalam urutan apapun bukan berurutan memerintahkan langkah-langkah.



**Von Nuemann computer architecture**

* Metode-Metode Desain Program

Metode-Metode Desain Program mengarahkan kepada paradigma bahasa pemograman yang baru dengan perbaharuan dan bahasa pemograman baru.Berikut perkembangan Metodologi Pemrograman  
-Tahun 1950-an dan awal 1960-an: aplikasi sederhana; khawatir tentang efisiensi mesin.  
-masa akhir 1960an: efisiensi orang menjadi penting, mudah dibaca, struktur kontrol yang lebih baik.

-Masa akhir 1970an : data abstraksi.  
-masa pertengahan 1980an: domain dan data kompleksitas ; pemrograman berorientasi obyek.

-Masa saat ini : web dan lingkungan jaringan; komputasi terdistribusi.

**4.Kategori Bahasa Pemograman**

Ada 4 kategori bahasa pemrograman :

1. Imperative, bahasa yang berbasiskan *von Neumann Archittecture*, diamana memori yang sama digunakan untuk data dan memberikan instruksi kepada program. Mayoritas computer modern menggunakan *von Neumann Archittecture* sebagai basis nya.
2. Functional, bahasa yang berbasiskan fungsi matematika yang merupakan struktur anggota dari suatu set yang disebut domain set sampai ke set yang lain yang disebut range set.
3. Logic, bahasa yang berbasiskan logika yang formal yang mengekspresikan fakta dan aturan – aturan yang berkaitan dengan suatu masalah pada sebuah domain.
4. Markup/programming hybrid, bahasa yang berbasiskan teks dan tags yang berhubungan dengan dokumen elektronik, operating system, aplikasi dan program.

**5.Metode Implementasi**

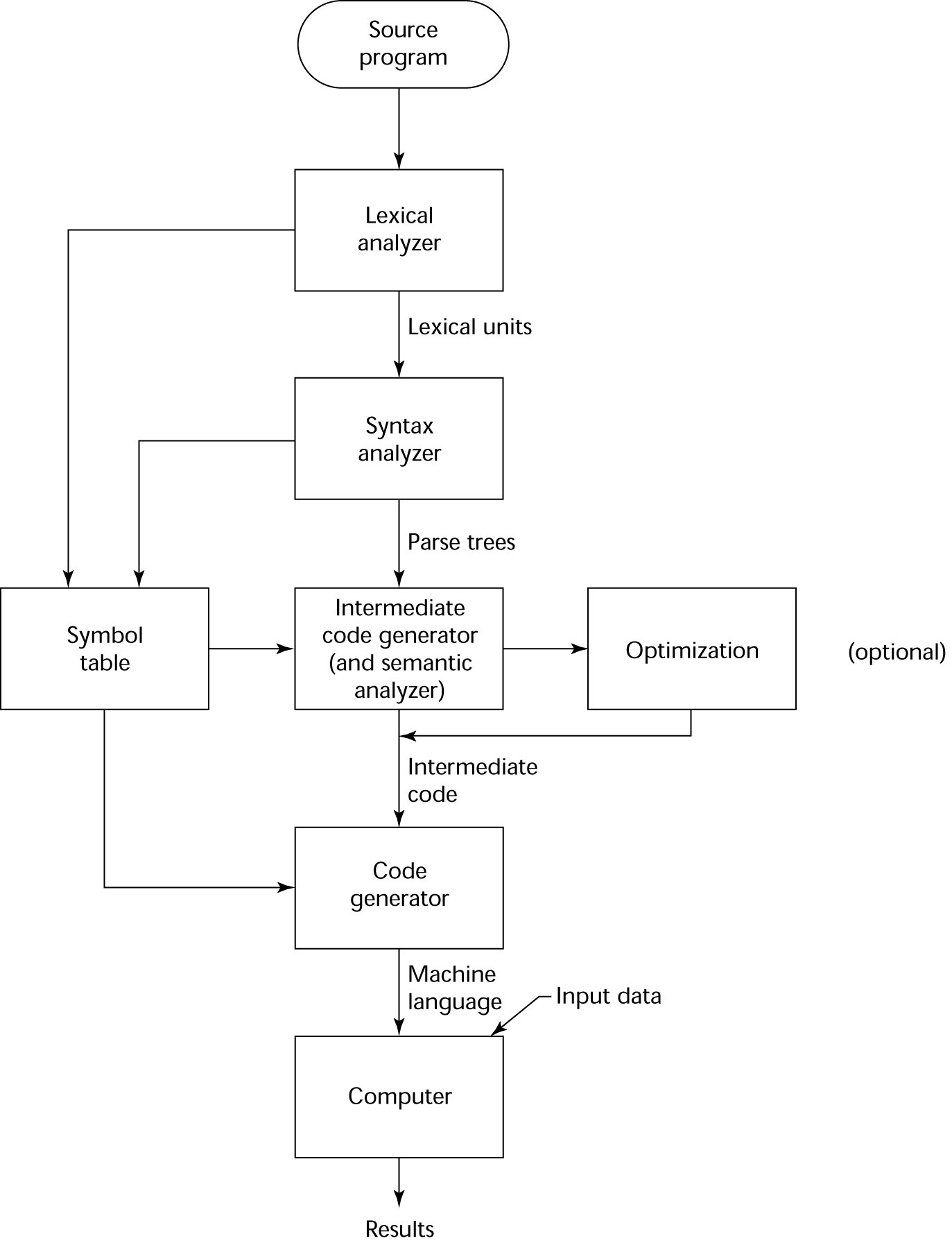
Ada 3 metode untuk mengimplementasikan bahasa pemrograman :

1. Compilation, dengan metode ini compiler akan menerjemahkan suatu program menjadi *machine language*, yang bisa menjadi suatu input untuk interpreter ato compiler yang lain.

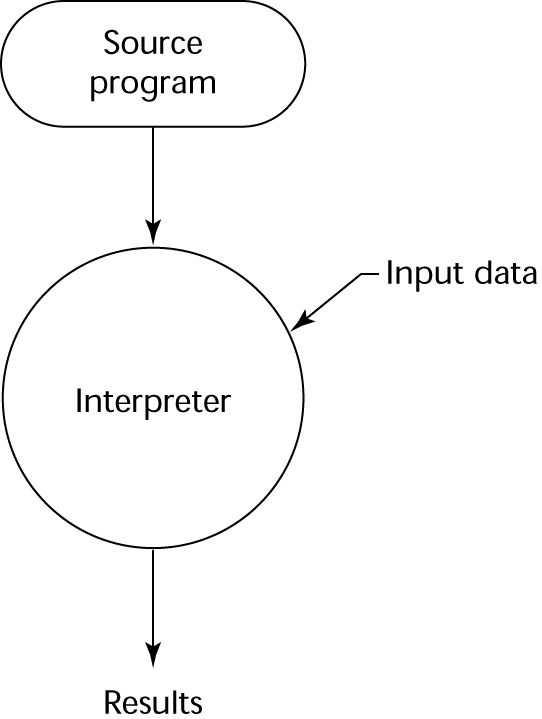
Compilation juga terbagi atas beberapa fase :

1. Lexical analysis, perubahan karakter di dalam program sumber menjadi unit lexical.
2. Syntax analysis, merubah unti lexical menjadi *parse trees*, yang merepresentasikan struktur syntax dari suatu program.
3. Semantics analysis, menghasilkan *intermediate code*.
4. code generation, merubah *intermediate code* menjadi kode yang dapat dimengerti oleh mesin sehingga siap digunakan.

Compilation biasa digunakan untuk aplikasi komersial.

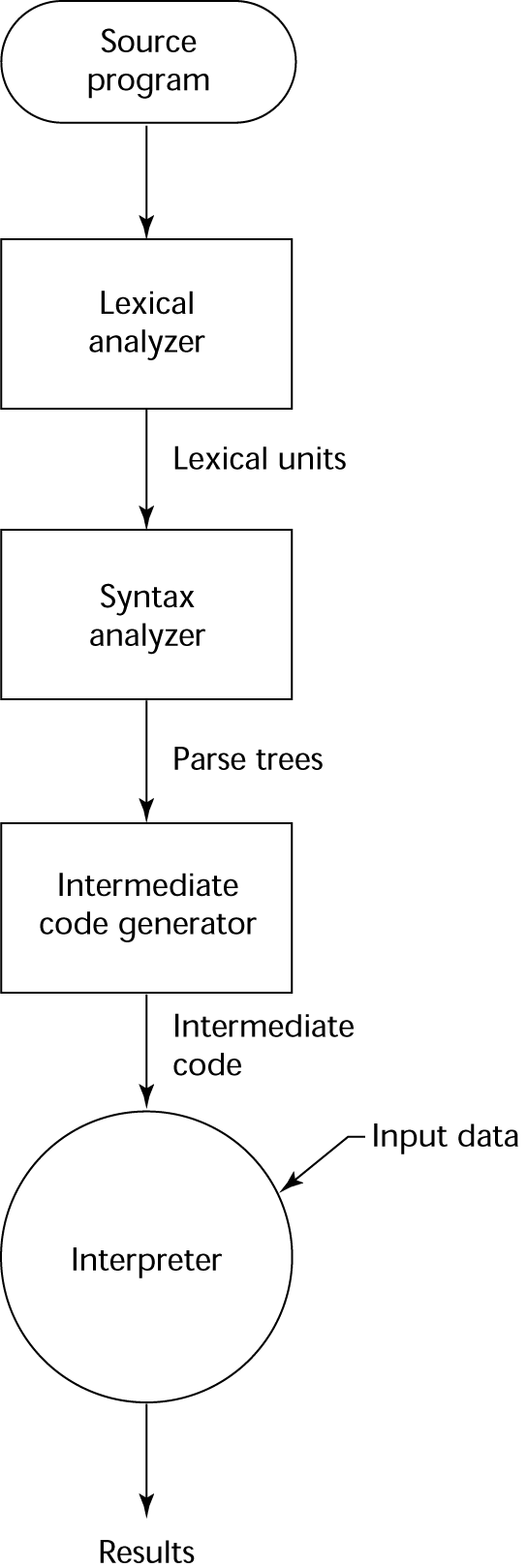
  
Gambar Proses *Compilation*

1. Pure Interpretation, dengan metode ini suatu program akan ditafsirkan oleh interpreter. Metode ini tidak membutuhkan translasi dan lebih mudah untuk digunakan karena error bisa ditemukan dengan cepat, akan tetapi memiliki waktu eksekusi yang lebih lama dan memakan lebih banyak memori.  
   Pure Interpretation biasa digunakan pada program – program kecil.



Gambar Proses *Pure Interpretation*

1. Hybrid Implementation Systems, merupakan gabungan dari Compilation dan Pure Interpretation. Dalam metode ini bahasa yang berlevel tinggi dari suatu program akan diterjemahkan menjadi bahasa level menengah agar dapat di interpretasi dengan mudah.   
   Hybrid Implementation Systems biasa digunakan untuk program kecil dan sedang.



Gambar Proses *Hybrid Implementation System*

**BAB 2**

**Syntax dan Semantic**

1. **Syntax** 
   1. **Pengenalan**

* **Sejarah Singkat PHP**
* 1995 : pertama kali dibuat oleh Rasmus Lerdorf bernama Form Interpreted (FI) yang merupakan kumpulan skrip untuk mengolah data formulir dari web
* 1997 : PHP/FI 2.0 dirilis dengan kemampuan PHP/FI yang dapat diimplimentasikan dalam program C. Sebuah perusahaan bernama Zend menulis ulang interpreter PHP menjadi lebih bersih, lebih baik, dan lebih cepat.
* 1998 : Zend merilis interpreter baru untuk PHP yaitu PHP 3.0 dan singkatan PHP diubah menjadi akronim berulang PHP: Hypertext Preprocessing.
* 1999 : Zend merilis interpreter PHP baru yaitu PHP 4.0 dimana merupakan versi PHP yang paling banyak dipakai pada awal abad ke-21. Karena, kemampuannya untuk membangun aplikasi web kompleks tetapi tetap memiliki kecepatan dan stabilitas yang tinggi.
* 2004 : Zend merilis PHP 5.0. Versi ini memasukkan model pemrograman berorientasi objek ke dalam PHP untuk menjawab perkembangan bahasa pemrograman ke arah paradigma berorientasi objek.
* **Pengenalan Syntax dan Semantics**

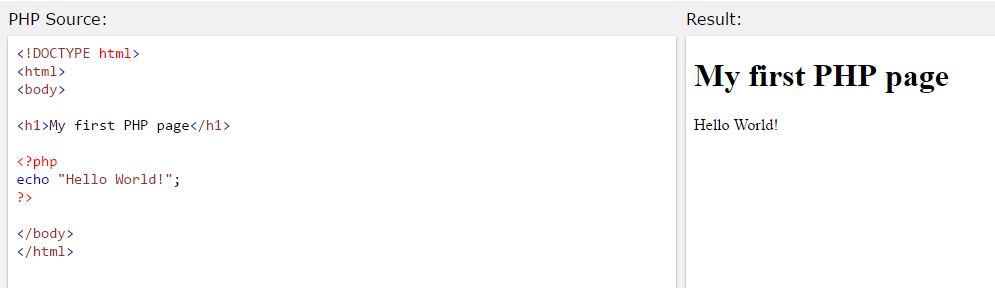
Tugas pendeskripsian bahasa pemograman yang ringkas dan mudah dimengerti ialah tugas yang sulit namun sangat penting dalam menuntukan jumlah pemakaian Bahasa pemograman tersebut kedepannya. Salah satu masalahnya adalah perbedaan tingkat pemahaman seseorang dalam mengerti deskripsi bahasa pemograman tersebut. Bahasa pemograman haruslah dapat disandikan kembali oleh sang pembaca mengacu pada refensi bahasa pemograman. Pembelajaran penulisan bahasa pemograman dapat ditulis dengan syntax dan semantics.

Syntax adalah bentuk atau struktur ekspresi, pernyataan, dan unit program. Semantic adalah arti dari ekspresi, pernyataan, dan unit program.Contoh pengunaan syntax dan semantics dalam Bahasa PHP (Hypertext Protocol):

|  |  |
| --- | --- |
| Syntax : | Semantics : |
| <?php  $i=1;  **while ($i <= 10)**  **{**  **echo "$i";**  **echo "<br />";**  **$i=$i+1;**  **}**  ?> | Ketika kondisi terpenuhi,pernyataan terkait akan dihasilkan dan mengulang proses lagi  Jika $i<=10 , setelah itu  Mencetak nilai i ,mengulang sampai syarat tak lagi terpenuhi |

Walaupun sering dipisahkan saat menjelaskan,syntax dan semantic memiliki hubungan yang dekat. Dalam mendesain bahasa pemograman yang baik, semantics harus mengikuti syntax secara langsung. Pendeskripsian syntax lebih mudah pendeskripsian semantic, diakibatkan pendeskripsian syntax yang ringkas dan diterima secara universal, tapi tidak untuk pendeskripsian semantics.

* **Beberapa Syntax Dasar pada Bahasa Pemograman PHP**



1

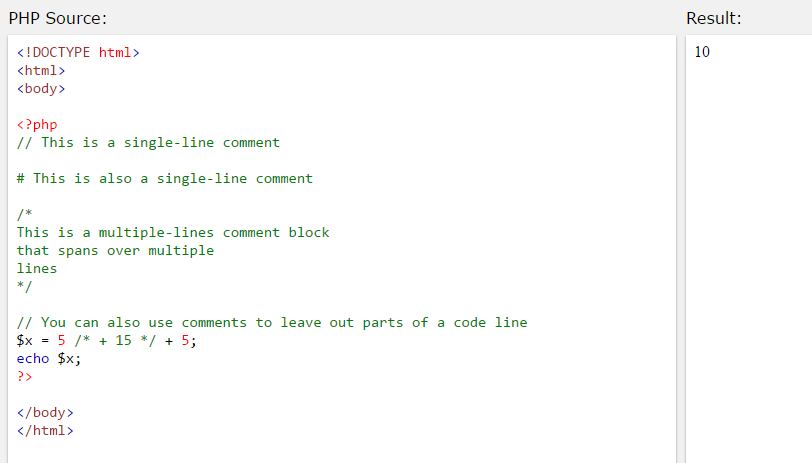
2,3

1

1.Syntax “<?php” sebagai pembuka dan “?>” sebagai penutup.

2.Syntax “echo” untuk menampilkan data

3.Setiap baris atau statement dari PHP harus diakhiri dengan semicolon (;).



7,8

6

5

4

4. Syntax PHP dapat digunakan bersamaan dengan HTML

5. Pembuatan komentar yang berisi 1 baris, komentar diawali dengan //   atau #

6. Untuk pembuatan komentar lebih dari 1 baris, komentar diapit oleh /\* ...  \*/

7. Nilai variabel dapat dengan mudah ditampilkan dengan menggunakan echo, baik nilai yang berupa bilangan maupun string. Karena, variable dalam PHP tidak terikat dengan tipe data tertentu

8. Syntax “$” untuk menyimpan suatu variable.

* 1. **Masalah Umum dalam Pendeskripsian Syntax**

Masalah umum yang sering muncul saat mendeskripsikan syntax biasanya berupa pengidentifikasian terminology dari suatu syntax. Sebuah bahasa baik itu bahasa natural (contoh bahasa Indonesia) atau bahasa buatan (contoh PHP) terdiri dari sekumpulan karakter dari beberapa alfabet.Sekumpulan karakter inilah yang disebut pernyataan.

Dalam bentuk formal, pendeskripsian syntax biasanya tidak diikuti deskripsi dari syntax dalam unit yang lebih kecil.Unit syntax yang kecil ini disebut dengan *Lexemes. Lexemes* dalam bahasa pemograman terdiri dari operator,kata khusus dan angka sesungguhnya. *Lexemes* dapat dibagi kedalam beberapa grup seperti *variable*, *identifier*, *methods*, dan *classes*. Setiap grup *lexemes* ini biasa diwakili oleh token. Jadi, token adalah nama kategori dari *lexemes.* Contoh penggunaan *lexemes* dan token dalam bahasa PHP:

|  |  |
| --- | --- |
| **Language** | **PHP** |
| Sentence | $umur = $tahunskrg-$tahunlahir ; |
| Lexeme | Token |
| umur  tahunskrg  tahunlahir | Identifier |
| = | equal\_sign |
| - | min\_op |
| ; | Semicolon |

Secara general, bahasa pemograman dapat digambarkan dengan dua cara yang berbeda yaitu *recognition* dan *generation*. *Recognition (Language Recognizers)* menerima sebuah bahasayang dapat menerima string sebagai input. *Recognition (Language Recognizers)* dapat menerima atau menolak suatu inputan. Sebagai contoh, jika sebuah mesin M mengenali sekumpulan kata (*string*) dari suatu bahasa B, dan menerima inputan yang disediakan *string*. Maka, M menerima S ( S yang di dalam B). Sebaliknya bila M menolak S , maka S tidak ada dibagian B*.*

*Generation (Language Generator)* membentuk *string* dari sebuah bahasa dan menyediakan deskripsi dari pembentukan bahasa. Jika sebuah *Generation (Language Generator)* mampu membangun seluruh string yang ada di bahasa B dan setiap string S yang dibentuk oleh generator ada di B. Maka dapat disimpulkan bahwa *Generator* adalah untuk bahasa S. Sebaliknya, apabila *Generator* tak dapat menyusun string S maka S bukan berada dalam B.

* 1. **Metode Formal dalam Pendeskripsian Syntax**

Sebelumnya dalam pendeskripsian syntax,dibentuk *context-text free* dan *regular* sampai pada akhirnya terbentuknya BNF (*Backus-Naur Form*). BNF adalah sebuah bahasa yang dapat mendeskripsikan bahasa pemograman yang lain. BNF menggunakan abstraksi untuk struktur syntactic. Sebagai contoh :

<assign> →<var> = <expression>

Teks yang berada sebelah kiri dari tanda panah,disebut *left-hand side(LHS),* dimana sebuah abstraksi didefinisikan. Sedangkan teks yang berada disebelah kanan tanda panah adalah *right-hand side (LHS)* yang dapat terdiri dari *tokens, lexemes* dan *referensi* yang mengacu ke abstraksi lainnya. Abstraksi pada BNF deskripsi, atau grammar, biasa disebut nonterminal symbols, atau nonterminal. Sedangkan, *lexemes* dan *tokens* dari aturan biasa di sebut terminal symbols, atau terminal. Definisi yang ganda dapat ditulis dengan sebuah peraturan yaitu dipisahkan dengan tanda “|” yang mengartikan “OR”. BNF deskripsi adalah kumpulan dari aturan. Sebagai contoh :

<if\_stmt> →**if**( <logic\_expr> ) <stmt>

<if\_stmt> →**if**( <logic\_expr> ) <stmt> **else**<stmt>

atau

<if\_stmt> →**if**( <logic\_expr> ) <stmt>

|**if**( <logic\_expr> ) <stmt> **else**<stmt>

Syntax juga dapat dijabarkan dengan didefinisikan menggunakan rekulsif, sebagai contoh :

<ident\_list> →identifier

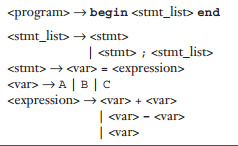
|identifier, <ident\_list>

Mendefinisikan bahwa <ident\_list> sebagai token (identifier) tunggal atau identifier yang diikuti oleh koma, dan instance lain dari <ident\_list>.

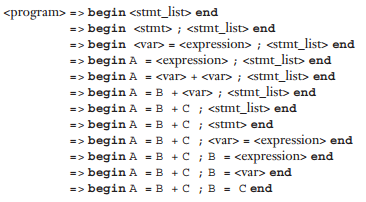
* **Grammar and Derivation**

Grammar adalah alat generative untuk mendefinisikan bahasa. Kalimat dari bahasanya di tentukan melalui urutan aturan aplikasi yaitu diawali dengan *grammar* non-terminal yang disebut ***start symbol***. Urutan dari aturan aplikasi disebut dengan ***derivation***.

Sebagai contoh :



Contoh *derivation* dari program :

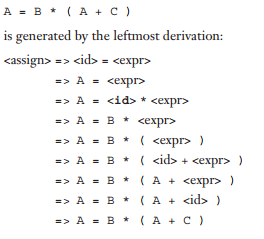


Derivasi ini dimulai dari symbol awal ,untuk contoh diatas berupa <program>. Setiap string pada sebuah *derivation* termasuk <program> disebut dengan *sentential form*. Pada derivasi ini, nonterminal yang dapat dipindah selalu *leftmost-terminal* pada *sentential form* sebelumnya. Derivasi ini akan berhenti sampai *sentential form* tidak terdapat nonterminals. *Sentential form* hanya terdiri dari 1 terminal atau lexemes atau 1 simbol. Sebuah derivasi juga dapat *rightmost* atau apabila tidak ada leftmost dan rightmost. Aturan derivasi tidak berpengaruh pada bahasa yang dihasilkan dari sebuah *grammar.*

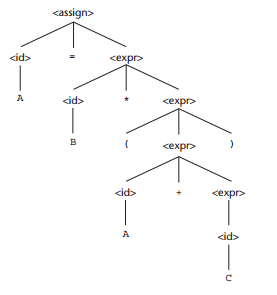
* **Parse Tree**

Grammar dapat dideskripsikan kedalam bentuk system yang hirarkis yang disebut *parse tree.* Sebagai contoh :

Contoh bentuk pernyataan yang biasa untuk pernyataan A=B\*(A+C):



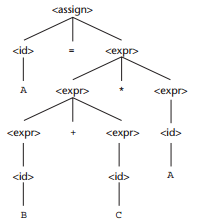
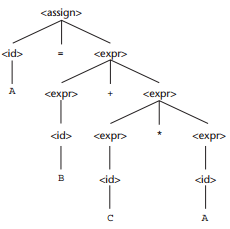
Contoh bentuk Parse Tree pernyataan A=B\*(A+C):



* **Ambiguitas**

Grammar dinyatakan ambigu jika dan hanya dapat dikerjakan dengan sentential form yang memiliki 2 atau lebih distinct parse tree. Sebagai contoh :

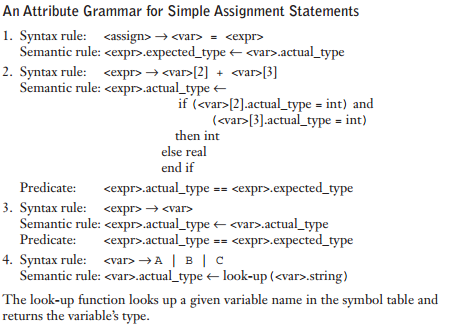
A=B+C\*A

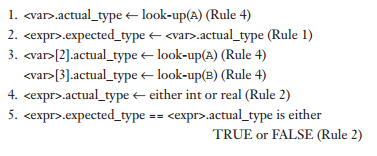
* *

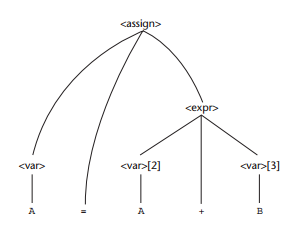
* 1. **Tata Bahasa Atribut**

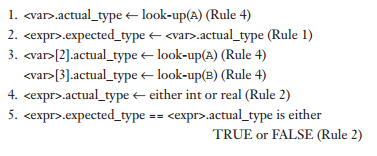
Tata bahasa atribut adalah perangkat yang digunakan untuk menggambarkan lebih dari struktur bahasa pemrograman. Tata bahasa atribut juga dapat diartikan sebagai pemanjangan untuk tata bahasa *context-free grammar*, dengan penambahan berikut:

* Untuk masing-masing simbol grammar x ada satu set A (x) dari nilai atribut,
* Setiap aturan memiliki satu set fungsi yang mendefinisikan atribut tertentu dari nonterminals dalam aturan,
* Setiap aturan memiliki (mungkin kosong) set predikat untuk memeriksa konsistensi atribut.



contoh tata bahasa atribut untuk A=A+B :



****

* 1. **Semantics**

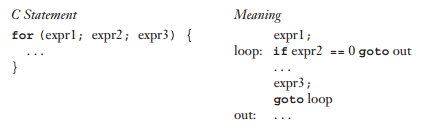
**2.Semantic**

Semantics adalah penjelaskan arti dari sebuah program dengan mengeksekusi statement pada mesin. Semantic di bagi 3 yaitu:

**1.**      **Operational semantic**

Pendekatan ini mendefinisikan suatu mesin buatan (abstrak) dengan instruksi-instruksi primitive, tidak perlu realistic, tetapi cukup sederhana supaya tidak muncul kesalahpahaman. Deskripsi semantic dari bahasa pemrograman menentukan suatu translasi ke kode.

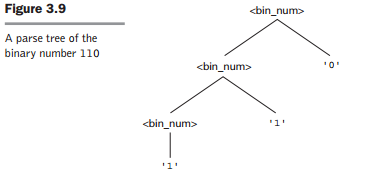
Contoh :



**2.**      **Denotational semantics**

Pada pendekatan ini, diberikan suatu fungsi yang memetakan program-program computer yang di tunjuk ke dalam bentuk nilai-nilai abstrak secara matematika (angka, nilai kebenaran, fungsi matematika, dsb).

Contoh :



**3.**      **Axiomatic semantic**

Pada pendekatan ini di definisikan suatu tindakan program yang di bangun dengan property logika yang menyimpan status computer sebelum dan sesudah dieksekusi.

Contoh :

**a = b + 1 {a > 1}**

One possible precondition : {1,2,5,10}

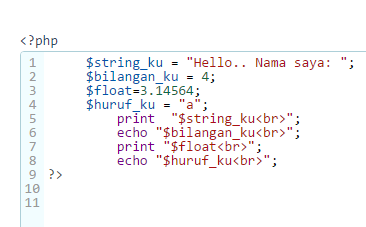
Weakest precondition : {b > 0}

**BAB 3**

**Name, Bindings, and Scopes**

1. **Name**

Name atau dapat disebut juga penamaan merupakan sekumpulan string yang digunakan untuk memberi nama pada variable yang kita pakai saat membuat suatu koding.Tidak seperti bahasa pemograman Fortran 95+ yang penamaannya terbatas pada 31 karakter,pada bahasa Java,C#,PHP dan C++ , tidak ada pembatasan dalam jumlah pengunaan nama.Berikut adalah contoh pemberian tata nama variabel dalam bahasa pemograman PHP :



* Nama dan tipe variabel tidak perlu dideklarasikan terlebih dahulu seperti halnya bahasa Pascal atau C/C++.Sehingga 1 variabel dapat memuat lebih dari 1 tipe data tanpa adanya pendeklarasian sebelumnya.
* Nama variabel diawali dengan tanda $
* Penamaan variabel dalam bahasa PHP bersifat *case sensitive.*Contoh variabel bunga,Bunga BUNGA dan BuNga dianggap sebagai variabel yang berbeda
* Nama variabel harus diawali dengan huruf atau underscore (\_)
* Nama variabel hanya boleh dituliskan dengan alpha numeric a-z, A-Z, 0-9 dan underscore (\_)
* Nama variabel yang terdiri lebih dari satu kata, dapat dipisahkan dengan underscore (\_)
* Variabel tidak dapat dibat dari predefined variables .Predefined Variables atau

Variabel Sistem PHP, adalah beberapa variabel yang telah didefenisikan secara sistem oleh PHP, dan kita sebaiknya tidak membuat variabel dengan nama yang sama.

Beberapa contoh Predefined Variables dalam PHP adalah:

$GLOBALS , $\_SERVER , $\_GET , $\_POST , $\_FILES , $\_COOKIE , $\_SESSION , $\_REQUEST , $\_ENV, $php\_errormsg, $HTTP\_RAW\_POST\_DATA, $http\_response\_header, $argc, $argv, $this.

1. **Variabel**

Variabel Program adalah sebuah abstraksi dari sel memori komputer atau kumpulan sel. Programmer sering memikirkan nama variabel sebagai nama untuk pengalokasian memori, tapi ada banyak lagi dari sekedar nama.Langkah berubah dari bahasa mesin ke bahasa perakitan sebagian besar adalah menggantikan alamat mutlak memori numerik untuk data dengan nama, membuat program lebih mudah dibaca

Sebuah variabel dapat dicirikan sebagai atribut: (nama, alamat, nilai, jenis, seumur hidup, dan ruang lingkup):

* Nama

Nama variabel adalah nama yang paling umum digunakan dalam program.Mereka memudahkan programmer untuk memasukkan nilai dalam suatu variabel yang berbeda. Dalam konteks ini hal yang sama dengan penamaan yang telah dibahas di halaman sebelumnya

* Alamat

Alamat dari variabel adalah alamat memori mesin yang dikaitkan. Dalam banyak bahasa, adalah mungkin untuk variabel yang sama dihubungkan dengan alamat yang berbeda pada waktu yang berbeda dalam program. Sebagai contoh, jika subprogram memiliki variabel lokal yang dialokasikan dari tumpukan run-time ketika subprogram dipanggil , panggilan yang berbeda mungkin menghasilkan variabel memiliki alamat yang berbeda. Ini adalah arti inisiasi yang berbeda dari variabel yang sama.

Alamat dari variabel kadang-kadang disebut nilai l, karena alamatlah yang diperlukan ketika nama variabel muncul di sisi kiri sebuah tugas.Hal ini dimungkinkan untuk memiliki beberapa variabel yang memiliki alamat yang sama. Saat lebih dari satu nama variabel dapat digunakan untuk mengakses lokasi memori yang sama, variabel disebut alias. Aliasing adalah halangan untuk membaca karena memungkinkan variabel diubah nilainya untuk variabel yang berbeda. Sebagai contoh, jika variabel bernama total dan jumlah adalah alias, setiap perubahan nilai total juga perubahan nilai jumlah dan sebaliknya. Seorang pembaca dari program harus selalu ingat bahwa total dan jumlah adalah nama yang berbeda untuk sel memori yang sama.

Alias ​​dapat dibuat dalam program dalam beberapa cara berbeda.Salah satu cara umum di C dan C ++ adalah dengan jenis gabungan (union)mereka.

Dua variabel pointer adalah alias ketika mereka menunjuk ke lokasi memori yang sama. Hal yang sama berlaku untuk variabel referensi. Semacam ini aliasing hanyalah efek samping dari sifat pointer dan referensi. Ketika C ++ pointer diatur ke titik di bernama variabel, pointer, ketika di referensi ulang, dan nama variabel yang alias.

* Jenis/Tipe

Jenis variabel menentukan rentang nilai variabel dapat menyimpan dan set operasi yang didefinisikan untuk nilai-nilai dari jenis. Sebagai contoh, tipe int di Java menentukan berbagai nilai dari negative –2147483648 sampai positif 2147483647 dan operasi aritmatika untuk penambahan, pengurangan, perkalian, pembagian, dan modulus.

* Nilai

Nilai variabel adalah isi dari sel memori atau sel diasosiasikan dengan variabel.Lebih mudah untuk memikirkan memori komputer dalam hal sel abstrak, daripada sel fisik. Sel-sel fisik, atau individu unit dialamatkan, dari yang memori komputer kontemporer adalah ukuran byte, dengan byte biasanya adalah delapan bit panjangnya. Ukuran ini terlalu kecil untuk sebagian besar variabel program.Sebuah sel memori abstrak memiliki ukuran yang dibutuhkan oleh variabel yang terkait.Sebagai contoh, meskipun nilai-nilai floating-point dapat menempati empat byte fisik dalam implementasi tertentu dari bahasa tertentu, nilai floating-point dianggap menempati sel memori abstrak tunggal.Nilai dari masing-masing jenis nonstructured sederhana dianggap menempati sel abstrak tunggal.Sejak saat itu, sel memori jangka berarti sel memori abstrak.

Sebuah nilai variabel kadang-kadang disebut-nilai r nya karena itulah yang dibutuhkan ketika nama variabel muncul di sisi kanan pernyataan penugasan. Untuk mengakses nilai r, nilai-l harus ditentukan terlebih dahulu.Klasifikasi tersebut tidak selalu sederhana.

1. **Bindings**

**3.1 Konsep Bindings dan Bindingsnya**

Binding dalam suatu program merupakan suatu kumpulan/gabungan diantara atribut dengan komponen suatu program, contohnya diantara tipe data suatu nilai variable dengan suatu variable nama, nama dan lokasi memori, nama dan fungsi, suatu variable dan nilainya, atau diantara suatu operasi dan suatu symbol. Tetapi secara khusus, bindings merupakan kesatuan diantara suatu nama dengan objek yang dimaksudkan.

**3.2 Binding Time**

Binding time merupakan suatu momen didalam lingkaran proses suatu program dimana binding terjadi, singkatya, binding time adalah waktu ketika dilaksanakannya suatu binding.

Penggunaan binding time yang paling banyak digunakan

* Languange design time, menggabungkan symbol operator menjadi suatu operasi
* Language implementation, menggabungkan tipe floating point menjadi suatu

representasi

* Program translation/compile time, menggabungkan suatu variable ke suatu tipe
* Load time, menggabungkan variable static bahasa pemograman ke suatu sel memori
* Program execution/run time, menggabungkan variable local nonstatic menjadi sel memori

Contoh dalam bahasa PHP :

$count = $count + 5;

Beberapa binding dan waktu yang mengikat mereka untuk bagian-bagian dari tugas ini Pernyataan adalah sebagai berikut:

• Jenis *count* terikat pada compile time .

• Himpunan kemungkinan *count* dari jumlah terikat pada compiler design time.

• Arti dari simbol operator + terikat pada compile time ketika jenis operan yang telah ditentukan.

• Representasi internal dari literal 5 terikat pada desain compiler design time.

• Nilai hitung terikat pada waktu eksekusi(run time) dengan pernyataan ini.

Ada 2 tipe storage binding yaitu :

-Allocation : mengambil cell dari kolam memori cell yang tersedia

-Deallocation : sel memori yang telah tidak terikat dari variabel dikembali ke kolom memori yang tersedia.

* Tipe Binding secara umum :
* Binding Static

Suatu binding dinyatakan static ketika binding terjadi sebelum run time dan tetap tidak berubah selama ekseukis program berlangsung. Deklarasi eksplisit adalah pernyataan dalam program yang berisi daftar nama-nama variabel dan menetapkan bahwa mereka adalah jenis tertentu. Deklarasi implisit merupakan perkumpulan variabel dengan jenis melalui konvensi umum, bukan pernyataan deklarasi. Dalam hal ini, penampilan pertama dari nama variabel dalam Program merupakan deklarasi implisit. Kedua deklarasi eksplisit dan implisit membuat binding statis untuk jenis.

Bahasa pemrograman yang paling banyak digunakan yang menggunakan jenis statis mengikat eksklusif dan dirancang sejak pertengahan 1960-an membutuhkan deklarasi eksplisit dari semua variabel (Perl, JavaScript, Ruby, dan ML beberapa pengecualian).

Implisit variabel jenis binding dilakukan oleh prosesor bahasa, baik kompilator atau penerjemah.Ada beberapa dasar yang berbeda untuk implisit variabel jenis binding.Yang paling sederhana ini adalah penamaan konvensi.di hal ini, compiler atau interpreter mengikat variabel untuk tipe berdasarkan bentuk sintaksis nama variabel. Sebagai contoh, dalam Fortran, sebuah identifier yang muncul dalam sebuah program yang tidak secara eksplisit dinyatakan secara implisit menyatakan menurut konvensi berikut: Jika identifier dimulai dengan salah satu surat saya, J, K, L, M, N atau, atau versi huruf kecil mereka, itu adalah implisit dinyatakan jenis Integer; jika tidak, secara implisit dinyatakan menjadi tipe Real.

Meskipun memberikan kenyamanan kecil untuk programmer, deklarasi implisit dapat merugikan kehandalan karena mereka mencegah kompilasi Proses dari mendeteksi beberapa kesalahan ketik dan programmer.di Fortran, variabel yang sengaja dibiarkan dideklarasikan oleh programmer yang diberikan jenis standar dan mungkin atribut yang tak terduga, yang bisa menyebabkan haluskesalahan yang sulit untuk mendiagnosa. Banyak programmer Fortran sekarang termasuk deklarasi tidak Implisit dalam program mereka.Deklarasi ini menginstruksikan compiler untuk tidak secara implisit mendeklarasikan variabel apapun, sehingga menghindari potensi masalah variabel sengaja dideklarasikan.

Beberapa masalah dengan deklarasi implisit dapat dihindari dengan mewajibkan nama untuk jenis tertentu untuk memulai dengan karakter khusus tertentu. untuk Misalnya, di Perl setiap nama yang diawali dengan $ adalah skalar, yang dapat menyimpan baik string atau nilai numerik. Jika nama dimulai dengan @, itu adalah sebuah array; jika mulai dengan%, itu adalah structure.4 hash ini menciptakan ruang nama yang berbeda untuk berbeda ketik variabel. Dalam skenario ini, nama-namaapple dan% apel yang tidak terkait, karena masing-masing dari namespace yang berbeda.Selain itu, program pembaca selalu tahu jenis variabel ketika membaca namanya.Perhatikan bahwa desain ini berbeda dari Fortran, karena Fortran memiliki kedua deklarasi implisit dan eksplisit, sehingga jenis variabel dapat belum tentu ditentukan dari ejaan namanya.

Jenis lain dari deklarasi tipe implisit menggunakan konteks. Ini kadang-kadang disebut inferensi tipe.Dalam kasus sederhana, konteksnya adalah jenis nilai ditugaskan untuk variabel dalam sebuah pernyataan deklarasi.Misalnya, di C # var deklarasi variabel harus menyertakan nilai awal, yang tipe dibuat dengan jenis variabel. Pertimbangkan deklarasi berikut:

var sum = 0;

var total = 0.0;

Nama var = "Fred";

Jenis-jenis sum, total, dan nama yang int, float, dan tali, masing-masing. Perlu diingat bahwa ini statis diketik variabel-mereka jenis yang tetap untuk masa unit di mana mereka dinyatakan.

* Binding Dinamis

Suatu binding dinyatakan dinamis apabila binding terjadi ketika eksekusi dilakukan atau dapat kita ganti ketika eksekusi suatu program dilaksanakan. Dengan tipe binding dinamis, jenis variabel tidak ditentukan oleh deklarasi pernyataan, juga tidak bisa ditentukan oleh ejaan namanya.Sebagai gantinya, variabel terikat ketika ditugaskan nilai dalam sebuah pernyataan penugasan.Ketika pernyataan penugasan dijalankan, variabel yang ditugaskan terikat dengan jenis nilai ekspresi pada sisi kanan tugas.Seperti tugas juga dapat mengikat variabel ke alamat dan sel memori, karena nilai-nilai berbagai jenis mungkin memerlukan jumlah yang berbeda dari penyimpanan.Setiap variabel dapat ditugaskan setiap nilai tipe.Selain itu, variabel Jenis dapat mengubah beberapa kali selama eksekusi program.Itu penting untuk menyadari bahwa jenis variabel yang jenis dinamis terikat mungkin sementara.

Keuntungan utama dari binding dinamis variabel terikat untuk jenis adalah bahwa ia menyediakan lebih pemrograman fleksibilitas. Sebagai contoh, sebuah program untuk memproses data numerik dalam bahasa yang menggunakan tipe dinamis mengikat dapat ditulis sebagai program generik, yang berarti bahwa ia mampu menangani data jenis numerik. Apapun tipe data adalah masukan akan diterima, karena variabel di mana data yang akan disimpan dapat terikat untuk jenis yang tepat saat data ditugaskan untuk variabel setelah memasukkan. Sebaliknya, karena mengikat statis dari jenis, satu tidak bisa menulis C program untuk memproses data tanpa mengetahui jenis data yang.

Sebelum pertengahan 1990-an, bahasa pemrograman yang paling umum digunakan digunakan tipe statis mengikat, pengecualian utama adalah beberapa fungsional bahasa seperti LISP. Namun, sejak saat itu telah terjadi pergeseran yang signifikan untuk bahasa yang menggunakan tipe dinamis mengikat.Python, Ruby, JavaScript, dan PHP, jenis mengikat dinamis. Misalnya, script JavaScript mungkin berisi pernyataan berikut:

list = [10.2, 3.5];

Terlepas dari jenis sebelumnya variabel bernama daftar, tugas ini menyebabkan ia menjadi nama array tunggal dimensi panjang 2. Jika pernyataan

list = 47;

mengikuti contoh tugas sebelumnya, daftar akan menjadi nama variabel skalar.

Ada dua kerugian untuk tipe dinamis mengikat. Pertama, hal itu menyebabkan program menjadi kurang dapat diandalkan, karena kemampuan error-deteksi compiler berkurang relatif terhadap compiler untuk bahasa dengan jenis statis binding. Jenis dinamis mengikat memungkinkan variabel apapun yang akan ditugaskan nilai dari jenis apa pun. Jenis yang salah dari sisi kanan tugas tidak terdeteksi sebagai kesalahan; bukan, jenis sisi kiri hanya berubah ke salah ketik. Sebagai contoh, misalkan dalam program JavaScript tertentu, saya dan x saat ini nama-nama variabel numerik skalar dan y adalah saat ini nama dari array. Selanjutnya, anggaplah bahwa program perlu penugasan pernyataan

i = x;

tetapi karena kesalahan kata kunci, ia memiliki pernyataan penugasan

i = y;

Dalam JavaScript (atau bahasa lain yang menggunakan tipe dinamis mengikat), tidak ada kesalahan terdeteksi dalam pernyataan ini oleh interpreter-jenis variabel bernama i hanya berubah ke array. Tapi penggunaan kemudian saya akan berharap untuk menjadi skalar, dan hasil yang benar akan menjadi mustahil. Dalam bahasa dengan jenis statis mengikat, seperti Java, kompiler akan mendeteksi kesalahan dalam tugas saya = y, dan program tidak akan bisa eksekusi.

Kelemahan terbesar dari tipe dinamis mengikat adalah biaya.Biaya pelaksanaan atribut dinamis mengikat cukup besar, terutama di waktu eksekusi.Memeriksa jenis harus dilakukan pada waktu berjalan. Selain itu, setiap variabel harus memiliki deskriptor run-time yang terkait dengan itu untuk menjaga arus ketik. Penyimpanan digunakan untuk nilai variabel harus dari berbagai ukuran, karena nilai-nilai jenis yang berbeda membutuhkan jumlah yang berbeda dari penyimpanan.

Akhirnya, bahasa yang memiliki tipe dinamis mengikat untuk variabel biasanya diimplementasikan menggunakan interpreter murni daripada compiler.Komputer tidak memiliki instruksi yang operan jenis tidak diketahui pada waktu kompilasi.Oleh karena itu, kompilator tidak bisa membangun instruksi mesin untuk ekspresi A + B jika jenis A dan B tidak diketahui pada waktu kompilasi.Interpretasi murni biasanya Dibutuhkan setidaknya 10 kali lebih lama seperti halnya untuk mengeksekusi kode mesin setara.Tentu saja, jika bahasa diimplementasikan dengan juru murni, waktu untuk melakukan Tipe binding dinamis disembunyikan oleh waktu keseluruhan interpretasi, sehingga tampaknya lebih murah di lingkungan itu. Di sisi lain, bahasa dengan jenis statis binding jarang dilaksanakan oleh interpretasi murni, karena program di bahasa ini dapat dengan mudah diterjemahkan ke sangat efisien kode mesin versi.

Lifetime variable dimulai ketika cell spesifik terikat, dan berakhir ketika tidak terikat dari cell itu.

Terdapat 4 kategori tipe binding berdasarkan *lifetime* yaitu :

1. Static

Static = sel yang terikat pada memory cell sebelum eksekusi dimulai, dan tetap terikat pada memory cell yang sama sampai program eksekusi selesai.

Kelebihan : semua addressing pada static variable bersifat langsung.

Kekurangan : mengurangi fleksibilitas(tidak support untuk reculsive program)

2. Stack dynamic

Stack dynamic variable = storage binding dibuat ketika deklarasi statement dijelaskan, tapi tipe nya bersifat terikat.

Kelebihan : support dynamic storage(reculsive)

Kekurangan : waktu akses yang lambat karena addressing tidak langsung

3.  Explicit heap dynamic

Explicit heap dynamic = abstrak memory cell yang di alokasi dan dealokasi dengan instruksi eksplicit run-time yang ditulis oleh programmer. Variable nya hanya bisa dialokasikan oleh pointer atau reference variable. Sebagai contoh :

int \*intnode; // Create a pointer

intnode = new int; // Create the heap-dynamic variable

. . .

delete intnode; // Deallocate the heap-dynamic variable

// to which intnode points

Explicit heap dynamic variable pada tipe int dibuat oleh operator new. Variable ini bisa di referensikan melalui pointer , intnode. Nantinya variable di deallocation dengan deleteoperator.

Kelebihan : storage manajemen yang dinamis.

Kekurangan : tidak efisien dan tidak handal.

4.  Implicit heap dynamic

Implicit heap dynamic terikat dengan heap storage ketika ada nilai yang ditugaskan.

Sebagai contoh:

highs = [74, 84, 86, 90, 71];

variable bernama highs sudah di deklarasi di sebelumnya, dan sekarang digunakan pada array 5 data.

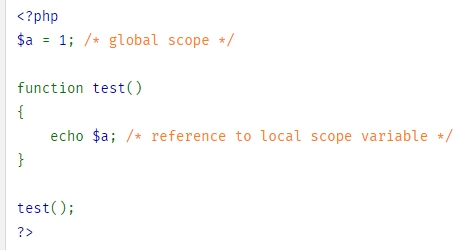
Kelebihan : memiliki fleksibilitas yang memperbolehkan high generic code untuk ditulis ulang.

Kekurangan : run-time overhead mempertahankan semua atribut dinamis, yang akan mencakup jenisarray subscript dan range.

1. **Scope**

Scope adalah ruang lingkup suatu variable dimana jangkauan perintah program masih bisa mengakses suatu variable. Jika kita mendefenisikan sebuah variabel, maka variabel tersebut dapat diakses oleh seluruh kode program pada halaman yang sama. Namun jika variabel tersebut di defenisikan di dalam sebuah fungsi, variabel itu belum tentu bisa diakses dari luar fungsi tersebut. Hal inilah yang dimaksud dengan scope.

Variabel yang didefenisikan di dalam sebuah fungsi, secara default tidak dapat diakses oleh kode program di luar fungsi tersebut hal ini yang disebut **local variabel**. Dan begitu juga sebaliknya, variabel yang didefenisikan di luar fungsi, tidak bisa diakses dari dalam fungsi.Jika kita tetap ingin menggunakan variabel yang didefenisikan di luar fungsi dan sebaliknya, PHP memperbolehkan hal tersebut dengan menambahkan sebuah kata kunci ‘global’ sebelum pendefenisian nama variable variabel ini disebut **global variabel**. Kata kunci global membuat fungsi dapat mengakses variabel yang didefenisikan diluar fungsi atau dengan kata lain variable global dapat digunakan oleh semua fungsi dari sebuah program.



Contoh global dan local scope

Static variable adalah jenis variabel yang mempertahankan nilainya pada setiap pemanggilan fungsi. Untuk variabel normal, nilai dari variabel tersebut akan secara otomatis dihapus pada saat fungsi selesai dijalankan, dan akan dibuat ulang pada saat fungsi dipanggil. Namun jika sebuah variabel dinyatakan sebagai static variabel, maka nilai variabel tersebut akan tetap dipertahankan walaupun fungsi telah selesai dijalankan.

Dynamic variable adalah didasarkan pada urutan pemanggilan subprogram, bukan pada hubungan spasial mereka satu sama lain. Tapi ruang lingkup dapat ditentukan hanya pada run time.Beberapa masalah datang dari scoping dinamis:

Pertama, selama rentang waktu awal ketika subprogram dimulai pelaksanaannya dan berakhir ketika eksekusi yang berakhir, variabel lokal dari subprogram semua terlihat setiap subprogram mengeksekusi lain, terlepas dari kedekatan tekstual. Tidak ada cara untuk melindungi variabel lokal dari aksesibilitas ini. Subprogram selalu dilaksanakan di lingkungan terdekat dari si pemanggil; Oleh karena itu hasil scoping dinamis dalam program kurang dapat diandalkan dibandingkan scoping statis.

Juga, scoping dinamis membuat program menentukan makna referensi untuk variabel non-lokal. Ini bisa menjadi mungkin bagi pembaca manusia.lebih sulit untuk dibaca, karena urutan pemanggilan subprogram harus diketahui untuk

**BAB 4**

**Data Type**

Data type menjelaskan kumpulan objek data dan sebuah set operasi yang telah ditetapkan pada objek tersebut. Descriptor adalah kumpulan atribut dari sebuah variabel.Sebuah objek mewakili tipe user-defined (data abstrak).

1. **Primitive Data Types**

Tipe data primitive adalah Tipe data yang mampu menyimpan satu nilai tiap satu variabel. Tipe data primitive merupakan tipe data dasar yang sering dipakai oleh program. Contoh tipe data primitive adalah tipe numerik (integer dan real), tipe data karakter/char, tipe data boolean.

Dalam PHP tipe data tidak dideklarasikan oleh programmer akan tetapi ditentukan secara otomatis oleh Intepreter PHP. Jadi anda tidak perlu deklarasikan tipe data tersebut jika ingin memberikan nilai pada suatu variabel.

Ada 8 macam tipe data pada PHP, diantaranya :

**1.1** **Integer**

Tipe data integer adalah tipe data yang berguna untuk menyimpan bilangan bulat, bukan desimal. Sebagai contoh (1),(2),(3),(4),(5),(-1),(-2),(-3),(-4) dan lain-lain. Tipe data ini memiliki range antara -2,147,483,648 sampai dengan +2,147,483,648 platform 32bit.

Namun bila terdapat data diluar jangkauan tersebut, maka secara otomatis [PHP](http://viruspintar.blogspot.com/) akan mengkonversikannya ke dalam tipe data Floating Point.

Integer merupakan Tipe Data yang berisikan bilangan bulat negatif dan positif atau tanpa ada nilai pecahan.Tipe ini juga dapat dinyatakan dalam bentuk oktal (berbasis 8), desimal (berbasis 10), heksadesimal (berbasis 16). Seperti dalam contoh berikut :

$oktal = 031;

$desimal = 25;

$heksadesimal = 0×10;

**1.2.** **Double Floating**

Tipe data floating point numbers biasa juga disebut dengan “double”, ” float” atau “real” adalah tipe data yang berguna untuk menyimpan bilangan desimal. Sebagai contoh (0.1),(1.3),(1.7),(1.8),(9.7),(2.4) dan lain-lain.

Contoh penulisannya :

<?php

$a = 1.234;

$b = 1.2e3;

$c = 2E-6;

$d = 1.2E8;

?>

**1.3.**  **Boolean**

Tipe Data ini adalah tipe data yang paling sederhana. Hanya berupa **true** atau **false**. Cara memasukan ke dalam variabel adalah dengan memberikan nilai **true** atau **false**pada variabel tersebut. Penulisannya tidak dipengaruhi penggunaan huruf besar atau kecil.

Contoh penulisannya :

     <?php

    $a = TRUE; // bernilai 1

    $b = true; // bernilai 1

      $c = FALSE; // bernilai 0

      $d = false; // bernilai 0

      ?>

Setiap Tipe Data dalam PHP mempunyai Boolean untuk melakukan seleksi kondisi. dan Tipe Data tersebut akan memberikan nilai **true** atau **false** pada seleksi kondisi tersebut.

* ·Pada Variabel yang bertipe Data integer=0 (nol) mempunyai boolean false, selain itu true.
* ·Pada Variabel yang bertipe Data float=0.0 (nol) mempunyai boolean false, selain itu true.
* ·Pada Tipe Data string yang kosong, dan string="0" mempunyai boolean false,selain itu true.
* ·Array yang tidak mempunyai element mempunyai boolean false, selain itu true.
* ·Objek yang tidak mempunyai member mempunyai boolean false, selain itu true.
* ·Tipe NULL dan Variabel yang belum terisi mempunyai boolean false.

**1.4.**   **String**

String adalah sebuah tipe data yang terdiri dari kata, bisa berupa kata tunggal maupun kalimat. Penulisan string harus diapit oleh tanda petik, baik petik tunggal(‘ ‘) maupun petik ganda (” “)

|  |  |
| --- | --- |
| Contoh penulisan tipe data string : | |
|  | <?php  $kata1 = 'string diapit petik tunggal';  $kata2 = "string diapit petik ganda";  ?> |

Perbedaan antara petik ganda dan petik tunggal adalah, ketika kita menggunakan petik tunggal maka variabel tidak akan terbaca nilainya. Beda halnya dengan penggunaan petik ganda yang akan membaca nilai dari suatu variabel.

Lebih spesifiknya :

<?php

$kata = "Haloo Berooo";

echo 'nama saya adalah $kata';

//jika menggunakan tanda petik tunggal browser

//akan menampilkan $kata

echo "nama saya adalah $kata";

//jika menggunakan tanda petik tunggal browser

//akan menampilkan Haloo Beroo

?>

Jika string yang tertutup dalam tanda kutip ganda (” “), PHP akan menafsirkan escape sequence lebih untuk karakter khusus. Maka penulisan untuk karakter khusus tersebut harus ditambahkan slash didepan karakter tersebut.

**1.5.**   **Objek**

Tipe data object dibuat dengan tujuan agar para programmer terbiasa dengan OOP. Tipe data object bisa berupa bilangan, variabel atau fungsi. Data ini dapat dimasukkan ke dalam kode program sehingga meringkas beberapa fungsi dan memperkecil ukuran file.

Agar lebih mudah dapat menggunakan variabel dengan nilai string :

class Nama

{

    var $jenis = “Beroo Haloo”;

    function ganti\_nama($jenis)

{

       $this->jenis = $jenis;

      }

}

$nama = new Nama;

echo $nama->jenis;

$nama->ganti\_nama(“Haloo Beroo”);

echo “<br>”.$nama->jenis;

Hasilnya :

Beroo Haloo

Haloo Beroo

**1.6.**   **Array**

Array atau Larik merupakan Tipe Compound Primitif, terdapat pada bahasa-bahasa pemrograman lain.Tipe data array digunakan untuk menyimpan banyak data dalam satu variabel. Jenis array dalam php ada 3 (tiga) macam yakni :

* Numeric Array

<?php

$hari[0]="Senin";

$hari[1]="Selasa";

$hari[2]="Rabu";

$hari[3]="Kamis";

$hari[4]="Jum'at";

$hari[5]="Sabtu";

$hari[6]="Minggu";

//menampilkan tanpa perulangan.

echo "sekarang adalah hari $hari[3] <br />" ;

//menampilkan menggunakan perulangan

for($i=0;$i<=6;$i++){

echo $hari[$i]."<br/>";

}

?>

Pada contoh numeric array diatas dapat disimpulkan bahwa pemanggilan data array tersebut menggunakan angka, angka dimulai dari 0 sampai jumlah data yang dimasukkan. $hari[0] adalah pemanggilan untuk data pertama, $hari[1] adalah pemanggilan untuk data kedua dan seterusnya.

* Associative Array

<?php

$hari = array("Senin"=>"Ayam","Selasa"=>"Lele",

"Rabu"=>"Tempe","Kamis"=>"Tahu",

"Jum”at"=>"Teri","Sabtu"=>"Krupuk","Minggu"=>"Puasa");

//menampilkan tanpa perulangan.

echo "Menu hari senin adalah $hari[Senin]" ;

?>

Pada contoh associative array diatas dapat disimpulkan bahwa pemanggilan data array tersebut menggunakan [string](file:///F:\\web\\Tipe%20Data%20Dalam%20PHP%20_%20Kode%20PHP.htm" \l "con_string)yang ditentukan oleh kita, dengan kata lain pemanggilan menggunakan angka kita ganti menjadi string yang kita inginkan.

* Multidimensional Array

Pengertian multidimension arrayadalah array didalam array. Pemanggilan array ini bisa menggunakan numeric array maupun associative array. $makan[0]["pagi"] dapat diartikan sebagai menampilkan nilai array pertama dan array pagi dari variabel makan, $makan["senin"]["pagi"] dapat diartikan sebagai menampilkan nilai array senin dan array pagi dari variabel makan.

<?php

$makan = array(

array("pagi"=>"bubur","siang"=>"ayam","sore"=>"bebek"),

array("pagi"=>"bubur","siang"=>"tahu","sore"=>"tempe")

);

//menampilkan tanpa perulangan.

echo "Menu makan pada hari senin pagi adalah " . $makan[0]["pagi"];

echo "<br />";

echo "Menu makan pada hari senin pagi adalah " . $makan[1]["sore"];

?>

**1.7.**   **Null**

NULL adalah Tipe Data yang tidak memuat apapun. Setiap Variabel yang diset menjadi Tipe Data NULL ini akan menjadikan Variabel tersebut **kosong**, seperti dalam penggunaan fungsi **unset()**. Cara untuk menset Variabel menjadi NULL adalah dengan memasukan nilai NULL pada variabel yang dimaksud seperti pada

Contoh berikut:

<?php

# Nilai awal $foo adalan 10

$foo = 10;

# $foo akan menjadi NULL

$foo = NULL;

# Sama dengan memanggil unset()

unset($foo);

?>

**1.8.**   **Resources**

Tipe Data Spesial yang satu ini di khususkan untuk menyimpan*resource,* sumber atau alamat. Variabel tersebut hanya dapat diciptakan oleh suatu fungsi khusus yang mengembalikan nilai berupa resource seperti penggunaan fungsi **fopen, opendir, mysql\_connect, mysql\_query** dan semacamnya.

**2.Character String Types**

Dalam bahasa PHP Tipe data string adalah tipe data yang berisi text, kalimat, atau kumpulan karakter. Sebagai contoh, “a”, “saya sedang belajar PHP” atau “tUT0r1al pHp?!” semuanya adalah string.PHP menyediakan 4 cara penulisan tipe data string, yakni **Single Quoted**, **Double Quoted**, **Heredoc**, dan **Nowdoc**.

1. Penulisan Tipe Data String dengan Single Quoted

Penulisan tipe data string menggunakan *single quoted* atau tanda petik satu (karakter ‘) merupakan cara penulisan string yang paling sederhana. Kita tinggal membuat sebuah kata atau kalimat, dan menambahkan tanda petik satu di awal dan akhir kalimat.

Untuk string yang didalamnya juga terdapat tanda petik satu, kita harus mendahuluinya dengan karakter *backslas*h (\) agar tidak dianggap sebagai penutup string. Dan jika di dalam string anda ingin menulis tanda *backslash*, kita harus menulisnya dengan 2 kali (\\).Berikut adalah contoh penulisan tipe data string menggunakan metode single quoted:

<?php

$string1='Ini adalah string sederhana';

$string2='Ini adalah string

yang bisa memiliki beberapa

baris';

$string3='Dia berkata: "I\'ll be back"';

$string4='Anda telah berhasil menghapus C:\\xampp\\htdoc';

$string5='Kalimat ini tidak akan pindah ke: \n baris baru';

$string6='Variabel juga tidak otomatis ditampilkan $string1 dan $string3';

echo $string1; echo "<br>";

echo $string2; echo "<br>";

echo $string3; echo "<br>";

echo $string4; echo "<br>";

echo $string5; echo "<br>";

echo $string6;

?>

#### 2. Penulisan Tipe Data String dengan Double Quoted

Cara kedua dalam penulisan tipe data string dalam PHP adalah dengan *menggunakan Double Quoted* atau tanda petik dua (karakter “). Walaupun seperti tidak ada perbedaan dengan menggunakan single quote, hasil yang di dapat akan sangat berbeda.

Dengan double quoted, PHP akan memproses karakter-karakter khusus seperti carriage return (\n), dan karakter tab (\t) dan juga memproses setiap variabel (yang ditandai dengan tanda $ didepan kata).

Di karenakan metode double quoted melakukan pemrosesan terlebih dahulu, maka untuk menampilkan karakter khusus seperti tanda petik (karakter ‘), tanda dollar (karakter $) dan tanda-tanda khusus lainnya, kita harus menggunakan backslash (karakter \). Berikut adalah contoh penulisan tipe data string menggunakan metode double quoted:

<?php

$string1="Ini adalah string sederhana";

$string2="Ini adalah string

yang bisa memiliki beberapa

baris";

$string3="Dia berkata: \"I'll be back\"";

$string4="Anda telah berhasil menghapus C:\\xampp\\htdoc";

$string5="Kalimat ini akan akan pindah ke: \n baris baru";

$string6="Variabel akan otomatis ditampilkan: $string1 dan $string3";

echo $string1; echo "<br \>";

echo $string2; echo "<br \>";

echo $string3; echo "<br \>";

echo $string4; echo "<br \>";

echo $string5; echo "<br \>";

echo $string6;

?>

Perhatikan perbedaannya pada hasil **$string3**, **$string5** dan **$string6**. Pada **$string3**, kita harus mem*-blackslash* tanda petik dua karena itu merupakan karakter khusus dalam **double quoted string**.

Pada **$string5**, tanda **\n** yang merupakan karakter khusus untuk baris baru, tapi karena kita menampilkannya di browser, karakter ini tidak akan terlihat, tetapi jika kita menulis hasil string ini kedalam sebuah file text, kalimat tersebut akan terdiri dari 2 baris.

Pada **$string6**, terlihat bahwa string dengan petik dua akan memproses variabel **$string1** dan **$string3** sehingga tampil hasilnya di *web browser*. Fitur ini akan sangat bermanfaat jika kita sering menampilkan variabel didalam sebuah string.

#### 3. Penulisan Tipe Data String Dengan Heredoc

Cara penulisan tipe data string yang ketiga yaitu dengan fitur PHP yang disebut **heredoc**. Fitur ini digunakan untuk membuat tipe data string yang dapat berisi beberapa baris kalimat. Dibandingkan dengan menggunakan single quote dan double quote, pembuatan string dengan heredoc tidak terlalu sering digunakan.

Agar lebih jelas, berikut adalah contoh penulisan tipe data string dengan heredoc:

<?php

$IPK=3.9;

$string1 = <<<end

Saya sedang belajar PHP

di duniailkom.com <br />

Kali ini tentang pembahasan

mengenai "PHP", <br /> dan berharap

bisa dapat IPK $IPK :)

end;

echo $string1;

?>

Seperti yang terlihat dari contoh diatas, fitur Heredoc ditandai dengan tanda “<<<” untuk memulai string, lalu diikuti dengan karakter penanda akhir string. Dari contoh tersebut kata “end” pada awal string adalah penanda akhir string. Anda bebas mengganti kata “end” dengan kata atau karakter lain, sepanjang kata tersebut bisa dijamin tidak akan muncul didalam string.

Setelah karakter penanda string, baris pertama setelahnya adalah awal dari string. String ini dapat mencakup beberapa baris, sampai ditemukan karakter penanda akhis string yang kita definisikan di awal (yaitu kata “end”). Setelah ditemukan karakter penanda akhir string, maka pendefenisian string berakhir.

Perhatikan juga bahwa di dalam kalimat diatas, kita menggunakan karakter \n dan variabel $IPK. Seluruh karakter ini diproses oleh PHP, sehingga mirip dengan fitur double quoted string.

#### 4. Penulisan Tipe Data String Dengan Nowdoc

Cara penulisan tipe data keempat dalam PHP yaitu dengan fitur Nowdoc. Fitur ini hampir sama dengan Heredoc, namun dengan pengecualian: karakter khusus dan variabel tidak akan diproses oleh PHP, atau mirip dengan single quoted string.

Berikut adalah contoh penulisan tipe data string menggunakan metode Nowdoc:

<?php

$IPK=3.9;

$string1 = <<< 'selesai'

Saya sedang belajar PHP

di \n duniailkom.com <br />

Kali ini tentang pembahasan

mengenai "PHP", <br /> dan berharap

bisa dapat IPK $IPK :)

selesai;

echo $string1;

?>

Jika dilihat sekilas, tidak ada perbedaan cara penulisan metode nowdoc dengan heredoc, namun perhatikan karakter penanda akhir string. Kali ini kita menggunakan karakter ‘selesai’ sebagai penanda akhir string. Dan yang membedakannya dengan heredoc adalah, nowdoc menambahkan single quoted untuk karakter penanda akhir string. Kita menulis ‘selesai’ (dengan tanda kutip satu) untuk mengawali string.

Dari tampilan yang dihasilkan, nowdoc memproses string sama dengan single quoted string, dimana karakter khusus dan variabel tidak diproses sama sekali, sehingga dalam tampilan akhir anda dapat melihat tanda \n dan variabel $IPK ditulis sebagai string.

**3.User-Defined Ordinal Types**

Tipe data ordinal adalah satu dari sekian jangkauan nilai yang dapat dengan mudah di hubungkan dengan sekumpulan integer - integer positif.  
Secara umum ada dua jenis tipe data ordinal :

1. Tipe enumerasi adalah sebuah kumpulan nilai pasti seperti nama konstan yang tersedia dan disebut satu-persatu dalam definisi. Fungsinya untuk memudahkan dalam tipe data grup. Enumeration type diimplementasikan sebagai Integer.

Contoh pada bahasa C# :

**enum** days {mon, tue, wed, thu, fri, sat, sun};

1. Tipe subrange diimplementasikan sebagai bagian dari tipe data lain dan berada dalam sebuah jangkauan yang telah di tetapkan oleh programmer. Subrange types diimplementasikan

seperti tipe parent dengan kode dimasukkan (oleh kompiler) untuk membatasi tugas untuk

subrange variabel

Contoh pada bahasa Ada :

**type** Days **is** (mon, tue, wed, thu, fri, sat, sun);

**subtype** Weekdays **is** Days **range** mon..fri;

**subtype** Index **is** Integer **range** 1..100;

Day1: Days;

Day2: Weekday;

Day2 := Day1;

**4.Array Types**

Array adalah kumpulan dari beberapa elemen sejenis dari elemen data yang masing-masing elementnya dapat diidentifikasi dari posisinya (*index)*pada kumpulan data yang terkait pada elemen pertama.*Indexing* adalah pemetaan dari penunjuk ke elemen data.Contoh array dan indexing dalam bahasa pemograman PHP adalah :

<?php

$cars = array("Volvo", "BMW", "Toyota");

echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";

?>

Slice adalah beberapa substruktur dari sebauh array dan tidak lebih dari sebuah mekanisme referensi dan hanya berfungsi baik pada bahasa pemograman yang memeliki operasi *array.*Contoh pada bahasa pemograman *Phyton*:

vector = [2, 4, 6, 8, 10, 12, 14, 16] vector (3:6) is a three-element array mat = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] mat[0][0:2] is the first and second element of the first row of mat

Subscript Binding and Array Categories

-          Static = rentang subscript yang statis terikat dan alokasi penyimpanan statis (dilakukan

sebelum run time).Contohnya pada bahasa C dan C++.

-          Fixed stack-dynamic = rentang subscript yang statis terikat, namun alokasi dilakukan pada

saat deklarasi elaborasi selama eksekusi.Contoh pada bahasa

pemograman C#.

-          Stack-dynamic = rentang subscript dan alokasi penyimpanan dinamis terikat pada

waktu elaborasi. Contohnya pada bahasa C dan C++.

-          Fixed heap-dynamic = rentang subscript dan storage binding keduanya tetap, setelah

penyimpanan dialokasikan. Contohnya pada bahasa C dan C++.

-          Heap-dynamic = pengikatan rentang subscript dan alokasi penyimpanan dinamis dan dapat

berubah beberapa kali selama seumur hidup array. Contohnya pada bahasa

C dan C++,Perl,JavaScript,Phython dan Ruby.

Tipe-tipe array :

1. *Heterogeneous Array*

* Heterogeneous array adalah satu di mana unsur-unsur tidak perlu dari jenis yang sama.
* Rectangular array adalah array multidimensional di mana semua baris memiliki jumlah yang sama elemen dan semua kolom memiliki jumlah yang sama dari elemen.Array Rectangular bermodelkan meja persegi panjang. jagged array adalah di mana panjang dari baris tidak perlu menjadi sama. Sebagai contoh, sebuah matriks jagged dapat terdiri dari tiga baris, satu dengan 5 elemen, satu dengan 7 elemen, dan satu dengan 12 elemen.

*Heterogeneous Array* hanya terdapat dalam bahasa pemograman Perl, Python, JavaScript,

and Ruby

1. *Associative Array*

Associative Arrays adalah koleksi tak berurut elemen data yang diindeks oleh jumlah yang sama nilai-nilai yang disebut *key.*Contoh pada bahasa pemograman PHP :

<?php  
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");  
echo "Peter is " . $age['Peter'] . " years old.";  
?>

1. Tipe *Record*

Tipe *Record* adalah kumpulan elemen data yang masing-masing elemen dapat dikenali dengan nama.Contoh penggunaan tipe *record* pada pemograman bahasa COBOL,pengunaannya pada bahasa Java dan C# didefinisikan sebagai data kelas dengan perulangan *record*  juga dianggap sebagai *nested class :*

employee.name =”Freddie”

employee.hourlyRate=13.20

Pernyataan *assignment* membentuk table (record) dengan nama employee dan dengan 2 elemen name and hourlyRate yang telah diinialisasi.

1. Tipe *Tuple*

Tipe *Tuple* adalah tipe data yang sama dengan tipe data *record* ,namun tidak pemberian nama pada elemen.Contoh pada bahasa pemograman F# :

**let** tup = (3, 5, 7) **let** a, b, c = tup This assigns a tuple to a tuple pattern (a, b, c)

1. Tipe *List*

Lists pertama kali didukung dalam bahasa pemrograman fungsional pertama, LISP.Pada bahasa pemograman LISP penggunaan *list*  di batasi dengan kurung dan tidak menggunakan koma ,seperti :

(A B C D)  atau (A (B C) D)

Sedangkan dalam bahasa pemograman PHP,*list* adalah menetapkan seolah-olah mereka adalah array. Seperti array (), ini bukan fungsi, tetapi “language construct”. list () digunakan untuk menetapkan daftar variabel dalam satu operasi.Contoh penggunaan tipe *list* pada bahasa pemograman PHP :

<?php $info = array('coffee', 'brown', 'caffeine'); // Listing all the variables list($drink, $color, $power) = $info; echo "$drink is $color and $power makes it special.\n";

// Listing some of them list($drink, , $power) = $info; echo "$drink has $power.\n";

// Or let's skip to only the third one list( , , $power) = $info; echo "I need $power!\n";

// list() doesn't work with strings list($bar) = "abcde"; var\_dump($bar); // NULL ?>

1. Tipe *Union*

*Union* adalah jenis yang variabel dapat menyimpan nilai-nilai berbagai jenis di berbagai kali selama pelaksanaan program.Tipe *union* biasa dipakai pada bahasa pemograman C,C++,F#,dan Ada. Sebagai contoh kebutuhan untuk jenis *union*,

mempertimbangkan tabel konstanta untuk compiler, yang digunakan untuk menyimpan

konstanta ditemukan dalam program yang dikompilasi.Contoh penggunaan union pada bahasa C :

**union** flexType {

**int** intel;

**float**  floatEL;

**};**

**union** flexType el1;

**float x;**

….

el1.intEL=27;

x=el1.floatEL;

1. Tipe *Pointer* dan *Reference*

Tipe *pointer* dalah satu di mana variabel memiliki rentang nilai yang terdiri alamat memori dan nilai khusus, nil. Nilai nil bukan alamat yang valid dan digunakan untuk menunjukkan bahwa pointer saat ini tidak dapat digunakan untuk referensi sel memori. Tipe *pointer* menyediakan cara untuk mengurus memori dinamis dan *indirect addressing.*Sebuah pointer dapat digunakan untuk mengakses lokasi dimana penyimpanan dibentuk secara dinamis yang disebut *heap*.Dalam PHP, tidak ada pointer, hanya ada penggunaan tipe *references*.

Tipe reference mirip dengan pointer, dengan perbedaan mendasar: Sebuah pointer mengacu ke alamat di memori, sementarareferensi mengacu pada suatu objek atau nilai dalam memori. Akibatnya, meskipun itu natural untuk melakukan aritmatika pada alamat, tidak masuk akal untuk melakukan aritmatika pada referensi.Referensi dalam PHP merupakan sarana untuk mengakses variabel konten yang sama dengan nama yang berbeda.

**Assign by reference**

Referensi PHP memungkinkan Anda untuk membuat dua variabel merujuk pada konten yang sama. Artinya, ketika Anda lakukan:

<?php

$a =& $b;

?>

itu berarti bahwa $ a dan $ b dengan isi yang sama.

Note:$ a dan $ b adalah sama di sini. $ a tidak menunjuk ke $ b atau sebaliknya. $ a dan $ b yang menunjuk ke tempat yang sama.

* Passing by reference / passing dengan referensi

<?php

function foo(&$var)

{

$var++;

}

$a=5;

foo($a);

// $a is 6 here

?>

Tidak ada tanda referensi pada fungsi call - only pada definisi fungsi. Definisi fungsi saja sudah cukup benar untuk lulus argumen dengan referensi. Pada PHP 5.3.0, Anda akan mendapatkan peringatan yang mengatakan bahwa "call-time pass-by-reference" sudah ditinggalkan ketika anda menggunakan & di foo(& $ a);. Pada PHP 5.4.0, “call-time pass-by-reference” telah dihapus, jadi menggunakan itu akan meningkatkan fatal eror/kesalahan fatal.

* Returning references / kembali referensi

Returning by reference berguna saat Anda ingin menggunakan fungsi untuk menemukan yang variabel referensi harus terikat. Jangan menggunakan return-by-reference untuk meningkatkan kinerja.

* Unsetting references

Ketika Anda Unset References, Anda hanya memecahkan ikatan antara nama variabel dan isi variabel. Ini tidak berarti bahwa konten variabel akan hancur. Sebagai contoh:

<?php

$a = 1;

$b =& $a;

unset($a);

?>

Tidak akan unset $b, hanya $a. Hal ini mungkin berguna untuk berpikir tentang hal ini sebagai analog dengan Unix panggilan unlink.

* Spotting references

Banyak konstruksi sintaks dalam PHP dilaksanakan melalui mekanisme referensi, jadi segala sesuatu yang disebutkan di sini tentang referensi mengikat juga berlaku untuk konstruksi ini. Beberapa konstruksi, seperti lewat dan kembali dengan referensi, yang disebutkan di atas. konstruksi lain yang menggunakan referensi adalah:

- Global references

Ketika Anda mendeklarasikan variabel sebagai $ var global yang Anda sebenarnya menciptakan referensi ke variabel global. Itu berarti, ini adalah sama dengan:

<?php

$var =& $GLOBALS["var"];

?>

Ini juga berarti bahwa unsetting $ var tidak akan unset variabel global.Dalam metode objek, $this selalu referensi ke objek pemanggil.

**5.Type Checking**

*Type checking* adalah aktivitas dalam memastikan bahwa operand dan operator yang dipakai telah sesuai saat compile-time maupun run-time merupakan tipe kompatibel. *Type checking* membantu mengurangi kemungkinan adanya bug dalam suatu program pada computer dan memastikan bahwa bagian – bagian dari program tersebut tetap terhubung.  
  
Jika sebuah spesifikasi sebuah bahasa memiliki peraturan penulisan yang kuat dan selalu mendeteksi error dalam penulisan maka bahasa tersebut adalah bahasa yang kuat (Strongly Typed Language). Akan tetapi jika tidak, maka bahasa tersebut adalah bahasa yang lemah (Weakly Typed Language). Tipe kompatibel adalah salah satu yang legal bagi operator, atau diperbolehkan di bawah aturan bahasa yang akan dikonversi secara implisit, dengan kode compiler yang dihasilkan, untuk tipe legal. Konversi otomatis ini disebut coercion.  
  
Data dapat dikatakan cocok apabila legal bagi operator atau diperbolehkan oleh peraturan bahasa untuk dikonversi secara implisit oleh compiler-generated code menjadi sebuah tipe data yang legal. Kecocokan data sangat penting dalam konversi dan operasi. Dua tipe data dikatakan cocok jika mereka memiliki tipe data yang sama.  
Ada dua cara untuk mencocokan tipe – tipe data :

1. Kecocokan berdarsarkan tipe, dimana dua variable dapat dikatakan cocok jika dan hanya jika mereka dideklarasi secara bersamaan atau dideklarasi menggunakan tipe data yang sama.
2. Kecocokan berdarsarkan struktur, dimana dua variable dikatakan cocok apabila keduanya memiliki struktur data yang identic.

Type error adalah cara implementasi yang salah dari operator kepada operand. Jika semua tipe binding static, maka hampir semua type checking bersifat static. Jika tipe binding dynamic, maka hampir semua type checking akan bersifat dynamic.

**BAB 5**

**Expression and Assignment**

*Expression* adalah hal dasar untuk menentukan komputasi dalam bahasa pemograman.Ini merupakan hal yang krusial bagi programmer untuk mengerti syntax,semantic dan ekspresi dari bahasa pemograman yang dipakai.Untuk mengerti evaluasi ekspresi,penting bagi programmer untuk mengenal aturan dari evaluasi operator dan operand. Aturan evaluasi ekspresi diatur oleh hukum asosiatif dan *precedence.*

*Expression* dibagi dalam 2 jenis yaitu *aritmethic expression* dan *relational and Boolean expression.*

**1.*Aritmethic Expression***

*Aritmethic Expression* pada bahasa pemograman kebanyakan diambil dari konvensi yang terdapat dalam matematika.Ekspresi aritmatikaterdiri dari operator,operand,*parentheses,*dan *function call* seperti contoh berikut :

$a = $a + 8;

Keterangan :

$a = operand

+ = operator

8 = operand

Operator dalam ekspresi aritmatika berupa:

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Functionality** | **Example** |
| + | Addition | $x =$ y + 6; |
| - | Subtraction | $y =$ x – 5; |
| \* | Multiply | $y = $y \* 3; |
| / | Division | $z = $x/$y; |
| % | Modulo | $A = 10 % 3; |
| ++ | Increment | $x++; |
| -- | Decrement | $z--; |
| () | Scope / Priority | $x=(2+3)\*5 |

Dalam penggunaannya ,operator dapat dimodifikasi cara penulisannya menjadi sebagai berikut :

|  |  |
| --- | --- |
| **Expression** | **Combined Operator** |
| $a =$a + $b; | $a += $b; |
| $a = $a – $b; | $a-= $b; |
| $a = $a \*$ b; | $a \*= $b; |
| $a = $a /$b; | $a /= $b; |
| $a = $a % $b; | $a %= $b; |
| $a = $a ^ $b ; | $a ^= $b; |

Operator juga bisa dibagi 3 berdasarkan operandnya yaitu :

* *Unary* ,yaitu dimana operator hanya terdiri dari 1 operand.

Contoh : pengunaan *increment* dan *decrement* (i++ / ++i dan i-- / --i)

* *Binary*, yaitu dimana operator terdiri dari 2 operand

Contoh : $a= $a + 2 ;

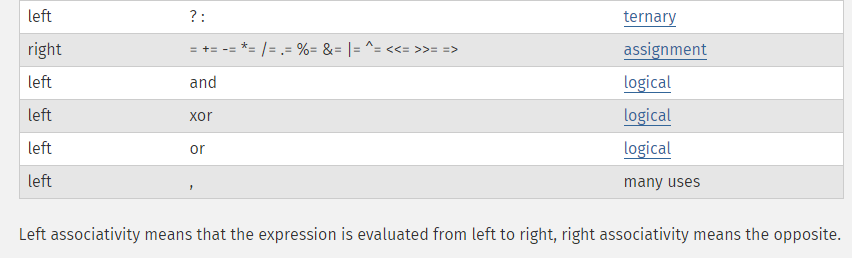
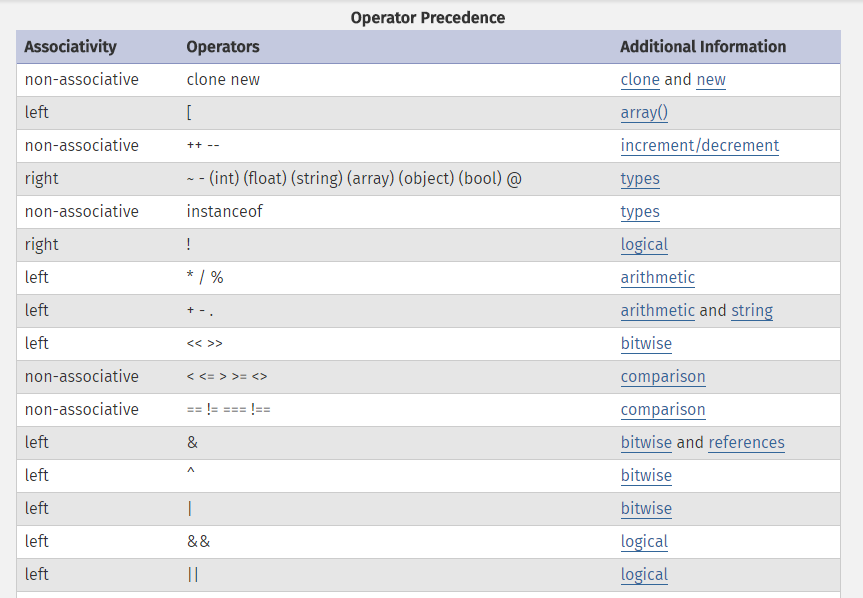
* *Ternary* yaitu dimana operator terdiri dari 3 operand

Contoh : $var == 42 ? $a : $b;

Precedence merupakan aturan prioritas yang menentukan tingkatan dari suatu operator aritmatika.Semakin tinggi tingkat suatu operator ,maka operator tersebut akan dikerjakan terlebih dahulu.Sedangkan,asosiatif adalah aturan prioritas yang memiliki precedence (urutan prioritas)yang sama .Berikut adalah urutan prioritas(precedence) dari ekspresi aritmatika pada bahasa pemograman PHP yang dibagi pada 4 bagian yaitu :

* Parentheses [ tanda kurung () ]
* Operator Unary
* \*, /
* +,-

Urutan prioritas dapat dijabarkan lebih lengkap pada tabel urutan prioritas dan aturan asosiatif operator aritmatika pada bahasa PHP :



* ***Overloaded Operator***

Penggunaan sebuah operator untuk lebih dari satu tujuan disebut **operator overloaded**. Contoh berbahaya dari overloading operator adalah ampersand (&) misal

x= &y;

dapat menyebabkan alamat dari y ditempatkan pada x. ada masalah dengan penggunaan multiple ampersand.

- Pertama penggunaan symbol yang sama untuk 2 operasi yang berbeda dapat

menyebabkan kerugian pada proses pembacaan.

-Kedua keying error sederhana dapat meninggalkan operand pertama untuk bitwise

dan operasi dapat bepergian tanpa diketahui oleh compiler, karena diinterpretasikan

sebagai address operator.

* ***Type Conversion***

Tipe conversion bertipe narrowing atau widening.

* Conversion narrowing mengubah nilai ke tipe yang tidak dapat disimpan bahkan semua pendekatan dari tipe original. Sebagai contoh mengubah tipe data double menjadi float adalah narrowing conversion, karena range dari double lebih besar dari float.
* Widening conversion mengubah nilai ke tipe yang dapat dimasukkan ke pendekatan dari semua nilai tipe original. Sebagai contoh mengubah tipe data int menjadi float disebut widening conversion

Jenis-Jenis Tipe Konversi :

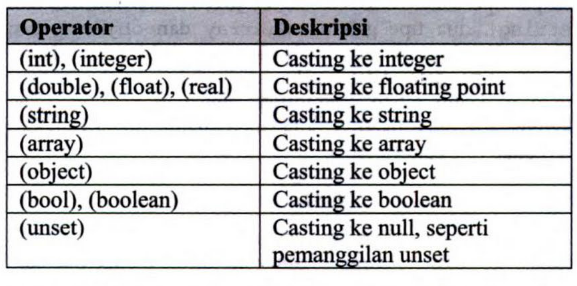
* *Implicit Type* *Conversion* atau *Type Promotion* adalah *Type conversion* yang terjadi secara otomatis tanpa sentuhan programmer.Umumnya ,tipe konversi ini menggunakan coercion dalam bentuk widening conversion .

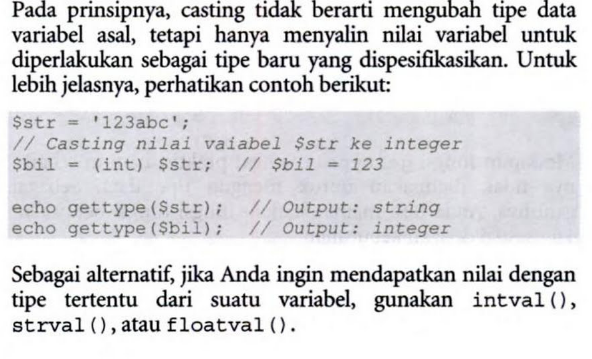
Coercion adalah bentuk decision yang menentukan apakah operator boleh memiliki operand dari tipe yang berbeda.Nantinya,Coercion akan diinisialisasikan oleh compiler. Ketika 2 operand dari sebuah operator tidak memiliki tipe yang sama dan legal dalam sebuah bahasa pemrogaman, compiler harus memilih diantara nya untuk dipaksa. Namun,Coercion memiliki kelemahan dalam deteksi error dari sebuah program..

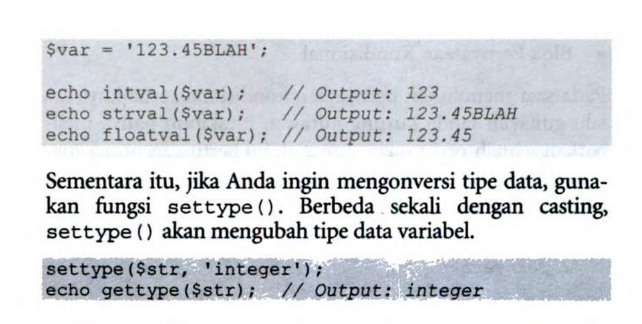
Uniknya,sebuah variabel dalam PHP dapat diisi oleh berbagai tipe data. Karena,nantinya PHP akan otomatis mengkonversinya tanpa diinstruksikan menggunakan fitur *Type Juggling* . *Type Juggling*  adalah sebuah proses untuk menentukan jenis tipe data yang “*cocok*” dengan operasi saat itu, dan PHP akan menkonversinya secara langsung (implisit).

* Explicit Type Conversion adalah Konversi tipe yang dilakukan secara manual oleh sang programmer.

Untuk kasus-kasus di mana PHP mungkin tidak biasanya mengkonversi variabel, Anda memiliki pilihan untuk secara eksplisit memaksa bahasa untuk mencoba jenis konversi dengan menggunakan apa yang disebut tipe casting.







**2.*Relational and Boolean Expression***

*Relational Expression* (Operator relasi / operator pembanding )adalah operator yang digunakan untuk melakukan suatu pembanding antara 2 ekspresi atau nilai untuk mendapatkan nilai true atau false.Bahkan relasi ekspresi juga dapat dipakai untuk beberapa pernyataan Boolean.Berikut adalah pengunaan *relational expression* pada bahasa pemograman PHP :

|  |  |  |  |
| --- | --- | --- | --- |
| *Operator* | *Name* | *Example* | *Result* |
| *==* | *Equal* | *$x == $y* | *Returns true if $x is equal to $y* |
| *===* | *Identical* | *$x === $y* | *Returns true if $x is equal to $y, and they are of the same type* |
| *!=* | *Not equal* | *$x != $y* | *Returns true if $x is not equal to $y* |
| *<>* | *Not equal* | *$x <> $y* | *Returns true if $x is not equal to $y* |
| *!==* | *Not identical* | *$x !== $y* | *Returns true if $x is not equal to $y, or they are not of the same type* |
| *>* | *Greater than* | *$x > $y* | *Returns true if $x is greater than $y* |
| *<* | *Less than* | *$x < $y* | *Returns true if $x is less than $y* |
| *>=* | *Greater than or equal to* | *$x >= $y* | *Returns true if $x is greater than or equal to $y* |
| *<=* | *Less than or equal to* | *$x <= $y* | *Returns true if $x is less than or equal to $y* |

Contoh pengunaan operator relasi/pembanding pada PHP :

<?php  
$x = 100;    
$y = "100";  
var\_dump($x == $y); // returns true because values are equal  
?>

*Boolean expressions* adalah pemakaian operand *Boolean* dan Hasil yang berupa *Boolean* (true atau false). *Boolean expressions* terdiri dari variabel *boolean*, konstanta *Boolean,* ekspresi relasional, dan operator *Boolean*. Berikut adalah pengunaan *boolean expression* pada bahasa pemograman PHP :

|  |  |  |  |
| --- | --- | --- | --- |
| Operator | Name | Example | Result |
| and | And | $x and $y | True if both $x and $y are true |
| or | Or | $x or $y | True if either $x or $y is true |
| xor | Xor | $x xor $y | True if either $x or $y is true, but not both |
| && | And | $x && $y | True if both $x and $y are true |
| || | Or | $x || $y | True if either $x or $y is true |
| ! | Not | !$x | True if $x is not true |

Contoh pengunaan operator boolean pada PHP :

<?php  
$x = 100;    
$y = 50;  
if ($x == 100 or $y == 80)

{  
    echo "Hello world!";  
}  
?>

**3.*Short Circuit Evaluation***

Sebuah short-circuit evaluation dari ekspresi adalah satu di mana hasilnya dapat ditentukan tanpa perlu mengevaluasi semua operand dan / atau operator.

Sebagai contoh, *x=(5 \* a) \* (b / 5 - 1)*

Ketika a=0, tidak perlu untuk mengevaluasi (b / 5 - 1) atau melakukan perkalian kedua. karena 0 \* x = 0 untuk setiap x.

PHP menjalankan operasi logika dengan prinsip short-circuit, yaitu jika dengan memeriksa satu perintah saja sudah didapati hasil logikanya, maka perintah lain tidak akan dijalankan dengan operator berikut

* && (and)
* || (or)

Contoh :

$hasil = $a AND $b AND $c AND $d;

Jika pada saat program dijalankan $a sudah bernilai FALSE, maka variabel $b, $c dan $d tidak akan diperiksa lagi, karena apapun nilai variabel tersebut, hasilnya akan tetap.

Hal ini akan berguna untuk kasus-kasus tertentu, seperti contoh berikut:

<?php

$result = fopen($filename) or exit();

?>

Fungsi exit() dalam kode PHP tersebut (yang akan membuat program PHP berhenti diproses) tidak akan dijalankan selama fopen($filename) bernilai true

**4.*Assignment Expression***

*Assignment statement*adalah proses untuk menyimpan sebuah value dalam suatu lokasi yang di namai sebagai variable.Syntax umum ketika menggunakan assignment statement :

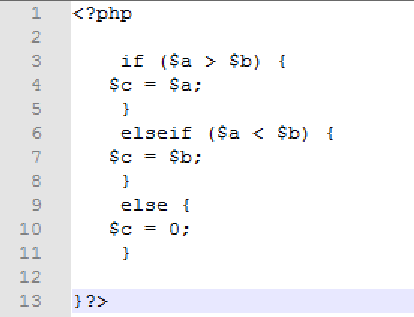
$<nama\_variable> <operator> <data>

Contoh penggunaan assignment statement :  
 $b = 1;  
Dimana ‘b’ sebagai <nama\_variable>, ‘=‘ sebagai <operator> dan ‘1’ sebagai <data>

* *Assignment Statements: Conditional Targets*

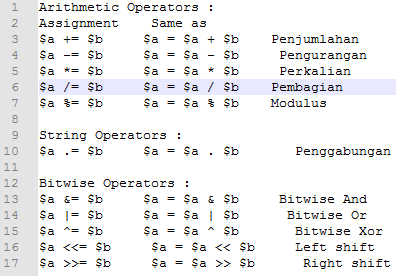
Tipe assignment statement yang memiliki syarat atau kondisi tertentu

Contoh :

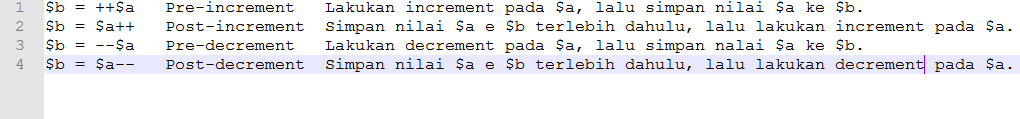


* *Assignment Statements: Compound Assignment Operators*

Suatu metode atau cara cepat untuk menentukan apa saja yang di butuhkan dalam sebuah assignment. Bentuk dari assignment ini dapat disingkat. Contoh :



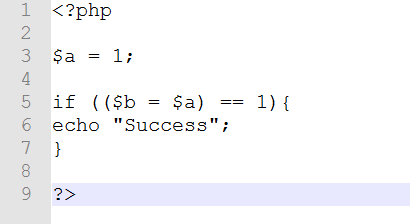
* *Assignment Statements: Unary Assignment Operators*



* *Assignment as an Expression*

Assignment statement dapat digunakan sebagai expression dan sebagai operand pada expression lainnya.

Contoh :

****

**4.*Mixed-Mode Expression***

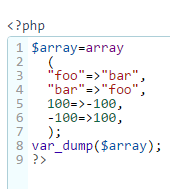
Mixed Mode Assignment adalah assignment operator yang di mixed-mode (dapat dikonversi nilainya ke dalam jenis data yang berbeda).

Berdasarkan hasilnya,mixed mode dapat dibagi menjadi 2 jenis :

* Demotion ,adalah mixed mode assignment yang mengkonversikan nilai besar menjadi nilai yang lebih kecil
* Promotion (widening),adalah mixed mode assignment yang mengkonversikan nilai kecil menjadi nilai yang besar

Mixed mode assignment hanya terdapat pada bahasa pemograman PHP :

Mixed mode diperbolehkan.Sebagai contoh dalam pemakaian array pada PHP,jenis integer dan string dapat digabung secara bersamaan karena PHP tak dapat membedakan penamaan array yang menggunakan angka ( indexted array) dan penamaan array dengan huruf (associative array ).Contoh :



Dalam semua bahasa yang memungkinkan mixed-mode assignment, coercion berlangsung hanya setelah ekspresi sisi kanan telah dievaluasi. Sebagai contoh :

int a, b;

float c;

c = a / b;

Karena c adalah float, nilai-nilai a dan b bisa dipaksa untuk diubah ke dalam bentuk float sebelum perhitungan, yang nantinya bisa menghasilkan nilai yang berbeda untuk c daripada jika pemaksaan yang tertunda (misalnya, jika yang 2 dan b adalah 3).

**BAB 6**

**Control Structure Statement**

*Control Structure* adalah kumpulan pernyataan yang mampu mengatur hasil(eksekusi). Pengaturan hasil eksekusi biasanya dapat mengubah alur dari jalannya suatu pemograman seperti mengulangi eksekusi dari beberapa pernyataan,dan melewati suatu kondisi. Pernyataan yang memiliki kemampuan pengaturan ini disebut *control statement.* Dari cara kerjanya, *control structure* dibagi kedalam 2 tipe pernyataan yaitu *selection statement* dan *iterative statement*.

**1.*Selection Statement***

*Selection statement* adalah sebuah pernyataan yang menyediakan fitur untuk memilih diantara 2 atau lebih jalan eksekusi yang terdapat didalam program.Biasanya contohnya berupa if,if-else,else-if,dan switch-case.*Selection statement* dapat dibagi kedalam 2 kategori umum yaitu : *two-way selection* dan *n-way(multiple selection).*

* 1. ***Two-Way Selection Statement***

*Two-way selection* merupakan tipe pernyataan statement yang paling dasar dan hampir sama bentuknya dengan *imperative language.* *Keyword* yang biasa digunakan adalah if dan if-else *(compound-statement)*. Bentuk dari keyword ini juga dapat dimanipulasi ke dalam bentuk yang lain seperti nested-if, dideklarasikan ke dalam 1 pernyataan atau lebih atau menggunakan ekspresi relasi atau boolean atau aritmatika bahkan campuran dari ketiga ekspresi tersebut sebagai ekspresi dari selector.Berikut contoh-contoh dari penggunaan *two-way selection statement* pada bahasa pemograman PHP :

|  |  |  |
| --- | --- | --- |
| No. | *Keyword* | Contoh Penggunaan |
| 1 | IF | <?php  $t=date("H"); //mengambil data jam dari  waktu komputer server  if ($t>"0" && $t<"5" )**\***  {  echo "Sekarang masih fajar";  }  ?>  **\*contoh manipulasi *control expression* dengan menggunakan ekspresi aritmatika ,relasi dan boolean** |
| 2 | IF-ELSE | <?php  $t=date("H");  if ($t<"5")  {  echo "Sekarang masih fajar";  }  else  {  echo "Sekarang tidak fajar lagi";  }  ?> |
| 3 | NESTED-IF | <?php  $t=date("H");  if ($t<"5")  {  echo "Sekarang masih fajar";  }  if ($t>"5")  {  echo "Sekarang tidak fajar lagi";  }  ?> |

* 1. ***Multiple Statement***

*Multiple selection statement* memperbolehkan seleksi satu dari banyak keseluruhan dari pernyataan atau tumpukan pernyataan. *Keyword* yang biasa digunakan adalah switch-case dan if-elseif. Penggunaan if-elseif juga dapat memakai “*go-to”* namun hal ini tidak disarankan karena dapat memperburuk jalur jalannya program menjadi berantakan.Berikut contoh penggunaan  *multiple statement* pada bahasa PHP :

|  |  |  |
| --- | --- | --- |
| No. | *Keyword* | Contoh Penggunaan |
| 1 | IF-ELSEIF | <?php  $t = date("H");// data diambil dari komputer  if ($t < "10") {  echo "Have a good morning!";  } elseif ($t < "20") {  echo "Have a good day!";  } else {  echo "Have a good night!";  }  ?> |
| 2 | SWITCH-CASE | <?php  $favcolor = "red";  switch ($favcolor) {  case "red":  echo "Your favorite color is red!";  break;  case "blue":  echo "Your favorite color is blue!";  break;  case "green":  echo "Your favorite color is green!";  break;  default:  echo "Your favorite color is neither red, blue, nor green!";  }  ?> |

***2.Iterative Statement***

*Iterative statement* adalah eksekusi dari sebuah atau beberapa pernyataan yang didapatkan dari *iteration* and *recursion. Iterative statement* dapat juga disebut loop.Keyword yang biasa dipakai ialah for,while dan do-while.Berdasarkan prioritas eksekusinya,loop dibagi dua yaitu pretest loop dan posttest loop.

Pretest loop adalah proses pengulangan dimana eksekusi akan dijalankan sesuai instruksi. Maka,apabila data tidak sesuai dengan instruksi maka tidak muncul hasil dari program.Contoh pretest loop adalah while dan for .Sebaliknya Posttest loop adalah proses pengulangan dimana paling tidak eksekusi dijalankan 1 kali terlebih dahulu walaupun nantinya pada proses pengecekan dimana data tidak sesuai dengan instruksi. Contoh pretest loop adalah while dan for.Berdasarkan pembacaan kondisi syarat perulangan,*iterative statement* dapat dibagi 2 yaitu *counter-controlled loops* dan *logically controlled loops.*

**2.1 *Counter-Controlled Loops***

*Counter-Controlled Loops* adalah pernyataan yang memiliki variabel loop(variabel yang menampung nilai perhitungan) yang bermakna menspesifikasi *intial* (nilai awal dari variabel loop)and *terminal* (nilai akhir dari variabel loop), dan *stepsize value* (perubahaan nilai variabel loop). Contoh dari *counter-controlled loops* adalah pengunaan for. Berbeda dengan bahasa pemograman yang lain,PHP memiliki *keyword* tambahan yaitu for-each yang digunakan pada saat perulangan yang menggunakan array.Berikut contoh penggunaan *counter-controlled loops*:

|  |  |  |
| --- | --- | --- |
| No. | *Keyword* | Contoh Penggunaan |
| 1 | FOR | <?php  for ($x = 0; $x <= 10; $x++)  {  echo "The number is: $x <br>";  }  ?> |
| 2 | FOR-EACH | <?php  $colors = array("red","green","blue", "yellow");  foreach ($colors as $value) {  echo "$value <br>";  }  ?> |

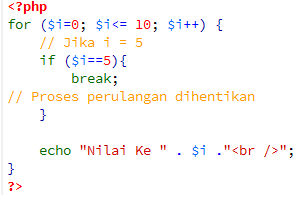
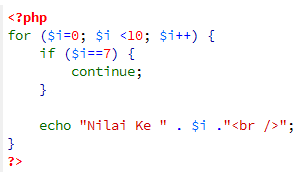
**2.2 *Logically Controlled Loops***

Dalam beberapa kasus,kumpulan statement perlu dieksekusi berulang kali namun perulangan itu tidak berdasarkan perhitungan namun ekspresi boolean.Pada kasus seperti ini,maka perulangan dapat menggunakan *logically controlled loops.*Contoh dari *logically controlled loops* adalah *while* dan *do-while*.*While* tergolong dalam kategori pretest loop sedangkan *do-while* tergolong dalam kategori posttest loop.Berikut contoh penggunaan *logically controlled loops* pada bahasa pemograman PHP :

|  |  |  |
| --- | --- | --- |
| No. | *Keyword* | Contoh Penggunaan |
| 1 | WHILE | <?php  $x = 1;  while($x <= 5)  {  echo "The number is: $x <br>";  $x++;  }  ?> |
| 2 | DO-WHILE | <?php  $x = 1;  do  {  echo "The number is: $x <br>";  $x++;  } while ($x <= 5);  ?> |

* ***User-Located Loop Control Mechanisms***

Terkadang programmer perlu menempatkan *loop control* di dalam body.Karena itu beberapa bahasa pemograman memberikan mekanisme untuk melakukan hal tersebut.Mekanisme ini memungkinkan programmer untuk dapat keluar dari loop saat berada di dalam body.*Keyword* pada *user-located loop control mechanisms* berupa break dan continu. Pada break ,setelah memenuhi kondisi untuk break maka perulangan langsung dihentikan.Sedangkan pada continu,setelah memenuhi kondisi untuk continue ,maka hanya perulangan pada kondisi tersebut dihentikan (dilewati/skip)dan dilanjutkan dengan perulangan berikutnya.Berikut contoh pengunaan *user-located loop control mechanism* pada bahasa pemograman PHP :

contoh pengunaan continue

contoh pengunaan break

* ***Iteration Based on Data Structure***

Tidak seperti counter loop dan logical loop,struktur looping ini dikontrol oleh jumlah element dari sebuah struktur data.Pada *iteration based on data structure* memiliki sebuah fungsi bernama iterator yang bertugas mencari element baru yang belum dikunjungi.Perulangan akan berhenti saat iterator gagal menemukan element baru lagi.Pada bahasa pemograman PHP , *iteration based on data structure* menggunakan iterator yang telah didefinisikan terlebih dahulu untuk akses iterative pada array unik PHP,contoh :

**reset $list**

**echo** (“first number : ” +**current**($list) + “<br/>”);

**while** ($current\_value=**next**($list))

**echo** (“Next number : ” + $current\_value + “</br>”);

- **current** menunjuk pada elemen terakhir yang diakses pada saat iterasi.

- **next** iterator berpindah pada **current** ke elemen berikutnya.

- **reset** memindahkan **current** ke elemen pertama

**BAB 7**

**Subprogram**

**1.Pengenalan Subprogram**

Subproram adalah beberapa blok program fundamental yang menjadi konsep terpenting dalam desain bahasa pemograman.Subprogram memiliki beberapa karakteristik berupa:

* Setiap subprogram memiliki satu titik masuk
* Pemanggilan nya tergantung selama eksekusi dari subprogram yang di panggil
* Control selalu mengembalikan ke pemanggil ketika memanggil menghentikan eksekusi subprogram

***2.Local Referencing Enviroments***

1. Variabel Stack Dynamic

|  |  |
| --- | --- |
| Keuntungan | Kekurangan |
| Mendukung rekursi | Memakan banyak waktu saat alokasi dan dealokasi data pada waktu inisialisasi |
| Penyimpanan dapat dibagikan kepada beberapa subprogram | Tidak dapat diakses secara langsung |
| Tidak history-sensitif |

1. Variabel Statis

|  |  |
| --- | --- |
| Keuntungan | Kekurangan |
| Memakan banyak waktu saat alokasi dan dealokasi data pada waktu inisialisasi | Mendukung rekursi |
| Tidak dapat diakses secara langsung | Penyimpanan dapat dibagikan kepada beberapa subprogram |
| Tidak history-sensitif |

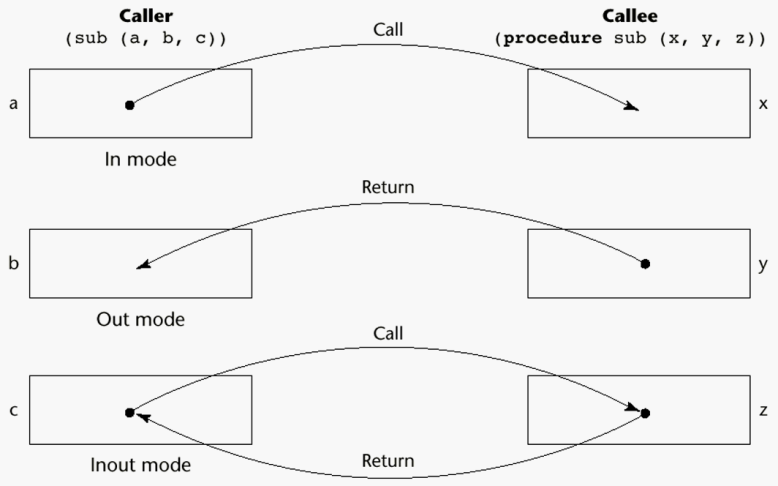
* **Nested subprogram**

Nested subprogram adalah subprogram yang diambil sesuai dengan keperluan, jadi tidak perlu menjalankan semua fungsi dari program untuk memanggil subprogram.

**3.Metode *Parameter-Passing***

Metode *Parameter-Passing* adalah cara parameter dipanggil untuk dan dari subprogram yang dipanggil.Terdapat 3 bentuk berbeda dari bentuk semantik *Parameter-Passing* yaitu :

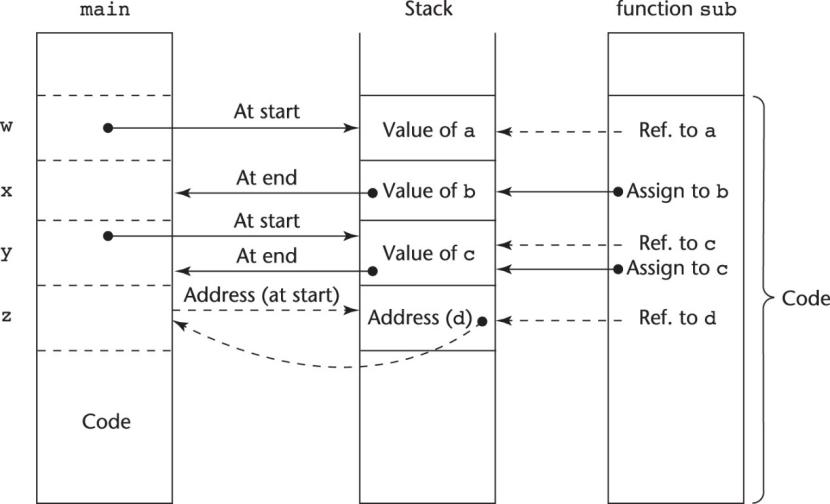
* In mode adalah bentuk *parameter passing* yang mendapat data dari parameter asal(yang muncul di bagian pemanggilan)
* Out mode adalah bentuk *parameter passing yang* mengirimkan data ke parameter asal(yang muncul di bagian pemanggilan)
* In-out mode adalah bentuk *parameter passing yang* menerima data dari parameter asal(yang muncul di bagian pemanggilan) dan mengirimkan data ke parameter asal



Gambar model *parameter-passing* in mode,out mode dan inout mode

* **Implementasi Bentuk dari *Parameter Passing***

Pada bahasa pemograman yang umum ,komunikasi parameter biasanya mengambil tempat pada saat *run-time-stack.*Berikut gambaran dari implementasi metode *parameter-passing* :



*Function header:* ***void*** *sub(****int*** *a,* ***int*** *b,* ***int*** *c,* ***int*** *d)*

*Function call in main: sub(w, x, y, z)*

*(pass w by value, x by result, y by value-result, z by reference)*

Berbagai bentuk telah dikembangkan oleh perancang program untuk mengimplementasikan 3 parameter dasar yang telah dijelaskan sebelumnya.Berikut bentuk implementasi bentuk dari *parameter passing.*

**1.*Pass-by-value***

***Pass-by-value*** adalah  mengirimkan parameter berdasarkan nilai variabel asalnya yang akan dihubungkan terhadap paramater fungsi pemanggil yang bertindak sebagi variabel local pada subprogram dan mengimplementasikan semantik *in-mode.*

Keuntungan : dari segi scalar sangat cepat saat *linkage* dan waktu saat mengaksesnya

Kerugian : Penggunaan model *copy* membutuhkan penyimpanan tambahan yang dibutuhkan oleh parameter formal (parameter yang didefinisikan) baik pada pemanggil subprogram dan di subprogram yang dipanggil.Sehingga dapat memakan memori penyimpanan yang cukup besar apabila penggunaan array dengan elemen yang banyak .

Contoh *pass-by-value* pada bahasa pemograman PHP :

<?php  
function fruit($type = "cherry")  
{  
return "Fruit you chose is $type.";  
}  
echo fruit();  
echo "<br>";  
echo fruit(null);  
echo "<br>";  
echo fruit("Strawberry");  
?>

**2.*Pass-by-result (out mode)***

*Pass-by-result* adalah model implementasi dari parameter *out-mode.*Saat parameter mengalami *parameter-pass-by-result,*tidak ada nilai yang dipindahkan ke subprogram. Parameter formal (parameter yang didefinisikan) yang terkait bertindak sebagai variabel lokal ,akan tetapi sebelum dikembalikan kepada *caller,*nilai tersebut kembali ke parameter asal(yang muncul di bagian pemanggilan) pemanggil dengan pemindahan secara fisik.

Keuntungan : dari segi scalar sangat cepat saat *linkage* dan waktu saat mengaksesnya

Kerugian : Penggunaan model *copy* membutuhkan penyimpanan tambahan yang lebih besar

**3.*Pass-by-value-result (inout mode)***

*Pass-by-value-result* adalah model implementasi dari parameter *inout-mode****.****Pass-by-result* adalah kombinasi dari *pass-by-value* dan *pass-by-result .*Terkadang dia disebut juga sebagai *pass-by-copy*.Pada *passing* jenis ini parameter formal memiliki data penyimpanan lokal .*Pass-by-value-result*  memiliki kerugian berupa kebutuhan akan memori tambahan yang cukup besar.Contoh pada bahasa pemograman C++:

int x = 0;

int main() {

foo(x);

}

void foo(int a) {

x = 3;

a++;

}

- Parameter di foo awalnya diatur ke nilai 0, dan dianggap sebagai variabel lokal.

- Variabel maka x global diatur ke 3

- Maka nilai dari bertambah dari 0 ke 1

- Ketika kontrol kembali ke pemanggil, nilai adalah disalin kembali ke angkasa pemanggil, dan nilai 1 ditugaskan untuk variabel yang dilewatkan ke foo,yang terjadi menjadi variabel global x.

- Variabel x memiliki nilai 1 pada akhir program

**3.*Pass-by-name (inout mode)***

*Pass-by-name* adalah model implementasi dari parameter *inout-mode*yang proses pemindahannya tidak dapat dihubungkan dengan bentuk implementasi tunggal.Saat parameter mengalami *pass-by-name* ,parameter asal secara tekstual disubtitusi dengan parameter formal di semua kemunculan pada subprogram.Perbedaan yang mencolok adalah parameter formal pada *pass-by-name* mengikat ke sebuah metode akses pada saat pemanggilan subprogram tetapi *actual -binding* untuk sebuah nilai atau alamat ditunda sampai parameter formal telah dimasukkan atau telah disesuaikan.Contoh pada bahasa pemograman C++:

void main() {

int i = 0;

int x[2] = {0,1};

foo(i,x[i]);

printf("%d %d\n",x[0],x[1]);

}

int foo(int a, int b) {

int i = 100;

a++;

b = 2;

}

Hasil akan berupa “0 2”

a++ akan menambah nilai dari i menjadi 1 set ke 1 pada b = 2, akan mengatur x [i] untuk ke, di mana x [i] dievaluasi dalam pemanggil ini konteks. Dengan demikian, karena saya sekarang sama dengan 1, kami menetapkan x [1] untuk 2;

PERINGATAN: 'i' dideklarasikan di foo tidak pernah diperhitungkan karena

           di call-by-nama satu mengevaluasi argumen dalam konteks

pemanggil! Jadi satu menggunakan 'i' dideklarasikan di utama.

**3.*Pass-by-reference (inout mode)***

*Pass-by-reference* adalah model implementasi dari parameter *inout-mode****.****Pass-by-reference* disebut juga sebagai *pass-by-sharing*.Bukan menduplikat data,*pass-by-reference* mengirimkan jalan akses,alamat,untuk memanggil subprogram.Contoh pada bahasa pemograman PHP :

<?php  
function foo(&$var)  
{  
    $var++;  
}  
  
$a=5;  
foo($a);  
// $a is 6 here  
?>

Keuntungan :

-proses *passing* sangat efesien baik dalam waktu dan penyimpanan(karena tidak dilakukannya penduplikatan)

Kerugian :

-Akses yang lebih lambat ke parameter formal apabila dibandingkan dengan *pass-by-value*

**-**Terbentuknya *aliases* yang tidak diiginkan

-Terbentuknya *collisions.*

* ***Parameter* yang Merupakan Subprogram**

Dalam pemograman ,sejumlah situasi yang akan mudah diselesaikan apabila nama dari subprogram dapat dijadikan parameter untuk subprogram lainnya.Namun hal ini memiliki detail yang memusingkan seperti pada masalah *type checking* parameter dari pengaktifan subprogram yang dibagikan sebagai parameter.Pada bahasa C dan C++ fungsi tak dapat dibagikan sebagai parameter namun dapat dilakukan apabila dari pointer ke fungsi.Kompilasi kedua dengan parameter yang merupakan subprogram hanya muncul pada bahasa pemograman yang memperbolehkan *nested subprogram.*Persoalan mengenai *reference environment* untuk mengeksekusi subprogram yang dibagikan pun dapat dilakukan .Berikut 3 pilihannya:

1.      Shallow binding

Lingkungan pernyataan *call*  yang mempengaruhi pengiriman subprogram

Sangat natural untuk dynamic scoped

2.      Deep binding

Lingkungan dari definisi dari subprogram yang dikirim

Paling natural untuk static scoped languange

3.      Ad hoc binding

Lingkungan dari statement yang di panggil yang mengirim subprogram

* ***Calling Subprograms Indirectly***

Memanggil subprograms secara tidak langsung merupakan suatu aksi yang terjadi apabila terdapat beberapa subprogram yang akan dipanggil dan subprogram yang tepat untuk digunakan masih belum diketahui sampai eksekusi dilakukan.Contoh penggunaan dari pemanggilan subprograms secara tidak langsung adalah Event Handling dan Graphical User Interface.

* ***Overloaded Subprograms***

## Overloaded subprograms merupakan subprogram yang memiliki nama yang sama dengan subprogram yang lainnya dalam satu lingkungan referensi yang sama dan setiap versi dari suatu overloaded subprograms memiliki protocol yang berbeda.

Contoh:

<?php  
class MethodTest  
{  
    public function \_\_call($name, $arguments)  
    {  
        // Note: value of $name is case sensitive.  
        echo "Calling object method '$name' "  
             . implode(', ', $arguments). "\n";  
    }  
  
    /\*\*  As of PHP 5.3.0  \*/  
    public static function \_\_callStatic($name, $arguments)  
    {  
        // Note: value of $name is case sensitive.  
        echo "Calling static method '$name' "  
             . implode(', ', $arguments). "\n";  
    }  
}  
  
$obj = new MethodTest;  
$obj->runTest('in object context');  
  
MethodTest::runTest('in static context');  // As of PHP 5.3.0  
?>

* **Generic Subprograms**

Suatu subprogram generic atau polymorphic mengambil parameter dari berbagai tipe dalam aktivasi - aktivasi yang berbeda. *Overloaded subprogram* adalah *ad hoc polymorphism*.Suatu subprogram yang menggunakan generic parameter yang digunakan di expression type yang mendeskripsikan tipe dari parameter tersebut disebut parameter polymorphism.

Contoh Generic Subprograms :

class MyClass implements MyInterface {

    protected $name;

    public function doThis() {

        // code that does this

    }

    public function doThat() {

        // code that does that

    }

    public function setName($name) {

        $this->name = $name;

    }

}

* ***User-Defined Overloaded Operators***

Overloading Operator adalah operator yang di gunakan sebagai pengganti fungsi inputan, output ataupun bisa fungsi yang lainya. Operator Overloading bertujuan untuk memudahkan operasi berbagai type data sesuai dengan kebutuhan user. Dengan adanya operator overloading ini kita dapat membuat sebuah fungsi yang mendefinisikan ulang operator sehingga dapat melakukan perintah khusus terhadap suatu objek.  
Sayang nya tidak semua bahasa pemrograman yang mendukung Overloading Operator. Dalam PHP sendiri Overloading Operator dibatasi dan dapat digunakan pada Array Access interface dan Operator extension.

* ***Closures***

Closures adalah subprogram dan juga merupakan referensi dari ruang lingkup nya sendiri. Dalam closures selalu ada minimal satu variable yang terikat, ini bertujuan agar ketika closures dipanggil lebih dari sekali maka variable yang terikat tetap sama dan tidak akan berubah. Referensi dari luang lingkup closures sangat penting jika subprogram bersifat global dan dapat dipanggil dimana saja dan kapan pun dalam sebuah program. Closures hanya diperlukan dalam nested scope dan jika closures bersifat global atau bisa di panggil dari mana saja.  
Bahasa pemrograman yang tidak mendukung penggunaan nested scope tidak membutuhkan closures.

* ***Coroutines***

Coroutines adalah subprogram yang memiliki multiple entry points, yang dikontrol oleh coroutines itu sendiri. Dikenal juga sebagai symmetric control karena mekanisme coroutines yang called (yang dipanggil) dan caller (yang memanggil) coroutines memiliki derajat yang sama. Pemanggilan coroutines disebut resume. Resume pertama adalah pada saat coroutine dimulai, sedangkan resume berikutnya akan dimulai setelah statement terakhir dieksekusi pada coroutine, sehingga coroutines dapat saling me-resume sampai selama – lamanya .  
Coroutines memperbolehkan quasi concurrent execution terhadap coroutines, dimana eksekusi coroutines dilaksanakan secara bersamaan tetapi tidak saling menimpah.

**BAB 8**

**Abstract Data Type**

**1.Konsep Abstraksi Data**

Konsep abstraksi data adalah proses representasi data dan program dalam bentuk sama dengan pengertiannya, dengan menyembunyikan rincian / detil dari implementsai.Abstraksi data adalah tingkatan pengguna dalam memandang bagaimana sebenarnya data diolah dalam sebuah system database.Abstraksi data bertujuan untuk memudahkan pengguna dalam penggunaan data dan dengan kontrol akses dapat menutup rincian yang tak penting dari unit diluar lampiran yang dipakai.

**2.Introduksi Tipe Data Abstraksi**

Sebuah tipe data abstrak merupakan tipe data yang dapat didefinisikan oleh user (user-defined) yang memenuhi dua kondisi berikut :

* Representasi objek dari tipe disembunyikan dari unit program yang menggunakan *object* ,maka mungkin hanya operasi yang tersedia pada definisi tipe

Keuntungan dari abstraksi data :

-*Reliability*

*-*Mengurangi jangkauan dari variabel dan kode yang harus diperhatikan oleh programmer

*-*Konflik akibat penamaan berkurang

* Deklarasi tipe dan protocol operasi pada tipe *objects*  tersedia pada unit sintatik tungal.Unit program lainnya hanya memperbolehkan untuk membentuk variabel yang tipenya terdefinisikan.

Keuntungan dari abstraksi data :

-Memberikan metode dari organisasi program

-*Aids modifiability* (semua yang terkait stuktur data selalu bersama)

-Kompilasi terpisah

Contoh penggunaan penggunaan abstraksi data pada bahasa pemograman C++:

**class** Stack {

**private**:

**int** \*stackPtr, maxLen, topPtr;

**public**:

Stack() { // a constructor

stackPtr = **new int** [100];

maxLen = 99;

topPtr = -1;

};

~Stack () {**delete** [] stackPtr;};

**void** push (**int** number) {

**if** (topSub == maxLen)

cerr << ″Error in push - stack is full\n″;

**else** stackPtr[++topSub] = number;

};

**void** pop () {…};

**int** top () {…};

**int** empty () {…};

}

***3.Parameterized Abstract Data Types***

Penggunaan *parameterized abstract data types* sangat mempermudah pekerjaan.Contohnya,kita dapat mendesain sebuah tumpukan dari tipe data abstrak yang dapat menyimpan elemen tipe scalar dibandingkan daripada menulis tumpukan abstraksi secara terpisah untuk tipe scalar yang berbeda.Contoh *parameterized abstract data types* pada C++:

· Sebuah tipe data abstrak parameter berarti bahwa tipe data generik

· Ada dan C ++ memungkinkan untuk generik atau parameter tipe data abstrak

· Generik jenis ini dianggap template.

template <class Type>  
class Stack {  
 private:  
 Type \*stackPtr;  
 const int maxLen;  
 int topPtr;  
 public:  
 Stack() { // Constructor for 100 elements  
 stackPtr = new Type[100];  
 maxLen = 99;  
 topPtr = -1;  
 }

Stack(int size) { // Constructor for a given number

stackPtr = new Type[size];

maxLen = size – 1;

topSub = -1;

}  
 ...  
}

* **Constructor**

Constructor berfungsi sebagai inisialisasi sebuah data dari suatu instance. Dalam class harus dibuat.

* **Destructor**

Destructor berfungsi sebagai penghancur sebuah data dari suatu constructor. Dalam class harus dibuat jika tidak di buat maka memori akan penuh karena data tidak pernah di hancurkan.

**4.Konstruksi *Encapsulation***

Dalam bahasa pemrograman, enkapsulasi digunakan untuk merujuk kepada salah satu dari dua konsep terkait tetapi berbeda, dan kadang-kadang untuk kombinasi daripadanya:

* Mekanisme bahasa untuk membatasi akses langsung ke beberapa komponen objek.
* konstruksi bahasa yang memfasilitasi bundling data dengan metode (atau fungsi lainnya) yang beroperasi pada data tersebut.

Enkapsulasi dapat digunakan untuk menyembunyikan data anggota dan fungsi anggota. Berdasarkan definisi ini, enkapsulasi berarti bahwa representasi internal dari sebuah objek umumnya tersembunyi dari pandangan luar definisi objek.

**5. Enkapsulasi**

Enkapsulasi adalah pengelompokan subprogram dan data yang mereka memanipulasi.Sebuah enkapsulasi menyediakan sebuah sistem abstrak dan organisasi logis untuk koleksi perhitungan terkait.Mereka sering ditempatkan di perpustakaan dan tersedia untuk digunakan kembali dalam program lain daripada yang digunakan mereka ditulis.

**6.*Nested Subprogram***

Mengatur program dengan membuat definisi subprogram yang bersarang *(nested)* dalam subprogram yang lebih besar untuk digunakan.*Nested subprogram* didukung pada bahasa pemograman Ada,Fortran 95+,Phyton,Java Script dan Ruby.

***7.Naming Encapsulation***

Penamaan enkapsulasi mendefinisikan lingkup nama yang membantu dalam menghindari ini nama konflik. Setiap perpustakaan(library) dapat membuat enkapsulasi penamaan sendiri untuk mencegah Nama-nama yang dari bertentangan dengan nama-nama yang didefinisikan dalam perpustakaan lain atau di kode klien. Setiap bagian logis dari sistem perangkat lunak dapat membuat enkapsulasi penamaan dengan tujuan yang sama.

**BAB 9**

**Object-Oriented Programming**

**1.Pengenalan *Object-Oriented Programming***

OOP (*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. Jadi setiap bagian dari suatu permasalahan adalah objek, dan objek itu sendiri merupakan gabungan dari beberapa objek yang lebih kecil lagi. Saya ambil contoh Pesawat, Pesawat adalah sebuah objek. Pesawat itu sendiri terbentuk dari beberapa objek yang lebih kecil lagi seperti mesin, roda, baling-baling, kursi, dll. Pesawat sebagai objek yang terbentuk dari objek-objek yang lebih kecil saling berhubungan, berinteraksi, berkomunikasi dan saling mengirim pesan kepada objek-objek yang lainnya. Begitu juga dengan program, sebuah objek yang besar dibentuk dari beberapa objek yang lebih kecil, objek-objek itu saling berkomunikasi, dan saling berkirim pesan kepada objek yang lain.

**2. Konsep OOP (*Object Oriented Programming*)**

Sebuah bahasa pemograman dikatakan *object oriented program* harus memiliki 3 fitur yaitu :

1. ***Abstract Data Type*(telah dibahas pada bab sebelumnya)**
2. **Pewarisan (Inheritance)**

* Kita dapat mendefinisikan suatu kelas baru dengan mewarisi sifat dari kelas lain yang sudah ada.
* Penurunan sifat ini bisa dilakukan secara bertingkat-tingkat, sehingga semakin ke bawah kelas tersebut menjadi semakin spesifik.
* Sub kelas memungkinkan kita untuk melakukan spesifikasi detail dan perilaku khusus dari kelas supernya.
* Dengan konsep pewarisan, seorang programmer dapat menggunakan kode yang telah ditulisnya pada kelas super berulang kali pada kelas-kelas turunannya tanpa harus menulis ulang semua kode-kode itu.

Tipe data abstrak biasanya disebut kelas (*classes*).Perumpanan dari kelas (*classes*) biasanya disebut objek (*object*)*.*Sedangkan,sebuah kelas yang diturukan biasa disebut *derived class* atau *subclass.*Sebuah kelas yang diturunkan dari kelas yang telah diturunkan sebelumnya disebut sebagai *superclass* atau *parent class.*Sebuah subprogram yang mendefinisikan operasi pada objek disebut *methods.*

*Message* adalah istilah dimana adanya pemanggilan ke *method*.Keseluruhan kumpulan dari *methods* sebuag objek biasa dipanggil *message protocol* atau *message interface.Message* memiliki 2 bagian yaitu nama dari *method* dan objek tujuan.Pada kasus yang sederhana,sebuah *classs* menurunkan seluruh entisitas dari *parent.*

Inheritance dapat di komplikasikan dengan akses control terhadap enkapsulasi entities

o   Class dapat menyembunyikan entities dari subclass nya

o   Class dapat menyembunyikan entities dari client nya

o   Class dapat menyembunyikan entities dari client nya tetapi memperbolehkan subclass mengakses nya

 Sebuah *class* juga dapat memodifikasi method dari class turunan nya

o   Method yang baru melakukan *override* terhadap method yang lama

o   Method pada parent di *overridden*

 Terdapat tiga cara membedakan dari *class parent* nya:

1.*Class parent* dapat mendefinisikan beberapa variabel atau *method* nya untuk memiliki private

akses, yang berarti mereka tidak akan terlihat oleh kelas turunan nya.

2.Subclass dapat menambahkan variable atau method untuk yang di wariskan dari parent.

3.Subclass dapat memodifikasi sikap dari satu atau banyak method yang diwariskan.

 Terdapat 2 jenis variable pada sebuah class :

1.     *Class variable* – dimiliki oleh kelas daripada objek ,untuk itu hanya ada satu salinan untuk kelas .

2.      *Instance variable* – setiap objek dari suatu kelas memiliki variabel *instance*nya sendiri yang disimpan pada *state* objek.Satu-satunya perbedaan yang dimiliki setiap objek dari kelas yang sama hanya terdapat pada state variabel *instancenya.*

Terdapat 2 jenis method pada sebuah class :

1.     *Class method* – menerima pesan kepada class.

2.      *Instance method* – menerima pesan kepada objek.

Jika suatu *class* baru menjadi *subclass* dari sebuah *single parent class* (kelas induk tunggal),maka proses penurunannya disebut sebgai *single inheritance.*Namun,jika sebuah kelas memiliki lebih dari satu kelas induk(*parent class*) maka proses penurunannya disebut *multiple inheritance*.Ketika jumlah *class* yang terhubung pada *single inheritance,*maka hubungan antar keduanya akan ditampilkan pada *derivation tree*.Sebaliknya,pada *multiple inheritance* ,hubungan antar keduanya dapat dilihat pada *derivation graph*.Ada 1 kekurangan dari *inheritance* yang meningkatkan kemungkinan untuk pengunaan kembali yaitu pembentukan ketergantungan antar kelas pada hierarki *inheritance.*Contoh penggunaan *inheritance* pada bahasa pemograman C++ *:*

**class** base\_class {

**private**:

**int** a;

**float** x;

**protected**:

**int** b;

**float** y;

**public**:

**int** c;

**float** z;

};

**class** subclass\_1 : **public** base\_class { … };

// In this one, b and y are protected and

// c and z are public

**class** subclass\_2 : **private** base\_class { … };

// In this one, b, y, c, and z are private,

// and no derived class has access to any

// member of base\_class

1. **Dynamic Binding**

* *Polymorphic variable* dapat di definisikan pada kelas yang dapat melakukan referensi

objek kelas dan objek dari turunan nya

* Ketika hierarki kelas mengandung kelas yang *override* method biasanya disebut dengan

*polymorphic variable*, binding ke method akan dynamic

* Mengizinkan system software lebih mudah di perpanjang ketika dikembangkan dan

maintenance

* Abstrak method adalah salah satu yang tidak dimiliki pada definisi (hanya mendefinisikan

protocol)

* Abstrak kelas memiliki setidak nya satu virtual method
* Abstrak kelas tidak dapat di instansikan
* ***Exclusivity* dari objek**

-          Semua adalah objek

o   Keuntungan : elegant dan purity

o   Kekurangan : lambat pada operasi objek yang sederhana

-          Menambah objek untuk menyempurnakan typing system

o   Kelebihan : cepatnya operasi simple objek

o   Kekurangan : hasil yang membingungkan berupa tipe sistem (2 tipe entities)

-          Memasukkan gaya imperative typing sistem untuk primitive tetapi membuat semua nya menjadi objek

o   Kelebihan : cepat operasi pada objek sederhana, system typing kecil secara relative

o   Kekurangan : masih membingungkan karena memiliki 2 tipe sistem

Jika kelas turunan adalah kelas parent. Maka objek pada kelas turunan harus bersifat sama dengan objek pada kelas parent.Subclass dapat menambah variable dan method dan override method turunan dengan cara compatible

* **Single dan multiple inheritance**

Multiple inheritance mengizinkan kelas baru untuk di warisi dari 2 atau lebih kelas

-    Kekurangan multiple inheritance :

o   Bahasa dan implementasi yang kompleks

o   Potensial tidak efisien – dynamic binding memiliki biaya lebih disbanding dengan

multiple inheritance

-     Kelebihan :

o   Terkadang sangat diperlukan dan berguna

* **Alokasi dan dealokasi objek**

-          Jika bersifat ADT, maka dapat di alokasikan dari semua tempat

o   Alokasi dari run-time stack

o   Secara explicit membuat pada heap

-          Jika heap-dynamic referensi dapat melalui pointer atau referensi variable

o   Memudahkan assignment – melakukan referensi ulang secara implicit

-          Jika objek stack dynamic, akan memiliki masalah pada subtype – object slicing

**3.Implementasi Pembentukan OOP**

Terdapat 2 bagian dari pendukung bahasa pemograman untuk OOP yang memunculkan pertanyaan menarik bagi pengguna bahasa pemograman yaitu : ruang penyimpanan untuk *instance variable* dan pesan *dynamic binding* kepada *methods.*

-Ruang penyimpanan untuk data *instance*

Pada,C++ ,kelas didefinisikan sebagai perpanjangan dari *record structutre*(kesamaan ini terletak pada pemberian saran struktur penyimpanan untuk variabel *instance* dari sebuah kelas *instance*)dari sebuah *record.*Format inilah yang disebut dengan ***class instance record*** (CIR).Struktur dari CIR bersifat statis sehingga hanya dibangun pada waktu kompilasi dan digunakan sebagai sebuah template untuk kreasi pembentukan data dari class instance.Ketika sebuah derivasi terjadi maka CIR dari subclass adalah salinan dari *parent class*,dengan masukan dari variabel baru instance yang ditambahkan pada akhir.Karena struktur CIR yang statis,akses ke semua variabel instance dapat diselesaikan sama halnya di record,menggunakan *constant offset* dari permulaan CIR .Hal ini membuat akses menjadi lebih efisien pada bidang records.

-*Dynamic Binding* pada *methods*  
Metode dalam kelas yang statis terikat kebutuhan tidak terlibat dalam CIR; metode yang akan secara dinamis terikat harus memiliki entri dalam CIR Panggilan untuk secara dinamis metode terikat dapat dihubungkan ke kode yang sesuai melalui pointer di CIR Struktur penyimpanan kadang-kadang disebut maya metode tabel (vtable) panggilan metode dapat direpresentasikan sebagai offset dari awal vtable

**BAB 10**

**Concurrency**

***1.Pengenalan Concurrency***

*Concrrency* adalah sebuah tugas(*task*) atau proses yang dimana unit program dapat dieksekusi bersamaan dengan unit program yang lain.Sebuah tugas (*task*) berbeda dengan subprogram biasa bila :

-Sebuah tugas dapat mulai secara implisit

-Ketika sebuah unit program memulai eksekusi sebuah tugas,maka tidak perlu dilakukan penundaan

-Ketika sebuah eksekusi tugas telah selesai,control tidak kembali kepada pemanggil

-Sebuah tugas biasanya dijalankan secara bersamaan

*Concurrency* pada sebuah mesin dapat terjadi pada 4 buah level :

-         Instruksi level (eksekusi dua atau lebih instruksi mesin serentak)

-         Statement level (eksekusi dua atau lebih statement bahasa tingkat tinggi serentak)

-         Unit level (eksekusi dua atau lebih subprogram unit serentak)

-         Program level (eksekusi dua atau lebih program serentak)

Secara umum,sebuah tugas dapat dibagi menjadi 2 kategori :

* *Heavyweight task* ,sebuah tugas yang mengeksekusi ruang alamatnya sendiri
* *Lightweight task* ,sebuah tugas yang berjalan pada ruang alamat yang sama.

*Lightweight task* dapat lebih efisien dibandingkan *Heavyweight task,*karena hanya butuh usaha kecil untuk mengatur eksekusinya.Sebuah tugas (*task)* dapat berhubungan dengan lainnya melalui variabel non-lokal yang dibagi. Sebuah tugas (*task)* dikatan *disjoint* apabila tidak berhubungan atau tidak mempenaruhi *task* lainnya pada program tersebut.Karena,*task* pada umumnya bekerja bersama untuk menyelesaikan masalah untuk itu mereka tidak disebut *disjoint*,mereka harusnya menggunakan beberapa bentuk komunikasi atau mengsinkronisasikan eksekusi atau membagi data mereka.

*Synchronization* adalah mekanisme yang mengkontrol *task* yang akan dieksekusi.Ada 2 jenis *synchronization* yang diperlukan saat *task* lagi berbagi data :

Terdapat 2 tipe synchronization :

- *Cooperation synchronization*

*Cooperation synchronization* adalah *synchronization* yang dibutuhkan antara *task* A dan *task* B ketika *task* A harus menunggu *task* B untuk menyelesaikan aktivitas spesifik sebelum *task* A dapat melanjutkan eksekusi.

-   Competition synchronization

*Competititon synchronization* adalah *synchronization* yang dibutuhkan antara 2 *task* ketika keduanya membutuhkan resource yang tidak dapat digunakan secara serentak.

Komunikasi antara *task* sangat penting untuk *synchronization* disediakan oleh :

- Shared nonlocal variables

- Parameters

- Message passing

*Scheduler* berfungsi memberikan sinkronisasi ke mekanisme eksekusi *task* yang terlambat.Kontrol eksekusi *task* di atur oleh program bernama *scheduler.*

Macam-macam tahap eksekusi *task*:

-         New adalah tahap dimana sebuah *tasks* dibuat tapi belum dijalankan

-         Ready adalah tahap dimana sebuah *tasks* siap untuk dijalankan tapi belum berjalan

-         Running adalah tahap dimana sebuah *tasks* yang sedang berjalan / eksekusi

-         Blocked adalah tahap dimana sebuah *tasks* yang diblok ketika sedng berjalan / interrupted oleh sesuatu

-         Dead adalah tahap dimana sebuah *tasks* yang tidak lagi aktif

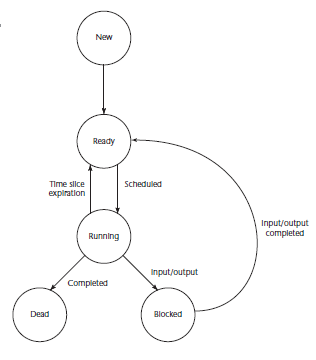


Diagram dari tahap *task*

Satu hal yang penting dalam eksekusi *task* yaitu bagaimana *task* yang dipilih telah selesai untuk berpindah ke tahapp *running* ketika *task* yang saat itu sedang berjalan telah menjadi *blocked* atau *time slice* telah berakhir? Berbagai algoritma telah digunakan untuk pilihan ini ,beberapa berdasarkan level prioritas yang spesifik .Algoritma yang melakukan tindakan pemilihan ini diimplementasikan di *scheduler.* Asosiasi antara ekseksi *concurrent* dengan kegunaan dari sumber yang dapat dibagi menghasilkan konsep *liveness.*Liveness adalah karakteristik dari program unit yang mungkin / tidak mungkin memiliki sequential code (unit akan menyelesaikan eksekusinya).Sedangkan deadlock adalah semua task pada concurrent environment kehilangan liveness nya.Deadlock merupakan ancaman yang serius terhadap *reliability* suatu program.

***2.*Masalah dalam mendesain *Concurrency***

* *Competition and cooperation synchronization* (yang paling penting)
* *Controlling task scheduling*
* *How can an application influence task scheduling*
* *How and when tasks start and end execution*
* *How and when are tasks created*

Masalah dalam mendesain *concurrency* dapat diatasi dengan 3 hal berikut yaitu:*semaphores,*

*monitors,*dan *message passing.*

* ***Semaphore***

*Semaphore* adalah data struktur mengandung *counter* dan *queue* untuk menyimpan *task descriptor* (*task descriptor* adalah data struktur yang menyimpan semua informasi relevan tentang tahap eksekusi dari sebuah *task*) / mekanisme simple yang dapat digunakan untuk memberikan sinkronisasi *task*.*Semaphore* dapat digunakan untuk implementasi *guard*  yaitu sebuah alat linguistic yang dimana kode yang dijaganya dapat dieksekusi dalam sebuah kondisi spesifik yang benar.Jadi,sebuah *guard*  hanya dapat dipakai sekali untuk sebuah *task* untuk mengakses data yang dibagi struktur data pada waktu tertentu.

Semaphore hanya memiliki dua operasi, *wait*  dan release (awalnya disebut *Passeren*  dan *Vrygerem* oleh Dijkstra). Semaphore dapat digunakan untuk menyediakan *competition* dan *cooperation synchronization.*

* ***Cooperation Synchronization* dengan *Semaphores***

Contohnya adalah *shared buffer.*

* *Buffer* diimplementasikan sebagai tipe data abstrak dengan operasi *DEPOSIT* dan *FETCH* yang menjadi acara untuk mengakses *buffer.*
* Menggunakan 2 *semaphore* untuk *cooperation* yaitu : *emptyspots* dan *fullspots.*
* Balasan dari *semaphore* biasanya digunakan unyuk menyimpan angka dari *emptyspots* dan *fullspots* pada *buffer.*
* DEPOSIT pertama kali harus memeriksa adanya ruang pada *buffer*.Bila ada,maka *counter* dari *emptyspots* akan dikurangi dan nilai akan dimasukkan. Bila tidak ada ,pemanggil akan disimpan pada barisan dari *emptyspots.*
* Setelah DEPOSIT selesai,maka *counter of fullspots* akan dinaikkan
* FETCH pertama kali wajib memeriksa fullspots untuk melihat apaka adanya nilai.Jika ada *fullspot*,*counter* dari *fullspots* akan berkurang dan nilai akan dihilangkan .Jika tidak ada nilai,maka pemanggil harus memindahkan pada antrian *fullspots.*Ketika FETCH telah selesai , maka *counter* of *emptyspots* akan naik

Operasi FETCH dan DEPOSIT pada *semphorres* dapat dipakai melalui 2 operasi *semaphores* yaitu *Wait* dan *Release*.

Berikut adalah contoh penggunaan operator wait dan release pada semaphore :

wait(aSemaphore)

**if**aSemaphore’s counter > 0 **then**

decrement aSemaphore’s counter

**else**

put the caller in aSemaphore’s queue

attempt to transfer control to some ready task

(if the task ready queue is empty, deadlock occurs)

**end if**

release(aSemaphore)

**if**aSemaphore’s queue is empty (no task is waiting) **then**

increment aSemaphore’s counter

**else**

put the calling task in the task-ready queue

transfer control to a task from aSemaphore’s queue

**end**

* ***Competition Synchronization* dengan *Semaphores***

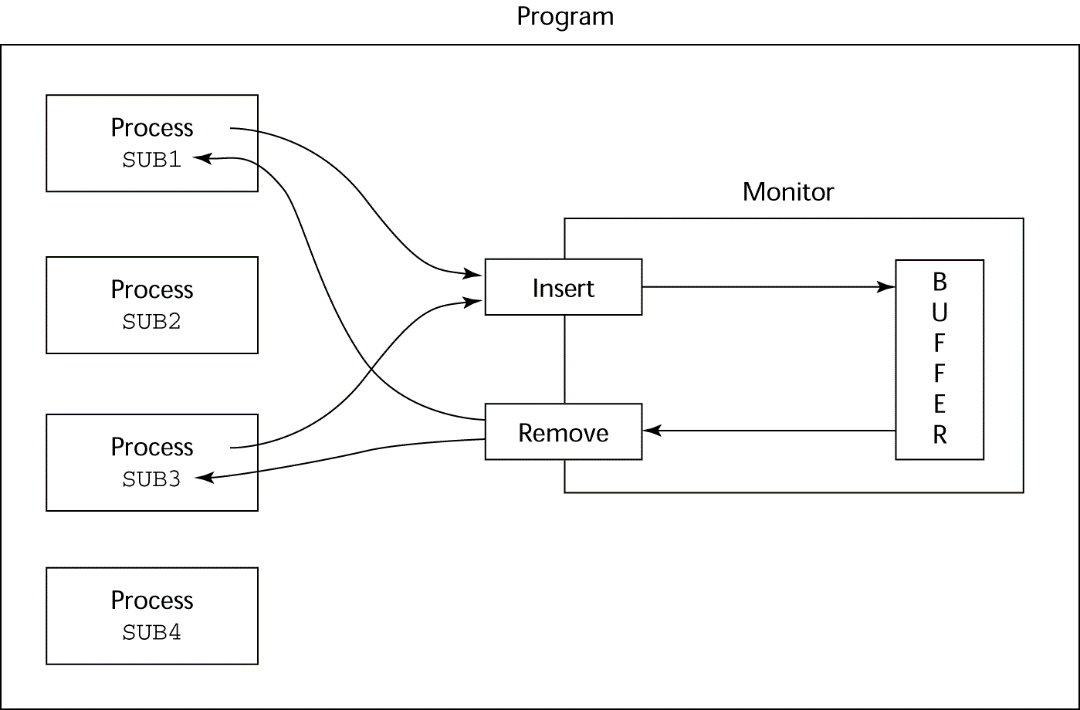
*Competition Synchronization* dengan *Semaphores* menggunakan *semaphore* ketiga yang dinamakan *access* yang digunakan untuk control akses.*Counter* dari akses hanya memiliki nilai 0 dan 1 .Inilah yang disebut sebagai *binary semaphore*.Perhatikan juga *wait* dan *release* harus atomic.

* ***Monitors***

*Monitors* bertugas mengenkapsulasi *shared data* dan operasi nya untuk membatasi akses. Monitor adalah abstract data type (ADT) untuk *shared data*.*Monitors* terdapat pada bahasa pemograman Ada,Java dan C#.*Monitors* dapat digunakan untuk menyediakan *competition* tanpa *semaphores* dengan bertanggung jawab dalam mentransfer sinkronisasi ke system *run-time.*

* ***Cooperation Synchronization* dengan *Semaphores***

*Cooperation* diantara proses masih merupakan *task* programming.Programmer harus memastikan *shared buffer* tidak mengalami *underflow* atau *overflow.*

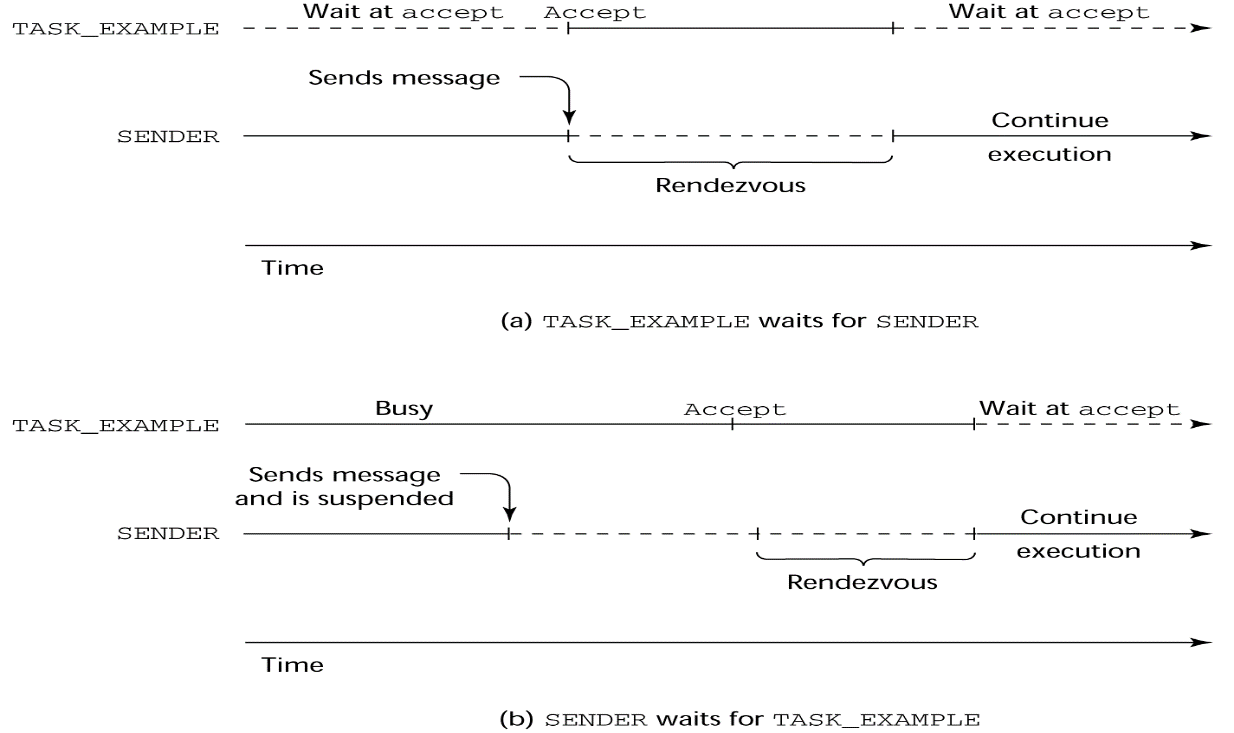
**

* ***Competition Synchronization* dengan *Semaphores***

*Shared data* bertempat tinggal di monitor dari pada di unit klien.Implementasi monitor memastikan sinkronisasi akses dengan memperbolehkannya sekali akses pada suatu waktu.Prosedur pemanggilan monitor menunggu secara implisit jika monitor sibuk pada saat pemanggilan.

* ***Message Passing***

*Message Passing* adalah model general dari *concurrency.Message passing* dapat melakukan *semaphores* dan *monitors*. *Message passing* tidak hanya untuk *competition synchronization*.Komunikasi *task* dapat diibaratkan sebagai konsultasi dokter-kebanyakan waktu dihabiskan untuk menunggu anda atau dokter,namun ketika keduanya siap ,maka akan bertemu.



Linimasa waktu pertemuan

*Server task* adalah sebuah *task*  yang menerima *clauses* bukan kode lain .Sebuah *task* tanpa menerima *clauses* disebut sebagai *actor task.*Sebuah *actor task* dapat mengirimkan pesan kepada *actor task lainnya.*Sebuah pengirim harus mengetahui *entry name* dari penerima tapi tidak sebaliknya.

***3.Multiple Entry Points***

Sebuah *task* dapat memiliki lebih dari 1 titik masuk dengan cara spesifikasi *task* memiliki klausa masuk untuk setiapnya dan *task body* dapat menerima klausa dari setiap klausa masuk dan memindahkannya pada sebuah *select clause* yaitu dalam sebuah *loop.*Contoh penggunaan *task* dengan lebih dari 1 titik masuk :

**task body** Teller **is**

**loop**

**select**

**accept** Drive\_Up(formal params) **do**

...

**end** Drive\_Up;

...

**or**

**accept** Walk\_Up(formal params) **do**

...

**end** Walk\_Up;

...

**end select**;

**end loop**;

**end** Teller;

**BAB 11**

**Exception Handling and Event Handling**

**1.Pengenalan dan Cara Kerja *Exception Handling***

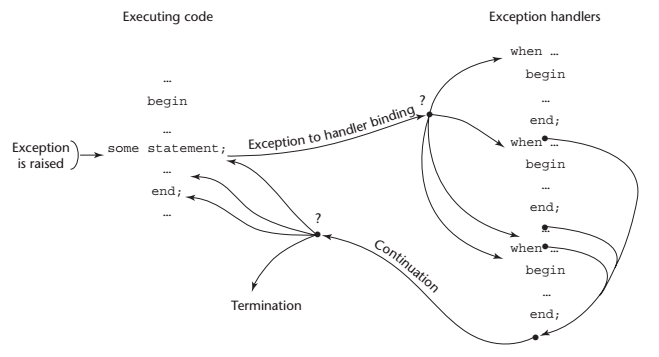
Banyak sistem perangkat keras komputer yang mampu mendeteksi run-time eror tertentu.Setelah mampu mendeteksinya,program akan menyediakan kemampuan untuk memperbaiki kesahalan dan menjalankannya.Namun,bahasa pemrograman yang pada awalnya dirancang dan dilaksanakan sedemikian rupa dimana program tidak bisa mendeteksi atau mencoba untuk menangani kesalahan(eror) tersebut. Dalam bahasa ini, terjadinya kesalahan tersebut hanya menyebabkan program yang akan dihentikan dan kontrol untuk ditransfer ke sistem operasi. Reaksi sistem operasi khas untuk kesalahan run-time adalah untuk menampilkan pesan diagnostik, yang mungkin berarti dan karena itu berguna, atau sangat samar. Setelah menampilkan pesan, program ini dihentikan.

Para perancang bahasa pemograman dari kebanyakan bahasa kontemporer telah mencantumkan mekanisme yang memungkinkan program untuk bereaksi dengan cara standar untuk kesalahan(eror) run-time tertentu, serta peristiwa yang tidak biasa program lain yang terdeteksi. Program juga diberitahu ketika peristiwa tertentu yang terdeteksi oleh perangkat keras atau sistem perangkat lunak, sehingga mereka juga dapat bereaksi terhadap peristiwa ini. Mekanisme ini secara kolektif disebut penanganan eksepsi (*exception handling*). Mungkin alasan yang paling masuk akal beberapa bahasa tidak termasuk penanganan eksepsi adalah kompleksitas itu menambah bahasa.

Pengecualian akan ada peristiwa yang tidak biasa pada suatu program, yang salah(eror) atau tidak terdeteksi oleh software atau hardware,yang membutuhkan proses khusus disebut sebagai eksepsi(*Exception*).Sedangkan,pengolahan(processing) khusus yang mungkin diperlukan saat pengecualian terdeteksi adalah disebut penanganan eksepsi**(*Exception Handling*)**. Pengolahan ini dilakukan oleh unit kode atau segmen disebut handler pengecualian(*Exception Handler*).Eksepsi(exception) dimunculkan ketika peristiwa yang terkait terjadi.

Setelah exception handler mengeksekusi, baik kontrol dapat mentransfer ke suatu tempat dalam program luar kode handler atau eksekusi program hanya dapat mengakhiri. Kami memberi istilah ini pertanyaan kontrol kelanjutan setelah eksekusi handler, atau hanya kelanjutan. Penghentian jelas pilihan yang paling sederhana, dan yang terbaik dalam banyak kondisi kesalahan(eror) pengecualian. Namun, dalam situasi lain, terutama yang berhubungan dengan yang tidak biasa tapi peristiwa yang salah, pilihan melanjutkan eksekusi adalah yang terbaik. Desain ini disebut *resumption*.Dengan kemampuan *Exception Handling,*sebuah program memiliki keuntungan sebagai berikut :

* Pendeteksian kesalahan kode sangat melelahkan untuk ditulis dan mengacaukan program
* penanganan eksepsi mendorong programmer untuk mempertimbangkan banyak kemungkinan kesalahan yang berbeda
* Pembentukan pengecualian memungkinkan penggunaan kembali tingkat tinggi kode *exception handling*

Exception handling control flow.

**2. Alternatif dari *Exception Handling***

Sebuah bahasa pemograman yang tidak memiliki kemampuan *exception handling* masih dapat mendefinisikan,mendeteksi,mendapatkan dan menangani eksepsi(*user defined,softwarer detected*) yaitu dengan cara:

-Mengerimkan parameter pembantu atau menggunakan *return value* untuk mengindikasikan *return stats* dari sebuah subprogram.

-Mengirimkan label parameter untk semua subprogram (*error return* adalah untuk label yang dikirim)

-Mengirimkan sebuah *exception handling* subprogram kepada semua subprogram.

**3. *Exception Handling* pada Bahasa Pemograman PHP**

Exception merupakan kondisi pengecualian pada program. Exception akan mengubah alur program ke mode normal jika terjadi error. Dengan kata lain, exception mengabaikan error yang terjadi dan mengerjakan proses tertentu saat exception tertangkap. Ketika exception terjadi karena dipicu oleh error, status kode program akan aman karena eksekusi program akan dialihkan ke kode program penanganan exception (exception handler). Jadi pada kondisi ini exception handler akan meresume eksekusi dari status kode yang tersimpan lalu menghentikan eksekusi script dan melanjutkannya dengan kode pada lokasi lain didalam program.

Berikut adalah kejadian yang terjadi ketika sebuah pengecualian/eksepsi dipicu:

* Pernyataan yang menyatakan kondisi kode saat ini tersimpan
* Eksekusi kode akan beralih ke fungsi handler pengecualian yang telah ditetapkan secara custom.
* Tergantung pada situasi, handler dapat melanjutkan eksekusi dari kondisi kode yang disimpan, mengakhiri eksekusi script atau melanjutkan script dari lokasi yang berbeda dalam kode

Pada bahasa pemograman PHP,kita hanya dapat menemukan *exception* dan mengubah kode tersebut secara manual. Untuk menggunakan *exception*(pencarian *exception*) pada PHP dilakukan dengan metode-metode berikut :

* *Basic use of Exceptions*

Ketika eksepsi dilemparkan, kode berikut tidak akan dieksekusi, dan PHP akan mencoba untuk menemukan pencocokan "menangkap" blok.Jika eksepsi tidak tertangkap, kesalahan fatal (fatal eror) akan dikeluarkan dengan pesan "*Uncaught exception*".contoh seperti berikut :

<?php  
//create function with an exception  
function checkNum($number) {  
  if($number>1) {  
    throw new Exception("Value must be 1 or below");  
  }  
  return true;  
}  
  
//trigger exception  
checkNum(2);  
?>

Nantinya ,akan muncul pesan error yang mennjukkan *exception* seperti berikut :

**Fatal error**: Uncaught exception 'Exception'  
with message 'Value must be 1 or below' in C:\webfolder\test.php:6  
Stack trace: #0 C:\webfolder\test.php(12):  
checkNum(28) #1 {main} thrown in **C:\webfolder\test.php** on line **6**

Untuk mencegah error seperti contoh diatas,kita perlu membentuk kode yang menangani eksepsi yaitu :

***Try, throw and catch***   
Kode Pengecualian(exception) yang tepat harus mencakup:

1. *Try* - Sebuah fungsi menggunakan pengecualian harus berada dalam "try" blok. Jika pengecualian tidak memicu, kode akan berlanjut seperti biasa. Namun jika pengecualian memicu, eksepsi "thrown"
2. *Throw* - Ini adalah cara memicu pengecualian. Setiap "throw" harus memiliki setidaknya satu "catch"
3. *Catch* - Sebuah "catch" blok mengambil pengecualian dan menciptakan sebuah objek yang berisi informasi pengecualian

Berikut contoh penggunaan try,throw,catch

<?php  
//create function with an exception  
function checkNum($number) {  
  if($number>1) {  
    throw new Exception("Value must be 1 or below");  
  }  
  return true;  
}  
  
//trigger exception in a "try" block  
try {  
  checkNum(2);  
  //If the exception is thrown, this text will not be shown  
  echo 'If you see this, the number is 1 or below';  
}  
  
//catch exception  
catch(Exception $e) {  
  echo 'Message: ' .$e->getMessage();  
}  
?>

Hasil kode di atas akan menghasilkan pesan error seperti berikut :

Message: Value must be 1 or below

Penjelasan dari script kode di atas :

Pada kode diatas terlihat proses memunculkan exception (try), melemparkan (throw) dan menangkap exception (catch) dan menampilkan informasi exception. Program dimulai dengan membuat fungsi checkNum yang melemparkan exception jika angka yang dimasukkan lebih besar dari 1.  
  
Selanjutnya exception dipicu dengan perintah try yang menjalankan fungsi checkNum(2). Exception yang dihasilkan oleh perintah try selanjutnya ditangkap oleh perintah catch yang kemudian menampilkan Pesan Error yang diset oleh perintah throw pada fungsi checkNum.

* *Creating a custom exception handler*

Untuk membuat pengecualian *custom handler*(*custom exception handler*) maka harus dibuat kelas khusus dengan fungsi yang dapat dipanggil ketika pengecualian(exception) terjadi di PHP. Kelas harus menjadi perpanjangan dari kelas pengecualian.

Kelas pengecualian kustom (*custom exception handler*) mewarisi(inherit) sifat dari kelas pengecualian PHP dan anda dapat menambahkan fungsi kustom untuk itu,seperti contoh berikut :

<?php  
class customException extends Exception {  
  public function errorMessage() {  
    //error message  
    $errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFile()  
    .': <b>'.$this->getMessage().'</b> is not a valid E-Mail address';  
    return $errorMsg;  
  }  
}  
  
$email = "someone@example...com";  
  
try {  
  //check if  
  if(filter\_var($email, FILTER\_VALIDATE\_EMAIL) === FALSE) {  
    //throw exception if email is not valid  
    throw new customException($email);  
  }  
}  
  
catch (customException $e) {  
  //display custom message  
  echo $e->errorMessage();  
}  
?>

Penjelasan script kode diatas:

Kelas baru adalah salinan dari kelas pengecualian sebelmnya dengan penambahan fungsi errorMessage ( ). Karena itu adalah salinan dari kelas sebelumnya, dan mewarisi properti dan metode dari kelas sebelumnya, kita dapat menggunakan metode kelas pengecualian seperti getline ( ) dan getFile ( ) dan getMessage ( ) .

Kode di atas melempar pengecualian dan menangkap dengan kelas pengecualian kustom :

* The *customException ( ) class* dibuat sebagai perpanjangan dari kelas pengecualian sebelumnya . Dengan cara ini mewarisi semua metode dan properti dari kelas pengecualian sebelumnya
* The fungsi *errorMessage ( )* dibuat. Fungsi ini mengembalikan pesan kesalahan jika alamat e -mail tidak valid
* Variabel email $ diatur ke string yang bukan alamat e – mail yang valid
* The ” try ” blok dijalankan dan eksepsi dilemparkan karena alamat e -mail tidak valid
* The ” catch” blok menangkap pengecualian dan menampilkan pesan kesalahan
* *Multiple exceptions*

Hal ini dimungkinkan script untuk menggunakan beberapa pengecualian untuk memeriksa beberapa kondisi. Hal ini dimungkinkan untuk menggunakan beberapa blok if..else, switch, atau nest beberapa pengecualian(multiple exception).   
Pengecualian ini dapat menggunakan kelas eksepsi yang berbeda dan kembali pesan kesalahan yang berbeda:

<?php  
class customException extends Exception {  
  public function errorMessage() {  
    //error message  
    $errorMsg = 'Error on line '.$this->getLine().' in '.$this->getFile()  
    .': <b>'.$this->getMessage().'</b> is not a valid E-Mail address';  
    return $errorMsg;  
  }  
}  
  
$email = "someone@example.com";  
  
try {  
  //check if  
  if(filter\_var($email, FILTER\_VALIDATE\_EMAIL) === FALSE) {  
    //throw exception if email is not valid  
    throw new customException($email);  
  }  
  //check for "example" in mail address  
  if(strpos($email, "example") !== FALSE) {  
    throw new Exception("$email is an example e-mail");  
  }  
}  
  
catch (customException $e) {  
  echo $e->errorMessage();  
}  
  
catch(Exception $e) {  
  echo $e->getMessage();  
}  
?>

Penjelasan script kode diatas:

1. The *customException () class* dibuat sebagai perpanjangan dari kelas pengecualian sebelumnya. Dengan cara ini mewarisi semua metode dan properti dari kelas pengecualian sebelumnya
2. The fungsi errorMessage ()dibuat. Fungsi ini mengembalikan pesan kesalahan jika alamat e-mail tidak valid
3. Variabel $ email diatur ke string yang merupakan alamat e-mail yang valid, tetapi berisi string “contoh”
4. The “try” blok dijalankan dan pengecualian tidak dilempar pada kondisi pertama
5. Kondisi kedua memicu pengecualian karena e-mail berisi string “example”
6. The “catch” blok menangkap pengecualian dan menampilkan pesan kesalahan yang benar

Jika pengecualian dilemparkan adalah dari customException kelas dan tidak ada customException menangkap, hanya menangkap dasar pengecualian, pengecualian akan ditangani di sana.

* *Re-throwing an exception*

Kadang-kadang, ketika eksepsi dilemparkan, kita mungkin ingin menanganinya berbeda dari cara standar. Hal ini dimungkinkan untuk melemparkan pengecualian untuk kedua kalinya dalam "catch" blok. Sebuah script harus menyembunyikan kesalahan sistem dari pengguna.Kesalahan sistem mungkin penting untuk sang pembuat kode, tetapi tidak menarik bagi pengguna.Untuk membuat lebih mudah bagi pengguna kita, dapat kembali melempar pengecualian dengan pesan user friendly:

<?php  
class customException extends Exception {  
  public function errorMessage() {  
    //error message  
    $errorMsg = $this->getMessage().' is not a valid E-Mail address.';  
    return $errorMsg;  
  }  
}  
  
$email = "someone@example.com";  
  
try {  
  try {  
    //check for "example" in mail address  
    if(strpos($email, "example") !== FALSE) {  
      //throw exception if email is not valid  
      throw new Exception($email);  
    }  
  }  
  catch(Exception $e) {  
    //re-throw exception  
    throw new customException($email);  
  }  
}  
  
catch (customException $e) {  
  //display custom message  
  echo $e->errorMessage();  
}  
?>

Penjelasan script kode diatas:

Kode di atas tes jika email-address berisi string “contoh” di dalamnya, jika tidak, pengecualian tersebut kembali dilemparkan:

1. The *customException () class* dibuat sebagai perpanjangan dari kelas pengecualian sebelumnya. Dengan cara ini mewarisi semua metode dan properti dari kelas pengecualian sebelumnya.
2. The fungsi errorMessage ()dibuat. Fungsi ini mengembalikan pesan kesalahan jika alamat e-mail tidak valid
3. Variabel $ email diatur ke string yang merupakan alamat e-mail yang valid, tetapi berisi string “example”
4. The “try” blok berisi “try” blok lain untuk membuatnya mungkin untuk kembali membuang-pengecualian
5. Pengecualian dipicu karena e-mail berisi string “example”
6. The “catch” blok menangkap pengecualian dan re-melempar “customException”
7. The “customException” tertangkap dan menampilkan pesan kesalahan

Jika pengecualian tidak tertangkap dalam arus “try” blok, ia akan mencari blok catch pada “higher levels”.

* *Setting a top level exception handler*

The fungsi set\_exception\_handler ()menetapkan fungsi yang ditetapkan pengguna untuk menangani semua pengecualian tertangkap.Dengan contoh seperti berikut :

<?php  
function myException($exception)  
{  
echo “<b>Exception:</b> ” , $exception->getMessage();  
}

set\_exception\_handler(‘myException’);

throw new Exception(‘Uncaught Exception occurred’);  
?>

Output dari kode di atas harus sesuatu seperti ini:

pengecualian: *Uncaught Exception occurred*

Dalam kode di atas tidak ada blok “catch”. Sebaliknya, handler pengecualian tingkat atas dipicu. Fungsi ini harus digunakan untuk menangkap pengecualian tidak tertangkap.

**Aturan untuk pengecualian**

* Kode dapat dikelilingi dalam blok try, untuk membantu menangkap pengecualian potensial
* Masing-masing mencoba blok atau “membuang” harus memiliki minimal satu sesuai blok catch
* Beberapa blok catch dapat digunakan untuk menangkap kelas yang berbeda dari pengecualian
* Pengecualian dapat dibuang (atau re-dilemparkan) dalam blok catch dalam blok try

Sebuah aturan sederhana: Jika Anda melemparkan sesuatu, Anda harus menangkapnya.

**4.Pengenalan *Event Handling***

Sebuah acara(event) adalah pemberitahuan bahwa sesuatu yang spesifik telah terjadi, seperti klik mouse pada tombol grafis. Sebenarnya, acara adalah obyek yang secara implisit diciptakan oleh sistem run-time dalam menanggapi tindakan pengguna, setidaknya dalam konteks di mana penanganan event(event handling).

Event handler adalah segmen kode yang dieksekusi dalam menanggapi penampilan sebuah acara(event). Event handler memungkinkan program untuk menjadi responsif terhadap tindakan penggun

**BAB 12**

**Functional Programming Language**

**1.Pengenalan**

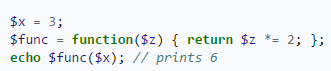
Bahasa pemrograman yang imperative adalah bahasa yang berbasiskan *von Neumann architecture*, menggunakan memori yang sama untuk data dan untuk memberikan instruksi kepada program, dimana mayoritas komputer moderen menerapkan bahasa ini. Sedangkan bahasa pemrograman yang functional adalah bahasa yang berbasiskan fungsi matematika, yang merupakan gambaran struktur anggota dari suatu domain set menuju ke range set.

**2.Mathematical Function**

Sebuah fungsi matematika adalah pemetaan dari keanggotaan sebuah himpunan yang disebut himpunan domain ,jika untuk himpunan lain itu disebut himpunan *range.*

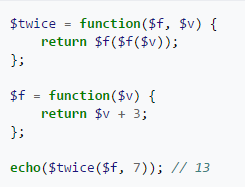
* Ekspresi Lambda

Ekspresi lambda mendeskripsikan fungsi yang tidak memiliki nama dengan menspesifikasikan parameter dari satu fungsi dengan menaruh parameter setelah ekspresi lambda.



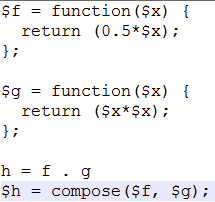
* *Functional Form*

A higher-order function atau functional form adalah fungsi yang menggunakan fungsi – fungsi lain yang lebih simpel sebagai parameter atau sebagai hasil atau keduanya. Biasa juga disebut *operator* dan *functional.*



* *Function Composition*

Komposisi fungsi adalah mekanisme untuk menggabungkan fungsi – fungsi simpel untuk membentuk fungsi yang lebih kompleks. Dengan menggunakan dua fungsi sebagai parameter dan menghasilkan fungsi yang nilainya berasal dari parameter pertama yang diimplementasikan ke parameter yang kedua.



* Apply-to-all

Suatu *functional form* yang menggunakan sebuah fungsi sebagai parameter dan menghasilkan daftar nilai yang didapat dari penerapan fungsi tersebut kepada setiap element di dalam parameter.

**3.Bahasa Pemograman *Functional* Pertama : LISP**

* *LISP Data Types and Structures*

LISP memiliki tipe data yang terdiri dari atom dan daftar, yang disimpan sebagai satu daftar yang saling berhubungan. Bentuk dari LISP biasa berupa kumpulan sublist dan atom. LISP sebenarnya merupakan bahasa yang tidak bisa diketik.

* *LISP Interpretation*

Dalam notasi lambda diamana sebuah daftar (A, B, C), jika diinterpretasikan sebagai data maka hanya akan menjadi sebuah daftar sederhana yang memiliki tiga atom yaitu A, B dan C. Sedangkan jika diinterpretasikan sebagai aplikasi dari sebuah fungsi maka A merupakan sebuah fungsi yang diimplementasikan ke dua parameter B dan C.

* Primitive Functions & Lambda Expressions
* Primitive Arithmetic Functions : + , - , \*, /, ABS, SQRT, REMAINDER, MIN, MAX  
  Contoh : (+ 5 2) menghasilkan 7
* Lambda Expressions, bentuknya berdarsarkan dari notasi lambda.  
  Contoh : (LAMBDA (x) (\* x x), dimana x adalah variabel yang terikat.
* Lambda expressions bisa diaplikasikan kepada parameter.  
  Contoh : ((LAMBDA (x) (\* x x)) 7)
* Ekspresi lambda bisa memiliki berapapun parameter.  
  Contoh : (LAMBDA (a b x) (+ (\* a x x) (\* b x)))
* Special Form Function: DEFINE

DEFINE memiliki dua bentuk :

1. Untuk mengikat simbol kedalam suatu ekspresi  
   Contoh : (DEFINE 3.141593)
2. Untuk mengikat nama kedalam suatu ekspresi lambda  
   Contoh : (DEFINE (square x) (\* x x))

* Output Functions

Output functions biasanya tidak dibutuhkan karena *intrerpreter* selalu menampilkan hasil dari sebuah fungsi yang telah dievaluasi pada bagian teratas.

Scheme memiliki PRINTF yang memiliki fungsi yang sama dengan echo pada PHP.

* Numeric Predicate Functions
* #T (or #t) adalah benar dan #F (or #f) adalah salah (terkadang () juga digunakan sebagai salah).
* =, <>, >, <, >=, <=
* EVEN?, ODD?, ZERO?, NEGATIVE?
* Fungsi NOT(!) membalikan logika dari sebuah ekspresi Boolean

Control Flow adalah susunan dari pernyataan, instruksi dan pemanggilan fungsi dari sebuah program yang imperative yang akan dijalankan.

* List Functions

QUOTE – mengambil satu parameter dan mengembalikan parameter tersebut tanpa dievaluasi. QUOTE dibutuhkan karena *interpreter* dari Scheme yang bernama EVAL selalu mengevaluasi parameter sampai ke fungsi dari aplikasi sebelum mengaplikasikan sebuah fungsi. QUOTE digunakan untuk menghindari evaluasi dari parameter ketika sedang tidak cocok.  
(QUOTE (A B)) sama dengan ‘(A B).LIST adalah sebuah fungsi untuk membuat sebuah list dari berapapun jumlah parameter.(LIST ‘apple ‘orange ‘grape) **returns** ( apple orang grape)

* Predicate Function

EQ? menggunakan dua ekspresi sebagai parameter (biasanya dua atom), dan akan menghasilkan #T (Benar) jika kedua parameter memiliki penunjuk nilai yang sama, selain dari itu maka akan memberikan hasil #F (Salah).  
Jika (EQ? 'A 'A) maka #T.  
Jika (EQ? 'A 'B) maka #F.

EQV? sama seperti EQ? tetapi EQV? bisa digunakan untuk simbol dan atom. EQV? merupakan perbandingan nilai dan bukan penunjuk nilai.  
 Jika (EQV? 3 3) maka #T  
 Jika (EQV? 'A 3) maka #F

LIST? menggunakan satu parameter dan menghasilkan #T (Benar) jika parameter tersebut adalah sebuah daftar, selain dari itu maka #F (Salah).  
Jika (LIST? '(apple orang grape)) maka #T

NULL? menggunakan satu parameter dan menghasilkan #T (Benar) jike parameter tersebut adalah daftar yang kosong, selain dari itu maka #F (Salah).  
Jika (NULL? '(())) maka #F

Example Scheme Function

* **member** menggunakan atom dan daftar yang simpel kemudian menghasilkan #T (Benar) jika atom tersebut ada pada daftar, selain dari itu maka #F (Salah).
* **equalsimp** menggunakan dua daftar yang simpel sebagai parameter dan menghasilkan #T (Benar) jika keduanya sama, selain dari itu maka #F (Salah).
* **equal**  menggunakan dua daftar umum sebagai parameter dan menghasilkan #T jika kedua daftar tersebut sama, selain dari itu maka #F (Salah).
* **append** menggunakan dua daftar sebagai parameter dan menghasilkan parameter pertama dengan anggota dari parameter kedua yang ditambahkan pada bagia akhir.
* Tail Recursion in Scheme

Sebuah fungsi adalah ekor dari rekursif apabila pemanggilan rekursifnya adalah operasi terakhir dalam fungsi tersebut. Semua fungsi yang merupakan ekor rekursif dengan otomatis dapat di ubah oleh compiler menjadi iteration untuk membuatnya lebih cepat dan merupakan salah satu keharusan pada Scheme.

* Functions That Build Code

Sangat mungkin di Scheme untuk menetapkan fungsi yang membuat kode Scheme dan meminta interpretasinya karena *interpreter* merupakan fungsi yang tersedia bagi user yang merupakan EVAL.

* Adding a List of Numbers

Parameter merupakan daftar angka yang akan ditambahkan, dimana **adder** memasukan sebuah operator + dan mengevaluasi hasil dari daftar tersebut.

Membandingkan bahasa Functional and Imperative :

* + Bahasa Imperative:

-Eksekusi yang effisien

-Semantic yang komplex

-Syntax yang komplex

-Concurrency adalah design dari programmer

* Functional Languages:

-Semantic yang simpel

-Syntax yang simpel

-Eksekusi yang kurang effisien

-Program secara otomatis bisa dibuat bersama

**BAB 13**

**Logic Programming Language**

**1.Pengenalan *Logic Programming Language***

Logic programming merupakan suatu program yang menggunakan bahasa logika, dan kemudian bahasa logika tersebut di ekspresikan dengan bentuk symbol – symbol logika. Logic programming menggunakan proses inferensi logika untuk membuat hasil. Logic programming hanya bersifat deklaratif artinya hanya spesifikasi dari hasil yang dinyatakan bukan proses detail untuk memproduksi mereka.

**2.Pengenalan Dasar Kalkulus Predikat (*Calculus Predicate*)**

* **Proposisi**

Proposisi merupakan statement logika yang dapat bersifat benar atau tidak benar. Poposisi terdiri dari berbagai macam objek dan hubungan dari suatu objek dengan objek lainnya.

* **Simbol Logika**

Merupakan suatu logika yang dapat digunakan sebagai cara yang dasar untuk logika yang lebih formal. Simbol logika berfungsi untuk :

* Mengekspresikan proposisi.
* Mengekspresikan hubungan diantara dua proposisi.
* Mendeskripsikan bagaimana suatu proposisi yang baru dapat disimpulkan dari proposisi yang lain.

Bentuk tertentu dari suatu symbol logika yang digunakan untuk logic programming disebut *predicate calculus*

* **Representasi Objek**

Objek dalam suatu proposisi direpresentasikan dalam dua bentuk sederhana yaitu :

* Konstan : suatu simbol yang merepresentasikan objek
* Variabel : suatu simbol yang dapat merepresentasikan objek yang berbeda dalam waktu yang berbeda juga.
* **Bentuk Compound**

Atomic proposisi ada proposisi yang mengandung compound term .Compound term merupakan elemen dari suatu relasi matematika yang cara penulisannya mirip dengan penulisan dalam fungsi matematika (dapat ditulis dalam bentuk tabel dan fungsi matematika adalah pemetaan). Bentuk compound tersusun dari dua bagian, yaitu :

* Functor : Simbol fungsi yang menamai suatu hubungan
* Tuple : Daftar parameter yang sudah tersusun

Contoh :

student(jon)                             // jon is student

            like(seth, OSX)                       // seth like OSX

            like(nick, windows)                //nick like windows

            like(jim, linux)                         //jim like linux

* **Bentuk Proposisi**

Proposisi dapat dinyatakan dalam dua bentuk, yaitu :

* Fact : Proposisi yang telah dianggap benar
* Query : Kebenaran dari suatu proposisi masih belum ditentukan

Proposisi compound harus memiliki dua atau lebih proposisi atomic dan proposisinya dihubungkan oleh operator.

**Operator Logika**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Symbol** | **Example** | **Meaning** |
| Negation | ¬ | ¬ a | not a |
| Conjunction | ∩ | a ∩ b | a and b |
| Disjunction | ∪ | a ∪ b | a or b |
| Equivalence | ≡ | a ≡ b | a is equivalent to b |
| Implication | ⊃  ⊂ | a ⊃ b  a ⊂ b | a implies b  b implies a |

**Quantifier**

|  |  |  |
| --- | --- | --- |
| **Name** | **Example** | **Meaning** |
| Universal | ∀X.P | For all X, P is true |
| Existential | ∃X.P | There exists a value of X such that P is true |

**3.Logical Programming**

PROLOG adalah kependekan dari ***PROgramming in LOGic***, yang berarti pemrograman

logika. Pemrograman Prolog menggunakan bahasa deklaratif, dimana pemrogram

memberi faktadan aturanuntuk selanjutnya diselesaikan oleh Prolog secara deduktif

sehingga menghasilkan suatu kesimpulan. Hal ini berbeda dengan bahasa prosedural

seperti Pascal, Fortran, C, atau yang sejenis, dimana pemrogram memberi perintah atau

penugasan untuk memecahkan persoalan langkah demi langkah, sehingga sering disebut

sebagai *programming with assignment*. Disamping itu, berbeda dengan pemrograman

fungsional, pemrograman logika ini menggunakan relasi, bukan fungsi sehingga sangat

sesuai untuk implementasi sistem pakar.Selain itu, Programmingnya tidak procedural artinya program tidak menyatakan hasilnya akan dikomputasi, tapi lebih ke menyatakan bentuk dari hasilnya.

* **Menyusun List**

Mendeskripsikan karakteristik dari suatu list yang telah di susun, bukan proses aransemen ulang list.

Contoh :

sort(old\_list, new\_list) ⊂ permute (old\_list, new\_list) ∩ sorted (new\_list)

sorted (list) ⊂ ∀j such that 1≤ j < n, list(j) ≤ list (j+1)

**4.Dasar Prolog**

* Terms

Terms merupakan suatu konstan, variabel, atau struktur. Konstan merupakan suatu atom atau integer. Atom merupakan nilai simbol dari prolog. Atom bisa terdiri dari suatu string kata, digit, dan kata yang bersifat lowercase atau dari string suatu karakter ASCII yang dapat diprint. Terms dapat dibedakan menjadi dua jenis yaitu Variabel dan Struktur.

* Statement

Terdiri dari :

* + Fact Statements

Digunakan untuk hipotesis, Klaus nya Headless Horn

Contoh :

female(shelley).

male(bill).

father(bill, jake).

* + Rule Statements

Digunakan untuk hipotesis, Kalusnya Headless Horn.

* + - Sisi kanan : antecedent (bagian if)

Bisa single term atau conjunction

* + - Sisi kiri : consequent (bagian then)

Harus single term

* + - Conjunction

Term – term yang dipisahkan oleh operasi logika AND

Contoh :

ancestor(mary,shelley):- mother(mary,shelley).

Dapat menggunakan variabel untuk menggeneralisasi suatu maksud :

parent(X,Y):- mother(X,Y).

parent(X,Y):- father(X,Y).

grandparent(X,Z):- parent(X,Y), parent(Y,Z).

* + Goal Statements

Digunakan untuk membuktikan suatu teorema, tapi teorema nya dalam bentuk proposisi yang kita ingin sistemnya buktikan atau tidak. Format nya sama dengan Klaus Headless Horn, yaitu

man(fred)

Proposisi conjunctive dan proposisi dengan variabel juga dapat digunakan

father(X, mike)

**5.Menginferensikan Proses Suatu Prolog**

Jika suatu goals merupakan proposisi compound, maka setiap facts merupakan subgoal. Untuk membuktikan suatu goal itu benar, kita harus menemukan hubungan aturan dan/atau fakta inferensi.Contoh, untuk goal Q

P2 :- P1

P3 :- P2

…

Q :- Pn

Proses untuk membuktikan suatu subgoal disebut *matching, satisfying,* atau *resolution*. Matching merupakan proses membuktikan suatu proposisi. Satisfying merupakan proses membuktikan subgoal. Bottom-up resolution disebut forward chaining, Top-down resolution disebut backward chaining. Pengimplementasian prolog menggunakan backward chaining.

* Strategi Subgoal

Ketika suatu goal memiliki lebih dari satu subgoal, maka kita dapat menggunakan

* + Pencarian depth-first : Mencari bukti lengkap subgoal pertama sebelum mencari bukti lengkap untuk subgoal berikutnya;atau
  + Pencarian breadth-first : Mengerjakan semua subgoal secara parallel
* Bactracking

Ketika suatu goal memiliki banyak subgoal, dan ketika gagal membuktikan suatu subgoal, maka menggunakan subgoal sebelumnya untuk mencari solusi alternatifnya merupakan pengertian dari backtracking. Bactracking bisa memakan banyak waktu dan tempat karena harus mencari semua bukti yang memungkinkan untuk setiap subgoal.

* Aritmatika Sederhana

Prolog mensupport variabel integer dan juga aritmatika integer. Operator **is** : menggunakan ekspresi aritmatika sebagai operand kanan dan variabel sebagai operand kiri. Contoh :

A **is** B / 17 + C

Penggunaan operator ini tidak sama dengan statement assignment.

Contoh :

speed(ford,100).

speed(chevy,105).

speed(dodge,95).

speed(volvo,80).

time(ford,20).

time(chevy,21).

time(dodge,24).

time(volvo,24).

distance(X,Y) :- speed(X,Speed),

time(X,Time),

Y **is** Speed \* Time.

* Trace

Merupakan struktur built-in yang menampilkan instantiasi dari setiap langkah. Tipe pelaksanaan model tracing :

* Call
* Exit
* Redo
* Fail

Contoh :

likes(jake,chocolate).

likes(jake, apricots).

likes(darcie, licorice).

likes(darcie, apricots).

trace.

likes(jake, X), likes(darcie, X).

1. 1 Call: likes(jake, \_0)?

(1) 1 Exit: likes(jake, chocolate)

(2) 1 Call: likes(darcie, chocolate)?

(2) 1 Fail: likes(darcie, chocolate)

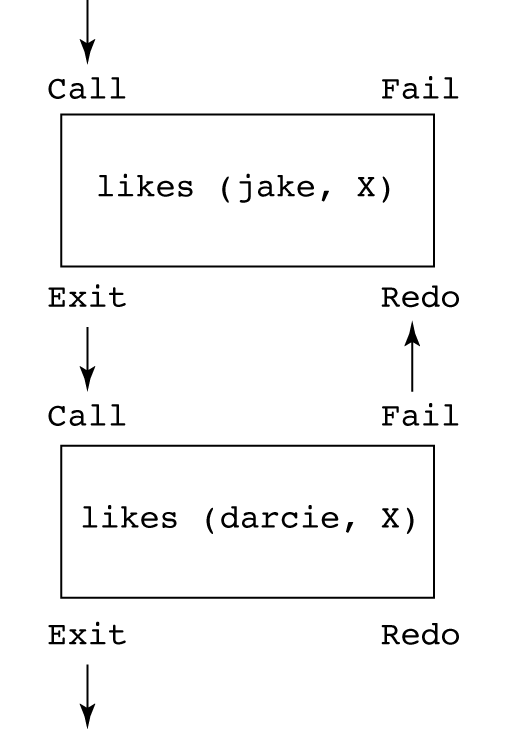
1. 1 Redo: likes(jake, \_0)?

(1) 1 Exit: likes(jake, apricots)

(3) 1 Call: likes(darcie, apricots)?

(3) 1 Exit: likes(darcie, apricots)

X = apricots



* Struktur List

Struktur data basic lain yang bukan merupakan proposisi atomic yang sudah pernah kita lihat merupakan suatu list. List merupakan urutan dari berbagai macam elemen. Elemen dapat terdiri dari atom, proposisi atomik, atau terms lain. Contoh :

[apple, prune, grape, kumquat]

[] ***(****empty list)*

[X | Y] ***(****head X and tail Y)*

Contoh lain :

reverse([], []).

reverse([Head | Tail], List) :-

reverse (Tail, Result),

append (Result, [Head], List).

member(Element, [Element | \_]).

member(Element, [\_ | List]) :-

member(Element, List).

* **Kekurangan Prolog**
* Resolusinya bersifat order control
* Asumsi yang bersifat closed-world
* Masalah negasi
* Terbatasnya intrinsik
* **Penggunaan Logic Programming**
* Sistem manajement relasi database
* Sistem expert
* Proses bahasa natural

Referensi :

Bab 1:

* <https://www.scribd.com/doc/27463710/Language-Evaluation-Criteria>
* <https://muhammadghazali.wordpress.com/2008/04/24/kenapa-harus-belajar-pemrograman/>
* <http://www.w3schools.com/php/php_exception.asp>
* <https://abangpreuner.wordpress.com/2014/02/04/php-exception-handling/>
* <http://www.nulis-ilmu.com/2014/11/penanganan-exception-pada-php.html>
* Robert W. Sebesta - Concept of Programming Languages (Tenth Edition), Chapter 1
* https://binusmaya.binus.ac.id/newStudent/#/class/resources.COMP6060/007420/1610/LEC/9239

Bab 2 :

* http://arcana70.blogspot.co.id/2014/04/kbp-pert-2.html
* Robert W. Sebesta - Concept of Programming Languages (Tenth Edition), Chapter 3
* https://binusmaya.binus.ac.id/newStudent/#/class/resources.COMP6060/007420/1610/LEC/9239

Bab 3 :

* https://binusmaya.binus.ac.id/newStudent/#/class/resources.COMP6060/007420/1610/LEC/9239
* <http://www.w3schools.com/php/php_variables.asp>
* http://arcana70.blogspot.co.id/2014/04/kbp-pert-3.html
* Robert W. Sebesta - Concept of Programming Languages (Tenth Edition), Chapter 5

Bab 4 :

* <http://ajarwebs.blogspot.co.id/2012/11/tipe-data-dalam-php.html>
* <http://www.duniailkom.com/mengenal-tipe-data-string-dan-cara-penulisan-string-dalam-php/>
* <http://www.w3schools.com/php/php_arrays.asp>
* <http://php.net/manual/en/language.references.php>
* <http://php.net/manual/en/function.list.php>
* <http://arcana70.blogspot.co.id/2014/03/kbp-pert-4.html>
* en.wikipedia.org
* [www.quora.com](http://www.quora.com)
* [www.stackoverflow.com](http://www.stackoverflow.com)
* https://binusmaya.binus.ac.id/newStudent/#/class/resources.COMP6060/007420/1610/LEC/9239
* Robert W. Sebesta - Concept of Programming Languages (Tenth Edition), Chapter 6

Bab 5 :

* <http://php.net/manual/fa/language.operators.precedence.php>
* [https://books.google.co.id/books?id=Y9opn9g-weEC&pg=PA15&lpg=PA15&dq=tipe+konversi+eksplisit+dan+implisit+php&source=bl&ots=VY3Rbn0pgG&sig=\_6jqTq6WgidzXTsU6atN\_Bc2HzQ&hl=id&sa=X&ved=0ahUKEwjwnvG16efPAhXKtY8KHUiOABQQ6AEIIjAB#v=onepage&q=tipe%20konversi%20eksplisit%20dan%20implisit%20php&f=false](https://books.google.co.id/books?id=Y9opn9g-weEC&pg=PA15&lpg=PA15&dq=tipe+konversi+eksplisit+dan+implisit+php&source=bl&ots=VY3Rbn0pgG&sig=_6jqTq6WgidzXTsU6atN_Bc2HzQ&hl=id&sa=X&ved=0ahUKEwjwnvG16efPAhXKtY8KHUiOABQQ6AEIIjAB" \l "v=onepage&q=tipe%20konversi%20eksplisit%20dan%20implisit%20php&f=false)
* http://arcana70.blogspot.co.id/2014/03/kbp-pert-5\_23.html
* https://binusmaya.binus.ac.id/newStudent/#/class/resources.COMP6060/007420/1610/LEC/9239
* Robert W. Sebesta - Concept of Programming Languages (Tenth Edition), Chapter 7

Bab 6 :

* http://www.dreamincode.net/forums/topic/84706-pretest-and-posttest-loop-help/
* https://prezi.com/m8ni9bsqa5fc/kbp/
* http://www.dumetschool.com/blog/Penggunaan-Fungsi-Break-dan-Continue-pada-PHP
* http://www.w3schools.com/php/php\_looping.asp
* Robert W. Sebesta - Concept of Programming Languages (Tenth Edition), Chapter 8
* http://arcana70.blogspot.co.id/2014/06/kbp-pert-6.html
* https://binusmaya.binus.ac.id/newStudent/#/class/resources.COMP6060/007420/1610/LEC/9239

Bab 7 :

* Robert W. Sebesta - Concept of Programming Languages (Tenth Edition), Chapter 9
* https://www.hscripts.com/tutorials/php/function/passing-arguments.php
* http://arcana70.blogspot.co.id/2014/04/kbp-pert-7.html
* https://cseweb.ucsd.edu/classes/sp05/cse130/lecture\_notes/subprograms\_student.txt
* http://php.net/manual/en/language.references.pass.php
* <http://groups.engin.umd.umich.edu/CIS/course.des/cis400/maxim/lectures/chp8.htm>
* <http://php.net/manual/en/language.oop5.overloading.php>
* <https://code.tutsplus.com/tutorials/understanding-and-applying-polymorphism-in-php--net-14362>
* [diporw.blogspot.co.id](http://diporw.blogspot.co.id)
* [stackoverflow.com](http://stackoverflow.com)
* 7seasons.wordpress.com
* en.wikipedia.org
* https://binusmaya.binus.ac.id/newStudent/#/class/resources.COMP6060/007420/1610/LEC/9239

Bab 8 :

* Robert W. Sebesta - Concept of Programming Languages (Tenth Edition), Chapter 11
* <http://arcana70.blogspot.co.id/2014/05/kbp-pert-8.html>
* <https://en.wikipedia.org/wiki/Encapsulation_(computer_programming)>
* https://binusmaya.binus.ac.id/newStudent/#/class/resources.COMP6060/007420/1610/LEC/9239
* <http://www.saviost.net/object-oriented-programming-oop-encapsulation-inheritance-polymorphism-abstraction/>

Bab 9 :

* Robert W. Sebesta - Concept of Programming Languages (Tenth Edition), Chapter 12
* <http://teknosains.com/i/belajar-basic-oop-object-oriented-programming-dengan-php>
* <https://fatihamaliah.wordpress.com/2013/04/02/pengertian-konsep-oop-object-oriented-programming/>
* <http://arcana70.blogspot.co.id/2014/06/kbp-pert-9.html>
* https://binusmaya.binus.ac.id/newStudent/#/class/resources.COMP6060/007420/1610/LEC/9239

Bab 10 :

* Robert W. Sebesta - Concept of Programming Languages (Tenth Edition), Chapter 13
* <http://arcana70.blogspot.co.id/2014/06/kbp-pert-10.html>
* http://panduofunited.blogspot.co.id/2012/02/pengertian-concurrency.html
* https://binusmaya.binus.ac.id/newStudent/#/class/resources.COMP6060/007420/1610/LEC/9239

Bab 11 :

* <http://www.w3schools.com/php/php_exception.asp>
* <https://abangpreuner.wordpress.com/2014/02/04/php-exception-handling/>
* <http://www.nulis-ilmu.com/2014/11/penanganan-exception-pada-php.html>
* Robert W. Sebesta - Concept of Programming Languages (Tenth Edition), Chapter 14
* https://binusmaya.binus.ac.id/newStudent/#/class/resources.COMP6060/007420/1610/LEC/9239

Bab 12 :

* http://arcana70.blogspot.co.id/2014/06/kbp-pert-12.html
* Robert W. Sebesta - Concept of Programming Languages (Tenth Edition), Chapter 15
* https://www.tutorialspoint.com/lisp/lisp\_data\_types.htm
* https://binusmaya.binus.ac.id/newStudent/#/class/resources.COMP6060/007420/1610/LEC/9239

Bab 13 :

* http://arcana70.blogspot.co.id/2014/06/kbp-pert-13.html
* https://binusmaya.binus.ac.id/newStudent/#/class/resources.COMP6060/007420/1610/LEC/9239
* Robert W. Sebesta - Concept of Programming Languages (Tenth Edition), Chapter 16