LAWS

of the

SOCIETY

for

LITERARY and SCIENTIFIC

IMPROVEMENT

ESTABLISHED

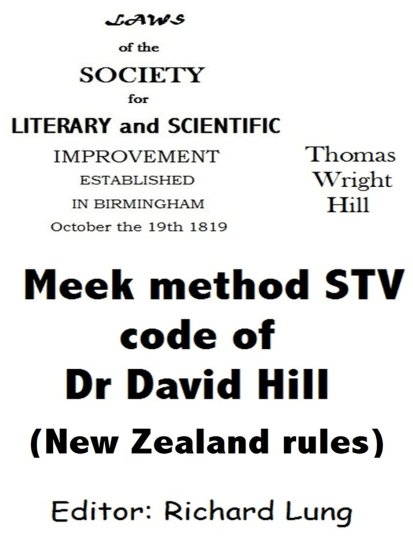
IN BIRMINGHAM

October the 19th 1819

Thomas
Wright
Hill

Meek method STV code of Dr David Hill (New Zealand rules)

Editor: Richard Lung


LAWS
of the
SOCIETY
for
LITERARY and SCIENTIFIC
IMPROVEMENT
ESTABLISHED
IN BIRMINGHAM
October the 19th 1819

Thomas
Wright
Hill

**Meek method STV
code of
Dr David Hill
(New Zealand rules)**

Editor: Richard Lung

Meek method STV code of Dr David Hill (New Zealand rules)

Thomas Wright Hill invented effective representation, and practised it, 200 years ago, in committee elections at his boys school. His descendant, Dr David Hill wrote the open source software, which led to the worlds first use of the Meek method computer count in official elections. An extension of its concept of Single Transferable Vote (STV) “keep values” is intended for an eventual programming of the FAB STV system.

It is well-known that clubs are run by cliques. (This is also true of governments.) Thomas Wright Hill knew this, in the first recorded use of transferable voting, by election rules to the Society for Literary and Scientific Improvement, 1819. (2019 the bicentenary of his remedy.) The main ingredients to the Single Transferable Vote (STV) are there: each candidate is elected, to several seats, on a quota or proportion of the votes, so all factions are represented. Surplus votes, to the quota, are transferable to the voters next preferences (on a random sample basis).

A direct descendant of Thomas Wright Hill and his famous son Rowland Hill, Dr David Hill wrote an open source code for the Meek method STV computer count. Adapted to New Zealand rules, this was a world first for official elections.

The editor (Richard Lung) is author of “FAB STV: Four Averages Binomial Single Transferable Vote.” This uses an extension of the key Meek method concept of the “keep value” (while discarding quota reduction and so-called premature exclusion of candidates).

Table of Contents.

Thomas Wright Hill, 1819: First known record of quota-preferential elections.

Rowland Hill witness of T.W. Hill schoolboys committee.

Dr David Hill, Meek STV program (New Zealand rules):

Preface (Stephen Todd).
Brief description of STV by Meek method.
Meek method and FAB STV (Richard Lung).
Points about the nzmeek program.
Notes to the NZ Meek Code.
NZ MEEK CODE

The Editor

Thomas Wright Hill, 1819.

First known record of quota-preferential elections (Editor).

Anthony Tuffin and David Hill were committee members of the Electoral Reform Society, elected by the Single Transferable Vote (STV) or quota-preferential method (as the Australians call it). Anthony promised David that, if he survived him, he would commemorate the bicentenary, in 2019, of the first known record of the quota-preferential principle of elections. This was the work of Davids direct ancestor, Thomas Wright Hill.
More information can be found on the STVaction website, which Anthony set-up after resigning from the committee of the Electoral Reform Society, because it was not devoting enough time to electoral reform. Specifically its original legal terms were to work for the Single Transferable Vote, tho the society name was changed from the Proportional Representation Society, a title still used by its sister organisation in Australia.

In 1819, a small Birmingham club, the "Society for Literary and Scientific Improvement" was based on the principle of equality, in participation and in representation on the committee. The subsequent quotation from the rule book shows an advanced awareness of standards of discourse, as practised in the sciences. A greater originality is in the election rules for equality of representation, which adopt a quota count, or an equal number of votes for each committee member to be elected. As all the members will not likely receive equal votes, proportional representation is democratically determined, by transfer of surplus votes to second and third or further orders of preference.

The problem of which voters next preferences are to count, in the transfer of surplus votes, is decided by lot. This is not a reliable method for a small club but it originates a widespread practise in STV elections, with large numbers of voters.
In statistics, the law of large numbers determines that a random sample, of all a prior prefered candidates voters, just sufficient to equal that candidates surplus votes (to the quota) will be representative of all those voters next preferences, to a high level of probability.

It is said that the Hill rules were not the genuine STV article. Hill appears not to give a formula for the quota, just requiring a quota of five members to elect a representative. Then again, the formula for the quota, in STV elections, is not the settled matter it is generally thought to be, even amongst specialists. Originally, Thomas Hare proposed the quota that goes by his name (Votes divided by seats: V/S). H R Droop wrote to him suggesting the quota that now goes by his name (Votes divided by one more than the number of seats: V/{S+1}). These quotas are respectively the maximum and minimum proportions of the the vote, required to elect candidates.

For large numbers of seats, it does not make much difference which quota is used. Hare did not heed Droop. Hare and Mill and HG Wells stipulated large constituencies. But politicians, anxious to reduce the competition, have relentlessly pressed for the smallest constituencies, usually single members (no competition at all for monopolistic representatives). In single member districts, of any size, the Hare quota does not work at all, because it requires all the votes to elect the one representative, whereas the Droop quota requires only half the voters. If both get exactly half, a random decision is made.

Both quotas are marginally undemocratic. In the single member case, the maximum quota can require a level of unanimity only achieved by some voters deference to the wishes of others. Whereas the minimum quota allows the election of a candidate who is not significantly prefered to the loser.

Hence, this editor (Richard Lung) proposed the average of the two quotas as more representative than either. As quotas form a harmonic series, their average is the harmonic mean. This is calculated by adding the inverted two quotas, dividing by two, and inverting the sum. Hence, the Harmonic Mean Quota proceeds:

$\{(S/V) + (S+1)/V\}/2 = (2S+1)/2V$.

Invert again for: $2V/(2S+1) = V/(S + 1/2)$, which is the HM quota.

The HM quota is the first of the four averages to Four Averages Binomial Single Transferable Vote (FAB STV).

Thought is not born fully grown. Hill laid down essentials (which most of the world has not grasped yet) to an evolution of election method, that is still in progress. To this day, scarcely anywhere in the world have representative elections been attained, to standards, that in other sciences, would be regarded as an acceptable efficiency.

Out of the tangle of contingencies, that arise from running a club, a young man , two hundred years ago, thought deeply, to take the first known steps to preferential equality of election. We all know that clubs tend to fall into the hands of a clique that put off others. Few realise that a remedy rests in more adequate election rules.

This early nineteenth century rule book does more than anticipate better representation for small clubs. Transferable voting is scalable to the nations of the world. Their governments are club cliques, maintaining themselves in power by restrictive practises, of two main kinds, either (or both) elective dictatorships (that don't properly recognise electoral equality in the count) and party oligarchies (that don't recognise votes of personal preference).

Hence, many ethnic minorities try to break away to form their own nations. Little fish in a big pond want to be big fish in a little pond. As John Stuart Mill elaborated, a few decades later, in the mid nineteenth century, that fission comes from mistaking maiorocracy, the tyranny of the majority, as democracy.

Society for Literary and Scientific Improvement (Thomas Wright Hill)

"It has been thought highly desirable, that every member of the Society should be, as nearly as possible, upon an equality, that all may feel alike interested in the success of the whole. In order to accomplish this important object, every regular auditor is expected. according to the rules of the Society, to deliver a lecture in his turn. Thus, instead of the Society being divided into two parties, one consisting of lecturers, the other of critics, every member feels himself called upon to listen to the others with candour and attention, as he is aware, that the time will come when he shall require the same consideration. from them."

"It will be observed also, on a perusal of the Laws, that each lecture is followed by a discussion. Thus care is insured on the part of the lectnrer that he shall not attempt a subject which he has not well studied; and an opportunity is given to every member to obtain an explanation of any thing advanced, which he may not have understood, or to express his opinions on the questions that may arise, and by these means, correct or confirm his own ideas. But the principal advantage of a discussion is, that it calls forth the individual exertion of every member, by inciting each to take a part in the general instruction, and thus afording constant inducements to private reading and study...."

"The details of management of a Society like this, may, on a superficial view, appear of little importance; those, however, who have had opportunities of closer examination, will, it is presumed, agree with the members if this Institution, in considering an attention to such particulars as necessary, not only to the well being, but to the permanent existence of an association, for whatever purpose it may be formed."

"With views like these, the Society for Literary and Scientific Improvement have been anxious to establish a mode of electing the Committee, that should secure (as nearly as possible) an accurate representation of the whole body: not only because it appeared reasonable that the members would feel interested in the welfare of the Institution, in proportion as the arrangements and regulations met their own views and wishes, but because experience proves that, owing to imperfect methods of choosing those who are to direct the affairs of a Society, the whole sway sometimes gets into the hands of a small party; and is exercised, perhaps unconsciously, in a way that renders many persons indifferent, and alienates others, until all becomes listlessness, decay, and dissolution...."

"The mode of election shall be as follows. A ticket shall be delivered to each member present, with his own name at the head of it, immediately under which he shall write the name of the member whom he may wish to represent him in the Committee. The votes thus given shall be delivered to the president, who, after having assorted them, shall report to the meeting the number of votes given for each nominee. Every one who has five votes shall be declared a member of the committee; if there are more than five votes given to any one person, the surplus of votes, (to be selected by lot) shall be returned to the electors whose names they bear, for the purpose of their making other nominations, and this process shall be repeated till no surplus votes remain, when all the inefficient votes shall be returned to the respective electors, and the same routine shall be gone through a second time, and also a third time if necessary; when if a number is elected, equal in all to one half of the number of which the committee should consist, they shall be a committee; and if at the end of the meeting the number is not filled up, by unanimous votes of five for each member of the committee, given by those persons whose votes were returned to them at the end of the third election, then this committee shall have the power, and shall be required, to choose persons to fill up their number; and the constituents of each member so elected shall, if necessary, be determined by lot."

Hill schoolboys committee.

Thomas Wright Hill suggested the principle of proportional representation, as recorded by Rowland Hill (of post office fame) teaching in his fathers school. Quoting Enid Lakeman (How Democracies Vote):

"...his pupils were asked to elect a committee by standing beside the boy they liked best. This first produced a number of unequal groups, but soon the boys in the largest groups came to the conclusion that not all of them were actually necessary for the election of their favourite and some moved on to help another candidate, while on the other hand the few supporters of an unpopular boy gave him up as hopeless and transferred themselves to the candidate they considered the next best."

"The final result was that a number of candidates equal to the number required for the committee were each surrounded by the same number of supporters, with only two or three boys left over who were dissatisfied with all those elected.

This is an admirable example of the use of STV."

[Editor: Hill invented the simplest, clearest and yet most advanced form of transferable voting. His example is an Interactive STV in action. It is more responsive, flexible and dynamic than any formal routine laboriously laid out for returning officers. If people cared as much about democracy, as they do about wars and a host of other destructions and distractions, they would scale-up Interactive STV with a mass computer count. This begs the question of the algorithm to be used. FAB STV, a continuation of Meek method, might be a starting point.]

Dr David Hill: Meek STV program
(New Zealand rules).

Table of contents

Preface

Courtesy of Shane Wood of InfoPower Ltd, via Stephen Todd, the open source code of Dr David Hill's Meek (New Zealand rules) STV program is included with three explanatory files from David Hill. It is written in Pascal, which I [Stephen Todd, retired civil servant] understand is regarded as pretty ancient now, when it comes to computer languages. For what it's worth, when I told Brian Wichmann [former editor of "Voting Matters"] (in an e-mail on 3 August 2018) that the calculator was being re-written, he said that his ideal implementation "would be to use the system which can be used to prove a system formally correct, the SPARK subset of Ada."

(The New Zealand Department of Internal Affairs are going to commission the re-write of the NZ STV calculator. At the moment, the calculator does not provide for the filling of extraordinary [casual] vacancies by recount, with the remaining elected candidates [at the original election] being guarded.)

Brief description of STV by Meek method.

Superficial level.

1. Votes are made by preference ordering.

2. Each voter is counted as having one vote which goes, at the first count, to the first preference.

3. A quota is calculated as the minimum number of votes needed for election. The quota will be reduced during the count, if the circumstances allow.

4. If a candidate achieves more than a quota, surplus votes are removed, and passed to other candidates according to the further choices of the relevant voters.

5. When there is no surplus available, the candidate who currently has fewest votes must be excluded. Votes assigned to that candidate are passed to other candidates according to the further choices of the relevant voters.

Detailed level.

1. Each candidate has an associated "keep value" which applies to every vote, and every portion of a vote, received by that candidate. Initially every candidate's keep value is 100%.

2. Each time the votes are counted, it is done in the following way: suppose that candidate A's keep value is 80%, candidate B's is 50% and candidate C's is 100%. Then a ballot paper listing CAB (in that order) would be counted as:

100% of a vote to C
Nothing to A or B (because C has taken the lot).

A ballot paper listing ABC (in that order) would be counted as:

80% of a vote to A
10% of a vote to B (i.e. 50% of the remaining 20%)
10% of a vote to C (i.e. 100% of the remainder).

A ballot paper listing BA (in that order) would be counted as:

50% of a vote to B
40% of a vote to A (i.e. 80% of the remaining 50%)
10% of a vote to non-transferable (because this remaining 10% has run off the end of the list).

3. After each count of the papers, the current quota is calculated as:

(votes - non-transferable) / (seats + 1).

4. Any candidate having a surplus over the quota is deemed elected, and given a new keep value, calculated as:

(previous keep value) times (current quota) divided by (current votes for candidate).

Thus, for example, a candidate who has 4/3 times the votes necessary for election, needs to keep only 3/4 of what that candidate previously kept.

5. After every such change, to one or more candidates, the votes are recounted using the new keep values.

6. If at the end of any count of the votes, the number deemed elected is less than the number of seats, but no candidate has a

surplus, the candidate having fewest votes is excluded, and that candidate's keep value is reset to 0%.
The votes are then recounted. (If an exclusion is necessary and two or more candidates have equal fewest votes, then the one who had fewest votes at the earliest point at which they had unequal votes is to be excluded. If there is still a tie, one of the tied candidates is to be chosen by lot for exclusion).

7. In section 6 above "no candidate has a surplus" must, in practice, be interpreted as a very small value rather than literally zero (in the present implementation 1/10000 of a vote as the total surplus is used). However, it is usually clear before this is reached that an exclusion will be required and which candidate it must be. In that case the exclusion may be made at once, giving a short cut which cannot change the final result.

Main differences between traditional STV and Meek method.

There are four main differences between traditional STV and Meek's method.

1. After an exclusion, the count behaves as if started again with the excluded candidate withdrawn. This works well, and gives greater consistency of results, when combined with the other features of the method, but it would be no good adopting that one on its own. In combination with traditional STV rules it causes trouble.

2. If the number of non-transferables increases for any reason, the quota is reduced to allow for it. This is not the same as merely electing without reaching the quota as at the final stage of traditional rules, because the new quota applies at once to all candidates, including those already-elected giving them all new surpluses to be

redistributed. This would cause far too much extra work to be practicable for hand-counting.

3. When appropriate, votes are passed to already-elected candidates. This implements precisely what each individual ballot paper says and, taken together with the other changes, does not (as might at first appear) waste votes, for the newly-created surpluses are available for further redistribution.

4. Short lists (those that do not number all the candidates and, in particular, those that give only a first preference) are treated in the same way as any others i.e. following what the ballot papers say, in proportion to the numbers involved. There is no need to divide into transferables and non-transferables and treat them differently.

Meek method and FAB STV

(Richard Lung).

Table of contents

Meek method STV is more than just a putting of traditional STV into computer language. It takes advantage of the power of electronic computation to make a more consistent (and therefore complicated) count of voters preferences, strictly in accordance with their orders of choice.

The traditional STV hand count is made managable by taking a few short-cuts. Once a popularly prefered candidate already has the elective proportion or quota of votes, the hand count simply passes over, to the next preference, any more votes they may receive.

Meek method keeps on counting all the accumulation of surplus votes, that elected candidates may further receive. This has the information value of keeping track, in full, of the relative popularity of elected candidates. This information is encoded in a simple arithmetic formula called the keep value.

At this point, FAB STV extends the use of the keep value, because it calculates the keep values of all candidates, even if they do not reach the quota for their election. The keep value, of candidates in deficit of a quota, measures their relative unelection.

This is vital to the working of FAB STV, which is not decided by a single count but by an average of more than one count, so that a candidate, not elected on one count, may do so much better, on the other count, that, on average, they are elected.

Yes, FAB STV is a statistic, which at bottom is what elections generally are. A difference is the use of up to four different averages, to arrive at the most representative result! The Binomial part of FAB STV governs this averaging of counts.

I should give notice that FAB STV does not just extend Meek method use of the keep value, it also discards two complicated features, quota reduction (which FAB STV does not and cannot work with) and exclusion of candidates, when election surplus votes run out. (Critics call this “premature exclusion.”)
The main thing in coding FAB STV is to keep the baby of "keep values" and throw out the bathwater of quota reduction and premature exclusion. (Quota reduction is peculiar to Meek method. Premature exclusion afflicts STV less than the rest of the worlds elections.)

Dr Hill produced some auxiliary files governing the Meek method exclusion of candidates. They are not included here, because FAB STV does not exclude candidates, or declare candidates elected, while the counting procedure is still on-going.

There are orders of FAB STV, governed by (a non-commutative version of) the binomial theorem. First order STV probably is considerably simpler than Meek method, and may be all that ordinary elections require. Higher order counts become exponentially more complicated, tho within the same algebraic rules.

The higher orders offer consistent in-depth analysis, that just might make for an excellent search engine or information retrieval algorithm, as a match for the exponential growth of human knowledge. But that could only be known on testing an operationalised FAB STV.

Voting methods for the High Street!

If customers, to a long line of shops, had existing voting methods, nearly all the worlds official electoral systems would allow them only in one shop in the entire high street! That is what the illiterate X vote is like. (Customers could be given multiple X votes but that would be like giving an exclusive ticket, to shop, to one predominant type or faction of customers.)

The simple plurality (FPTP) system for shoppers only gives custom to one out of a minimal choice of one or two shops.

The party list system only allows each customer one kind of shop, which one, being determined by the relevant trade association.

The single transferable vote allows shoppers to transfer their custom, in order of popularity, proportionally representing all the trades. STV is incomparably the best system for customers or voters.

But it has its limits and an evolved system can do considerably more. All the worlds official elections are, preferentially speaking, one-way streets. FAB STV would be the only preferential two-way street. I don't go into how the count works here, but the least preferences represent the shops that some customers never frequent. (For instance, some people might never drink alcohol or place bets.) Many shops, people might go in, but never buy from. These are analogous to voters abstentions, and FAB STV records and counts them.

In short, FAB STV is unique in recording all the preferential information, or a whole dimension of choice. Even conventional STV probably only records a significant fraction of an electoral dimension.

Party list systems may record a significant fraction of the dimension of choice but the recording is confined to the party list makers, and is non-transferable between associations. Party list systems are particular suffrage of parties rather than universal suffrage of all the people.

Whereas simple plurality counts particularise or restrict to a minimal choice.

Re Points about the nxmeek program

Higher orders of FAB STV grow in complexity but it starts without many of the problems catered-for in the notes on Meek method, by

Dr David Hill. I'm not a programmer but mention some points for a FAB STV programmer.

Point 1. FAB STV also does not count equality of preferences.
Point 3 is important and remarked at the start of the actual NZ Meek Code.
Point 5. The FAB STV quota is (my) Harmonic Mean quota (with high proportionality of many seats), but its program should allow for automatic change of quota, like to the Droop quota, as normally used for STV.

FAB STV uses all the preference information. And, unlike the Meek quota, it always treats the quota in terms of the total votes, not merely "active" votes at various stages in the count.

Points 6 to 8 are no problem, as FAB STV does not exclude candidates, nor decide who is elected, during the stages of the count. FAB STV averages a series of counts, not deciding or comparing winners and losers till all the counting has been done.

Point 14. FAB STV gives keep values for all candidates (not just elected candidates, as with Meek method) and thus establishes the order of popularity, If an elected candidate steps down, the runner-up is already known, without needing another election (for a "casual vacancy").
This is a considerable point in favor of FAB STV, which I think I forgot to stress (as it is not among the most important reasons) in my book, "FAB STV: Four Averages Binomial Single Transferable Vote."

Re the file, Notes to the NZ Meek Code

The Notes, of David Hill, refer 37 points, where relevant to his file of the NZ Meek Code.
The editor, who is not a programmer, would like to advise that much

of the concern, of Meek method, is to keep track of the winners and losers, and declare them, during the stages of the count. But FAB STV is free of all those complications, suspending judgment till the end result of systematic recounts, based on its extended use of the Meek method "keep values". As David Hill point 23 says, the keep values are the heart of the Meek method, which is the main sequence of coding that might help a transformation from Meek method to a FAB STV program.

Points about the nzmeek program.

Table of contents

1. The nzmeek program is taken from my [David Hill] election program using only the Meek method rules, and taking out all instructions that allow for extensions not included in the New Zealand rules.

In particular, those extensions allow for users who wish to allow voters to express equality of preference rather than only a strictly ordered list, and for general constraints in which a maximum or minimum is specified for certain categories of candidate.

2. The Local Electoral Act mentions <--4Computer Journal <--5 Algorithm 123 (with minor modifications). The modifications to be found in my algorithm are listed below. It should be noted that, for any real election, the candidates who would be elected would be unchanged, except possibly by item 4 below, where the resolving of a tie might turn out differently.

3. Variables declared as of "real" type are replaced by a pair of variables of "integer" type, to hold the integral part and the fractional part of each such number respectively. This enables the arithmetic method to be defined so precisely that every computer must give identical results, whereas the precise nature of "real" type is not specified.

4. Algorithm 123 has too much output (probably necessary in 1987 when the method could be regarded as more experimental than it is now), and does not give output in some forms that might be wanted.

The volume of output is therefore reduced and further forms of output catered for.

5. Algorithm 123 defines the quota to be (active votes) divided by (seats + 1) precisely. Special steps then have to be taken to disallow the election of one too many candidate in exceptional circumstances. Adding one billionth of a vote to the above formula is required by the [New Zealand] Act, will never cause any difficulty and makes the exceptional case impossible (provided that there are no guarded candidates).

6. If a candidate must be excluded and two or more are currently equal with lowest votes, Algorithm 123 chooses at random from them. This version, as the Act requires, looks, for each pair, at who was ahead the first time they were different, using a random choice only if that does not settle it. I have found it convenient to give each candidate a random number to start with in case it is wanted later, rather than do so only when the need arises.

7. Algorithm 123 iterates to a conclusion at each stage before deciding whether any candidate is now to be elected or excluded. It is now known that this is unnecessary. The same result can be reached more simply by electing anyone as soon as they pass the quota, and by excluding anyone who is further adrift than the total surplus currently available, so these changes are made.

8. In deciding when convergence has been reached, Algorithm 123 looks separately for each elected candidate's surplus to be small enough. Looking instead at the total surplus is simpler and required by the Act.

9. Very minor changes, such as in terminology, or in the number of characters to be allowed in an election title, do not need to be specified here.

10. The program, as enclosed, is designed to take data in the same form as specified in Algorithm 123, in a file called VOTES.DAT.

For example

5 2 [means] 5 candidates for 2 seats

-3 -4 [means] candidates 3 and 4 have withdrawn

11 5 4 2 0 [means] 11 votes have candidate 5 as first preference, 4 as second preference, 2 as third, and no more.

4 1 2 3 0

51 1 5 4 0

1 2 4 5 0 etc

56 3 4 2 5 0

120 5 3 0

0

"Adam" Names of candidates

"Boaz"

"Cain"

"Daniel"

"Eve"

"Example" Title of election

This is free-format so that

```
5 2 -3 -4 11 5 4 2 0 4 1 2 3 0 51 1 5

4 0 1 2 4 5 0 56 3 4 2 5 0 120 5 3 0 0

"Adam""Boaz""Cain""Daniel""Eve""Example"
```

has exactly the same meaning, while being less human-friendly.

11. It will also accept a shorter coded version of the data file as used internally. Details can be supplied if wanted.

12. It produces output on files VOTES.RES and VOTES.OEI which give the information in normal form or extended form, where the latter is too long-winded for general use but includes every detail that might be wanted in special circumstances.

13. In addition it gives file VOTES.RLT. This is not intended for reading by people but as the necessary information to allow a computer to read it to produce other forms of output.

14. If there are guarded candidates, for a re-run of the votes for a casual vacancy where those previously elected and willing to continue must not be excluded, a separate file is required called VOTES.CON. The format would be such as

```
10 5

4

7

0
```

where the first line must indicate the same numbers of candidates and seats as the VOTES.DAT file. Here candidates number 4 and number 7 are to be guarded and the 0 line terminates it.

Notes to the NZ Meek Code.

Table of contents

Note 1: string is not standard Pascal but a compiler-supplied extension. The argument in its declaration is a maximum number of characters it may contain, but it can be shorter. The length function can determine its actual number of characters whenever needed.

Note 2: byte is not standard Pascal but a compiler-supplied subtype of integer, restricted to range 0..255.

Note 3: integer2 is not standard Pascal but a compiler-supplied subtype of integer, restricted to range -32768..32767.

Note 4: integer1 is not standard Pascal but a compiler-supplied subtype of integer, restricted to range -128..127.

Note 5: word is not standard Pascal but a compiler-supplied subtype of integer, restricted to range 0..65535.

Note 6: putting characters into the screen array does not show them on the screen, which has to be done separately. The purpose of the array is to act as a memory of what is on the screen so that, if it has to be temporarily overwritten, it can be restored later.

Note 7: the paintscreen routine is designed to give my particular form of output. You may well want to do something quite different, so I shall not give more detail of it than is there already unless you ask for it.

Note 8: gotoxy is a compiler-supplied routine that moves to the position on the screen given by its arguments, in characters from the left and in rows from the top.

Note 9: putchattr is a compiler-supplied routine that puts a character on the screen, in the background colour and character colour indicated in its second and third arguments, repeating it the number of times shown by the fourth argument. It does not move the current position on the screen.

Note 10: the write and writeln procedures mentioned each print out one or more characters, followed by going onto the next line in the case of writeln. There are similar procedures called read and readln for input. If the first argument is a channel name the output is on that channel, otherwise it is on the screen. This ambiguity of meaning of the first argument, I regard as a serious fault in the design of Pascal.

Note 11: getkeyboard is a compiler-supplied routine to find the next character typed. More details can be supplied if needed.

Note 12: cursoron and cursoroff are compiler-supplied routines whose meanings should be obvious.

Note 13: wherex and wherey are compiler-supplied routines that return the current column and row on the screen.

Note 14: time is a compiler-supplied routine whose four arguments return the current time as hours, minutes, seconds and hundredths of a second.

Note 15: sound is a compiler-supplied routine to sound a note of the given frequency, continuing until soundoff is called.

Note 16: close is a compiler-supplied routine to close a file. The closure is permanent if the second argument is true, but only temporary if false.

Note 17: this routine is my way of implementing the rule for "ahead at first difference". There are other, quite different, ways of doing it and this method should not be regarded as the one-and-only way.

Note 18: it may be unnecessary to have a separate routine for this but the original version also deals with special cases where equality of preference is allowed and then it is more worth while having a separate routine to do it.

Note 19: storing vote values in the scale of 50 means that only three characters are necessary for values up to 124999.

Note 20: the divide routine is pretty direct but trying other approaches to see if they speed it up, they make little difference, so I prefer to keep it simple.

Note 21: working in scale of 10 is not really the most efficient for a computer, but it helps the human to see what it is doing, and the difference in efficiency is slight.

Note 22: outtwice does the same output to both the normal and the extended output channels.

Note 23: countvotes is invoked at each stage to find how many votes each candidate now has using the latest keep values, so it is the heart of the method.

Note 24: after each count the order of the tiebreak numbers is reversed, so that if A is ahead of B if a tie occurs on an even stage, then B is ahead of A if it occurs on an odd stage. Even if someone could see how to manipulate things to make the pseudo-random method favour a particular candidate, which is most unlikely, it would have the opposite effect half the time.

Note 25: this method of implementing the random choice for breaking ties is certainly not the only way it could be done. You may prefer a quite different approach.

Note 26: clreol is a compiler-supplied routine which clears the screen from the current point to the end of the line.

Note 27: initscreen is a compiler-supplied routine that has to be called before screen handling routines (other than simple output) can be used.

Note 28: clrvideo is a compiler-supplied routine to clear the screen.

Note 29: in the complete program coloured can be set true or false by the user. For this demonstration I have fixed it at true, but left in the code for what happens otherwise, for information.

Note 30: in the complete program the user can set randomethod to a choice of three methods and rules may also be given different values. For this version they are fixed, rules=7 indicates New Zealand rules.

Note 31: in the complete program, banner gets set to indicate the user's name and licence number.

Note 32: where things cannot easily be done within Pascal, the compiler gives the option of calling the command routine which accesses MS-DOS instructions direct. The purpose here, and over the next few lines, is to find the size in bytes of the main input file, votes.dat, and take special action if it is too big.

Note 33: here is where that special action is taken. memavail is a compiler-supplied routine to report the memory available. If filesize is close to it okmm is set to false, and this will be used to make ramm an external file instead of a file within "ram". This gives slower access but allows bigger data files to be used.

Note 34: this message varies in the complete program to indicate which STV rules have been used.

Note 35: fstat is a compiler-supplied routine that returns true if the file mentioned exists or false if not.

Note 36: this, and the lines that follow, is one of the trickiest bits. Suppose we have to elect 5, 3 are guarded and 3 others pass the quota in the same count. We then have one who has passed the quota but cannot be elected, but which one? The solution that I have adopted is to reset the quota as a pseudoquota that nobody can reach. Carrying on with the count, but keeping the pseudoquota fixed, will then exclude lowest candidates and redistribute their votes until the right number of candidates remains. This problem is unlikely ever to arise in New Zealand where guarded candidates occur only for those who had enough votes to be elected previously, but a program must allow for the possibility.

Note 37: initscreen was used earlier to prepare the screen for special output. It is used again to restore the screen afterwards.

NZ MEEK CODE

Table of contents

```
program nzmeek(datafile,outt,tmpp,rslt,confile,oei,ramm,rann);
{Count the votes using Meek style STV rules. To make sure
that every computer will give the same results, there are
no "real" variables or operations in this program.
Instead numbers that would be "real" are represented by a
pair of integers, or (if necessarily fractional) by a single
integer. An integer pair represents the part before the
decimal point in the first, and a 9-figure part after the
point in the second. Thus, for example, 346 in the first
and 704000000 in the second means 346.704. An integer
representing a fraction similarly has 9 digits, so that,
for example, 496000000 means 0.496. As an extension, a
fraction can be 1000000000 meaning 1.0. If using Algol 68
it would be worth declaring these as types, but Pascal
does not allow operations on new types to be defined, so
it is not worth it. Wherever names such as max1 and max2

are used, it is implied that they are a pair in the

above sense}

{$i paspc} {Although in the form of a comment, this is an instruction

to include extra compiler-supplied features}

const maxcandidates=50;

namelength=20;

titlength=40;

type candidates=1..maxcandidates;

candrange=0..maxcandidates;

boolvector=array[candidates]of boolean;

nametype=string[namelength];

titname=string[titlength];

var mincand,numcandidates:candidates;

numguarded,numwithdrawn:candrange;

nontrans1,nontrans2,quota1,quota2,total,temp1,temp2,surplus1,surp
lus2,

minvotes1,minvotes2,nextmin1,nextmin2,nn,count,numinv,filesize,

randomcount,aheadcount:integer;
```

```
finish,randomused,aheadused,coloured,incomplete,someone,continu
e,okmm,

constrained,gdlast,pseudoquota,changed:boolean;

guarded,ignore:boolvector;

cand,numseats,numelected,numexcluded,electindex,numzero,

colgrn,colred,colcyn,colyel,colwht,randomethod,fracdigits,rules,

stopp,conmarker,numb,rw:byte;{note 1}

table,iteration,countno:integer2;{note 2}

datafile,outt,tmpp,rslt,confile,oei,ramm,rann:text;

title,str1:titname;

banner:string[78];{note 3}

reply,str2:string[80];

cw:char;

ahead,electarray:array[candidates]of byte;

deemexcl:array[candidates]of integer1;{note 4}

tiebreak:array[candidates]of word;{note 5}

votes1,votes2,keep:array[candidates]of integer;

status:array[candidates]of(hopeful,elected,excluded);

name,shortname:array[candidates]of nametype;
```

```
screentype,nextxleft,nextyleft,nextxright,nextyright,vertdivi,hordivi,

papercolour,counter:byte;

screen:array[1..25,1..80]of char;{note 6}

procedure paintscreen(top:byte);{note 7}

{Print the screen to contain the displayed results as they occur}

var row,col,numdumps,xl,xr,xm,len,paper,ink:byte;

procedure head(xx,len,yy:byte);

var head0:string[8];

col:byte;

begin

if len=7 then head0:='Elected' else head0:='Excluded';

for col:=1 to len do

begin

screen[yy,xx]:=head0[col];xx:=xx+1

end

end{head};

begin{paintscreen}

if top=0 then

{Prepare whole screen}
```

```
begin

clrscr;

for row:=1 to 25 do

for col:=1 to 80 do screen[row,col]:=' ';

len:=length(title);

xl:=40-len div 2;

for col:=1 to len do

begin

screen[1,xl]:=title[col];xl:=xl+1

end;

{screentype=1 if 1 column needed for elected names, 1 for excluded
names

2 if 1 column for elected names, 2 for excluded names

3 if 2 columns for elected names, 1 for excluded names

4 means less than 1 for elected, 2+ for excluded

5 means 2+ for elected, less than 1 for excluded.

vertdivi is the vertical dividing point between elected and excluded.

hordivi is the horizontal dividing point if screentype is 4 or 5}

numdumps:=numcandidates-numwithdrawn-numseats;
```

```
if(numseats<21)and(numdumps<21)then

begin

screentype:=1;

vertdivi:=40;hordivi:=0

end

else if(numseats<21)and(numdumps<41)then

begin

screentype:=2;

vertdivi:=29;hordivi:=0

end

else if(numseats<41)and(numdumps<21)then

begin

screentype:=3;

vertdivi:=53;hordivi:=0

end

else if numseats<21 then

begin

screentype:=4;

vertdivi:=29;hordivi:=62-numdumps
```

```
end

else if numdumps<21 then

begin

screentype:=5;

vertdivi:=53;hordivi:=22-numdumps

end

else

begin

screentype:=6;

vertdivi:=53;hordivi:=numseats-14

end;

if screentype=1 then

begin

xl:=9;xm:=0;xr:=49

end

else

begin

xl:=8;xm:=32;xr:=56

end;
```

```
{Put the "Elected" and "Excluded" headings in place}

head(xl,7,3);

if(screentype=3)or(screentype>4)then head(xm,7,3);

if screentype=5 then head(xr,7,3) else head(xr,8,3);

if(screentype=2)or(screentype=4)then head(xm,8,3);

if screentype=4 then head(xl,8,hordivi+1) else

if screentype=5 then head(xr,8,hordivi+1) else

if screentype=6 then head(xm,8,hordivi+1);

{Monochrome screen uses | to divide left screen from

right screen instead of green and red colours}

if not coloured then

begin

if hordivi=0 then

begin

for row:=3 to 25 do screen[row,vertdivi]:='|'

end

else

begin

if screentype=4 then
```

```
begin

for col:=1 to vertdivi do screen[hordivi,col]:='-';

for row:=3 to hordivi-1 do screen[row,vertdivi]:='|'

end

else if screentype=6 then

begin

for col:=29 to 53 do screen[hordivi,col]:='-';

for row:=3 to hordivi-1 do screen[row,53]:='|';

for row:=hordivi+1 to 25 do screen[row,29]:='|'

end

else

begin

for col:=vertdivi to 80 do screen[hordivi,col]:='-';

for row:=hordivi+1 to 25 do screen[row,vertdivi]:='|'

end

end

end;

{Do title section}

for row:=1 to 2 do
for col:=1 to 80 do

begin

gotoxy(col,row);putchattr(screen[row,col],colcyn,colyel,1)

end

end;

{top=0 means all lines to be done, otherwise only from top
downwards}

for row:=3 to 25 do

if(top=0)or(row>=top)then

for col:=1 to 80 do

begin

if col<=vertdivi then paper:=colgrn else paper:=colred;

if row=3 then ink:=colyel else ink:=colwht;

if hordivi>0 then

begin

if(screentype=4)and(row>=hordivi)then

begin

paper:=colred;

if(col<vertdivi)and(row=hordivi+1)then ink:=colyel
```

```
end
else if screentype=5 then
begin
if row<hordivi then paper:=colgrn;
if(col>vertdivi)and(row=hordivi+1)then ink:=colyel
end
else if screentype=6 then
begin
if(row>hordivi)and(col>29)and(col<54)then paper:=colred;
if(row=hordivi+1)and(col>29)and(col<54)then ink:=colyel
end
end;
gotoxy(col,row);putchattr(screen[row,col],paper,ink,1)
end;
{Set up initial screen coordinates for elected (left) and
excluded (right) candidate names}
if top=0 then
begin
nextyleft:=5;
```

```
if hordivi=0 then nextyright:=5 else nextyright:=hordivi+3;

if screentype=1 then

begin

nextxleft:=9;nextxright:=49

end

else

begin

nextxleft:=8;

if(screentype=2)or(screentype=6)then nextxright:=32 else

if screentype=4 then nextxright:=8 else nextxright:=56

end

end

end{paintscreen};

procedure show(cand:candidates;elected:boolean;var
nextx,nexty:byte);

{Print on screen the name of the candidate just elected

or excluded, in position specified by nextx and nexty which

are updated. The same information is put into the screen

array, so that the screen can be rewritten later if necessary}
```

```
var paper,k:byte;
begin
gotoxy(nextx,nexty);{note 8}
if elected then paper:=colgrn else paper:=colred;
for k:=1 to 20 do
begin
screen[nexty,nextx+k-1]:=name[cand,k];
putchattr(name[cand,k],paper,colwht,1);{note 9}
gotoxy(nextx+k,nexty)
end;
nexty:=nexty+1;
if nexty=25 then
begin
if screentype=1 then nextx:=nextx+40 else nextx:=nextx+24;
nexty:=5
end
end{show};
{For output to screen, the standard write and writeln procedures will
serve, though I do not much like them. For input from keyboard I
must use my own routines instead of read and readln, so as (1) to
catch escape key when necessary and (2) if the user makes an error
to catch it myself and not leave them to the mercy of compiler or
operating-system error messages}{note 10}
function readkey:byte;
{Reads one character from the keyboard and returns an integer
to represent it.
Returns 0..9 for digits 0..9,
12 for enter
15 for delete or back-delete
20 for space
99 for everything else.
If however the character is Esc, a jump is taken out of the program}
var hh:char;
c,h:byte;
begin
getkeyboard(hh,c);h:=ord(hh);{note 11}
if h=0 then
begin
```

```
if c=83 then readkey:=15 else readkey:=99

end

else

case h of

48..57:readkey:=h-48;

13:readkey:=12;

8:readkey:=15;

27:exitprog(1);

32:readkey:=20;

otherwise readkey:=99

end

end{readkey};

function readint(len:byte):integer;

{Reads, from the keyboard, a non-negative integer in decimal

notation of not more than len characters, where 0<len<6,

terminated with enter. If more than len digits are entered,

the last digit is overwritten. The cursor is switched on

during the reading but switched off at exit}

var y,j,k,xx,yy:byte;
```

```
w:integer;

val:array[1..6]of byte;

begin

if len<1 then len:=1 else

if len>5 then len:=5;

cursoron;{note 12}

xx:=wherex;yy:=wherey;{note 13}

for j:=1 to len+1 do val[j]:=10;

for k:=1 to len do write(' ');

gotoxy(xx,yy);j:=1;

repeat

y:=readkey;

if y<10{a digit 0..9} then

begin

val[j]:=y;

if j<len then j:=j+1

end

else if y=15{delete} then

begin
```

```
if(j>1)and((j<len)or(val[len]=10))then j:=j-1;

val[j]:=10

end;

gotoxy(xx,yy);

for k:=1 to len do

if val[k]=10 then write(' ') else write(chr(val[k]+48));

gotoxy(xx+j-1,yy)

until(y=12)and(val[1]<10);

{i.e. until enter with at least one digit before it}

cursoroff;{note 12}

j:=1;w:=0;

while val[j]<10 do

begin

w:=10*w+val[j];j:=j+1

end;

readint:=w

end{readint};

procedure newline(k:byte);

var j:byte;
```

```
begin
for j:=1 to k do writeln
end{newline};
procedure anykey(m:byte);
{Prints the message 'Press any key to continue' on line
m of screen. In actual use m is always 24 or 25 and a
shorter banner is wanted if on line 24 because it is
then within a "window". Waits until a key is pressed
before returning}
var c:char;
y:byte;
begin
gotoxy(26-m,m);putchattr(' ',lightgrey,black,2*m+30);
gotoxy(22,m);write('Press any key to continue');
getkeyboard(c,y);
gotoxy(22,m);write(' ':10,'Wait ...',' ':10)
end{anykey};
function tim:integer;
var m,n,s,t:integer;
```

```
begin
time(m,n,s,t);tim:=((60*m+n)*60+s)*100+t{note 14}
end{tim};
procedure delay(centisecs:word);
{Causes the computer to wait centisecs hundredths of
a second before proceeding. It is a little more
complicated than would otherwise be necessary to
allow for the possibility that midnight might occur
during the wait}
var j,q:integer;
begin
j:=tim;q:=(j+centisecs)mod 8640000;
while(j>q)and(j>8630000)do j:=tim;
while j<q do j:=tim
end{delay};
procedure signal(k:byte);
{If k=0 sounds a low note. If k=1 or k=2 sounds a higher note,
once or twice respectively. If k>2 sounds k times alternating
between two notes a perfect fifth apart. All notes, and gaps
between notes, last for 2/25 of a second}
var j:byte;
begin
if k=0 then
begin
sound(550);delay(8);{note 15}
soundoff
end
else
for j:=1 to k do
begin
if(k=2)or odd(j) then sound(880) else sound(1320);
delay(8);soundoff;delay(8)
end
end{signal};
procedure shutdown(var x:text);
{Closes file x after writing Û as an extra line on the end. This
is necessary if an operating system stores files without their
final CRLF. The Û is never actually used, but the previous line
```

```
with final CRLF is safe}

begin

writeln(x,chr(219));close(x,true){note 16}

end{shutdown};

function codecan(n:byte):char;

{Turns a candidate number into the equivalent character}

begin

if n>77 then codecan:=chr(n+50) else codecan:=chr(n+48)

end{codecan};

function decodecan(m:char):byte;

{Turns a character into the equivalent candidate number}

begin

if m>'~' then decodecan:=ord(m)-50 else decodecan:=ord(m)-48

end{decodecan};

function decodenum(m:char):byte;

{Turns a character into the equivalent number of candidates

or number of seats}

begin

if m>'~' then decodenum:=ord(m)-66 else
if m>'?' then decodenum:=ord(m)-64 else decodenum:=ord(m)+156
end{decodenum};
procedure adjust(var a1,a2:integer);
{a1 and a2 are a pair representing a real. If a2 is out of
range, they are adjusted to bring it back into range. The
result should never be negative but, as a precaution in case
of rounding error, if it is, the value is reset to 0.0}
begin
while a2<0 do
begin
a1:=a1-1;a2:=a2+1000000000;
if a1<0 then
begin
a1:=0;a2:=0
end
end;
while a2>999999999 do
begin
a1:=a1+1;a2:=a2-1000000000
```

```
end
end{adjust};
procedure update;{note 17}
{Update the ahead array to show who was ahead at first difference.
The numbers assigned are 0, 2, 4, etc., the larger being ahead
of the smaller. If equal at all stages so far the average number
for the tied group is used. The steps of 2 make this always an
integer}
var cand:candidates;
max,val,number:byte;
list:array[candidates]of byte;
procedure up(number:byte);
{number gives the number of candidates in a tied ranking,
to be separated if possible.}
var high:integer1;
low,n,k:byte;
max1,max2:integer;
continue:boolean;
arra1,arra2:array[candidates]of integer;
```

```
begin
{list contains the candidate numbers of those concerned.
high and low are found as highest and lowest values in
the group if they can all be separated}
high:=ahead[list[1]]+number-1;low:=high-2*number+2;
for n:=1 to number do
begin
arra1[n]:=votes1[list[n]];arra2[n]:=votes2[list[n]]
end;
continue:=true;
while continue do
begin
{Find highest value of votes among those concerned and
set k to number of candidates with that highest value}
max1:=-1;max2:=0;
for n:=1 to number do
if(arra1[n]>max1)or(arra1[n]=max1)and(arra2[n]>max2)then
begin
k:=1;
```

```
max1:=arra1[n];max2:=arra2[n]

end

else if(arra1[n]=max1)and(arra2[n]=max2)then k:=k+1;

{Set ahead to new sole value or average value and reset

arra1 and arra2 arrays so that these cannot be highest again}

for n:=1 to number do

if(arra1[n]=max1)and(arra2[n]=max2)then

begin

ahead[list[n]]:=high-k+1;

arra1[n]:=-2;arra2[n]:=0

end;

high:=high-2*k;

continue:=high>=low

end

end{up};

begin{update}

incomplete:=false;

max:=2*numcandidates-2;

val:=0;
```

```
while val<=max do
begin
number:=0;
{Find number of candidates whose present ahead is val}
for cand:=1 to numcandidates do
if ahead[cand]=val then
begin
number:=number+1;
list[number]:=cand
end;
{If there is more than 1 of them, see if they now differ
in number of votes to allow separation. If no number
exceeds 1, incomplete will be false on exit and update
will not be entered again}
if number>1 then
begin
up(number);
incomplete:=true;
val:=val+number
```

```
end;

val:=val+1

end

end{update};

function incand:byte;{note 18}

{Reads a candidate number from the coded data and

converts it to the equivalent integer}

var ch:char;

begin

repeat

read(ramm,ch)

until(ch>='0');

incand:=decodecan(ch)

end{incand};

function innum:byte;

{Reads a number of candidates or seats from the

coded data and converts it to the equivalent integer}

var ch:char;

begin
```

```
repeat

read(datafile,ch)

until ch>' ';

innum:=decodenum(ch)

end{innum};

function inval:integer;{note 19}

{Reads a vote value from the coded data and

converts it to the equivalent integer}

var i:integer;

ch:char;

begin

repeat

read(ramm,ch)

until ch>='$';

{A vote value is three characters unless the first two

would be 00 when $ replaces them}

if ch='$' then

begin

repeat
```

```
read(ramm,ch)

until ch>='0';

i:=ord(ch)-48

end

else

begin

{A three character value meaning three digits in

the scale of 50}

if ch='0' then i:=0 else i:=2500*(ord(ch)-48);

repeat

read(ramm,ch)

until ch>='0';

i:=i+50*(ord(ch)-48);

repeat

read(ramm,ch)

until ch>='0';

i:=i+ord(ch)-48

end;

inval:=i
```

```
end{inval};

procedure multiply(var c1,c2:integer;a1,a2,b2:integer);

{Multiply the pair a1, a2 by the fraction b2, putting

the result into the pair c1,c2. The operation is split

into parts to ensure no overflow. The result is rounded

up if not exact}

var c,d,e,a11,a12,a13,a21,a22,a23,b21,b22,b23:integer;

begin

if(a1=1)and(a2=0)then

begin

c1:=0;c2:=b2

end

else if b2=1000000000 then

begin

c1:=a1;c2:=a2

end

else if(a1=0)and(a2=0)or(b2=0)then

begin

c1:=0;c2:=0
```

```
end

else

begin

{Split each of a1, a2 and b2 into three parts}

a11:=a1 div 1000;a13:=a1 mod 1000;

a12:=a11 mod 1000;a11:=a11 div 1000;

a21:=a2 div 1000;a23:=a2 mod 1000;

a22:=a21 mod 1000;a21:=a21 div 1000;

b21:=b2 div 1000;b23:=b2 mod 1000;

b22:=b21 mod 1000;b21:=b21 div 1000;

{Multiply the appropriate parts but avoid multiplying by

parts that are usually zero (if they are)}

if a11=0 then

begin

if a12=0 then d:=1000*a13*b21+a21*b21+a13*b22

else d:=1000*(a13*b21+a12*b22)+a21*b21+a13*b22+a12*b23

end

else d:=1000*
(a13*b21+a12*b22+a11*b23)+a21*b21+a13*b22+a12*b23;
```

```
if d>999999 then

begin

c:=d div 1000000;d:=d mod 1000000

end

else c:=0;

e:=a23*b23;

if e mod 1000>0 then e:=e div 1000+1 else e:=e div 1000;

e:=e+a23*b22+a22*b23

+1000*((a23*b21)mod 1000+(a22*b22)mod 1000+(a21*b23)mod
1000);

if e mod 1000000>0 then e:=e div 1000000+1 else e:=e div
1000000;

d:=1000*d+a22*b21+a21*b22+e+a13*b23

+(a23*b21)div 1000+(a22*b22)div 1000+(a21*b23)div 1000;

if d>999999999 then c:=c+1;

c2:=d mod 1000000000;

if a11=0 then

begin

if a12=0 then c1:=c else c1:=c+a12*b21

end
```

```
else c1:=c+1000*a11*b21+a12*b21+a11*b22

end

end{multiply};

procedure divide(var c2:integer;a1,a2,b1,b2:integer);{note 20}

{Divide the pair a1, a2 by the pair b1, b2 putting the

result into the fraction c2. It is known that the

result must always be less than 1.0. a and b are

changed within the procedure but, having been called

by value, this does no harm. The result is rounded

up if not exact}

var n:integer;m:byte;

procedure muldiv;{note 21}

{Multiply a by 10 and divide n by 10}

begin

n:=n div 10;

if n>0 then

begin

a1:=10*a1+a2 div 100000000;a2:=10*(a2 mod 100000000)

end
```

```
end{muldiv};

begin{divide}

c2:=0;

if(a1>0)or(a2>0)then

begin

n:=1000000000;

{Standardise the numbers with n keeping the score}

while(b1>a1)or(b1=a1)and(b2>a2)do muldiv;

while n>0 do

begin

m:=0;

{See how many times b can be subtracted from a. Answer in m}

while(b1<a1)or(b1=a1)and(b2<=a2)do

begin

a1:=a1-b1;a2:=a2-b2;

adjust(a1,a2);m:=m+1

end;

c2:=c2+m*n;

{Move along one place and try again for next digit}
```

```
muldiv

end;

{round up if not exact}

if(a1>0)or(a2>0)then c2:=c2+1

end

end{divide};

procedure findquota;

{Calculate quota as active votes over seats + 1}

var x1,x2:integer;

nn:byte;

begin

if not pseudoquota then

begin

x1:=total-nontrans1;x2:=-nontrans2;

adjust(x1,x2);nn:=numseats+1;

quota1:=x1 div nn;

x1:=100000*(x1 mod nn)+x2 div 10000;

{+1 at the end increases the result by one billionth of a

vote above the exact quota. This saves trouble later and
(unlike Droop's original +1 whole vote) it is very hard

to believe that it can possibly do any harm}

quota2:=10000*(x1 div nn)+(10000*(x1 mod nn)+x2 mod 10000)div
nn+1;

adjust(quota1,quota2);

if(rules=7{nz})and(numseats=1)then

begin

if quota2>1 then quota1:=quota1+1;

quota2:=0

end

end

end{findquota};

procedure outtwice(line:boolean;ss:string;guarded:boolean);{note
22}

begin

if line then writeln(outt);

if line then writeln(oei);

writeln(outt,ss);writeln(oei,ss);

gdlast:=guarded

end{outtwice};
```

```
procedure printout(index:byte);

{Print on the output channel for serial output the current

situation. This is used only when there is an event to

report - not at every iteration. index gives the nature

of the event requiring printout. See the messages printed

at the end of this procedure for details}

var arg1,arg2:integer;

j:byte;

cand:candidates;

procedure wrt(var
channel:text;part1,part2:integer;dig1,dig2:integer1);

{Print on channel file the number represented by part as

part1 before the point, part2 after the point, in total

width of dig1 (or more if needed) with dig2 figures

after the point. If a fractional value is to be printed

it is held in part1 with part2 set negative, and is then

printed as 100 times its true value (followed by % if part2 = -1)}

var j:byte;

k,pc:integer;
```

```
begin
dig1:=dig1-dig2-1;
if dig1<1 then dig1:=1;
pc:=part2;
if pc<0 then
begin
part2:=100*(part1 mod 10000000);part1:=part1 div 10000000
end;
k:=50000000;
for j:=1 to dig2-1 do k:=k div 10;
part2:=part2+k;adjust(part1,part2);
write(channel,part1:dig1,'.');
for j:=1 to dig2 do
begin
write(channel,part2 div 100000000:1);
part2:=10*(part2 mod 100000000)
end;
if pc=-1 then write(channel,'%')
end{wrt};
```

```
begin{printout}

{if no count has been made since printout last used, then

there is no point in repeating it}

if iteration=countno then writeln(outt) else

begin

countno:=iteration;

table:=table+1;writeln(outt);

{If index=5 or index=8 all seats are filled and printing of details

beyond that message is not required}

if(index<>5)and(index<>8)then

begin

writeln(outt);writeln(outt);

writeln(outt,' ':20,title);

writeln(outt);

write(outt,' Table: ',table:1,' Count: ',iteration:1);

{Calculate j as the number of spaces required before

printing quota to align it with votes}

if not pseudoquota then

begin
```

```
if quota1<10 then j:=1 else if quota1<100 then j:=2 else

if quota1<1000 then j:=3 else if quota1<10000 then j:=4 else j:=5;

j:=j+fracdigits;

if table<10 then j:=17-j else if table<100 then j:=16-j else j:=15-j;

if iteration>99 then j:=j-2 else if iteration>9 then j:=j-1;

if(rules=7{nz})and(numseats=1) then

write(outt,' ':j,'Absolute majority: ')

else write(outt,' ':j,'Quota: ');

wrt(outt,quota1,quota2,1,fracdigits)

end;

writeln(outt);writeln(outt);

write(outt,'Candidate',' ':12,'Keep Transfer Votes');

writeln(outt);writeln(outt);

j:=1;

for cand:=1 to numcandidates do

if not ignore[cand] then

begin

write(outt,name[cand]);

wrt(outt,keep[cand],-1,6,1);wrt(outt,1000000000-keep[cand],-1,8,1);
```

```
{Allow for possibility of slight rounding error by

setting arg to quota instead of to votes if

necessary and justified}

if not pseudoquota and(status[cand]=elected)and((votes1[cand]
<quota1)

or(votes1[cand]=quota1)and(votes2[cand]<quota2))then

begin

arg1:=quota1;arg2:=quota2

end

else

begin

arg1:=votes1[cand];arg2:=votes2[cand]

end;

wrt(outt,arg1,arg2,10,fracdigits);

if status[cand]=elected then write(outt,' >>> Elected')

else if status[cand]=excluded then

begin

write(outt,' Excluded');ignore[cand]:=true

end;
```

```
writeln(outt);

{Put in blank line every 5 to improve readability}

if(numcandidates>9)and(j mod 5=0)and

(cand<>numcandidates)then writeln(outt);

j:=j+1

end;

writeln(outt);

write(outt,'Non-transferable');

wrt(outt,nontrans1,nontrans2,30,fracdigits);writeln(outt);

writeln(outt);write(outt,'Total');

wrt(outt,total,0,41,fracdigits);writeln(outt);

writeln(outt);writeln(outt);

{Also write information for result sheet output}

write(tmpp,iteration:1,' ');

if pseudoquota then wrt(tmpp,0,0,1,9)else
wrt(tmpp,quota1,quota2,1,9);

writeln(tmpp);

for cand:=1 to numcandidates do

begin
```

```
wrt(tmpp,votes1[cand],votes2[cand],1,9);write(tmpp,' ');

wrt(tmpp,keep[cand],-2,1,7);writeln(tmpp)

end;

wrt(tmpp,nontrans1,nontrans2,1,9);writeln(tmpp)

end

end;

if numseats>1 then

begin

writeln(oei);

if index=2 then

begin

writeln

(oei,'As lowest difference exceeds total surplus, lowest candidate');

write(outt,'Lowest candidate ');

outtwice(false,'cannot overtake, so can safely be excluded.',false)

end

else if index=3 then

begin

write(outt,'No surplus remains');
write(oei,'Remaining surplus < 0.0001');

outtwice(false,

', so lowest candidate must be excluded.',false)

end

else if index=4 then

begin

outtwice(false,

'All remaining candidates can be elected, for',false);

outtwice(false,

'their number is no greater than the seats available.',false)

end

else if index=5 then outtwice(false,

'because all seats are now filled.',false)

else if index=6 then outtwice(false,'Exclude all with zero votes.',false)

else if index=7 then

begin

outtwice(false,'Exclude those with zero votes, but retaining',false);

outtwice(false,'enough candidates to fill all seats.',false)

end
```

```
else if index=8 then

outtwice(false,'Elect all guarded candidates and exclude
others.',false)

else if index=9 then

begin

write(outt,'No surplus remains');

write(oei,'Remaining surplus negligible');

outtwice(false,

', so lowest candidate must be excluded.',false)

end

end

end{printout};

procedure exclude(cand:candidates;runnerup:boolean);

{Exclude cand}

begin

{Announce exclusion on screen}

show(cand,false,nextxright,nextyright);

{And on output files for serial output}

str2:=concat('>>>>> ',shortname[cand],' excluded');
```

```
if aheadused then
str2:=concat(str2,' (using "ahead at first difference" rule)')
else if randomused then str2:=concat(str2,' (using random choice)');
if length(str2)<72 then
str2:=concat(str2,' <<<<<');
outtwice(true,str2,false);
{And for result sheet output}
writeln(tmpp,'-2 ',cand:1);
deemexcl[cand]:=-table;status[cand]:=excluded;
if not runnerup then keep[cand]:=0;
numexcluded:=numexcluded+1;
if randomused then
begin
writeln(tmpp,'''2',iteration:1);
randomused:=false
end;
if aheadused then
begin
writeln(tmpp,'''7',iteration:1);
```

```
aheadused:=false

end

end{exclude};

procedure elect(cand:candidates);

{Set cand as elected and store the fact in electarray}

begin

if numelected<numseats then

begin

status[cand]:=elected;numelected:=numelected+1;

if guarded[cand] then

begin

guarded[cand]:=false;numguarded:=numguarded-1

end;

electindex:=electindex+1;electarray[electindex]:=cand

end

end{elect};

procedure showelect;

{Take candidates in electarray and announce their election

in order of their numbers of votes}
```

```
var count,candref,maxcandref:byte;

begin

{Find candidate with most votes}

for count:=1 to electindex do

begin

maxcandref:=1;

while electarray[maxcandref]=0 do maxcandref:=maxcandref+1;

for candref:=maxcandref+1 to electindex do

begin

if electarray[candref]>0 then

if(votes1[electarray[candref]]>votes1[electarray[maxcandref]])or

(votes1[electarray[candref]]=votes1[electarray[maxcandref]])and

(votes2[electarray[candref]]>votes2[electarray[maxcandref]])

then maxcandref:=candref

end;

{Announce on screen}

show(electarray[maxcandref],true,nextxleft,nextyleft);

{And on output files for serial output}

outtwice(true,concat('>>>>> ',
shortname[electarray[maxcandref]],' elected <<<<<'),false);

{And for result sheet output}

deemexcl[electarray[maxcandref]]:=table;

writeln(tmpp,'-1 ',electarray[maxcandref]:1);

{Stop the same one from being found again}

electarray[maxcandref]:=0

end;

{Restart the electarray table}

electindex:=0

end{showelect};

function ask(eg:candidates):candidates;

{If a random choice is necessary for whom to exclude and

randomethod is interactive, ask the user to choose.

eg is an example candidate of those involved}

var count:byte;

cand,row,top:integer;

ok:boolean;

poss:boolvector;

begin
```

```
count:=0;

for cand:=1 to numcandidates do

begin

{poss array is set to say whether cand is a possible one

to be asked about. count is found as the number of possibles}

poss[cand]:=(status[cand]<>excluded)and not guarded[cand]

and(votes1[cand]=votes1[eg])and(votes2[cand]=votes2[eg]);

if poss[cand] then count:=count+1

end;

{If there is only 1 possible, there is no need to ask after all}

if count<2 then ask:=eg else

begin

{The asking will overwrite part of the result screen.

Calculate where the top of the window must be}

top:=22-(count+2)div 3;

ink(black);

if coloured then

begin

paper(cyan);papercolour:=cyan
```

```
end

else

begin

paper(lightgrey);papercolour:=lightgrey

end;

{Form the window, but not using compiler-supplied window method}

for row:=top to 25 do

begin

gotoxy(1,row);putchattr(' ',papercolour,black,80)

end;

gotoxy(3,top);

writeln('A random choice is necessary for whom to exclude.');

write(' The relevant candidates are:');

count:=0;writeln;

for cand:=1 to numcandidates do

if poss[cand] then

begin

{Print candidate numbers as well as names as user selection

is made by number. Print 3 candidates per line}
```

```
write(' ',cand:2,' ',name[cand]);

count:=count+1;

if count mod 3=0 then writeln

end;

ok:=true;

repeat

{Signal to user that a decision is wanted}

if ok then signal(3) else signal(6);

gotoxy(5,24);

write('Please choose one at random and give the candidate number:
');

cand:=readint(2);

ok:=cand<=numcandidates;

if ok then ok:=poss[cand]

{Refuse impossible candidates}

until ok;

{Restore screen}

paintscreen(top);

ask:=cand
```

```
end

end{ask};

procedure adjustdata;

var j,nm,num:byte;

ch:char;

str1:string[10];

ended:boolean;

begin

num:=numcandidates-numexcluded;

reset(ramm);rewrite(rann);

read(ramm,ch);write(rann,ch);

ended:=false;

while not ended do

begin

{copy value of vote from ramm to rann}

if ch<>'$' then

begin

read(ramm,ch);write(rann,ch)

end
```

```
else if ramm^='0' then ended:=true;

if not ended then

begin

read(ramm,ch);write(rann,ch);

{copy vote with relevant adjustments}

nm:=0;read(ramm,ch);

while ch<>'0' do

begin

if status[decodecan(ch)]<>excluded then

begin

nm:=nm+1;

{If nm=num it means that all remaining candidates are

mentioned so last one should not be output}

if nm<num then write(rann,ch)

end;

read(ramm,ch)

end;

read(ramm,ch);write(rann,'0',ch)

end
```

```
else write(rann,'0')

end;

if not okmm then

begin

close(rann,true);assign(rann,'votes.rnn')

end;

reset(rann);

if not okmm then

begin

close(ramm,true);assign(ramm,'votes.rmm')

end;

rewrite(ramm);

{Modified data in temporary file rann. Copy it over to ramm which

is the expected name outside this routine}

while not eof(rann) do

begin

read(rann,ch);write(ramm,ch)

end

end{adjustdata};
```

```
procedure countvotes;{note 23}

var cand,mm,marker:byte;

ended:boolean;

count,value1,value2,tot1,tot2:integer;

begin

iteration:=iteration+1;

reset(ramm);

nontrans1:=0;nontrans2:=0;

for cand:=1 to numcandidates do

begin

votes1[cand]:=0;votes2[cand]:=0

end;

count:=inval;

while count>0 do

begin

value1:=1;value2:=0;

cand:=incand;ended:=false;

while cand>0 do

begin
```

```
if not ended then
begin
if keep[cand]>0 then
begin
ended:=status[cand]=hopeful;
if ended then
begin
votes1[cand]:=votes1[cand]+count*value1;
multiply(temp1,temp2,count,0,value2);
votes1[cand]:=votes1[cand]+temp1;
votes2[cand]:=votes2[cand]+temp2;
adjust(votes1[cand],votes2[cand]);
value1:=0;value2:=0
end
else
begin
multiply(temp1,temp2,value1,value2,keep[cand]);
value1:=value1-temp1;value2:=value2-temp2;
adjust(value1,value2);
```

```
votes1[cand]:=votes1[cand]+count*temp1;

multiply(temp1,temp2,count,0,temp2);

votes1[cand]:=votes1[cand]+temp1;

votes2[cand]:=votes2[cand]+temp2;

adjust(votes1[cand],votes2[cand])

end

end;

if(value1=0)and(value2=0)then ended:=true

end;

cand:=incand

end;

nontrans1:=nontrans1+count*value1;

multiply(temp1,temp2,count,0,value2);

nontrans1:=nontrans1+temp1;nontrans2:=nontrans2+temp2;

adjust(nontrans1,nontrans2);

count:=inval

end;

for cand:=1 to numcandidates do tiebreak[cand]:=10000-
tiebreak[cand] {note 24}
```

```
end{countvotes};

function lowestcand:candidates;

{Find the continuing (non-guarded) candidate with fewest votes}

var cand,candy:candidates;

begin

candy:=1;

while(status[candy]<>hopeful)or guarded[candy] do
candy:=candy+1;

for cand:=candy+1 to numcandidates do

if(status[cand]=hopeful)and not guarded[cand] then

begin

if(votes1[cand]<votes1[candy])or(votes1[cand]=votes1[candy])and

(votes2[cand]<votes2[candy])then

begin

{New lowest so far}

candy:=cand;randomused:=false;

aheadused:=false

end

else
if(votes1[cand]=votes1[candy])and(votes2[cand]=votes2[candy])then
```

```
begin
if ahead[cand]<>ahead[candy] then
begin
{Equal lowest on votes but lowest on ahead criterion}
if ahead[cand]<ahead[candy] then candy:=cand;
randomused:=false;aheadused:=true
end
else
begin
{Equal lowest on votes and on ahead. Break tie
with random choice}
if tiebreak[cand]<tiebreak[candy] then candy:=cand;
randomused:=true;aheadused:=false
end
end
end;
if aheadused then aheadcount:=aheadcount+1;
{If a random choice was in the end necessary then, if
randomethod is interactive, ask the user to make the
```

```
necessary selection with candy as an example of
those relevant}
if randomused then
begin
randomcount:=randomcount+1;
if randomethod=3 then candy:=ask(candy)
end;
lowestcand:=candy
end{lowestcand};
procedure settiebreaks;{note 25}
{Give each candidate a four digit random number in case it
is needed as a tie breaker. Ensure that no two candidates
get the same number}
var seed1,seed2,seed3:integer2;
cand,candy:byte;
ok:boolean;
function random:word;
{Returns a pseudo-random four-digit integer, rectangularly
distributed between 0001 and 9999. Based on Wichmann and
```

```
Hill, Algorithm AS 183, Appl. Statist. (1982) 31, 188-190.

At first entry sets seeds}

var h,m,s,u:integer;

begin

if seed1=0 then

begin

{seeds need setting. Use clock reading unless user has

asked for fixed method}

if randomethod=1 then

begin

h:=5;m:=0;s:=0;u:=0

end

else time(h,m,s,u);{note 14}

seed1:=numcandidates+h+s;

seed2:=numseats+m+u;

seed3:=(total+10000*(total mod 10)) mod 30323

end

else h:=1;

for m:=1 to h do
```

```
begin
seed1:=(171*seed1)mod 30269;
seed2:=(172*seed2)mod 30307;
seed3:=(170*seed3)mod 30323
end;
random:=((seed1*10000)div 30269+(seed2*10000)div 30307+
(seed3*10000)div 30323)mod 10000
end{random};
begin{settiebreaks}
seed1:=0;
for cand:=1 to numcandidates do
repeat
tiebreak[cand]:=random;ok:=true;
for candy:=1 to cand-1 do
if tiebreak[cand]=tiebreak[candy] then ok:=false
{Keep trying until number is unique}
until ok
end{settiebreaks};
procedure readname(var tag:string;maxlen:byte);
```

```
{Read candidate's name or title from data file and return it in tag.
If a candidate's name also set shortname with no trailing spaces}
var sub:byte;
ch:char;
begin
tag:='';
{Find opening "}
repeat
read(datafile,ch)
until ch='"';
sub:=0;read(datafile,ch);
{Read characters until closing " and add them to tag up
to a maximum length of maxlen}
while ch<>'"'do
begin
if sub<maxlen then
begin
sub:=sub+1;tag:=concat(tag,ch)
end;
```

```
read(datafile,ch)

end;

if maxlen<titlength then

begin

{This is a candidate's name, not the title of election}

shortname[cand]:=name[cand];

{Make up to maxlen characters with trailing spaces}

while sub<maxlen do

begin

sub:=sub+1;tag:=concat(tag,' ')

end

end

end{readname};

procedure pwlz(k:integer);

{pwlz means Print With Leading Zeroes, on oei output channel}

var n:integer;

m:byte;

begin

write(oei,'.');n:=100000000;
```

```
repeat

m:=k div n;write(oei,m:1);

k:=k mod n;n:=n div 10

until n=0

end{pwlz};

procedure shownumbers;

{Shows numbers at bottom of screen to let user know that something

is happening. Puts information to oei output channel too}

var wx,wy:byte;

begin

wx:=wherex;wy:=wherey;{note 13}

if coloured then

begin

paper(cyan);ink(black)

end

else

begin

paper(black);ink(lightgrey)

end;
```

```
gotoxy(1,25);clreol;{note 26}

gotoxy(5,25);write(surplus1+surplus2/1000000000.0:16:6);

gotoxy(35,25);write(temp1+temp2/1000000000.0:16:6);

gotoxy(wx,wy);

if numseats>1 then

begin

writeln(oei);

write(oei,'Total surplus = ',surplus1:1);pwlz(surplus2);

write(oei,' ':5,'Lowest difference = ',temp1:1);pwlz(temp2);

writeln(oei)

end

end{shownumbers};

function reed:integer;

{Finds next number on datafile}

var r:word;

c:char;

begin

repeat

read(datafile,c)
```

```
until(c>='0')and(c<='9');

r:=ord(c)-48;

while(datafile^>='0')and(datafile^<='9')do

begin

read(datafile,c);r:=10*r+ord(c)-48

end;

reed:=r

end{reed};

procedure oeioutput;

var arg1,arg2:integer;

cand:candidates;

begin

if iteration>1 then writeln(oei);

writeln(oei);

writeln(oei,' ':20,title);

writeln(oei);write(oei,'Iteration: ',iteration:1,' ');

if not pseudoquota then

begin

if(rules=7{nz})and(numseats=1) then write(oei,'Absolute majority')
```

```
else write(oei,'Quota');

write(oei,': ',quota1:1);pwlz(quota2)

end;

writeln(oei);writeln(oei);

write(oei,'Candidate',' ':12,' Keep',' ':11,' Votes',' ':7,'Ahead');

if randomethod<3 then write(oei,' Random');

writeln(oei);writeln(oei);

for cand:=1 to numcandidates do

begin

write(oei,name[cand]);

if keep[cand]=1000000000 then write(oei,' 1.000000000') else

begin

write(oei,' 0');pwlz(keep[cand])

end;

write(oei,votes1[cand]:8);pwlz(votes2[cand]);

write(oei,ahead[cand]:7);

if randomethod<3 then write(oei,tiebreak[cand]:8);

writeln(oei);

if(numcandidates>9)and(cand mod 5=0)and
```

```
(cand<>numcandidates)then writeln(oei)

end;

writeln(oei);

write(oei,'Non-transferable',nontrans1:25);pwlz(nontrans2);

writeln(oei);writeln(oei);

writeln(oei,'Total',total:36,'.000000000')

end{oeioutput};

procedure constraintquery;

{Asks whether any constraints exist. The procedure is

entered anyway but, if not applying, only to set the

necessary markers to show no constraints}

var candy:integer1;

answered:boolean;

procedure message;

{Message and finish if pointless to continue}

begin

clrscr;newline(3);

if numguarded=numseats then

begin
```

```
writeln(' No election necessary. All guarded candidates');

writeln(' are elected. All others are excluded.');

outtwice(true,' No election necessary. All guarded candidates',false);

outtwice(true,' are elected. All others are excluded.',false)

end

else

begin

writeln(' Impossible - number of guarded candidates');

writeln(' exceeds number of seats.');

outtwice(true,' Impossible - number of guarded candidates',false);

outtwice(true,' exceeds number of seats.',false)

end;

shutdown(outt);shutdown(oei);shutdown(rslt);

exitprog(2)

end{message};

begin{constraintquery}

constrained:=false;

{A guarded candidate is constrained to be elected eventually}

numguarded:=0;
```

```
for candy:=1 to numcandidates do guarded[candy]:=false;

if fstat('votes.con')then

begin

assign(confile,'votes.con');reset(confile);

read(confile,candy);read(confile,candy);

read(confile,candy);

while candy>0 do

begin

guarded[candy]:=true;numguarded:=numguarded+1;

writeln(tmpp,'''4',name[candy],'''');

outtwice(not gdlast,concat(shortname[candy],' is guarded.'),true);

read(confile,candy)

end;

{message prints message and terminates program}

if numguarded>=numseats then message

end

end{constraintquery};

begin{main program}

numinv:=0;randomcount:=0;
```

```
aheadcount:=0;

pseudoquota:=false;gdlast:=false;

conmarker:=0;

initscreen;cursoroff;{note 27}

clrvideo;{note 28}

coloured:=true;{note 29}

if coloured then

begin

paper(blue);ink(yellow);

for rw:=1 to 25 do

begin

gotoxy(1,rw);putchattr(' ',blue,yellow,80)

end

end;

gotoxy(1,3);

write(' Wait ...');

if coloured then

begin

colgrn:=green;colred:=red;
```

```
colcyn:=cyan;colyel:=yellow;

colwht:=white

end

else

begin

colgrn:=black;colred:=black;

colcyn:=black;colyel:=lightgrey;

colwht:=lightgrey

end;

randomethod:=1;rules:=7;{note 30}

banner:='New Zealand demonstration';{note 31}

command('dir votes.dat >votes.tep',nn);{note 32}

assign(tmpp,'votes.tep');reset(tmpp);

repeat

readln(tmpp,str1)

until(pos('VOTES',str1)>0)and(pos('DAT',str1)>0);

filesize:=0;nn:=1;

while(str1[nn]<'0')or(str1[nn]>'9')do nn:=nn+1;

repeat
```

```
filesize:=10*filesize+ord(str1[nn])-48;

nn:=nn+1

until str1[nn]=' ';

erase(tmpp);

assign(outt,'votes.res');rewrite(outt);

assign(oei,'votes.oei');rewrite(oei);

assign(rslt,'votes.rlt');rewrite(rslt);

{All candidates have initial keep value of 1.0}

for cand:=1 to maxcandidates do keep[cand]:=1000000000;

{Form a ramfile ramm, provided that would not be too big}

okmm:=filesize<0.9*memavail;{note 33}

if okmm then ramfile(ramm) else assign(ramm,'votes.rmm');

rewrite(ramm);

assign(datafile,'votes.dat');reset(datafile);

assign(tmpp,'votes.tip');rewrite(tmpp);

{Pass over any initial spaces, tabs or newlines}

while(datafile^=chr(32))or(datafile^=chr(9))do get(datafile);

if(datafile^>='0')and(datafile^<='9')then

begin
```

```
{Non-coded data on votes.dat}

read(datafile,numcandidates,numseats);

{Pass over any spaces, tabs or newlines}

while(datafile^=chr(32))or(datafile^=chr(9))do get(datafile);

{Look for withdrawals}

read(datafile,temp1);

if temp1<0 then

repeat

keep[-temp1]:=0;read(datafile,temp1)

until temp1>0;

{Code data and put it into ramm}

while temp1>0 do

begin

if temp1<50 then write(ramm,'$',chr(temp1+48))

else write(ramm,chr(temp1 div 2500+48),

chr((temp1 mod 2500)div 50+48),chr(temp1 mod 50+48));

repeat

temp2:=reed;write(ramm,codecan(temp2))

until temp2=0;
```

```
read(datafile,temp1)

end;

write(ramm,'$0')

end

else

begin

{Coded data on votes.dat}

{Pass over any spaces, tabs or newlines}

while(datafile^=chr(32))or(datafile^=chr(9))do get(datafile);

numcandidates:=innum;numseats:=innum;

{Pass over any spaces, tabs or newlines}

while(datafile^=chr(32))or(datafile^=chr(9))do get(datafile);

{Look for withdrawals}

if datafile^='-' then

repeat

get(datafile);read(datafile,cw);

keep[decodecan(cw)]:=0

until datafile^<>'-';

{Copy direct to ramm, but omitting any spaces or newlines}
```

```
stopp:=0;

repeat

read(datafile,cw);

if cw='$' then stopp:=1 else

if(stopp=1)and(cw='0')then stopp:=2 else stopp:=0;

if cw<>' ' then write(ramm,cw)

until stopp=2

end;

if not okmm then

begin

close(ramm,true);assign(ramm,'votes.rmm')

end;

reset(ramm);

outtwice(false,banner,false);

reply:='Version 6.7.7 - NZ rules';{note 34}

outtwice(false,reply,false);

writeln(rslt,banner);writeln(rslt,reply);

writeln(outt,' Number of candidates = ',numcandidates:1);

writeln(outt,' Number of seats = ',numseats:1);
```

```
if fstat('votes.con')then{note 35}

outtwice(true,'Note that this election was subject to
constraints.',false);

total:=0;table:=0;

iteration:=0;countno:=0;

numelected:=0;numexcluded:=0;

randomused:=false;aheadused:=false;

electindex:=0;

incomplete:=true;numwithdrawn:=0;

for cand:=1 to numcandidates do

begin

readname(name[cand],namelength);status[cand]:=hopeful;

ignore[cand]:=false;

{Exclude any withdrawn candidate}

if keep[cand]=0 then

begin

status[cand]:=excluded;numexcluded:=numexcluded+1;

numwithdrawn:=numwithdrawn+1;writeln(tmpp,'''1',name[cand]);

outtwice(false,concat('Withdrawn before count - ',name[cand]),false)
```

```
end;

deemexcl[cand]:=0;ahead[cand]:=numcandidates-1

end;

readname(title,titlength);

close(datafile,true);

if numinv>0 then

begin

if rules=1 then write(outt,' ');

write(outt,'Number of in');

if rules=1 then write(outt,'valid') else write(outt,'formal');

writeln(outt,' votes = ',numinv:1)

end;

{Read value of first vote}

count:=inval;

while count>0 do

begin

total:=total+count;

repeat

cand:=incand
```

```
until cand<1;

{Read next vote value}

count:=inval

end;

settiebreaks;constraintquery;

{Find number of digits to show in fractional parts in output}

fracdigits:=4;

if total>999 then fracdigits:=fracdigits-1;

if total>99 then fracdigits:=fracdigits-1;

if total>9 then fracdigits:=fracdigits-1;

{Initiate output for result sheet}

writeln(tmpp,title);

writeln(tmpp,numcandidates:1,' ',numseats:1,' ',numinv:1,' ',total:1);

for cand:=1 to numcandidates do writeln(tmpp,shortname[cand]);

findquota;

cursoroff;

if coloured then

begin

paper(blue);ink(yellow)
```

```
end;

clrscr;newline(4);

writeln(' ':10,'The numbers that will appear at the bottom of the
screen');

writeln(' ':10,'during the count are of no importance, except to
indicate');

writeln(' ':10,'that calculations are continuing. As long as they are');

writeln(' ':10,'changing, there is no need to kick the computer even
if');

writeln(' ':10,'the next decision seems to be taking a long time.');

if not okmm then

begin

writeln;

writeln(' ':10,'As this is such a large election, the first such numbers');

writeln(' ':10,'may not appear for some time, and they may change
only');

write(' ':10,'infrequently. Don''t kick the computer anyway.')

end;

anykey(25);

paintscreen(0);
```

```
repeat
if numcandidates-numexcluded<=numseats then
begin
{All remaining candidates can be elected, for
their number is no greater than the seats available}
printout(4);someone:=false;
for cand:=1 to numcandidates do
if status[cand]=hopeful then
begin
elect(cand);someone:=true
end;
if someone then showelect;
finish:=true
end
else if numelected+numguarded=numseats then
begin
{All guarded candidates can be elected, for
their number is no greater than the seats available.
All others must be excluded}
```

```
printout(8);someone:=false;

for cand:=1 to numcandidates do

if guarded[cand] then

begin

elect(cand);someone:=true

end

else if status[cand]=hopeful then exclude(cand,false);

if someone then showelect;

finish:=true

end

else

begin

countvotes;

{Update the ahead array unless already complete}

if incomplete then update;

findquota;oeioutput;

surplus1:=0;surplus2:=0;

someone:=false;numzero:=0;

numb:=0;
```

```
for cand:=1 to numcandidates do

if(status[cand]=hopeful)and not guarded[cand] then

begin

if(votes1[cand]>quota1)or(votes1[cand]=quota1)and

(votes2[cand]>=quota2)then numb:=numb+1

end;

if numelected+numguarded+numb>numseats then{note 36}

begin

if numguarded>0 then

begin

quota1:=total+1;quota2:=0;

outtwice(true,'Because of guarded candidates, too many
have',false);

outtwice(false,'become electable. Special action follows.',false);

pseudoquota:=true

end

else quota2:=1

end;

minvotes1:=quota1;minvotes2:=quota2;
```

```
nextmin1:=quota1;nextmin2:=quota2;

{Find mincand as continuing non-guarded candidate with minimum
votes,

numzero as number with zero votes,

minvotes as minimum number of votes,

nextmin as the number of votes of the lowest candidate

other than mincand. (NB nextmin may equal minvotes)}

for cand:=1 to numcandidates do

begin

if(status[cand]=hopeful)and not guarded[cand] and

(votes1[cand]=0)and(votes2[cand]=0)then

begin

numzero:=numzero+1;mincand:=cand

end

else

if(status[cand]=hopeful)and not guarded[cand] and

((votes1[cand]<minvotes1)or(votes1[cand]=minvotes1)

and(votes2[cand]<minvotes2))then

begin
```

```
nextmin1:=minvotes1;nextmin2:=minvotes2;
minvotes1:=votes1[cand];minvotes2:=votes2[cand];
if numzero=0 then mincand:=cand
end
else
if(status[cand]=hopeful)and not guarded[cand] and
((votes1[cand]<nextmin1)or(votes1[cand]=nextmin1)
and(votes2[cand]<nextmin2))then
begin
nextmin1:=votes1[cand];nextmin2:=votes2[cand]
end
else
if(votes1[cand]>quota1)or(votes1[cand]=quota1)and
(votes2[cand]>=quota2)then
begin
{Find the total surplus}
surplus1:=surplus1+votes1[cand]-quota1;
surplus2:=surplus2+votes2[cand]-quota2;
adjust(surplus1,surplus2);
```

```
if status[cand]=hopeful then
begin
elect(cand);someone:=true
end
end
end;
if someone then
begin
{printout with parameter 1 has no special message}
printout(1);showelect
end;
if numelected<numseats then
begin
finish:=false;
if numzero<2 then
begin
{Find gap between lowest and next lowest}
if numzero=1 then
begin
```

```
temp1:=minvotes1;temp2:=minvotes2

end

else

begin

temp1:=nextmin1-minvotes1;

temp2:=nextmin2-minvotes2;

adjust(temp1,temp2)

end;

shownumbers;

if(temp1>surplus1)or(temp1=surplus1)and(temp2>surplus2)then

begin

{Lowest candidate cannot overtake, so can

safely be excluded}

printout(2);exclude(mincand,false)

end

else if(surplus1=0)and(surplus2<100000)then

begin

{No surplus remains, so lowest candidate

must be excluded}
```

```
printout(3);exclude(lowestcand,false)

end

end

else

begin

temp1:=0;temp2:=0;

shownumbers;

if(surplus1=0)and(surplus2<100000)then

begin

if numcandidates-numexcluded-numzero>=numseats then

begin

{Exclude all with zero votes}

printout(6);

for cand:=1 to numcandidates do

if(status[cand]=hopeful)and not guarded[cand] and

(votes1[cand]=0)and(votes2[cand]=0)then exclude(cand,false)

end

else

begin
```

```
{Exclude those with zero votes, but retaining
enough candidates to fill all seats}
printout(7);
for cand:=1 to numcandidates-numexcluded-numseats do
exclude(lowestcand,false)
end
end
end
end
else
{numelected=numseats}
finish:=true
end;
if not finish then
begin
changed:=false;
for cand:=1 to numcandidates do
if status[cand]=excluded then
begin
```

```
if(votes1[cand]>0)or(votes2[cand]>0)then changed:=true

end

else if(votes1[cand]>quota1)or(votes1[cand]=quota1)and

(votes2[cand]>quota2)then

begin

{Find new keep value as present keep value

times quota/votes}

multiply(temp1,temp2,quota1,quota2,keep[cand]);

divide(temp2,temp1,temp2,votes1[cand],votes2[cand]);

if keep[cand]<>temp2 then changed:=true;

if temp2>999999999 then keep[cand]:=1000000000 else
keep[cand]:=temp2

end;

if not changed then

{Special case. Lowest candidate must be excluded}

begin

printout(9);exclude(lowestcand,false)

end

end
```

```
until finish;

someone:=false;

for cand:=1 to numcandidates do

if status[cand]=hopeful then someone:=true;

if someone then

begin

{Exclude remainder because all seats are now filled}

for cand:=1 to numcandidates do

if status[cand]=hopeful then exclude(cand,true);

printout(5)

end;

writeln(outt);writeln(outt);

writeln(outt,'Election complete');shutdown(outt);

shutdown(oei);

{Terminator for result sheet data}

writeln(tmpp,'-3');close(tmpp,true);

assign(tmpp,'votes.tip');reset(tmpp);

{Form output for result sheet}

writeln(rslt,rules+100:1);
```

```
for cand:=1 to numcandidates do

writeln(rslt,cand:1,' ',deemexcl[cand]:1,' ',keep[cand]:1);

writeln(rslt,'0 0 0');

{Copy from temporary file}

for counter:=1 to numcandidates+2 do

begin

readln(tmpp,reply);

while reply[1]='"' do readln(tmpp,reply);

writeln(rslt,reply)

end;

readln(tmpp,reply);

continue:=true;

while((reply[1]='"')or(reply[1]='-'))and continue do

begin

if reply[1]='-' then

begin

writeln(rslt,reply);

if reply[2]='3' then continue:=false;

if continue then readln(tmpp,reply)
```

```
end

else readln(tmpp,reply)

end;

{Some of the order of items needs changing from tmpp to rslt.

This change is effected by putting some items into ramm
temporarily}

if not okmm then

begin

erase(ramm);ramfile(ramm)

end;

repeat

rewrite(ramm);

for counter:=1 to numcandidates+2 do

begin

writeln(ramm,reply);readln(tmpp,reply);

while reply[1]='' do readln(tmpp,reply)

end;

repeat

writeln(rslt,reply);readln(tmpp,reply);
```

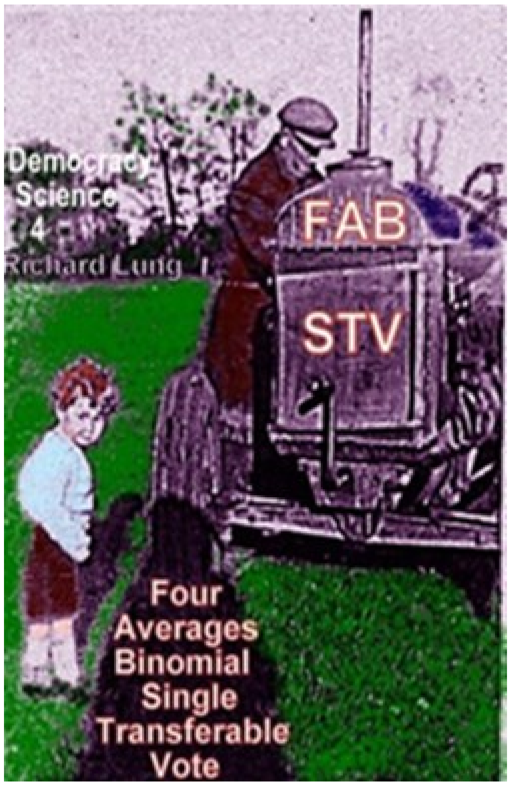
```
while reply[1]='' do readln(tmpp,reply);

finish:=(reply[1]='-')and(reply[2]='3')

until finish or(reply[1]<>'-');

reset(ramm);

for counter:=1 to numcandidates+2 do

begin

readln(ramm,str2);writeln(rslt,str2)

end;

if finish then writeln(rslt,'-3')

until finish;

reset(tmpp);

{Copy footnote indicators to result file}

finish:=false;

while not finish do

if eof(tmpp)then finish:=true else

if tmpp^=chr(219) then finish:=true else

begin

readln(tmpp,reply);

if reply[1]='' then writeln(rslt,reply)
```

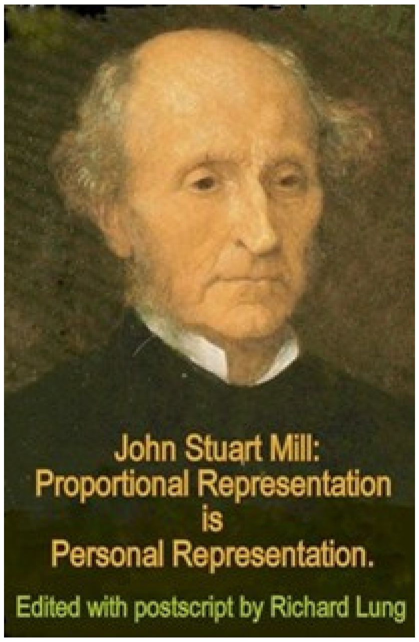
```
end;

shutdown(rslt);erase(tmpp);

anykey(25);

cursoron;initscreen{note 37}

end.
```

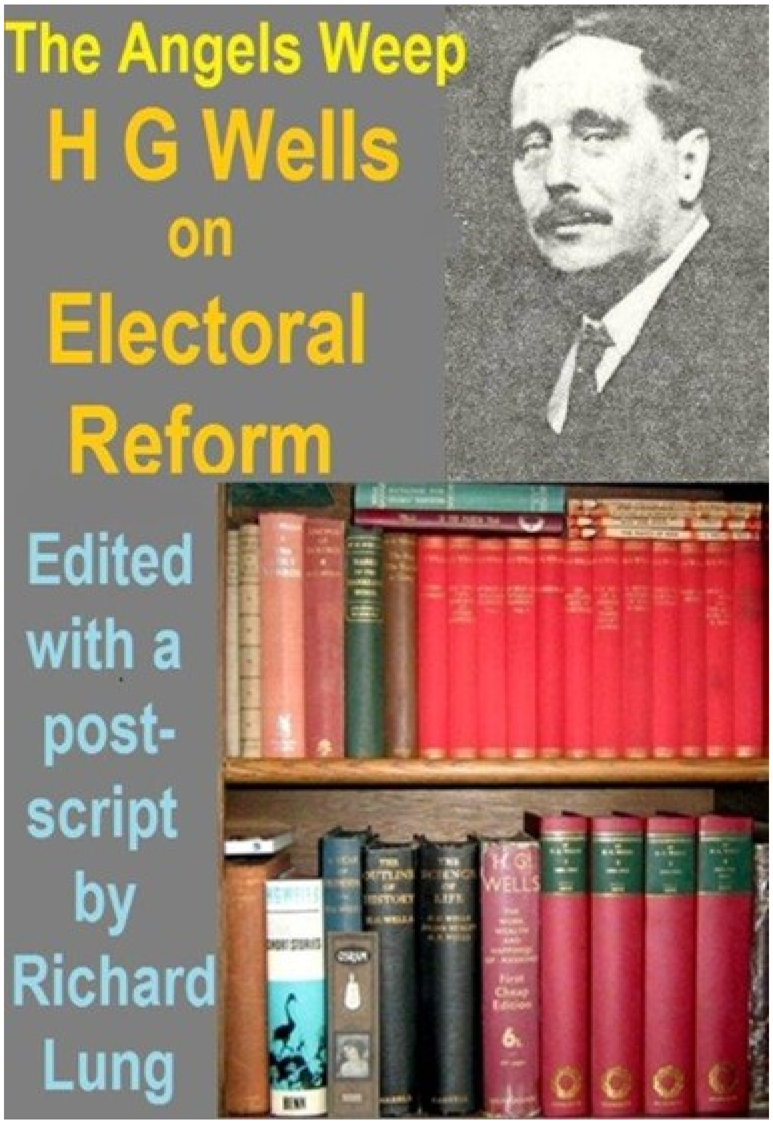
The Editor

Table of contents

Richard Lung has edited two other historic landmarks in electoral research and reform, by John Stuart Mill and H.G. Wells, as well as a Democracy Science series of books of his own:
Peace-making Power-sharing.
Scientific Method of Elections.
Science is Ethics as Electics.
FAB STV: Four Averages Binomial Single Transferable Vote.




John Stuart Mill:
Proportional Representation
is
Personal Representation.
Edited with postscript by Richard Lung



The Angels Weep

H G Wells

on

Electoral Reform

Edited with a post-script by Richard Lung