Modul Pertemuan ke 2
CMC101 Topik Dalam Pemrograman
Sumber : kompilasi dari berbgaia sumber

1. BAHASA PEMROGRAMAN DAN SEJARAHNYA
Untuk melaksanakan tugasnya maka komputer akan diperintah oleh sebuah sistem, perintah dasar komputer ini disimpan ke dalam sebuah sistem yang disebut sebagai sistem operasi seperti DOS, Unix, Windows, Linux, Solaris dsb. Perintah-perintah tersebut tentunya menggunakan bahasa mesin yang oleh komputer hanya mengenal angka 1 dan 0 (binary) dimana angka 1 dipresentasikan sebagai sebuah wadah yang mimiliki listrik dan angka 0 tidak memiliki listrik (Tegangan).

Untuk membuat sebuah sistem dikomputer maka diperlukan sebuah alat (tools) yang kita sebut sebagai bahasa pemrograman, jadi yang terlintas dibenak kita saat ini adalah "Alat yang dibuat untuk membuat program (sistem) disebut sebagai bahasa pemrograman", sehingga Program-Program seperti Macro/Turbo Assembly, Turbo C, Microsft Visual C++, C++ Builder, Microsoft Visual Basic, Delphi dsb digunakan untuk membuat aplikasi bukan sebagai aplikasi untuk mengolah data seperti Microsoft Excel, SPPS dsb.

Atau dapat juga dikatakan bahwa bahasa pemrograman adalah bahasa yang digunakan pada komputer untuk melakukan tugas tertentu. Seperti juga bahasa yang digunakan manusia secara umum, bahasa pemrograman banyak sekali jenisnya. Bahasa pemrograman dapat dikelompokkan berdasarkan tingkatan, seperti bahasa tingkat rendah (low Level), bahasa tingkat sedang (Mid Level Language), bahasa tingkat tinggi .

1.1. Sebelum 1940
Pada jaman ini terdapat bahasa pemrograman yang pertama kali muncul sebelum adanya komputer modern, artinya bahasa pemrograman lebih tua dari komputer itu sendiri. Pada awal kemunculannya, bahasa pemrograman masih dalam bentuk kode-kode bahasa mesin. Bahasa mesin merupakan bahasa yang terdiri atas kode-kode mesin dan hanya dapat diinterpretasikan langsung oleh mesin komputer. Bahasa mesin ini tergolong bahasa tingkat rendah, karena hanya berupa kode 0 dan 1 seperti disampaikan pada bagian atas.

1.2. Periode 1940-an
Dengan bahasa mesin ditemukan banyak kesulitan untuk pengembangan dan perbaikan pada program yang dibuat saat itu, Tahun 1940-an komputer bertenaga listrik dibuat, dengan kecepatan yang sangat terbatas dan kapasitas memori yang mencukupi untuk programmer memprogram, kemudian terciptalah bahasa assembly (Assembly language). Bahasa assembly adalah bahasa simbol dari bahasa mesin.

Setiap kode bahasa mesin memiliki simbol sendiri dalam bahasa assembly. Misalnya Move untuk memindahkan isi data, ADD untuk penjumlahan, MUL untuk perkalian, SUB untuk pengurangan, dan lain-lain. Penggunaan bahasa Asembly dirasa belum sempurna karena selain sulit untuk diimplementasikan, ternyata bahasa ini juga sulit jika sang programer ingin mengembangkan program buatannya. Pada tahun 1948, Konrad Zuse mempublikasikan sebuah paper tentang bahasa pemrograman miliknya yakni Plankalkül. Bagaimanapun, bahasa tersebut tidak digunakan pada masanya dan terisolasi terhadap perkembangan bahasa pemrograman yang lain. Beberapa bahasa pemrograman yang berkembang pada masa itu antara lain:

- Plankalkül (Konrad Zuse) – 1943
- ENIAC coding system – 1943
- C-10 – 1949

1.3. Periode tahun 1950-an sampai dengan tahun 1960-an

Mulai tahun 1950 dibuatlah bahasa pemrograman modern, yang turun-temurun dan tersebar luas hingga saat ini. Bahasa ini menggunakan istilah atau reserved word yang dekat dengan bahasa manusia seperti READ untuk membaca, WRITE untuk menulis dsb. Dalam perkembangannya Bahasa Tingkat Tinggi juga terdiri dari beberapa metode pemrograman, yaitu Procedural Programing dan Object Oriented Programing. Letak perbedaannya yaitu, jika pada procedural programing program dijalankan dengan menggabungkan variable, procedure-procedure yang saling keterkaitan dan berjalan berurut, sedangkan pada OOP seluruh task dijalankan berdasarkan kedalam object.

- FORTRAN (1955), the “FORmula TRANslator”, ditemukan oleh John W. Backus dll.
- LISP, the “LISt Processor”, ditemukan oleh John McCarthy dll.
- COBOL, the COmmon Bussines Oriented Language, dibuat oleh the Short Range Commitee, dan Grace Hopper berperan sangat besar disini.

Overview:

- Regional Assembly Language – 1951
- Autocode – 1952
- FORTRAN – 1954
- FLOW-MATIC – 1955
- COMTRAN – 1957
- LISP – 1958
- ALGOL – 1958
- COBOL – 1959
- APL – 1962
- SIMULA – 1962
- BASIC – 1964

- PL/I -1964

1.4. Periode 1967-1978: Menetapkan Paradigma Fundamental.

Periode diantara tahun 60-an sampai dengan 70-an membawa pengaruh yang besar dalam perkembangan bahasa pemrograman. Kebanyakan dari pola bahasa pemrograman yang utama yang saat ini banyak digunakan:

- Simula, ditemukan pada akhir 60-an oleh Nygaard dan Dahl sebagai superset dari Algol 60, merupakan bahasa pemrograman pertama yang didesain untuk mendukun pemrograman berorientasi object.
- C, sebuah tahapan awal dari sistem bahsa pemrograman, yang dikembangkan oleh Dennis Ritchie dan Ken Thompson di Bell Labs antara tahun 1969 dan 1973.
- Smalltalk (pertengahan tahun 70-an) menyajikan desain ground-up yang lengkap dari sebuah bahasa yang berorientasi objek.
- Prolog, didesain pada tahun 1977 oleh Colmerauer, Roussel, and Kowalski, merupakan bahasa pemrograman logika yang pertama.
- ML membangun sebuah sistem polimorfis (ditemukan oleh Robin Miller pada tahun 1973) diatas sebuah Lisp, yang merintis bahasa pemrograman fungsional bertipe statis.

Beberapa bahasa pemrograman yang berkembang dalam periode ini termasuk:

- Pascal – 1970
- Forth – 1970
- C – 1970
- Smaltalk – 1972
- Prolog – 1972
- ML – 1973
- SQL – 1978

1.5. Periode 1980-an: konsolidasi, modul, performa

1980s adalah tahun dari konsolidasi relatif. C++ dikombinasikan dengan sistem programming dan berorientasi obyek. Pemerintah Amerika Serikat menstandardisasi Ada, sebuah sistem pemrograman yang bertujuan untuk digunakan para kontraktor untuk bertahan. Di Jepang dan di tempat lain, penjumlahan luas yang telah di selidiki disebut" generasi ke lima" bahasa-bahasa yang menyatukan logika pemrograman konstruksi. Masyarakat bahasa fungsional gerak ke standarisasi ML dan Cedal. Dibandingkan dengan menemukan paradigma-paradigma baru, semua pergerakan ini menekuni gagasan-gagasan yang ditemukan di dalam dekade sebelumnya.

Bagaimanapun, satu kecenderungan baru di dalam disain bahasa adalah satu fokus yang ditingkatkan di pemrograman untuk sistem besar-besaran melalui penggunaan

dari modul, atau kesatuan organisasi besar-besaran dari kode. Modula, Ada, dan ML semua sistem modul terkemuka yang dikembangkan pada 1980-an.

Beberapa bahasa pemrograman yang berkembang dalam periode ini termasuk:

- Ada – 1983
- C++ – 1983
- Eiffel – 1985
- Perl – 1987
- FL (Backus) – 1989

1.6. Periode 1990-an: Visual

Pada periode ini bahasa selain berorientasi objek juga sudah dikembangkan berbasi Visual sehingga semakin mudah untuk membuat program aplikasi, diawali oleh Python dan Microsoft Visual Basic 1 pada tahun 1991, Delphi yang dikembangkan dari Pascal for windows akhirnya pada tahun 1997 Visual Basic 5 diluncurkan dengan kemudahan koneksi ke database, OO Cobol sudah ditemukan dalam versi windows. Bagi kebanyakan programmer database tidak dapat dipungkiri bahwa era 1990an merupakan era yang paling produktif semenjak bahasa pemrogrammar diciptakan.
Beberapa bahasa pemrograman yang berkembang dalam periode ini termasuk

- Haskel – 1990
- Python – 1991
- Java – 1991
- Ruby – 1993
- OO Cobol
- Lua – 1993
- ANSI Common Lisp – 1994
- JavaScript – 1995
- PHP – 1995
- C# – 2000
- JavaFX Scrip, Live Script,
- Visual Basic

1.7. Periode 2000an

Pada saat ini ada kecenderungan para vendor bahasa pemrograman untuk menggiring programmer hanya dengan mengggunakan produk mereka untuk membuat program meski kita sadari bahwa sulit rasanya untuk membuat program yang tangguh hanya dengan satu bahasa pemrograman, hal ini tentunya dilakukan dengan tujuan kelangsungan usaha mereka, namun terlepas dari semua itu terdapat dua konsepsi besar dalam periode ini dimana kemudahan berbasis visual sudah mulai digiring ke basis internet dan mobile, dengan bermunculan webservice dan berbasis net dan a mobile flatform.

Konsep pertama yang dicermati adalah konsepsi Microsoft dimana dengan Visual Net akan menyediakan berbagai bahasa pemrograman seperti VB Net , VC++ Net, ASP

NET yang di compile dengan berbagai bahasa akan tetapi berjalan pada satu sistem operasi yakni windows. (Compile any program run one system)

Konsepsi Kedua, Merupakan konsep yang terbalik dari konsep pertama yakni apa yang ditawarkan Sun Microsystem melalui produknya Java, J2ME, JDK, yakni dicompile dengan satu bahasa pemrograman (java) dan berjalan dibanyak sistem operasi. (Compile one program running any system)

Selain itu periode ini juga merupakan jamannya CMS (Content Manajemen System), lompatan pengembangan PHP Script begitu cepat, dimana untuk membuat website atau portal telah tersedia banyak template, Banyak modul-modul yang siap pakai sehingga programmer atau webmaster tidak perlu lagi mempelajari semua script html dan bahasanya, tinggal merangkai modul yang tersedia sehingga dalam beberapa hari saja sebuah web sudah dapat dibuat. Apa yang ditawarkan Mambo, PhkNuke dan Jomla saat ini sangat memudahkan para desainer web.

Beberapa bahasa pemrograman yang berkembang dalam periode ini termasuk

- Tcl/Tk,
- O'Caml,
- Ruby,
- Phyton 3.1,
- Java 6 JDK, JED, Java Beans, J2ME
- Microsoft Visual Net (VB Net, C++ Net, ASP NET) 2008
- Java Scrip Template oleh Mambo, PhpNuke, Jomla

2. TINGKAT KEDEKATAN DENGAN KOMPUTER

Menurut tingkat kedekatannya dengan mesin komputer, bahasa pemrograman terdiri dari:

1. Bahasa Mesin, yaitu memberikan perintah kepada komputer dengan memakai kode bahasa biner, contohnya 01100101100110
2. Bahasa Tingkat Rendah, atau dikenal dengan istilah bahasa rakitan (bahasa Inggris *Assembly*), yaitu memberikan perintah kepada komputer dengan memakai kode-kode singkat (kode *mnemonic*), contohnya MOV, SUB, CMP, JMP, JGE, JL, LOOP, dsb.
3. Bahasa Tingkat Menengah, yaitu bahasa komputer yang memakai campuran instruksi dalam kata-kata bahasa manusia (lihat contoh Bahasa Tingkat Tinggi di bawah) dan instruksi yang bersifat simbolik, contohnya {, }, ?, <<, >>, &&, ||, dsb.
4. Bahasa Tingkat Tinggi, yaitu bahasa komputer yang memakai instruksi berasal dari unsur kata-kata bahasa manusia, contohnya begin, end, if, for, while, and, or, dsb.

2.1. Sejarah pandangan pertama tiga generasi

Istilah "generasi pertama" dan "generasi kedua" bahasa pemrograman tidak digunakan sebelum coining dari istilah "generasi ketiga." Bahkan, tak satupun dari ketiga istilah yang disebutkan dalam kompendium awal bahasa pemrograman. Pengenalan generasi ketiga dari teknologi komputer bertepatan dengan penciptaan generasi baru bahasa pemrograman. Pemasaran untuk pergeseran generasi dalam mesin tidak berkorelasi dengan beberapa perubahan penting dalam apa yang disebut tingkat tinggi bahasa pemrograman, dibahas di bawah, memberikan konten teknis untuk perbedaan kedua / ketiga-generasi antara bahasa pemrograman tingkat tinggi juga, dan mengubah nama refleks bahasa assembler sebagai generasi pertama.

2.2. Generasi Kedua

Artikel utama: Generasi kedua bahasa pemrograman

Bahasa pemrograman generasi kedua, awalnya hanya disebut bahasa pemrograman tingkat tinggi , diciptakan untuk menyederhanakan beban pemrograman dengan membuat ekspresi yang lebih seperti modus normal ekspresi pemikiran yang digunakan oleh programmer. Mereka diperkenalkan pada akhir 1950-an, dengan FORTRAN mencerminkan kebutuhan programmer ilmiah, ALGOL mencerminkan upaya untuk menghasilkan tampilan standar Eropa / Amerika.

Masalah yang paling penting yang dihadapi oleh para pengembang dari tingkat kedua bahasa adalah pelanggan meyakinkan bahwa kode yang dihasilkan oleh para penyusun dilakukan dengan baik-cukup untuk membenarkan meninggalkan pemrograman assembler. Dalam pandangan skeptisisme luas tentang kemungkinan memproduksi program efisien dengan sistem pemrograman otomatis dan fakta bahwa inefisiensi tidak bisa lagi disembunyikan, para pengembang yakin bahwa jenis sistem yang mereka ada dalam pikiran akan banyak digunakan hanya jika mereka bisa menunjukkan bahwa hal itu akan menghasilkan program hampir seefisien yang tangan kode dan melakukannya pada hampir setiap pekerjaan. Compiler FORTRAN dipandang sebagai tour de force-dalam produksi berkualitas tinggi kode, bahkan termasuk "… a Monte Carlo simulasi pelaksanaannya … sehingga dapat meminimalkan transfer barang antara toko dan indeks register."

2.3. Generasi ketiga

Artikel utama: generasi ketiga bahasa pemrograman

Pengenalan generasi ketiga dari teknologi komputer bertepatan dengan penciptaan generasi baru bahasa pemrograman. [1] Fitur penting dari generasi ketiga bahasa adalah hardware-kemerdekaan mereka, ekspresi yaitu algoritma dengan cara yang independen dari karakteristik mesin yang algoritma akan berjalan.

Beberapa atau semua dari sejumlah perkembangan lain yang terjadi pada saat yang sama dimasukkan dalam 3GLs.

Interpretasi diperkenalkan. 3GLs Beberapa disusun , proses analog dengan penciptaan dieksekusi kode mesin lengkap dari kode assembly, perbedaan adalah bahwa dalam bahasa tingkat tinggi tidak ada lagi hubungan, satu-ke-satu, atau bahkan linier antara petunjuk source code kode mesin instruksi dan. Compiler dapat menargetkan

hardware yang berbeda dengan memproduksi terjemahan yang berbeda dari perintah kode sumber yang sama.
Penafsir, di sisi lain, pada dasarnya menjalankan instruksi kode sumber itu sendiri – jika seseorang menemukan sebuah "add" instruksi, ia melakukan tambahan itu sendiri, daripada keluaran instruksi tambahan akan dieksekusi kemudian. Mesin-kemerdekaan dicapai dengan memiliki interpreter yang berbeda dalam kode mesin dari platform yang ditargetkan, yaitu penafsir itu sendiri umumnya harus dikompilasi. Interpretasi bukanlah "muka" linear, tetapi model alternatif untuk kompilasi, yang terus ada di samping, dan lainnya, baru-baru ini dikembangkan, hibrida. Lisp adalah bahasa interprepreted awal.
Para 3GLs awal, seperti Fortran dan Cobol , adalah spaghetti kode , yaitu mereka memiliki gaya yang sama dari aliran kontrol sebagai assembler dan kode mesin , membuat penggunaan berat dari goto pernyataan. pemrograman terstruktur [2] diperkenalkan model di mana program itu dilihat sebagai hirarki blok bersarang daripada daftar linear instruksi. Misalnya, programmer terstruktur adalah untuk memahami sebuah loop sebagai blok kode yang diulang, bukan perintah begitu banyak diikuti oleh melompat mundur atau goto. Pemrograman terstruktur kurang tentang *kekuasaan* – dalam arti satu tingkat yang lebih tinggi commannd ekspansi ke berbagai tingkat rendah yang – dari *keselamatan.* Pemrogram berikut ini jauh kurang rentan untuk membuat kesalahan. Pembagian kode ke blok, subrutin dan modul lainnya dengan antarmuka jelas-didefinisikan juga memiliki manfaat produktivitas dalam memungkinkan programmer banyak untuk bekerja pada satu proyek. Setelah diperkenalkan (dalam ALGOL bahasa), pemrograman terstruktur dimasukkan ke dalam hampir semua bahasa, dan dipasang dengan bahasa yang awalnya tidak memilikinya, seperti Fortran , dll
Struktur Blok juga dikaitkan dengan depresiasi variabel global , sumber kesalahan yang sama dengan goto . Sebaliknya, bahasa terstruktur diperkenalkan scoping leksikal dan manajemen penyimpanan otomatis dengan stack.
Fitur lain yang tingkat tinggi adalah pengembangan dari sistem tipe yang melampaui jenis data kode mesin yang mendasari, seperti string , array dan catatan .
Dimana 3GLs awal adalah tujuan khusus, (misalnya ilmu pengetahuan atau perdagangan) upaya telah dilakukan untuk menciptakan tujuan umum bahasa, seperti C dan Pascal . Sementara ini menikmati sukses besar, domain bahasa tertentu tidak menghilang.

2.4. Sebuah karakterisasi alternatif dari tiga generasi pertama

Karena setidaknya 1979, banyak penulis telah menggunakan karakterisasi yang berbeda dari generasi bahasa pemrograman.

2.4.a. Generasi Pertama

Dalam kategorisasi ini, *generasi pertama bahasa pemrograman* mengacu pada angka kode mesin , yaitu petunjuk numerik secara langsung sesuai dengan petunjuk perangkat keras individu.

Awalnya, tidak ada penerjemah yang digunakan untuk mengkompilasi atau merakit sumber assembler untuk menghasilkan kode mesin numerik. Generasi pertama instruksi pemrograman yang masuk melalui switch panel depan dari sistem komputer. Manfaat utama dari pemrograman dalam kode mesin adalah bahwa kode pengguna menulis dapat berjalan sangat cepat dan efisien, karena secara langsung dieksekusi oleh CPU . Namun, kode mesin adalah jauh lebih sulit untuk belajar dari yang lebih tinggi bahasa pemrograman generasi, dan itu jauh lebih sulit untuk mengedit jika terjadi kesalahan. Selain itu, jika instruksi perlu ditambahkan ke dalam memori di lokasi tertentu, maka semua instruksi setelah titik penyisipan perlu dipindahkan ke membuat ruang dalam memori untuk menampung instruksi baru. Melakukannya pada panel depan dengan switch bisa sangat sulit.

2.4.b. Generasi Kedua

Generasi kedua bahasa pemrograman mengacu pada (simbolis) bahasa assembly . Istilah ini diciptakan untuk memberikan perbedaan dari sebelumnya bahasa kode mesin dan tingkat yang lebih tinggi bahasa pemrograman generasi ketiga (3GL) seperti Fortran , COBOL dan Algol . Generasi kedua bahasa pemrograman memiliki sifat sebagai berikut:

- Kode assembly simbolis dapat dibaca dan ditulis oleh seorang programmer. Untuk menjalankan pada komputer harus dikonversi ke dalam bentuk mesin yang dapat dibaca, proses yang disebut perakitan.
- Bahasa adalah satu-ke-satu korespondensi dengan instruksi mesin dari keluarga prosesor tertentu dan lingkungan.

Majelis bahasa kadang-kadang digunakan dalam kernel dan driver perangkat (meskipun C umumnya digunakan untuk ini di kernel modern), tetapi lebih sering menemukan penggunaan dalam pengolahan yang sangat intensif seperti game, video editing, grafis manipulasi / render.
Salah satu metode untuk membuat kode seperti ini dengan memungkinkan compiler untuk menghasilkan mesin-dioptimalkan perakitan versi bahasa fungsi tertentu. Sumber perakitan kemudian tangan-tuned, memperoleh wawasan baik brute-force dari algoritma mesin mengoptimalkan dan kemampuan intuitif optimizer manusia.

2.4.c. Generasi ketiga

Generasi ketiga bahasa pemrograman (3GL) awalnya disebut semua bahasa pemrograman pada tingkat yang lebih tinggi dari perakitan. Sedangkan instruksi individu dari bahasa generasi kedua dalam satu-ke-satu korespondensi dengan instruksi mesin individu (yaitu mereka yang dekat dengan domain mesin), bahasa generasi ketiga bertujuan untuk menjadi lebih dekat ke domain manusia. Instruksi beroperasi pada tingkat, lebih tinggi abstrak, lebih dekat dengan cara berpikir manusia, dan setiap instruksi individu dapat diterjemahkan ke dalam sejumlah (besar kemungkinan) mesin-tingkat instruksi. Bahasa generasi ketiga dimaksudkan untuk lebih mudah digunakan daripada bahasa generasi kedua. Dalam rangka untuk menjalankan pada komputer yang sebenarnya, kode yang ditulis dalam bahasa generasi ketiga harus dikompilasi baik secara langsung ke dalam kode mesin, atau ke perakitan, dan kemudian dirakit. Kode yang ditulis dalam bahasa generasi ketiga

umumnya dapat dikompilasi untuk dijalankan pada komputer yang berbeda menggunakan berbagai arsitektur perangkat keras.
Pertama kali diperkenalkan pada akhir 1950-an, FORTRAN , ALGOL dan COBOL adalah contoh awal dari bahasa generasi ketiga.
Bahasa generasi ketiga cenderung baik seluruhnya (atau hampir seluruhnya) independen dari hardware, seperti untuk keperluan umum bahasa seperti Pascal , Java , FORTRAN , dll, meskipun beberapa telah ditargetkan pada prosesor tertentu atau arsitektur keluarga prosesor, seperti , misalnya PL / M yang ditargetkan pada prosesor Intel, atau bahkan C , yang sebagian auto-increment dan auto-decrement idiom seperti * (c + +) berasal dari hardware PDP-11 ini yang mendukung auto-increment dan auto-decrement mode pengalamatan tidak langsung, dan di mana C pertama kali dikembangkan.
Kebanyakan "modern" bahasa (BASIC , C , C + + , C # , Pascal , Ada , dan Jawa) juga generasi ketiga bahasa.
Dukungan 3GLs Banyak terstruktur pemrograman .

2.4.d Generasi Kemudian

Awalnya, bahasa pemrograman semua pada tingkat yang lebih tinggi dari perakitan yang disebut "generasi ketiga", tetapi kemudian, istilah "generasi keempat" diperkenalkan untuk mencoba membedakan (kemudian) bahasa deklaratif baru (seperti Prolog dan domain- spesifik bahasa) yang diklaim beroperasi pada tingkat yang lebih tinggi, dan dalam domain bahkan lebih dekat ke pengguna (misalnya pada tingkat bahasa alami) daripada asli, bahasa tingkat tinggi seperti imperatif Pascal , C , Algol , Fortran , BASIC , dan lain-lain
"Generasi" klasifikasi bahasa tingkat tinggi (generasi ke-3 dan kemudian) tidak pernah sepenuhnya tepat dan kemudian mungkin ditinggalkan, dengan klasifikasi yang lebih tepat mendapatkan penggunaan umum, seperti object-oriented , deklaratif dan fungsional. C memunculkan C + + dan kemudian Java dan C # , Lisp untuk CLOS , Ada ke Ada 2.012 , dan bahkan COBOL untuk COBOL2002 , dan bahasa baru telah muncul dalam "generasi" juga.

3. PARADIGMA BAHASA PEMROGRAMAN

Bahasa pemrograman dikenal pertama kali sejak penemuan komputer digital pada tahun 1940-an. Bahasa pemrograman mulai berkembang dan dikembangkan sejak tahun 1950-an dengan dimulainya bahasa assembly yang mengiringi berkembangnya komputer untuk keperluan komersial. Bahasa komputer yang pertama kali adalah FORTRAN yang merupakan singkatan dari Formula Translation. Bahasa ini dibuat oleh John Bacus pada awal 1950-an hingga awal 1960-an. Bahasa ini bermanfaat dalam dunia perbankan dan pencatatan keuangan atau akuntansi.
Setelah itu bahasa pemrograman mulai beragam dalam segi jumlah maupun fungsinya. Bahasa yang dikembangkan di era tersebut misalnya dalah COBOL yang merupakan singkatan dari Common Bussiness Oriented Languange oleh Grace

Hopper dan LISP yang merupakan singkatan dari List Processing yang dibuat oleh John McCarthy. Lalu kemudian bahasa-bahasa pemrograman berkembang semakin pesat dan muncul bahasa pemrograman tingkat tinggi yang lebih multifungsi seperti C, C++, BASIC, PASCAL dan ADA. Dari berbagai bahasa pemrograman tersebut, bahasa C adalah yang paling populer dan memiliki turunan yang digunakan untuk kepentingan lain seperti PHP untuk Web dan Java untuk multiplatform application.

Paradigma pemrograman suatu sudut pandang dalam dunia pemrograman dan menjadi suatu pendekatan khusus dalam memecahkan suatu persoalan dalam menyelesaian masalah pemrograman. Ada banyak cara untuk menyelesaikan masalah mengikuti aliran atau "genre" tertentu dari program dan bahasa. Empat paradigma pemrograman yang sangat mendasar yaitu:

3.1. Imperative Programming

Paradigma ini didasari oleh konsep mesin Von Newman (stored program concept) sekelompok tempat penyimpanan (memori), yang dibedakan menjadi memori instruksi dan memori data, masing-masing memori tersebut dapat diberi nama dan dinilai, selanjutnya instruksi akan dieksekusi satu persatu secara sekuensial oleh sebuah proses tunggal.

Program dalam paradigma ini berdasarkan pada struktur informasi di dalam memori dan manipulasi dari informasi yang disimpan tersebut. Kata kunci yang digunakan dalam paradigma ini adalah:

Algoritma + struktur data = program

Kelebihan dari paradigma ini adalah efisiensi karena lebih dekat dengan konsep mesin, kekurangan adalah batasan yang sangat mengikat sehingga terkadang menyulitkan programmer yang tidak terbiasa. Contoh bahasa pemrograman yang menggunakan paradigma imperative atau procedural adalah: Algol, Pascal, Fotran, Basic, Cobol, C, Ada dan Perl.

Pemrograman Imperatif mempunyai karakteristik berupa status dan instruksi/perintah untuk mengubah status program. Status diwakili oleh variabel sedangkan instruksi diwakili oleh statement. Ciri-ciri pemrograman imperatif yaitu :

a. Adanya instruksi/command/perintah/kalimat-kalimat perintah
   contoh : GOTO 10
b. Adanya status yang berubah
   contoh : dengan adanya perintah GOTO 10 maka status program akan loncat mengerjakan statement yang ada di line number 10

3.1.a. Variabel dan Penugasan

Variabel adalah identifier/pengenal yang berisi data yang dapat berubah-ubah nilainya di dalam program. Penugasan(Assignment) adalah suatu aksi yang menyebabkan peletakan atau pemberian suatu nilai di suatu lokasi atau variabel

Contoh :

A = 3 (statement penugasan pada bahasa BASIC)

A := 3 (statement penugasan pada bahasa PASCAL)

3.1.b. Statement/Perintah Tidak Terstruktur

Statement/perintah tidak terstruktur merupakan statement yang berisi perintah untuk mengerjakan statement tertentu yang diidentifikasi dengan suatu label, baik menggunakan statement GOTO bersyarat maupun tanpa syarat. Salah satu program yang mengandung perintah tidak terstruktur adalah bahasa BASIC.

Contoh program:

```
 10 LET A = 5
20 LET B = A + 1
30 GOTO 50
40 LET B = A * 2
50 PRINT B
60 END
```

3.2. Functional Programming

Functional programming disebut-sebut sebagai paradigma pemrograman para hipster (yang mainstream saat ini adalah imperative programming dan OOP), karena baru naik beberapa tahun belakangan semenjak banyak startup di Silicon Valley beralih menggunakan bahasa-bahasa pemrograman yang menganut paradigma ini. Sebenarnya functional programming ini bukanlah sesuatu yang baru, karena akar dari functional programming itu adalah Lambda Calculus yang sudah ada dari tahun 1930. Bahasa pemrograman yang sudah mulai menganut paradigma ini pun sudah ada dari tahun 1950, yaitu Lisp yang dikembangkan oleh John McCarthy.

Functional programming sekarang ini mulai populer kembali karena kebutuhan yang tinggi akan high-reliable dan high-performance application. Selain itu konsep functional programming juga menjanjikan suatu product aplikasi yang terbebas dari banyaknya bug. Kita bisa mengaplikasikan functional programming dengan untuk pemrograman sehari-hari yang bertujuan membuat aplikasi kita lebih reliable.

Paradigma programming fungtional melakukan komputasi sebagai evaluasi dari mathematical functions dan menghindari keadaan (state) dan perubahan (mutable) data. Penekanan aplikasi dari functions, sangat kontras dengan gaya imperative programming, yang menekannkan perubahan keadaan (state)

Bahasa Pemrograman fungsional sering disebut aplikatif. Program yang terdiri dari teori fungsi matematika atau algoritmik Pemrograman Fungsional itu sendiri adalah suatu program yang setiap persoalan diselesaikan dengan menggunakan sebuah fungsi matematika. Dengan cara menggambarkan sebuah kasus dengan suatu fungsi matematika dan memberikan input atau domain lalu mengeluarkan hasil atau output yang disebut range.

Functional programming ini memiliki cara kerja atau proses kerja dengan fungsi kelas satu , dimana fungsi yang di susun pada sebuah program tersebut dapat dikirm sebagai sebuah argumen untuk fungsi yang lainnya.

Pemrograman Fungsional berdasarkan konsep matematika dari sebuah fungsi meliputi:

- Suatu set fungsi primitif
- Suatu set format fungsional
- Aplikasi operasi
- Operator
- Sebuah simbol untuk pengoperasian suatu fungsi, Jenis Operator yang masih sama dengan prosedural paradigma:
- Aritmatika (+,-,/,*)
- Logika (and.or,not,xor)
- Boolean
- Untai
- Himpunan
- Relasi

Functional programming paradigma sangat bergantung pada immutable data structure sehingga dapat memberi keuntungan tersendiri, seperti kemudahan saat melakukan unit testing karena output-nya akan selalu sama dengan yang diharapkan tanpa ada perubahan data. Kode yang kita tulis pun akan lebih mudah terbaca, karena functional programming sangat fokus kepada hal apa yang ingin diselesaikan, tidak seperti imperative programming yang lebih menekankan bagaimana cara menyelesaikannya.

Functional programming pada dasarnya dapat diterapkan ke bahasa pemrograman apa pun, hanya saja ada beberapa fitur yang tidak tersedia atau fitur yang tidak optimal. Berbeda jika kita memang coding menggunakan bahasa pemrograman yang dirancang untuk paradigma functional programming.

Pemrograman Fungsional dikenal mampu menyediakan dukungan yang lebih baik untuk pemrograman yang terstruktur dari pada pemrograman imperatif. Contoh pemrograman fungsional yakni LISP, Scheme, Haskell, ML, APL dan LOGO.

Contoh program:

LISP merupakan sebuah bahasa ekspresi, awal nya di rancang untuk memproses List dengan memanipulasi symbol. Di mulai pertama Implementasi LISP, digunakan untuk mendefinisikan dirinya sendiri.
Mulanya LISP adalah bahasa yang sangat kecil dan sederhana, yaitu:

- Fungsi untuk membentuk dan mengakses list
- Mendefinisikan fungsi baru
- Mendeteksi kesamaan
- Evaluasi ekspresi
- Kendali Program: Rekursi dan Kondisi tunggal

LISP merupakan bahasa yang memiliki fitur unik. Bahasa LISP menjadi media yang menyenangkan untuk dipelajari yang berupa bentukan-bentukan pemrograman dan struktur data serta dapat menghubungkan ke fitur-fitur bahasa yang mendukungnya.
Selain itu, adanya penambahan pada LISP yaitu :

- Fungsi untuk penstrukturan data
- Kendali program
- Aritmatika real dan integer
- I/O
- Penyuntingan fungsi LISP
- Penelusuran eksekusi program.

3.2.a. Ekspresi Fungsional

adalah sebuah teks yang terdiri dari nama simbol, operator/fungsi, ( ), yang dapat menghasilkan nilai hasil dari evaluasi ekspresi. Hasilnya dapat berupa nilai numerik atau Boolean.

3.2.b. Ekspresi Rekursif

disebut rekursif jika definisi tersebut mengandung terminologi dirinya sendiri.
Ekspresi rekursif direalisasikan dengan membuat fungsi rekursif dan didasari analisis rekurens.

3.2.c. Mendefinisikan Type

Struktur dan nama type

- Atom adalah angka atau seutu karakter yang pada penulisannya di awali dengan sebuah tanda ' (quote)
- List adalah kumpulan dari atom atau list
- Selektor : Untuk menentukan komponen-komponen type
- Konstruktor : Untuk menentukan type
- Predikat : Untuk menentukan karekteristik dan pemeriksaan besaran

3.3. Object Oriented Programming

Konsep OOP bermula pada era 1960-an. Sebuah bahasa pemrograman Simula memperkenalkan berbagai konsep yang mendasari OOP dengan SIMULA I (1962-65) dan Simula 67 (1967). Kemudian pada tahun 70-an, bahasa pemrograman Smalltalk menjadi yang pertama kali disebut object-oriented.
Pada tahun 1980-an, dua bahasa pemrograman ADA (US Department of Defense) dan PROLOG (the Japanese “Fifth Generation Computer Project”) dipercayai akan bersaing ketat sebagai bahasa pemrograman yang paling dominan. Namun justru OOP yang menjadi paradigma pemrograman yang paling dominan sampai sekarang. Bahasa pemrograman yang object-oriented seperti C++ pada tahun 80-an menjadi populer. Pada tahun 90-an, bahasa-bahasa pemrograman seperti Java mulai

menerapkan OOP. Sampai pada 2002, Microsoft Visual Studio memperkenalkan bahasa object-oriented baru yang diberi nama C#. Disusul VB.NET yang merupakan penyempurnaan Visual Basic 6.0 yang tidak mendukung OOP. Contoh Pemrograman yang berbasis Paradigma Objek Oriented adalah Visual Foxpro, Delphi, Java, C#, C++, Pascal, VB.Net, Smalltalk, Simula, Ruby, Python, Eifel dan PHP.

3.3.a. Mengapa menggunakan OOP?

Programming tipe ini bekerja dengan baik untuk program kecil yang berisi code relative sedikit, tetapi pada saat program menjadi besar, mereka cenderung susah untuk di-manage dan di-debug. Dalam usaha untuk me-manage program, struktur programming diperkenalkan cara untuk mem-break down code-code tersebut melalui functions dan procedures.

Ini adalah sebuah langkah perbaikan, namun pada saat program dijalankan dalam sebuah fungsi bisnis yang kompleks dan berinteraksi dengan sistem lain, maka kelemahan dari struktur metodologi programming muncul kepermukaan meliputi program menjadi lebih susah untuk dimaintain. Fungsi yang tersedia, susah untuk diubah tanpa harus mempengaruhi fungsi sistem secara keseluruhan.Programming tidak baik untuk team development. Programmers harus mengetahui setiap aspek bagaimana program itu bekerja dan tidak menyebabkan terisolasi usaha mereka atas aspek yang lain dari sistem.

Butuh usaha yang keras untuk menterjemahkan Business Models dalam programming models. Mungkin dapat bekerja dengan baik pada saat terisolasi tapi tidak pada saat terintegrasi dengan sistem lain.

3.3.b. Pengertian OOP (Object Oriented Programming)

OOP (Object Oriented Programming) adalah suatu metode pemrograman yang berorientasi kepada objek. Tujuan dari OOP diciptakan adalah untuk mempermudah pengembangan program dengan cara mengikuti model yang telah ada di kehidupan sehari-hari. Jadi setiap bagian dari suatu permasalahan adalah objek, objek itu sendiri merupakan gabungan dari beberapa objek yang lebih kecil lagi. Begitu juga dengan program, sebuah objek yang besar dibentuk dari beberapa objek yang lebih kecil, objek-objek itu saling berkomunikasi, dan saling berkirim pesan kepada objek yang lain.

3.3.c. Konsep OOP (Object Oriented Programming)

a. Kelas Abstrak (Class Abstraksi)

Kelas merupakan deskripsi abstrak informasi dan tingkah laku dari sekumpulan data.

Kelas dapat diilustrasikan sebagai suatu cetak biru(blueprint) atau prototipe yang digunakan untuk menciptakan objek.

Kelas merupakan tipe data bagi objek yang mengenkapsulasi data dan operasi pada data dalam suatu unit tunggal.

Kelas mendefinisikan suatu struktur yang terdiri atas data kelas (data field), prosedur atau fungsi (method), dan sifat kelas (property).

b. Enkapsulasi (encapsulation)

Istilah enkapsulasi sebenarnya adalah kombinasi data dan fungsionalitas dalam sebuah unit tunggal sebagai bentuk untuk menyembunyikan detail informasi.

Proses enkapsulasi memudahkan kita untuk menggunakan sebuah objek dari suatu kelas karena kita tidak perlu mengetahui segala hal secara rinci.

Enkapsulasi menekankan pada antarmuka suatu kelas, atau dengan kata lain bagaimana menggunakan objek kelas tertentu.

Contoh: kelas mobil menyediakan antarmuka fungsi untuk menjalankan mobil tersebut, tanpa kita perlu tahu komposisi bahan bakar, udara dan kalor yang diperlukan untuk proses tersebut.

c. Pewarisan (Inheritance)

Kita dapat mendefinisikan suatu kelas baru dengan mewarisi sifat dari kelas lain yang sudah ada.

Penurunan sifat ini bisa dilakukan secara bertingkattingkat, sehingga semakin ke bawah kelas tersebut menjadi semakin spesifik.

Sub kelas memungkinkan kita untuk melakukan spesifikasi detail dan perilaku khusus dari kelas supernya.

Dengan konsep pewarisan, seorang programmer dapat menggunakan kode yang telah ditulisnya pada kelas super berulang kali pada kelas-kelas turunannya tanpa harus menulis ulang semua kodekode itu.

d. Polimorfisme (polymorphism)

Polimorfisme merupakan kemampuan objekobjek yang berbeda kelas namun terkait dalam pewarisan untuk merespon secara berbeda terhadap suatu pesan yang sama.

Polimorfisme juga dapat dikatakan kemampuan sebuah objek untuk memutuskan method mana yang akan diterapkan padanya, tergantung letak objek tersebut pada jenjang pewarisan.

3.3.d. Karakteristik OOP (Object Oriented Programming)

Semua adalah objek.

Komputasi dilakukan dengan komunikasi antar objek. Setiap objek berkomunikasi dengan objek yang lain melalui pengiriman dan penerimaan pesan.

Sebuah pesan merupakan permintaan atas sekumpulan aksi dengan semua argumen yang diperlukan untuk menyelesaikan suatu tugas tertentu.

Setiap objek memiliki memori sendiri, yang dapat terdiri dari objek-objek lainnya.

Setiap objek adalah wakil atau representasi dari suatu kelas. Sebuah kelas dapat mewakili sekelompok objek yang sama.

Kelas merupakan kumpulan tingkah laku yang berkaitan dengan suatu objek. Jadi, semua objek yang merupakan wakil dari kelas yang sama dapat melakukan aksi yang sama pula.
Kelas-kelas diorganisasikan ke dalam struktur pohon yang berakar tunggal, yang dinamakan dengan jenjang pewarisan (inheritance hierarchy).
Setiap objek pada umumnya memiliki tiga sifat, yaitu keadaan, operasi dan identitas objek.
Operasi merupakan tindakan yang dapat dilakukan oleh sebuah objek.
Keadaan objek merupakan koleksi dari seluruh informasi yang dimiliki oleh objek pada suatu saat.
Informasi yang terkandung pada objek tersebut pada akhirnya memberikan identitas khusus yang membedakan suatu objek dengan objek lainnya.

Contoh Program:

```
class Kendaraan{
int posisi1;
int kecepatan;
int posisi2;
int pergerakan;
int getPosisi1(){
return posisi1;
}
void setPosisi1(int theposisi1){
posisi1 = theposisi1;
}
int getKecepatan(){
return kecepatan;
}
void setKecepatan(int thekecepatan){
kecepatan = thekecepatan;
}
posisi2 bergerak(){
int jarak;
int waktu;
posisi2 = getKecepatan * waktu;
}
}
class Mobil extends Kendaraan{
}
class KendaraanTestDrive{
Mobil avanza = new Mobil;
avanza.setPosisi1(30);
avanza.setKecepatan(45);
```

```
avanza.bergerak();
}
```

3.4. Logic Programming

Pemrograman Logika dalam arti pertama dan lebih luas menimbulkan beberapa implementasi, seperti yang oleh Fischer Black (1964), James Slagle (1965) dan Cordell Green (1969), yang pertanyaan-menjawab sistem dalam semangat nasihat McCarthy -taker. Foster dan Absys Elcock's (1969), di sisi lain, mungkin bahasa pertama yang secara eksplisit dikembangkan sebagai bahasa pemrograman assertional.

Logika pemrograman, dalam arti luas, penggunaan logika matematika untuk pemrograman komputer. Dalam pandangan pemrograman logika, yang dapat ditelusuri setidaknya sejauh [1958] proposal saran-taker John McCarthy, logika digunakan sebagai bahasa representasi murni deklaratif, dan teorema-prover atau model-generator digunakan sebagai pemecah masalah. Tugas pemecahan masalah dibagi antara programmer, yang bertanggung jawab hanya untuk memastikan kebenaran program dinyatakan dalam bentuk logis, dan teorema-prover atau model-generator, yang bertanggung jawab untuk memecahkan masalah efisien.

Namun, logika pemrograman, dalam arti sempit yang lebih umum dipahami, adalah penggunaan logika baik sebagai bahasa representasi deklaratif dan prosedural. Hal ini didasarkan pada kenyataan bahwa penalaran mundur teorema-prover diterapkan pada kalimat deklaratif dalam bentuk implikasi. Contoh bahasa pemrograman kelompok ini adalah Prolog (Programming in Logic)

Contoh program:

Jika B1 dan … dan Bn kemudian H
memperlakukan implikasi sebagai prosedur tujuan-pengurangan:
untuk menunjukkan / memecahkan H, menampilkan / memecahkan B1 dan … dan Bn.

Misalnya, memperlakukan implikasinya:

Jika Anda menekan tombol sinyal alarm,
maka Anda waspada pengemudi kereta dari kemungkinan darurat
sebagai prosedur:
Untuk peringatan pengemudi kereta dari darurat mungkin,
tekan tombol sinyal alarm.

Perhatikan bahwa ini konsisten dengan interpretasi BHK logika konstruktif, di mana implikasi akan diinterpretasikan sebagai solusi solusi masalah H tertentu B1 … Bn. Namun, ciri pemrograman logika adalah bahwa set formula dapat dianggap sebagai program dan mencari bukti dapat diberikan arti komputasi. Hal ini dicapai dengan

membatasi logika yang mendasari untuk sebuah fragmen “berkelakuan baik” seperti klausa Horn atau herediter Harrop formula. Lihat D. Miller et al., 1991.

Seperti dalam kasus murni deklaratif, programmer bertanggung jawab untuk memastikan kebenaran program. Tetapi karena pencarian bukti otomatis umumnya infeasible, logika pemrograman seperti umumnya dipahami juga bergantung pada programmer untuk memastikan bahwa kesimpulan yang dihasilkan secara efisien. Dalam banyak kasus, untuk mencapai efisiensi, orang perlu untuk menyadari dan mengeksploitasi perilaku pemecahan masalah dari prover-teorema. Dalam hal ini, pemrograman logika adalah sebanding dengan pemrograman konvensional menggunakan program untuk mengontrol perilaku pelaksana program. Namun, tidak seperti program-program penting konvensional, yang hanya memiliki interpretasi prosedural, program logika juga memiliki interpretasi, deklaratif logis, yang membantu untuk memastikan kebenaran mereka. Selain itu, program-program tersebut, yang deklaratif, berada pada tingkat konseptual murni lebih tinggi daripada program-program penting, dan pelaksana program mereka, karena dalil-Prover, beroperasi pada tingkat konseptual lebih tinggi daripada konvensional kompiler dan interpreter.

Beberapa bahasa dirancang mendukung lebih dari satu paradigma. Sebagai contoh C++ merupakan campuran antara bahasa imperative dan object-oriented, sedangkan Leda dirancang untuk mendukung paradigm pemrograman imperative, object-oriented, functional, dan logic.

Masing-masing paradigma tersebut mempunyai strategi analisa yang khusus untuk memecahkan persoalan. Setiap paradigma mempunyai kekurangan dan kelebihan sehingga tidak semua persoalan dapat dipecahkan dengan satu jenis paradigma, sehingga diperlukan analisis secara menyeluruh terhadap persoalan yang akan diselesaikan sebelum menentukan paradigm pemrograman seperti apa yang akan digunakan untuk menyelesaikan persoalan tersebut.