Macros

Fast and secure multiple-option tests

Jordi Saludes

Introduction

Some of us are teachers and have to evaluate the performance of students many times a year. In such cases multiple-option tests are a very common solution and with the introduction of optical mark readers automation of the grading process has become possible. However in case of space or time constraints, such devices give no real help: In a crowded examination room we have to produce several versions of the exam to prevent peeking.[1] So we have to give the reader service a solution sheet for each version. On the other hand when the reader serves a big community the delay in getting the grades is often too large (at least to give feedback to the students).

The convenience of using TEX to typeset exams in different versions by scrambling questions and answers was first addressed in [D]. I tried to go a step further to help those of us that can not rely on special hardware. I propose a procedure to easily make many versions of multiple-option tests, that relieves the teacher of a boring task at the cost of charging the students with a bit more of work.

The teacher prepares the test using a macro package to be described below (see an example in appendices A and B). Running the test file produces tests in many versions.

In these tests, answers are labeled by (apparently random) integer numbers. To answer a test the student, with the help of a pocket calculator, adds the label numbers of the answers he thinks to be right and writes the totals in the bottom of each page. The student identification and page totals are the only marks allowed in the exam. To avoid errors in the addition, label numbers displayed are in fact multiplied by a small factor like $d = 7$ or $d = 13$ (that we call the *detection factor*), thus the total must be also multiple of d.

Since different versions have different label numbers and moreover answers are scrambled, this is a secure protection against copying. Second, since the answers are coded in only a number, entering exams in a computer for grading can be very fast. (See some suggestions at the end of the paper.)

The knapsack problem. It is an ancient puzzle: Given the total weight s of a knapsack and the weight $w_1, \ldots, w_n$ of individual objects, determine which objects are in the bag. In the general case, this problem is hard when the number of objects is large. Given the difference of computational effort on solving the knapsack problem (going from s to the objects) versus stating it (going from objects to s), this problem can be used as a *trapdoor* function, giving a public key cryptosystem [S], [Ro].

In this paper I slightly modify this cryptosystem to adapt it for doing tests in the classroom. Students give their answers by adding the label numbers of chosen options (encrypting by different public keys). To get back the answers, the teacher (who knows the private key) decrypts the total.

In what follows, *knapsack problem* refers to the following slightly more general problem:

$(\mathcal{B}, s)$: *Let $\mathcal{B} = \{B_1, \ldots, B_N\}$ be a collection of mutually disjoint sets B_i of positive integers. Given s a positive integer determine $b_i \in B_i \cup \{0\}$ for $i = 1, \ldots, N$ such that $s = b_1 + \ldots + b_N$.*

In other words, take an object out of some of the bags in $\mathcal{B}$ such that the total weight be s.

In general $(\mathcal{B}, s)$ has no solution or, given a solution exists, it is possibly not unique. Finding a solution in general amounts to checking all the different sums, which is not feasible when the total number of elements is large.

When applied to the case of student tests, B_i will be the set of label numbers for the answers of question i. Choosing $b_i \in B_i$ means marking the corresponding answer to question i and taking $b_i = 0$ corresponds to skip this question. This way, determining the marked answers from the total s implies the solving of a knapsack problem.

It is important to use a labeling family $\mathcal{B}$ for which the knapsack problem has a unique solution, for otherwise we would be unable to decide among several test markings with the same sum. It is also important that the teacher can easily solve the problem whereas it must be difficult for anyone else. In the following two sections we consider how to manage that.

Mixed-radix sequences. It is clearly true that when

$$\sum_{i=1}^{n} \max B_i < \min B_{n+1}, \tag{1}$$

the knapsack problem has a unique solution (provided it exists) and moreover, it is really easy to solve the problem by comparing s with the elements of $B_1 \cup \ldots \cup B_N$ arranged in decreasing sequence.

Let us construct a such problem with a given number of elements: Suppose we want B_i to have n_i elements for $i = 1, \ldots, N$. Take a sequence w_i recursively defined as

$$\begin{aligned} w_1 &= 1, \\ w_i &= (1 + n_{i-1}) w_{i-1} \quad \text{for } i = 2, \ldots, N. \end{aligned} \tag{2}$$

and set $B_i = \{ j w_i \mid j = 1, \ldots, n_i \}$ for $i = 1, \ldots, N$. It is easy to show that this collection fulfills condition (1). The sequence (w_i) is called a mixed-radix

[1] Peeking is a very popular sport in Spain.

sequence[2] for (n_i). An algorithm for solving the problem using (w_i) will be given below.

Modular arithmetic. We say that integers m and m' are congruent modulo k ($m \equiv m' \pmod{k}$) if and only if $m - m'$ is a multiple of k. Given m a positive integer there is exactly one $0 \le r < k$ such that $m \equiv r \pmod{k}$. This number is the remainder of the integer division of m by k and will be denoted $m \bmod k$.

We will consider now other knapsack problems related to equation (2). Take

$$k \ge (1 + n_N)w_N \tag{3}$$

and $1 < v < k$ an integer coprime with k (i.e., $\gcd(v,k) = 1$), and consider the problem for $\tilde{\mathcal{B}} = \{\tilde{B}_1, \ldots, \tilde{B}_N\}$ where $\tilde{B}_i = vB_i \bmod k$. Since the map $x \mapsto vx \bmod k$ is additive, it relates weights and totals of $(\mathcal{B}, s)$ with the corresponding ones of $(\tilde{\mathcal{B}}, \tilde{s})$. Therefore the new problem also has the uniqueness property, albeit condition (1) is not longer true. In fact modular multiplication totally distorts the order relation of (1) making the knapsack problem much more difficult.

We will use just this kind of knapsack systems to label the answers of the students sheets. Since, given a sum, there is exactly one combination of answers adding to this total. But, on the other hand, not having the answer labels a clear order it is not feasible to recover the answers from the sum. Thus we can safely have a stack of answered sheets on our desk with the students nosing around.

The problem is far easier for the teacher, since she knows not only the sum $\tilde{s}$ and v but also the key k. To get the answers she has to:

1. Solve $\tilde{s} \equiv vs \pmod{k}$ for s. This implies computing the *multiplicative inverse* $\bar{v}$ of v modulo k, namely a positive integer such that $v\bar{v} \equiv 1 \pmod{k}$.
2. Find $b_1 \ldots, b_n$ in the original problem for s. Since this problem has a mixed-radix sequence $(w_1, \ldots, w_N)$ we can easily find the solution by iterated integer division.

See below in the grading section for a description of the algorithms used.

[2] These sequences appear, for example, in the old British monetary system and in the measure of time [K]: To convert days, hours, minutes and seconds from/to seconds, we consider the sequence 59, 59, 23 that gives $w_1 = 1$, $w_2 = 60$, $w_3 = 3600$, $w_4 = 86400$.

Preparing a test

To make a test, we write a file `foo.tex` such as the one in appendix B. We begin the file with `\input knst.tex` to load the macros and give commands like `\title` or `\date` to complete the header. Now we issue `\plainversion` to instruct TeX we are in *plain mode* and then `\test{`⟨*k*⟩`}` to begin the exam. We choose the key k in the range of TeX integers, its magnitude depending on the number of questions per page wanted: the more questions the larger the value (see the section on choosing numbers). This number (to be kept private) allows the encryption/decryption of the test. Now we give the questions as follows

```
\qtn ⟨question text⟩
 \anw ⟨1st option text⟩
          \anw ⟨2nd option text⟩
   ⋮
\endqtn
```

We mark each right answer using `\Anw` instead of `\anw`. The package will display a warning if there is no marked option or an error if there are many. Remember that in the student versions, questions (and answers inside questions) will be scrambled, so avoid using expressions such ‘In the question above’. However we can write ‘None of the above’ options, by using `\fix`. This command will keep following options (inside the current question) in place.

Anyway, if we want to keep the question order while scrambling answers we can put a `\fixqtn` right after the `\test` command.

We can force a page break with `\newpage` between `\endqtn` and the next `\qtn`. The file ends with `\endtest`.

Making student versions. Now we delete (or comment out) `\plainversion` in `foo.tex` and provide a version number $1 < v < k$ by assigning v to counter `\verno` before the `\test` command. I suggest to take v large since that way the generated label numbers appear as randomly chosen. As stated before, k and v must be coprimes, otherwise we will get an

```
Invalid stepper/version number
```

error. (See also the section on choosing numbers for suggestions on how to take these.)

We are now in *production mode*. Each time we run TeX on this modified file we obtain a student test (version v). Instead of manually changing `\verno` for each different version of the test, we can use `\stepversion` command as explained below.

The mechanism. In both plain and production mode, the package computes the label numbers of answers and uses vertical boxes to compose questions and answers.

In plain mode (when `\plainversion` is issued), label numbers are not displayed and $v = 1$ is assumed. TEX keeps track of w_i's, info about questions, page breaks, and records all this stuff in file `foo.ans` (as `\opts` and `\pagebot` commands).

Note that a page break may occur by

- Key overflow. As soon as $w_i > k$, the page is terminated in order to keep condition (3) true. The question being processed will be placed in the next page. In the log file `foo.log`, this condition is signaled by a '`*`' just before the shipping of the page.
- Normal page completion. This is the result of undisturbed TEX page builder. This case includes forcing page breaks using `\newpage`.

In both cases TEX writes a `\pagebot` line in `foo.ans`. This command will be used later in production mode to determine the page breaks. Each question generates a TEX `\mark` carrying the question number i and corresponding w_i. This information is used to fill the `\pagebot` and to correct future overestimated w_i's.

In production mode, the boxes containing question and answer text are scrambled using the code in [M]: Depending on the parity of a shift register r questions/answers are appended or prepended to the current list. We can use `\SRset` to set r. The value $r = 0$ means no scrambling at all. In this mode `\newpage` instructions are skipped, since page breaking depends only on previously recorded `\pagebot`s.

The macro package deals with the following files:

- `foo.dvi`. In plain mode it contains the plain version of the test with questions numbered and correct options marked. In production mode it contains a scrambled student test with label numbers.
- `foo.ans` is generated at plain step and read on production. It contains relevant information for breaking pages and grading, such as the key k and the detection factor d. For each question it includes the number of options and the position of the correct answer.
- `foo.aux` is an auxiliary file generated by `\stepversion` containing v and r assignments. If we put `\stepversion{`$\langle t\rangle$`}` just before `\endtest`, where $1 < t < k$ is an integer coprime with k, TEX will step both the scrambler r and the current version number by $v \leftarrow t \cdot v \bmod k$, and then it will write commands to set the new value of v and r in `foo.aux`. This way, without having to change `foo.tex`, each new TEX run will give a new test with different scrambling and version number.

To work like that, command `\test` in production mode obtains the value of v as follows:

1. If `\verno` > 0 use this value as v; this branch is taken when we set `\verno` by hand at the beginning of the test;
2. Otherwise search for a file named `foo.aux` and expand it.
3. If this file is not found, read command from console. This option could be useful in environments supporting pipes (like UNIX).

Grading exams

To grade an exam we have to collect the totals and version number v. A program then computes the multiplicative inverse $\bar{v}$ modulo the key k and uses it to decrypt totals and obtain the options that the student chose.

I give now the algorithms to decode and grade exams from the total and version number in the `awk` language. The reasons are twofold: First, this language is easily readable, and second, the program really works without the hassle of declaring variables, opening files, etc. However, I do not recommend this program for real life usage, because it works with only one page (a total), and because computations are done in floating point. In my machine this means exact integer arithmetic if the magnitude of the numbers involved is less than 2^{31}. Thus we can run into a loss of precision when using large keys and/or version numbers and, what is worse, no notification of that loss will be given.

For more serious usage, I have a C implementation which uses a *Binary Extended Euclid Algorithm* to avoid multiplication of large numbers [C, page 18] and it is not limited to a unique page. I intend to submit it to the CTAN archives.

Decoding. Assume we have a data file, with lines containing version number v, total $\tilde{s}$ and an identifier for each student.

```
    38353 11073202 First student
    2567   4672433 Second student
⋮
```

Concatenate `foo.ans` with the data file and run `awk` using the following code:

First initialize some variables

```
BEGIN   {wa=1;nqn=0;}
```

Read instructions from the ans file getting k, d and computing w_i using (2). The ans file includes a \Key line that gives k and the detection factor d (columns 2 and 3):

```
/Key/     {key=$2; df=$3; next}
/pagebot/ {next} # ignore pagebot
```

It has also an \opts line for each question, which describes the number of options, the index of the correct answer, and the current value of w_i at the moment it was computed. Note that due to differences between the point where w_i is computed and when the TeX page builder activates, the values of the last column in an \opts line can be overvalued; I use this column only to have a look at the inner workings of the package. The true values of w_i are computed one at a time as follows

```
/opts/    {nqn++;w[$2]=wa; wa*=($3+1);
           rb[$2]=$4; next}
```

On lines from the data file, print the line and get v and $\tilde{s}$. Using the *Extended Euclid Algorithm* as in [K], [C], solve the equation

$$v\bar{v} + km = 1,$$

(it gives $\bar{v}$ in u1)

```
 {print $0; # print entire record
 u1=1; u3=$1; v1=0; v3=key;
 while(v3!=0) {
   q=int(u3/v3);
   t1=u1-v1*q;
   t3=u3-v3*q;
   u1=v1; u3=v3;
   v1=t1; v3=t3;
 }
 if(u1<0) u1+=key;
```

The instructions $t_i \leftarrow u_i - qv_i$ are the critical ones: If q and v_i were large at some step, the result could be wrong.

We have now to compute $s = \bar{v}\tilde{s} \bmod k$ where $\tilde{s}$ is in column 2. It is not possible to do it by straight multiplication since both $\bar{v}$ and $\tilde{s}$ are large, so we proceed by converting $\bar{v}$ to binary radix and converting multiplications to iterative modular sums in order to give the plain sum s in s:

```
 for (i=0; u1>0; u1=int(u1/2))
  bb[i++]=u1%2;
 sp=($2/df)%key;
 for (s=0;i>0;) {
  s=(s+s)%key;
  if (bb[--i]==1) s=(sp+s)%key;
 }
```

Now the quotients of iterate division by $w_1, \ldots, w_n$ will give the exam answers $b_1, \ldots, b_n$.

```
for(i=nqn;i>0;i--) {
 b[i]=int(s/w[i]); # answer of qtn i
 s%=w[i];
}
```

Comparing b_i with the correct answer gives the grade. Note that a null quotient means an unanswered question.

```
for(i=1;i<=nqn;i++) printf "%d ", b[i];
print "";         # new line
nra=0; nwa=0;
for(i=1;i<=nqn;i++) {
 if (b[i]==0)     # No answer
  c=" ";
 else
  if (b[i]==rb[i]) {
   c="+"; nra++  # Right
  } else {
   c="-"; nwa++  # Wrong
  }
 printf "%s ", c;
 }
print "";         # new line
print nra,"right,",nwa,"wrong.";
}
```

Customizing

Style commands are grouped near the end of the macro file. They include

- \headline: the headline of each page in production mode;
- \pheadline: the headline in plain mode;
- \footline and \pfootline: the same for footlines;
- \qtnprompt: material to be put in front of question in plain mode. Usually the question number;
- \anwprompt expands to material that goes before each option in production mode. Uses \labelno which delivers the suitable label number;
- \df: the detection factor d. Fine detection factors are 7 or 13 (but not 4, 3 or 11) which detect a lot of mistypings and transpositions;
- \preanwskip must expand to a vertical skip between question and first option;
- \qtnskip must expand to vertical skip between questions.

Choosing numbers. Since w_i's grow exponentially (see (2)), we are usually forced to take k very large but bounded by 2^{31} (the greatest TEX integer), thus limiting the number of questions per page. For example, no more than 19 true/false questions fit on a page. On the other hand, taking k near that upper bound will cause us trouble when used in combination with the `\stepversion` command. A rule of thumb is: The product of the stepper t by the key should already be a TEX integer, namely $tk < 2^{31}$.

Besides k, we have to choose $1 < v < k$ coprime with k. This is easy to manage simply by trying v at random: In case v is not coprime with k, we will get an

```
    Invalid stepper/version number
```

error displayed on TEXing, so we have to try again. I recommend to take v large, for otherwise the answer labels of the initial questions appear in increasing sequence, thus providing students with unwanted clues on the setting and the key. A few experiments will quickly familiarize us with the right procedure.

A way to get a good v at first try consists on taking k free from low prime factors: From [S] we see that, given k, the ratio of valid v's is $\phi(k)/k$ where $\phi(k)$ stands for the Euler function. This ratio depends mainly on $1 - 1/p$ where p is the least prime factor of k. We can get both k and v coprime and large by taking prime numbers[3] and multiplying them until we have a sufficiently large k. Use the same procedure for v, but now take care to not choose any of the primes used in k.

Final notes and hints.

- Bring a portable computer at the examination room and grade exams *ipso facto*.
- Collect totals in a memory pocket calculator and upload to the main computer later.
- If your students have e-mail access, take the exam and provide each student with a control number depending on the total he gives. Ask them to send you a message with the version number, total and control. Process your mail box to give a list of grades or write a program for mailing back the grade to the originator.
- Using the `dviconcat` utility in mass production can save you a lot of work. Instead of making a bunch of `dvi` files, use that tool to concatenate all the versions into a big `dvi`. This way the printer driver has to initialize only once.

References

[C] Henri Cohen, *A Course in Computational Algebraic Number Theory*, Graduate Texts in Mathematics. Springer (1993).

[D] Don De Smet, *TEX Macros for Producing Multiple-Choice Tests*, *TUGboat* **12**, 2 (1991).

[K] D. E. Knuth, *Seminumerical Algorithms*, Addison-Wesley (1981).

[M] Hans van der Meer, *Random Bit Generator in TEX*, *TUGboat* **15**, 1 (1994).

[Ri1] P. Ribenboim, *The Book of Prime Number Records*, Springer.

[Ri2] P. Ribenboim, *The Little Book of Big Primes*, Springer.

[Ro] Kenneth H. Rosen, *Elementary Number Theory and its Applications*, Addison-Wesley (1993).

[S] Manfred R. Schroeder, *Number Theory in Science and Communication*, Springer (1990).

◇ Jordi Saludes
Edifici de l'ETSEIT (TR5)
Colom, 11
08222 Terrassa, Spain
`saludes@grec.upc.es`

[3] See [Ri1] or [Ri2] for a table of such numbers.

Appendix A: Example

A test **April 1995**

Name: ....

Each question has only one right answer. For each page, add the numbers at the left side of your answers and write the total by the Σ. Divide the total by 7. If your addition is right, the quotient must be integer (But this does not mean that your choices were correct). Do not forget to write down your name, but do not write anything else in the exam sheet.

Boolean question

2097123 ◇ True;
4194246 ◇ False

This question has a 'fixed' option at the end.

4295536 ◇ Not so well;
2147768 ◇ This is the right option;
6443304 ◇ Clearly wrong;
8591072 ◇ None of the above.

Example question with a displayed equation

$$\int_{\partial S} \omega = \int_S d\omega.$$

1610826 ◇ Option f;
1342355 ◇ Option e;
805413 ◇ Option c;
536942 ◇ Option b (and right);
268471 ◇ Option a;
1073884 ◇ Option d;
1879297 ◇ Option g;

Fuzzy question

6291369 ◇ Quite true;
3941021 ◇ Quite false;

Is this the last one?

1590673 ◇ Who knows;
3181346 ◇ Choose me;
4772019 ◇ None of the above;
6362692 ◇ All of the above.

Page 1. Version 38353. $\Sigma =$ $\Sigma/7 =$

Appendix B: File foo.tex

```
\input knst.tex
\date{April 1995}
\title{A test}
\verno=38353\SRset{162521}
%%\plainversion
\test{1234531}
\qtn Example question with
a displayed equation
$$\int_{\partial S}\omega=
  \int_S d\omega.$$
 \anw Option a;
 \Anw Option b (and right);
 \anw Option c;
 \anw Option d;
 \anw Option e;
 \anw Option f;
 \anw Option g;
\endqtn

\qtn This question has
a ‘fixed’ option at the end.
 \Anw This is the right option;
 \anw Not so well;
 \anw Clearly wrong;
 \fix\anw None of the above.
\endqtn

\qtn Boolean question
 \anw True;
 \Anw False
\endqtn

\qtn Fuzzy question
 \Anw Quite true;
 \anw Quite false;
\endqtn

\qtn Is this the last one?
 \anw Who knows;
 \anw Choose me;
 \fix\Anw None of the above;
 \anw All of the above.
\endqtn
%\stepversion{1243}
\endtest
```

Appendix C: The macros

```
%%% knapsack test macros.
\catcode`@=11
%% Shift Register
%% from H. van der Meer, TUB 15, 1
\newcount\@SR
\def\@SRconst{2097152}
\def\SRset#1{\global\@SR#1\relax}%
\def\@SRadvance{\bgroup
 \ifnum\@SR<\@SRconst\relax
  \@A=0
 \else\@A=1\fi
 \ifodd\@SR
  \advance\@A by1 \fi
 \global\divide\@SR by2
 \ifodd\@A
  \global\advance\@SR\@SRconst\relax\fi
 \egroup}

%% Arithmetic
\newcount\@A
\newcount\@B
\newcount\@C
\newcount\@x \@x=0\relax
\newcount\@y \@y=0\relax
\newcount\@w
\newcount\key
\newcount\df \df=1\relax
\def\qtnno{\the\@x\relax}
\def\inc@#1{\advance#1 by1\relax}
\def\mod@A#1{\@B=\@A \divide\@B by#1
 \multiply\@B by#1
 \advance\@A by-\@B\relax}
\def\gcd#1#2{{\@A=#1 \@C=#2
 \loop\mod@A\@C
  \ifnum\@A>0
  \@B=\@C \@C=\@A \@A=\@B
 \repeat
 \ifnum\@C=1\else
  \errmessage{Invalid stepper/version
    number "#1"}\fi}}
\def\adv@w{{\inc@\@y
 \global\multiply\@w by \@y}}
\def\labelno{{\@A=\@B
 \mod@A\key\multiply\@A by\df \the\@A}}

%% Boxing and unboxing
\newtoks\pfootline
\newtoks\pheadline
\newbox\lqtn@bx
\newbox\lanw@bx
\newbox\canw@bx
\newbox\cqtn@bx
\def\new@page{\unvbox\lqtn@bx
 \vfill\penalty-10000
 \global\@w=\verno \adv@w}
\def\append@bx#1#2{%
 \setbox#1=\vbox{\unvbox#1 #2}}
\def\prepend@bx#1#2{%
 \setbox#1=\vbox{#2\unvbox#1}}
\def\add@anw{\@SRadvance
 \iffix@anw
  \global\append@bx{\lanw@bx}{\theanw}%
 \else
  \ifodd\@SR
   \global\prepend@bx{\lanw@bx}{\theanw}%
  \else
   \global\append@bx{\lanw@bx}{\theanw}%
  \fi\fi}
```

```
\def\add@qtn{\@SRadvance
 \iffix@qtn
  \global
   \append@bx{\lqtn@bx}{\unvbox
     \cqtn@bx}%
 \else
  \ifodd\@SR
   \global
    \prepend@bx{\lqtn@bx}{\unvbox
      \cqtn@bx}%
  \else
   \global
    \append@bx{\lqtn@bx}{\unvbox
      \cqtn@bx}%
  \fi\fi}
\def\fix{\global\fix@anwtrue\relax}
\def\fixqtn{\global\fix@qtntrue\relax}
\def\qtn{\global\inc@\@x
 \@y=0 \@B=0
 \ifplain\else
  \ifnum\@x>\botqtn
   \new@page\@next\fi\fi
 \rightanw=0
 \global\fix@anwfalse
 \setbox\cqtn@bx=\vbox\bgroup
  \noindent\qtn@pr}
\def\endqtn{\egroup\add@anw
 \ifplain\ifnum\rightanw=0
  \immediate\write16{There is no
   option marked in question
   \the\@x}\fi\fi
 \setbox\cqtn@bx=\vbox{\box\cqtn@bx
  \preanwskip\box\lanw@bx
  \qtnskip}%
 \adv@w\wr@anw
 \ifplain
  \ifnum\@w>\key
   \message{*}\new@page\fi
 \else
  {\@A=\@w \mod@A\key \global\@w=\@A}\fi
 \mark{\noexpand
  \themark{\the\@x}{\the\@w}}
 \message{.}\add@qtn}
\def\Anw{\ifnum\rightanw>0
  \errmessage{There are several
  options marked in question
  \the\@x}\fi
 \global\rightanw=\@y
 \global\inc@\rightanw
 \anw}
\def\anw{\egroup
 \ifnum\@y=0\else
  \add@anw\fi
 \advance\@B by\@w\relax\inc@\@y
 \setbox\canw@bx=\vtop\bgroup
  \relax\noindent}
\newif\ifplain
\plainfalse
\newcount\verno \verno=0
```

```
\newcount\rightanw
\newif\iffix@anw
\newif\iffix@qtn
\fix@qtnfalse
\let\read@line=\relax
\let\@read=\relax
\def\theanw{\hbox{\vrule
 height10pt width\z@\relax
 \anwprompt\ \box\canw@bx\hfill}}
\newcount\verno
\newwrite\anwfile
\newwrite\auxfile
\newdimen\labelw
\def\test#1{\key=#1
 \setbox3=\hbox{\multiply\key by\df
  \the\key}%
 \labelw=\wd3\relax
 \ifnum\verno<1\relax
  \immediate
   \openin\auxfile=\jobname.aux\relax
  \read\auxfile to\@read \@read
  \immediate\closein\auxfile\relax\fi
 \gcd{\the\verno}{\the\key}
 \@SRadvance
 \global\@w=\verno
 \header
 \setbox\lqtn@bx=\vbox{}
 \ifplain
  \message{Plain version}\SRset{0}%
  \global\@w=1
 \else
  \message{Version \the\verno}
  \immediate
   \openin\anwfile=\jobname.ans\relax
 \fi\@next}
\def\endtest{%
 \ifplain
  \write\anwfile{\string
   \Key\space\the\key\space
   \the\df\space}%
  \closeout\anwfile
 \else
  \closein\anwfile\fi
 \unvbox\lqtn@bx\vfill\supereject\end}
\def\stepversion#1{\gcd{#1}{\the\key}%
 \immediate
  \openout\auxfile=\jobname.aux\relax
 {\ifnum\@SR=0
   \@SR=\@SRconst
   \divide\@SR by3 \fi
  \@SRadvance
  \@A=\verno
  \multiply\@A by#1\relax
  \mod@A\key\relax
  \immediate\write\auxfile{\string
   \verno=\the\@A\space
   \string\SRset{\the\@SR}}}%
 \immediate\closeout\auxfile}
\def\plainversion{\plaintrue\verno=1
```

```
  \let\newpage=\new@page
  \def\wr@anw{\immediate
    \write\anwfile{\string\opts
     \space\the\@x\space\the\@y\space
     \the\rightanw\space\the\@w\space}}
  \immediate
   \openout\anwfile=\jobname.ans\relax
  \def\qtn@pr{\llap{\qtnprompt}}
 \def\anwprompt{\hbox to\labelw{\hss
  \ifnum\rightanw=\@y
   $\bullet$\ \fi}}
 \footline=\pfootline
 \headline=\pheadline
 \let\@next=\relax
 \output={\botmark\plainoutput}
 \def\add@qtn{\unvbox\cqtn@bx\penalty-50}}
 \def\themark#1#2{\immediate
  \write\anwfile{\string\pagebot
   \space #1\space}%
  \global\divide\@w by#2\relax}
 \let\wr@anw=\relax
 \let\qtn@pr=\relax
 \def\@next{\read\anwfile to\@read
  \@read}
 \def\Key #1 #2 {\@next}
 \def\opts #1 #2 #3 #4 {\@next}
 \def\pagebot #1 {\def\botqtn{#1}}
 \def\botqtn{1000}
 \let\newpage=\relax

 %% Style commands
 \def\qtnprompt{\bf\qtnno.\ }
 \pfootline={\hss
  Plain version $\dots$\hss}
 \pheadline={\ifnum\folio>1
  \thetitle\hss\folio\else\hss\fi}
 \footline={Page \folio. Version
  {\tt\the\verno}.
  $\Sigma=$\hskip7em
  $\Sigma/\the\df=$\hfill}
 \def\date#1{\def\thedate{#1}}
 \def\title#1{\def\thetitle{#1}}
 \def\header{
  \line{\bf\thetitle\hfill\thedate}%
  \smallskip
  \line{Name:\ \dotfill}%
  \medskip
 {\baselineskip=9pt
  \noindent\small
  Each question has
  only one right answer.
  For each page, add the numbers at
  the left side of
  your answers and write the total by
  the $\Sigma$. Divide the total
  by~\the\df. If your addition is right,
  the quotient must be integer
  (But this does not mean
  that your choices were correct).
  Do not forget to write down your name,
  but do not write anything else in the
  exam sheet.\par}
  \medskip}
 \df=7
 \def\anwprompt{\hbox to\labelw{\it
  \hss\labelno\ $\diamond$}}
 \def\qtnskip{\vskip 20pt plus25pt }
 \let\preanwskip=\smallskip
 \font\small=cmr8
```