Contoh Soal 1. Cari Maksimum

(Soal ini pernah diberikan dalam OSN 2003, Balikpapan)

Pembahasan oleh : Ilham Kurnia, Alumni TOKI 2003

Nama Program:	`CARI.PAS`
Batas *Run-time*:	1 detik / test case
Nama File Masukan:	`CARI.IN`
Nama File Keluaran:	`CARI.OUT`

Buatlah sebuah program yang menghasilkan keluaran berupa sebuah program dalam Pascal standar yang mencari maksimum dari *N* (1 ? *N* ? 13) bilangan tanpa menggunakan *loop*. Agar keluaran mudah dibaca dan dimengerti, maka aturan-aturan berikut harus dipenuhi:

1. Keluaran diawali dengan sebuah baris berisi "`program Maksimum (input, output);`" (tanpa tanda kutip).
2. Baris kedua berisi pendeklarasian variabel-variabel dengan tipe integer yang dibutuhkan. Nama variabel berasal dari *N* huruf pertama pada abjad (semuanya huruf kecil). Berikan tepat satu spasi antara `var` dengan variabel pertama, koma dengan variabel berikutnya, variabel terakhir dengan `:`, dan `:` dengan `integer`.
3. Baris ketiga berisi "`begin`".
4. Baris keempat berisi sebuah perintah `read`, yang harus membaca nilai semua variabel. Berikan tepat satu spasi antara koma dengan variabel berikutnya.
5. Baris-baris berikutnya berisi komparasi (bila diperlukan) antara variabel-variabel sehingga secara pasti ditemukan variabel yang menyimpan nilai tertinggi. Komparasi dilakukan dengan menggunakan perintah "`if then else`". Kondisi dari setiap perintah `if` adalah sebuah pertidaksamaan lebih besar (`>`). Bagian perintah "`if then`" dan "`else`" masing-masing berada pada baris yang berbeda. Urutan komparasi adalah sesuai dengan abjad. Jadi, komparasi variabel `b` dengan variabel `d` tidak boleh mendahului komparasi variabel `b` dengan variabel `e` dan sebagainya.
6. Variabel yang secara alfabetis muncul lebih dahulu harus menempati posisi sebelum tanda `>` pada setiap pengecekan kondisi. Beri tepat satu spasi antara perintah `if` dengan variabel pertama, variabel pertama dengan `>`, `>` dengan variabel kedua, dan variabel kedua dengan perintah `then`.
7. Berilah indentasi (jorokan) sebanyak 2 spasi pada baris-baris yang merupakan bagian perintah yang dilakukan bila kondisi yang ada terpenuhi (setelah `if then`, sebelum `else`). Beri pula indentasi dengan yang sama besar pada baris-baris yang merupakan bagian perintah yang dilakukan (setelah `else`) bila kondisi yang ada bernilai salah.
8. Apabila komparasi yang dilakukan telah dapat memastikan variabel mana yang menyimpan nilai terbesar, maka tulis isi perintah tersebut dengan menggunakan perintah "`writeln(...)`", misalnya `writeln(d)`.
9. Harus ada tepat 2^{N-1} perintah `writeln`.
10. Hanya ada tepat tiga karakter titik koma (`;`) pada keluaran.
11. Akhiri keluaran dengan "`end.`".
12. Keluaran harus dapat di*compile* dengan baik.

Lihat contoh yang diberikan untuk mempermudah Anda mengerti mengenai aturan-aturan di atas.

FORMAT MASUKAN (Nama File: `CARI.IN`)

Masukan terdiri dari satu baris yang berisi bilangan bulat *N*.

CONTOH MASUKAN

```
3
```

FORMAT KELUARAN (Nama File: `CARI.OUT`)

Keluaran yang dihasilkan harus sesuai dengan ketentuan yang tertulis di deskripsi soal.

CONTOH KELUARAN

```
program Maksimum (input, output);
var a, b, c : integer;
begin
read(a, b, c);
if a > b then
  if a > c then
    writeln(a)
  else
    writeln(c)
else
  if b > c then
    writeln(b)
  else
    writeln(c)
end.
```

Catatan
Perhatikan baik-baik penggunaan spasi.

1. Pembahasan Soal "Cari Maksimum"

Tujuan dari soal ini adalah untuk mengetahui seberapa jauh kemampuan peserta untuk mengikuti petunjuk yang diberikan selain untuk mengetahui tingkat kemampuan peserta dalam hal rekursi dan menulis output. Apabila semua aturan yang ada diikuti dengan teliti, maka yang menjadi masalah hanyalah bagaimana cara merekursi sehingga tidak ada satupun kemungkinan yang terlewat.

Bagi yang belum tahu rekursi, rekursi adalah suatu cara untuk menyelesaikan masalah dengan memecahkan masalah dalam bentuk yang sama, tapi dengan parameter yang lebih kecil. Salah satu contoh yang mudah adalah menghitung nilai faktorial dengan menggunakan rekursi:

```
function faktorial(n : integer) : integer;
begin
if n = 0 then faktorial := 1
else faktorial := faktorial(n - 1) * n;
end;
```

Bila dirumuskan,

$$faktorial(n) = \begin{cases} faktorial(n-1)*n, & n > 0 \\ 1 & n = 0 \end{cases}$$

Untuk membuat sebuah rekursi yang sukses, maka ada 2 kasus yang perlu kita kenali:

1. Kasus dasar (*base case*): kasus di mana kita dapat menentukan secara pasti apa yang dilakukan (biasanya jawaban dasar dari permasalahan kita).
2. Kasus rekursif: kasus di mana jawaban dari permasalahan yang kita hadapi baru diketahui setelah kita tahu jawaban dari permasalahan yang lebih kecil.

Dalam permasalahan ini, yang menjadi kasus dasar adalah ketika bilangan yang paling besar sudah diketahui secara pasti. Pada saat itu, yang perlu kita lakukan hanyalah menuliskan jawabannya dengan menggunakan perintah `writeln`. Yang perlu sedikit pemikiran adalah kasus rekursif: syarat dan parameter rekursinya.

Kasus rekursif pada persoalan ini muncul pada perintah `if then else`. Hal ini disebabkan karena adanya percabangan dalam kemungkinan jawaban. Misal, ketika kita membandingkan `a` dan `b`, ada 2 kemungkinan yang terjadi: `a = b`, atau `a < b`. Untuk kemungkinan `a = b`, nilai variabel `a` kita "teruskan" sebagai variabel yang berisi nilai terbesar. Sementara itu, untuk kemungkinan `a < b`, nilai variabel `b` kita "teruskan" sebagai variabel yang berisi nilai terbesar. Nilai parameternya mengecil dengan makin sedikitnya variabel yang harus dibandingkan.

Setelah fungsi rekursifnya telah selesai dibuat, maka kita tinggal memikirkan hal-hal yang lebih rinci, seperti jumlah spasi pada awal baris untuk setiap tingkat jorokan, adanya hanya 3 titik koma, adanya `begin` dan `end`, dan sebagainya.

Dalam soal ini, disediakan 10 test case dengan rincian sebagai berikut:

No	*N*	Banyak Baris pada Keluaran
1	1	6
2	2	9
3	4	27
4	6	99
5	8	387
6	9	771
7	10	1539
8	11	3075
9	12	6147
10	13	12291

Berikut adalah source code solusi model:

```
program CariMaksimum(input, output);
const MAXN = 26;
      InFile = 'CARI.IN';
      OutFile = 'CARI.OUT';
      SMALL_A = 96; { + 1 }

var n : integer;

{ write x spaces }
procedure pad(x : integer);
var i : integer;

begin
for i := 1 to x do write(' ');
end;

procedure recurse(prev, lvl : integer);

begin
if lvl = n then
  { base case }
  begin
  pad((lvl - 1) shl 1);
  writeln('writeln(', chr(prev + SMALL_A), ')');
  end
else
  { recursive case }
  begin
  { if prev > lvl + 1 }
  pad((lvl - 1) shl 1);
  writeln('if ', chr(prev + SMALL_A), ' > ',
                chr(lvl + 1 + SMALL_A), ' then');
  recurse(prev, lvl + 1);

  { else }
  pad((lvl - 1) shl 1);
  writeln('else');
  recurse(lvl + 1, lvl + 1);
```

```
  end;
end;

procedure process;
var i : integer;

begin
{ write header }
writeln('program Maksimum (input, output);');

{ write variables }
write  ('var a');
for i := 2 to n do write(', ', chr(i + SMALL_A));
writeln(' : integer;');

{ write main program }
writeln('begin');

  { readln }
  write  ('read(a');
  for i := 2 to n do write(', ', chr(i + SMALL_A));
  writeln(');');

  { ifs and elses }
  recurse(1, 1);

writeln('end.');
end;

begin
assign(input, InFile);
assign(output, OutFile);
reset(input);
rewrite(output);
read(n);

process;

close(input);
close(output);
end.
```

Tradisional

Contoh Soal 2. Menghitung Perulangan

(Soal ini pernah diberikan dalam OSN 2003, Balikpapan)
oleh : Ilham Kurnia, Alumni TOKI 2003

Nama Program:	`HITUNG.PAS`
Batas *Run-time*:	1 detik / test case
Nama File Masukan:	`HITUNG.IN`
Nama File Keluaran:	`HITUNG.OUT`

Diberikan dua buah untaian huruf (*string*), hitung berapa kali string kedua muncul sebagai bagian dari string pertama. Asumsikan bahwa tiap kemunculan dari string kedua pada string pertama boleh saling menimpa (*overlap*). Panjang string pertama maksimal 10000, sementara panjang string kedua maksimal 200. String didefinisikan sebagai untaian karakter-karakter dengan kode ASCII 32 – 127 yang dibatasi oleh karakter-karakter dengan kode ASCII yang tidak termasuk dalam jangkauan 32 – 127 tersebut.

FORMAT MASUKAN (Nama File: `HITUNG.IN`)

Masukan terdiri dari dua baris. Baris pertama berisi string pertama, sementara baris kedua berisi string kedua.

CONTOH MASUKAN

```
abcdefabcghiabcabcjklmnlabcw
abc
```

FORMAT KELUARAN (Nama File: `HITUNG.OUT`)

Keluaran hanya terdiri dari sebuah baris berisi sebuah bilangan bulat yang menyatakan banyak kemunculan string kedua pada string pertama.

CONTOH KELUARAN

```
5
```

2. Pembahasan Soal "Menghitung Perulangan"

Inti dari "Menghitung Perulangan" tidak lain adalah berulang kali menyimulasikan perintah *find*, yang kita sering temui pada *software-software word processing* seperti Notepad, Edit, Star Office, dan sebagainya. Ada beberapa cara yang dapat digunakan untuk melakukan hal ini. Akan tetapi, untuk tingkat ini, hanya satu cara yang akan dibahas di sini, yaitu dengan *Brute Force*.

Permasalahan ini bisa dibagi menjadi 2 bagian: membaca input dan mencari keberadaan substring. Ada satu hal yang menyebabkan membaca input menjadi bagian tersendiri, yaitu panjang string pertama melebihi 255. Oleh sebab itu, kita tidak dapat serta merta melakukan pembacaan dengan menggunakan readln (walaupun nanti kita akan lihat sebuah pengecualian).

Salah satu cara untuk mengatasi hal ini adalah dengan mendefinisikan sebuah tipe variabel sendiri yang berupa array[1..10000] of char. Untuk membaca sebuah baris pada masukan, maka kita baca karakter demi karakter sampai kita temui karakter penanda akhir baris. Di Linux, karakter penanda akhir baris adalah ASCII #10 alias *newline character*, sementara pada Windows, karakter penanda akhir baris adalah rentetan ASCII #13 atau *linefeed character* dan ASCII #10.

Seperti yang dikatakan di atas, ada beberapa cara yang dapat dilakukan untuk menentukan keberadaan substring pada sebuah string, tapi kita akan hanya membahas satu saja. Cara *brute force* mengiterasi semua karakter pada string dan membandingkan setiap karakter pada bagian string yang sedang diiterasi dengan karakter-karakter dari substring. Bila setiap perbandingannya adalah benar, maka substring tersebut ada bagian dari string. Kita hanya cukup menghitung berapa kali perbandingan tersebut benar untuk mengetahui berapa kali kemunculan substring pada string tersebut.

Berikut source code yang memecahkan masalah ini dengan menggunakan cara di atas:

```
program MenghitungPerulangan(input, output);

Const InFile = 'HITUNG.IN';
      OutFile = 'HITUNG.OUT';
      MAXA = 10000;
      MAXB = 200;

var a : array[1..MAXA + 1] of char;
    b : array[1..MAXB + 1] of char;
    cta, ctb, ans : integer;
    i, j : integer;
    cek : boolean;

begin
assign(input, InFile);
assign(output, OutFile);
reset(input);
rewrite(output);

cta := 0; ctb := 1; ans := 0;

while not eoln do
  begin
```

```
  inc(cta);
  read(a[cta]);
  end;

{ read input }
b[ctb] := #13;
while (b[ctb] = #10) or (b[ctb] = #13) do read(b[ctb]);
while not eoln and not eof do
  begin
  inc(ctb);
  read(b[ctb]);
  end;

{ find substring locations }
for i := 1 to cta - ctb + 1 do
  begin
  cek := true;
  for j := 1 to ctb do
    if a[i + j - 1] <> b[j] then begin cek := false; break; end;
  if cek then inc(ans);
  end;

writeln(ans);

close(input);
close(output);
end.
```

Adapun solusi lain yang lebih mudah adalah untuk menggunakan tipe ANSIString yang diberikan oleh Free Pascal. Dengan menggunakan tipe ini, kita tinggal menggunakan perintah readln, writeln, copy, dan pos. Hal ini dimungkinkan karena ANSIString dapat menyimpan untaian karakter dengan jumlah karakter hampir tak terhingga, tapi tetap mempertahankan sifat-sifat yang dimiliki oleh tipe variabel string. Dengan demikian, kita cukup melakukan satu iterasi saja, yaitu iterasi untuk mencari posisi dimana ada kemunculan substring pada string. Walaupun demikian, waktu yang dibutuhkan oleh program ini lebih kurang sama dengan program di atas karena perintah pos sendiri melakukan iterasi untuk mencari posisi substring. Di bawah ini adalah contoh source codenya:

```
program MenghitungPerulanganAlternatif(input, output);

var a, b : ANSIString;
    la, i, j, ans : integer;

begin
assign(input, 'HITUNG.IN');
assign(output, 'HITUNG.OUT');
reset(input);
rewrite(output);

readln(a);
readln(b);
la := length(a);
i := 1; j := 1; ans := 0;

{ cari substring sebanyak-banyaknya }
```

```
while j <> 0 do
  begin
  j := pos(b, copy(a, i, la));
  i := i + j;
  if j <> 0 then inc(ans);
  end;

writeln(ans);

close(input);
close(output);
end.
```

Seperti halnya soal "Cari Maksimum", ada 10 test case yang digunakan. Berikut rinciannya:

No.	Panjang String 1	Panjang String 2	Keterangan
1	1	1	Tidak ada kemunculan
2	1	2	Tidak ada kemunculan
3	495	1	Mengetes pembacaan input, dengan string 2 adalah sebuah spasi
4	10000	2	String 1 berisi hanya 3 jenis karakter
5	100	3	String 1 berisi hanya 3 jenis karakter
6	500	5	String 1 berisi hanya 2 jenis karakter, sementara string 2 berisi hanya sebuah jenis karakter
7	1520	20	String 1 berisi hanya 4 jenis karakter random dan disisipkan sebuah kemunculan dari string 2
8	10000	200	Mengetes kasus terbesar. String 1 dan 2 hanya terdiri dari sebuah jenis karakter
9	4517	40	Sama seperti 6, tapi string 2 berisi 2 buah jenis karakter
10	8188	100	Sama seperti 1, tapi string 2 juga berisi 3 jenis karakter

Perhiasan

(Soal ini pernah diberikan dalam OSN 2003, Balikpapan)
oleh : Ilham Kurnia, Alumni TOKI 2003

Nama Program:	**HIAS.PAS**
Batas *Run-time*:	**3 detik / test case**
Nama File Masukan:	**HIAS.IN**
Nama File Keluaran:	**HIAS.OUT**

"Be Jeweled" adalah sebuah permainan populer di komputer palm. Ada 81 batu mulia yang disusun sebagai matriks 9 × 9. Tujuan dari permainan ini adalah untuk memperoleh skor setinggi-tingginya dari hasil sekali penukaran dua batu mulia yang bersebelahan pada suatu susunan. Dua batu mulia yang bersebelahan dapat ditukarkan apabila setelah pertukaran, ada beberapa batu mulia yang dapat dihitung nilainya.

Penilaian dilakukan dengan cara sebagai berikut:
1. Bila ada tiga atau lebih batu mulia dari jenis yang sama berjejer secara horisontal, maka batu-batu mulia tersebut akan dihitung.
2. Bila ada tiga atau lebih batu mulia dari jenis yang sama berjejer secara vertikal, maka batu-batu mulia tersebut akan dihitung.
3. Bila ada batu mulia yang memenuhi syarat 1 dan 2, maka batu mulia tersebut hanya dihitung sekali saja.
4. Skor yang diperoleh = 5 * *T* ^ (banyak batu mulia yang terhitung – 3), dimana *T* adalah tingkat kesulitan dari susunan/matriks batu-batu mulia (1 ? *T* ? 50).

Agar mudah, semua batu mulia akan direpresentasikan sebagai huruf besar. Ambil contoh sub-matriks (dengan tingkat kesulitan 3) berikut:

A	B	C	D	E	F
B	D	R	P	R	A
C	P	P	R	P	B
D	B	C	P	B	C
E	A	B	P	A	D
F	D	F	E	F	E

Tukar pasangan yang diarsir.

A	B	C	D	E	F
B	D	R	R	R	A
C	P	P	P	P	B
D	B	C	P	B	C
E	A	B	P	A	D
F	D	F	E	F	E

Batu mulia yang diarsir masuk dalam hitungan. Untuk penukaran pasangan ini, skor yang diperoleh adalah $5 \times 3^{9-3} = 3645$

Tugas Anda adalah untuk membuat program yang mencari semua langkah terbaik (yaitu yang dapat menghasilkan skor tertinggi) dari sebuah matriks dengan ketentuan tertera di atas. Untuk memperjelas keluaran, maka hanya 2 macam pertukaran yang diperbolehkan: pertukaran dengan batu mulia sebelah kanan, atau pertukaran dengan batu mulia sebelah bawah.

FORMAT MASUKAN (Nama File: `HIAS.IN`)

Masukan terdiri dari 10 baris. Baris pertama berisi bilangan bulat *T*, sementara pada sembilan baris lainnya, setiap barisnya terdiri atas tepat 9 karakter. Setiap karakter melambangkan sebuah jenis batu mulia. Anda dapat berasumsi bahwa pa da masukan tidak ada tiga atau lebih batu mulia yang telah berjejer secara horisontal maupun vertikal.

CONTOH MASUKAN

```
3
AAJXXEXXS
ASAJAXJEX
EJEXAXSJX
SAJXEJSJA
SXESJEJEX
AESJSSEXA
EJEEAAJSJ
EEAEXJASJ
AEXASSXJS
```

FORMAT KELUARAN (Nama File: `HIAS.OUT`)

Apabila tidak ditemukan satupun langkah pertukaran, maka pada keluaran hanya tulis "`tidak ada`" pada baris pertama. Bila ada langkah pertukaran, pada baris pertama tulis skor maksimum yang dapat diperoleh untuk matriks pada masukan. Pada baris-baris selanjutnya, tulis semua langkah terbaik yang ada, terurut membesar berdasarkan nomor baris, kemudian nomor kolom, dan terakhir arah (bawah terlebih dahulu). Formatnya adalah *Nomor_baris Nomor_kolom* `bawah/kanan`. Setiap angka dipisahkan oleh tepat satu spasi.

CONTOH KELUARAN

```
135
6 2 bawah
7 1 kanan
9 8 kanan
```

CONTOH MASUKAN 2

```
1
ABCDEFGHI
IHGFXDCBA
ABCDEFGHI
IHGFXDCBA
ABCDEFGHI
IHGFXDCBA
ABCDEFGHI
IHGFXDCBA
ABCDEFGHI
```

CONTOH KELUARAN 2

```
tidak ada
```

Catatan

Ada kemungkinan variabel dengan tipe `integer`, atau `cardinal` tidak dapat memuat dengan baik hasil keluaran untuk sejumlah kecil test case.

3. Pembahasan Soal "Perhiasan"

Manusia dapat dengan mudah melihat jawaban dari suatu papan pada permainan "Be Jeweled" dengan mudah karena manusia dapat melihat sesuatu sebagai suatu keseluruhan. Komputer, di lain hal, tidak dapat. Pada satu saat, hanya ada satu hal yang bisa dilakukan komputer. Akan tetapi, komputer bisa melakukannya dalam frekuensi yang tinggi. Dengan demikian, bila kita bisa secara sistematis atau tersusun per langkah mengetahui cara melakukan pencarian jawaban, maka kita dapat melakukan bisa menerapkan langkah-langkah tersebut dalam komputer.

Dalam hal ini, kita pertama-tama melihat proses-proses kecil apa saja yang akan diperlukan yang dapat memudahkan kita. Berikut adalah dua proses yang sekiranya dapat membantu program dalam menyelesaikan masalah:

1. Proses menukar isi 2 variabel alias *swapping.*
2. Proses menghitung berapa banyak huruf yang sama berjejer secara horizontal maupun vertikal dengan acuan suatu lokasi.

Untuk proses yang pertama kita cukup membuat sebuah prosedur yang dengan mengandalkan suatu variabel tambahan untuk menukarkan isi dua variabel seperti yang dicontohkan di bawah:

```
procedure swap(var x : tipe_variabel; var y : tipe_variabel);
var tmp : tipe_variabel;

begin
tmp := x; x := y; y := tmp;
end;
```

Proses kecil kedua dapat dilakukan ke empat arah: atas, bawah, kanan, dan kiri. Keempat arah ini dapat kita kelompokkan menjadi 2 kelompok: mendatar dan menurun. Yang dimaksud dengan arah di sini adalah ke mana kita akan melakukan iterasi untuk mengecek kesamaan karakter dengan posisi di awal. Berikut sebagian kodenya:

```
function hitung (brs, klm : integer; arah : tipe_arah) : integer;
var i, j : integer;

begin
if arah = mendatar then
  begin
  { ke kiri }
  if matriks[brs, klm - 1] = matriks[brs, klm] then
    begin
    i := klm - 1;
    while i > 1 do
      begin
      if matriks[brs, i - 1] <> matriks[brs, klm] then break;
      dec(i);
      end;
    end
  else i := klm;

  { ke kanan }
  if matriks[brs, klm + 1] = matriks[brs, klm] then
```

```
    begin
    j := klm + 1;
    while j < klm - 1 do
      begin
      if matriks[brs, j + 1] <> matriks[brs, klm] then break;
      inc(j);
      end;
    end
  else j := klm;

  hitung := j - i + 1;
  end
else if arah = menurun then
  ke atas dan ke bawah sama dengan ke kiri dan ke kanan pada
  cara di atas, tapi yang diubah adalah brs, bukan klm.
end;
```

Perhitungan secara kasar dan cepat menunjukkan bahwa ada maksimal sebanyak $9^2 * 2$ jawaban. Kita cukup mengecek setiap posisi ditukarkan dengan yang di kanan dan di bawahnya (bila memungkinkan), dengan cara menghitung banyaknya huruf yang berjejer pada suatu kelompok arah. Dengan menggunakan aturan-aturan yang telah disebutkan, kita tinggal menghitung banyaknya karakter yang valid. Namun demikian, untuk mendapatkan nilai penuh, kita masih membutuhkan sebuah prosedur lagi untuk menghitung nilai akhir.

Prosedur penghitungan hasil akhir ini tidak lain adalah prosedur untuk menghitung nilai perkalian dengan hasil sebuah bilangan bulat yang sedemikian besar, yang hingga saat ini tidak dapat dimuat pada tipe data konvensional (byte, integer, longint, dsb.). Prosedur ini dapat diandaikan sebagai simulasi cara perkalian yang sering kita lakukan pada waktu kita duduk di sekolah dasar. Implementasi dari prosedur ini biasanya dilakukan dengan menggunakan sebuah array bilangan bulat sementara untuk menampung hasil dari perkalian. Setelah nilai maksimum dihitung, maka kita tinggal mengeluarkan hasilnya saja.

Spesifikasi Test Case

No.	Level	Keterangan
1	1	Tidak ada langkah yang valid
2	3	Mendatar, 3 huruf sederet dengan tukar kanan dengan huruf terkiri
3	5	Mendatar, 3 huruf sederet dengan tukar bawah dengan huruf terkiri
4	7	Mendatar, 3 huruf sederet dengan tukar kanan dengan huruf terkanan
5	9	Mendatar, 3 huruf sederet dengan tukar bawah dengan huruf terkanan
6	11	Menurun, 3 huruf sederet dengan tukar kanan dengan huruf teratas
7	13	Menurun, 3 huruf sederet dengan tukar bawah dengan huruf teratas
8	15	Menurun, 3 huruf sederet dengan tukar kanan dengan huruf terbawah
9	17	Menurun, 3 huruf sederet dengan tukar bawah dengan huruf terbawah
10	19	Mendatar, 3 huruf sederet dengan tukar bawah di pojok kanan bawah
11	21	Menurun, 3 huruf sederet dengan tukar kanan di pojok kanan bawah
12	23	Huruf L
13	25	Huruf L diputar 90°
14	27	Tanda plus, lebar 1 huruf tanpa bagian tengahnya
15	29	Tanda plus, lebar 2 huruf tanpa bagian tengahnya
16	31	Papan catur
17	33	Papan catur hanya kotak putih
18	35	Huruf T, dengan atasan tebal 2 huruf

19	37	Beberapa huruf X
20	39	Huruf t, dengan atasan tebal 2 huruf
21	41	Papan catur, dengan besar kotak 1 × 2
22	43	Kombinasi acak, bentuk gunung
23	45	Huruf O
24	47	Kombinasi acak
25	49	Plus besar, dengan tebal garis horizontalnya 2 huruf

Berikut adalah model solusinya. Mohon dicatat bahwa model di bawah ini adalah kode lomba, sehingga penulisan kodenya tidak begitu rapi dan tidak ada komentar. Bila ada kesempatan, penulis akan mengubah beberapa bagian untuk ditulis secara lebih terstruktur.

```
program Perhiasan (input, output);

Const InFile = 'HIAS.IN';
      OutFile = 'HIAS.OUT';
      MaxDigit = 30;
      MaxAns = 200;
      Size = 9;

Type TPos = record
       x, y, t : integer;
       end;

var mat : array[1..Size] of string[Size];
    str : array[1..MaxAns] of TPos;
    lvl, ans, cts, ctv : integer;
    value : array[1..MaxDigit] of integer;

procedure readdata;
var i : integer;

begin
readln(lvl);
for i := 1 to Size do readln(mat[i]);
end;

procedure swap(x1, y1, x2, y2 : integer);
var tmp : char;

begin
tmp         := mat[x1, y1];
mat[x1, y1] := mat[x2, y2];
mat[x2, y2] := tmp;
end;

function cekLeft(y, x : integer; v : char) : integer;
var i, tx, ty : integer;

begin
tx := x - 1; ty := y;
i := 0;
while (tx > 0) and (mat[ty, tx] = v) do begin inc(i); dec(tx); end;
cekLeft := i;
end;
```

```
function cekRight(y, x : integer; v : char) : integer;
var i, tx, ty : integer;

begin
i := 0; tx := x + 1; ty := y;
while (tx <= size) and (mat[ty, tx] = v) do begin inc(i); inc(tx); end;
cekRight := i;
end;

function cekUp(y, x : integer; v : char) : integer;
var i, tx, ty : integer;

begin
i := 0; tx := x; ty := y - 1;
while (ty > 0) and (mat[ty, tx] = v) do begin inc(i); dec(ty); end;
cekUp := i;
end;

function cekDown(y, x : integer; v : char) : integer;
var i, tx, ty : integer;

begin
i := 0; ty := y + 1; tx := x;
while (ty <= size) and (mat[ty, tx] = v) do begin inc(i); inc(ty); end;
cekDown := i;
end;

function countDown(r, c : integer) : integer;
var i, j, k : integer;

begin
j := 0; k := 0;
i := cekLeft(r, c, mat[r, c]) + cekRight(r, c, mat[r, c]);
if (i > 1) then j := i;
i := cekUp(r, c, mat[r, c]);
if (i > 1) then j := j + i;

i := cekLeft(r + 1, c, mat[r + 1, c]) + cekRight(r + 1, c, mat[r + 1, c]);
if (i > 1) then k := i;
i := cekDown(r + 1, c, mat[r + 1, c]);
if (i > 1) then k := k + i;
if (j > 1) and (k > 1) then countDown := j + k + 2
else if (j > 1) or (k > 1) then countDown := j + k + 1
else countDown := 0;
end;

function countRight(r, c : integer) : integer;
var i, j, k : integer;

begin
j := 0; k := 0;
i := cekUp(r, c, mat[r, c]) + cekDown(r, c, mat[r, c]);
if (i > 1) then j := i;
i := cekLeft(r, c, mat[r, c]);
if (i > 1) then j := j + i;

i := cekUp(r, c + 1, mat[r, c + 1]) + cekDown(r, c + 1, mat[r, c + 1]);
if (i > 1) then k := i;
```

```
i := cekRight(r, c + 1, mat[r, c + 1]);
if (i > 1) then k := k + i;

if (j > 1) and (k > 1) then countRight := j + k + 2
else if (j > 1) or (k > 1) then countRight := j + k + 1
else countRight := 0;
end;

procedure countAnswer;
var i, j : integer;

begin
value[1] := 5; ctv := 1;
for i := 1 to ans - 3 do
  begin
  for j := 1 to ctv do value[j] := value[j] * lv;
  j := 1;
  while (j <= ctv) or (value[j] <> 0) do
    begin
    value[j + 1] := value[j + 1] + value[j] div 10;
    value[j] := value[j] mod 10;
    inc(j);
    end;
  if (value[j] <> 0) then ctv := j else ctv := j - 1;
  end;
end;

procedure process;
var i, j, k, l : integer;

begin
for i := 1 to Size do
  for j := 1 to Size do
    begin
    {down}
    if (i < Size) and (mat[i, j] <> mat[i + 1, j]) then
      begin
      swap(i, j, i + 1, j);
      k := countDown(i, j);

      if k = ans then
        begin
        inc(cts);
        str[cts].x := i;
        str[cts].y := j;
        str[cts].t := 0;
        end
      else if k > ans then
        begin
        ans := k;
        cts := 1;
        str[1].x := i;
        str[1].y := j;
        str[1].t := 0;
        end;
      swap(i, j, i + 1, j);
      end;
```

```
    {right}
    if (j < Size) and (mat[i, j] <> mat[i, j + 1]) then
      begin
      swap(i, j, i, j + 1);
      k := countRight(i, j);

      if k = ans then
        begin
        inc(cts);
        str[cts].x := i;
        str[cts].y := j;
        str[cts].t := 1;
        end
      else if k > ans then
        begin
        ans := k;
        cts := 1;
        str[1].x := i;
        str[1].y := j;
        str[1].t := 1;
        end;
      swap(i, j, i, j + 1);
      end;
    end;
end;

procedure writeOutput;
var i : integer;

begin
if ans = 0 then begin writeln('tidak ada'); exit; end;
for i := ctv downto 1 do write(value[i]);
writeln;
for i := 1 to cts do
  begin
  write(str[i].x, ' ', str[i].y, ' ');
  if str[i].t = 1 then writeln('kanan') else writeln('bawah');
  end;
end;

begin
assign(input, InFile);
assign(output, OutFile);
reset(input);
rewrite(output);

readdata;
process;
countAnswer;
writeOutput;

close(input);
close(output);
end.
```

Sebuah catatan singkat, soal ini diambil dari seleksi Tim Olimpiade Komputer Hong Kong tingkat junior 2003 dengan sedikit modifikasi.

Contoh Soal 4. Nilai Terbesar

Pembahasan : Fajran Iman Rusadi, Alumni TOKI

Source Code: MAXROUTE.PAS/C/CPP
Input: MAXROUTE.IN
Output: MAXROUTE.OUT
Waktu Eksekusi: 2 detik

Perhatikan segitiga di bawah ini. Tulis sebuah program yang menghitung nilai terbesar dari angka-angka yang ada dalam jalur yang dimulai dari atas sampai ke bawah segitiga. Setiap langkah dalam jalur dapat secara diagonal ke bawah kiri atau kanan.

```
    7
   3 8
  8 1 0
 2 7 4 4
4 5 2 6 5
```

Jalur yang menghasilkan jumlah terbesar adalah 7-3-8-7-5 yang menghasilkan 30.

FORMAT INPUT

Baris pertama berisi sebuah bilangan bulat R (1 <= R <= 100) yang menyatakan ukuran segitiga. Baris ke dua sampai R+1 berisi bilangan-bilangan bulat (0 <= N <= 99) pada setiap baris segitiga.

CONTOH INPUT (File: MAXROUTE.IN)

```
5
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
```

FORMAT OUTPUT

Output hanya 1 baris yang terdiri dari 1 integer M, merupakan jumlah terbesar yang dapat dibuat dari spesifikasi di atas.

CONTOH OUTPUT (File: MAXROUTE.OUT)

```
30
```

Nilai Terbesar - Pembahasan

Dalam soal ini kita harus mencari jalur yang harus dilalui sehingga kita akan mendapatkan jumlah angka yang terbesar.

Ada beberapa cara yang dapat digunakan untuk menyelesaikan permasalahan ini. Pertama, kita benar-benar mencari semua jalur yang mungkin dilalui dan membandingkan jumlah yang didapat untuk setiap jalur.

```
    7
   3 8
  8 1 0
 2 7 4 4
4 5 2 6 5
```

Pada contoh di atas, akan ada 2^(n-1) dengan n = 5 yaitu 16 buah kemungkinan jalur. Jika kita lihat pada spesifikasi input, ukuran segitiga dapat mencapai nilai 100 sehingga akan ada 2^99 = 633825300114114700748351602688 kemungkinan jalur. Sungguh angka yang sangat besar jika kita harus mencobanya satu-persatu.

Oleh karena itu, kita harus memikirkan cara lain yang dapat digunakan untuk menyelesaikan permasalahan ini dengan jumlah perulangan sesedikit mungkin.

Cara yang kedua adalah dengan menggunakan teknik *dynamic programming*, yaitu suatu cara penyelesaian masalah dengan menyelesaikan masalah yang lebih kecil terlebih dahulu sehingga hasilnya dapat digunakan untuk menyelesaikan masalah yang lebih besar lagi.

Jika kita perhatikan, pada segitiga dengan tiga buah angka (n = 2), maka jalur yang dipilih oleh angka yang ada di atas adalah jalur yang menuju ke angka yang paling besar dari dua angka yang berada di bawah.

```
 2
4 5
```

Pada susunan segitiga di atas, jalur yang dipilih oleh angka 2 adalah jalur yang menuju angka 5, karena dari kedua angka yang ada di bawah (yaitu 4 dan 5), angka yang paling besar adalah angka 5.

Untuk memperoleh jumlah angka pada jalur yang dilalui, kita cukup menjumlahkan angka yang berada di atas dengan angka dibawahnya yang kita pilih. Pada contoh di atas, jumlah angka-angka pada jalur yang diambil oleh angka 2 adalah 2 + 5 = 7.

Dengan berbekal teknik di atas, kita dapat mencari nilai terbesar yang dapat dicapai dari semua kemungkinan jalur yang ada pada sebuah segitiga angka. Kita hanya perlu menjalankan teknik tersebut baris demi baris dari baris paling bawah segitiga.

Secara umum, teknik yang akan dipakai adalah dengan menjumlahkan setiap angka yang ada dengan angka terbesar yang ada di bawah angka tersebut. Proses ini dimulai dari angka-angka yang berada pada baris paling bawah dan terus berlanjut sampai angka yang berada di paling atas.

Berikut adalah simulasi penerapan teknik tersebut pada contoh input yang diberikan.

```
1.
        7
      3   8
    8   1   0
  2   7   4   4
4   5   2   6   5


2.
        7
      3   8
    8   1   0
  7   12  10  10
4   5   2   6   5
```

```
3.
        7
      3   8
    20  13  10
  7   12  10  10
4   5   2   6   5


4.
        7
      23  21
    20  13  10
  7   12  10  10
4   5   2   6   5


5.
        30
      23  21
    20  13  10
  7   12  10  10
4   5   2   6   5
```

Setelah kita sampai pada angka paling atas, kita akan mendapatkan jumlah terbesar yang kita cari, yaitu angka yang berada pada posisi paling atas tersebut, dalam contoh ini adalah 30.

Source Code :

```
var
   f : TEXT;
   f2 : TEXT;
   n : integer;
   x,y : integer;
   segitiga : array[1..100,1..100] of integer;
   sampah : char;
   max : integer;
begin
   assign(f,'MAXROUTE.IN');
   reset(f);
   read(f, n);

   for x := 1 to n do
   begin
      for y := 1 to x do
      begin
         read(f,segitiga[x][y]);
      end;
   end;
   close(f);

   for x := n-1 downto 1 do
   begin
      for y := 1 to x do
      begin
         if (segitiga[x+1][y] < segitiga[x+1][y+1]) then
          max := segitiga[x+1][y+1]
          else
          max := segitiga[x+1][y];
       segitiga[x][y] := segitiga[x][y] + max;
      end;
```

```
   end;

   assign(f2,'MAXROUTE.OUT');
   rewrite(f2);
   writeln(f2,segitiga[1][1]);
   close(f2);
end.
```

Contoh Soal 5.Ekspresi Aritmatika

Pembahasan : Fajran Iman Rusadi, Alumni TOKI

Source Code: EKSPRESI.PAS/C/CPP
Input: EKSPRESI.IN
Output: EKSPRESI.OUT
Waktu Eksekusi: 1 detik

Program anda harus dapat membaca string masukan yang berisi ekspresi aritmetika yang terdiri atas operator ^, *, /, +, -, dan menuliskan urutan pengerjaannya yang benar. Misalnya:

```
a-b+c/d*e/f^g-h*j
```

Untuk menentukan urutan pengerjaan dalam penulisannya, operator-operator tersebut diberikan tingkat prioritas: ^ paling tinggi, kemudian * dan / pada prioritas yang sama, dan terakhir + dan - pada prioritas yang sama. Dengan adanya tingkatan prioritas ini maka f^g harus dikerjakan sebelum e/f atau g-h. Jika prioritas sama, maka yang disebelah kiri akan dikerjakan lebih dahulu dari yang di sebelah kanan. Untuk contoh di atas c/d dikerjakan terlebih dahulu dari pada d*e.

Dengan menggunakan nama variabel sementara **xi** untuk menerima hasil pengerjaaan suatu operasi, maka salah satu urutan pengerjaan ekspresi tersebut adalah:

```
x1=a-b
x2=c/d
x3=x2*e
x4=f^g
x5=x3/x4
x6=x1+x5
x7=h*j
x8=x6-x7
```

FORMAT INPUT

Masukan terdiri dari satu baris teks ekspresi aritmetika dengan panjang < 256 karakter. Operator pangkat ditulis dengan simbol '^', operator kali dengan simbol '*', operator bagi dengan simbol '/', operator tambah dengan simbol +', dan operator kurang dengan simbol '-'.

Operand-operand-nya sendiri adalah menggunakan karakter huruf tunggal (a-z, A-Z) untuk memudahkan anda membaca masukan. Dalam ekspresi tidak ada karakter spasi atau karakter lainnya selain huruf atau karakter simbol operator tersebut di atas.

FORMAT OUTPUT

Keluaran berisikan baris-baris operasi untuk mengerjakan ekspresi masukan yang dibantu oleh variabel-variabel sementara **xi**.

Agar keluaran menjadi unik maka urutan sedapat mungkin dari kiri ke kanan ekspresi kecuali kalau terkait dengan prioritas. Misalnya a-b harus ditulis lebih dahulu dari c/d karena a-b tidak begantung pada hasil c/d. Variabel-variabel sementara **xi** dituliskan sebagai karakter **x** dan bilangan **i** dengan **i** membesar dari baris pertama ke baris terakhir.

CONTOH INPUT (File: EKSPRESI.IN)
```
a-b+c/d
```

CONTOH OUTPUT (File: EKSPRESI.OUT)
```
x1=a-b
x2=c/d
x3=x1+x2
```

Contoh 2, EKSPRESI.IN
```
c/d*e/f^g
```

Contoh 2, EKSPRESI.OUT
```
x1=c/d
x2=x1*e
x3=f^g
x4=x2/x3
```

Contoh 3, EKSPRESI.IN
```
a-b+c/d*e/f^g-h*j
```

Contoh 3, EKSPRESI.OUT
```
x1=a-b
x2=c/d
x3=x2*e
x4=f^g
x5=x3/x4
x6=x1+x5
x7=h*j
x8=x6-x7
```

Ekspresi Aritmatika - Pembahasan

Untuk mengerjakan soal semacam ini, kita dapat menggunakan bantuan struktur data stack (tumpukan). Stack merupakan sebuah tempat menyimpan data dengan aturan kita bisa memasukkan data dengan bebas dan ketika ingin mengambil data, data yang dapat kita ambil adalah data yang terakhir kali dimasukkan. Istilah kerennya adalah LIFO (Last In First Out).

Dalam suatu ekspresi aritmatika, simbol-simbol atau karakter-karakter yang ada dapat kita bedakan menjadi dua jenis, yaitu operator dan operand. Operator adalah simbol yang menandakan suatu operasi aritmatika seperti penjumlahan, pengurangan, perkalian, dan pembagian. Sedangkan operand adalah simbol yang dikenai suatu operasi oleh suatu operator, dalam hal ini adalah angka di sekeliling operator.

Kita dapat menggunakan dua buah stack untuk menampung operand dan operator yang ada dalam suatu ekspresi aritmatika. Aturan yang dipakai adalah, untuk setiap dua operand yang berada pada puncak stack (Top of Stack/TOS) operand akan dikenai operasi oleh operator yang berada pada puncak stack operator. Hasil dari operasi ini akan menjadi suatu simbol tertentu yang merupakan sebuah operand juga, sehingga harus dimasukkan kembali ke dalam stack operand. Proses pengeluaran dua operand dan satu operator untuk menghasilkan sebuah operand baru, dalam pembahasan ini, kita istilahkan dengan POP.

POP akan dilakukan jika kita menemukan dua buah kondisi.

1. Ketika operator yang berada pada Top of Stack dari stack operator memiliki prioritas yang lebih besar atau sama dengan prioritas operator yang baru kita baca dari ekspresi aritmatika. Proses ini bertujuan untuk mendahulukan operasi-operasi yang memiliki prioritas yang lebih besar atau sama.

2. Ketika sudah tidak ada lagi input yang dapat dibaca. Proses ini dilakukan untuk mengosongkan stack sekaligus mengeluarkan operasi-operasi aritmatika yang belum dilakukan (karena memiliki prioritas yang lebih kecil).

Setelah menganalisa cara mengerjakan soal, sekarang kita mulai membuat potongan-potongan source code solusi.

Operasi POP

Operasi ini dilakukan dengan cara mengeluarkan dua buah simbol pada stack operand dan satu buah simbol pada stack operator, kemudian mengoperasikannya sehingga didapatkan satu buah operand baru yang dapat dimasukkan kembali ke dalam stack operand.

```
Procedure POP;
Var
       OperandBaru         : String;
       Operand1, Operand2  : String;
       Operator            : String;
Begin

       OperandBaru := 'x' + IndeksOperandBaru;

       { ambil 2 operand dan 1 operator }
       Operand1 := StackOperand[OperandTOS];
       OperandTOS := OperandTOS - 1;

       Operand2 := StackOperand[OperandTOS];
       OperandTOS := OperandTOS - 1;

       Operator := StackOperator[OperatorTOS];
       OperatorTOS := OperatorTOS - 1;

       { cetak hasil }
       WriteLn(OperandBaru, '=', Operand1, Operator, Operand2);

       { masukkan operand baru }
       OperandTOS := OperandTOS + 1;
       StackOperand[OperandTOS] := OperandBaru;

End;
```

Nilai dari variabel OperatorBaru berisi sebuah simbol 'x' dan sebuah indeks yang akan terus bertambah setiap prosedur POS dipanggil.

OperandTOS dan OperatorTOS merupakan sebuah variabel yang menunjukkan indeks array yang merupakan Top of Stack.

Operasi pembacaan ekspresi aritmatika

Misalkan kita memiliki sebuah variabel Ekspresi dengan tipe String yang berisi sebuah ekspresi aritmatika. Kita akan membaca isi dari variabel ini karakter per karakter. Ketika karakter yang dibaca

merupakan simbol operand, simbol tersebut langsung kita masukkan ke dalam stack operand. Sedangkan ketika simbol operator yang dibaca, kita harus membandingkan prioritas operator tersebut dengan operator-operator yang berada di dalam stack operator. Ketika operator dalam stack memiliki prioritas yang lebih kecil besar atau sama dengan, prosedur POP akan dipanggil. Proses membandingkan ini berakhir ketika tidak ada lagi operator yang berada di dalam stack atau ketika operator dalam stack memiliki prioritas yang lebih kecil daripada operator yang baru dibaca dari variabel Ekspresi.

```
For i := 1 To Length(Ekspresi) do
Begin

        If Ekspresi[i] in [a..z,A..Z] Then
        Begin

                { masukkan operand ke dalam stack }
                OperandTOS := OperandTOS + 1;
                StackOperand[OperandTos] := Ekspresi[i];
        End
        Else
        Begin
                { dahulukan operasi yang lebih tinggi atau sama prioritasnya }
                While (OperatorTOS > 0) AND (PrioritasOperatorBaruLebihKecil) Then
                        POP;

                { masukkan operator ke dalam stack }
                OperatorTOS := OperatorTOS  + 1;
                StackOperator[OperatorTOS] := Ekspresi[i];

        End;

End;
```

Operasi pengosongan stack

Setelah ekspresi aritmatika dibaca semua, kita harus memastikan stack kembali kosong agar seluruh operasi benar-benar dilakukan.

```
While OperatorTOS > 0
        POP;
```

Hasil

Hasil dari langkah-langkah yang telah dibahas di atas, dapat dilihat dalam source code solusi.

Dalam source code solusi, ada satu buah stack lagi yang digunakan untuk menampung nilai prioritas dari suatu operator, sehingga proses membandingkan prioritas dapat lebih mudah dilakukan.

Source Code :

```
Program EkspresiAritmatika;

Const
    TAMBAH              = 1;
    KURANG              = 1;
    KALI                = 2;
    BAGI                = 2;
    PANGKAT             = 3;
```

```
Var
    StackOperand        : Array[1..150] of String;
    StackOperator       : Array[1..150] of Char;
    StackOperatorKode   : Array[1..150] of Byte;
    Indeks              : Integer;
    F                   : Text;
    StrInput            : String;
    i                   : Integer;
    PosisiOperand       : Integer;
    PosisiOperator      : Integer;
    Temp                : String;
    KodeOperator        : Integer;

Procedure Pop;
Begin
    Indeks := Indeks + 1;
    Str(Indeks, Temp);
    Temp := 'x' + Temp;

    WriteLn(F,          Temp,          '=',          StackOperand[PosisiOperand-1],
StackOperator[PosisiOperator], StackOperand[PosisiOperand]);

    PosisiOperand := PosisiOperand - 1;
    StackOperand[PosisiOperand] := Temp;

    PosisiOperator := PosisiOperator - 1;
End;

Begin
    Assign(F, 'EKSPRESI.IN');
    Reset(F);
    ReadLn(F, StrInput);
    Close(F);

    Assign(F, 'EKSPRESI.OUT');
    ReWrite(F);

    PosisiOperand := 0;
    PosisiOperator := 0;
    For i := 1 To Length(StrInput) do
    Begin
        If StrInput[i] in ['a'..'z','A'..'Z'] Then
        Begin
            PosisiOperand := PosisiOperand + 1;
            StackOperand[PosisiOperand] := StrInput[i];
        End
        Else
        Begin
            Case StrInput[i] of
            '+': KodeOperator := TAMBAH;
            '-': KodeOperator := KURANG;
            '*': KodeOperator := KALI;
            '/': KodeOperator := BAGI;
            '^': KodeOperator := PANGKAT;
            End;

            While (PosisiOperator > 0) AND (StackOperatorKode[PosisiOperator] >=
KodeOperator) do
                Pop;

            PosisiOperator := PosisiOperator + 1;
            StackOperator[PosisiOperator] := StrInput[i];
```

```
            StackOperatorKode[PosisiOperator] := KodeOperator;
        End;
    End;

    While PosisiOperator > 0 do
        Pop;

    Close(F);

End.
```

Contoh Soal 6. Faktorial

Source Code: FAKTOR.PAS/C/CPP
Input: FAKTOR.IN
Output: FAKTOR.OUT
Waktu Eksekusi: 1,5 detik

Faktorial dari suatu bilangan N didefinisikan sebagai berikut:

N! = 1 * 2 * 3 * 4 * ... * N-1 * N

Contohnya, 12! = 1*2*3*4*5*6*7*8*9*10*11*12 = 479001600

Digit paling kanan yang bukan nol dari 12! adalah 6 dan ada 2 buah nol mengikutinya.

Buatlah program yang dapat menghitung digit paling kanan yang bukan nol dan jumlah angka nol yang mengikuti dari sebuah bilangan N!

FORMAT INPUT

Sebuah baris yang berisi sebuah bilangan bulat N (1 <= N <= 1000000)

CONTOH INPUT (File: FAKTOR.IN)

```
12
```

FORMAT OUTPUT

Sebuah baris berisi dua buah bilangan bulat X dan Y yang dipisahkan oleh spasi. X berisi digit paling kanan yang bukan nol dari N! dan Y adalah jumlah nol yang mengikuti X pada N!

CONTOH OUTPUT (File: FAKTOR.OUT)

```
6 2
```

Contoh Soal 7. Teka-Teki Silang

Source Code: TEKATEKI.PAS/C/CPP
Input: TEKATEKI.IN
Output: TEKATEKI.OUT
Waktu Eksekusi: 2 detik

Pada sebuah permainan teka teki silang, kita akan dihadapkan oleh kotak-kotak yang harus kita isi dengan jawaban dari pertanyaan yang ada. Jawaban bisa diisi secara mendatar (horizontal) dan menurun (vertikal).

Jika diberi suatu kotak-kotak teka-teki silang, kita dapat dengan mudah melihat ada berapa jawaban yang dapat harus diisi pada kotak-kotak tersebut. Diasumsikan satu jawaban minimal terdiri dari 2 huruf.

Contoh: (simbol '-' melambangkan kotak kosong dan '#' melambangkan kotak hitam yang tidak dapat diisi)

```
- - - - #
- - # # -
- - - - -
- # # - -
# - - - -
```

Pada kotak isian di samping, kita dapat melihat ada lima buah jawaban mendatar dan empat buah jawaban menurun.

Buatlah program yang dapat menganalisa suatu kotak-kotak isian teka-teki silang untuk menghitung ada berapa jawaban (mendatar dan menurun) yang dapat diisi pada kotak-kotak tersebut.

FORMAT INPUT

Baris pertama: Sebuah bilangan bulat N, 3 <= N <= 200, N adalah ukuran panjang sekaligus lebar dari kotak teka-teki silang.

Baris ke-2 sampai ke N+1: Setiap baris terdiri dari N buah karakter '-' atau '#' yang melambangkan kotak teka-teki silang.

CONTOH INPUT (File: TEKATEKI.IN)

```
5
----#
--##-
-----
-##--
#----
```

FORMAT OUTPUT

Sebuah baris dengan 2 buah bilangan N dan M yang melambangkan jumlah jawaban mendatar dan menurun.

CONTOH OUTPUT

```
5 4
```