BAB II
MATERI PENUNJANG

2.1. Permutasi dan Kombinasi

Teori peluang lahir dan berkembang pada permulaan abad ke-17. Ketika itu, seorang penjudi bangsawan Perancis bernama Chevalier de Mere minta pertolongan kepada Blase Pascal (1623-1662) untuk menyelesaikan masalah tentang permainan dadu. Bersama dengan Pierre de Fermat (1601-1665) mencoba untuk memecahkannya, akhirnya dari penyelesaian masalah itu muncul Ilmu Hitung Peluang. Dalam peluang dibagi menjadi beberapa cabang salah satunya mengenai permutasi dan kombinasi.

Karena luasnya cabang-cabang ilmu hitung peluang, sehingga pada sub bab 2.1., tidak akan membahas peluang secara keseluruhan tetapi hanya akan membahas tentang permutasi berulang dan kombinasi.

2.1.1. Permutasi berulang

Permutasi berulang merupakan perkembangan dari bentuk permutasi, untuk memahami pengertian permutasi berulang akan diuraikan dahulu definisi-definisi sebagai berikut:

Definisi 2.1.1.

n! = n faktorial = 1.2.3. ... . (n-1).(n).

0! = 1.

Teorema 2.1.1.

Banyaknya permutasi r unsur, yang diambil dari n unsur yang berlainan (n ≥ r) adalah

$P(n,r) = n.(n-1).(n-2). \ldots .(n-r+1) = \frac{n!}{(n-r)!}$

Bukti:

Dari n unsur yang berlainan, unsur pertama dapat diambil dari n unsur, sehingga ada n cara. Unsur kedua dapat dipilih dari n-1 unsur, karena salah satu unsurnya sudah diambil. Sehingga ada n-1 cara, hal ini dilanjutkan sampai seluruh permutasi r diambil. Jadi menurut prinsip perkalian, banyaknya permutasi r yang diambil dari n unsur yang berlainan adalah n.(n-1).(n-2). ... .(n-r+1)cara.

Dengan menggunakan definisi 2.1.1., sehingga dapat ditulis:

$P(n,r) = \frac{n!}{(n-r)!}$ ■

Lemma 2.1.1.

Banyaknya permutasi r unsur, yang diambil dari n unsur yang berlainan untuk n = r adalah

P(r,r)= r!.

Bukti:

Dari teorema 2.1.1., diketahui bahwa permutasi $P(n,r)= \frac{n!}{(n-r)!}$, dimana $n \geq r$

Jika n = r maka $P(r,r) = \frac{r!}{(r-r)!} = \frac{r!}{0!} = r!$ ■

Sebagai contoh, misalkan diketahui tiga huruf A, B, dan C. Sehingga permutasi dari ketiga huruf itu adalah:

ABC ACB BAC BCA CAB CBA .

Definisi 2.1.2.

Permutasi berulang adalah permutasi r unsur yang diambil dari n unsur yang berlainan dengan unsur yang diambil boleh ditulis berulang ($n \geq r$).

Kalau permutasi dari ketiga huruf A, B, dan C yang diambil boleh ditulis berulang, diperoleh susunan huruf seperti berikut:

AAA AAB AAC . . . BBA BBB BBC . . . CCA CCB CCC

maka permutasi ini disebut *permutasi berulang*. Dari uraian diatas maka permutasi berulang.

Teorema 2.1.2.

Banyaknya permutasi r unsur yang diambil dari n unsur yang berlainan, dengan tiap unsur yang tersedia boleh ditulis berulang ($n \geq r$)adalah n^r.

Bukti:

Kita menghitung jumlah dari cara yang dibentuk pada r unsur dengan menggunakan prinsip perkalian. Elemen pertama dipilih dengan n cara, elemen kedua juga dipilih dengan n cara dan seterusnya sampai dengan r faktor.

Maka akan ada $\underbrace{n.n.n. \ldots .n}_{r \text{ faktor}} = n^r$ cara untuk menyelesaikan masalah dari pemilihan r unsur dari himpunan yang mempunyai n unsur. ■

Dari uraian diatas maka banyaknya permutasi berulang dari tiga huruf A, B, dan C dapat ditentukan dengan cara sebagai berikut:

1. Ada tiga cara untuk memilih huruf pertama, yaitu A, B, dan C.
2. Karena ketiga huruf tersebut boleh berulang, maka ada 3 cara untuk memilih huruf kedua Berdasar kaidah perkalian atau teorema 2.1.2., banyaknya susunan itu seluruhnya adalah:

$$3 \times 3 \times 3 = 3^3 = 27$$

2.1.2. Kombinasi

Untuk memahami kombinasi, dimisalkan dari tiga huruf A, B, dan C yang akan diambil dua huruf dengan tidak memperhatikan urutannya sehingga AB = BA, AC = CA, begitu pula BC = CB. Jadi terdapat 3 pilihan yaitu AB, AC, dan

BC atau disebut kombinasi 2 unsur diambil dari 3 unsur yang berbeda.

Definisi 2.1.3.

Suatu kombinasi r unsur yang diambil dari n unsur yang berlainan ialah suatu pilihan dari r unsur tanpa memperhatikan urutannya ($n \geq r$), dan dinotasikan dengan $\binom{n}{r}$.

Banyaknya kombinasi r unsur dari n unsur yang berlainan dapat dihitung dengan teorema 2.1.3. berikut ini, yang pembuktiannya menggunakan teorema 2.1.1., dan lemma 2.1.1., sebagai berikut:

Teorema 2.1.3.

Banyaknya kombinasi r unsur dari n unsur yang berlainan ($n \geq r$) adalah

$$\binom{n}{r} = \frac{n!}{r!(n-r)!}$$

Bukti:

Banyaknya permutasi n unsur diambil r unsur adalah $P(n,r) = \frac{n!}{(n-r)!}$, disini urutan diperhatikan. Dari r unsur dapat dibuat r! permutasi, dengan prinsip perkalian

$P(n,r) = \binom{n}{r} \cdot r!$.

Menurut teorema 2.1.1.

$$P(n,r) = \frac{n!}{(n-r)!}$$, sehingga terbukti bahwa

$$\binom{n}{r} = \frac{n!}{r!(n-r)!}$$ ■

2.2. Pohon Permutasi

Definisi 2.2.1.

Suatu **graf** G adalah suatu sistem yang terdiri dari himpunan V(G) berhingga tidak kosong dari elemen-elemen yang disebut node dan himpunan E(G) yang mungkin kosong dari pasangan-pasangan tidak terurut antara dua node yang disebut jalur.

Jika u dan v adalah node-node dari G dan jalur e = uv atau e = vu adalah jalur yang menghubungkan node u dan node v.

Definisi 2.2.2.

Sebuah **lintasan** dengan panjang n-1 dalam graf G adalah susunan jalur-jalur berbentuk $v_1v_2, v_2v_3, \ldots, v_{n-1}v_n$ atau dapat pula dinotasikan $e_1, e_2, \ldots, e_i$ dengan e_i mewakili v_iv_{i+1}. Dimana $v_i \neq v_j$ untuk setiap $i,j = 1,2, \ldots, n$. Node v_1 disebut node awal dan node v_n disebut node akhir dari lintasan tersebut. Apabila lintasan mempunyai node awal dan node akhir yang sama maka disebut **cycle**.

Definisi 2.2.3.

Pohon adalah suatu graf terhubung tanpa memuat cycle.

Teorema 2.2.1.

Jika u dan v adalah node dalam pohon T, maka node u dan node v dihubungkan oleh tepat satu lintasan.

Bukti:

Andaikan T memuat dua lintasan yang menghubungkan node u dan node v, sebut saja P dan Q. Karena P dan Q adalah lintasan yang berbeda, bisa jadi ada node x (misalkan x=u) dan termasuk diantara keduanya, sehingga node yang mengikuti x pada P berbeda dengan node yang mengikuti x pada Q

Pilih y sebagai node pertama dari P yang mengikuti x dan juga termasuk di Q (misal y = v), sehingga menghasilkan dua lintasan dari node x ke node y, dimana ada dua lintasan di T. Hal ini kontradiksi dengan definisi 2.2.3., bahwa T adalah pohon. Oleh karena itu node u dan node v dihubungkan oleh tepat satu lintasan ■

Pohon permutasi merupakan nama tree yang digunakan untuk menggambarkan peletakkan ratu-ratu dengan node-node pada persoalan delapan ratu.

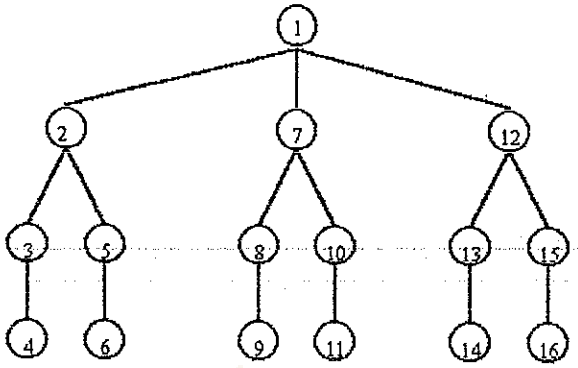
Pohon permutasi untuk persoalan delapan ratu disajikan di sub bab 3.2., yaitu pada pohon permutasi untuk semua kemungkinan dan pohon permutasi untuk semua kemungkinan dengan aturan 8 ratu.

Definisi 2.2.4.

Komponen-komponen pada pohon permutasi dapat didefinisikan sebagai berikut:

Node	:	Titik yang didalamnya berisi nomer urutan tertentu.
Node akar	:	Node yang paling atas
Node induk atau E-node	:	Node indikator untuk mencari node-node lain yang ada dibawahnya.
Node anak	:	Node yang yang dipilih oleh E-node yang secara otomatis akan berubah menjadi E-node apabila masih ada node-node yang lain dibawahnya.
Node daun	:	node terbawah yang masih dihubungkan dengan lintasan.
x[i]	:	ratu ke-i.
w	:	node ke-i yang dipilih.
e(x[i]=j)	:	jalur pada saat ratu ke-i diletakkan pada baris ke-i kolom ke-j.
Level ke-i	:	level peletakkan ratu ke-i.

Contoh dari pohon permutasi sebagai berikut:



Penjelasann lebih jauh tentang pohon permutasi ini akan diuraikan pada bab III.

2.3. Backtracking

Backtracking adalah pencarian node tujuan dengan arah pemrosesan pelacakan mundur. Backtracking dapat untuk menyelesaikan suatu permasalahan atau pencarian solusi optimal.

Nama backtracking pertama kali dikenalkan oleh D.H. Lehmer pada tahun 1950, dengan orang pertama yang mempelajari prosesnya adalah R.J. Walker yang dikembangkan dalam algoritmanya, dan penerapan dalam aplikasi dikembangkan oleh Golomb dan Baumert.

Backtracking dimulai dari node akar, kemudian dibuat tingkat yang berikutnya dengan cara mencari semua kemungkinan node pada node yang ada dibawahnya.

Definisi 2.3.1.

Pohon dengan n-node adalah suatu pohon yang memuat node-node

$v_1, v_2, v_3, \ldots, v_n$

dimana

v_i = node ke-i untuk $1 \leq i \leq n$.

Teorema 2.3.1.

Jika jalur e dari sebuah graf G yang menggabungkan node induk dan node anak, e = uv dan u adalah node yang dipilih sebelum v pada backtracking di dalam graf G, maka v adalah node anak.

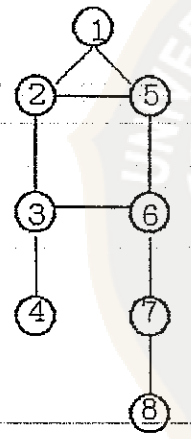
Bukti:

Dengan hipotesa, v adalah node yang dipilih pertama kali setelah u dipilih. Subgraf dari pelacakkan mundur ditunjukkan oleh semua node pada G, yang dikunjungi dari yang pertama hingga sampai yang terakhir oleh u adalah pohon T. Akar adalah u karena setiap node yang berdekatan dengan u, dipilih oleh u adalah node dalam T dan u lebih dahulu yang dipilih sebelum v, sehingga u sebagai node induk dengan node yang berada dibawahnya sebagai node anak. ■

Berdasarkan teorema 2.3.1., dan definisi 2.3.1., alur dari backtracking sebagai berikut:

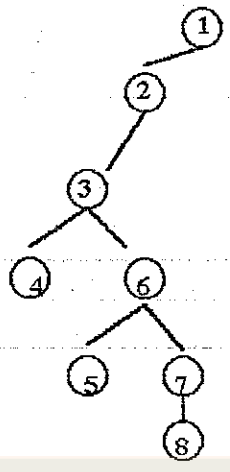
1. w = v_1, v_1 sebagai node akar sekaligus sebagai node induk (u).
2. w = v_i node ke i dengan i yang paling minimum yang dipilih . Chek apakah elemen yang terakhir dipilih merupakan node tujuan.
   a. jika ya → selesai
   b. jika tidak → chek apakah
      1. jika i < n maka kembali ke langkah 2
      2. jika i = n maka ke langkah 3
3. [backtrack] w = u kembali ke langkah 2

Contoh:



Jalur yang ingin dicari dari 1 menuju ke 8

Sehingga dalam satu kasus node induk, node anak, dan backtrackingnya dapat digambarkan sebagai berikut:



Keterangan:

1. v_1 sebagai satu-satunya node yang aktif, sehingga secara otomatis akan menjadi node akar karena menjadi node teratas, sekaligus menjadi node induk yang bertugas mencari node-node yang ada dibawahnya, kemudian diambil salah satu node, Misal v_2, karena v_2 dipilih oleh v_1 berdasarkan teorema 2.3.1., v_2 menjadi node anak.
2. v_2 menjadi node terujung yang bertugas mencari node-node yang

ada dibawahnya (node anak) sehingga v2 menggantikan tugas v1 sebagai node induk.

3. v2 memilih v3, v3 sebagai node induk memilih v4, karena tidak ada node yang menghubungkan untuk menuju ke v8 maka akan backtrack kembali ke v3.
4. v3 kembali sebagai node induk, kemudian memilih v6.
5. v6 sebagai node induk memilih v5, karena tidak ada jalur yang menghubungkan v8, maka akan backtrack kembali ke v6.
6. v6 sebagai node induk memilih v7, kemudian v7 sebagai node induk memilih v8.

2.4. Algoritma

Algoritma adalah sekumpulan instruksi yang menjelaskan langkah-langkah yang teratur dan berhingga yang menggambarkan aktivitas penyelesaian suatu masalah. Sehingga dengan langkah yang terperinci dan terurut diharapkan akan mempermudah pembuatan logika program sebagai dasar penulisan program sehingga lebih mudah dibaca dan dikomunikasikan dengan orang lain.

Algoritma yang baik memenuhi kriteria sebagai berikut:

1. Algoritma tersusun dari sejumlah langkah operasional yang terurut.
2. Susunan langkah operasional algoritma mempunyai awal dan akhir.

3. Urutan langkahnya jelas, tidak berbelit-belit dan terarah.
4. Algoritma mempunyai tujuan yang jelas, tepat guna dan efektif dalam memecahkan suatu masalah.
5. Kompatibel terhadap semua bahasa pemrograman.

Cara menyatakan algoritma

1. Dengan bahasa alamiah (Natural Language).
   Algoritma ditulis dengan bahasa ungkapan bahasa sehari-hari
2. Dengan sandi semu (Pseudocode).
   Algoritma yang telah ditulis dengan menggunakan perintah bahasa pemrograman tertentu ditambah dengan bahasa alamiah.
3. Dengan flowchart (Bagan Alir).
   algoritma ditulis dalam bentuk simbol/bentuk diagram.

2.5. Rekursif

Suatu obyek dikatakan rekursif apabila obyek perulangannya merupakan bagian dari dirinya sendiri atau obyek awal merupakan bagian dari obyek perulangan. Rekursif tidak hanya muncul di dunia matematika, tetapi juga dalam kehidupan sehari-hari, seperti pada gambar iklan yang berisi dirinya sendiri.

Rekursif sangat penting dalam matematika. Beberapa contoh antara lain seperti:

1. Bilangan natural
   a. 1 adalah bilangan natural
   b. Bilangan berikutnya dari bilangan natural adalah bilangan natural.
2. Fungsi faktorial n!(untuk bilangan bulat positif)
   a. 0!= 1
   b. Jika n > 0 maka n! = n .(n-1)!
3. Stuktur pohon
   a. 0 adalah pohon (disebut pohon kosong)
   b. Jika t_1 dan t_2 adalah pohon maka bisa dibentuk suatu pohon apabila mempunyai node akar yang sama dan node node-node pada t_1 dan t_2 tidak ada yang sama.

Algoritma rekursif hanya sesuai apabila suatu persoalan yang akan diselesaikan, atau nilai fungsi yang akan dihitung, atau struktur data yang akan diproses sudah dinyatakan dalam bentuk rekursif. Secara umum, program rekursif dapat dinyatakan sebagai komposisi ρ dari statemen dasar S_i(yang tidak berisi P) dan P itu sendiri sebagai obyek awal.

$$P = \rho\ [S_i, P] \quad . \quad . \quad . \quad . \quad . \qquad (2.4.1.)$$

Piranti yang perlu dan cukup untuk menyatakan program rekursif adalah prosedur atau function. Sudah merupakan hal yang biasa untuk menggunakan sejumlah

parameter lokal dalam prosedur, yaitu sekumpulan peubah, konstanta, tipe, dan prosedur yang didefinisikan secara lokal ke procedur ini dan tidak dikenal di luar prosedur tersebut. Setiap kali prosedur itu diaktifkan secara rekursif, himpunan peubah lokal yang baru akan dibagun. Meskipun mereka mempunyai nama yang sama dengan himpunan peubah lokal dari pemanggilan sebelumnya, nilai-nilai mereka berbeda satu sama lain.

Seperti statemen perulangan, prosedur rekursif juga memungkinkan adanya komputasi yang tidak pernah berakhir, sehingga dalam hal inipun perlu diperhatikan adanya kondisi untuk menghentikan proses eksekusi program. Persyaratan dasarnya jelas bahwa pemanggilan rekursif ke procedur P harus memperhatikan batasan yang ditentukan dimisalkan q yang pada suatu ketika harus bernilai salah. Dengan demikian, algoritma rekursif dapat dinyatakan dengan formal, yaitu :

$$P \equiv \text{if } q \text{ then } \rho\ [S_i, P] \quad \text{atau} \ldots\ldots \qquad (2.4.2.)$$

$$P \equiv \rho\ [S_i, \text{if } q \text{ then } P] \ldots\ldots \qquad (2.4.3.)$$

Teknik dasar yang digunakan untuk mendemonstrasikan bahwa perulangan harus berakhir adalah dengan mendefinisikan fungsi f(x) (x adalah sekumpulan peubah dalam program), sedemikian rupa dan dapat membuktikan bahwa f(x) akan berkurang nilainya pada setiap perulangan. Dengan cara yang sama, penghentian program

rekursif dapat dibuktikan dengan memperlihatkan bahwa setiap eksekusi P mengurang nilai f(x). Cara untuk menyakinkan adanya penghentian adalah menggunakan (nilai) parameter, misalnya n, dalam P dan secara rekursif memanggil P dengan n-1 sebagai nilai parameter. Penggantian kondisi q dengan n>0 akan memberikan garansi proses rekursif akan berakhir. Hal ini dapat dinyatakan dengan:

P ≡ if n>0 then ρ [S_i, P(n-1)] atau . . . (2.4.4.)

P ≡ ρ [S_i, if n>0 then P(n-1)] . . . (2.4.5.)

Pada waktu kondisi n=0 rekursif akan berhenti, dan P(0) adalah harga awal, sehingga harga awal diperlukan untuk proses rekursif.

2.6. Bahasa Pemrograman

Sebelum kita membahas bahasa pemrograman Pascal terlebih dahulu akan dibahas pengertian dari program.

2.6.1. Program

Program adalah seperangkat perintah/ langkah-langkah yang memerintahkan komputer secara tepat bagaimana menangani suatu masalah lengkap dengan penyelesaiannya yang ditulis dalam bahasa program komputer.

Program terdiri dari instruksi-instruksi yang dikontrol dan operasi-operasi pada data yang teratur yang ditujukan ke komputer.

Ada beberapa hubungan yang penting dalam rancangan program baik yaitu :

1. Tahan Uji.

   Program harus dapat menyelesaikan masalah-masalah yang disajikan serta dapat mendeteksi kemungkinan kesalahan pemakaian (oleh operator/pengguna).

2. Perawatan.

   Program harus dapat diubah dan dimodifikasi dengan mudah jika diperlukan.

3. Keluwesan.

   Program harus dapat ditransfer ke komputer yang berbeda-beda dengan modifikasi minimum.

4. Ramah dan mudah dimengerti.

   Program harus dapat dibaca dan dimengerti baik oleh programmer maupun oleh pemakainya.

5. Penampilan.

   Program harus efisien dan cepat.

6. Penyimpanan.

   Program harus tersimpan dengan baik sehingga sewaktu-waktu dapat digunakan lagi dan hemat dalam pemakaian ruang penyimpanan(memori).

Tahapan-tahapan pembuatan program adalah sebagai berikut:

1. Pemahaman masalah.

Programmer harus tahu persis untuk apa program dibuat. Biasanya diketahui dari spesifikasi masukan, pemrosesan dan keluaran yang diinginkan.

2. Merencanakan metode yang digunakan untuk menyelesaikan masalah.

   Perencanaan tergantung pada besar kecilnya suatu permasalahan. Program yang besar dapat dibuat oleh beberapa programmer dan setiap programmer membuat sebagian program secara terpisah setelah itu disatukan dan ditest secara keseluruhan. Proses ini dinamakan dengan integrasi.

3. Penulisan program/pengkodean.

   Tahapan penulisan program dapat dikatakan berhasil setelah adanya konsep pemecahan masalah yang berupa algoritma. Pengkodean tidak harus selesai menjadi program yang besar dalam suatu kesempatan, tetapi dilakukan langkah demi langkah yang diselingi dengan proses pelacakan kesalahan (debugging).

4. Pengujian dan proses pembetulan kesalahan.

   Kebenaran dari suatu program harus ditest terlebih dahulu sebelum dieksekusi. Pengetesan terhadap kebenaran sintax penulisan akan dilaksanakan oleh komputer itu sendiri. Sedangkan pengetesan terhadap kebenaran algoritma merupakan tipe kesalahan yang sulit dilacak, kesalahan algoritma tidak dapat dilacakn oleh compiler.

Ciri kesalahan algoritma adalah program dapat dijalankan namun memberikan hasil yang salah.

5. Dokumen program.

Hasil kerja programmmer dalam memghasilkan suatu program sangat penting untuk didokomentasikan secara lengkap. Program yang baik mempunyai dokumen pemrograman secara lengkap dan terpelihara. Tujuan dokumen adalah memberikan informasi penting yang berkaitan dengan pemrograman.

2.6.2. Bahasa Pemrograman Pascal

Pascal merupakan bahasa tingkat tinggi (high level language) yang orientasinya pada segala tujuan, dirancang oleh prof. Niklous Wirth dari Technical University di Zurich, Switzerland.

Namun kata pascal diambil sebagai penghargaan terhadap Blaise Pascal, ahli matematik dan Philosoph terkenal abad 17 dari Perancis.

Prof Wirth memperkenalkan compiler bahasa pascal pertama kali untuk computer CDC 6000 (Control data Corporation), yang dipublikasikan pada tahun 1971 dengan tujuan untuk membantu mengajar program komputer secara sistematis khususnya untuk memperkenalkan pemrograman yang terstruktur (Structured Programming). Jadi Pascal merupakan bahasa yang ditujukan untuk membuat program terstruktur.

Dalam waktu yang singkat Pascal telah menjadi bahasa yang populer di kalangan pelajar universitas dan merupakan bahasa yang diajarkan di beberapa perguruan tinggi. Dalam menyusun suatu program seorang pemrogram harus2 mengetahui karakter, pengenal, dan bentuk umum program Pascal seperti berikut:

a. Karakter-karakter Yang Diperlukan

Bahasa Pascal mempunyai karakter A sampai Z, baik huruf besar maupun huruf kecil, karakter 0 sampai 9, dan simbol-sibol khusus seperti +,-,(),{},[].

b. Pengenal

Pengenal adalah nama yang diberikan kepada suatu elemen program. Elemen tersebut bisa berupa konstanta, variabel, nama fungsi, nama prosedur maupun suatu nama program. Pengenal dapat disusun dari karakter huruf, maupun karakter bilangan dengan beberapa aturan yang harus dipatuhi.

Aturan-aturan pada pengenal adalah;

1. Nama pengenal harus dimulai dengan karakter huruf.
2. Karakter kedua dan seterusnya bisa berupa kombinasi angka dan huruf, baik huruf besar maupun huruf kecil tetapi tiodak boleh memakai karakter seperti ? , (blank), # dan sebagainya.

Sebagai penghubung kata dapat digunakan karakter "_" (garis bawah).

3. Beberapa nama sudah digunakan secara khusus oleh Pascal untuk maksud-maksud tertentu, sehingga tidak boleh lagi dipakai sebagai pengenal. Nama-nama ini disebut sebagai kata cadangan (reserved word). Nama-nama ini tidak perlu dihafal karena jika kebetulan digunakan dalam program sebagai nama pengenal, maka pada waktu pemrosesan oleh kompiler hal itu akan diberitahukan sebagai kesalahan.

Tabel-1. Kata-kata Cadang

and	end	nil	string
array	file	not	set
begin	for	of	then
case	function	or	to
const	go to	packed	type
div	if	procedure	until
do	in	program	var
down to	label	record	while
else	mod	repeat	with

5. Beberapa nama yang disebut pengenal standar juga telah mempunyai arti khusus, tetapi didefininisikan kembali, maka mereka juga dapat digunakan sebagai pengenal. Jika pengenal standar ini digunakan sebagai pengenal biasa maka arti khususnya tidak akan digunakan.

Tabel_2. Pengenal Standard

Konstanta	:	false, true, max.
Tipe	:	boolean, char, real, integer.
Fungsi	:	abs, arctan, cis, eof,eoln, exp,ln, odd, ord, pred, round, sin, sqr, sqrt, succ, trunc.
Var	:	pengenal yang nilainya dapat berubah pada waktu pemrosesan program. Semua variabel yang akan digunakan dalam program harus didekalarasikan terlebih dahulu di dalam program, deklarasi variabel dilakukan dengan deklarasi var

c. Bentuk Umum Program Pascal

Program dalam bahas Pascal mempunyai bentuk atau struktur tertentu. Bentuk ini harus dipenuhi agar program tidak disalahkan oleh kompiler. Kompiler yang dimaksud adalah penterjemah program Pascal ke dalam bahasa mesin yaitu sistem Turbo Pascalnya itu sendiri.

Bentuk umum program Pascal adalah sebagai berikut:

```
PROGRAM     Nama_program;
CONST       Deklarasi_konstanta;
TYPE        Deklarasi_tipe_data;
VAR         Deklarasi_variabel;
FUNCTION    Deklarasi_fungsi;
PROCEDURE   Deklarasi_prosedur;

BEGIN    ---(Program Utama)---
  Statemen;
  . . . .
END.
```

Secara ringkas struktur suatu program Pascal terdiri dari:

1. Judul Program
2. Blok Program
   a. Deklarsi
      Deklarasi konstanta.
      Deklarasi tipe data.
      Deklarasi variabel.
      Deklarasi fungsi.
      Deklarasi prosedur.
   b. Bagian Pernyataan.

Bagian pernyataan merupakan bagian yang terakhir dari suatu blok program. Bagian ini diawali dengan sebuah kata cadang BEGIN dan diakhiri dengan kata cadang END diikuti dengan tanda "." , pernyataan menunjukkan suatu tindakan yang akan dikerjakan oleh program. Tindakan yang dilakukan oleh program tergantung dari instruksi-instruksi yang diberikan.