Instructions

December, 1990

The purpose this set of macros, hereafter , is to assist in the preparation of multiple choice tests. The first part of the preparation process involves typing in the test. Begin by typing
problem], and then type the problem just as you normally would in TEX. Then you type the multiple–choice answers: proceed each wrong answer with a
wrong] command and proceed the correct answer with a
correct] command. If you do not want any multiple choice answers, just don't type any. The problems will be numbered automatically when typeset, but we will also refer to the first problem in the file, the second problem in the file and so on. The file should conclude with an
end] or
bye] statement.

There is also a
note] command: when you have some material which is not a problem but which should appear in the test you type
note] and then the material. For example, if you have a true-false section, you might want to put a note at the start of this part to warn the students and to tell them how much each of these problems is worth. Notes should not have answers nor will they be numbered in the output, but they do acquire a number based on their order in the file: the first note is note 1, and so on. (Actually, notes can be numbered if you want and indeed the entire labelling process for both note and problems can be controlled by you, the user [see the Extras section below].)

You may have up to five different answers, but you don't need to have that many: just type in as many as you want (up to five), but if you have any answers, exactly one of them must be marked correct. If the problem is a true-false question just type
Tf] or
tF] after you are done with the problem. This tells that there are two answers; if you typed
Tf] then TRUE is the correct answer; if you typed
tF] then the correct answer is FALSE.

Any time during this process, you may TEX the document. You will get the problems set in the same order that they occupy in the file, and each problem will have the answers typeset after the problem, again in their natural order, with an underline next to the one that you indicated was the correct one. The notes will also appear at the location that they occupy in the file. The spacing between the lines will be rather cramped, but this will be adjusted later. The purpose of this part is just to get the data entered and to let you print it out for proof–reading. By default, your problems will be TEX'ed with no magnification at this stage, but, in the final run, magnification will be automatically set to
magstep1]. (You can change this with the
setmag] command:
setmag]{1000} is equivalent to
magnification] 1000 and must be invoked near the beginning of your document. If you do it too late you will get the TEX incompatible magnifications error. If you want even the proof–reading

runs at
magnification] 1200, just include
magnification] 1200 at the start of your file.)

When you are entering the test, behaves much like TEX itself, so you should get the desired results easily. One caveat is that braces tend to get removed in the input process so you should be generous. If you want "aardvark" to be bold, the usual TEX way to achieve this is to type {
bf] aardvark }. Having so few braces is dangerous with since the braces may get stripped off, turning the rest of your test turn into bold. The safe way to avoid this is to type {{
bf] aardvark }} so that, if a layer of braces is removed, one still remains and if both get through, no harm is done.

For people who like to write their own macros and things, life is a little more difficult and some understanding of how works is needed before you will achieve the desired results. First of all, the commands
problem] and
note] are actually macros and hence may contain nothing that any other macro may not contain:e.g. newcount] is
outer] and hence not permitted after the first
note] or
problem]. All the material between two consecutive
problem] commands (or a consecutive
note] and
problem] command, or two consecutive
note] commands, etc.) is read in, processed, and stored in a box: each answer is also read in, processed, boxed; and then the boxes are assembled into the correct order and put into the corresponding problem box. The net result of this is that material in the body of the test is nested inside several boxes and/or brace pairs.

Macros which are defined BEFORE the first
note] or
problem] command should work as expected, but any macro defined after this should be defined with
gdef] or it probably will not work when you apply it. If you need to change a constant that you have defined, you should probably use the
global] command or the constant will not change where you think it should. will usually read your file several times (once for each version that it is producing). Some macros should not be read twice and this does not happen to macros defined before the first
note] or
problem] command. The mechanism which insure this behavior does result in a peculiarity: *if you must define some macro to be*
outer], it must contain the letter **w** *somewhere in its name*, or will be unable to handle more than one version of your test at a time. Also, for people who do alignments,
+] has been set to
tabalign]. Macros defined after the first
note] or
problem] command will be read each time a new version of the test is produced and so should be

designed not to crash when read twice.

does not do too much to plain TEX and so should run with other macro packages. It does modify plain TEX's
shipout],
end] and
bye] commands and so should be loaded after any package which also modifies these. The author has not tried to run under AMSTEX, but it might work (try to produce a format file in which you have TEX'ed the macros using AMSTEX). Under LATEX things may be more difficult.

uses plain TEX's
headline] and
footline]. If you need headlines or footlines (particularly to write your own page numbering routines) we have supplied
myheadline] and
myfootline]. See the section, Extras, below for details.

A final warning: will need to input your file several times during one run, which it does using the standard TEX
input] command. This places some restrictions on file names. The worst restriction for a Macintosh user is that your file name must NOT contain any SPACES. (There are other offending characters, but spaces are the most common.) For safety, only use the letters a–z, A–Z, or numbers. If you want spaces, use the standard "kludge" of x_y for x y.

xbPermutations and typesetting. Once you have the data entered, it is easy to permute either the answers or the problems (or both). Of course you have to describe to how you want the permutations done. A *version* of the test consists of the text you typed in plus a choice of ordering for the answers to each problem and a selection and ordering for the problems. For each version, you must describe a selection of problems and notes from your file and in what order you want them to appear in the typeset document. You must also describe a permutation for the answers to each problem for each version.

Each version of your test has a number associated to it, which is used to explain to the version to which your permutation data applies. There is a variable,
firstversion], which is 1 by default but which you may set to any value you wish. There is also a variable,
lastversion], and thinks that your versions are numbered consecutively from
firstversion] up to
lastversion]. There is probably some limit to the number of versions, but the author has not had the patience to locate it (at this time it is at least greater than 50). (Versions with numbers greater than 100000 are used by to indicate it is in a special situation and these numbers should not be used by the casual user.)

By default,
lastversion] is 0, but when you include the command
lastversion] $=n$ in your file, you will TEX all the versions of the test from
firstversion] to

lastversion]. By adjusting
firstversion] and
lastversion] you can TEX any one version of the test or any range of them.

Version 0 is special (and is the one which occurs if you have not yet included a
lastversion =]n command). It will give you a printout of the test with problems and answers having the same order as they do in the file, but with the spacing between problems suppressed. Furthermore, the correct answer has an underline next to it. You may also set
lastversion =]$-n$: you will get a printout with all the permutations for version n, but the spacing will be the usual version 0 "cramped–style". (This is useful for giving versions to TA's or colleagues to have them worked.)

In the coming paragraphs, we will describe how to explain to how you want the answers for each problem permuted (from their order in the file) for each version and which problems and notes (and in what order) from the file you wish included in each version.

Each multiple–choice problem for a version needs to have a permutation to tell in what order to set the answers. Problems which are not multiple choice do not have permutations and neither do problems which are true-false: for these, the choice is always true, then false.

We begin by describing how to permute the answers corresponding to a fixed version, say v. The simplest way to do this is to place a permutation command in each problem. If there are 5 answers, a typical permutation command looks like
perm]v:*baced*, where v is the version number. If there are only four answers, you may leave out the e, or you may put it last. A command
perm] w:*cdeab*, where w is a number different from v will be ignored when typesetting version v, but will come into its own when typesetting version w.

When you typeset version v, the answers for this problem will be permuted from their order in the file as follows: the first answer in this version will be the old (b); the second will be the old (a); the third the old (c); the fourth the old (e); and the last, the old (d). (Recall that you can get a printout of the answers in their natural order, with the correct one indicated, by just TEX'ing the file with
lastversion]=0 [which is the default].)

The
perm] command for the problem may be placed anywhere after the
problem] command which begins the problem and before the next
problem] or
note] command.

There are two additional ways to produce the permutation data for the answers. You may want to declare a global permutation. The command
setglobalperms]{
perm]v_1:*abcde*
perm]v_2:*cdeba* ,etc.} sets every permutation for every problem for version number v_1 to *abcde*; the permutation for every problem for version number v_2 will be *cdeba*. You may declare as many of

these as you need and they need not be in any particular order. If you only want four answers, you can use all 5 letters and make the last one e, or you need only use the first 4. This command needs to be placed before the first
problem] or
note] command. It is most useful if you have entered the answers in the file in an order that you want to typeset. If you want version 1 to have the answers in their natural order, the one command, setglobalperms]{
perm]1:*abcde* }, will do this. A shorter version of this command which also works is
setglobalperms]{
perm]1:}.

Note: All
perm] commands are just a bit fussy as to spacing. The permtation itself (the *abcde* part) may not have any spaces in it. What happens is that reads up to the colon to see if the version number applies. If it does, it begins to read in letters, one at a time until it hits a character that is not an a–e, at which point it quits. (This same algorithm applies if it is trying to skip the rest of a
perm] command because the version for it does not apply.) The permutation that will be built is the one which sends 'a' to the first character read; 'b' to the second; 'c' to the third; and so on. If quits before reading 5 characters, the unread ones are send to themselves. Hence
perm]*v*:*ea* will result in a map which is not a permutation since the last answer in the file is to be set as 'a' (from the
perm] command) and as 'e' (by default since 5 letters were not read). This is NOT COOL.

You may put a
perm] *v*: command in a particular problem and it will take precedence over the global one for version number v of that problem. This means that you can do a global permutation which looks good for all but a small number of problems, and then permute the answers for these problems separately.

You may also enter a complete list of permutations for a version all at once. The command
setpermlist]{$\cdots$} will do this. It needs to be placed before the start of the test proper:i.e. before the first
problem] or
note] command. It can be placed in a separate file as long as this file is
input]'ed sometime before the start of the test proper. The material between the braces has a rather rigid format. You start with a"
perm]*v*:" command; on a new line (or at least after a space) type the permutation for problem 1, say *abcde*; on the next line type the permutation for the second problem; etc. until you have one permutation for each problem, one permutation per line. If there are only four answers for a given problem you can either use all five letters with e last again or just use a–d. As usual, you may not have any spaces between the letters. In this mode you may include more material after the permutation as long as it contains no spaces and begins with some character other than a–e. This material is ignored by , but can be helpful to you. For example, you can put the problem number after the permutation: a line which looks like *cdeab*17 is a good way to remember that *cdeab* is attached to problem 17. BUT, remember *cdeab* 17 is a bad ERROR because of the space. After you have entered the data for one set of permutations, you may enter the data for another or just close the command with a }. The

perm] commands do not need to be in any particular order. You must also remember that true-false questions may not have permutations, but if you create this file by just working your way through a copy of the test (with the problems in their natural order) you will generate the correct file. There are also other programs which you can use to generate a file of random permutations which you can easily edit to be fodder for this macro.

If you have two (or more) versions with the same permutations for the answers, you may type "
perm] v_1&v_2:" and then the list: this is equivalent to "
perm] v_1:" and the list followed by "
perm] v_2:" and the same list.

You may still put a "
perm] v:" command in a problem and it will take precedence over the one for version number v set from the list. It seems rare that you would want to do this and you still must have an entry in the list for the problem even though it is to be ignored. (It is occasionally useful for trying out a new permutation before putting it in the list.) You may also use a different method to enter the permutations for different versions: the method for one version could be global; the method for another could be from a list; and the method for a third could be from
perm] commands after each problem.

If encounters a multiple–choice problem for which it can find no permutation for the answers, it displays a message, uses *abcde* as a default permutation, and continues. You will need to re–TEX the file after entering the desired permutation, but this error will not mess up the run too badly so usually you may as well finish and see if you have forgotten more than one permutation.

There are two other errors associated with multiple–choice answers. If no
correct] answer is given for a problem, complains by writing a note to the log file for you. This error is annoying, but will not mess up your run too badly: assumes that the correct answer is the first one in your file and goes on (this can be serious if you believe the subsequent marked answer sheet). A more serious problem is to have two (or more)
correct] answers. Again will complain in the log file. If you really have just marked two answers correct, then this is not so serious, but the most common cause of this error is to forget a
problem] command. now thinks your problem text is part of an answer, which will look strange, but more seriously it now thinks you have ten answers for this problem. This is too many; will over–write some registers; and all bets as to subsequent behavior are off.

Once the permutation data for the answers is in place you may consider modifying the order of the problems. If you wish all versions to have the same problems in the same order as in the file, then you are done.

You may also prepare versions of the test using a subset of the problems in the file and these problems may appear in any order you wish. To do this you need the
setproblempermutations]{$\cdots$} command. The material between the braces has the following format. Begin with a "
version]v:" to explain to the reordering for version v. Next comes a stream of data to tell how to unbox the data it has saved. The entries in this list are separated by COMMA's and a number means to unbox the problem which had that number in the file. There are two other possible

entries in this stream: a "pb." (the period after the "pb" is important) will force a page break at this point; an "n.x" will unbox note number x at this point. You need not use the entire list of problems, just runs through your list doing the unboxing and page breaks as requested, and then quits when it reaches the end of the list. (Perhaps
setproblempermutations] is not the best name for this command since it will select a proper subset of the problems as well as permute them.)

If you want the same problem list for two (or more) versions, you can type "
version]v_1&v_2:" and then the list.

When you are typesetting a version of the test using the natural order for the problems, you can force a page break with the
pagebreak] command. will not break in the middle of a note or problem. Hence the break occurs immediately after the note or problem in which the
pagebreak] command is given and will cause the next problem or note to appear at the top of a new page. If you are typesetting version v, and there is a
version]v in
setproblempermutations], any
pagebreak] commands in the file are ignored and the page breaks are generated from the version information as discussed above.

Of course will also insert page breaks whenever it must in order to get the material to fit on a page (except that breaks can not occur in the middle of notes or problems). (If you have hired Thomas Mann to write word problems for you, you can imitate a page break in the middle of a problem by writing the first part of the problem as a note; include a
pagbreak] command in the note and finish the problem as a problem. [After reading the Extras section, you can even label your note with
global]
notelabel=]{{
count0]=
probcounter]
advance]
count0 by1]
hss]
number]
count0.]
]
] }} if you have not changed how labels problems. Be sure to kill
notelabel] before the next note is produced with
global]
notelabel=]{}.])

takes the extra space on a page and distributes it evenly amongst the problems before it sets each page. It does this by inserting a
vfil] after the box containing each problem. None of the extra space is appended to a note. As we will explain in the section on spacing, you can specify the minimum amount of space after a problem. One way to handle the page breaks is to just put the minimum space you want after each

problem and let put the breaks where it wants. It is not necessary to have "pb."'s if you like 's break points.

A common error in both answer and problem permutations is to include the same answer or problem twice. will warn you that this has happened with an "Empty $\begin{cases}\text{answer}\\ \text{problem}\\ \text{note}\end{cases}$ box!" message. In the typeset document the repetition will result in a blank box corresponding to the location of the second (and any subsequent) usage.

xccAnswer Sheets. also generates an answer sheet for the test. You should have a
title] {Math 999} and a
date] {December 25} command near the start of your file. This will put the title and date on the copy used for proof–reading and also puts this title and date on the answer sheet. (A
twolinetitle] command is available if you need two lines: it needs two variables, the first line and then the second line. There is also a
comment] {whatever} command which adds a comment line just under the date and an
answersheetfootline]{whatever} command which adds the material to the bottom of the page. Most of the answer sheet is taken up with lines for the students to mark their answers, but we also include a line for their name, a line for you to record their score, and optional lines for your name and their section.

The section number line can be suppressed by the
nosection] command and the line for your name can be suppressed by the
noprofessor] command. (The
Professor] {Professor Hilbert} command will add the professor line but fills it in with "Professor Hilbert".)

For each version of the test, will generate an answer sheet with the correct answer marked with a black dot. This sheet comes at the end of the test questions for that version. It is labeled with a version number and lists how many answers were a, how many were b, etc.

Immediately after the marked answer sheet for version 1 is produced, also produces an unmarked answer sheet. This unmarked sheet also includes your footline text, which is suppressed in the marked versions because that space is used for the answer information. Sometimes, because of the way you have permuted the problems, the answer sheets for different versions ought to look different. Whenever this happens, produces a new unmarked answer sheet immediately after the corresponding marked one. Whenever an unmarked answer sheet is produced also writes a "UNMARKED ANSWER SHEET" to the log file.

The answer sheet feature can be suppressed with the
noanswersheet] command. If this command is included in your file, then will not produce any answer sheets, but it will generate a file which contains the information as to how the tests were typeset and what the correct answers are. This file is called
jobname.answer] :i.e. if the file containing your test is called My_Test or My_Test.tex, this file will be called My_Test.answer. (You did remember no spaces in file names, didn't you?) You can process this file if desired to produce your own answer sheets. The format of this file is described in appendix A.1 below. You can also use the file to produce the standard answer sheets at a later

date, using the
writeanswersheet#1] command.

To do this, create a seperate file with your answer sheet formatting commands as usual and finish the file with
writeanswersheet]
input] file name
end] (but NOT
bye]). The file "file name" should be the one produced by when it was typesetting the tests with the
noanswersheet] command. (There should still be no spaces or other weird characters in the file name, but otherwise it can be any name you like [modulo your machine's limitations on file names]. Most likely it will still be "something.answer".) If you prefer you can modify the file produced by directly. Add your formatting commands before the original contents of this file (the material that is in the file when you first open it). Then add
writeanswersheets] just before the original material and an
end] just after it. The command
writeanswersheets] will not work unless it is the last command before the
end] (which cannot be replaced with a
bye]).

By default, the program puts a line with a "Page n" between the last problem on page $n-1$ and the first problem on page n (for $n \geq 2$) to help the students avoid marking the wrong numbered problem. This can be replaced with just a blank line or suppressed altogether. You can also get it to put "Page 1" just before problem 1 if you wish. These changes are accomplished with
pageannouncements=] {$xxxx$}, where $xxxx$ can be "none" (type {none} with no spaces) , or "blank" (again no spaces), or "withfirstpage", or "usual". If the entry is "none" then you get no extra lines at all; if it is "blank" then you get blank lines. If it is "usual" then you get the effect described above (and this is what you will get with no
pageannouncements] command at all). The "withfirstpage" will add the "Page 1" line. If you use anything else, you will get a message and be returned to the default.

By default, these page announcements are in 12pt roman, centered in their column, but you can change this if you wish. The actual text is produced by a macro,
mypageannouncement], which you may redefine to suit yourself. The default is
def]
mypageannouncement]{Page
number]
pagecounter]}, where
pagecounter] is a variable you may use in your own macro. For example, to move the announcements to the left hand edge of the column, simply put the following definition in your file:
def]
mypageannouncement]{
hbox] to
answersheetlinewidth]{Page
number]
pagecounter]

hfil]}}. Recall from below that
answersheetlinewidth] is a variable which holds the width of the column.

It is possible to put the data into two columns. The
twocolumn] command will do this: the format is
twocolumn] {b} where b tells the program where to do the break: if you put in a number for b, then that numbered problem appears at the top of the second column. If you put in "p.r" for b, r a number, then the line with "Page r" will appear at the top of the second column. (If you are suppressing page announcements then the next problem will appear.) If you put in "n.r", then the first problem or page break after note r will appear at the top of the second column.

It is also possible to select the *style* for the entries in the answer sheet as well as the labels in the main body of the test. We discuss this in the next section.

The answer sheet is composed of three boxes and material at the bottom of the page. The location of these items on the page can be adjusted. The first box is the header box which contains the title, date, any comment, the area for the student's name, professor, and section number. (This box is actually named
header] and you may use it in your own macros if you wish.) It is set so that its upper right corner has the coordinates specified by
rightheader] and
vertheader]. By default these are 0in. and -.5in. respectively, but can be adjusted by the user at will with a
global]
rightheader]={whatever} command. A second box contains the lines for you to record the student's score. It can be adjusted by changing the
righttotals] and
verttotals] dimensions, and the name of this box is
totals].

There is another box, the multiple–choice box, which contains the area for the students to mark their answers. The location of this box can be adjusted by changing the
rightmultchoice] and
vertmultchoice] dimensions. The defaults here are 0in., .8in. but the best way to adjust any of these boxes is with the
global]
advance] command. For instance "
global]
advance]
vertmultchoice]**by 1in**" will lower the multiple–choice box by 1in. on the page. The name of this box is
abcdebox].

Warning: Since the file is read several times,
advance]'s should be used cautiously. An
advance] like that suggested above should only appear before the first
note] or

problem] command or on subsequent answer sheets the multiple–choice box will migrate steadily down the page.

By default, the material at the bottom of the answer sheet is centered. If you prefer some other convention,
skipforfootlineofanswersheet] can adjust this for you. For example,
skipforfootlineofanswersheet]=0pt will left justify the material.

The length of a line of multiple choice answers is set by
answersheetlinewidth] and can be adjusted by setting it with a
global] command or changing it with a
global]
advance] command. The space after the problem number and before the first answer is set with problemnameskip]. In two column mode the length of a line of answers is roughly half of answersheetlinewidth] and the two columns are separated by a space of
columnspace]. The space between each line of answers is set with
multiplechoiceskip]. The space between the labels ((a), (b),etc. by default) is simply set using hfil]'s between the labels. You can also write your own problem numbering schemes as described in the Extras section below.

Finally, if all this is not enough to produce the perfect answer sheet, you may define your own routines. You may define a new command,
mymarkedanswersheet], to produce marked answer sheets and
myunmarkedanswersheet] to produce unmarked answer sheets. will call your commands rather than processing the data as above. Your routines should compose the answer page (or pages) and eject] them.

There are three commands,
placeheaderbox],
placetotalsbox] and
placemultchoicebox] which you can use in your macros to get the usual boxes. Hence
def]
mymarkedanswersheet#1]{
placeheaderbox]
placetotalsbox]
placemultchoicebox]
vfil]
eject]}
let]
myunmarkedmarkedanswersheet]=
mymarkedanswersheet] is how the usual answer sheets are generated. This relatively easy access to the output routines for the answer sheets is the reason for no "everyanswersheet" token list since if you really need one just write your own
mymarkedanswersheet] code. As a further example, note that the commands
def]
mymarkedanswersheet#1]{
setbox]

header]=
hbox]{ your header material }
box]
header]
placetotalsbox]
placemultchoicebox]
vfil]
eject]}
let]
myunmarkedmarkedanswersheet]=
mymarkedanswersheet] will produce an answer sheet with your header material in place of the standard stuff. The difference between marked and unmarked answers sheets is that for the marked ones passes the
abcdebox] with the answers marked and for the unmarked ones the
abcdebox] is unmarked.

xcLabel Styles. Both in the answer sheet and in the test, we need labels for the multiple choice answers. By default these are the lower case letters a–e, but they can be changed to any other scheme the user likes. The
setlabels] command will do this:
setlabels] $ABCDE$ will change them to upper–case;
setlabels] $rghyt$ will produce a scheme no one can remember except the computer. You must still enter answer permutations as through the labels were a–e, but they will print out in your chosen labels.

Each label has a style (e.g. italics, or roman, or typewriter, etc.). The style for the labels in the test itself is set with
let]
labelstyle] =
sl] or whatever. (The default is roman.) The style for the labels in the answer sheet is set with
let]
ansstyle] =
sl] or whatever. (The default is again roman.) Technically, these styles are macros, and if you are familiar with TEX fonts and font families, you can even write your own label styles.

Each label also has a border: to get an (a), the style is
sl] and the border is a pair of parentheses. This can be set with for the test with
let]
labelborder]=
parenthborder], which is the default. For the answer sheet, use
let]
ansborder]=
parenthborder], which is also the default. Another choice is
let]
ansborder]=
boxborder] which places a square box around the label of size
boxwidth] which is dropped a distance of

boxdepth] below the baseline. (
boxwidth] and
boxdepth] can be adjusted if needed by the usual
global]
advance] command.) The user can also create their own border by redefining the
ansborder] and/or
labelborder] commands. As an example;
def]
labelborder]#1{[{#1}} will create [a as a border. A more complicated example is
font]
magcmsy]=cmsy10 scaled 1200
def]
labelborder#1]{
hbox] **to0pt**{
textfont2=]
magcmsy]
hss]$
bigcirc]$
textfont2=]
tensy]
hss]}
hbox] **to0pt**{
hss]#1
hss]}} (where the
font] command is a "kludge" to get 14pt cmsy). This will produce a–e with circles around them:e.g. ⓑ .

xdProblem Spacing. TEX has a large number of spacing conventions which has partially disabled to get things to work well. has certain spacing parameters of its own in order to allow you to get the spacing you want. Descriptions of spacing parameters for the answer sheets occur in the Answer Sheet section above.

The body of a problem has a certain interline spacing. The TEX command
lineskip] = **4pt** will set this to 4pt. for the problem being set when it was invoked. This command in a note will set the interline spacing for the note. Without a
global] command, the result is local to the problem or note. You can use this to open up (or tighten) the spacing between the lines of a problem or note. If you want the interline spacing for all problems to be 4pt. (but you do not want to make the change global because you do not like its effect on your notes or your answers, you can say
everyproblem]{
lineskip] = **4pt**}. (See the description of
everyproblem] and
everynote] in the Extras section.)

In the problem a small amount of math–surround often improves the display of the problem, but it rarely improves the display of the answers, so we do not want to change
mathsurround] globally. (Math–surround is space that TEX inserts around math mode stuff.) The

plain TEX command
mathsurround = 2pt] will set the math–surround to 2pt.'s in the problem with the change being local to the problem. (
everyproblem] can be used to make a change in every problem.)

Perhaps here is also the place to remark that
everymath] has been set to
displaystyle].

The most common spacing we wish to adjust is to set the minimum space after a problem. The commands
afterproblemskip] and
normalafterproblemskip] do this: "
afterproblemskip] = 1in" puts a minimum of one inch of space after the problem in which it is invoked;
normalafterproblemskip] = 1in puts a minimum of one inch of space after every problem beginning with the next one. (There are also a
normalafternoteskip] and an
afternoteskip].)

When there is more than one line of answers for a problem, you can control the space between lines with the
answerlineskip] and
normalanswerlineskip] commands. As usual, the first is local to the problem and the second begins with the next problem.

The commands
answerskips#1before#2after] and
normalanswerskips#1before#2after] put some space between the label and the start of the answer (the dimension #1) and it insures that there is a certain minimum space after the answer (the dimension #2).

Note: These skips are true TEX skips, but because they are deeply buried, they behave much like dimensions. An
afternoteskip]=1in. will put an inch of space after the note, but
afternoteskip]=0pt plus 1fil will do nothing because the glue is set long before the box containing the note is put into the main vertical list.

xeExtras. You have four choices of how the pages will be numbered in . These choices are selected by setting the variable
pagenumberstyle]. If this is 0 you get no page numbers at all (this can also be accomplished with nopagenumbers]). The number 1 is the default and you get a number in the upper left hand corner, except for the first page, which has no number. The number 3 is the same except that the first page also has a number. The number 2 will number all the pages at the bottom with the numbered centered; and the number 4 will do the same except that the first page has no number. These numbers should be set with a
global]

pagenumberstyle]=r, and the requested style will go into effect at the next
shipout], which, given TEX may be the page containing the command or, perhaps the previous page.

There is a token,
everyversion], which will be expanded just before beginning to process the text for each version. For example,
everyversion=]{
ifodd]
version]
relax]
global]
pagenumberstyle]=1
else]
global]
pagenumberstyle]=4
fi]} will alternate the location of the page numbers on the versions. This example also demonstrates that
version] is actually a counter, accessible to the user, which holds the version number of the version being typeset.

everyversion] is expanded just before the data needed to process the version is assembled. Hence it should be possible to alter the version number within
everyversion] and have things work out correctly. Here, for example is a scheme which skips certain versions (1 and 3) and does the rest (
ifversion] is discussed below). The specifics of
killversion] come from examining the code for : roughly, we skip over the code which produces the output for this run and then ease back into the loop which is working through all the versions: the input for
killversion] does the skip; the
fi] is to close off the
if] from
ifversion] (its usual
fi] is part of the code we skipped); and the
global] is to restore the last
global] we skipped from
killversion] but that we really want. The code is
def]
killversion]#**1**
global#2]
global]{
fi]
global]}
everyversion] ={
ifversion1&3:]
killversion]
fi]}

There are also
everyproblem] and
everynote] which are tokens expanded just before processing each problem or each note. You may change these commands after the test has begun, but then they are buried and so need to be preceded by
global] before they will have any effect. Since
everynote] and
everyproblem] are expanded at the start of a note or problem, they must be defined sometime before you want them. (If
everyproblem] is defined in the text of a problem, it will not take effect until the next problem.)

There is also a command
ifversion#1:#2]
fi]. For example, if
ifversion 2&3&5:]
afterproblemskip =1in]
fi] is placed in the text of a problem, then, in versions 2, 3 and 5, the
afterproblemskip] will be 1in; in all other versions it will be whatever
normalafterproblemskip] is. There is not an "ifnotversion" command, but if you put
ifversion 2&3&5:]
else]
afterproblemskip =1in]
fi] in your file, this will set
afterproblemskip] to 1in except in versions 2, 3 and 5. This command is intended to be used in the body of the test. The
ifversion] command is expanded as it is read, so if you put it before the first
problem] or
note] command it is only read once and hence will have little effect. If you need it executed before the first
note] or
problem], just put it in
everyversion].

Finally there is a token list
everyanswerbox]. This token is expanded just before the box containing the answers is attached to the box containing the problem. The default for this token list is
everyanswerbox] ={
ifnum]
lastversion] < 1
relax]
vskip 4pt]
else]
vskip 12pt]
fi]}. (This routine puts 4pt.'s of space between the bottom of the problem and the top of the answers in "cramped–space" runs and 12pt.'s of space otherwise.) The command immediately following
everyanswerbox] is
box] followed by the number of the box containing the answers. You could write a command,

def]
afterbox]
box#1]{ something } which could process this box:
def]
afterbox]
box#1]{
vskip 12pt]
box#1] }
everyanswerbox =]{
afterbox]} is equivalent to the usual routine except that it always puts 12pt.'s of space between the problem and the answers;
def]
afterbox]
box#1]{
vskip 12pt]
hbox to]
hsize]{
hskip 1cm]
box#1]
hss] }}
everyanswerbox =]{
afterbox]} will shift all the answer boxes 1 cm. to the right. has also stored the individual boxes with the answers which are still available to you (see the section on "Answer Typesetting" below).

The numbering of the problems is set by a token list,
problemlabel]. By default, this is
problemlabel=]{
number]
probcounter].
]
] }. These tokens precede the problem and extend into the left margin (the end of the list is the beginning of the left margin). There is also a token list,
answerlabel], which is used to label the answer sheet. The same expansion scheme applies: the right end of the list is fixed and it extends as needed to the left. The commands
problemlabel=]{{
ifnum]
probcounter]< **11**
relax] **A:**
number]
probcounter.]
else] **B:**
count0=]
probcounter]
advance]
count0] **by -10**
number]
count0.]

fi] }} will precede problems 1–10 with “A:”; problems 11 on will be numbered as “B:1.”, B:2.”,etc. If you add
let]
answerlabel =]
problemlabel] then this style will also be adopted for the answer sheet, but this is up to you.

There is also a token list,
notelabel], which puts a label before each note, extending into the left margin just like
problemlabel]. By default this token list is empty, but it can be set to anything you want.

Both the plain TEX
headline] and
footline] token lists are managed by and so are not available to the user. As indicated in the introduction, we have supplied
myheadline] and
myfootline] to replace them. If you define a headline or a footline token list using
myheadline] or
myfootline], your material replaces 's material and is centered automatically using
hfil]'s (so you can left or right justify using
hfill]'s). The command,
myfootline]={Footlines
hfill]}, will put “Footlines” at the base of every page, beginning at the left margin. It will also kill page numbers in styles 2 and 4 since the material in
footline] is now yours and not 's. At least one use for these commands is to produce other page numbering schemes. At
shipout] time, the variable
pageno] contains the page number of the page being shipped out. Hence
myfootline]={
ifodd]
version]
number]
pageno].
else]
number]
pageno]
fi]}
nopagenumbers] will produce a scheme in which page numbers are centered at the bottom of the page and have periods after them in odd numbered versions and no periods after them in the even versions.

The command
sevenrm] is available to produce the roman font at 7pts.

There is a command,
pfreadremark#1], which is a macro designed to let you put in a remark to yourself or your coauthors while you are preparing the test. Ideally you should remove these before making a production run, but if you forget, these remarks do not print in versions with a number other than 0. (It will send

a "YOU STILL HAVE A REMARK IN THE FILE" message to the log file. The "Problem ⋯" messages that you have noticed in the log file are put out after the problem is finished, so if your remark is in a problem it is in the problem below the "YOU STILL HAVE A REMARK IN THE FILE" message.) Be warned that
pfreadnote] is a macro, so your text needs to be inclosed in braces.

There are two commands for producing fractions. The command
frac]#1#2 sets a fraction with #1 in the numerator and #2 in the denominator. It is set with the numerator and denominator in plain TEX's
textstyle], which is bigger than the usual default in plain TEX for setting fractions. There is also a smallfrac]#1#2 which also sets a fraction but this time using the standard plain TEX conventions.

Two commands which are of use to Textures users are the
picture#1 by#2(#3)] and
scaledpicture#1 by#2(#3 scaled #4)] commands. An explanation of these can be found in the Textures manual, but briefly #3 is the name of a picture in the Textures "pictures" window, #1 and #2 are the two dimensions listed in the window of the picture you want. The #4 in the
scaledpicture] is a TEX scale factor (e.g. 500, 1000,1200,etc.) The result of either of these commands is a picture contained in a box which just encloses it. Hence
raise 1in]
hbox]{
picture#1 by#2(#3)]} will raise the picture up an inch.

xfAnswer Typesetting. Normally you should not need to know how typesets your answers, but occassionally you may want to achieve some special effect. This section explains how preforms its job and how you may interact with it.

Each answer is boxed up as it is read. At the end of the problem, locates the correct permutation to use, adds labels to the answers and puts the resulting boxes into box registers
bba],
bbb],
bbc],
bbd] and
bbe]. The answer with label (a) is in box
bba], etc. The answers, but without the labels, are stored in box registers
ba],
bb],
bc],
bd] and
be]. There is a variable,
correctcounter], which hold the position of the correct answer (0 = (a), 1 = (b), etc.).

Then determines how to place these boxes. There are 7 arrangements which may use. An imposed requirement is that the number of answers from one line of answers to the next is never increasing.

We may be able to place all our answers on one line: the command which does this is called
fivezero]. Or we could place four on one line and one on the next if needed: this command is called

fourone]. (
fourone] has a built in predilection for setting the answers as three on one line and two on the next if this is also possible. There is a TEX
if] command,
iftruefourone] which does this. The default is
truefouronefalse] but if you say
global]
truefouronetrue], thereafter, will set all your "four-one"'s as four answers on the first line and one on the second.)

There is a command,
threetwo], which sets three answers on one line and tries to set two on the next: if called when this is not possible, it defaults to
threeone], which sets three answers on one line and one on each of the next two lines.

The is a TEX
if] command,
ifthreetwoB], available to the user. There are two ways we could get three answers on line one and two answers on line two. By preference, will try to begin the second answer, (f), on line two under the second answer, (b), on line one (case A): if it can not do this because answer (d) is too long, will try to begin answer (f) under the third answer, (c), on line one (case B). Some people do not like this second arrangement and would prefer that the answers appeared on three lines with one answer each on the last two lines rather than case B of "three-two". The command
threetwoBfalse] will do this. By default, we have
threetwoBtrue], and with a
global] command this can be turned on and off during the run to force some answers as in "three-two-case B" and others to set as "three-one-one".

There are commands,
twotwo], which tries to set two lines of two answers each and a third line of one, which defaults to twoone] when it can not do this:
twoone] sets two answers on line one and one answer per line afterwards.

Finally,
onezero] just sets each answer on a line by itself.

These routines are semi–available to the user. First note that it is the users responsibility to see to it that the requisite number of answers will fit on the first line (else you will get an "overfull hbox" message). After the first line, the routines manage the remaining ones. By saying
global]
everyanswerbox =]{
vskip 14pt]
answerbox] }
gdef]
answerbox]
box#1]{
threeone]

box#1]}, all problems after this command will have their answers set in "three-one-one" format. This will surely not work well in general, but by changing
everyanswerbox] before a problem ends and then restoring it to its old value in the next problem, one can force the answers of a particular problem to be set in "three-one-one" style when might naturally set it in "three-two" style. To facilitate such changes, we have defined a
normaleveryanswerbox] which is the same as default
everyanswerbox]: hence
global]
everyanswerbox]=
normaleveryanswerbox] will restore
everyanswerbox] to its default value (unless the user has also redefined
normaleveryanswerbox]).

There is a routine,
handbreak#1], which will set a problem: if you wish every version of a particular problem set as, say
threetwo], just put
handbreak]
threetwo] into that problem and it will happen. If you only want this for certain versions, put in, for example,
handbreak]{
ifversion] **1&3:**
relax]
threetwo]
fi]}. In general, the variable, **#1** from above, should be one of the line setting routines discussed above.

§A.1 Answer File format.

If there is a
noanswersheet] command, will generate a data file called
jobname.answer].

The first entry in this file is **Version**, followed immediately by a number, usually 1 and a
par]. Then comes a list of entries, seperated by
par]'s, until either the file ends or we encounter another **Version** followed immediately by the next version number.

These additional entries are of one of five types.

1.As a note is put into the main vertical list, it adds "n.x" to this file where x is the number of this note in the version of the test being printed.

2.As a problem with no multiple–choice answers is put into the main vertical list, it adds a "P.x" line to the file, where x is the number of this problem in the version of the test being printed.

3.As a multiple–choice problem is put into the main vertical list, it adds an "$x.y.z$" to the file, where x is the problem number in the test being printed; y is a number in the range 0 to 4 indicating which answer is correct (0 is (a)) and z is the number of multiple choice answers for this problem.

4.A true-false problem puts a "t.x" line into the file, where x is 0 if TRUE is the correct answer and 1 if FALSE is the correct answer.

5.The fifth type of entry is generated using TEX's
mark] mechanism. When an item of type 1–4 is added to the file, also emits a
mark] with the same data. Then, as a page is shipped out, TEX remembers the entry for the last mark data on the page, and adds an "M." followed by the last mark data to the file. (If the last item on the page is note x for example, there is an "n.x" in the file and the file will also have an "M.n.x" in it.) As usual with TEX, the page break is often not found until we have put an item or two too many into the main vertical list, so the "M.n.x" of our example probably comes several entries after the "n.x" to which it corresponds.

The mark data is not emitted until after the
everyanswerbox] is expanded, so your
everyanswerbox] routine can manage the mark data for items of type 3 and 4 as well.

If is making the answer sheets, this same data is generated, it just isn't saved. It is important that, if is to generate the answer sheets, the mark data be correct and be what expects. (If you are going to process the
jobname.answer] file, you can arrange the mark data to suit yourself, at least for items of type 3 and 4.) Most uses of
everyanswerbox] do not alter the problem number or the number of answers or even which answer is correct, so usually the mark data is correct without any work on your part. If you need it, the problem number is
probcounter], the correct answer is
correctcounter] and the number of answers is
numberps].

§A.2 Some technical data. The routines and variables listed above (and in the list below) represent the commands and variables to which the user has access. THIS IS NOT YET CORRECT (with some effort you can still clobber variables you shouldn't)! There are however a great many other routines and variables in . They have been protected from the casual user by including a '?' as part of their name after first declaring '?' to have category code 11 at the start of . Of course we redeclare the category code of '?' to be 12 at the end of the macros for , so, under normal circumstances, the user may write and use other macros freely with .

also plays games with the category code of the letter w. needs to read the file once for each version of the test desired and the material before the first note or problem should not be read twice just in case the user has defined some macro which will not do well when read a second time. It is easy enough to skip over material at the start of the test the second time the file is
input] UNLESS some of it has been declared
outer]. The most common situation in which this occurs in TEX is with the various

new] commands. Hence, by changing the category code of w whenever we start to re–read the file, glides harmlessly past some
ne] commands until it hits a
note] or
problem] command, at which point the category code of w is restored. We have also redeclared the plain TEX alignment command,
+] (which is
outer]), to be
tabalign]. Futhermore, will also pass harmlessly over your
outer] macros if they contain a w, but otherwise it will halt with an error message, your macro will be turned into mush, and the whole project should probably be abandoned. (If you have some macros which should be read every time a new version starts up, you can include them in the
everyversion] token list.)

The advantages of reading the file each time far outweigh these minor annoyances. It means that we need only a minimum amount of storage so rarely has memory problems. The worst case from the point of view of memory usage occurs when we TEX a version in which we permute the problems. For this case we have adopted a compromise strategy: reads the file only once, but only stores those problems and notes that it is going to need to assemble the test. Furthermore, only reads the file until it has found all the needed material, after which, it stops reading. could have TEX'ed much larger tests by reading the file to find a needed item and then reading it again to find the next one, but this seemed to be sufficiently slow that it was not implemented. (It also seems rare that one wants a hundred question test with the problems permuted, and if you do then perhaps is not the tool to generate the intitial file. Once you have the file with the problems and notes in the order you wish, will TEX the file, essentially regardless of its size. Of course you had better shut off the answer sheet macros and produce the sheets yourself or you will overflow TEX's memory.) A practical limit to the size test will accommodate is about 50 problems. (After this, the answer sheet macors cannot fit the stuff on one page so you will certainly have to make your own answer sheets. If you select more than 50 problems, memory begins to get a little tight: the author has TEX'ed 50 a problem test with
tracingstats=2] and observed about 6000 words of memory were still untouched. With 55 problems, we had 499 word of memory still untouched. Your statistics will vary depending on how complicated your problems and answers are.)

Pseudo–Index

everymath]

everynote]

everyproblem]

everyversion]

firstversion]

fivezero]

fourone]

frac#1#2]

header]

ifthreetwoB]

ifturefourone]

ifversion#1:#2]

labelborder]

labelstyle]

lastversion]

multiplechoiceskip]

myfootline]

myheadline]

mymarkedanswersheet]

mypageannouncement#1]

myunmarkedanswersheet]

noanswersheet]

noprofessor]

normalafternoteskip]

normalafterproblemskip]

normalanswerlineskip]

normalanswerskips#1before#2after]

nosection]

note]

noteline]

notelineskip]

numberps]

onezero]

pageannouncements=#1]

pagebreak]

pagecounter]

pageno]

parenthborder#1]

perm#1:]

pfreadnote#1]

placeheaderbox]

placemultchoicebox]

placetotlasbox]

picture#1 by#2(#3)]

probcounter]

problem]

problemlabel]

problemnameskip]

Professor#1]

scaledpicture#1by#2(#3scaled#4)]

setglobalperms#1]

setlabels#1#2#3#4#5]

setpermlist#1]

setproblempermutations#1]

sevenrm]

skipforfootlineofanswersheet]

smallfrac#1#2]

Tf]

tF]

threeone]

threetwo]

title#1]

totals]

twocolumn#1]

twolinetitle#1#2]

twoone]

twotwo]

version]

writeanswersheet#1]

wrong]