1. INTRODUCTION

* 1. Reasons for Studying Concepts of Programming Languages

Sangatlah umum jika seorang murid mempertanyakan bagaimana dan apa keuntungan yang akan mereka terima dengan memperlajari sebuah konsep bahasa pemrograman. Setiap ilmu pelajaran tentunya seharusnya ditopang oleh sebuah konsep sebagai pijakannya, sama halnya dengan mempelajari sebuah bahasa pemrograman, bagaimana mungkin terciptanya pengaplikasian yang baik dan benar jika kita tidak memahami konsep dari bahasa pemrograman secara mendalam, mempelajari konsep bahasa pemrograman memberikan kita pengaruh dalam mengekspresikan bahasa dan pemikiran kita ke dalam sebuah bahasa pemrograman yang dapat menciptakan sebuah program.

Dalam pengembangan setiap aplikasi maupun sebuah program mengerti dan memahami konsep dari bahasa pemrograman dapat memudahkan dan memberikan kita akses lebih dalam memperoleh pengaplikasian yang jauh lebih praktis, simpel, dan sesuai dengan harapan.

Kesadaran lebih banyak jenis fitur bahasa pemrograman dapat mengurangi keterbatasan tersebut dalam pengembangan perangkat lunak.

Mungkin dapat dikatakan bahwa mempelajari kemampuan bahasa lain tidak membantu seorang programmer menjadi terpaksa untuk menggunakan bahasa yang tidak menjadi kemampuannya tersebut. Argumen tersebut tidak dapat bertahan lama, namun, karena sering konstruksi bahasa dapat disimulasikan dalam bahasa lain yang tidak mendukung mereka konstruksi langsung.

Memperluas pemahaman untuk memilih bahasa yang sesuai. Banyak programmer professional yang memiliki sedikit pendidikan formal di ilmu komputer, bukan, mereka telah belajar pemrograman sendiri atau melalui program pelatihan in-house. Program pelatihan tersebut sering mengajarkan satu atau dua bahasa yang secara langsung relevan dengan pekerjaan saat ini di dalam organisasi. Banyak programmer lain menerima pelatihan formal mereka di masa lalu. Bahasa yang mereka pelajari kemudian tidak lagi digunakan, dan banyak fitur sekarang tersedia dalam bahasa pemrograman yang tidak dikenal secara luas. Hasil latar belakang ini menunjukkan bahwa banyak programmer, ketika diberikan pilihan bahasa untuk sebuah proyek baru, terus menggunakan bahasa yang telah akrab baginya, bahkan walaupun bahasa yang mereka gunakan tidak cocok dengan proyek yang mereka terima.

Meningkatkan kemampuan untuk belajar sebuah bahasa baru. pemrograman komputer adalah hal yang fundamental, dan metodologi desain, pengembangan perangkat lunak, dan bahasa pemrograman masih dalam kondisi berevolusi secara terus menerus. Hal ini membuat pengembangan perangkat lunak merupakan profesi yang menarik, tetapi juga berarti bahwa belajar terus menerus adalah penting. Proses belajar bahasa pemrograman baru dapat berlangsung dalam jangka yang panjang dan sulit, terutama bagi seseorang yang nyaman dengan hanya satu atau dua bahasa dan tidak pernah memperlajari konsep bahasa pemrograman, menjadi jauh lebih mudah untuk mempelajari sebuah bahasa permrograman dengan memasukkan konsep-konsep ini ke dalam desain bahasa yang tengah dipelajari.

* 1. Programming Domain
* [General purpose applications](https://en.wikipedia.org/wiki/Application_software)
* Rapid [software prototyping](https://en.wikipedia.org/wiki/Software_prototyping)
* [Financial time series analysis](https://en.wikipedia.org/wiki/Financial_time_series_analysis)
* [Natural language processing](https://en.wikipedia.org/wiki/Natural_language_processing)
* [Artificial intelligence reasoning](https://en.wikipedia.org/wiki/Artificial_intelligence_reasoning)
* [Expert systems](https://en.wikipedia.org/wiki/Expert_systems)
* [Gameplay Programming](https://en.wikipedia.org/w/index.php?title=Gameplay_Programming&action=edit&redlink=1)
* [Relational database querying](https://en.wikipedia.org/wiki/Relational_database_querying)
* [Theorem proving](https://en.wikipedia.org/wiki/Theorem_proving)
* [Systems design and implementation](https://en.wikipedia.org/wiki/Systems_design_and_implementation)
* [Application scripting](https://en.wikipedia.org/wiki/Application_scripting)
* [Domain-specific applications](https://en.wikipedia.org/wiki/Domain-specific_applications)
* [Programming education](https://en.wikipedia.org/wiki/Programming_education)
* [Internet](https://en.wikipedia.org/wiki/Internet)
* [Symbolic mathematics](https://en.wikipedia.org/wiki/Symbolic_mathematics)
* [Numerical mathematics](https://en.wikipedia.org/wiki/Numerical_mathematics)
* [Statistical applications](https://en.wikipedia.org/w/index.php?title=Statistical_applications&action=edit&redlink=1)
* [Text processing](https://en.wikipedia.org/wiki/Text_processing)
* [Matrix algorithms](https://en.wikipedia.org/w/index.php?title=Matrix_algorithms&action=edit&redlink=1)
* [Image processing](https://en.wikipedia.org/wiki/Image_processing)/[video processing](https://en.wikipedia.org/wiki/Video_processing)/[computer vision](https://en.wikipedia.org/wiki/Computer_vision)
  1. Language Evaluation Criteria

Selain hanya memeriksa konsep yang mendasari berbagai konstruksi / fitur dari bahasa pemrograman, Sebesta bertujuan juga untuk mengevaluasi fitur tersebut sehubungan dengan bagaimana mereka mempengaruhi proses pengembangan perangkat lunak, termasuk pemeliharaan.

Jadi dia menetapkan kriteria beberapa evaluasi yaitu Readability, Writability, Reliabilty, dan Cost dan beberapa karakteristik bahasa pemrograman yang harus dipertimbangkan ketika mengevaluasi bahasa sehubungan dengan kriteria tersebut.

* Readability :

Sebuah bahasa pemrogrmana harus dapat dengan mudah untuk dibaca dan dipahami, dan dapat diimplementasikan ke dalam sebuah program.

* Writability :

Dalam kaitannya dengan hal ini, meninjau bagaimana suatu bahasa pemrograman dapat digunakan dalam pengembangan sebuah program, dan kemudahan dalam penulisannya. Mendukung abstraksi, dalam hal ini mendukung programmer untuk mengartikan dan mendefinisikan sebuah bahasa pemrograman dan dituliskan sesuai dengan pemahamannya masing-masing.

* Reliability :

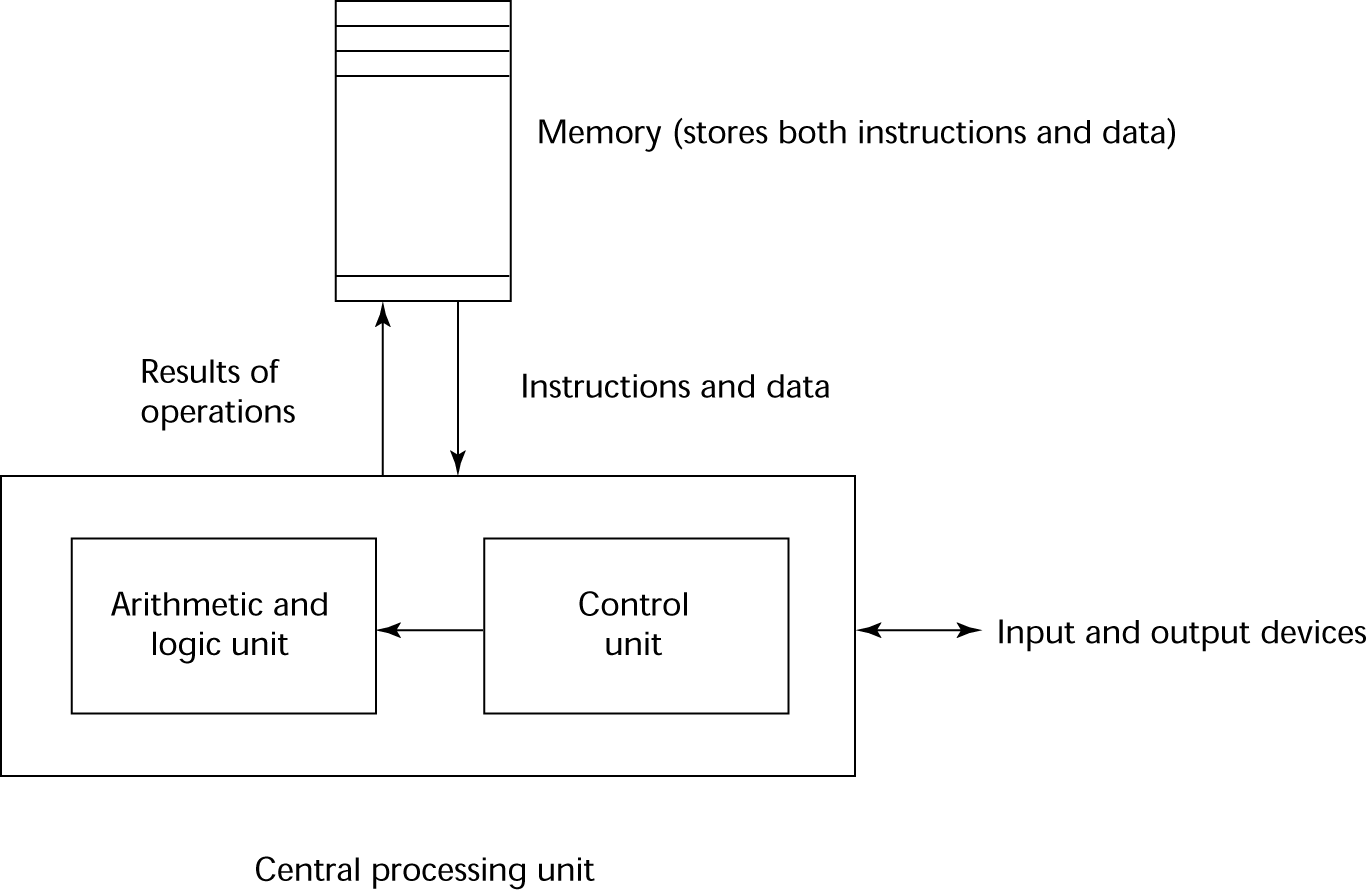
Didukung oeh berbagai spesifikasi dan dalam segala kondisi. Sebagai contoh kurangnya reliability, pertimbangkan versi bahasa C yang sebelumnya, di mana compiler tidak berusaha untuk memastikan bahwa argumen yang dilewatkan ke fungsi adalah dari jenis yang tepat.

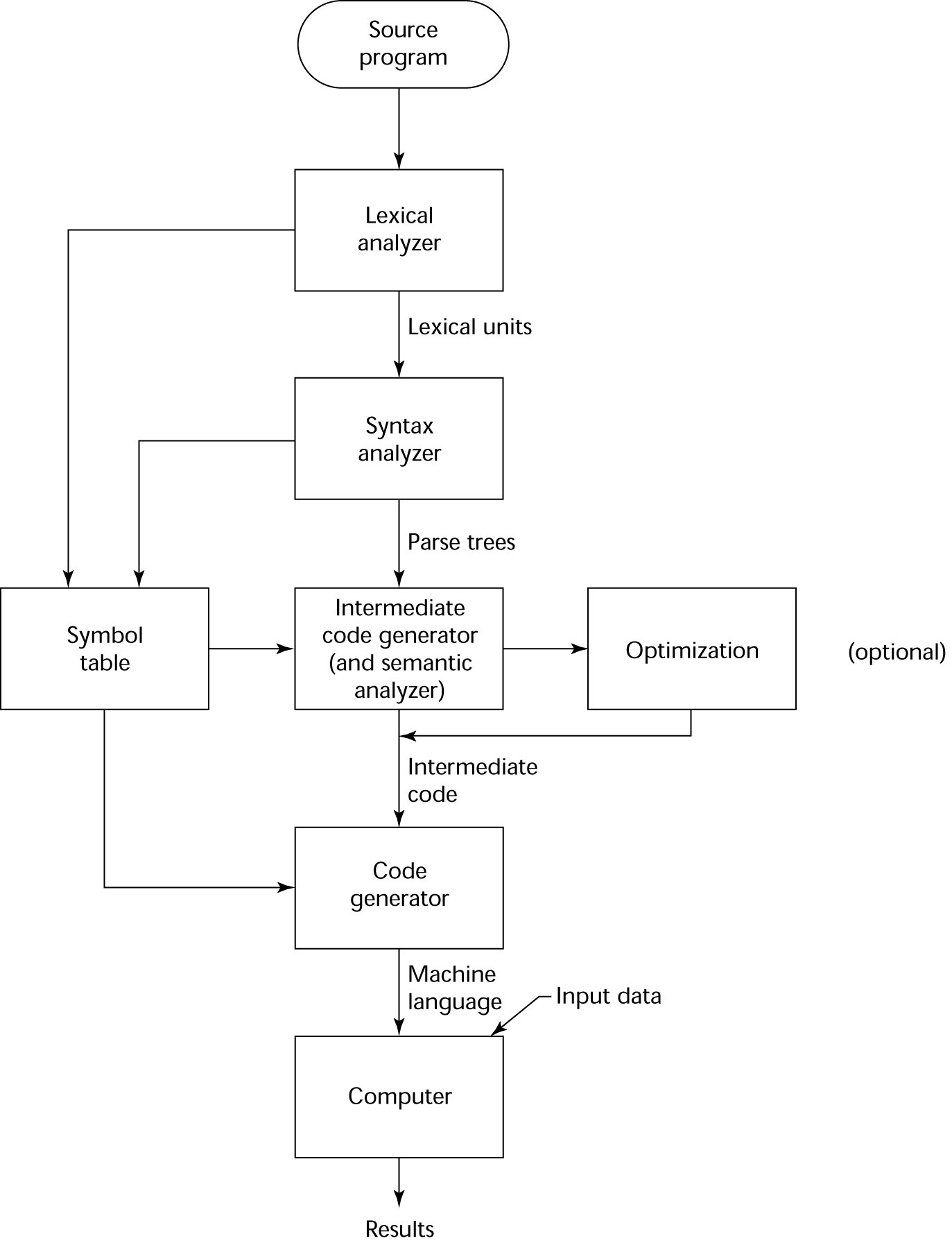
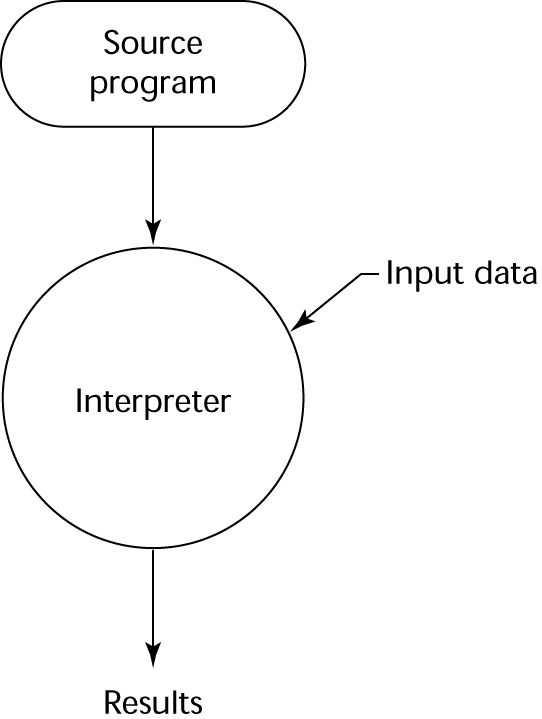
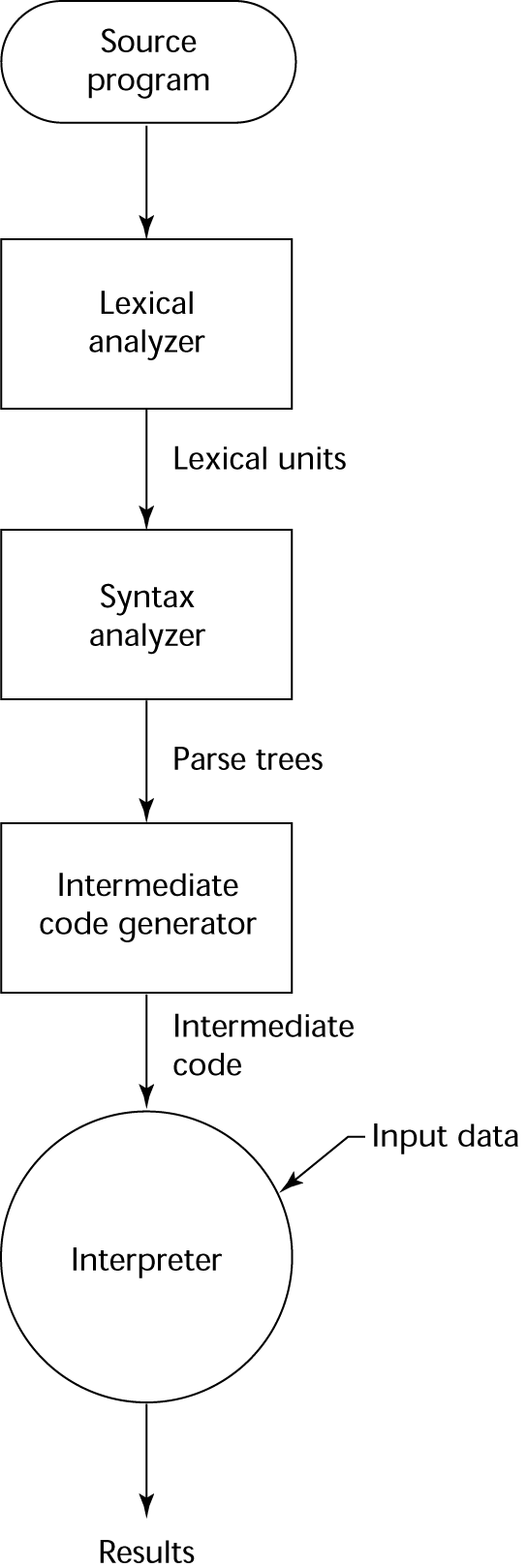
Readablitiy dan Writability: Keduanya memiliki pengaruh terhadap Reliabilty dalam arti bahwa programmer lebih mungkin untuk menghasilkan program yang mendukung segala kondisi ketika menggunakan bahasa yang memiliki sifat ini.

* Cost :
* Pelatihan Programmer
* Penulisan dan perawatan program.
* Melakukan kompilasi program.
* Pengimplementasiian ke dalam sistem.
* Kekurangan yang terdapat didalam Reliability, ex : Kegagalan software bisa menjadi mahal.
  1. Influences on Language Design

Mesin Von Neumann baik sebagai sejarah dan perspektif struktural, kita meneliti arsitektur komputer disimpan dalam program pertama. Sebagian besar komputer masih berfungsi dengan konsep yang sangat mirip dengan EDVAC.

Konsep penyimpanan program komputer sering dikaitkan dengan John von Neumann, yang merupakan orang pertama yang mengajukan idenya. Proposalnya pada tahun 1945 menggambarkan EDVAC (Electronic Discrete Variable Computer).

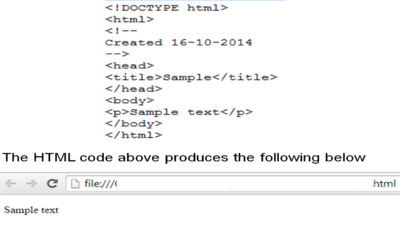
Implementasi pertama dari penyimpanan program komputer, menggunakan konstruksi von Neumann, adalah IAS (Institute for Advanced Studies, Princeton) komputer. Selesai pada tahun 1952, desain diambil dari proposal oleh John von Neumann, et al. Seperti dijelaskan longgar di [Sethi] (dengan perbaikan diambil dari [STALLINGS]) :

* 1. Language Categories
* Imperative.
* Functional.
* Logic.
* Markup/programming hybrid .
  1. Implementation Methods
* Compilation
* Program ditranslasikan ke dalam bahasa mesin, termasuk sistem-sistem JIT.
* Translasi lambat, eksekusi berlangsung cepat.
* Fase-fase kompilasi :
* Lexical Analysis: Mengkonversi karakter dalam sumber program ke dalam unit lexical.
* Syntax Analysis: Mengubah unit lexical menjadi parse tree yang merepresentasikan struktur sintaks dari program.
* Semantics Analysis: Membentuk kode.
* Code Generation: Kode mesin berhasil terbentuk.
* Proses Compilation :
* Kegunaan: Large commercial applications.
* Pure Interpretation
* Program di interpretasikan oleh program lain yang disebut interpreter.
* Tidak proses translasi.
* Kemudahan dalam pengimplementasian program.(Run time error dapat dengan mudah dideteksi).
* Ekesekusi cenderung lambat (10 sampai 100 kali lebih lambat dari kompilasi program).
* Membutuhkan memori yang lebih banyak.
* Proses Pure Interpretation :
* Kegunaan : Program kecil atau ketiak efisiensi bukan menjadi permasalahan.
* Hybrid Implementation Systems
* Persetujuan antara Kompiler dengan Interpretasi Murni.
* Eksekusi lebih cepat dibandingkan dengan Interpretasi murni.
* Proses Hybrid Implementation Systems :
* Kegunaan : Sistem besar maupun kecil atau ketika efisiensi bukan menjadi perhatian yang utama.
  1. HTML

HyperText Markup Language (HTML) bahasa yang digunakan dalam pengembangan sebuah halaman dari web dan aplikasi web, dengan malakukan kombinasi dengan bahasa seperti [Cascading Style Sheets](https://en.wikipedia.org/wiki/Cascading_Style_Sheets) (CSS), dan [JavaScript](https://en.wikipedia.org/wiki/JavaScript), membentuk tiga serangkai teknologi landasan untuk World Wide Web. Web dokumen HTML penerimaan browser dari webserver atau dari penyimpanan lokal dan membuat mereka ke dalam halaman web multimedia. HTML menggambarkan struktur dari halaman web semantik dan awalnya termasuk isyarat dalam kemunculan dokumen.

Elemen dalam HTML adalah blok bangunan dari halaman HTML. Dengan konstruksi HTML, gambar dan benda-benda lain, seperti formsmay interaktif tertanam ke dalam halaman yang diberikan. Ini menyediakan sarana untuk membuat dokumen terstruktur oleh struktural semantik untuk teks seperti judul, paragraf, daftar, link, kutipan dan item lainnya. elemen HTML digambarkan oleh tag, ditulis menggunakan “<” dan diakhhiri dengan “>”. Tag seperti <img /> dan <input /> merupakan konten dalam halaman. Lainnya seperti <p> ​​... </ p> mencakup dan memberikan informasi tentang dokumen teks dan dapat mencakup tag lain sebagai sub-elemen. Browser tidak menampilkan tag HTML, tetapi menggunakan mereka untuk menafsirkan isi dari halaman.

HTML dapat menanamkan program yang ditulis dalam bahasa scripting seperti JavaScript yang dapat mempengaruhi perilaku dan isi halaman web. Pencantuman CSS mendefinisikan tampilan dan tata letak konten. World Wide Web Consortium (W3C), pengelola dari kedua HTML dan standar CSS, telah mendorong penggunaan CSS lebih HTML presentasi eksplisit sejak tahun 1997.

* Sejarah
* Pada tahun 1986 ISO (International Standar Organization) spesifikasi (spesifikasi adalah setara dengan cetak biru ) untuk sebuah bahasa baru yang disebut SGML dirilis. Ini dimaksudkan untuk membuat penampilan dokumen platform independen.
* [](https://en.wikipedia.org/wiki/File:HTML_Example_Code.png)[](https://en.wikipedia.org/wiki/File:Tim_Berners-Lee_April_2009.jpg)Pada tahun 1989 proposal diciptakan oleh Tim Berners-Lee untuk sistem dokumen hypertext. Hal ini kemudian menghasilkan penciptaan HTTP (mekanisme untuk memindahkan data antara komputer Anda dan komputer server), URL (skema pengalamatan) dan HTML (bahasa sendiri). Ini dikenal sebagai HTML pada Oktober 1990.
* Versi pertama bekerja pada komputer NeXT dan hanya diproses file teks.
* Pada tahun 1991 kode dan spesifikasi ditempatkan di internet. Dalam beberapa tahun berbagai browser diberi kode untuk mendukung standar, dan HTML segera mulai penangkapan di seluruh internet.
* Pada tahun 1993 sebuah kelompok bernama NCSA (National Center for Supercomputing Applications) mulai merilis sebuah browser yang disebut Mosiac pada berbagai platform.
* Versi pertama dari browser baru yang revolusioner yang disebut Netscape dirilis dalam bentuk Beta pada tahun 1994. Ini dengan cepat menjadi browser yang paling populer yang tersedia.
* Pada tahun 1995 rilis final dari browser baru yang disebut Internet Explorer telah mengeluarkan pada public. Browser ini didasarkan pada Mosiac (pada kenyataannya, jika Anda memilih “About” dari menu “Help” Anda masih bisa melihat bahwa dalam informasi hak cipta) dan turun ke awal yang sangat lambat.
* Selama beberapa tahun berikutnya perang dikembangkan antara dua browser utama (Netscape dan Internet Explorer).
* Pada tahun 1996 namun browser lain dirilis – satu ini disebut Opera. Tidak seperti IE dan Netscape, Opera ditulis dari awal dan tidak bergantung pada apapun pembangunan sebelumnya. Oleh karena itu, browser sangat unik, dan juga sangat kecil dan cepat.
* Pada tahun 1999 Internet Explorer dengan cepat menjadi browser yang paling populer. Pada tahun 1999 dan 2000, spesifikasi untuk sebuah bahasa baru yang disebut XML yang dirilis. Proposal berada di udara untuk membuat spesifikasi HTML baru yang menggabungkan HTML dan XML menjadi satu paket.
* XML : XML (Extensible Markup Language) adalah bahasa markup untuk keperluan umum yang disarankan oleh W3C untuk membuat dokumen markup keperluan pertukaran data antar sistem yang beraneka ragam. XML merupakan kelanjutan dari HTML (HyperText Markup Language) yang merupakan bahasa standar untuk melacak Internet.

1. SYNTAX & SEMANTICS
   1. What is Syntax & Semantic

* Sintaks & Semantik :

Sintaks merupakan kumpulan aturan yang mendefinisikan suatu bentuk bahasa. Sintaks mendefinisikan bagaimana suatu kalimat dibentuk sebagai barisan/urutan dari pemilihan suatu kata dasar. Kata bukan merupakan sesuatu yang mendasar. Kata dikonstruksikan dengan karakter-karakter alfabet. Dengan menggunakan aturan ini maka suatu kalimat dapat dikatakan legal atau tidak legal. Sebagai contoh, dalam keyword bahasa C (seperti while, do,if, dan else), identifier, angka, operator, dan seterusnya, merupakan kata dalam suatu bahasa.Sintaks dalam bahasa C mengatur cara mengombinasikan kata-kata tersebut ke dalam suatu statement dengan bentuk yang benar sehingga dapat disusun suatu program yang dapat berjalan dengan benar. Sintaks tidak mengerti apapun tentang isi atau arti dari suatu kalimat, aturan semantik yang bisa mengartikannya.

Sintaks dari bahasa pemrograman didefinisikan dengan dua kumpulan aturan, yaitu aturan lexical dan aturan syntactic. Aturan leksikal menspesifikasikan kumpulan karakter yang terdapat dalam alfabet dari bahasa dan cara supaya karakter-karakter tersebut dikombinasikan ke dalam kata-kata yang valid dan dapat diterima. Sebagai contoh adalah bahasa Pascal yang tidak memperhatikan huruf besar dan huruf kecil, sedangkan bahasa C dan Ada memperhatikan hal tersebut.

Sintaks berfungsi menyediakan bentuk-bentuk notasi untuk komunikasi antar programmer dan pemroses bahasa pemrograman sehingga dapat mempermudah pembuatan suatu program.

Semantic mendefinisikan arti dari program yang benar secara sintak dari bahasa tersebut. Semantic suatu bahasa membutuhkan semacam ekspresi untuk mengirimkan suatu nilai kebenaran (TRUE, FALSE, NOT atau nilai integer). Dalam banyak kasus, program hanya dapat dieksekusi jika benar, serta mengikuti aturan sintaks dan semantik.

* Menotasikan semantic :
* Programmer harus paham dengan apa arti dari sebuah pernyataan.
* Kompiler harus mengetahui apa yang dilakukan oleh kontruksi sebuah bahasa pemrograman.
* Memungkinkan adanya bukti kebenaran.
* Memungkinkan adanya Generator Compiler.
* Desainer bisa mendeteksi ambiguitas dan inkonsistensi
* Syntaks dan semantik menyediakan definisi dari bahasa pemrograman.

* 1. Syntax in HTML

HTML disusun oleh tag-tag yang membuka dan menutup kode dari HTML, dan tag-tag tersebut dituliskan ke dalam sebuah file teks yang berekstensi “.html”/”.htm”. Lantas bagaimana sebuah browser dapat membedakan yang manakah merupakan bagian dari tag dan atribut serta yang bukan.

Format penulisan dalam HTML adalah sebagai berikut :

<tagname>

Sebuah tagname dibuka oleh (<) dan ditutup oleh (>), dan semua yang berada didalamnya akan dikenali oleh browser sebagai sebuah tag dari HTML. Sebuah tagname tentunya akan membutuhkan sebuah tagname penutup yang dituliskan dengan “</tagname>”. Susunan tagname penutup tersebut ditandai dengan tanda slash yang menjadi tanda bagaimana sebuah browser mengenali tagname penutup sebagai sebuah penutup dari sebuah tagname.

Atribut ataupun karakter yang diapit oleh kedua tagname pembuka dan penutup disebut dengan konten, konten inilah yang nantinya akan ditampilkan ke dalam sebuah halaman web. Contohnya adalah sebagai berikut :

<p>HELLO WORLD I LOVE HTML</p>

Konten tersebut akan ditampilkan dalam sebuah halaman web dengan tag yang berektensi sebagai paragraf. Beberapa tag ada yang disebut dengan singleton tag, yaitu tag yang beridiri sendiri, dan tidak membutuhkan tag penutup yang mengakhiri sebuah konten, sedikit berbeda dengan HTML dalam XHTML dalam tag diakhiri dengan slash.

Contohnya adalah tag Line break : **HTML:** <br>, **XHTML:** <br />

Jadi, Sintaks tag dalam HTML, adalah sebagai berikut :

* Penulisan tag HTML umumnya ditulis secara berpasangan. Ex: <head>…</head>.
* Tag HTML tidak case sensitive.
* Diperbolehkan adanya nested tag .
* Setiap tag mempunyai nama-nama yang spesifik.
* Atribut suatu tag diletakan setelah nama tag.
* Nilai dari atribut ditulis dengan diikuti “=“.
  1. **Semantic Element & Non Semantic Element in HTML**

**Semantik dalam HTML adalah menegaskan mengenai makna dari sintaks yang tertulis di dalam sebuah barisan kode. Sintaks dan semantic merupakan dua hal yang sulit untuk dipisahkan, dan keduanya saling berkaitan satu sama lainnya.**

* Contoh elemen-elemen *non-semantic* :
* <div>
* <span>
* <class>

Isinya tidak memiliki makna.

* Contoh elemen-elemen *semantic*: <form>
* <table>
* <header>
* <footer>

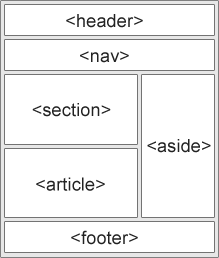
Semua tag ini memiliki makna yang jelas.

* Elemen-elemen semantic dalam HTML 5

Banyak web-web yang mengandunng kode HTML berupa :

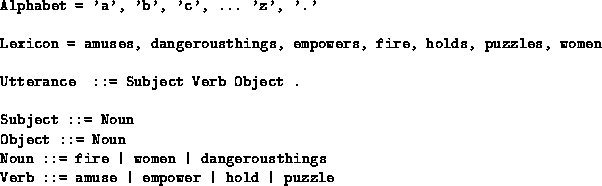
<div id="nav"> <div class="header"> <div id="footer">  
to indicate navigation, header, and footer.

HTML5 menawarkan elemen-elemen semantik untuk mendefinisikan bagian-bagian yang berbeda dari sebuah halaman web:

* <article>
* <aside>
* <details>
* <figcaption>
* <figure>
* <footer>
* <header>
* <main>
* <mark>
* <nav>
* <section>
* <summary>
* <time>
* HTML5 <details> = Berupa penjelasan tambahan yang dapat ditampilkan ataupun ditutup oleh pengguna.
* HTML5 <main> = Mendefinisikan bagian utama dari sebuah dokumen.
* HTML5 <mark> = Memberikan tanda dan tanda penjelas pada sebuah teks.
* HTML5 <summary> = Bagian yang ditampikan pada bagian atas dari <detail>.
* HTML5 <time> = Membuat tanggal dan waktu.
* HTML5 <section> = Mendefinisikan sebuah bagian dalam sebuah dokumen.
* HTML5 <article> = Merupakan bagian terpisah, yang berisi sebuah konten. Tag <article> pada umumnya digunakan untuk :
* Forum Post.
* Blog Post.
* Newspaper Article.
* HTML5 <header> = Menandakan bagian judul.header dari sebuah dokumen, umumnya digunakan sebagai konten untuk pembuka atau pengenalan.
* HTML5 <footer> = Bagian footer dari sebuah dokumen, berupa informasi mengenai isi dari konten-konten lainnya, pada umumnya seperti penulis.author, informasi copyright, link yang berisi informasi lanjutan, dan informasi kontak.
* HTML5 <nav> = Elemen link navigasi.
* HTML5 <aside> = Elemen mendefinisikan beberapa konten selain dari konten itu ditempatkan (seperti sidebar). Konten aside harus berkaitan dengan isi sekitarnya.
* HTML5 <figure> dan <figcaption> = Figure caption dipakai untuk memberikan penjelasan mengenai sebuah gambar.
  1. **Grammar**

**Dalam sebuah pelajaran bahasa tentunya kita tidak akan asing lagi dengan yang namanya grammar, serupa dengan bahasa-bahasa lainnya, sebuah bahasa pemrograman juga mempunyai grammar yang menjadi acuan dalam penulisan dari sebuah bahasa pemrograman.**

**Di dalam bahasa komputer, mencakup communication protocol, yang ditentukan dengan menggunakan sejenis meta-aturan. Di dalam bahasa komputer jauh sedikit lebih sederhana, dan sebagian besar ditentukan menggunakan Grammar dari grammers, diciptakan oleh dua ilmuwan komputer besar, Backus dan Naur, dan dinamai mereka Backus Naur Form atau BNF. Sebuah BNF terdiri dari satu set aturan produksi untuk kalimat. Sebuah kalimat yang selesai (menganggapnya sebagai sebuah pernyataan program, atau dalam kasus HTML, dokumen) terdiri dari kumpulan terminal atau kata-kata dasar dari suatu bahasa, disatukan menurut aturan. Kata-kata dapat ditarik dari tata bahasa sederhana disebut leksikon, yang pada dasarnya adalah aturan ejaan, dan mereka terdiri dari karakter (huruf) yang diambil dari alfabet. Hanya daftar huruf yang diperbolehkan.**

Contoh BNF

Trivial Nadsat Grammar.

Legal Utterances :

* fire holds dangerousthings
* women puzzles fire
* dangerousthings amuses women

* 1. Grammar in HTML

Berikut adalah Grammar (BNF) untuk SGML/HTML

O, grave accent -->

<!ENTITY Oslash "&#216;" -- capital O, slash -->

<!ENTITY Otilde "&#213;" -- capital O, CHARSET

BASESET "ISO 646:1983//CHARSET

International Reference Version (IRV)//ESC 2/5 4/0"

DESCSET 0 9 UNUSED

9 2 9

11 2 UNUSED

13 1 13

14 18 UNUSED

32 95 32

127 1 UNUSED

BASESET "ISO Registration Number 100//CHARSET

ECMA-94 Right Part of Latin Alphabet Nr. 1//ESC 2/13 4/1"

DESCSET 128 32 UNUSED

160 95 32

255 1 UNUSED

CAPACITY SGMLREF

TOTALCAP 150000

GRPCAP 150000

SCOPE DOCUMENT

SYNTAX

SHUNCHAR CONTROLS 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

19 20 21 22 23 24 25 26 27 28 29 30 31 127 255

BASESET "ISO 646:1983//CHARSET

International Reference Version (IRV)//ESC 2/5 4/0"

DESCSET 0 128 0

FUNCTION RE 13

RS 10

SPACE 32

TAB SEPCHAR 9

NAMING LCNMSTRT ""

UCNMSTRT ""

LCNMCHAR ".-"

UCNMCHAR ".-"

NAMECASE GENERAL YES

ENTITY NO

DELIM GENERAL SGMLREF

SHORTREF SGMLREF

NAMES SGMLREF

QUANTITY SGMLREF

NAMELEN 34

TAGLVL 100

LITLEN 1024

GRPGTCNT 150

GRPCNT 64

FEATURES

MINIMIZE

DATATAG NO

OMITTAG NO

RANK NO

SHORTTAG NO

LINK

SIMPLE NO

IMPLICIT NO

EXPLICIT NO

OTHER

CONCUR NO

SUBDOC NO

FORMAL YES

APPINFO NONE

>

<!DOCTYPE HTML [

<!-- Jul 1 93 -->

<!-- Regarding clause 6.1, SGML Document:

[1] SGML document = SGML document entity,

(SGML subdocument entity |

SGML text entity | non-SGML data entity)\*

The role of SGML document entity is filled by this DTD,

followed by the conventional HTML data stream.

-->

<!-- DTD definitions -->

<!ENTITY \% heading "H1|H2|H3|H4|H5|H6" >

<!ENTITY \% list " UL | OL | DIR | MENU ">

<!ENTITY \% literal " XMP | LISTING ">

<!ENTITY \% headelement

" TITLE | NEXTID |ISINDEX" >

<!ENTITY \% bodyelement

"P | HR | \%heading |

\%list | DL | ADDRESS | PRE | BLOCKQUOTE

| \%literal">

<!ENTITY \% oldstyle "\%headelement | \%bodyelement | #PCDATA">

<!ENTITY \% URL "CDATA"

-- The term URL means a CDATA attribute

whose value is a Uniform Resource Locator,

as defined. (A URN may also be usable here when defined.)

-->

<!ENTITY \% linkattributes

"NAME NMTOKEN #IMPLIED

HREF \%URL; #IMPLIED

REL CDATA #IMPLIED -- forward relationship type --

REV CDATA #IMPLIED -- reversed relationship type

to referent data:

PARENT CHILD, SIBLING, NEXT, TOP,

DEFINITION, UPDATE, ORIGINAL etc. --

URN CDATA #IMPLIED -- universal resource number --

TITLE CDATA #IMPLIED -- advisory only --

METHODS NAMES #IMPLIED -- supported public methods of the object:

TEXTSEARCH, GET, HEAD, ... -- ">

<!-- Document Element -->

<!ELEMENT HTML O O (( HEAD | BODY | \%oldstyle )\*, PLAINTEXT?)>

<!ELEMENT HEAD - - ( TITLE? & ISINDEX? & NEXTID? & LINK\*

& BASE?)>

<!ELEMENT TITLE - - RCDATA

-- The TITLE element is not considered part of the flow of text.

It should be displayed, for example as the page header or

window title.

-->

<!ELEMENT ISINDEX - O EMPTY

-- WWW clients should offer the option to perform a search on

documents containing ISINDEX.

-->

<!ELEMENT NEXTID - O EMPTY>

<!ATTLIST NEXTID N NAME #REQUIRED

-- The number should be a name suitable for use

for the ID of a new element. When used, the value

has its numeric part incremented. EG Z67 becomes Z68

-->

<!ELEMENT LINK - O EMPTY>

<!ATTLIST LINK

\%linkattributes>

<!ELEMENT BASE - O EMPTY -- Reference context for URLS -->

<!ATTLIST BASE

HREF \%URL; #IMPLIED

>

<!ENTITY \% inline "EM | TT | STRONG | B | I | U |

CODE | SAMP | KBD | KEY | VAR | DFN | CITE "

>

<!ELEMENT (\%inline;) - - (#PCDATA)>

<!ENTITY \% text "#PCDATA | IMG | \%inline;">

<!ENTITY \% htext "A | \%text" -- Plus links, no structure -->

<!ENTITY \% stext -- as htext but also nestedstructure --

"P | HR | \%list | DL | ADDRESS

| PRE | BLOCKQUOTE

| \%literal | \%htext">

<!ELEMENT BODY - - (\%bodyelement|\%htext;)\*>

<!ELEMENT A - - (\%text)>

<!ATTLIST A

\%linkattributes;

>

<!ELEMENT IMG - O EMPTY -- Embedded image -->

<!ATTLIST IMG

SRC \%URL; #IMPLIED -- URL of document to embed --

>

<!ELEMENT P - O EMPTY -- separates paragraphs -->

<!ELEMENT HR - O EMPTY -- horizontal rule -->

<!ELEMENT ( \%heading ) - - (\%htext;)+>

<!ELEMENT DL - - (DT | DD | \%stext;)\*>

<!-- Content should match ((DT,(\%htext;)+)+,(DD,(\%stext;)+))

But mixed content is messy. -Dan Connolly

-->

<!ELEMENT DT - O EMPTY>

<!ELEMENT DD - O EMPTY>

<!ELEMENT (UL|OL) - - (\%htext;|LI|P)+>

<!ELEMENT (DIR|MENU) - - (\%htext;|LI)+>

<!-- Content should match ((LI,(\%htext;)+)+)

But mixed content is messy.

-->

<!ATTLIST (\%list)

COMPACT NAME #IMPLIED -- COMPACT, etc.--

>

<!ELEMENT LI - O EMPTY>

<!ELEMENT BLOCKQUOTE - - (\%htext;|P)+

-- for quoting some other source -->

<!ELEMENT ADDRESS - - (\%htext;|P)+>

<!ELEMENT PRE - - (#PCDATA|\%inline|A|P)+>

<!ATTLIST PRE

WIDTH NUMBER #implied

>

<!-- Mnemonic character entities. -->

<!ENTITY AElig "&#198;" -- capital AE diphthong (ligature) -->

<!ENTITY Aacute "&#193;" -- capital A, acute accent -->

<!ENTITY Acirc "&#194;" -- capital A, circumflex accent -->

<!ENTITY Agrave "&#192;" -- capital A, grave accent -->

<!ENTITY Aring "&#197;" -- capital A, ring -->

<!ENTITY Atilde "&#195;" -- capital A, tilde -->

<!ENTITY Auml "&#196;" -- capital A, dieresis or umlaut mark -->

<!ENTITY Ccedil "&#199;" -- capital C, cedilla -->

<!ENTITY ETH "&#208;" -- capital Eth, Icelandic -->

<!ENTITY Eacute "&#201;" -- capital E, acute accent -->

<!ENTITY Ecirc "&#202;" -- capital E, circumflex accent -->

<!ENTITY Egrave "&#200;" -- capital E, grave accent -->

<!ENTITY Euml "&#203;" -- capital E, dieresis or umlaut mark -->

<!ENTITY Iacute "&#205;" -- capital I, acute accent -->

<!ENTITY Icirc "&#206;" -- capital I, circumflex accent -->

<!ENTITY Igrave "&#204;" -- capital I, grave accent -->

<!ENTITY Iuml "&#207;" -- capital I, dieresis or umlaut mark -->

<!ENTITY Ntilde "&#209;" -- capital N, tilde -->

<!ENTITY Oacute "&#211;" -- capital O, acute accent -->

<!ENTITY Ocirc "&#212;" -- capital O, circumflex accent -->

<!ENTITY Ograve "&#210;" -- capital tilde -->

<!ENTITY Ouml "&#214;" -- capital O, dieresis or umlaut mark -->

<!ENTITY THORN "&#222;" -- capital THORN, Icelandic -->

<!ENTITY Uacute "&#218;" -- capital U, acute accent -->

<!ENTITY Ucirc "&#219;" -- capital U, circumflex accent -->

<!ENTITY Ugrave "&#217;" -- capital U, grave accent -->

<!ENTITY Uuml "&#220;" -- capital U, dieresis or umlaut mark -->

<!ENTITY Yacute "&#221;" -- capital Y, acute accent -->

<!ENTITY aacute "&#225;" -- small a, acute accent -->

<!ENTITY acirc "&#226;" -- small a, circumflex accent -->

<!ENTITY aelig "&#230;" -- small ae diphthong (ligature) -->

<!ENTITY agrave "&#224;" -- small a, grave accent -->

<!ENTITY amp "&#38;" -- ampersand -->

<!ENTITY aring "&#229;" -- small a, ring -->

<!ENTITY atilde "&#227;" -- small a, tilde -->

<!ENTITY auml "&#228;" -- small a, dieresis or umlaut mark -->

<!ENTITY ccedil "&#231;" -- small c, cedilla -->

<!ENTITY eacute "&#233;" -- small e, acute accent -->

<!ENTITY ecirc "&#234;" -- small e, circumflex accent -->

<!ENTITY egrave "&#232;" -- small e, grave accent -->

<!ENTITY eth "&#240;" -- small eth, Icelandic -->

<!ENTITY euml "&#235;" -- small e, dieresis or umlaut mark -->

<!ENTITY gt "&#62;" -- greater than -->

<!ENTITY iacute "&#237;" -- small i, acute accent -->

<!ENTITY icirc "&#238;" -- small i, circumflex accent -->

<!ENTITY igrave "&#236;" -- small i, grave accent -->

<!ENTITY iuml "&#239;" -- small i, dieresis or umlaut mark -->

<!ENTITY lt "&#60;" -- less than -->

<!ENTITY nbsp "&#32;" -- should be NON\\_BREAKING space -->

<!ENTITY ntilde "&#241;" -- small n, tilde -->

<!ENTITY oacute "&#243;" -- small o, acute accent -->

<!ENTITY ocirc "&#244;" -- small o, circumflex accent -->

<!ENTITY ograve "&#242;" -- small o, grave accent -->

<!ENTITY oslash "&#248;" -- small o, slash -->

<!ENTITY otilde "&#245;" -- small o, tilde -->

<!ENTITY ouml "&#246;" -- small o, dieresis or umlaut mark -->

<!ENTITY szlig "&#223;" -- small sharp s, German (sz ligature) -->

<!ENTITY thorn "&#254;" -- small thorn, Icelandic -->

<!ENTITY uacute "&#250;" -- small u, acute accent -->

<!ENTITY ucirc "&#251;" -- small u, circumflex accent -->

<!ENTITY ugrave "&#249;" -- small u, grave accent -->

<!ENTITY uuml "&#252;" -- small u, dieresis or umlaut mark -->

<!ENTITY yacute "&#253;" -- small y, acute accent -->

<!ENTITY yuml "&#255;" -- small y, dieresis or umlaut mark -->

<!-- deprecated elements -->

<!ELEMENT (\%literal) - - CDATA>

<!ELEMENT PLAINTEXT - O EMPTY>

<!-- Local Variables: -->

<!-- mode: sgml -->

<!-- compile-command: "sgmls -s -p " -->

<!-- end: -->

]>

1. NAMES, BINDING & SCOPE
   1. Names

* Panjang Kata :
* Jika terlalu pendek, tidak dapat dikonotasikan.
* Contoh Bahasa :
* FORTRAN 95 : Maksimal 31.
* C99 : Tidak ada batas maksimal, namun hanya 63 karakter awal yang utama, juga penamaan lainnya maksimal hanya 31.
* C#, Ada, Java : Tidak ada batas.
* C++ : Tidak ada batas, namun penimplementasi menentukan bagian yang pertama.
* Karakter Spesial/Unik :
* PHP : Semua variabel harus diawali dengan tanda Dollar “$”,
* Perl : Semua variabel harus diawali dengan karakter special, yang menjelaskan/menspesifikasikan tipe variable.
* Ruby : Nama variabel diawali dengan @, dan untuk variabel yang diawali dengan @@ merupakan class variables.
* Case Sensitivity :
* Kekurangan : Readability (Nama yang terlihat sama memiliki makna yang berbeda).
* Nama dalam bahasa C case sensitive.
* Beberapa bahasa pemrograman tidak case sensitive.
* C++, Java, dan C# dengan nama yang sudah dikenal menggunakan mixed case.
* Kata Spesial/Unik :
* Membantu agar lebih mudah untuk dibaca dengan menggunakan pembatas yang memisahkan setiap klausa-klausa dari masing-masing pernyataan. Ex : Fortran.
* Real VarName (Real adalah tipe data diikuti dengan nama, Real merupakan keyword).
* Permasalahan dengan reserved words : Jika terlalu banyak, maka akan terjadi bentrok (Ex : COBOL dengan 300 reserved words);

* 1. Name in HTML
* Atribut nama HTML <a>
* HTML <a> tag

Example

The name attribute specifies the name of an anchor:

<a name="B2020">Binusian 2020</a><a name="B2020">Binusian 2020</a>

* Definisi dan Penggunaan

Atribut nama merupakan bagian dari elemen, yang menjadi dihubungkan dengan elemen di dalam JavaScript. Untuk elemen Form digunakan ketika data dikirim, elemen Map atribut nama di distribusikan dengan tag <img>’s yang menghubungkan antara atribut gambar dan map. Untuk elemen Meta atribut nama mengandung informasi dan nilai dari atribut sebuah konten. Elemen Param, digunakan secara berasama-sama dengan nilai atribut yang dispesifikasikan oleh tag <object>.

* Dalam penggunaannya atribut nama dapat digunakan pada beberapa elemen-elemen sebagai berikut :

|  |
| --- |
| Elements |
| [<button>](http://www.w3schools.com/TAgs/tag_button.asp) |
| [<fieldset>](http://www.w3schools.com/TAgs/tag_fieldset.asp) |
| [<form>](http://www.w3schools.com/TAgs/tag_form.asp) |
| [<iframe>](http://www.w3schools.com/TAgs/tag_iframe.asp) |
| [<input>](http://www.w3schools.com/TAgs/tag_input.asp) |
| [<keygen>](http://www.w3schools.com/TAgs/tag_keygen.asp) |
| [<map>](http://www.w3schools.com/TAgs/tag_map.asp) |
| [<meta>](http://www.w3schools.com/TAgs/tag_meta.asp) |
| [<object>](http://www.w3schools.com/TAgs/tag_object.asp) |
| [<output>](http://www.w3schools.com/TAgs/tag_output.asp) |
| [<param>](http://www.w3schools.com/TAgs/tag_param.asp) |
| [<select>](http://www.w3schools.com/TAgs/tag_select.asp) |
| [<textarea>](http://www.w3schools.com/TAgs/tag_textarea.asp) |

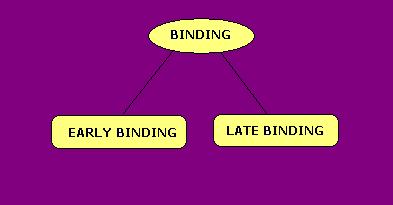
* 1. Binding

Binding adalah pengikatan alamat dengan cara instruksi dan data yang berada di disk sebagai file yang dapat dieksekusi, dengan melakukan pemetaan ke alamat memori. Sebagian besar sistem memperbolehkan sebuah proses dari user, untuk melakukan peletakkan disembarang tempat di dalam memori fisik. Maksudnya, jika alamat di dalam komputer di mulai dari 0000, maka alamat pertama dari user tidak harus di awali dengan 0000. Alamat pada source program pada umumnya merupakan alamat simbolik, sebuah compiler umumnya melakukan pengikatan alamt simbolik ke alamat relokasi. Pengikatan instruksi dan data ke alamat memori dapat dilakukan pada saat :

* Compile Time : Jika memori telah diketahui sejak awal, kode absolut dapat diaktifkan, jika terjadi perubahan alamat maka harus dilakukan compile ulang.
* Load Time :Membangkitakn kode relokasi jika memori belum diketahui pada saat kompilasi.
* Execution Time : Pengikatan ditunda sampai waktu eksekusi jika proses dapat dipindahkan dari satu segmen memori ke segmen yang lainnya, memerlukan adanya dukunagn hardware untuk meletakan alamat (misalnya register basis dan limit).

Binding dibagi menjadi dua jenis :

* Early Binding (Static Binding).
* Late Binding (Dynamic Binding).



* 1. Binding in HTML
* Tujuan : Binding pada HTML bertujuan untuk menagsosiasikan elemen DOM, atau sekedar memberikan tampilan yang lebih menarik pada suatu halaman web, karena HTML berfungsi untuk menampilkan dokumen pada halaman web, dan dengan melakukan binding pada CSS maka dapat memperindah tampilan suatu halaman web, karena CSS berfungsi sebagai pemberi tampilan desain pada suatu halaman web.

Contoh Binding :

<div data-bind="html: details"></div>

<script type="text/javascript">

var viewModel = {

details: ko.observable() // Initially blank

};

viewModel.details("<em>Click me<a href='click.html'>here</a>.</em>"); // HTML content appears

</script>

* Binding HTML Element to Data

Menampilkan data yang disediakan oleh Data Source Object (DSO), penulis mengikat elemen yang ada pada halaman HTML ke DSO. Kemampuan termasuk menudukung untuk perbaruan data pada unsur terikat dan format data di mana data yang ditampilkan (baik HTML atau teks biasa).

Elemen HTML yang dapat di binding di bagi menjadi dua, yaitu single valued dan tabular consumers.

* Binding a Single Valued Element to Data

Melakukan binding ke data dengan DATASCR dan DATAFLD atau DHTML.

* Binding a Single-Valued Element to Data

Cara memberikan sebuah text box, penulis sebuah halaman dapat melakukan binding element to data sebagai berikut :

<INPUT TYPE=TEXTBOX DATASRC="#dsoComposers" DATAFLD="compsr\_last">

* Binding a Single-Valued Element to Data at Design Time

Karena elemen tabel adalah data konsumen tabular, hal itu bergantung pada unsur-unsur yang tekandung mengikat ke ladang individu dalam kumpulan data yang disediakan oleh DSO. unsur yang terkandung berfungsi sebagai template, dan mereka diulang sekali untuk record dalam kumpulan data. Tabel menentukan dataSrcattribute. unsur yang terkandung menentukan atribut dataFld dan mewarisi dataSrc dari meja. Ketika mengikat meja untuk data, tidak mungkin untuk dataBind untuk tr jika itu terkandung oleh tfoot atau THead.

* Elemen yang Mendukung Data Binding
* Button : Membuat sebuah tombol.
* Div : Menampilkan kumpulan dari teks.
* Frame : Frame berlaku data yang diberikan oleh DSO untuk atribut src-nya; dengan demikian, nilai yang dibind harus mewakili URL. Bind berhasil, DSO memasok data ke Frame yang akan di bind harus hadir di bagian kepala dari file HTML yang berisi elemen frameset
* IFrame : Sebuah elemen iframe berlaku data yang diberikan oleh DSO untuk atribut src-nya; dengan demikian, data harus mewakili URL. Berbeda dengan kasusnya dengan frame, DSO dapat dinyatakan pada halaman manapun.
* Img : Elemen img menerima suplai data dari DSO untuk mengalokasikan, mengunggah, dan menampilkan gambar pada atribut src tertentu.
* Label : Menjelaskan elemen lain yang di bind pada halaman tersebut. Label di sediakan oleh DSO pada judul halamannya.
* Legend : Menambahkan judul pada kotak yang dibentuk oleh elemen fieldset.
* Marquee : Membuat tampilan teks yang muncul di antara pembuka dan tag penutup.
* Span : Hampir sama seperti div, span digunakan untuk mempuat tampilan inline text dan teks HTML dengan jumlah yang terbatas.
* Input : Membuat sebuah input sesuai dengan fungsinya untuk mengendalikan sesuatu (ex : Input type = Checkbox, Input type = Hidden, Input Type = Password, Input type = Radio, dan Input type = Text).
* Checkbox : Membuat sebuah pilihan Boolean, dengan pilihan yang dapat dipilih lebih dari satu pilihan yang telah disediakan.
* Password : Bersifat sama seperti teks pada umumnya, namun tidak ditampilkan secara langsung kepad User.
* Radio : Membuat beberapa alternative pilihan, yang hanya dapat dipilih satu pilihan/tidak lebih, berbeda dengan checkbox yang dapat memilih lebih dari seatu pilihan.
* Text : Membuat teks biasa.
* Object : Ketika dataSrc dan dataFld ditetapkan pada objek, Internet Explorer mencoba untuk melakukan binding ke properti default objek. Atribut defaultbind ditentukan dalam jenis informasi obyek unik mengidentifikasi properti default. Jika properti default tidak ditentukan dengan cara ini, Internet Explorer menggunakan properti dengan DISPID 0.
* Select : Membuat sebuah combo box ataupun list box.
* Text Area : Membuat teks area untuk menginput suatu data.
* Applet : Internet explorer mendukung binding dengan tag parameter dari applet.
  1. Scope

Dalam pemrograman komputer, scope merupakan asosiasi dari nama dengan batas tertentu dari variabel di dalam suatu fungsi. Variabel adalah bagian dari program komputer di mana binding ini berlaku: di mana nama bisa digunakan untuk merujuk pada suatu entitas.

Dalam praktek untuk sebagian besar bahasa pemrograman, bagian dari program mengacu pada sutu bagian dengan porsi dari sumber kode tertentu, dan dikenal sebagai lingkup leksikal. Namun dalam beberapa bahasa, terdapat bagian dari program yang juga mengacu pada porsi waktu run (jangka waktu selama eksekusi), dan dikenal sebagai lingkup dinamis.

Dalam kebanyakan kasus, resolusi nama berdasarkan lingkup leksikal lebih mudah untuk digunakan dan untuk diterapkan. kesulitan dasar timbul dalam masking nama, forward declarations, dan mengangkat, sementara yang jauh lebih halus timbul dengan variabel non-lokal, khususnya di penutupan.

* Lexical Scope & Dynamic Scope

Perbedaan yang paling mendasar dalam scoping adalah apa yang menjadi arti "bagian dari program". Dalam bahasa pemrogrmaan dengan lingkup leksikal (juga disebut lingkup statis), resolusi nama tergantung pada lokasi dalam sumber kode dan pada konteks leksikal, yang didefinisikan oleh dimana variabel atau fungsi bernama didefinisikan. Sebaliknya, dalam bahasa dengan lingkup dinamis resolusi nama tergantung pada keadaan program ketika nama ditemui yang ditentukan oleh konteks eksekusi atau konteks saat dipanggil. Dalam prakteknya, dengan ruang lingkup leksikal definisi variabel diselesaikan dengan mencari fungsi, sedangkan dengan lingkup dinamis fungsi yang melakukan panggilan akan di cari. Tentu saja, di kedua aturan, pertama kita mencari definisi lokal dari variabel. Kebanyakan bahasa modern lebih menggunakan scoping leksikal untuk variabel dan fungsi, meskipun scoping dinamis digunakan dalam beberapa bahasa, terutama beberapa seperti Lisp, beberapa "scripting" dalam bahasa seperti Perl, dan beberapa bahasa Template. Bahkan dalam bahasa leksikal scoped, ruang lingkup untuk penutupan dapat membingungkan bagi yang belum mengetahuinya, karena ini tergantung pada konteks leksikal mana penutupan didefinisikan, tidak di mana saat itu dipanggil.

* 1. Scope in HTML

Scope digunakan dalam menentukan apakah sebuah sel judul, merupakan bagian untuk kolom, baris atau kelompok kolom atau baris. Atribut scope tidak memiliki efek visual dalam web browser biasa, tetapi dapat digunakan oleh pembaca untuk melihatnya secara langsung melalui layar.

Elemen yang digunakan : <th> & <td>.

* Value

|  |  |
| --- | --- |
| Value | Description |
| col | Defines that the associated cell is a header for a column. |
| row | Defines that the associated cell is a header for a row. |
| colgroup | Defines that the associated cell is a header for a column group. |
| rowgroup | Defines that the associated cell is a header for a row group. |

## Contoh th dan td

1. **<html>**
2. **<head>**
3. **<meta** http-equiv="Content-Type" content="text/html; charset=iso-8859-1"**>**
4. **<title>**Ex for th & td**</title>**
5. **</head>**
6. **<body>**
7. **<table** width="200" border="1" cellpadding="2"**>**
8. **<tr>**
9. **<th** scope="col"**>**NUMBER **</th>**
10. **<th** scope="col"**>**NAME **</th>**
11. **<th** scope="col"**>**GRADE**</th>**
12. **<th** scope="col"**>**MARKS**</th>**
13. **<tr>**
14. **</tr>**
15. **<td>**1**</td>**
16. **<td>**Budi**</td>**
17. **<td>**100**</td>**
18. **<td>**A+**</td>**
19. **<tr>**
20. **</tr>**
21. **<td>**2**</td>**
22. **<td>**Udin**</td>**
23. **<td>**80**</td>**
24. **<td>**B+**</td>**
25. **</tr>**
26. **</table>**
27. **</body>**

Elemen th

1. **</html>**

|  |  |  |  |
| --- | --- | --- | --- |
| **NUMBER** | **NAME** | **GRADE** | **MARKS** |
| 1 | Budi | 100 | A+ |
| 2 | Udin | 80 | B+ |

Elemen td

1. DATA TYPES

* 1. Primitive Data Types (Simple)

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.

* Numeric

Tipe data numeric digunakan pada variabel atau konstanta untuk menyimpan nilai dalam bentuk angka. Tipe data ini terbagi atas integer, dan real.

* Integer

Integer Merupakan tipe data berupa bilangan bulat, terbagi atas beberapa kategori seperti table dibawah ini

|  |  |  |
| --- | --- | --- |
| **Tipe Data** | **Ukuran Tempat** | **Rentang Nilai** |
| **Byte** | 1 byte | 0 s/d +255 |
| **Shortint** | 1 byte | -28 s/d +127 |
| **Integer** | 2 bytes | -32768 s/d 32767 |
| **Word** | 2 bytes | 0 s/d 65535 |
| **Longint** | 4 bytes | 2147483648 s/d 2147483647 |

* Real

Real adalah bilangan yang berisi titik desimal atau jenis bilangan pecahan.

|  |  |  |
| --- | --- | --- |
| **Tipe Data** | **Ukuran Tempat** | **Rentang Nilai** |
| **real** | 6 bytes | 2.9 x 10-39 s/d 1.7 x1038 |
| **single** | 4 bytes | 1.5 x 1045 s/d 3.4 x 1038 |
| **double** | 8 bytes | 5.0 x 10-324 s/d 1.7 x 10308 |
| **extended** | 10 bytes | 3.4 x 10-4932 s/d 1.1 x 104932 |
| **comp** | 8 bytes | -9.2x 1018 s/d 9.2x 1018 |

* Karakter (char)

Karakter merupakan tipe data yang hanya mampu menyimpan 1 digit karakter.  Ukuran untuk tipe data karakter adalah 1 byte (1 byte = 8 bit). Adapun macam karakter yang ada sejumlah 256 macam karakter yaitu dari kode karakter (ASCII), 0 sampai dengan 255. Untuk penulisan karakter menggunakan tanda petik tunggal (‘ )  di depan dan belakang karakter yang ditulis. Contoh : ‘a’, ‘A’,’&’ dll.

Nilai-nilai yang termasuk karakter adalah :

* Karakter huruf : ‘a’..’z’,’A’..’Z’
* Karakter angka : ‘0’..’9’
* Karakter tanda baca : titik, koma, titik koma, titik dua dan sebagainya
* Karakter khusus : $, %, #, @ dan sebagainya.
* Boolean

Boolean merupakan tipe data logika, yang berisi dua kemungkinan nilai: TRUE (benar) atau FALSE (salah). Tipe data boolean memakai memori paling kecil.

* 1. Composite Data Types

Tipe Data Komposit merupakan tipe datayang dapat menampung banyak nilai, antara lain sebagai berikut.

* Array  
  Array atau sering disebut sebagai larik, adalah tipe data yang sudah terstruktur dengan baik, meskipun masih sederhana. Array mampu menyimpan sejumlah data dengan tipe yang sama (homogen) dalam sebuah variabel.
* Record atau struct

Record dikenal dalam bahasa Pascal/Delphi sedangkan Struct dikenal dalam bahasa C++. Berbeda dengan array, tipe data record mampu menampung banyak data dengan tipe data berbeda-beda (heterogen). Biasanya record digunakan untuk menampung data suatu obyek. Misalnya, siswa memiliki nama, alamat, usia, tempat lahir, dan tanggal lahir. Nama akan menggunakan tipe data string, alamat bertipe data string, usia bertipe data single (numeric), tempat lahir bertipe data string, dan tanggal lahir bertipe data date. Berikut ini contoh pengunaan record dalam Delphi.

* Image

Image, atau gambar, atau citra, merupakan tipe data grafik. Pada bahasa-bahasa pemrograman modern terutama yang berbasis visual, tipe data ini telah didukung dengan sangat baik.

* Date Time

Nilai data untuk tanggal (date) dan waktu (time) secara internal disimpan dalam format yang spesifik. Variabel atau konstanta yang dideklarasikan dengan tipe data Date dapat digunakan untuk menyimpan, baik tanggal maupun jam. Tipe data ini masuk dalam kelompok tipe data composite, karena merupakan bentukan dari beberapa tipe data.

* Object

Tipe data object digunakan untuk menyimpan nilai yang berhubungan dengan obyek-byek yang disediakan oleh Visual Basic, Delphi, dan bahasa pemrograman lain yang berbasis GUI. Sebagai contoh, apabila mempunyai form yang memiliki control Command button, yang kita beri nama Command1.

* Subrange

Tipe data subrange merupakan tipe data bilangan yang mempunyai jangkauan nilai tertentu sesuai dengan yang ditetapkan programmer. Biasanya, tipe data ini mempunyai nilai batas minimum dan nilai batas maksimum. Tipe data ini didukung dengan sangat baik dalam Delphi.

* Enumerasi

Tipe data ini merupakan tipe data yang mempunyai elemen-elemen yang harus disebut satu persatu, dan bernilai konstanta integer sesuai dengan urutannya. Nilai konstanta integer elemen ini diwakili oleh suatu nama variable yang ditulis di dalam kurung. Tipe data ini juga dijumpai pada Delphi, dan bahasa pemrograman deklaratif seperti SQL.

* 1. Other Data Types
* Tipe Data Terstruktur :
* Tipe Data String

Merupakan suatu data yang menyimpan array (larik), sebagai contoh 'ABCDEF' merupakan sebuah konstanta string yang berisikan 6 byte karakter. Ukuran Tempat untuk tipe data ini adalah 2 s/d 256 byte, dengan jumlah elemen 1 s/d 255.

* Tipe Data Set

Sebuah set merupakan suatu himpunan yang berisi nilai (anggota). Set merupakan Tipe data yang khusus untuk Pascal. Set dalam pemrograman sangat mirip dengan himpunan dalam ilmu matematika. Salah satu manfaat dari penggunaan tipe data set adalah untuk mengecek apakah suatu nilai muncul dalam suatu range tertentu. Misalnya, untuk menentukan apakah suatu karakter berupa Lower Case Letter (huruf kecil), mis. Ch adalah tipe Char, kita bisa menulis, if (Ch >= 'a') and (Ch <= 'z') then Writeln( Ch,' merupakan huruf kecil.'); atau, dengan notasi set, kita bisa menulis, if Ch in ['a'..'z'] then Writeln( Ch,' merupakan huruf kecil.');

* Tipe Data Pointer

Pointer merupakan variabel khusus yang berisi suatu address (alamat) di lokasi lain didalam memori. Suatu variabel yang points (menunjuk) ke sesuatu sehingga disebut pointer.

Ada dua macam pointer:

* Typed (tertentu) : merupakan pointer yang menunjuk pada tipe data tertentu pada variabel.
* Generic (umum) : merupakan pointer yang tidak menunjuk pada tipe data tertentu pada variabel.
  1. Data Types in HTML
* Case

Setiap definisi atribut berisi informasi tentang case-sensitivity dan nilai-nilainya. Informasi kasus disajikan dengan tombol berikut:

* CS

Nilai adalah case-sensitive (yaitu, agen pengguna menafsirkan "a" dan "A" berbeda).

* CI

Nilai adalah case- insensitive(misalnya, agen pengguna menafsirkan "a" dan "A" sebagai yang sama).

* CN

Nilai tidak mengalami perubahan case, misalnya, karena merupakan angka atau karakter dari set karakter dokumen.

* CA

Unsur atau atribut definisi sendiri memberikan informasi case.

* CT

Konsultasikan definisi jenis untuk rincian tentang case-sensitivity

Jika nilai atribut adalah daftar, tombol berlaku untuk setiap nilai dalam daftar, kecuali dinyatakan lain.

* Tipe dasar SGML

Definisi tipe dokumen menentukan sintaks HTML konten elemen dan atribut nilai menggunakan SGML token (misalnya, PCDATA, CDATA, NAMA, ID, dll)

CDATA adalah urutan karakter dari set karakter dokumen dan mungkin termasuk entitas karakter. Agen pengguna harus menginterpretasikan nilai atribut sebagai berikut:

* Ganti entitas karakter dengan karakter,
* Mengabaikan baris feed,
* Mengganti setiap carriage return atau tab dengan spasi tunggal.

Agen pengguna dapat mengabaikan terkemuka dan trailing ruang putih di CDATA nilai atribut (misalnya, "myval" dapat ditafsirkan sebagai "myval"). Penulis harus tidak menyatakan nilai atribut dengan memimpin atau tertinggal spasi.

Untuk beberapa HTML 4 atribut dengan CDATA nilai atribut, spesifikasi membebankan kendala lanjut pada set nilai-nilai hukum untuk atribut yang mungkin tidak diungkapkan oleh DTD.

Meskipun STYLE dan elemen SCRIPT menggunakan CDATA untuk model data mereka, untuk unsur-unsur ini, CDATA harus ditangani secara berbeda oleh agen pengguna. Markup dan entitas harus diperlakukan sebagai teks baku dan diteruskan ke aplikasi seperti. Kejadian pertama dari urutan karakter "</" (end-tag terbuka pembatas) diperlakukan sebagai mengakhiri akhir konten elemen. Dalam dokumen yang sah, ini akan menjadi tag akhir untuk elemen.

* ID dan NAMA token harus dimulai dengan huruf ([A-Za-z]) dan dapat diikuti oleh sejumlah huruf, angka ([0-9]), tanda hubung ( "-"), garis bawah ( "\_") , titik dua ( ":"), dan periode ( ".").
* IDREF dan IDREFS adalah referensi untuk token ID didefinisikan oleh atribut lainnya. IDREF adalah tanda tunggal dan IDREFS adalah daftar dipisahkan dengan spasi dari token.
* NOMOR token harus mengandung setidaknya satu digit ([0-9]).
* Warna

Nilai atribut tipe "color" (%color;) mengacu pada definisi warna. Sebuah nilai warna dapat berupa angka heksadesimal (diawali dengan tanda pagar) atau salah satu nama warna enam belas berikut. Nama-nama warna case-sensitive.

|  |  |  |  |
| --- | --- | --- | --- |
| Color names and sRGB values | | | |
| Description: https://www.w3.org/TR/html4/images/black.gif | Black = "#000000" | Description: https://www.w3.org/TR/html4/images/green.gif | Green = "#008000" |
| Description: https://www.w3.org/TR/html4/images/silver.gif | Silver = "#C0C0C0" | Description: https://www.w3.org/TR/html4/images/lime.gif | Lime = "#00FF00" |
| Description: https://www.w3.org/TR/html4/images/gray.gif | Gray = "#808080" | Description: https://www.w3.org/TR/html4/images/olive.gif | Olive = "#808000" |
| Description: https://www.w3.org/TR/html4/images/white.gif | White = "#FFFFFF" | Description: https://www.w3.org/TR/html4/images/yellow.gif | Yellow = "#FFFF00" |
| Description: https://www.w3.org/TR/html4/images/maroon.gif | Maroon = "#800000" | Description: https://www.w3.org/TR/html4/images/navy.gif | Navy = "#000080" |
| Description: https://www.w3.org/TR/html4/images/red.gif | Red = "#FF0000" | Description: https://www.w3.org/TR/html4/images/blue.gif | Blue = "#0000FF" |
| Description: https://www.w3.org/TR/html4/images/purple.gif | Purple = "#800080" | Description: https://www.w3.org/TR/html4/images/teal.gif | Teal = "#008080" |
| Description: https://www.w3.org/TR/html4/images/fuchsia.gif | Fuchsia = "#FF00FF" | Description: https://www.w3.org/TR/html4/images/aqua.gif | Aqua = "#00FFFF" |

Dengan demikian, nilai warna "# 800080" dan "Purple" keduanya merujuk pada warna ungu

* Length

HTML menentukan tiga jenis nilai panjang untuk atribut:

* Pixel: Nilai (% Pixel, dalam DTD yang) adalah bilangan bulat yang mewakili jumlah pixel dari kanvas (layar, kertas). Dengan demikian, nilai "50" berarti lima puluh piksel.
* Length: Nilai (%Length, dalam DTD) bisa berupa% Pixel; atau persentase dari ruang horisontal atau vertikal tersedia. Dengan demikian, nilai "50%" berarti setengah dari ruang yang tersedia.
* MultiLength: Nilai (% MultiLength, dalam DTD) bisa menjadi% Panjang; atau panjang relatif. Sebuah panjang relatif memiliki bentuk "i \*", di mana "i" adalah bilangan bulat. Ketika allotting ruang antara unsur-unsur bersaing untuk ruang itu, agen pengguna membagikan pixel dan persentase panjang pertama, kemudian membagi sisa ruang yang tersedia antara relatif panjang. Setiap panjang relatif menerima sebagian dari ruang yang tersedia yang sebanding dengan bilangan bulat mendahului "\*". Nilai "\*" setara dengan "1 \*". Dengan demikian, jika 60 piksel ruang yang tersedia setelah agen pengguna menganggarkan pixel dan persentase ruang, dan panjang relatif bersaing adalah 1 \*, 2 \*, dan 3 \*, 1 \* akan dibagikan 10 piksel, 2 \* akan dibagikan 20 piksel, dan 3 \* akan dibagikan 30 piksel.

Nilai panjang adalah kasus-netral.

* Tanggal dan Waktu

Formatnya adalah:

YYYY-MM-DDThh: mm: ssTZD

dimana:

      YYYY = empat digit tahun

      MM = dua digit bulan (01 = Januari, dll)

      DD = hari dua digit bulan (01 sampai 31)

      hh = dua digit dari jam (00 sampai 23) (am / pm tidak diperbolehkan)

      mm = dua digit menit (00 sampai 59)

      ss = dua digit kedua (00 sampai 59)

      TZD = penunjuk zona waktu

* Tipe Link

Penulis dapat menggunakan tipe link diakui berikut, tercantum di sini dengan interpretasi konvensional mereka. Dalam DTD,% LinkTypes mengacu pada daftar dipisahkan dengan spasi dari jenis link yang. karakter spasi tidak diperbolehkan dalam tipe link.

tipe link ini case-sensitive, yaitu, "Alternatif" memiliki arti yang sama dengan "alternatif".

agen pengguna, mesin pencari, dll bisa menafsirkan jenis ini link dalam berbagai cara. Misalnya, agen pengguna dapat memberikan akses ke dokumen terkait melalui bar navigasi.

* Alternate

Menunjuk versi pengganti dokumen yang link terjadi. Ketika digunakan bersama-sama dengan atribut lang, itu berarti versi terjemahan dokumen. Ketika digunakan bersama-sama dengan atribut media, itu berarti versi dirancang untuk media yang berbeda (atau media).

* Stylesheet

Mengacu pada style sheet eksternal. Lihat bagian style sheet eksternal untuk rincian. Ini digunakan bersama-sama dengan jenis link "Alternatif" untuk dipilih pengguna style sheet alternatif.

* Start

Mengacu pada dokumen pertama dalam koleksi dokumen. Jenis link ini memberitahu mesin pencari yang dokumen dianggap oleh penulis untuk menjadi titik awal dari koleksi.

* Next

Mengacu pada dokumen berikutnya dalam urutan linear dokumen. Agen pengguna dapat memilih untuk preload "berikutnya" dokumen, untuk mengurangi waktu beban yang dirasakan.

* Prev

Mengacu pada dokumen sebelumnya dalam seri memerintahkan dokumen. Beberapa agen pengguna juga mendukung sinonim "Sebelumnya".

* Content

Mengacu pada dokumen yang berfungsi sebagai daftar isi. Beberapa agen pengguna juga mendukung TOC sinonim (dari "Daftar Isi").

* Indeks

Mengacu pada dokumen menyediakan indeks untuk dokumen saat ini.

* Glossary

Mengacu pada dokumen menyediakan daftar istilah yang merujuk ke dokumen saat ini.

* Copyright

Mengacu pada pernyataan hak cipta untuk dokumen saat ini.

* Chapter

Mengacu pada dokumen melayani sebagai bab dalam koleksi dokumen.

* Section

Mengacu pada dokumen melayani sebagai bagian dalam koleksi dokumen.

* Subsection

Mengacu pada dokumen melayani sebagai ayat dalam koleksi dokumen.

* Appendix

Mengacu pada dokumen yang melayani sebagai lampiran dalam koleksi dokumen.

* Help

Mengacu pada dokumen penawaran bantuan (informasi lebih lanjut, link ke informasi sumber-sumber lain, dll)

* Bookmark

Mengacu bookmark. Sebuah bookmark adalah link ke titik masuk utama dalam dokumen diperpanjang. Atribut judul dapat digunakan, misalnya, untuk label penanda. Perhatikan bahwa beberapa bookmark dapat didefinisikan dalam setiap dokumen.

Penulis mungkin ingin menentukan jenis link tambahan yang tidak dijelaskan dalam spesifikasi ini. Jika mereka melakukannya, mereka harus menggunakan profil untuk mengutip konvensi yang digunakan untuk menentukan tipe link.

* Media Deskripsi

Berikut ini adalah daftar deskriptor media yang diakui (% MediaDesc di DTD) :

* Screen

Ditujukan untuk layar komputer non-paged.

* Tty

Ditujukan untuk media menggunakan karakter jaringan fixed-pitch, seperti teletypes, terminal, atau perangkat portable dengan kemampuan layar yang terbatas.

* Tv

Ditujukan untuk perangkat televisi tipe (resolusi rendah, warna, scrollability terbatas).

* Projection

Ditujukan untuk proyektor.

* Handheld

Ditujukan untuk perangkat genggam (layar kecil, monokrom, grafis bitmap, bandwidth terbatas).

* Print

Ditujukan untuk paged, bahan buram dan untuk dokumen dilihat pada layar dalam mode print preview.

* Braille

Ditujukan untuk perangkat umpan balik taktil braille.

* Aural

Ditujukan untuk synthesizer pidato.

* All

Cocok untuk semua perangkat

Versi mendatang dari HTML dapat memperkenalkan nilai-nilai baru dan memungkinkan nilai-nilai parameter. Untuk memudahkan pengenalan ekstensi ini, sesuai agen pengguna harus dapat mengurai nilai atribut media sebagai berikut:

1. EXPRESSION & ASSIGNMENT STATEMENTS
   1. Expression

Mendeskripsikan mengenai komputasi di dalam suatu bahasa pemrograman, untuk mengerti lebih jauh terdapat istilah operator dan operand di dalam suatu ekspresi.

A+B=C;

* “+, =” adalah operator.
* “A, B, C” adalah operand.

* 1. Arithmetic Expression

Di dalam ekpresi aritmatika terdiri dari operato-operator, dam operand.

* Unary Operator : Mempunyai 1 operand.
* Binary Operator : Mempunyai 2 operand.
* Ternary Operator : Mempunyai lebih dari 2 operand.

Typical Precedence Level dari ekpresi aritmatika adalah :

* Parentheses.
* Unary Operator.
* \*\*(Jika bahasa yang digunakan mendukung).
* \*, /.
* +,-.
  1. Relational and Boolean Expression
* Relational Expression
* Menggunakan relational operator dan operand.
* Merepresentasikan Boolean.
* Operator yang digunakan (!=, /=, ~=, .NE., <>, #).
  1. Expression & Assignment Statements in Javascript
* Operator

Javascript mempunyai operator yang hampir sama seperti pada bahasa C, dan bahasa-bahasa pemrograman lainnya. Yaitu :

* +
* -
* \*
* /

|  |  |  |
| --- | --- | --- |
| Nama | Operator | Arti |
| [Assignment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Assignment_Operators#Assignment) | x = y | x = y |
| [Addition assignment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Assignment_Operators#Addition_assignment) | x += y | x = x + y |
| [Subtraction assignment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Assignment_Operators#Subtraction_assignment) | x -= y | x = x - y |
| [Multiplication assignment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Assignment_Operators#Multiplication_assignment) | x \*= y | x = x \* y |
| [Division assignment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Assignment_Operators#Division_assignment) | x /= y | x = x / y |
| [Remainder assignment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Assignment_Operators#Remainder_assignment) | x %= y | x = x % y |
| [Exponentiation assignment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Assignment_Operators#Exponentiation_assignment) | x \*\*= y | x = x \*\* y |
| [Left shift assignment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Assignment_Operators#Left_shift_assignment) | x <<= y | x = x << y |
| [Right shift assignment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Assignment_Operators#Right_shift_assignment) | x >>= y | x = x >> y |
| [Unsigned right shift assignment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Assignment_Operators#Unsigned_right_shift_assignment) | x >>>= y | x = x >>> y |
| [Bitwise AND assignment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Assignment_Operators#Bitwise_AND_assignment) | x &= y | x = x & y |
| [Bitwise XOR assignment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Assignment_Operators#Bitwise_XOR_assignment) | x ^= y | x = x ^ y |
| [Bitwise OR assignment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Assignment_Operators#Bitwise_OR_assignment) | x |= y | x = x | y |

* Assigment Operators

Melakukan kalkulasi antara nilai pada operand kiri dengan operand yang berada di sebelah kanan.

* Comparison Operators

Melakukan perbandingan diantara beberapa operand dan mengembalikan nilai berupa nilai logika, bergantung nilainya apakah bernilai true atau false.

|  |  |  |
| --- | --- | --- |
| Operator | Penjelasan | Contoh True |
| [Equal](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Equality) (==) | Mengembalikan nilai true jika kedua operand sama | 3 == var1  "3" == var1  3 == '3' |
| [Not equal](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Inequality) (!=) | True jika kedua operand tidak sama | var1 != 4 var2 != "3" |
| [Strict equal](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Identity) (===) | Mengembalikan true jika kedua operand sama dan pada tipe yang sama | 3 === var1 |
| [Strict not equal](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Nonidentity) (!==) | Mengemablikan nilai true jika kedua operand pada tipe yang sama, namun tidak bernilai sama | var1 !== "3" 3 !== '3' |
| [Greater than](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Greater_than_operator) (>) | Mengembalikan nilai true jika operand kiri lebih besar dari operand kanan | var2 > var1 "12" > 2 |
| [Greater than or equal](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Greater_than_or_equal_operator) (>=) | Mengembalikan nilai true jika operand kiri lebih besar dari atau sama dengan operand kanan | var2 >= var1 var1 >= 3 |
| [Less than](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Less_than_operator) (<) | Mengembalikan nilai true jika operand kiri lebih kecil dari operand kanan | var1 < var2 "2" < 12 |
| [Less than or equal](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Less_than_or_equal_operator) (<=) | Mengembalikan nilai true jika operand kiri lebih kecil dari atau sama dengan operand kanan | var1 <= var2 var2 <= 5 |

* Arithmetic Operator

Melakukan kalkulasi seusai dengan nilai masing-masing operand yang dihubungkan oleh sebuah operator.

|  |  |  |
| --- | --- | --- |
|  | | |
| Operator | Deskripsi | Contoh |
| [Remainder](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Remainder) (%) | Sisa Pembagian | 12 % 5 returns 2. |
| [Increment](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Increment) (++) | Penambahan nilai, setara dengan +1 | If x is 3, then ++x sets x to 4 and returns 4, whereas x++ returns 3 and, only then, sets xto 4. |
| [Decrement](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Decrement) (--) | Pengurangan nilai setara dengan -1 | If x is 3, then --x sets x to 2 and returns 2, whereas x-- returns 3 and, only then, sets xto 2. |
| [Unary negation](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Unary_negation) (-) | Negasi | If x is 3, then -xreturns -3. |
| [Unary plus](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Unary_plus) (+) | Tambah | +"3" returns 3. +true returns 1. |
| [Exponentiation operator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Arithmetic_Operators#Exponentiation) (\*\*) | Eksponen | 2 \*\* 3 returns 8. 10 \*\* -1 returns 0.1. |

* Bitwise Operators

Melakukan perbandingan nilai untuk angka binary.

|  |  |  |
| --- | --- | --- |
| Operator | Penggunaan | Deskripsi |
| [Bitwise AND](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators#Bitwise_AND) | a & b | Returns a one in each bit position for which the corresponding bits of both operands are ones. |
| [Bitwise OR](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators#Bitwise_OR) | a | b | Returns a zero in each bit position for which the corresponding bits of both operands are zeros. |
| [Bitwise XOR](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators#Bitwise_XOR) | a ^ b | Returns a zero in each bit position for which the corresponding bits are the same. [Returns a one in each bit position for which the corresponding bits are different.] |
| [Bitwise NOT](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators#Bitwise_NOT) | ~ a | Inverts the bits of its operand. |
| [Left shift](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators#Left_shift) | a << b | Shifts a in binary representation b bits to the left, shifting in zeros from the right. |
| [Sign-propagating right shift](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators#Right_shift) | a >> b | Shifts a in binary representation b bits to the right, discarding bits shifted off. |
| [Zero-fill right shift](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Bitwise_Operators#Unsigned_right_shift) | a >>> b | Shifts a in binary representation b bits to the right, discarding bits shifted off, and shifting in zeros from the left. |

* Logical Operators

Nilai Boolean, mengembalikan nilai Boolean.

|  |  |  |
| --- | --- | --- |
| Operator | Penggunaan | Deskripsi |
| [Logical AND](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Logical_Operators#Logical_AND)(&&) | expr1 && expr2 | Jika semua bernilai True maka akan mengembalikan nilai True |
| [Logical OR](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Logical_Operators#Logical_OR)(||) | expr1 || expr2 | Jika salah satu bernilai True maka akan mengembalikan nilai True |
| [Logical NOT](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Logical_Operators#Logical_NOT)(!) | !expr | Memberikan nilai yang dinegasikan, ex: True menjadi False |

1. CONTROL STUCTURE STATEMENTS

* 1. Control Structure

Struktur kontrol adalah pernyataan kontrol dan laporan yang eksekusi itu control. Structur kontrol terbagi menjadi dua, yaitu Selection Statements dan Iterative Statements.

* Selection Statements

Sebuah pernyataan pilihan menyediakan sarana untuk memilih antara dua atau lebih jalur eksekusi

Dua kategori umum:

* Two-way selectors (If, Then, Else).
* Multiple-way selectors (Switch).
* Iterative Statements

Eksekusi berulang pernyataan atau senyawa pernyataan dicapai baik oleh iterasi atau rekursi. Kontrol loop penghitung, Sebuah pernyataan berulang menghitung memiliki variabel loop, dan sarana menentukan nilai awal dan terminal, dan stepsize.Counter Control Loop

* Syntax
* **for** ([expr\_1] ; [expr\_2] ; [expr\_3]) statement
* Expression pertama adalah inisialisasi.
* Expression kedua tidak boleh dikosongkan, kalau tidak akan menjadi *infinite loop*/loop terus menerus.
* Expression ketiga adalah increment(++) atau decrement(--)
* Umumny ketiga expression ini bertipe data integer.
  1. Control Structures in HTML

Jika Anda ingin halaman web Anda untuk melakukan hal yang berbeda tergantung pada situasi, Anda akan memerlukan bahasa pemrograman. Sebagai contoh, beberapa situs ingin memberikan fasilitas keanggotaan di mana orang dapat login ke situs, dan mengakses informasi tertentu. Situs lain menyediakan formulir umpan balik sehingga pengunjung dapat menghubungi mereka. Semua hal ini memerlukan fasilitas yang format dokumen sederhana tidak bisa lakukan.

JavaScript adalah salah satu bahasa pemrograman yang paling umum digunakan di Internet. Mereka digunakan oleh situs web untuk melakukan operasi lebih rumit. Dalam hal ini, kita akan membutuhkan JavaScript untuk membantu kita untuk memahami Kontrol Struktur di HTML.

* JavaScript
* Selections Statements
* IF

If statement akan dijalankan jika kondisi yang dimisalkan bernilai TRUE, jika bernilai FALSE maka tidak akan dijalankan atau akan menjalankan statement else.

if( Condition ) {

statement 1;

} else {

Statement 2;

}

* SWITCH

Switch statement adalah pernyataan yang akan melihat nilai suatu variable dan akan mengeksekusi statement tertentu sesuai dengan “CASE” nilai variable yang disediakan sampai break statement, jika tidak ada maka switch statement akan menjalankan default case atau tidak menjalankan program.

switch(myVar) {

case 1:

break;

case 2:

break;

default:

* Iterative Statements
* FOR

Pada perulangan FOR, inisialisasi variabel, syarat dan operasi bilangan ditulis dalam satu kelompok dan terpisah dari statemen program yang akan dijalankan. Statemen program akan dijalankan berulang selama kondisi syarat masih terpenuhi atau bernilai benar. Perulangan FOR dilakukan untuk meringkas penulisan perulangan menggunakan WHILE bilamana telah diketahui atau ditentukan jumlah perulangannnya.

for( starting\_initialise; continue\_as\_long\_as\_condition; do\_this\_each\_time )

Statement for berisikan 3 parameter, insialisasi yaitu menentukan nilai sebuah variable. Tes kondisi nilai variable, apakah masi memenuhi syarat atau tidak. Dan yang terakhir adalah operasi perubahan nilai, berisikan increment(++) atau decrement (--).

* WHILE

Pada perulangan menggunakan WHILE, statemen program juga akan dijalankan secara berulang selama kondisi syarat pada WHILE masih bernilai benar. Perulangan akan berhenti jika syarat pada WHILE bernilai salah.

var myVariable = 1;

while( myVariable <= 5 ) {

myArray[myVariable] = 1;

myVariable++;

}

* DO WHILE

Perulangan WHILE memiliki cara kerja yang mirip dengan DO WHILE, bedanya jika pada perulangan DO WHILE pengecekan kondisi dilakukan dibelakang maka pada WHILE pengecekan kondisi syarat dilakukan didepan. Pada perulangan menggunakan DO WHILE, pernyataan atau statemen program setidaknya akan dijalankan selama satu kali sedangkan jika menggunakan WHILE, pernyataan atau statemen program bisa saja tidak dijalankan jika saat pengecekan kondisi sudah bernilai salah. untuk lebih jelas bisa dilihat pada program berikut ini.

var myVariable = 1;

do {

myArray[myVariable] = 1;

myVariable++;

} while( myVariable <= 5 );

1. SUBPROGRAMS
   1. Function

* Fungsi adalah sub program yang mengembalikan (return) sebuah nilai bertipe data primitif (int, float, double, boolean, string, atau char)
* Mirip dengan fungsi dalam matematika yang tugasnya mengubah nilai input menjadi nilai output
* Umumnya fungsi disertai penggunaan parameter untuk input data

Fungsiadalah suatu bagian dari program yang dimaksudkan untuk mengerjakan suatu tugas tertentu dan letaknya dipisahkan dari bagian program yang menggunakannya. Fungsi digunakan agar program lebih mudah di perbaiki jika ada kesalahan dan mudah untuk di panggil berulang kali.

Fungsi berisi dengan statemen-statemen yang akan melakukan tugas yang diberikan kepada fungsi bersangkutan yang ditulis di dalam tanda kurung kurawal.

Bentuk umum dari suatu fungsi adalah :  
  
Tipe\_hasil nama\_fungsi (argumen1,argumen2,...)

{  
deklarasi;  
statemen-statemen;  
}

* 1. Procedure
* Prosedur adalah program yang mengerjakan aktivitas yang spesifik yang menghasilkan efek netto
* Efek netto yaitu perubahan kondisi awal menjadi kondisi akhir setelah prosedur dijalankan
* Prosedur bukan program yang berdiri sendiri , sehingga untuk menjalankan prosedur dengan dipanggil di program utama.

Prosedur adalah kumpulan ekspresi-ekspresi algoritma yang berguna untuk menjalankan proses tertentu. Dalam bahasa Java prosedur biasanya diawali dengan kata “void”. Dan kebanyakan aplikasi berjalan melalui prosedur.

* 1. Parameter

Fungsi memerlukan masukan yang dinamakan parameter, yang akan diolah oleh fungsi. Hasil akhir dari sebuah fungsi adalah sebuah nilai berupa nilai keluaran fungsi. Suatu fungsi secara umum terdiri dari dua buah komponen utama, yaitu definisi fungsi dan tubuh fungsi. Definisi fungsi berisi dengan tipe dari fungsi, nama dari fungsi dan argumen-argumennya jika digunakan.

* Parameter dapat digunakan baik di fungsi maupun prosedur. Ada 2 jenis parameter yaitu parameter aktual dan parameter formal.
* Parameter aktual digunakan saat memanggil prosedur, melalui parameter ini data dikirim baik berupa konstanta atau variabel.
* Parameter formal digunakan pada saat mendeklarasikan nama prosedur, melalui parameter ini data diterima dari parameter aktual, selalu berupa variabel, melakukan casting otomatis dengan urutan byte → short → int → long →float → double.
  1. Global & Local Identifier
* Konsekuensi penggunaan sub program adalah adanya identifier (variabel atau konstanta) global dan lokal
* Identifier global adalah identifier yang dikenali di semua bagian program, pada program diatas contohnya adalah nilaiAkhir dan nilaiHuruf
* Identifier lokal adalah identifier yang hanya dikenali oleh sub program yang mendeklarasikannya, contohnya adalah uts, uas dan tugas yang hanya dikenali oleh prosedur hitungNilai.
  1. Function vs Procedure

Perbedaan antara function dan procedure adalah : function akan mengembalikan suatu nilai pada pemanggilnya, sedangkan procedure tidak akan mengembalikan nilai apapun pada fungsi pemanggilnya. Untuk lebih jelasnya berikut contoh studi kasus program untuk menghitung luas persegi panjang dalam bahasa Java. Berikut ini merupakan source code procedure yang digunakan untuk menghitung luas persegi panjang

public **void** HitungLuasPersegiPanjang(int p, int l)  
{  
int luasPersegiPanjang;

luasPersegiPanjang = p \* l;

**System.out.println(“Luas persegi panjang = ” + ” ” + luasPersegiPanjang);**   
}

Berikut ini merupakan source code function untuk menghitung luas persegi panjang

public **int** HitungLuasPersegiPanjang(int p, int l)  
{  
int luasPersegiPanjang;

luasPersegiPanjang = p \* l;  
**return** luasPersegiPanjang;  
}

* **Cara memanggil procedure**

HitungLuasPersegiPanjang(p, l);

* **Cara memanggil function**

luas = HitungLuasPersegiPanjang(p, l);

1. ABSTRACT DATA TYPES

* 1. Abstract Data Types

Sebuah program dapat dikatakan baik apabila menggunakan abstraksi. Abstract Data Types atau lebih sering disebut dengan ADT, membagi ide-ide atau gagasan dari sebuh spesifikasi (Dengan apa yang sedang kita kerjakan dan dengan operasi seperti apa nantinya akan dijalankan) dan dengan impelementasi (Bagaimana hal yang sedang dikerjakan nantinya akan diimpelementasikan).

* Keuntungan menggunakan ADT :
* Kode menjadi lebih mudah untuk dimengerti.
* Implementasi dari ADT dapat diubah (untuk mendapatkan efisiensi yang lebih) tanpa harus mengubah ke program yang menggunakan ADT.
* ADT dapat digunakan kembali untuk program yang selanjutnya.

Bahasa pemrograman yang berbasis Object Oriented Programming (contohnya Java) memudahkan programmer untuk menggunakan ADT: Setiap ADT berkaitan dengan class (dalam Java) dan operasi pada ADT adalah kelas/tampilan dari public method. Sehingga pengguna ataupun klien hanya harus mengetahui tampilan dari methodnya saja, kemudian bukan merupakan implementasi yang sesungguhnya.

Dalam ADT terdapat dua bagian dari ADT :

* Bagian eksternal/public, yang terdiri dari :
* Gambaran konsep (Pandangan dari pengguna bagaimana sebuah objek terlihat, dan bagaimana strukturnya tersusun).
* Konsep dari operasi (Apa yang pengguna dapat lakukan terhadap ADT).
* Bagian Internal/private :
* Representasi (Bagaimana strukturnya disimpan).
* Impelemntasi dari operasi (Kode yang sesungguhnya).

Pengelompokan operasi pada ADT :

* Initialize.
* Add data.
* Access data.
* Remove data.
  1. Abstract Data Types in Java

Dalam bahasa pemrograman Java, semua tipe data user defined adalah class, Class adalah notasi yang digunakan oleh object oriented programming untuk menggambarkan tampilan dan fungsionalitas dari sebuh obyek yang dimanipulasi. ADT pada java di deskripsikan oleh lebih dari satu class, tetapi tidak semua class merupakan ADT, tetapi semua ADT diimplementasikan sebagai class.

* Perbedaaan antara Absract dan Interface
* Perbedaan yang paling utama adalah metode interface Java secara implisit abstrak dan tidak dapat memiliki implementasi. Sebuah abstract class Java  dapat memiliki sebuah metode contoh yang mengimplementasikan perilaku secara default.
* Variabel yang dideklarasikan dalam interface Java adalah dengan akhir default. Abstract class dapat berisi variabel non-final.
* Anggota interface Java bersifat publik secara default. Sebuah abstract class Java dapat memiliki fungsi biasa sebagai anggota kelas seperti swasta, dilindungi, dll ..
* Interface Java harus dilaksanakan dengan kata kunci "implements"; Sebuah abstract class Java harus diperluas dengan kata kunci "extend".
* Sebuah interface dapat memperpanjang interface Java lain saja, abstract class dapat memperpanjang class Java lain dan mengimplementasikan beberapa interface Java.
* Sebuah class  Java dapat mengimplementasikan beberapa interface tetapi dapat memperpanjang hanya satu abstract class.
* Interface yang abstract dan tidak dapat dipakai; Sebuah class abstract Java juga tidak bisa dipakai, tapi dapat dipanggil jika terdapat main ().
* Dibandingkan dengan abstract class  java, java interface lambat karena membutuhkan pengalokasian ekstra.

Interface berbeda dari kelas abstrak karena sebuah interface bukan kelas. Sebuah interface pada dasarnya merupakan jenis yang dapat dipenuhi oleh setiap class yang mengimplementasikan interface.

Setiap class yang mengimplementasikan interface harus memenuhi 2 syarat:

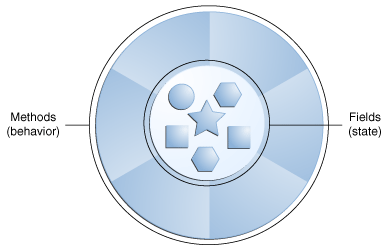
* Harus memiliki frase "implement interface\_name" pada awal class definiton.
* Harus mengimplementasikan semua judul metode yang tercantum dalam definisi interface.

1. OBJECT ORIENTED PROGRAMMING (OOP)

* 1. Object Oriented Programming (OOP)

Objek dunia nyata berbagi dua karakteristik: mereka semua memiliki bagian dan perilaku. Anjing memiliki bagian (nama, warna, berkembang biak, lapar) dan perilaku (menggonggong, mengambil, mengibaskan ekor). Sepeda juga memiliki bagian (gigi saat ini, saat pedal irama, kecepatan arus) dan perilaku (mengubah gigi, mengubah pedal irama, menerapkan rem). Mengidentifikasi bagian dan perilaku untuk objek dunia nyata adalah cara yang bagus untuk mulai berpikir dalam hal pemrograman berorientasi objek.

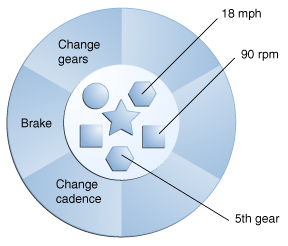
Luangkan waktu sebentar sekarang untuk mengamati benda dunia nyata yang berada di daerah Anda. Pastikan untuk menuliskan pengamatan Anda. Seperti yang Anda lakukan, Anda akan melihat bahwa benda dunia nyata bervariasi dalam kompleksitas; lampu desktop Anda mungkin hanya memiliki dua kemungkinan bagian (on dan off) dan dua kemungkinan perilaku (menghidupkan, mematikan), tetapi radio desktop Anda mungkin memiliki bagian tambahan (on, off, volume saat ini, stasiun saat ini) dan perilaku (hidupkan , matikan, meningkatkan volume, mengurangi volume, mencari, memindai, dan lagu). Anda juga mungkin melihat bahwa beberapa benda, pada gilirannya, akan juga berisi objek lain. pengamatan dunia nyata ini semua diterjemahkan ke dalam dunia pemrograman berorientasi objek.



A software object.

Software objek secara konseptual mirip dengan benda dunia nyata: mereka juga terdiri dari perilaku bagian dan terkait. Sebuah objek menyimpan bagian dalam bidang (variabel dalam beberapa bahasa pemrograman) dan mengekspos perilakunya melalui metode (fungsi dalam beberapa bahasa pemrograman). Metode beroperasi pada keadaan internal suatu objek dan berfungsi sebagai mekanisme utama untuk komunikasi “*object-to-object”*. Menyembunyikan keadaan internal dan mengharuskan semua interaksi yang akan dilakukan melalui metode obyek dikenal sebagai enkapsulasi Data - prinsip dasar pemrograman berorientasi objek.

Pertimbangkan sepeda, misalnya:



Sebuah sepeda dimodelkan sebagai objek perangkat lunak.

Dengan menghubungkan bagian (kecepatan saat ini, saat pedal irama, dan gigi saat ini) dan menyediakan metode untuk mengubah negara itu, objek tetap mengendalikan bagaimana dunia luar diperbolehkan untuk menggunakannya. Misalnya, jika sepeda hanya memiliki 6 gigi, metode untuk mengubah gigi bisa menolak setiap nilai yang kurang dari 1 atau lebih besar dari 6.

Bundling kode ke objek perangkat lunak individu menyediakan sejumlah manfaat, termasuk:

* Modularity: Kode sumber untuk suatu benda dapat ditulis dan dipelihara secara independen dari kode sumber untuk benda-benda lain. Setelah dibuat, sebuah objek dapat dengan mudah melewati sekitar di dalam sistem.
* Informasi-bersembunyi: Dengan berinteraksi hanya dengan metode obyek, rincian implementasi internal tetap tersembunyi dari dunia luar.
* Kode digunakan kembali: Jika suatu benda sudah ada (mungkin ditulis oleh pengembang perangkat lunak lain), Anda dapat menggunakan objek yang dalam program Anda. Hal ini memungkinkan spesialis untuk melaksanakan / test / debug kompleks, benda tugas-spesifik, yang kemudian dapat percaya untuk menjalankan kode Anda sendiri.
* Pluggability dan debugging kemudahan: Jika objek tertentu ternyata bermasalah, Anda hanya dapat menghapusnya dari aplikasi Anda dan pasang di objek yang berbeda sebagai penggantinya. Hal ini analog dengan memperbaiki masalah mekanis di dunia nyata. Jika baut istirahat, Anda menggantinya, tidak seluruh mesin.

1. CONCURRENCY
   1. Concurrency

Dalam concurrent program, beberapa aliran operasi dapat mengeksekusi secara bersamaan. Setiap aliran operasi mengeksekusi karena akan di program sekuensial kecuali fakta bahwa aliran dapat berkomunikasi dan mengganggu satu sama lain. Setiap urutan seperti instruksi disebut thread. Untuk alasan ini, program sekuensial sering disebut program single-threaded. Ketika program multi-threaded mengeksekusi, operasi di berbagai benang naannya disisipkan dalam subjek agar tak terduga untuk kendala yang dikenakan oleh operasi sinkronisasi eksplisit yang mungkin tertanam dalam kode. Operasi untuk setiap aliran secara ketat memerintahkan, tetapi interleaving operasi dari koleksi stream adalah belum ditentukan dan tergantung pada liku-liku dari pelaksanaan khusus dari program ini. Satu aliran dapat berjalan sangat cepat sementara yang lain tidak berjalan sama sekali. Dengan tidak adanya jaminan keadilan (dibahas di bawah), thread yang diberikan bisa kelaparan kecuali hanya `` runnable '' benang.

Sebuah benang runnable kecuali mengeksekusi operasi yang membutuhkan sinkronisasi khusus yang menunggu sampai kondisi tertentu terjadi. Jika lebih dari satu thread yang runnable, semua tapi satu thread mungkin kelaparan (tidak membuat kemajuan karena tidak ada operasi yang sedang dijalankan) kecuali bahasa membuat jaminan keadilan. Sebuah jaminan keadilan menyatakan bahwa operasi berikutnya di thread runnable akhirnya akan mengeksekusi. Java spesifikasi bahasa saat ini tidak memberikan jaminan keadilan tetapi kebanyakan *Java Virtual Machines* menjamin keadilan.

Threads dapat berkomunikasi satu sama lain dalam berbagai cara yang akan kita bahas secara rinci nanti dalam bagian ini. Bahasa pemrograman Java terutama bergantung pada variabel bersama untuk mendukung komunikasi antara proses, tetapi juga mendukung mekanisme signaling eksplisit.

Secara umum, menulis program bersamaan sangat sulit karena banyaknya kemungkinan interleavings operasi antara benang berarti eksekusi program adalah non-deterministik. Untuk alasan ini, bug Program mungkin sulit untuk mereproduksi. Selain itu, kompleksitas diperkenalkan oleh beberapa thread dan interaksi potensi mereka membuat program lebih sulit untuk menganalisis dan alasan tentang. Untungnya, banyak program konkuren termasuk sebagian besar aplikasi GUI mengikuti pola desain bergaya yang mengontrol kompleksitas yang mendasari.

Untuk menunjukkan beberapa masalah halus yang timbul dengan pemrograman jenis ini, pertimbangkan contoh berikut. Kami memiliki dua benang, A dan B, yang keduanya memiliki akses ke ct variabel. Misalkan, awalnya, ct adalah 0, tetapi ada tempat di kedua A dan B di mana ct bertambah.

|  |  |
| --- | --- |
| A | B |
| ... | ... |
| ct++; | ct++; |

Untuk increment x variabel, (i) nilai v dari x harus diambil dari memori, (ii) nilai baru v 'berdasarkan v, dan (iii) v' harus disimpan dalam lokasi memori yang dialokasikan untuk variabel x. Ini adalah tiga tindakan yang terpisah, dan tidak ada jaminan bahwa tidak ada thread lain akan mengakses variabel sampai ketiga dilakukan. Jadi itu mungkin, misalnya, bahwa urutan operasi dari kedua benang terjadi sebagai berikut:

*fetches ct = 0   
B fetches ct = 0   
A computes the value ct++ = 1   
A stores the value 1 in ct   
B computes new value ct++ = 1   
B stores the value 1 in ct*

Dengan urutan ini dari operasi, nilai akhir untuk ct adalah 1. Tapi dalam susunan lain yang mungkin (misalnya, jika A melakukan semua tindakan yang pertama), nilai akhir akan 2.

Sebuah strategi sederhana untuk mencegah bentuk gangguan (sering disebut kondisi lomba) adalah untuk membuat seluruh akses / memodifikasi siklus / toko untuk memperbarui variabel bersama atom, meskipun fakta bahwa siklus dilakukan dengan menggunakan beberapa instruksi mesin. operasi atom muncul untuk mengeksekusi sebagai instruksi mesin tunggal karena semua benang lainnya dipaksa untuk berhenti mengeksekusi sementara operasi atom mengeksekusi. Akibatnya, tidak mungkin bagi thread lain untuk mengamati nilai variabel diperbarui ketika operasi sedang berlangsung. Sebuah blok kode yang memerlukan eksekusi atom disebut bagian kritis. Beberapa bahasa pemrograman yang mendukung **concurrency** mencakup mulai / kurung akhir untuk melampirkan bagian kritis.

Mekanisme bagian kritis bekerja dengan baik dalam konteks menjalankan program multi-threaded pada komputer dengan prosesor tunggal (uniprocessor a) karena mengurangi untuk memastikan bahwa bagian kritis tidak interruptible (memungkinkan thread lain untuk menjalankan). Tetapi canggung dan tidak efisien pada multiprosesor karena memaksa semua prosesor tapi untuk menghentikan eksekusi selama bagian kritis. (Mesin virtual yang ada mengobati operasi baru yang memaksa pengumpulan sampah sebagai bagian kritis.

Mekanisme yang lebih baik untuk mencegah gangguan dalam program bersamaan yang dapat dijalankan pada Multiprocessors mengunci objek data. Ketika sebuah objek data dikunci oleh thread, tidak ada thread lain dapat mengakses atau memodifikasi data objek sampai benang penguncian rilis itu. Pada intinya, penguncian melemaskan konsep pelaksanaan atom sehingga relatif terhadap objek tertentu. Threads dapat terus menjalankan sampai mereka mencoba untuk mengakses objek terkunci.

Java bergantung pada objek penguncian untuk mencegah interferensi. Sebuah objek dapat dikunci selama pemanggilan metode hanya dengan awalan metode deklarasi dengan pekerjaan disinkronkan. Misalnya, untuk menentukan metode selisih disinkronkan, kita akan menulis:

synchronized void inc() { ct++; }

Kita juga dapat mendeklarasikan metode statis harus disinkronkan, yang mengunci obyek kelas (yang berisi semua variabel statis kelas) daripada contoh objek.

Fitur yang tidak biasa dari mekanisme kunci Java adalah fakta bahwa mengunci sebuah objek hanya menghambat pelaksanaan operasi yang dinyatakan sebagai disinkronisasi. Metode yang tidak dinyatakan sebagai disinkronisasi akan dijalankan bahkan ketika objek terkunci! Ada argumen yang kuat untuk kemampuan ini: mendukung definisi kelas yang partisi operasi dalam dua kelompok: mereka yang membutuhkan sinkronisasi dan mereka yang tidak. Tetapi juga mengundang bug sinkronisasi halus jika pengubah disinkronkan secara tidak sengaja dihilangkan dari satu definisi metode.

Tentu saja, concurrency hanya muncul di Jawa ketika sebuah program menggunakan lebih dari satu thread. Untuk mendukung terciptanya eksplisit thread baru, Java termasuk built-in kelas Thread abstrak, yang memiliki metode abstrak run (). Kita bisa membuat thread baru dengan (i) mendefinisikan memperluas Thread kelas yang mendefinisikan metode run (), (ii) membangun contoh baru dari kelas ini, dan (iii) memanggil metode start () pada contoh baru ini. Awal () metode benar-benar menciptakan thread baru sesuai dengan objek penerima (Thread) dan memanggil run () metode benang itu, sebanyak metode main () dipanggil di kelas akar ketika Anda menjalankan Java Virtual Machine . Sebagai contoh,

class Foo extends Thread {

// must have

public void run() {

...

}

}

Ketika konstruktor untuk Foo disebut, semua perhitungan untuk alokasi objek dan konstruktor doa dilakukan di thread saat ini; thread baru tidak diciptakan sampai metode start () dipanggil untuk Thread () objek. Untuk membuat dan memulai Foo thread baru, thread saat hanya dapat mengeksekusi kode

Thread t = new Foo();

t.start();

Atau, thread saat ini dapat menjalankan run () method dari t objek Thread hanya dengan melakukan operasi

t.run()

dari pada

t.start()

Asumsikan bahwa Foo baru benang t telah dibuat dan mulai. Di beberapa titik dalam pelaksanaan benang asli (sekarang berjalan bersamaan dengan benang t) dapat menunggu benang t untuk mengakhiri dengan menjalankan metode:

t.join();

// waits for the thread object to terminate.

Jadi kita bisa melihat hubungan dari dua benang kontrol sebagai berikut:

main

|

t.start

|\

| \

| |

| /

|/

t.join

|

|

Sinkronisasi beberapa thread tidak dikenakan beberapa overhead. Sebagai contoh, perhatikan kode Java berikut:

class PCount extends Thread {

// \*\* fields \*\*\*

static int sharedCtr = 0;

static final int cntPerIteration = 100000;

static final int noOfIterations = 10;

int id;

// \*\* constructors \*\*

PCount(int i) { id = i; }

// \*\* methods \*\*

void inc() {

sharedCtr++;

}

public void run() {

for (int i = 0; i < cntPerIteration; i++) inc();

System.out.println("Iteration #" + id +

" has completed; sharedCtr = " + sharedCtr);

}

public static void main(String[] args)

throws InterruptedException {

Thread[] tPool = new Thread[noOfIterations];

for (int j = 0; j < noOfIterations; j++) {

tPool[j] = new PCount(j);

}

for (int j = 0; j < noOfIterations; j++) {

tPool[j].start();

}

for (int j = 0; j < noOfIterations; j++) {

tPool[j].join();

}

System.out.println("Computation complete. sharedCtr = "

+ sharedCtr);

}

}

Dalam setiap iterasi, utama menciptakan thread baru. Setelah itu, semua akan disinkronisasi dan nilai akhir ditentukan.

penghitung tidak terkunci dalam contoh ini, dan update mungkin akan hilang karena masalah yang dijelaskan di atas. Kemungkinan dengan yang memperbarui kerugian dapat terjadi bervariasi tergantung pada jumlah benang. Sebagai contoh, dalam tes yang saya berlari beberapa bulan yang lalu

* untuk 1 million iterations, the program 65
* untuk 100,000 iterations, the program lost none.

Rupanya, bahkan dengan 100.000 benang, setiap iterasi terjadi dalam waktu slice tunggal.

Sinkronisasi benang perbaikan masalah update hilang, tapi itu benar-benar memperlambat program turun; bahkan untuk 100.000 iterasi.

Dalam modern yang model event-handling seperti di Jawa dan DrScheme, kami memiliki event handler tunggal yang mengeksekusi acara serial. Protokol ini menghemat overhead sinkronisasi dan menghilangkan potensi deadlock.

1. EXCEPTION HANDLING & EVENT HANDLING
   1. Exception Handling

Exception handling digunakan untuk merubah jalur code yang di eksekusi jika terjadi error yang dispesifikasikan. Kondisi ini disebut exception.

Biasanya hal berikut akan terjadi jika exception terjadi:

* Code yang ada akan tersimpan
* Code execution akan berubah menjadi fungsi exception handler yang telah ditetapkan
* Exception handler mungkin melanjutkan atau menghapus execution

Ketika exception dilaksanakan, code tidak akan jalan, dan PHP akan mencocokan exception tersebut.

Bila exception tidak cocok dengan code manapun. Fatal erorr akan menunjukkan “Uncaught Exception” message.

* 1. Event Handling

Event handling adalah penerimaan dari [event](http://searchsoa.techtarget.com/definition/event) dari beberapa [event handler](http://searchsoa.techtarget.com/definition/event-handler) yang berasal dari produser acara dan proses selanjutnya

HTML biasanya menggunakan JavaScript untuk melaksanakan Event. Misalnya jika kita melakukan sesuatu seperti meng-click suatu tombol, maka JavaScript akan mulai bekerja.

Event yang biasanya digunakan pada HTML

* Onchange = An HTML element has been changed
* Onclick = The user clicks an HTML element
* Onmouseover = The user moves the mouse over an HTML element
* Onmouseout = The user moves the mouse away from an HTML element
* Onkeydown = The user pushes a keyboard key

1. FUNCTIONAL PROGRAMMING LANGUAGES
   1. Function

Suatu bahasa dimana ekspresi disusun atas fungsi panggilan (bukan pernyataan).

* Fungsi: perintah-perintah yang terkumpul menjadi satu dan dapat menghasilkan suatu nilai.
* Disebut bahasa pemrograman fungsional karena memang pada program seluruh kodenya berupa fungsi-fungsi. Bahasa pemrograman fungsional merupakan salah satu bahasa pemrograman yang memperlakukan proses komputasi sebagai evaluasi fungsi-fungsi matematika. Isi dari Program fungsional tidak mengandung pernyataan perintah. Contoh bahasa pemrograman fungsional : Lisp, Scheme, ML, Haskell, Erlang.

12.2 Fungsional Pemrograman

* Itu bukan map atau reduce

Meskipun Anda akan melihat fungsi-fungsi ini dalam setiap bahasa fungsional, itu bukan apa yang membuat bahasa fungsional. Ini hanya sesuatu yang biasa menyerang hampir setiap kali Anda mencoba untuk mengambil efek samping dari pengolahan urutan hal.

* Ini bukan fungsi lambda

Sekali lagi, Anda mungkin akan melihat fungsi-kelas dalam setiap bahasa FP. Tapi itu adalah sesuatu yang alami muncul ketika Anda mulai membangun bahasa yang menghindari efek samping. Ini enabler, tapi tidak akar penyebab.

* Ini bukan jenis

Statis memeriksa jenis adalah alat yang sangat berguna, tapi itu bukan prasyarat untuk FP. Cadel adalah yang tertua bahasa pemrograman fungsional, dan bahasa dinamis tertua.

Jenis statis dapat sangat berguna sekalipun. Haskell menggunakan sistem jenisnya indah dalam serangan di sisi-efek. Tapi mereka tidak bahan yang membuat atau istirahat bahasa fungsional.

* 1. JavaScript bukan Fungsional Bahasa Pemrograman

Bahasa fungsional membantu Anda menghilangkan efek samping di mana Anda bisa, dan kontrol mereka di mana Anda tidak bisa. JavaScript tidak memenuhi kriteria ini. Bahkan, sangat mudah untuk menemukan tempat di mana JavaScript aktif mendorong efek samping.

Target paling mudah adalah ini. Tersembunyi masukan yang ada di setiap fungsi. Apa yang sangat ajaib tentang ini adalah bagaimana secara bebas artinya perubahannya. Bahkan ahli programmer JavaScript memiliki kesulitan melacak apa ini saat mengacu pada. Dari sudut pandang fungsional, fakta bahwa itu ajaib tersedia di semua bau desain.

Meskipun Anda tentu dapat memuat FP perpustakaan (Immutable.js, misalnya) dalam JavaScript, dan yang membuat pemrograman dalam gaya fungsional lebih mudah, itu tidak mengubah sifat dari bahasa itu sendiri.

(By the way, jika Anda suka perpustakaan fungsional yang mendapatkan popularitas di tanah JavaScript, bayangkan berapa banyak Anda ingin bahasa keseluruhan yang mendukung gaya fungsional.)

1. LOGIC PROGRAMMING LANGUAGE CONCEPT
   1. Logic Programming

Logika bahasa pemrograman, dalam arti luas, penggunaan logika matematika untuk pemrograman komputer. Dalam arti sempit yang lebih umum dipahami, adalah penggunaan logika baik sebagai bahasa representasi deklaratif dan prosedural. Menurut berbagai sumber di internet, tidak diperlukan logika untuk mempelajari dan membuat halaman web dengan HTML. Software web editor sudah mencakup berbagai hal mengenai HTML yang akan berguna untuk memperbaiki tampilan yang tidak dapat dilakukan web editor.

Logika Bahasa pemrograman atau yang biasa disebut dengan prolog adalah sebuah Bahasa non-prosedural yant terkait dengan artificial intelligence dan computational intelligence. Logika Bahasa pemrograman atau Bahasa deklaratif adalah Bahasa yang berbasis pada logika simbol. Kita akan menggunakan prolog untuk mendeskripsikan logika Bahasa pemrograman Karena prolog adalah satu-satunya penggunaan luas dari bahasa logika.

Sintaks dari logika bahasa pemrograman sangat berbeda dari bahasa imperative dan fungsional. Semantik dari logika program juga terlihat sedikit kemiripan dengan bahasa pemrograman imperative.

* 1. Prolog
* Data Type

Tipe data tunggal prolog adalah term. Terms adalah antara atoms, numbers, variables, atau compound terms.

* Atom adalah nama tujuan umum tanpa makna yang melekat.
* Numbers adalah float atau integer.
* Compound terms atau functor atau sejumlah argument adalah term di dalam term

Ex: truck\_year('Mazda', 1986) and 'Person\_Friends'(zelda,[tom,jim]

* String adalah Sebuah urutan karakter dikelilingi oleh tanda kutip setara dengan daftar kode karakter (numerik), umumnya di daerah pengkodean karakter , atau Unicode jika sistem mendukung Unicode. Ex, "to be, or not to be"
* *List* adalah koleksi memerintahkan istilah. Hal ini ditandai dengan tanda kurung persegi dengan syarat dipisahkan dengan koma atau dalam kasus daftar kosong, [] . Misalnya, [1,2,3] atau [red,green,blue] .
* Rules and Facts

Program Prolog menjelaskan hubungan, didefinisikan dengan cara klausa. Ada dua jenis klausa: fakta dan aturan. Aturan adalah dalam bentuk

Kepala: - Tubuh.

Dan dibaca sebagai "Kepala adalah benar jika tubuh benar". Tubuh Aturan ini terdiri dari panggilan ke predikat, yang disebut tujuan aturan ini. Built-in predikat ,/2 (yang berarti 2-arity Operator dengan nama , ) menunjukkan hubungannya tujuan, dan ;/2 menunjukkan pemisahan . Konjungsi dan disjunctions hanya dapat muncul dalam tubuh, bukan di kepala aturan.

* Loop dan Recursion

Algoritma iteratif dapat diimplementasikan dengan cara predikat rekursif.

* Proporsisi
* Proposisi sederhana, yang disebut proposisi atomik, terdiri dari istilah majemuk.
* Sebuah istilah majemuk ditulis dalam bentuk notasi fungsi matematika.

BLOG

|  |  |
| --- | --- |
| Nama | Nama Blog |
| William Hartanto | <http://williamhartanto.blog.binusian.org> |
| Richie Muliawan | <http://malkovann.blog.binusian.org/> |
| Ferdynand Cannavaro |  |
| Mulia Pratama |  |
| Muhammad Fardiansyah |  |

DAFTAR PUSTAKA

<https://msdn.microsoft.com/en-us/library/ms531385(v=vs.85).aspx>

<https://www.sitepoint.com/community/t/how-to-insert-a-variable-into-html-code-myvar-doesnt-evaluate/2456>

<http://www.w3schools.com/TAgs/tag_var.asp>

<http://www.w3resource.com/html/attributes/html-scope-attribute.php>

<https://en.wikipedia.org/wiki/Scope_(computer_science)>

<http://www.c-sharpcorner.com/blogs/concept-of-binding1>

<https://en.wikibooks.org/wiki/Introduction_to_Programming_Languages/Binding>

<http://www.w3schools.com/tags/att_a_name.asp>

<http://www.w3schools.com/tags/att_input_name.asp>

<http://www.w3schools.com/TAgs/att_name.asp>

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions_and_Operators>

<http://php.net/manual/en/language.expressions.php>

<http://php.net/manual/en/language.operators.assignment.php>

<https://www.safaribooksonline.com/library/view/javascript-the-definitive/9781449393854/ch04s11.html>

<https://www.w3.org/TR/html4/types.html>

<http://fantasai.tripod.com/qref/Appendix/datatype.html>

<http://php.net/manual/en/language.expressions.php>

<http://www.howtocreate.co.uk/tutorials/javascript/controls>

<https://docs.oracle.com/javase/tutorial/java/concepts/object.html>

<http://ferdataadhitya.blogspot.co.id/2010/05/sekilas-function-pada-java.html>

<http://www.slideshare.net/DevandyEnda/cara-membuat-fungsi-dan-prosedur-pada-java>

<http://groups.engin.umd.umich.edu/CIS/course.des/cis400/maxim/lectures/chp8.htm>

<http://ferdataadhitya.blogspot.co.id/2010/05/sekilas-function-pada-java.html>

<http://www.slideshare.net/DevandyEnda/cara-membuat-fungsi-dan-prosedur-pada-java>

<https://www.cs.rice.edu/~cork/book/node96.html>

<https://docs.oracle.com/javase/tutorial/java/concepts/object.html>

<https://www.cs.rice.edu/~cork/book/node96.html>