BAB I
PENDAHULUAN

1.1. LATAR BELAKANG

Dalam menyambut era perkembangan teknologi informasi, dituntut sumber daya manusia yang memilki kemampuan dalam menganalisis dan memecahkan setiap permasalahan berbasis komputerisasi. Beberapa tahun terakhir telah diadakan Olimpiade Sains Nasional (OSN) yang di antaranya terdapat bidang Komputer/Informatika untuk menjaring tunas-tunas baru yang akan tergabung dalam Tim Olimpiade Komputer Indonesia.

Dalam lomba *programming* ada beberapa kriteria penilaian, salah satunya *runtime* program. Program yang efektif salah satunya adalah memiliki *runtime* yang relatif singkat. *Runtime* adalah waktu atau periode yang dibutuhkan dalam mengeksekusi program. (Nugroho, 2004)

Hal-hal yang mempengaruhi *runtime* adalah jumlah input yang diproses dan beberapa faktor lain : kecepatan mesin, sistem operasi, kualitas *compiler* dan bahasa pemrograman. (Wirth, 2004)

Adapun cara untuk menyiasati kendala *runtime* dari jumlah input yang diproses adalah penggunaan tipe variabel, sehingga ketika eksekusi program sesuai dengan batas *runtime* yang ditentukan.

Contoh kutipan badan program yang menggunakan tipe variabel/*field* yang tidak sesuai:

type

MHS = **record** { data mahasiswa}

NIM : **integer**;

nama : **string**;

usia : **integer**;

jenkel : **char [20]**; { tipe field data yang tidak pas}

end;

{Program akan kurang efisien saat di-*compile*}

Dari kutipan contoh program di atas terdapat ketidaksesuaian tipe variabel/*field* yakni **char[20]**, seharusnya nilai *field*-nya **char[1]** karena dalam menentukan jenis kelamin bisa ada dua pilihan yaitu laki-laki "L", atau perempuan "P".

Agar lebih mengerti dengan jelas mengenai penggunaan efek tipe variabel dalam pembuatan program, maka penulis mengambil topik **" Analisis efek tipe variabel terhadap kecepatan running program study kasus jawaban soal olimpiade komputer"**.

1.2. PERUMUSAN MASALAH

Semakin ketatnya persaingan dalam lomba *programming* komputer tentunya mengharuskan untuk merumuskan suatu *runtime* yang tepat. Dalam menetapkan *runtime* yang sesuai, maka dalam penulisan skripsi ini penulis merumuskan masalah sebagai berikut:

Bagaimana pengaruh beberapa tipe variabel terhadap kecepatan *runtime* program?

1.3. BATASAN MASALAH

Dalam penelitian ini, agar sesuai dengan rumusan masalah yang telah ditentukan. Penulis memberikan batasan yaitu:

1. Dalam penelitian ini tidak membuat aplikasi, melainkan menguji aplikasi yang sudah ada dari sisi *runtime* program.

2. Sampel yang akan diuji adalah lima *coding* program utuh dari jawaban soal olimpiade komputer yaitu program analisa_ekspresi_aljabar, mencari_massa, mencari_multipalindrom_pada_suatu_tulisan, mencari_beda_tertinggi_bukit mengevaluasi_ekspresi_aljabar.
3. Sesuai dengan lima program yang akan diuji, maka tipe data yang digunakan hanya tipe data bilangan bulat yaitu : ShortInt, Integer, LongInt, Byte, Word.

1.4. TUJUAN

1. Untuk menganalisis aspek *runtime* program pada penggunaan beberapa tipe variabel.
2. Untuk mengetahui pengaruh penggunaan beberapa tipe variabel terhadap *runtime.*

1.5. MANFAAT

1. Manfaat bagi peneliti:

Dalam jangka panjang hasil riset dapat digunakan sebagai dasar pengambilan keputusan dalam usaha menyiasati masalah *runtime.*

2. Bagi mahasiswa lainnya

Untuk membantu mahasiswa lainnya yang ingin mengetahui bagaimana mengetahui perbedaan efek tiap variabel terhadap *runtime* program dan untuk membantu mahasiswa lainnya yang ingin membuat karya tulis tentang *programming*.

3. Bagi masyarakat luas

Untuk memberikan wawasan dan pandangan yang positif bagi masyarakat pada umumnya, serta pelajar pada khususnya mengenai dunia pemrograman komputer.

BAB II
TINJAUAN PUSTAKA

2.1 Konsep tipe data dan variabel

Dalam program terjadi pengolahan data menjadi informasi. Pengguna memberi masukan untuk kemudian diolah dan dimunculkan kembali kepada pengguna. Contohnya, program menghitung luas persegi panjang. Pengguna memasukkan nilai panjang dan lebar. Selanjutnya di-*Enter* keluarlah luas yang menjadi hasil perkalian panjang dan lebar. (Munir, 2007)

Program menggunakan memori komputer sebagai media penyimpanan data dan informasi. Data yang dimasukkan melalui *keyboard* akan dibaca oleh program lalu disimpan di memori (RAM). Hasil pengolahan juga dapat disimpan terlebih dulu di memori untuk kemudian ditampilkan di layar monitor. (Munir, 2007)

Dalam pengelolaan memori (sebagai media penyimpanan) digunakanlah konsep tipe data dan variabel. Memori dianalogikan sebagai rak lemari dengan slot-slot kecil penyusunnya yang berurutan dari bawah ke atas. Ada slot yang besar, ada pula yang kecil. Konsekuensinya, besar kecil isi yang bisa dimasukkan ke dalam slot tersebut tergantung pada besar kecilnya ukuran. Setiap slot diberi label yang berbeda antara satu sama lain. Tujuannya memudahkan pengaksesan dan tidak terjadi pengisian oleh data lain. (Munir, 2007)

Apabila rak adalah memori maka slot adalah bagian-bagian memori. Jenis tipe data mempengaruhi besar kecilnya ukuran slot. Contoh tipe data integer 2 byte, tipe data float 4 byte dan tipe data char memiliki besar 1 byte. Sedangkan label pada slot menandakan nama variabel dalam pemrograman. Sehingga dalam

program diawali dengan nama-nama variabel beserta jenis tipe datanya. Biasanya dikenal dengan deklarasi variabel, yang diterjemahkan oleh *compiler* untuk menyiapkan memori sebelum digunakan oleh program. (Munir, 2007)

Variabel adalah suatu pengenal yang *programmer* definisikan untuk menyimpan nilai tertentu dalam program pada saat program sedang berjalan *(runtime)*. Nilai tersebut juga dapat diubah pada saat program sedang berjalan sesuai dengan kebutuhan.

Deklarasi variabel membutuhkan tipe data, fungsinya untuk memberitahu *compiler* bahwa variabel tersebut digunakan untuk menyimpan nilai dengan tipe data bersangkutan. (Munir, 2007)

Konsep variabel dalam pemrograman ibarat konsep variabel dalam matematika. Misalnya, 1 = x < 5 | x ? R artinya x adalah variabel bertipe real (domain x adalah bilangan real), dengan range (jangkauan) 1 sampai 5.

Contoh:

```
Program namaprogram;

var
nama_variabel : tipevariabel;
        nama_variabel2 : tipevariabel2;
begin
(*bagian program utama*)
end.  (Nugroho, 2004)
```

2.2 Tipe data dalam pascal

a. Tipe bilangan bulat

Tabel 2.1 Pembagian tipe bilangan bulat

Tipe data	Ukuran (dalam byte)	Rentang nilai
ShortInt	1	-128 sampai 127
Integer	2	-32.768 sampai 32.767
Longint	4	-2.147.483.648 sampai 2.147.483.647
Byte	1	0 sampai 255
Word	2	0 sampai 65.535

(Raharjo, 2008)

Apabila nilai yang dimasukkan melebihi rentang maksimum, maka nilainya akan dikembalikan ke keadaan minimum, kemudian ditambah dengan sisa yang ada. Sebagai contoh, variabel x yang bertipe shortint. Melalui suatu perhitungan di dalam program, variabel x ternyata harus menyimpan nilai 200, sedangkan rentang maksimum tipe shortint 127. Hal ini akan menyebabkan nilai dari variabel x menjadi -56, yang berasal dari perhitungan 200-256. Angka 256 merupakan besarnya rentang antara -128 sampai dengan 127. (Raharjo, 2008)

Integer memiliki representasi nilai dari serangkaian bilangan biner. Komputer memproses per blok bit yang umumnya terdiri dari 8 bit (1 byte). Byte yaitu integer 8 bit yang bisa menampung tipe bilangan asli, dan shortint yaitu tipe integer 8 bit yang menyimpan bilangan bulat.

Ukuran 8 bit artinya untuk menyimpan tipe tersebut diperlukan memori sebesar 8 bit, nilai yang bisa disimpan adalah 0000 0000 sampai dengan 1111 1111 atau 0 sampai dengan 255. Untuk bisa menyimpan nilai negatif dalam biner biasanya

digunakan kebalikan dari representasi desimal ditambah dengan 1. Misalnya, menyimpan -5 desimal dengan merepresentasikan 5 yaitu 0000 0101, dibalik (*not*-kan) menjadi 1111 1010, dan ditambah 1 menjadi 11111011, sehingga -5 dalam biner adalah 11111011. Dengan cara ini maka nilai yang bisa ditampung dalam 8 bit adalah -128 sampai dengan 127. (Nugroho, 2004)

Operasi dasar untuk integer yaitu kali (disimbolkan dengan *), tambah (+), dan kurang (-). Misal, 5 + 5 = 10, 2 * 3 = 6, dan 3-2 = 1. Operasi pembagian (div) memberikan hasil pembagian yang dibulatkan, sehingga 4 div 2 = 2, dan 5 div 2 = 2. Operasi mod memberikan sisa dari hasil bagi sehingga 5 mod 2 = 1, dan 2 mod 5 = 2 (2/5 jika dibulatkan adalah nol, sisanya adalah 2 – 0* 5 = 2).

(Nugroho, 2004)

b. Tipe data real

Tabel 2.2 Pembagian tipe data real

Tipe data	Ukuran (byte)	Digit penting	Rentang nilai
Real	6	11 sampai 12	$2.9x10^{-39}$ sampai $1.7x10^{38}$
Single	4	7 sampai 8	$1.5x10^{-45}$ sampai $3.4x10^{38}$
Double	8	15 sampai 16	$5.0x10^{-324}$ sampai $1.7x10^{308}$
Extended	10	19 sampai 20	$3.4x10^{-4932}$ sampai $1.1x10^{4932}$
Comp	8	19 sampai 20	$-9.2x10^{18}$ sampai $9.2x10^{18s}$

(Raharjo, 2008)

Tipe ini digunakan untuk merepresentasikan bilangan *floating point* atau bilangan yang mengandung angka dibelakang koma. Operasi terhadap real yaitu: tambah (+), kali (*), minus (-) (sama seperti integer), dan pembagian (memakai simbol "/" yang menghasilkan bilangan real).

(Nugroho, 2004)

c. Tipe string

Tipe string merupakan kumpulan dari karakter yang terangkai menjadi satu kata ataupun kalimat, misalnya 'Pemrograman Pascal', 'Informatika', 'Bandung'.

Contoh operasi penambahan (konketenasi) string:

```
program concat_string;
var s1, s2, s3 : string
begin
        s1:= 'hello';
        s2:='world';
        s3:= s1 + ' '+ s2;
writeln (s3);
end.
```

Nilai 'hello world' adalah gabungan string s1, spasi, dan string s2. Sebuah karakter juga bisa digabung dengan sebuah karakter:

```
var
        s:string;
        c:char
begin
        writeln ('Masukkan sebuah huruf:'); readln(c);
        s:= 'Huruf yang Anda masukkan adalah '+c;
        writeln(s);
end.
```

(Nugroho, 2004)

d. *Tipe data boolean*

Boolean adalah tipe data yang memiliki nilai *true* (benar) dan *false* (salah). Tipe ini diperlukan dalam kondisi perulangan dan kondisional (menggunakan if).

Misalnya, 6 > 5 maka nilai ekspresi tersebut adalah *true* karena 6 lebih besar dari 5. Sedangkan, 6 < 2 maka nilai ekspresi *false.*

Dalam tipe boolean nilai 1 untuk *true* dan nilai 0 untuk *false*. Tipe boolean tidak bisa dioperasikan sebagai integer, ini berlaku hanya dalam bahasa Pascal. Namun, bilangan integer bisa dioperasikan dengan boolen dioperasikan melalui representasi bit pada bilangan integer, dan melakukan operasi boolean yang bersesuain terhadap bit (bit 1 untuk *true* , dan 0 untuk *false).* Dalam bahasa lain misalnya bahasa C, dua tipe ini bisa dipertukarkan. (Nugroho, 2004)

Contoh : 1 or 2 = 3

representasi biner untuk 1 desimal = 00000001

representasi biner untuk 2 desimal = 00000010

jika or-kan masing-masing bit hasilnya 00000011yang bernilai 3 desimal

Operator dalam boolean yaitu and, or, xor .

and menghasilkan *true*, jika keduanya bernilai *true*

or menghasilkan *true*, jika salah satunya bernilai *true*

xor menghasilkan nilai *true*, jika kedua nilai boolean berbeda

(Nugroho, 2004)

e. Tipe data karakter

Sebuah karakter direpresentasikan dengan bilangan integer 8 bit yang bernilai dari 0 sampai 255. Setiap nilai diberi simbol, misalnya nilai 65 dengan simbol A, 66 dengan simbol B, dan seterusnya.

Untuk yang bernilai 0-127 biasanya disebut dengan karakter ASCII (*American Standard Code For Information Interchange*). Dalam bahasa Pascal, terdapat fungsi ord untuk memetakan sebuah karakter menjadi nilainya, dan fungsi chr untuk melakukan proses kebalikannya.

Dalam tipe karakter tidak bisa dilakukan proses penjumlahan atau pengurangan. Tipe ini hanya bisa dioperasikan jika dikonversi menjadi integer dengan fungsi ord. (Nugroho, 2004)

2.3 Kompatibilitas tipe

Tipe data string tidak bisa dioperasikan dengan tipe integer. Pun halnya, dengan tipe data real tidak bisa langsung diisi dengan tipe data integer, karena mungkin dalam tipe data real mengandung pecahan.
Contoh penggunaan tipe data dengan banyak variabel:

```
a : integer ;
b : integer ;
d : integer ;
c : real ;
begin
        a :=1 ;
        b :=2 ;
        c :=a/b ;
        d :=a/b ;
end.
```

Operasi a/b menghasilan tipe data real, sehingga perintah terakhir d:=a/b tidak bisa di-*compile*. (Nugroho, 2004)

2.4 Konversi antar tipe data

Konversi antar beberapa tipe bisa dilakukan, contohnya konversi dari integer ke byte, real ke integer. Sebagian konversi melalui proses yang disebut *casting*. Contoh *casting* dengan shortint menjadi byte, atau byte menjadi integer, Namun, untuk tipe data real tidak bisa di-*cast* menjadi integer.
Konversi integer ke real bisa dilakukan. Sedangkan, dalam konversi real ke integer bisa menghasilkan *error*, karena ada beberapa yang dilakukan, apakah hasilnya akan dibulatkan ke atas atau ke bawah, atau bilangan di belakang koma semua dihilangkan.

Konversi antara yang *range*-nya lebih besar ke *range*-nya yang lebih kecil akan menghasilkan nilai yang tidak bisa diprediksi. Misalnya i adalah sebuah integer, dan b adalah sebuah byte, konversi dari i ke b (integer ke byte) akan

menghasilkan hasil yang benar jika i memiliki nilai antara 0 sampai 255, jika i memiliki nilai di luar itu maka nilai b tidak bisa diprediksi.
(Nugroho, 2004)

Pemeranan tipe data (*typecasting*) adalah menganggap suatu variabel yang memiliki tipe data tertentu ke tipe data lainnya yang kompatibel. Dengan kata lain, *typecasting* dapat dianggap sebagia proses konversi tipe dat. Secara umum proses ini dibagi menjadi dua, yaitu:

a. Konversi implisit. Konversi implisit berarti proses konversi secara otomatis dilakukan kompiler. Sebagai contoh, pada saat menjumlahkan bilangan bulat dan bilangan rill, maka kompiler secara otomatis akan melakukan konversi terhadap bilangan bulat tersebut menjadi bilangan riil. Perhatikan contoh kode berikut ini.

```
var
        a: integer;
        b, c : real;
begin
        a := 2;
        b := 3.56;
        c :=  a + b ; { a secara otomatis akan dikonversi ke tipe real}

        .....
end.
```

Pada kode diatas, bilangan 2 akan dikonversi menjadi 2.00. (Nugroho, 2004)

b. Konversi eksplisit. Berbeda dengan konversi implisit, disini konversi dilakukan sendiri secara eksplisit. Sebagai contoh, variabel a yang bertipe char dan ingin dianggap variabel tersebut sebagai tipe integer, maka dapat dilakukan *typecasting* dengan cara berikut:

```
integer (a);
```

Untuk lebih jelasnya, perhatikan contoh kode di bawah ini:

```
var
        a : char;
        i : integer;
begin
```

```
        a :=  'A';          {Mengisikan variabel a dengan karakter 'A'}
        i := integer (a); { Menganggap a sebagai variabel integer dan menyimpan
                                -nya ke dalam varaibel I}

        ….
end.
```

Dari kode tersebut, varaiabel i akan bernilai 65, yang merupakan kode ASCI dari karakter 'A'. Sebaliknya, apabila ingin melakukan konversi variabel I ke tipe char, maka akan menuliskannya seperti berikut:

```
char (i);
```

Sebagai catatan, tidak semua tipe data dapat dilakukan konversi ke tipe data lain. Sebagai contoh, tidak dapat dilakukan konversi data dari real ke tipe char, begitu juga sebaliknya. Bahasa Pascal memiliki dua buah prosedur yang sering sekali digunakan untuk melakukan konversi tipe data antara numerik dengan string, yaitu prosedur Val dan Str.

(Raharjo, 2008)

Prosedur Val

Prosedur ini digunakan untuk melakukan konversi dari tipe numerik ke tipe string. Berikut ini prototipe dari prosedur ini:

```
procedure val (S; var V ; var Code ; integer);
```

di mana S adalah string yang akan dikonversi ke tipe numerik. Hasil konversi akan disimpan ke dalam variabel V. maka variabel Code akan bernilai 0, jika tidak maka varaiabel Code bernilai selain 0. Perhatikan contohnya di bawah ini:

```
function StrToNumber ( const s : string) : real;
var
        hasil ; real;
        kode: integer;
begin
        val(s, hasil, kode);
        if (kode = 0) then begin
                StrToNumber := hasil;
        end;
end.
```

Fungsi StrToNumber dapat digunakan untuk melakukan konversi dari tipe string ke numerik. Berikut ini contoh pemanggilan fungsi tersebut.

```
var
        n : real;
begin
        n := StrToNumber ( '3.14');
        ….
end.
```

(Raharjo, 2008)

Sampai di sini, variabel n akan bernilai 3.14 (bukan string '3.14') sehingga varaiabel n tersebut dapat digunakan digunakan dalam operasi numerik.

Prosedur Str

Prosedur ini digunakan untuk melakukan konversi dari tipe numerik ke tipe string. Adapun prototipe dari prosedur ini sebagai berikut:

procedure Str (X; var S);

dimana X merupakan tipe numerik yang akan dikonversi. Hasil konversi akan disimpan ke dalam varaiabel S. Berikut contohnya:

```
{fungsi untuk melakukan konversi dari bilangan bulat ke string}
function IntToStr (x: longinr) : string;
var
        hasil : string;
begin
        str (x, hasil);
        IntToStr := hasil;
end;

{fungsi untuk melakukan konversi dari bilangan riil ke string}
function FloatToStr ( x: real) : string;
var
        hasil : string;
begin
        str (x, hasil);
        IntToStr := hasil;
end;
```

(Raharjo, 2008)

Adapun contoh penggunaan dua buah fungsi tersebut sebagai berikut:

```
var
```

```
        sInt, sFloat : string;
begin
        sInt := IntToStr (100); { variabel sInt akan berisi string '100'}
        sFloat := FloatToStr (2.12); { varaibel sFloat akan berisi string '2.12'}

        ….
end.
```
(Raharjo, 2008)

2.5 *Runtime* Program

Dalam Pascal ada perintah *unit system* dan *unit* DOS, yang berada dalam file TURBO.TPL. *Unit system* merupakan sebuah *runtime* Turbo Pascal yang mendukung semua proses yang dibutuhkan pada waktu *runtime* (eksekusi program). Unit ini secara otomatis ditambahkan dalam bahasa Pascal, walaupun tidak tercantum saat meng-*compile*. (Raharjo, 2008)

2.6 Turbo Profiler

Sofware yang mendukung dalam kemampuan analisis program dan menyediakan laporan statistika yang mendetail adalah Borland Turbo Profiler. Salah satu kemampuan turbo profiler adalah analisis waktu eksekusi program (*runtime*). (Borland, 1990)

Borland Turbo Profiler adalah perangkat lunak dalam siklus pengembangan. Setelah melakukan *coding* program yang diinginkan, Turbo profiler akan membantu program agar lebih cepat dan lebih efisien . Profiler adalah perangkat lunak untuk mengukur kinerja program dengan mencari *bottleneck*–nya. Borland Turbo Profiler memungkinkan untuk menyempurnakan program, membantu dalam mempercepat program. Profiler dijalankan di DOS. (Borland, 1990)

Profiller adalah salah satu *software* yang paling berguna dan penting setidaknya dipahami untuk pengembangan perangkat lunak yang baik. *Survey* menunjukkan bahwa hanya sebagian kecil dari *programmer* professional

yang benar-benar menggunakan profiler untuk meningkatkan performa programnya. Studi lain menunjukkan bahwa sebagian besar waktu *programmer* dipakai untuk menebak permasalahan dalam programnya. (Borland, 1990)

Apa keuntungan untuk menggunakan *software* ini? Pertama, profiler dapat meningkatkan kinerja program secara keseluruhan. Kedua, profiler memiliki kemampuan menghasilkan kode program yang efisien. Intinya adalah profiler seperti *debugging*, dalam siklus pengembangan program. Profiler juga digunakan untuk mengetahui waktu eksekusi (*runtime*) program yang berkaitan dengan alokasi memori. Ketika pemakaian memori cukup besar ini berakibat pada *runtime* program yang lama. (Borland, 1990)

Sebelum memulai profiler, tentukan bagian program yang ingin diprofiller. Suatu bagian dalam program di mana ingin dilakukan pengujian dan pengunpulan data statistik, sehingga akan diperoleh informasi untuk analisis dari program tersebut. Untuk menganalisis sejumlah *runtime* dalam program harus diketahui berapa banyak waktu yang dibutuhkan setiap program ketika *running*
(Borland, 1990)

2.7 Analisis Ragam *One-way* Anova

Analisis Ragam atau analisis varians adalah sebuah metode yang sering dipakai untuk analisis data. Dalam pengertian uji hipotesis, analisis ragam digunakan untuk menilai kesamaan nilai tengah beberapa populasi yaitu memeriksa apakah $\mu_1 = \mu_2 \ldots = \mu_p$ dengan tandingannya berupa pernyataan bahwa paling sedikit ada sepasang populasi yang berbeda. Untuk pembandingan hasilnya setara dengan hasil pengujian uji-t. (Walpole, 1995)

Sebaran F dapat digunakan untuk memeriksa hipotesis tentang kesamaan ragam populasi atau lebih popular disebut dengan F_{hitung} yang dapat dibandingka dengan tabel F (F_{tabel}). Untuk taraf $\alpha = 0.05$ yang cukup kecil berarti kecil peluang kedua populasi tersebut memiliki ragam yang sama.

Nilai F_{hitung} yang lebih besar dari F_{tabel} pada $\alpha = 0.05$ mengandung arti bahwa palings sedikit ada sepasang perlakuan yang berbeda dengan resiko jenis I sebesar 0.05.

Hakekatnya meskipun pendekatannya masalah ragam, tetapi yang diperiksa adalah nilai tengahnya, sedangkan tentang keragaman telah disederhanakan dengan asumsi kehomogenan ragam. Prosedur ini jelas kurang layak bila ragam tiap perlakuam tak sama. (Walpole, 1995)

Uji-t dapat dipakai untuk pembandingan 2 perlakuan, namun untuk pembandingan lebih dari 2 perlakuan sekaligus, uji F lebih hemat dan resiko α-nya mencakup pembandingan seluruh perlakuan.

Jika $F_{hitung} < F_{0.05}$ maka data tidak mendukung untuk menolak H_N bahwa semua perlakuan memiliki pengaruh yang sama yang bisa berarti tidak hanya nilai tengahnya tetapi juga ragamnya *homogen*. Akan tetapi bila ternyata diperoleh $F_{hitung} > F_{0.05}$, masalahnya adalah perbedaan ini mungkin tidak hanya menyangkut nilai tengahnya saja, tetapi juga ragamnya mungkin saling berbeda (dalam uji F, kehomogenan ragam hanya didasarkan pada asumsi). (Walpole, 1995)

Penolakan $H_N : \mu = \mu = \ldots = \mu_t$ akibat uji F memberikan dorongan untuk memeriksa lebih lanjut, perlakuan mana saja yang sebenarnya berbeda, oleh karena itu dapat dimengerti bila kemudian banyak orang mengembangkan metode pembandingan ganda dengan berbagai kelemahan atau kelebihannya.

Uji Bartlett sering digunakan untuk memeriksa kehomogenen ragam dalam bentuk pengujian hipotesis dengan H_N bahwa ragam antara perlakuan bersifat *homogen* . Sayangnya uji ini peka terhadap ketidak-normalan data, penolakan H_N mungkin hanya disebabkan oleh data yang tidak normal. Di samping itu tidak menolak H_N juga tidak membuktikan bahwa ragam antara perlakuan *homogen*, karena hal ini bisa disebabkan oleh kurangnya data atau datanya tidak normal.

Penggunaan uji F dalam analisis ragam cukup resisten terhadap ragam yang heterogenan atau ketak-normalan data terutama bila memiliki ulangan sama. Oleh karena itu uji Bartlett tidak dianjurkan sebelum melakukan proses analisis ragam. (Aunuddin, 2005)

Analisis ragam adalah suatu metode untuk menguraikan keragaman total data menjadi komponen – komponen yang mengukur berbagai sumber keberagaman.

Klasifikasi pengamatan berdasarkan satu kriteria disebut klasifikasi satu- arah. Misalkan ada k populasi. Dari masing- masing populasi itu diambil contoh berukuran n. Misalkan pula bahwa k populasi itu bebas dan menyebar normal dengan nilai tengah μ_1, μ_2, …., μ_k dan ragam sama.
Pengujian hipotesis:
$H_0 : \mu_1 = \mu_2 = \ldots = \mu_k$,
H_1 : sekurang-kurangnya dua nilai tengah tidak sama.
(Walpole, 1995)

One-way ANOVA digunakan untuk membandingkan apakah terdapat perbedaan atau kesamaan rata-rata antara tiga atau lebih kelompok data untuk suatu kategori tertentu. Asumsi yang digunakan adalah variabel data berdistribusi normal dan homogenitas varians antara kelompok data.

Dalam ANOVA terdapat ketentuan:
a. Jika signifikansi < 0.005 maka varian kelompok data tidak sama, jika signifikansi> 0.005 maka varian kelompok data sama.
b. Langkah-langkah uji ANOVA sebagai berikut:
   1. Merumuskan hipotesis
   2. Menentukan F hitung dan signifikansi
   3. Menetukan F Tabel

      F Tabel dicari pada signifikansi 0.05, df1 (jumlah kelompok data-1) dan df2 (n-3).

4. Kriteria pengujian :
    a. Jika F hitung ≤ F tabel, maka H_0 tidak ditolak.
    b. Jika F hitung > F tabel, maka H_0 ditolak.
5. Berdasarkan signifikansi:
    a. Jika signifikansi > 0.05, maka H_0 tidak ditolak.
    b. Jika signifikansi < 0.05, maka H_0 ditolak.
6. Membuat kesimpulan. (Yamin, 2009)

BAB III
METODELOGI PENELITIAN

3.1 Tempat dan Waktu Penelitian

Penelitian ini dilakukan di Jurusan Matematika Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Lampung. Waktu penelitian dilakukan selama semester ganjil dan semester genap tahun ajaran 2010-2011.

3.2 Alat dan Bahan

Pada tahapan kebutuhan minimum dimulai dengan mengidentifikasi, mengumpulkan studi *literature* mengenai program utuh yang merupakan jawaban dari soal olimpiade komputer.

Untuk melakukan penelitian ini menggunakan alat berupa perangkat keras *(hardware)* dan perangkat lunak *(software).*

1. Perangkat Keras (hardware) dengan spesifikasi :

   Personal Computer (PC):

   a. *Processor* AMD Turion Dual Core

   b. RAM 2 GB

   c. Hard disk 250 GB

2. Perangkat Lunak (Software) :

   a. Sistem Operasi : Sabiliy 11.04

   b. DOS emulator 1.4.0.1

c. Bahasa Pemrograman Borland Pascal 7

d. Turbo Profiler 2.2

e. SPSS 17

3.3 Analisis *Runtime* Program

Pada tahap ini menguji kesesuaian tipe data dalam variabel terhadap *runtime* program dan logika program. Pengujian *runtime* dengan menggunakan aplikasi turbo profiler 2.2 yang dijalankan di Sabiliy 11.04 dengan bantuan DOS emulator 1.4.0.1. Setelah dilakukan sepuluh kali percobaan dari masing- masing lima tipe data, didapat lima puluh data *runtime* yang dibuat dalam bentuk tabel. Selanjutnya lima puluh data *runtime* itu diuji statistik dengan menggunakan aplikasi SPSS 17 pada analisis ragam *one- way* Anova dan Boxplot.

3.4 Tahapan Penelitian

Dalam pelaksanaan penelitian ini ada beberapa tahapan yang dilakukan sebagai berikut:

a. Hipotesis Awal

Ada pengaruh tipe data dalam variabel terhadap kecepatan *runtime* program.

b. Menguji Hipotesis

Dalam menguji kebenaran hipotesis yang diajukan akan dilakukan rangkaian percobaan yang mengambil sampel lima program utuh dari jawaban soal olimpiade komputer. Tiap program akan diuji dengan berbagai tipe data dalam variabel untuk mendapatkan tabel *runtime* program.

c. Pengolahan Data

Data yang di dapat dari rangkaian percobaan tiap program diuji statistik untuk mengetahui kebenaran hipotesis awal. Pengujian hipotesisnya dengan metode *one -way* ANOVA dan Boxplot.

d. Kesimpulan

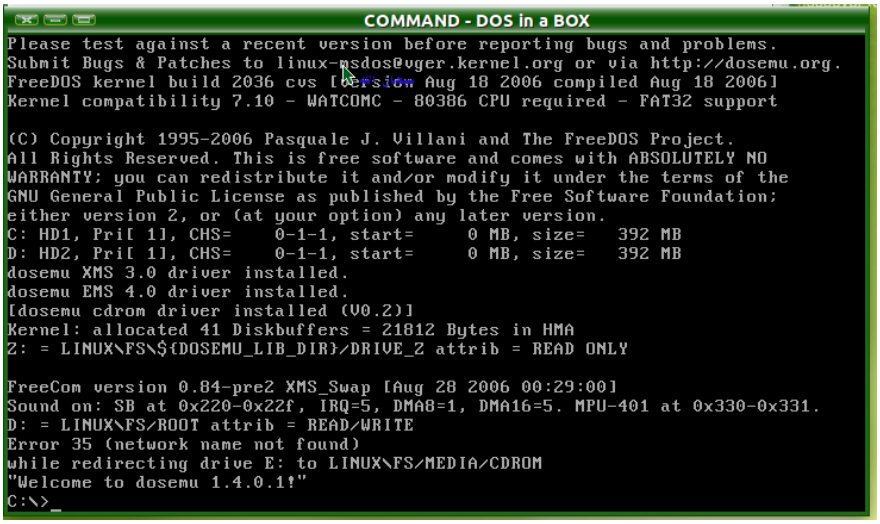
   Merumuskan bahwa tipe data dalam variabel memiliki pengaruh terhadap kecepatan *runtime* program.

BAB IV
HASIL DAN PEMBAHASAN

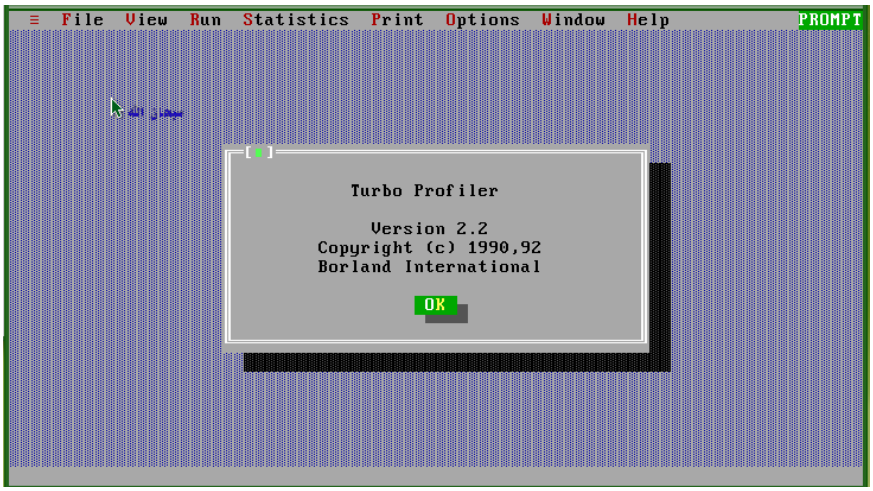
4.1 Hasil Penelitian

Untuk menentukan *runtime* lima program dengan lima tipe data bilangan bulat (longinteger, integer, word, byte, shortint) yang dibuat oleh Dwi Sakethi, M.Kom (Pembimbing I), digunakan aplikasi Turbo Profiler 2.2 yang dijalankan di Sabily 11.04, dengan bantuan DOS Emulator 1.4.0.1.

Berikut ini adalah tampilan dari aplikasi turbo profiler 2.2 yang dijalankan di Sabily 11.04 dengan bantuan dosemulator 1.4.0.1



Gambar 4.1 Tampilan dosemulator 1.4.1.0.1



Gambar 4.2 Tampilan turbo profiler 2.2

Menggunakan aplikasi Turbo Profiler 2.2 program yang akan dilihat *runtime*-nya adalah program mengevaluasi_ekspresi_aljabar, dengan *script* program sebagai berikut:

```
program mengevaluasi_ekspresi_aljabar;

uses crt;
var  ekspresi : string;
     jumlah_tanda,jumlah_suku,panjang : shortint;
     sekarang,i,suku_baru,j,reduksi,asli : shortint;
     asli_panjang,proses_ke,posisi : shortint;
     proses,ada : shortint;
     suku,hasil : array[1..100] of string;
     prioritas  : array[1..100] of shortint;
     karakter,string_temp : string;
     selesai_prioritas,selesai : boolean;
begin
  ekspresi := 'a-b+c-d-e-f-g-h-j';
  ekspresi := 'a-b+c/d';
  ekspresi := 'a-b+c/d*e/f^g-h*j';
  jumlah_tanda := 0;
  jumlah_suku  := 0;
```

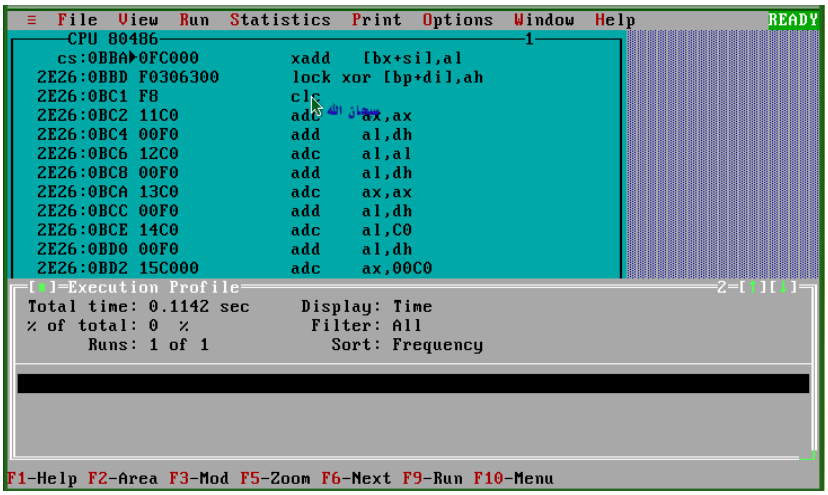
```
  panjang := length(ekspresi);
  for i:=1 to panjang do
     begin
         karakter := copy(ekspresi,i,1);
         suku[i]  := copy(ekspresi,i,1);
         if (karakter='+') or (karakter='-') then
            begin
               jumlah_tanda := jumlah_tanda + 1;
               prioritas[jumlah_tanda] := 3;
            end
         else
            if (karakter='*') or (karakter='/') then
               begin
                  jumlah_tanda := jumlah_tanda + 1;
                  prioritas[jumlah_tanda] := 2;
               end
         else
            if (karakter='^') then
               begin
                  jumlah_tanda := jumlah_tanda + 1;
                  prioritas[jumlah_tanda] := 1;
               end ;
     end;
     jumlah_suku := jumlah_tanda + 1;
     clrscr;
writeln('Ekspresi asal : ',ekspresi);
  writeln('jumlah suku : ',jumlah_suku);
  for i:=1 to jumlah_tanda do write(prioritas[i]);
writeln;
  proses := 0;
  repeat
     inc(proses);
     { cari prioritas yang mau dikerjakan }
```

```
      sekarang := 1;
      selesai_prioritas := false;
      repeat
         if prioritas[sekarang+1]<prioritas[sekarang] then
            begin
              inc(sekarang);
            end
         else
            selesai_prioritas :=true;
      until selesai_prioritas;
      write('proses : ',proses,' tanda : ',sekarang);
      str(proses,string_temp);
      hasil[proses] := 'x'+string_temp;
      writeln(hasil[proses]);
  until proses=jumlah_tanda;
  suku_baru := 0;
  asli := jumlah_suku;
  asli_panjang := panjang;
proses_ke := 0;
repeat
              suku_baru  := suku_baru + 1;
              str(suku_baru,string_temp);
{
              ada := 0;
              for i:=1 to jumlah_tanda do
                  begin
                     if prioritas[i] = prioritas[i+1] then
                        begin
                          ada := 1;
                        end;
                  end;
}
              if(prioritas[1]<prioritas[2])then posisi:=1 else posisi:=2;
```

```
        if (prioritas[suku_baru]>prioritas[suku_baru+1]) then
            begin
               hasil[suku_baru]:='x'+string_temp+'='+suku[2*posisi-
               1]+suku[2*posisi]+suku[2*posisi+1];
               suku[posisi]     := 'x' + string_temp;
               prioritas[posisi] := prioritas[suku_baru+1];
            end
        else
            begin
 hasil[suku_baru]:='x'+string_temp+'='+suku[1]+suku[2]+suku[3];
               suku[1]  := 'x' + string_temp;
            end;
        for j:=posisi to jumlah_tanda do
          begin
             prioritas[j]:=prioritas[j+1];
          end;
        for j:=posisi to panjang do
          begin
             suku[j]:=suku[j+2];
          end;
         for i:=1 tojumlah_tanda do write(prioritas[i],' ');writeln;
         for i:=1 to jumlah_suku do write(suku[i],' ');writeln;
        writeln(hasil[suku_baru]);
   panjang := panjang - 1;
jumlah_tanda := 0;
for i:=1 to asli_panjang-suku_baru do
   begin
      if (suku[i]='+')  or (suku[i]='-') then
         begin
            jumlah_tanda := jumlah_tanda + 1;
            prioritas[i] := 3;
         end
```

```
            else
            if (suku[i]='*')  or (suku[i]='/') then
               begin
                  jumlah_tanda := jumlah_tanda + 1;
                  prioritas[i] := 2;
               end
            else
            if (suku[i]='^')  then
               begin
                  jumlah_tanda := jumlah_tanda + 1;
                  prioritas[i] := 1;
               end;
         end;
until jumlah_tanda < 1;
end.
```

Setelah dilakukan eksekusi program mengevaluasi_ekspresi_aljabar, maka tampilan *runtime* program dapat dilihat sebagai berikut:



Gambar 4.1.1 *runtime* program mengevaluasi_ekspresi_aljabar

Berdasarkan hasil *runtime* program menganalisis_ekspresi_aljabar dengan pengulangan eksekusi program sebanyak sepuluh kali untuk masing-masing tipe data dapat dilihat pada tabel berikut:

Tabel 4.1.1 *runtime* program mengevaluasi_ekspresi_aljabar (program ini untuk menguraikan/menjabarkan sebuah ekspresi aljabar sampai dengan jumlah suku n)

Tipe data	Runtime (detik)									
	1	2	3	4	5	6	7	8	9	10
LongInt	0.1054	0.1113	0.1115	0.0882	0.0963	0.125	0.087	0.1156	0.0963	0.1157
Integer	0.0981	0.1072	0.0858	0.0887	0.0978	0.0888	0.0707	0.0799	0.0708	0.0886
Word	0.1163	0.104	0.089	0.104	0.1255	0.1221	0.1161	0.0704	0.1163	0.0949
Byte	0.1162	0.0798	0.0865	0.0706	0.0978	0.0948	0.1072	0.0886	0.1255	0.0766
ShortInt	0.0883	0.0884	0.0698	0.0793	0.0699	0.0881	0.0887	0.0794	0.0668	0.0931

Selanjutnya dengan aplikasi Turbo Profiler 2.2 program yang akan dilihat *runtime*-nya adalah program analisis_ekspresi_aljabar dengan *script* program sebagai berikut:

```
program analisa_ekpresi_aljabar;

{ versi sabtu }

uses crt;

var ekspresi : string;

    batas_kiri,batas_kanan,proses,sekarang,i,j,jumlah_tanda,panjang      :
shortint;

    tanda,karakter : array[1..255] of string[1];

    str_temp,suku_kiri,suku_kanan: string[2];

    prioritas : array[1..255] of shortint;

    cari_prioritas, selesai : boolean;

    substitusi : string[6];

    jumlah_tanda_asli,nilai: shortint;

    kode_error : integer;

begin

   clrscr;

   ekspresi := 'c/d*e/f^g';

   ekspresi := 'a-b+c/d*e/f^g-h*j';

   panjang  := length(ekspresi);

   jumlah_tanda := 0;

   proses := 0;

{ proses mencari jumlah operator dan operator apa saja
```

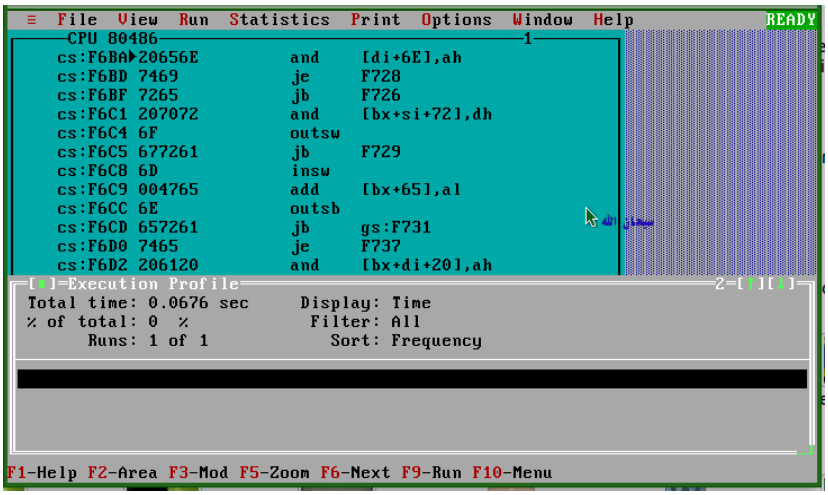
```
yang ada beserta tingkatnya}
 for i:=1 to panjang do
     begin
        karakter[i] := copy(ekspresi,i,1);
        if (karakter[i]='-') or (karakter[i]='+') then
           begin
              inc(jumlah_tanda);
              prioritas[jumlah_tanda] := 1;
              tanda[jumlah_tanda] := karakter[i];
           end
        else
           if (karakter[i]='/') or (karakter[i]='*') then
              begin
                 inc(jumlah_tanda);
                 prioritas[jumlah_tanda] := 2;
                 tanda[jumlah_tanda] := karakter[i];
              end
           else
             if (karakter[i]='^') then
              begin
                 inc(jumlah_tanda);
                 prioritas[jumlah_tanda] := 3;
                 tanda[jumlah_tanda] := karakter[i];
              end;
     end;
     jumlah_tanda_asli := jumlah_tanda;

     { mencari operator mana yang akan dikerjakan terlebih dahulu }
     cari_prioritas := false;
     repeat
        sekarang := 1;
        for i:=1 to jumlah_tanda do
           begin
```

```
                if prioritas[sekarang+1]>prioritas[sekarang] then
                   sekarang := sekarang+1
                else
                   cari_prioritas:=true;
             end;
       until cari_prioritas;
       selesai := false;
repeat
       inc(proses);
       writeln('ekspresi : ',ekspresi);
       write('tanda ke : ',sekarang,' yang mau dikerjakan ');
       textcolor(yellow+blink);writeln(tanda[sekarang]);
       textcolor(white);
       batas_kiri  := 2*sekarang-1;
       batas_kanan := 2*sekarang+1;
       writeln('batas kiri : ',batas_kiri,' batas kanan : ',batas_kanan);
       writeln('karakter batas kiri : ',karakter[batas_kiri],' ',
               'karakter batas kanan : ',karakter[batas_kanan]);
       suku_kiri  := karakter[batas_kiri];
       { kalau suku kiri=1 ini artinya x1,
         kalau suku kiri=2 ini artinya x2, dan seterusnya }
       val(suku_kiri,nilai,kode_error);
       if (nilai>0) then suku_kiri:='x'+suku_kiri;
       suku_kanan := karakter[batas_kanan];
       val(suku_kanan,nilai,kode_error);
       if (nilai>0) then suku_kanan:='x'+suku_kanan;
       substitusi := suku_kiri+tanda[sekarang]+suku_kanan;
       str(proses,str_temp);
      writeln('---------substitusi x',str_temp,'=',substitusi,'-----');
       ekspresi := '';
       for i:=1 to batas_kiri-1 do ekspresi:=ekspresi+karakter[i];
       ekspresi := ekspresi + str_temp;
       for i:=batas_kanan+1 to panjang do ekspresi:=ekspresi+karakter[i];
```

```
        writeln('ekspresi baru setelah direduksi : ',ekspresi);
{ proses seperti di awal kembali }
{proses mencari jumlah operator dan operator apa saja
  yang ada beserta tingkatnya }
   panjang  := length(ekspresi);
   jumlah_tanda := 0;
   for i:=1 to panjang do
       begin
          karakter[i] := copy(ekspresi,i,1);
          if (karakter[i]='-') or (karakter[i]='+') then
             begin
                inc(jumlah_tanda);
                prioritas[jumlah_tanda] := 1;
                tanda[jumlah_tanda] := karakter[i];
             end
          else
             if (karakter[i]='/') or (karakter[i]='*') then
                begin
                   inc(jumlah_tanda);
                   prioritas[jumlah_tanda] := 2;
                   tanda[jumlah_tanda] := karakter[i];
                end
             else
               if (karakter[i]='^') then
                begin
                   inc(jumlah_tanda);
                   prioritas[jumlah_tanda] := 3;
                   tanda[jumlah_tanda] := karakter[i];
                end;
       end;
       { mencari operator mana yang akan dikerjakan terlebih dahulu }
       cari_prioritas := false;
       writeln('jumlah tanda ',jumlah_tanda);
```

```
        repeat
           sekarang := 1;
           for i:=1 to jumlah_tanda do
              begin
                 if prioritas[sekarang+1]>prioritas[sekarang] then
                    sekarang := sekarang+1
                 else
                    cari_prioritas:=true;
              end;
              if jumlah_tanda=1 then cari_prioritas:=true;
        until cari_prioritas;
        selesai := false;
        if jumlah_tanda=1 then
           begin
               str(jumlah_tanda_asli,str_temp);
               writeln('--------
              x',str_temp,'=x',karakter[1],karakter[2],'x',karakter[3]);
               if jumlah_tanda=1 then selesai:=true;
           end;
       until
         selesai;
end.
```

Setelah dilakukan eksekusi program analisis_ekspresi_aljabar, maka tampilan *runtime* program dapat dilihat sebagai berikut:



Gambar 4.1.2 *runtime* program analisis_ekspresi_aljabar

Berdasarkan hasil *runtime* program analisis_ekspresi_aljabar dengan pengulangan eksekusi program sebanyak sepuluh kali untuk masing-masing tipe data dapat dilihat pada tabel berikut:

Tabel 4.1.2 *runtime* program analisis_ekspresi_aljabar (program ini digunakan untuk menentukan ekspresi baru aljabar setelah dilakukan operasi aljabar)

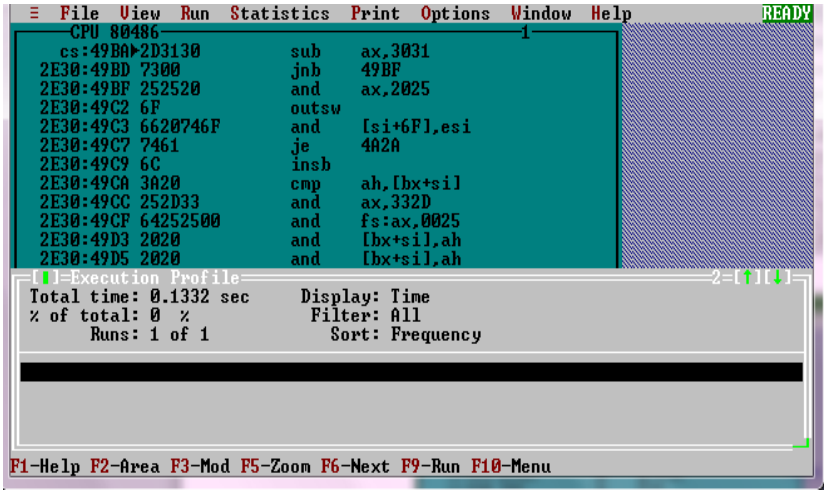
Tipe data	Runtime (detik)									
	1	2	3	4	5	6	7	8	9	10
LongInt	0.1044	0.0951	0.0841	0.1024	0.1135	0.1132	0.1044	0.114	0.0951	0.1115
Integer	0.1043	0.093	0.0921	0.1044	0.0952	0.0952	0.1043	0.0932	0.086	0.1044
Word	0.1044	0.1114	0.1032	0.0952	0.1135	0.1206	0.1042	0.0953	0.1134	0.1044
Byte	0.1024	0.0952	0.0951	0.1044	0.1133	0.0933	0.0861	0.1044	0.0953	0.1135
ShortInt	0.0932	0.095	0.0857	0.0954	0.0931	0.0938	0.0774	0.0952	0.0843	0.0947

Kemudian dengan aplikasi Turbo Profiler 2.2 program yang akan dilihat *runtime*-nya adalah program mencari_beda_tertinggi dengan *script* program sebagai berikut:

```
program mencari_beda_tertinggi;


uses crt;
var tertinggi, terendah, bukit : longint;
    { tertinggi adalah bukit yang paling tinggi
      terendah adalah bukit yang paling rendah
      bukit adalah data bukit yang ada }
```

```
    file_data   : text;
      { data disimpan dalam file teks bukit.dat }
    selisih        : longint;
begin
   clrscr; { hapus layar biar bersih }
   assign(file_data,'bukit.dat');
   reset(file_data);
   { membuka file data }
   tertinggi := 0;
   { nilai tertinggi diisi dengan nilai sekecil mungkin }
   terendah  := 300000;
   { nilai terendah diisi dengan nilai sebesar mungkin }
   while not eof(file_data) do
   { membaca data dari awal sampai data terakhir }
      begin
         readln(file_data,bukit);
         writeln(bukit);
         { hanya membaca satu variabel setiap kali proses baca data }
         if (bukit>tertinggi) then tertinggi := bukit;
         { jika ada bukit yang lebih tinggi maka nilai ditukar }
         if (bukit<terendah) then terendah  := bukit;
         { jika ada bukit yang lebih rendah maka nilai ditukar }
      end;
   close(file_data);
   selisih :=tertinggi-terendah;
   { selisih tinggi bukit terbesar adalah dengan mencari
     bukit tertinggi dan bukit terendah kemudian dikurangkan }
   writeln('tertinggi   : ',tertinggi);
   writeln('terendah    : ',terendah);
   writeln('Perbedaan tertinggi adalah ',selisih);
   { ini hasilnya bro ... }
end.
{ is it too easy problem ? }
```

Setelah dilakukan eksekusi program mencari_beda_tertinggi, maka tampilan *runtime* program dapat dilihat sebagai berikut:



Gambar 4.1.3 *runtime* program mencari_beda_tertinggi

Berdasarkan hasil *runtime* program mencari_beda_tertinggi dengan pengulangan eksekusi program sebanyak sepuluh kali untuk masing-masing tipe data dapat dilihat pada tabel berikut:

Tabel 4.1.3 *runtime* program mencari_beda_tertinggi (program ini digunakan untuk mencari beda ketinggian terbesar dari data bukit)

Tipe data	Runtime (detik)									
	1	2	3	4	5	6	7	8	9	10
LongInt	0.1594	0.1625	0.1596	0.1397	0.1332	0.1558	0.1574	0.1594	0.1625	0.1596
Integer	0.1043	0.093	0.0921	0.1044	0.0952	0.0952	0.1043	0.0932	0.086	0.1044
Word	0.1493	0.143	0.1356	0.1528	0.1522	0.1219	0.1394	0.1493	0.143	0.1356
Byte	0.122	0.1535	0.1364	0.1164	0.1361	0.1257	0.1494	0.122	0.1535	0.1364
ShortInt	0.1139	0.1619	0.1532	0.1333	0.1149	0.1377	0.1370	0.1168	0.1646	0.1516

Berikutnya dengan aplikasi Turbo Profiler 2.2 program yang akan dilihat *runtime*-nya adalah program mencari_multipalindrom_pada_suatu_tulisan dengan *script* program sebagai berikut:

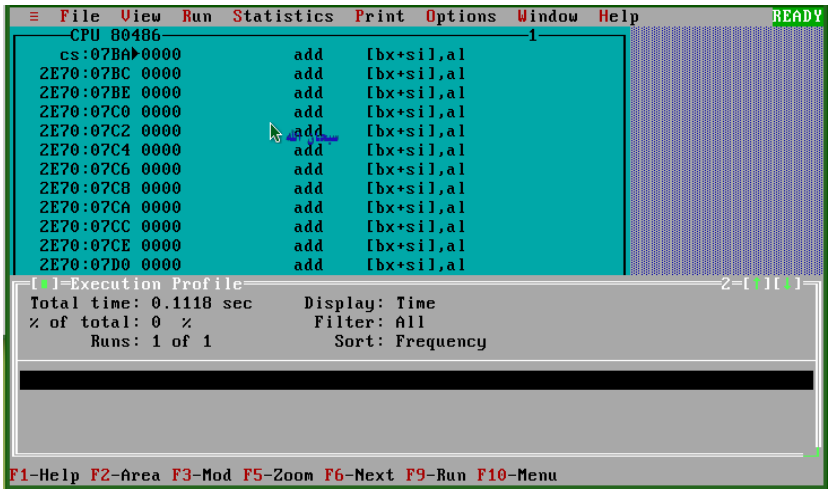
```
program mencari_multipaliandrom_pada_suatu_tulisan;

{ dikembangkan oleh dwi sakethi

                    dwijim@unila.ac.id

                    dwijim@gmail.com
```

```
                0816 403 432
  pada tanggal 18 agustus 2006 }
uses crt;
var file_data,file_hasil : text;
    tulisan_selesai,tulisan_asli,potongan_tulisan,tulisan : string;
    selesai : boolean;
    jumlah_paliandrom,panjang_tulisan_tetap,paliandrom : integer;
    mulai_tulisan_baru,jumlah_looping,batas_kanan : integer;
{
  fungsi ini memberikan nilai 1 jika kata yang dicek berupa
  paliandrom, jika kata yg dicek bukan merupakan paliandrom
  maka fungsi ini memberikan nilai 0
}
function cek_paliandrom(tulisan_yg_dicek : string):word;
var
hasil  : word;
    ii,panjang_tulisan_x : word;
    hasil_reverse        : string;
begin
  hasil             := 0;
  hasil_reverse     := '';
  panjang_tulisan_x := length(tulisan_yg_dicek);
  for ii:=panjang_tulisan_x downto 1 do
      begin
        hasil_reverse := hasil_reverse + copy (tulisan_yg_dicek,ii,1);
      end;
  if tulisan_yg_dicek=hasil_reverse then
     hasil:=1;
  cek_paliandrom := hasil;
end;
{ --- program utama --- }
```

```
begin
      clrscr;
   { membuka file data }
   assign(file_data,'multipal.in');
   reset(file_data);
   { membaca data tulisan }
   read(file_data,tulisan);
   writeln('tulisan asal : ',tulisan);
   { membuat file hasil }
   assign(file_hasil,'multipal.out');
   rewrite(file_hasil);
   { proses pencarian paliandrom dilakukan sampai batas
     akhir tulisan }
   selesai               := false;
   batas_kanan           := length(tulisan);
   panjang_tulisan_tetap := length(tulisan);
   potongan_tulisan      := tulisan;
   tulisan_asli          := tulisan;
   tulisan_selesai       := '';
   jumlah_paliandrom     := 0;
   repeat
     paliandrom :=cek_paliandrom(potongan_tulisan);
     writeln('asal : ',potongan_tulisan, ' cek : ',paliandrom);
     if paliandrom=0 then
        begin
           batas_kanan := batas_kanan - 1;
           potongan_tulisan := copy(potongan_tulisan,1,batas_kanan);
        end
     else
        begin
           writeln('paliandrom : ',potongan_tulisan);
```

```
            tulisan_selesai := tulisan_selesai + potongan_tulisan;
            mulai_tulisan_baru := length(potongan_tulisan)+1;
              potongan_tulisan:=copy(tulisan,mulai_tulisan_baru,panjang_tu
        lisan_tetap-mulai_tulisan_baru+1);
            panjang_tulisan_tetap := length(potongan_tulisan);
            batas_kanan := length(potongan_tulisan);
            tulisan := potongan_tulisan;
            if tulisan<>'' then
               jumlah_paliandrom := jumlah_paliandrom + 1;
            writeln;
          end;
      writeln('tulisan kontrol : ',tulisan_selesai);
      if tulisan_selesai=tulisan_asli then
         selesai := true;
   until selesai;
   writeln('jumlah paliandrom : ',jumlah_paliandrom);
   { tulisan hasil ke file output }
   writeln(file_hasil,jumlah_paliandrom);
   { menutup kembali file yg telah diakses }
   close(file_hasil);
   close(file_data);
   writeln;
end.
```

Setelah dilakukan eksekusi program mencari_multipalindrom_pada_suatu_tulisan maka tampilan *runtime* program dapat dilihat sebagai berikut:



Gambar 4.1.4 *runtime* program mencari_multipalindrom_pada_suatu_tulisan

Berdasarkan hasil *runtime* program mencari_multipalindrom_pada_suatu_tulisan dengan pengulangan eksekusi program sebanyak sepuluh kali untuk masing-masing tipe data dapat dilihat pada tabel berikut:

Tabel 4.1.4 *runtime* program mencari_multipalindrom_pada_suatu_tulisan (program ini digunakan untuk menentukan jumlah palindrom/kata yang sama jika dibolak-balik tulisannya, pada kata anavolimilana)

Tipe data	Runtime (detik)									
	1	2	3	4	5	6	7	8	9	10
LongInt	0.1384	0.1612	0.1538	0.1625	0.1206	0.1445	0.128	0.1618	0.1379	0.1533
Integer	0.1363	0.155	0.1494	0.1627	0.1185	0.1364	0.1267	0.1574	0.1346	0.152
Word	0.1308	0.1643	0.1352	0.1424	0.145	0.1399	0.1681	0.1644	0.1512	0.1496
Byte	0.1316	0.1498	0.1345	0.1486	0.1504	0.148	0.1449	0.1471	0.1261	0.1471
ShortInt	0.1173	0.1318	0.1269	0.1367	0.1116	0.1265	0.1156	0.1314	0.1120	0.1448

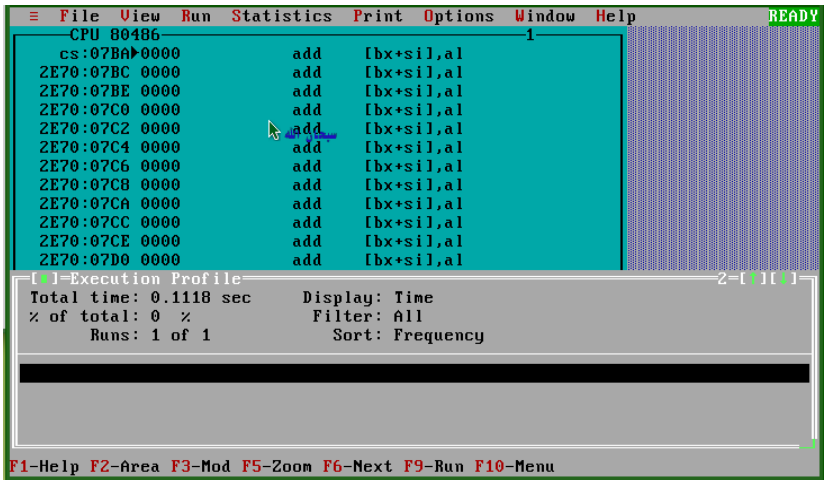
Yang terakhir adalah program mencari_massa yang akan dilihat *runtime*-nya dengan menggunakan aplikasi Turbo Profiler2.2. Adapun *script* programnya sebagai berikut:

```
uses crt;

 var

    isi,potongan,potongan_next : string;

    i,j,tingkat,k     : shortint;

    bilangan,kode    : integer;
```

```
    jum_c,kurung_ke   : shortint;
    tingkat_kurung,jumlah_c,awal,akhir,nilai : array [1..55] of shortint;
begin
  clrscr;
  writeln('program mencari massa');
  isi     := '((CH)2(OH2H)(C(H))O)3';
  tingkat := 0;
  kurung_ke := 0;
  jum_c := 0;
  for i:=1 to 25 do
    begin
       nilai[i] := 0;
       awal[i]  := 0;
       akhir[i] := 0;
       tingkat_kurung[i] := 0;
    end;
  for i:=1 to length(isi)-1 do
    begin
       potongan := copy(isi,i,1);
       if (potongan='(') then
          begin
             inc(tingkat);
             inc(kurung_ke);
             awal[kurung_ke] := i;
          end
       else
          if (potongan=')') then
             begin
                akhir[kurung_ke] := i;
                potongan_next := copy(isi,i+1,1);
                val(potongan_next,bilangan,kode);
```

```
                if (bilangan>1) then
                   begin
                     nilai[kurung_ke] := bilangan;
                   end;
                tingkat_kurung[kurung_ke] := tingkat;
       writeln('tingkat:',tingkat_kurung[kurung_ke], - ',awal[kurung_ke],
                        '-',akhir[kurung_ke],'kali : ',nilai[kurung_ke]);
                dec(tingkat);
              end;
    end;
    for i:=1 to kurung_ke do
      begin
       writeln('tingkat:',tingkat_kurung[i],'-',awal[i],'-',akhir[i],'
       kali : ',nilai[i]);
      end;
  readln;
end.
```

Adapun tampilan *runtime* program mencari massa, setelah dilakukan eksekusi programdapat dilihat sebagai berikut:



Gambar 4.1.5 *runtime* program mencari_massa

Berdasarkan hasil *runtime* program mencari_massa dengan pengulangan eksekusi program sebanyak sepuluh kali untuk masing - masing tipe data dapat dilihat pada tabel berikut ini:

Tabel 4.1.5 *runtime* program mencari_massa (program yang digunakan untuk mencari massa setiap unsur kimia pada $((CH)_2(OH_2H)(C(H))O)_3$)

Tipe data	Runtime (detik)									
	1	2	3	4	5	6	7	8	9	10
LongInt	0.1027	0.1102	0.1103	0.1117	0.1209	0.1206	0.1202	0.1199	0.11	0.1098
Integer	0.102	0.1030	0.1112	0.0927	0.1111	0.1109	0.1106	0.1017	0.1119	0.1003
Word	0.1181	0.1021	0.1113	0.093	0.1101	0.1206	0.1203	0.0929	0.1112	0.1088
Byte	0.0915	0.093	0.1008	0.1014	0.1021	0.1009	0.1112	0.0919	0.102	0.1015
ShortInt	0.0933	0.1013	0.1117	0.0920	0.1103	0.1116	0.1025	0.0934	0.1117	0.1023

4.2 Pembahasan

Data *runtime* dari kelima program pada sub bab sebelumnya, diperoleh dengan kondisi komputer tidak sedang menjalankan aplikasi (*single tasking*), karena pengujian kelima program tersebut dilakukan pada DOS Emulator 1.4.0.1.

Pengujian *runtime* program tidak membandingkan algoritma dari kelima program, melainkan hanya melihat pengaruh tipe data dalam variabel terhadap *runtime* masing - masing program. Setiap program hanya menggunakan satu algoritma dan satu tipe data. Jika setiap program menggunakan algoritma yang berbeda-beda, maka *runtime* program juga berubah. Sedangkan, jika setiap program menggunakan tipe data yang berbeda-beda, maka akan sulit menentukan tipe data yang mempengaruhi *runtime* program.

Selanjutnya hasil data *runtime* yang diperoleh kemudian dibandingkan untuk melihat *runtime* yang terkecil. Adapun alasan *runtime* terkecil yang digunakan untuk membandingkan kelima tipe data adalah untuk mengefisienkan pemakaian memori. Hal ini terlihat jelas bahwa program yang memiliki alokasi memorinya besar mengakibatkan *runtime*-nya lama. Begitupun sebaliknya. Disamping itu, hal ini juga bergantung pada batasan nilai dari kelima tipe data yang digunakan dalam kelima program.

Dalam pengujian *runtime* program, komputer memproses per blok bit yang umumnya terdiri dari 8 bit (1 byte). Byte yaitu integer 8 bit yang bisa menampung tipe bilangan asli, dan shortint yaitu tipe integer 8 bit yang menyimpan bilangan bulat.

Ukuran 8 bit untuk tipe data byte artinya untuk menyimpan tipe tersebut diperlukan memori sebesar 8 bit, nilai yang bisa disimpan adalah 0000 0000 sampai dengan 1111 1111 atau 0 sampai dengan 255.

Ukuran 16 bit untuk tipe data word artinya nilai yang bisa ditampung dalam 16 bit adalah 0000 0000 0000 0000 sampai dengan 1111 1111 1111 1111 atau 0 sampai dengan 65.535.

Untuk bisa menyimpan nilai negatif dalam biner biasanya digunakan representasi yang disebut sebagai komplemen 2. Komplemen 2 adalah kebalikan dari representasi desimal ditambah dengan 1.

Dalam tipe shortint untuk menyimpan -128 desimal, tentukan representasi untuk 128 yaitu 1000 0000, balik (*not*-kan) menjadi 0111 1111, dan tambahkan 1 menjadi 1000 0000, sehingga representasi -128 dalam biner adalah 1000 0000. Dengan cara komplemen 2, maka nilai yang dapat ditampung shortint dalam 8 bit adalah 1000 0000 sampai dengan 1111 111 atau -128 sampai dengan 127.

Sama halnya dengan tipe integer, untuk menyimpan -32.768 desimal, tentukan representasi 32.768 yaitu 1000 0000 0000 0000, balik (*not*-kan) menjadi 0111 1111 1111 1111 , dan tambahkan 1 menjadi 1000 0000 0000 0000, sehingga representasi -32.768 dalam biner adalah 1000 0000 0000 0000. Maka nilai yang dapat disimpan integer dalam 16 bit adalah 1000 0000 0000 0000 sampai dengan 1111 1111 1111 111 atau -32.768 sampai dengan 32.767.

Begitu pun halnya dengan tipe longint, untuk menyimpan -2.147.483.648 desimal, tentukan representasi 2.147.483.648 yaitu 1000 0000 0000 0000 0000 0000 0000 0000, balik (*not*-kan) menjadi 0111 1111 1111 1111 1111 1111 1111 1111 , dan tambahkan 1 menjadi 1000 0000 0000 0000 0000 0000 0000 0000, sehingga representasi -2.147.483.648 dalam biner adalah 1000 0000 0000 0000 0000 0000

0000 0000. Maka nilai yang dapat ditampung longint dalam 32 bit adalah 1000 0000 0000 0000 0000 0000 0000 0000 sampai dengan 1111 1111 1111 1111 1111 1111 1111 111 atau -2.147.483.648 sampai dengan 2.147.483.647.

Agar lebih jelasnya akan diperlihatkan representasi bit diagram slot memori untuk kelima tipe data dibawah ini:

Tipe data longinteger : 32 bit (-2.147.483.648 sampai dengan 2.147.483.647)

-2.147.483.648	1000	0000	0000	0000	0000	0000	0000	0000
2.147.483.647	1111	1111	1111	1111	1111	1111	1111	111-

Tipe data integer : 16 bit (-32.768 sampai dengan 32.767)

-32.768	10	00	00	00	00	00	00	00
32.767	11	11	11	11	11	11	11	1-

Tipe data word : 16 bit (0 sampai dengan 65.535)

0	00	00	00	00	00	00	00	00
65.535	11	11	11	11	11	11	11	11

Tipe data byte : 8 bit (0 sampai dengan 255)

0	0	0	0	0	0	0	0	0
255	1	1	1	1	1	1	1	1

Tipe data shortint : 8 bit (-128 sampai dengan 127)

-128	1	0	0	0	0	0	0	0
127	1	1	1	1	1	1	1	-

Berdasarkan diagram slot memori di atas:

a. Telah terpakai 1 bit untuk representasi nilai negatif pada *range* nilai 2.147.483.647 untuk tipe data longint, 32.767 untuk tipe data integer, 127 untuk tipe data shortint, atau dengan kata lain terjadi pengurangan 1 bit untuk *range* nilai yang terakhir.

b. Jelas bahwa *range* yang lebar akan berbanding lurus dengan alokasi/ daya tampung memori yang besar, sehingga mengakibatkan *runtime* -nya juga besar. Begitu pun sebaliknya.

Pengujian *runtime* untuk program mencari_beda_tertinggi terhadap kelima tipe data yaitu longint, integer, word, byte dan shortint dengan menguji setiap tipe data hingga diperoleh sepuluh sampel percobaan dari masing- masing tipe data. Setelah memperoleh data *runtime,* kelima tipe data dibandingkan rataannya. Untuk rataan tipe data longint: 0.1598, integer: 0.1520 , word: 0.1398, byte: 0.1320 dan shortint: 0.1350. Dari data *runtime* tersebut dinyatakan bahwa batasan nilai variabel terbaik dari program mencari_beda_tertinggi berada pada rentang tipe data byte dan shortint, sehingga dapat dikatakan dalam operasi programnya menentukan beda keinggian terbesar dari data bukit, maka jangkauan nilai operasinya antara -128 sampai dengan 127 atau 0 sampai dengan 255, namun untuk membatasi alokasi memori yang dibutuhkan dari penggunaan kedua tipe data yang sama- sama berukuran 1 byte tersebut, sebaiknya menggunakan rentang nilai yang paling maksimum dari hasil jenis operasi pada nilai variabelnya yaitu pada tipe data byte. Sedangkan, ketika program menggunakan tipe data longint, integer, atau word hal ini akan mempengaruhi efektifitas dari program. Maka dari itu dikarenakan kebutuhan program ini masih cukup dengan tipe data shortint (1 byte) atau byte (1 byte), diharapkan tidak menggunakan tipe data integer (2 byte) apalagi longint (4 byte) karena hal ini merupakan pemborosan memori sehingga memperlama *runtime.*

4.2.1 Analisis Ragam *One-way* Anova

Untuk melihat pengaruh tipe variabel terhadap kecepatan *running* program, dilakukan pengujian analisis ragam satu arah (*one-way* Anova) terhadap *runtime* lima tipe data dari lima program.

Uji kehomogenan ragam untuk kelima program dengan menggunakan SPSS 17 dan metode *levene statistic* diperoleh sebagai berikut:

Program mengevaluasi_ekspresi_aljabar

Levene Statistic	df1	df2	Sig.
1.178	4	45	.333

Berdasarkan hasil *ouput* di atas dapat diuraikan bahwa:

Rumusan hipotesis:

H_0 = data *runtime* kelima tipe data mempunyai ragam yang homogen.

H_1 = data *runtime* kelima tipe data tidak mempunyai ragam yang homogen.

Selanjutnya diperoleh nilai signifikan sebesar 0.333, apabila nilai ini di bandingkan dengan nilai alfa yaitu 0.05 maka nilai signifikan lebih besar dari nilai alfa, sehingga dapat disimpulkan tidak menolak H_0 dengan kata lain *runtime* kelima tipe data mempunyai varian yang homogen.

Selanjutnya dapat dilakukan uji Anova satu arah.

	Sum of Squares	Df	Mean Square	F	Sig.
Between Groups	.005	4	.001	5.871	.001
Within Groups	.009	45	.000		
Total	.014	49			

Berdasarkan tabel *ouput* di atas dapat dinyatakan:

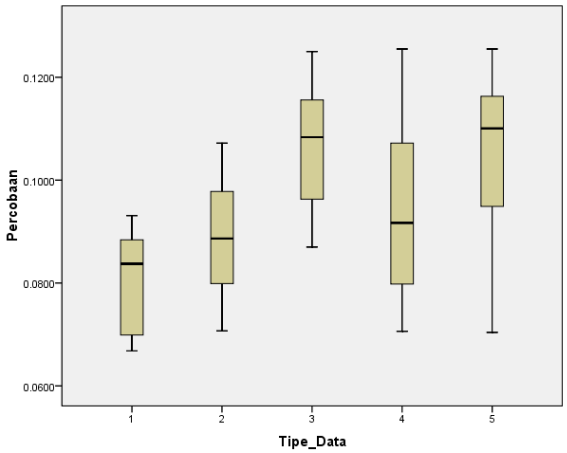
Rumusan hipotesis

H_0 : Tidak ada perbedaan rata-rata nilai *runtime* dari tipe data shortint, int, longint, byte, word.

H_a : Ada perbedaan rata-rata antara nilai *runtime* dari tipe data shortint, int, longint, byte, word.

Karena nilai signifikan tabel Anova 0.001 apabila dibandingkan dengan nilai alfa 0.05, maka nilai tabel Anova lebih kecil dari nilai alfa, sehingga dapat disimpulkan bahwa H_0 ditolak atau dengan kata lain ada perbedaan rata – rata nilai *runtime* kelima tipe data pada program mengevaluasi_ekspresi_aljabar.

Untuk melihat perbandingan *runtime* minimum di lima tipe data dalam bentuk grafik dapat dilakukan dengan Boxplot analisis.



Gambar 4.2.1 Boxplot program mengevaluasi_ekspresi_aljabar

Berdasarkan Boxplot di atas, nilai tengah data (median) dari *runtime* tipe data shortint lebih kecil dibandingkan lima tipe data dapat disimpulkan bahwa program mengevaluasi_ekspresi_aljabar memiliki *runtime* yang terkecil/minimum pada tipe data shortint.

Pada program mengevaluasi_ekspresi_aljabar tipe data shortint memiliki *runtime* terkecil dibandingkan keempat tipe data lainnya yaitu 0.8118 detik, maka dengan kata lain *best case running time* program mengevaluasi_ekspresi_aljabar pada tipe data shortint.

Program analisis_ekspresi_aljabar

Levene Statistic	df1	df2	Sig.
1.013	4	45	.411

Berdasarkan hasil *ouput* di atas dapat diuraikan bahwa:

Rumusan hipotesis:

H_0 = data *runtime* kelima tipe data mempunyai ragam yang homogen.

H_1 = data *runtime* kelima tipe data tidak mempunyai ragam yang homogen.

Selanjutnya diperoleh nilai signifikan sebesar 0.411, apabila nilai ini di bandingkan dengan nilai alfa yaitu 0.05 maka nilai signifikan lebih besar dari nilai alfa, sehingga dapat disimpulkan tidak menolak H_0 dengan kata lain *runtime* kelima tipe data mempunyai varian yang homogen.

Selanjutnya dapat dilakukan uji Anova satu arah.

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	.002	4	.001	7.591	.000
Within Groups	.004	45	.000		
Total	.006	49			

Berdasarkan tabel *ouput* di atas dapat dinyatakan:

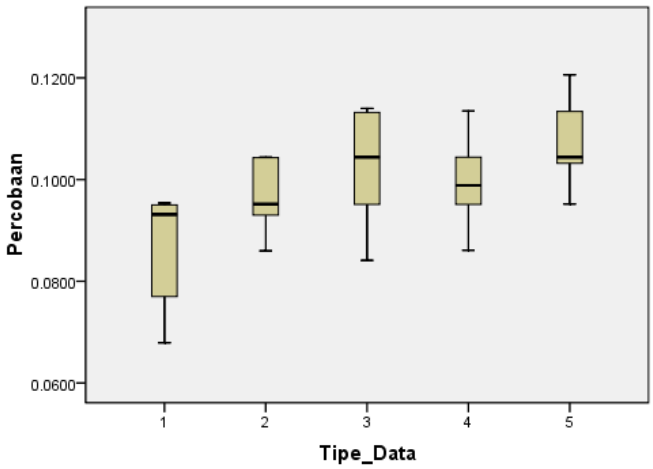
Rumusan hipotesis

H_0 : Tidak ada perbedaan rata-rata nilai *runtime* dari tipe data shortint, int, longint, byte, word.

H_a : Ada perbedaan rata-rata antara nilai *runtime* dari tipe data shortint, int, longint, byte, word.

Karena nilai signifikan tabel Anova 0.000 apabila dibandingkan dengan nilai alfa 0.05, maka nilai tabel Anova lebih kecil dari nilai alfa, sehingga dapat disimpulkan bahwa H_0 ditolak atau dengan kata lain ada perbedaan rata – rata nilai *runtime* kelima tipe data pada program analisis_ekspresi_aljabar.

Untuk melihat perbandingan *runtime* minimum di lima tipe data dalam bentuk grafik dapat dilakukan dengan Box Plot analisis.



Gambar 4.2.2 Boxplot program analisis_ekspresi_aljabar

Berdasarkan Boxplot di atas, nilai tengah data (median) dari *runtime* tipe data shortint lebih kecil dibandingkan lima tipe data dapat disimpulkan bahwa program analisis_ekspresi_aljabar memiliki *runtime* yang terkecil / minimum pada tipe data shortint.

Pada program analisis_ekspresi_aljabar tipe data shortint memiliki runtime terkecil dibandingkan keempat tipe data lainnya yaitu 0.9078 detik, maka dengan kata lain *best case running time* program analisis_ekspresi_aljabar pada tipe data shortint.

Program mencari_beda_tertinggi

Levene Statistic	df1	df2	Sig.
1.093	4	45	.371

Berdasarkan hasil *ouput* di atas dapat diuraikan bahwa:

Rumusan hipotesis:

H_0 = data *runtime* kelima tipe data mempunyai ragam yang homogen.

H_1 = data *runtime* kelima tipe data tidak mempunyai ragam yang homogen.

Selanjutnya diperoleh nilai signifikan sebesar 0.371, apabila nilai ini di bandingkan dengan nilai alfa yaitu 0.05 maka nilai signifikan lebih besar dari nilai alfa, sehingga dapat disimpulkan tidak menolak H_0 dengan kata lain *runtime* kelima tipe data mempunyai varian yang homogen.

Selanjutnya dapat dilakukan uji Anova satu arah.

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	.003	4	.001	2.979	.029
Within Groups	.010	45	.000		
Total	.012	49			

Berdasarkan tabel *ouput* di atas dapat dinyatakan:

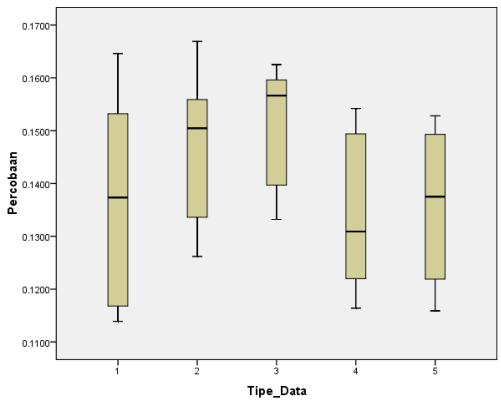
Rumusan hipotesis

H_0 : Tidak ada perbedaan rata-rata nilai *runtime* dari tipe data shortint, int, longint, byte, word.

H_a : Ada perbedaan rata-rata antara nilai *runtime* dari tipe data shortint, int, longint, byte, word.

Karena nilai signifikan tabel Anova 0.029 apabila dibandingkan dengan nilai alfa 0.05, maka nilai tabel Anova lebih kecil dari nilai alfa, sehingga dapat disimpulkan bahwa H_0 ditolak atau dengan kata lain ada perbedaan rata – rata nilai *runtime* kelima tipe data pada program mencari_beda_tertinggi.

Untuk melihat perbandingan *runtime* minimum di lima tipe data dalam bentuk grafik dapat dilakukan dengan Boxplot analisis.



Gambar 4.2.3 Boxplot program mencari_beda_tertinggi

Berdasarkan Boxplot di atas, nilai tengah data (median) dari *runtime* tipe data byte lebih kecil dibandingkan lima tipe data dapat disimpulkan bahwa program mencari_beda_tertinggi memiliki *runtime* yang terkecil / minimum pada tipe data byte.

Pada program mencari_beda_tertinggi tipe data byte memiliki *runtime* terkecil dibandingkan keempat tipe data lainnya yaitu 1.3541 detik, maka dengan kata lain *best case running time* program mencari_beda_tertinggi pada tipe data byte.

Program mencari_multipalindrom_pada_suatu_tulisan

Levene Statistic	df1	df2	Sig.
1.311	4	45	.280

Berdasarkan hasil *ouput* di atas dapat diuraikan bahwa:

Rumusan hipotesis:

H_0 = data *runtime* kelima tipe data mempunyai ragam yang homogen.

H_1 = data *runtime* kelima tipe data tidak mempunyai ragam yang homogen.

Selanjutnya diperoleh nilai signifikan sebesar 0.280, apabila nilai ini di bandingkan dengan nilai alfa yaitu 0.05 maka nilai signifikan lebih besar dari nilai alfa, sehingga dapat disimpulkan tidak menolak H_0 dengan kata lain *runtime* kelima tipe data mempunyai varian yang homogen.

Selanjutnya dapat dilakukan uji Anova satu arah.

	Sum of Squares	df	Mean Square	F	Sig.
Between Groups	.003	4	.001	5.356	.001
Within Groups	.007	45	.000		
Total	.011	49			

Berdasarkan tabel *ouput* di atas dapat dinyatakan:

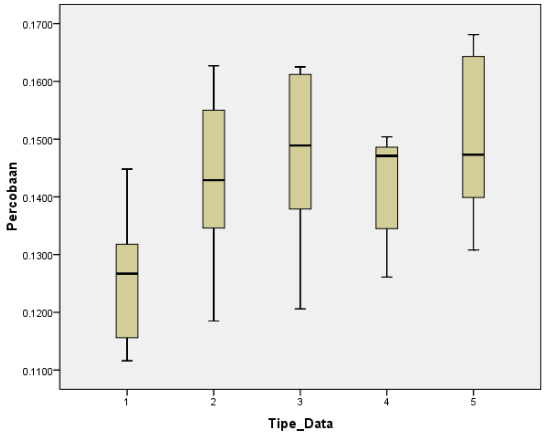
Rumusan hipotesis

H_0 : Tidak ada perbedaan rata-rata nilai *runtime* dari tipe data shortint, int, longint, byte, word.

H_a : Ada perbedaan rata-rata antara nilai *runtime* dari tipe data shortint, int, longint, byte, word.

Karena nilai signifikan tabel Anova 0.001 apabila dibandingkan dengan nilai alfa 0.05, maka nilai tabel Anova lebih kecil dari nilai alfa, sehingga dapat disimpulkan bahwa H_0 ditolak atau dengan kata lain ada perbedaan rata – rata nilai *runtime* kelima tipe data pada program mencari_multipalindrom_suatu_ tulisan.

Untuk melihat perbandingan runtime minimum di lima tipe data dalam bentuk grafik dapat dilakukan dengan Boxplot analisis.



Gambar 4.2.4 Boxplot program mencari_multipalindrom_pada_suatu_tulisan

Berdasarkan Boxplot di atas, nilai tengah data (median) dari *runtime* tipe data shortint lebih kecil dibandingkan lima tipe data dapat disimpulkan bahwa program mencari_multipalindrom_pada_suatu_tulisan memiliki *runtime* yang terkecil/ minimum pada tipe data shortint.

Pada program mencari_multipalindrom_ pada_suatu_tulisan tipe data shortint memiliki *runtime* terkecil dibandingkan keempat tipe data lainnya yaitu 1.2546

detik, maka dengan kata lain *best case running time* program mencari_multipalindrom_ pada_suatu_tulisan pada tipe data shortint.

Program mencari_massa

Levene Statistic	df1	df2	Sig.
.943	4	45	.448

Berdasarkan hasil *ouput* di atas dapat diuraikan bahwa:

Rumusan hipotesis:

H_0 = data *runtime* kelima tipe data mempunyai ragam yang homogen.

H_1 = data *runtime* kelima tipe data tidak mempunyai ragam yang homogen.

Selanjutnya diperoleh nilai signifikan sebesar 0.448, apabila nilai ini di bandingkan dengan nilai alfa yaitu 0.05 maka nilai signifikan lebih besar dari nilai alfa, sehingga dapat disimpulkan tidak menolak H_0 dengan kata lain *runtime* kelima tipe data mempunyai varian yang homogen.

Selanjutnya dapat dilakukan uji Anova satu arah.

	Sum of Squares	Df	Mean Square	F	Sig.
Between Groups	.001	4	.000	5.065	.002
Within Groups	.003	45	.000		
Total	.004	49			

Berdasarkan tabel *ouput* di atas dapat dinyatakan:

Rumusan hipotesis

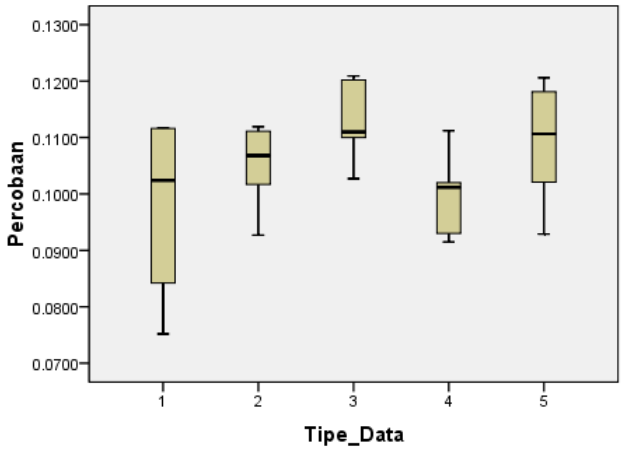
H_0 : Tidak ada perbedaan rata-rata nilai *runtime* dari tipe data shortint, int, longint, byte, word.

H_a : Ada perbedaan rata-rata antara nilai *runtime* dari tipe data shortint, int, longint, byte, word.

Karena nilai signifikan tabel Anova 0.002 apabila dibandingkan dengan nilai alfa 0.05, maka nilai tabel Anova lebih kecil dari nilai alfa, sehingga dapat

disimpulkan bahwa H_0 ditolak atau dengan kata lain ada perbedaan rata – rata nilai *runtime* kelima tipe data pada program mencari_massa.

Untuk melihat perbandingan *runtime* minimum di lima tipe data dalam bentuk grafik dapat dilakukan dengan Boxplot analisis.



Gambar 4.2.5 Boxplot program mencari_massa

Berdasarkan Boxplot di atas, nilai tengah data (median) dari *runtime* tipe data byte lebih kecil dibandingkan lima tipe data dapat disimpulkan bahwa program mencari_massa memiliki *runtime* yang terkecil / minimum pada tipe data byte.

Pada program mencari_massa tipe data byte memiliki *runtime* terkecil dibandingkan keempat tipe data lainnya yaitu 0.9963 detik, maka dengan kata lain *best case running time* program mencari_massa pada tipe data byte.

BAB V
SIMPULAN DAN SARAN

5.1 SIMPULAN

Berdasarkan hasil dari pembahasan, maka dapat disimpulkan sebagai berikut:

1. Terdapat pengaruh tipe data terhadap kecepatan running program (*runtime*).
2. Dari penelitian yang dilakukan lima tipe data terhadap lima program, maka:
   a. Tipe data shortint yang terbaik untuk program mengevaluasi_ekspresi_ aljabar.
   b. Tipe data shortint yang terbaik untuk program analisis_ekspresi_aljabar.
   c. Tipe data byte yang terbaik untuk program mencari_beda_tertinggi.
   d. Tipe data shortint yang terbaik untuk program mencari_multipalindrom_ suatu_tulisan.
   e. Tipe data byte yang terbaik untuk program mencari_massa.

5.2 SARAN

1. Menggunakan teknik pengujian statistika lain dalam menganalisis *runtime* program.
2. Penelitian lebih berkembang dengan penggunaan tipe data lain, misal tipe data bilangan real.