<TEAM-1, USER:TUTOR, TUTOR3.AUG.1,>, 8-Jul-88 18:18-PDT WBD ;;;;

Introduction

Welcome to TUTOR3! Now that you know a little more about the way files are structured in AUGMENT, we will take advantage of that knowledge in this new file. It is actually a structured file, and by turning on appropriate viewspecs, you can get a view of the table of contents of the file without having to read through the whole file. By now, you have enough basic editing skills to do most things, but you may be interested in learning more about a few specific topics. When you finish reading this paragraph, you should take a look at the contents of the file with a one-line, one-level view to see what is coming. Here's another shortcut: Very often you will want to see one line and one level or all lines and all levels. Viewspec "x" is one line, one level (like setting viewspecs d and t together), and viewspec "w" is all lines, all levels (like setting viewspecs c and s together). Turn on viewspec "x" (using the command "Set Viewspecs x<OK>") to view the main headings, and then turn on viewspec "w" and continue.

This is the third of the tutor files, and by the time you have completed it, you will have covered enough of the basic AUGMENT features to do most of what you will want to do. The files TUTOR, TUTOR2, and TUTOR3 do not come anywhere near covering all of the features of AUGMENT. In a sense, they just begin to scratch the surface. Some more advanced topics can be found in the file <TUTOR, TUTOR4,> which is similar to the first three, but covers topics which may not be of interest to everyone. When you have finished reading this file, you may jump to that file and read it, or you may copy it into your own directory if you wish to edit it.

AUGMENT is an enormous system. It is nearly impossible to give a complete description of one feature without talking about other features. In order to avoid rewriting large portions of the text, we will often use links to make references to other places in the file. For example, to find out how to copy the TUTOR4 file into your directory, see <0185>. To "follow" the link, simply type "Jump (to) Link", and mark anywhere between the "<" and ">" in the last sentence. A final <OK> will put you there. A "Jump (to) Return" command (see <0139>) will get you back here.

Finally, there is more material in this file than in the other two tutor files. In addition, it is slightly more concentrated in places, and you will find that you need to read, reread, and experiment before you understand the process completely. Take it easy. Do not try to read this whole file at once. Work on it a little at a time, and when you are finished, you will truly be an advanced AUGMENT user. Good luck!

(This last statement holds, and even more so, for the TUTOR4 file. Users of that file will find much of the material there very concentrated.)

More Jump commands

In this section, you will learn a few more Jump commands. The exercises almost invariably involve jumping around in this file, and then returning when you are done. To save time, it might be a good idea to turn on the statement numbers (use the "Set Viewspecs m<OK>" command), and before you go "exploring", note the number of the present statement so that you can get back with a "Jump (to) Link" command.

So far, our jumping commands include:

Jump (to) MARK -- The statement you mark will be put at the top of the screen.

Jump (to) Origin -- The origin will be put at the top of the screen.

Jump (to) Link -- This command will go almost anywhere, but you must know the statement number to which you wish to jump. A link can also include a file name and viewspecs, in the following general form:

link = filename,statement-number:viewspecs -- A sample Jump (to) Link command might be: Jump (to) Link tutor,3:my<OK>. This will take you to the file named TUTOR, to the third top-level statement, and when you arrive, the viewspecs in force will be m and y. Not all parts of a link are necessary. The following are also legal links:

23a (statement number 23a in the current file)

tutor,:c (beginning of the file named TUTOR, with viewspec "c")

:d (stay in the same place, but change to viewspec "d")

3:a (statement 3 in the current file, with viewspec "a")

Using statement numbers, it is possible to get around in a file pretty well, but there are a number of shortcuts. For example, suppose you are reading a file, and you come to a heading in which you have no interest. Presumably, if your file is well structured, all the statements at levels below (down from) the heading have to do with the heading, so you would like to skip them too, and get to the next main heading. If your statement numbers are on, you can tell by the number of the heading what the number of the next statement at the same level as the heading will be. For example, suppose that the heading has statement number 3c. What will the statement number of the next heading be? The answer is: 3d (if there is a next heading at that level). If there are no more statements at that level, the next information that might be of interest would be under statement 4.

So to get to the next main heading, you could look at the statement

numbers (turning them on first if necessary), figure out the number of the next statement at the same level, and then use the Jump (to) Link command. Alternatively, you can refer to such a statement as a "successor". A successor of a statement is the next statement in a file at the same level as the given statement. Put this statement at the top of your screen, and then answer the following questions about the menu below. What is the successor of BREAKFAST? of EGGS? of SANDWICH? of DINNER? A predecessor is the opposite of a successor; what is the predecessor of LUNCH? of FRIED? of EGGS?

```
BREAKFAST

  EGGS

    SCRAMBLED

    POACHED

    FRIED

  COFFEE

LUNCH

  SANDWICH

  MILK

DINNER
```

The successor of BREAKFAST is LUNCH, and of EGGS is COFFEE, while DINNER has no successor. The predecessor of LUNCH is BREAKFAST, and of FRIED is POACHED; EGGS has no predecessor. If you had trouble with this, reread the section, and try again.

The words Successor and Predecessor are just two more nouns that can be acted upon by the verb Jump. You can jump to Predecessor and to Successor.

After you type "Jump (to) Successor" and mark a statement, the successor of the statement you marked will be moved to the top of your screen. This is exactly the same as the other Jump commands. The statement referred to is the statement that will appear at the top of the screen at the end of the jump.

Thus, if statement 1 is at the top of your screen, and statement 2 appears as well, and you type Jump (to) Predecessor and mark statement 2, nothing will change because the predecessor of statement 2 (namely statement 1) is already at the top of your screen. It is the same kind of thing which would happen if you used the Jump (to) MARK command and marked the top statement on your screen.

Turn on your statement numbers, note where you are, and try using the Jump (to) Predecessor and Jump (to) Successor commands. Use them on statements that have no predecessors or successors to see what happens. When you are done, return here.

There are four other nouns similar to Predecessor and Successor which are commonly used. They come in two pairs of "opposites" -- Up and Down, and Next and Back. Up and Down are easy to understand. The Up of a statement is the statement that the given statement is under (where "under" means "a level below"). The Down is the first statement under the given statement. Next and Back are simply the very next statement in the file (at any level), and Back is the opposite -- the previous statement (again at any level).

Jump to this statement, and answer the following questions about the statements below. What is the Up of WYOMING? of HOUSTON? What is the Down of CANADA? of CODY? What is the Next of HOUSTON? of CANADA? What is the Back of VANCOUVER? of TEXAS? What is the Successor of the Up of CODY? What is the Next of the Back of WYOMING? What is the Successor of the Successor of the Down of UNITED STATES?

```
UNITED STATES
   ARKANSAS
   CALIFORNIA
     SAN FRANCISCO
   WYOMING
     CODY
   TEXAS
     AUSTIN
     HOUSTON
   VERMONT
UNITED STATES
   BRITISH COLUMBIA
     VANCOUVER
   ONTARIO
```

The Up of WYOMING is UNITED STATES; the Up of HOUSTON is TEXAS. The Down of CANADA is BRITISH COLUMBIA, and CODY has no Down. The Back of VANCOUVER is BRITISH COLUMBIA, and the Back of TEXAS is CODY.

The Successor of the Up of CODY is TEXAS, and the Next of the Back of WYOMING is still WYOMING. The Successor of the Successor of the Down of UNITED STATES is WYOMING.

Try jumping around in this file to try out these new commands (Jump (to) Next, Jump (to) Up, and so on). Return here when you are satisfied with your understanding of how they work.

A little thought will show you that Down and Next are going to be less useful than Up and Back. When you Jump Down from a statement, it is the same as Next, unless the statement has no Down. Jumping Up can take you a long way back in the file. The reason you might want to Jump Up is to find out the heading of the section you are reading. Similarly, Back can move you one statement back in the file, which can be very useful if you have jumped one statement too far. Next is only useful when you can see that a statement has ended at the bottom of your screen, which it is often difficult to tell. You are usually safer to jump to MARK and mark the bottom statement on the screen. This reshows some of the bottom statement on the top of the screen, but it assures you that you did not miss reading any text.

Two other Jump commands can be extremely useful from time to time. Every time you jump in a file, AUGMENT records exactly where you were, together with the viewspecs that were in effect at the time. A list of ten previous views is kept, and it is possible to return to any of these views. Thus, you can be momentarily distracted and Jump off somewhere without noting where you were, and return back to your previous position. The command to do this is Jump (to) Return <OK>. After you type this, you will be presented with a "flashback" of the first few characters from the top of your previous views, to which you answer "y" (or <OK>) for "yes", or "n" for "no". As soon as you answer "yes", you will be returned to that view. Don't worry about missing the one you want -- if you accidentally type "n" to it, the views will cycle around and give you a second chance. Try using the Jump (to) Return command now to see how it works.

The other useful command is similar, except that it refers to jumping back to other files you have viewed during a session. The mechanism is exactly the same, but you use the command Jump (to) Return File. Try jumping to somewhere in the <TUTOR,> file, and return here using the Jump (to) Return File command.

Creating and deleting files

Everything that you have done so far has been in files that already existed. In this section, you will learn how to create new files and how to delete old ones. In addition, this section will discuss some elementary facts about file control, that is, how to keep your online information organized and safe from accidental loss.

Creating a new file

The command to create a new file is simply "Create File". You will need to type a leading space because typing simply "c" gives you the Copy command, which will be discussed later. After you have typed "<SP>crf", you will be prompted for the name for the new file. If you type this, followed by <OK>, a new file by that name will be created, and you will be put in that file. Do not use spaces, periods, or semicolons in the filename as they serve another purpose (but dashes are fine). Use this command to create a new file named TEST, and after you are done, you will be in the newly created TEST file. Take a close look at the origin statement there, and then return here using the Jump (to) Return File command. Create the file now.

The origin statement of the file you just created should look something like this:

<TEAM-1, USER:DAVIS, TEST.AUG;1,>, 20-Jan-88 17:05 WBD ;;;;

The origin statement contains a lot of information about the file, and it is useful to be able to read it. The first word in your origin is the computer name. In the example above, the computer is TEAM-1. The second word is the logical unit name. The third word is the directory name. In the example, the origin statement was created on the USER logical unit in the DAVIS directory.

Following the directory name is the actual name of the file. The one you created is named TEST. Then there is a period, followed by "AUG". This is called a file extension, and identifies the type of file. In AUGMENT, almost all of the files you will deal with have the "AUG" extension (older files may have an "NLS" extension). Don't worry about it now. Following the extension is a semicolon, and following that is a number, which is the version number of the file. Since the file is newly created, its version number will naturally be 1. The update command will be discussed later, but for now, each time you update your file, the version number will increase by 1. Note how periods, commas, and semi-colons are used for filing purposes which is why they are not used for the actual filename.

All of the above information is contained within angle brackets, and outside those angle brackets is the date and time that the file was last updated. In the case of a newly created file, it is simply the date and time of creation. After the date and time is the ident of the person who last updated the file. Your ident should appear there. The four semicolons at the end of the origin statement mark the end of the information.

Updating a file

AUGMENT has a mechanism for protecting you from losing large amounts of work due to mistakes. It turns out that when you are editing a file, you are not really altering the actual file; your changes are simply kept as a set of modifications to the file. When you look at

the file on the screen, however, you see the file as though the modifications had really been made. A possible way to think of it is as if you put a clear plastic transparency over the real file, and marked your corrections on it. This way, if you make a bad mistake that destroys a large part of your file, you can throw away the plastic transparency, (delete the modifications), and you will be back where you started.

In an earlier lesson, we learned to Delete Modifications. This is how you get rid of all the corrections, or throw away the plastic transparency. Updating a file does just the opposite -- it merges the changes with your real file, and makes them permanent. The process of updating also increases the version number (which was mentioned in <0228> above) by 1.

A typical way to work is to update a file that you are working on every so often. In this way, the most work that you can lose is up to the last time you updated. Also, if you are experimenting with a new command that you do not completely understand, it is a good idea to update the file, and then try the command. If the command has unexpected consequences, you can simply delete the modifications and try again.

The command to update a file is "Update (file) New <OK>". (A short cut is to type the "Update" command followed by two <OK>s). Be sure to do this regularly to files that you use a lot. (There are actually four different kinds of update. "Update New" is the most common and useful, but see the TUTOR4 file if you are interested in the other kinds.)

Deleting a file

The command to delete a file is simply "Delete File". You will then be asked for the name of the file to be deleted, and three <OK>s will perform the deletion. (The extra <OK>s are for insurance against error.) Try deleting the TEST file that you just created.

When you delete a file, it is not really gone forever -- it is simply put on a list of deleted files. You can undelete a file with the "Undelete File" command. You then type the name of the file you deleted, and type <OK>. Undelete the TEST file now.

If you had really wanted to get rid of the TEST file forever, after you deleted it, you could have used the Expunge command. It is "Expunge (deleted files in) Directory <OK>". After you do this, all deleted files will be gone forever.

To make sure that you understand the material in this section, Jump to the TEST file (it should now be undeleted), Insert a few statements in it, and then Update it, Delete it, and Expunge it.

Some good advice

Update your files regularly, so you will not be able to make serious mistakes.

Give your files names that will be easily recognized later. Do not give them names like FILE1, FILE2, FILE3, etc.

Fairly early in your AUGMENT career, it is a good idea to learn as much as you can about file control. This is discussed in this file and in the first section of the TUTOR4 file. Some of the material there repeats what you have just learned, but most is new, and some of it is not easy. The better you understand the AUGMENT file mechanism, however, the easier it will be for you to do your work.

Finding out what files you have

Later on in this file (under the section on getting help, <0220>), you will learn how to see a list of all the files in your (or in any) directory. If you feel adventuresome, try to figure out how it works now. The command is "Show Directory".

Working with structure

So far, we have learned to move, insert, delete, and replace single statements, and to put them at different levels in a file. Whenever the level-adjustment prompt "L" comes up in the command window, you may type "d", "u", "uu", etc. to indicate that the new statement is to be inserted down a level, up a level, up two levels, etc., from the statement it is to follow. Note that it makes no sense to have a statement two levels down from a given statement with nothing in between -- what would the new statement be subordinate to?

For this reason, it is impossible to delete a statement that has a statement under it. Try it, if you haven't already noticed the problem in some of your earlier exercises. If you really want to delete a statement that has substructure, you must either delete the substructure as well, or move the substructure somewhere else so that it will be subordinate to something when the statement that was above it is deleted.

It is often extremely convenient to be able to talk about larger chunks of statements than just one. For example, if you had a book typed in, and if it were well structured, each chapter title would probably be a separate first-level statement; each section heading would be a second-level statement, and so on. If you were editing the book, it is easy to conceive of wanting to move around the chapters into a different order, to move around the sections of a chapter, etc. There are names for such things, and that is what this section of the lesson is about.

The branch

The following analogy is a useful way to think about the structure of an AUGMENT file. Think of a whole file as being a tree with the

origin statement as the trunk. Think of each of the main statements (1, 2, 3, and so on) as main branches off the trunk. Connected to each main branch may be smaller "subbranches" (1a, 1b, 1c, etc. are the main subbranches connected to the larger branch numbered 1). Similarly, 1b1, 1b2, etc. are the largest branches connected to the branch 1b, and so on.

You can get to any twig on the tree simply by starting at the main trunk, picking a main branch from the trunk, then picking a main subbranch of this branch, and so on, and eventually arrive at any twig. In an AUGMENT file, the same idea holds. Statement 1c5 is the fifth statement under statement 1c; statement 1c is the third statement under statement 1, and statement 1 is the first statement under the origin. If this section has seemed difficult, please reread it, because once you understand it, most of the rest of the discussion will be simple.

In AUGMENT, we define a "branch" to be a statement plus all of the statements below it. It is like an entire branch of a tree, including everything connected to it all the way out to the tiniest twigs. When we say "Delete Branch (at) 1", it is like cutting off main branch number 1 from the file "tree" (and, of course, everything connected to it).

We do not need to cut off such enormous branches, however. If we delete the branch at 1c3, it means to find the first main branch, follow it out to the third main subbranch (1a and 1b are the first two), and then follow that branch out to the third branch connected to it, and to cut it off there. Statement 1c3 will disappear, together with anything connected to it (1c3a, 1c3b, 1c3b1f, etc.)

To identify a branch, all we need to do is identify the main statement to which everything else is connected. Thus, when you say "Delete Branch (at)" and you mark a statement, the statements that will be deleted are the statement you marked and all of the statements below it.

As an example, consider the short menu below. The branch at MENU is the whole menu (nine statements); the branch at CEREAL includes three statements: CEREAL, WITH CREAM, and WITH MILK. The branch at COFFEE consists of only one statement, itself. How many statements are in the branch at DINNER? To check your answer, try using the Delete Branch command, and mark the statement "DINNER" below. Then try the same command on other parts of the menu until it is all deleted. (Please point only to statements in the menu, as Delete Branch can delete large portions of a file.)

```
MENU

   BREAKFAST

      CEREAL
```

```
        WITH CREAM
        WITH MILK
      COFFEE
    LUNCH
    DINNER
      STEAK
```

The word Branch is simply another noun like Statement, Word, Character, and so on. Many of the editing verbs you already know can be applied to it, such as Delete, Move, and Replace.

When you first begin to work with larger collections of AUGMENT entities, you will probably be surprised from time to time by the actions of various commands. The simplest is probably the following. Consider the statements below, and suppose that you wish to convert this set of statements:

```
  ABCD
    EFGH
  IJKL
```

to this:

```
  ABCD
  IJKL
    EFGH
```

Your first impulse might be to use the Move Statement command, and to mark IJKL to follow ABCD at the same level. Try it and see what happens.

Nothing appears to happen. Can you think of why? The reason is that in some sense, the statement EFGH "belongs" to statement ABCD, not to IJKL. If ABCD were a chapter title, EFGH would be the contents of that chapter. Inserting a new chapter title should not move the contents of the other chapter to be within it. When you tell the machine that the new statement is to follow an old chapter head, and to be at the same level, it assumes that it, too, will be a chapter title. It is therefore put where the next chapter title belongs. You can, however, move the EFGH to follow the IJKL, using the level adjustment to do it. Go back to the example, and try that.

The following exercise will thoroughly test your ability to move

around statements. Turn off the statement numbers (they would only be confusing at this point), and try to arrange the following statements so that they are "in order". In other words, 1a should follow 1, and should be down a level, etc. The two statements marked "extra" should be deleted. Jump to the statement marked QUIZ below, and try to order it without retyping anything -- you are only allowed to Move and Delete. Remember that you can move Branches as well as individual statements.

```
QUIZ

  1

    1c1a

      1a1

    1b

      2c1a

        2c1

        2d

    1d

  extra

    2c

      1c1

        2

  extra

    2a

      3

        1a

          1c

            2b

  2c1b

ANSWER

  1
```

```
    1a
      1a1
    1b
    1c
      1c1
        1c1a
    1d
  2
    2a
    2b
    2c
      2c1
        2c1a
        2c1b
    2d
  3
```

This is not really much of an answer, because it does not show you how to get there, but if you did succeed, you now have a good understanding of structure. If you had trouble, read the rest of this paragraph, and go back and try to finish up. Otherwise, go on. First, you cannot delete a statement with structure under it. The statements under any "extra" statement should first be moved out (Move Branch can move a number of statements at the same time). One possible method to approach the problem might be to get statements 1, 2, and 3 in the right positions. Then concentrate on getting 1a, 1b, and so on. Eventually, everything will be put in order.

There are a few things that many people find confusing about the concept of a branch. The main thing to remember is that branches can be different sizes. Absolutely every statement in your file defines a branch of some sort -- some of those branches are large (if they have a lot of statements under them), and some are only a single statement (if they have no statements under them).

Remember also that statements under a statement in some sense belong to that statement. This allows a statement and all the statements under it (those having to do with it) to be moved, deleted, and so

on as one unit. You are also prevented from unintentionally altering the structure of a file by simply inserting new information. If you want material to be moved out from under one statement to be under another, you must specifically command the machine to do so.

The group

The "group" is another large structural entity in AUGMENT. It consists of a number of consecutive branches (remember that a branch includes all the statements below a given statement).

Probably the best way to make the concept of a group clear is by way of the example below. A group might consist of the three statements POACHED through SCRAMBLED, or just the two statements FRIED through SCRAMBLED. The group from CEREAL through BEVERAGE consists of four statements, and the group from EGGS through CEREAL has seven statements (EGGS, POACHED, FRIED, SCRAMBLED, CEREAL, WHEATIES, and GRANOLA). Remember that the group is made up of the entire branches. The large group from BREAKFAST through LUNCH consists of all the statements (11 total). Study these examples carefully, and then go on.

```
BREAKFAST

  EGGS

    POACHED

    FRIED

    SCRAMBLED

  CEREAL

    WHEATIES

    GRANOLA

  BEVERAGE

LUNCH

  NO LUNCH - YOU'RE ON A DIET!
```

Two marks will be needed to identify a group, and the statements marked must be at the same level and under a common statement. The actual group is all the statements between and including these two marked statements, and all of the statements under any of them. In our example of an online book, chapters 3, 4, and 5 would make a group. This group would consist of the chapter headings for all three of those chapters, as well as the complete contents of the chapters. Smaller groups can consist of sets of sections of a given

chapter, or of a set of one or more paragraphs in a row, all at the same level.

To continue with the tree analogy, the deletion of a group would be similar to cutting off a set of consecutive sublimbs from a given limb.

Go back to one of the menus in this file (you might try to get there using the "Jump (to) Return" command -- remember it?), and try out various of the usual editing commands on Groups -- Delete Group, Move Group, Replace Group, etc. -- and then come back here. Remember that a group is like text in that two marks are needed to identify it -- you must identify both the beginning and the end of the group.

The plex

This section can safely be skipped (by jumping to the successor of "The plex", above), but for intermediate and advanced users, the concept of a plex can be extremely powerful. It is quite easy to state what a plex is -- it is simply the largest possible group containing a given statement. It requires only one mark, and this mark identifies all of the statements at the same level as the marked statement, together with all of the substructure below them. The idea is extremely useful for large sections of a file that are too big to fit on one screen -- you could call it a group, but it is inconvenient (or impossible!) to mark both ends of it. Go back to one of the menu files, and try out some of the editing commands using the Plex noun.

As was the case with Branches and Groups, plexes can come in all different sizes. The plex gotten by pointing to statement 1 includes the entire file except the origin statement. The plex at 1b1 would consist of all statements with numbers beginning with "1b", and having at least one more number in them. Possible statement numbers belonging to the plex at 1b1 might include: 1b1c, 1b12, 1b1c12w, 1b5a, and so on.

Again, considering the analogy of an AUGMENT file with a tree, deleting a plex is the same as stripping off all the branches connected to a given branch. If you use the Delete Plex command, you point to any of those branches that are to be deleted.

Editing using addresses

Up to now, the only method we have used to point to statements for the purposes of inserting new statements or deleting old ones is the MARK. If the statement is visible on the screen, this is by far the easiest way to do it, but often the chunk of textual information you are interested in is far bigger than what would fit on a screen (imagine an entire group of chapters of a book, for example). You may have noticed that after you type "is" for "Insert Statement", the prompt is "M/A:", where "M" meant "mark". The "A" means that

you can alternatively type in an address. An address can be many things, but for now, you can think of it simply as a statement number. When you need to identify a statement and you are presented with the "M/A:" prompt, you may either mark the appropriate statement, or type the address followed by <OK>.

Jump to this statement, turn on the statement numbers, and take a look at the menu below. If you haven't made any major changes to this file, BREAKFAST should be numbered 4G2A. Try inserting a new beverage (statement) to follow 4G2A3, and do it without using a MARK. (You don't need to worry about capitalizing the "G" and "A" in the address.) Try deleting the group from EGGS to CEREAL as follows: MARK the EGGS statement, and give the address of CEREAL (4G2A2). Notice that you can mix up addresses and marks in any way. Whenever an "A" appears in a prompt, you can give the address as well as mark the statement.

```
  BREAKFAST

    EGGS

      POACHED

      SCRAMBLED

      FRIED

    CEREAL

      WHEATIES

      GRANOLA

    BEVERAGE

      COFFEE

      MILK
```

This form of identifying statements and groups of statements is extremely useful when you are editing a large file. Typically, you will have a listing of the file printed out (with the statement numbers on), and you will mark up the paper with edits. You can then move around large chunks of the file without having them on the screen. It's a good thing, too; suppose you had to move statement 1 to follow statement 100!

Getting help

Question mark

At any point in any command, except when you are typing text, you may type a question mark. Do so now, and after you take a look,

type a command delete. What you saw was a listing of all the possible command words that you could have typed at that time. Many of them were probably familiar to you, and probably quite a few of them you have never heard of. Some of the possibilities had <> in front of them -- this means that you must first type a space before beginning the command. The question mark feature is very useful if you remember that there is a command to do such and such but you don't remember the exact command word.

Now type "j" (for "Jump"), and then type a question mark. What will appear on the screen is a list of all the possible things to which you can jump. There is no need to type a command delete after typing the question mark -- you can continue with the command as if nothing had happened. Suppose that you had forgotten the name "Link". You know that you want to move somewhere in the file, so you know that the command should begin with "Jump", so you type "j". Then you type a question mark, and as soon as the word "Link" appears on the screen, you say to yourself, "Oh yes -- that's it", and type "l". The command can then be completed as if the question mark had never been typed.

The question mark is extremely useful as a quick reminder, when you already know how the command works but have forgotten the exact command word. The question mark can also be useful to you later for learning new commands. Suppose that you want to sort something, and you have heard somewhere that there is a way to do so using AUGMENT. Type a question mark, looking for likely candidates, and sure enough, there is a Sort command listed. After you type "s", the word "Sort" appears in your command window, and there is another "C:" prompt. Another question mark will show you the kinds of things you can sort. When you get experienced with AUGMENT, you can often learn enough by this method to learn to do completely new tasks.

At this point, it should be mentioned that the fastest way to learn new things is to make mistakes. The faster you can make all your mistakes, the faster you will learn. If you are experimenting with a new command, be sure to Update your file first, and then no matter what you do, you can return to the previous state using the Delete Modifications command.

Help

Question mark, as described above, can be used throughout AUGMENT, even in subsystems that you have never heard of. Unfortunately, it only gives you information about the syntax of the command -- that is, it only tells you valid command words that can be typed -- it does not tell you at all how they work.

There is a very large set of files online, called "Help files", which contain information about how all the features of AUGMENT work. These files can be accessed at any point by typing "h" for the universal Help command. Before you type it, remember that to

exit, you need only type a command delete, and you will be right back where you started. To use the command, type "h" followed by a word or phrase about which you want help. For example, if you wish to find out the details about how to sort, type "hsort<OK>". If you do not get the response you need, try typing in a synonym for the word. The writers of the Help files have tried to guess what people might ask about, but they are only human. Go ahead now, and type "hsort<OK>", take a look at what you see, and then type a command delete.

You probably got a short description of the Sort command, followed by a "menu" that was made up of two numbered statements. To get more detailed information (suppose you wished to find out about "sorttypes", for example), you would simply type "1", followed by <OK>. You also have the option of typing another word for help on it (as if you were starting the command anew). For example, suppose you read the description of Sort, and wonder what the word STRUCTURE means. Simply type "structure<OK>", and you will get help on that.

The information in the Help files can be quite terse, so you may need to reread the description a few times before you understand it, but it is probably the most complete documentation of the system available. The Help command is actually quite powerful, but you can find out how it works by asking for help about Help (type "hhelp<OK>" and continue from there).

Online documentation

In addition to the Help files, there are online versions of many of the documents about the system that are usually distributed in printed form. These include the AUGMENT textbook, short tutorials on how to use various of the specialized subsystems, and many other things including such esoteric documents as programmer's documentation on how to write your own system code. Most of the relevant information is stored in a directory named USERGUIDES.

At this point, it is probably a good idea to talk about how you can look at files in a different directory other than your own. Up to now, whenever you have typed a file name (to Jump (to) Link to), the computer has assumed that you own the file, that is, that the file is in your directory.

You are not the only person using the computer. Twenty or thirty other people may be using the machine at the same time, and when you first "logged into" the system, you told it your name, and gave it a secret password to let the machine know that you are in fact you. For the purposes of the rest of this discussion, suppose that when you logged in, you used the name SMITH. This means that you have logged into the SMITH directory. Other people may have logged into the JONES directory or the DAVIS directory. One way to think of your directory is as a filing cabinet, with the files in it being the folders in your cabinet. Inserting text into a file is like adding information to the appropriate folder.

The nice thing is that both you (SMITH) and JONES can have a file with the same name (both of you, for example, can have a file named CALENDAR or PERSONNEL or LETTER). The computer knows that your calendar is different from JONES' calendar; it knows that when you are logged in, you want to look at the SMITH calendar, and when JONES is logged in, she wants to look at the JONES calendar. It may be, however, that you may want to look at the JONES calendar. How do you do this? If you type "Jump (to) Link calendar,<OK>", you will get to your own. The solution is to type "Jump (to) Link jones,calendar,<OK>". (Note that there is still a comma after "calendar", the file name).

Of course, JONES may have set her file to be secret from you, in which case you would get some kind of error message when you try to do this, but if not, her file named CALENDAR would appear on your screen. Later, in TUTOR4, you will learn a little more about file protection, and how to protect your files from other users, or to make them so that other users can write on them.

Anyway, suppose you wanted to look at the file named LOCATOR in the USERGUIDES directory. To look at it, you could use the command "Jump (to) Link userguides,locator,<OK>"; however, there is another AUGMENT command that does this for you. You just type "Jump (to) Locator <OK>". Remember to type a space before the command word "Locator", or AUGMENT will think you mean "Jump (to) Link". Don't do this now.

There is a command to find out the names of the files in another directory. It is the Show Directory command. Since "s" will get you "Sort" type <SP>"sh" to get "Show", and then "d" for "Directory", followed by the name of the directory you wish to see. If you then type two <OK>s, a list of all the files (that you are allowed to see) appears on your screen. When the screen fills up, simply type <OK> to see another screenful.

Another shortcut -- if you want to see the files in your own directory, you do not need to type your own directory name -- simply type <OK> when you are asked for the name of the directory.

You can learn a lot by exploring the files in the USERGUIDES directory, but we recommend that you do so not by jumping to the files directly, but rather by getting to them through the file named LOCATOR, which contains pointers to much more online information. Type the "Jump (to) Locator" command and look around now if you wish. There are instructions for using it at the beginning. If you want to jump to the files directly to practice using links to files in other directories, you can do that, too. If you have trouble, you should perhaps read the section (in this file) about advanced links, and then try again. When you are tired of looking at the USERGUIDES files, use the Jump (to) Return File command to come back here.

Feedback

Any time you have a problem that you simply cannot figure out, you can send a message to Feedback (use "FEEDBACK" as the name or FEED.MDC as the ident). The message will be sent via the AUGMENT Mail system, or the Journal. Within 24 hours during the work week, you should receive a response. Feedback can also be contacted for such things as setting up new directories, deleting old ones, and other special requests. In fact, you are welcome to send almost any question at all concerning the system to Feedback, and even if you do not find out exactly what you need to know, Feedback may at least point you in the right direction.

Intermediate editing

Verbs

Copy

The Copy verb does just what you think. It copies an object to somewhere else. The objects that can be copied include all of the ones you have used before, such as Character, Word, Text, Statement, Branch, etc. As with the commands Insert, Move, and so on, the Copy command copies a character to follow a character, a word to follow a word, a statement to follow a statement, and so on.

Another important use of the Copy verb is for copying entire files. The command is "Copy File (named) oldname<OK> (to a new file to be named) newname<OK><OK>". This command does not alter the information inside the file at all, and sometimes this is a little surprising. For example, if you copy the file named "tutor,tutor4," to a new file named "tutor4," in your own directory, and then jump to that file, you will note that the origin statement still identifies that file as being in the TUTOR directory. After you have made any change to the file and updated it, the origin statement will change to indicate the new location of the file.

Transpose

The Transpose verb is another verb, which, like Copy, operates on the usual editing entities. The Transpose command exchanges two objects of the kind that you specify. For example, if you type "Transpose Character", you will be given the option of marking two different characters, and after you type the final <OK>, the two characters will be exchanged. Transposing text requires four marks -- you must identify both the beginning and end of each piece of text. To exchange chapters 1 and 2 in an online book, you might use the Transpose Branch command, and mark the two chapter headers.

Replace All

There are often times when it would be nice to be able to do a

replacement not once, but every time it occurred in a certain piece of text. I am sure that you have seen form letters that begin "Dear Capt. Ahab,", and then later have a line containing something like: "I am sure, Capt. Ahab, that you would love to be a benefactor to our 'Save The Whales' organization ...". This could have been done by making a letter template with "XXX" wherever "Capt. Ahab" appears above, and the same letter can be customized by substituting other names for the XXX in subsequent mailings.

In any case, there is an AUGMENT command to do just this -- the "Replace All" command. The wholesale replacement may take place in a single statement only, a branch, a group, or even a plex or the entire file. The Replace All command has many forms, but in its simplest form, it works something like this:

"Replace All (occurrances in) Branch (at) MARK (of old) Word oldword<OK> (by word) newword<OK> <OK>". In this example, all occurrances of the word oldword will be replaced by newword. In this paragraph (statement), notice that "occurrences" has been misspelled throughout (as "occurrances"). Correct all of the misspellings now with a single Replace All command.

In the previous example, we replaced words by words; we can also replace for arbitrary text. There are some dangers, however. If we substitute the text "dog" for "cat", all the words "cat" will be changed to "dog", but the word "catalog" will be changed to "dogalog". Let the user beware. If you are not sure about exactly what a Replace All command will do, update your file first, then try it. If all goes well, fine. If not, (you get 2000 replacements instead of the 20 you expected), you can just Delete Modifications, and try again.

Try a few strange replacements on a small part of this file. "Replace All" for characters, words, and text. Try to figure out what will happen each time before you press the final <OK>. You might want to update the file first, as was mentioned earlier, so that you can delete modifications when you are done.

Notice that at the end of the command, you were presented with a "C/OK:" prompt, and until now you have answered <OK>. Your other options include typing "a" (for additional replacements in the same structure), and "s" (to show the status of the current command). Try doing more than one global replacement at the same time, and before you give the final <OK>, type "s" (for "Status") to see how the command works.

Append and Break

The basic structural entity of an AUGMENT file is the statement, and it very often corresponds to a paragraph of text. Sometimes, after you have written a paragraph, you notice that the text you have written really deals with two different ideas, and should be

broken apart into two different statements. Sometimes the opposite is the case -- two paragraphs should be joined to make one.

AUGMENT has two commands to perform these operations: Break and Append, respectively. Let us try the Break command on the following paragraph (which obviously discusses two different ideas).

For millions of years, dinosaurs roamed the earth. One of the largest was Tyrannosaurus Rex. To use the break statement command, type "bs" (for "Break Statement"), and MARK the last word you want to remain in the first of the two parts. Actually, "last word" is not precisely correct -- all characters up to a space or <TAB> or <RET> will remain. In the case of this paragraph, marking anywhere in the word "Rex" above will cause "Rex." (including the period) to remain in the first statement. After marking the breaking point, you may simply type <OK> to complete the action, or you may specify the level of the newly-created second statement relative to the present one (up, down, same level). Do this now.

Now let us learn to do the opposite -- append two statements together. First break this statement at "together" a few words ago, so that there will be a pair of statements to append. You may want to turn on numbers or blank lines to see exactly what happens. The command is "Append Statement" (typed "as"), followed by marking first the statement you wish to append to the end of the statement that you mark second. In other words, if your two statements were XXX and YYY, and you give the command "Append Statement (mark Y) (to) (mark X)", you will get XXXYYY. Presumably, you don't want the "F" of "First break ..." to be crammed against the "." in "together."; you would like to have a couple of spaces between them. Note that the noise words "join with" appear. At this point, you can type the two spaces you want to join the statements with.

If, for some reason, you wish to have nothing at all inserted between the two statements, simply type <OK>. If, for some reason, you wish to MARK the text that is to be inserted between the two statements, you may type the <OPT> key (<CTRL-U> or the F6 key on the IBM-PC) first, and then mark the appropriate text. Look at the prompt -- it tells you to do exactly that.

There is another form of the Append command that is often quite useful, especially when dealing with the conversion of files from other systems to AUGMENT files (this topic is covered in TUTOR4). It is the Append All command. It allows you to append all of the statements in some structure (Branch, Group, or Plex) to form a single large statement.

All of the statements in the structure are appended, including the substructure (if there is any). At the end of the Append All

command, you have the same option for specifying the text to go between the appended statements. Whatever you specify is inserted between all the statements.

A statement in an AUGMENT file can be at most 2000 characters long, and if you specify a large structure in the Append All command, AUGMENT will not be able to append all of them into a single statement. What actually happens in this case is that as many statements as possible are appended (without going over the 2000 character limit), and then a new statement is started.

Force

The Force command is used to change the capitalization (uppercase, lowercase, etc.) of text. It only affects alphabetic characters (not numbers, punctuation, etc.) You can force everything from a single character to an entire branch or plex. For example, suppose that you want to change this word: word to be all uppercase. Use the command "Force Word (at)", and mark the word (go ahead and do it), then give a final <OK>, and the word will be all capital letters.

Besides uppercase, there are two other "force modes" available -- lowercase and first letter upper. At the end of a Force command, such as "Force Word (at)", you can simply type <OK> to get uppercase, or you can type "l" for lowercase or "f" for first letter upper. What you will see is "Force Word (at) MARK Lower <OK>. Try both of these now.

Sometimes you find that you need to change the capitalization of a lot of words, in different places in your text, all to lowercase or first letter upper. In this case, you can change your default force mode to one of these others. After you set your mode to, say, first letter upper, then Force will force to first letter upper every time if you don't specify a mode in the Force command. If you then need to force to uppercase, you can just type "u" for uppercase. Alternatively, you can reset the default force mode back to uppercase.

The command to change your default force mode is "Set Force (mode to) Upper<OK>", "Set Force (mode to) Lower<OK>", and so on.

Content searching

There are many occasions when someone wants to find some text in a file, but has no idea where it is. "I wrote a paragraph about Mary's little lamb somewhere, but I don't know where to find it" is a typical problem. If you know that the statement in question contained the word "Mary", you could proceed as follows: Type the command "Jump (to) Content First Mary<OK>". The capitalization must be exact -- a search for "mary" or "MARY" would fail. If you try this command now, all you will accomplish is to put this statement

at the top of your screen (if it isn't there already) since it is the first in this file to contain the text "Mary".

The "First" in the "Jump Content First" says to look for the first occurrence of the specified text in the file. To do a search from where you are for some text, use the command "Jump (to) Content Next Mary<OK>" If the named context can be found later in the file, you will be moved to the first statement containing it.

You will find that quite often, you will be interested in looking at all the occurrences of some content. Use "Jump Content First to get to the first one, and then "Jump Content Next" to look at successive ones. A shortcut is the following: The <TAB> key on your terminal is equivalent to "Jump Content Next", using whatever content you specified last. Just press the <TAB> key when your prompt is "BASE C:". Now note where you are, and try some content searches for various things now. Return here when you are done. Be sure to try <TAB>.

The only restriction on content searches is that the character string searched for may not contain the double quote character ("). The capitalization must be exact as well.

There is a second method of content searching which will look for the specified text as a word only. The commands are "Jump Word First" and "Jump Word Next" Otherwise, they are identical to the commands discussed above; you can use the <TAB> key to repeat this sort of search also. The difference is that a "Jump Content Next the<OK>" might find those characters as the the word "the", or as part of the words "then", therefore", or "Athens", the "Jump Word ..." commands must find the content "the" as a word.

For experts, there is a way to specify content in a link. See TUTOR4 for details.

SIDs

Using statement numbers, it is possible to get to any statement in a file in a single step -- if you know the statement numbers. You may have a printed listing of the file with the statement numbers listed, but there is still a problem. As soon as you insert or delete a statement from your file, the number of every statement following it may change. This could make life difficult.

In fact, every statement in your files has two numbers -- one is the standard statement number that was discussed previously (such as 12, 1b5, 17q8w4, etc.), and another (called a statement identifier, or SID) which will never change. Every time a new statement is inserted into a file, it gets the next SID, starting with number 1. In a very old file there may be statements having SIDs in the tens of thousands.

The problem is, how do we distinguish between statement numbers and

SIDs? Does "3" refer to statement number 3 in the file, or to the third statement ever entered in the file? The answer is SIDs begin with a leading 0 (zero). Thus, "Jump (to) Link 03<OK>" will take you to the statement with SID 03, and "Jump (to) Link 3<OK>" will take you to statement number 3 (the third top-level statement), no matter what its SID is. Since statements tend to be added to files in a rather random fashion, there is very seldom any relationship between a statement's SID and its position in the file. (Note that a statement's statement number tells essentially everything about its position in the file.)

The only question remaining is, how can you find out what the SID of a statement is? It turns out that turning on SIDs is just another way of viewing a file, and so is controlled by a viewspec. Recall that viewspecs "m" and "n" turn on and off numbering, respectively; up to now, they have turned on and off statement numbers -- they can also turn on and off SIDs. There is a viewspec pair "I" and "J" (uppercase) whose setting decides whether SIDs or statement numbers will be shown. When you log in, viewspec "J" (show statement numbers, not SIDs) is automatically set (just as "n" -- don't show numbers at all -- is set automatically). To view SIDs, simply set viewspec "I" and turn on the numbering (viewspec "m"). SIDs rather than statement numbers will appear before each statement.

If you simply set viewspec "I", it will be set, but no numbers at all will be shown because "m" is not set. Setting viewspec "I" simply says "if numbers are turned on, then show the SIDs, not the statement numbers". Setting "I" without "m" appears to do nothing.

Go ahead and try out various combinations of the viewspecs "I", "J", "m", and "n" until you see what is going on. Also, try jumping around using SIDs wherever an address is expected. A SID can be used as the target of a Jump command, as in "Jump (to) Link 043<OK>", but it can also be used for editing purposes. Try inserting a new statement below this one with the SIDs on, and then delete it by typing its SID when you are given the "M/A:" prompt in the Delete Statement command.

This method can be used to move statements that are too far apart to both be viewed on the screen at the same time. Simply find the SID of the statement to be moved, and jump to the place you want to move it to. Suppose that the SID of the statement you want to move is 089. To move it to follow a statement on your screen, use the command "Move Statement (from) 089<OK> (to follow) MARK<OK>". Alternatively, you can find the SID of the statement you want the new statement to follow (suppose it is 0183), and jump to the statement to be moved. The command would then be "Move Statement (from) MARK (to follow) 0183<OK><OK>". (For real experts, there is no need to have either of the statements on the screen for the move -- simply give the SIDs of both).

There is another pair of viewspecs, "G" and "H", (again, uppercase) which behave very much like the pair "I" and "J". "H" is set

automatically when you enter AUGMENT, but it can be changed to "G" when desired using the Set Viewspecs command. This pair of viewspecs controls whether the numbering (which may be either SIDs or statement numbers) is displayed at the beginning or the end of the statement. If viewspec "n" (numbers off) is in force, the G/H setting makes no difference, just like the I/J pair. To see the effect of setting viewspec "G", viewspec "m" must be turned on. All possible combinations of "I", "J", "G", "H", "m", and "n" are possible. They come in three pairs, and the setting of the I/J and G/H pairs are not visible unless "m" is in effect. Try out now a number of combinations of these viewspecs.

Printing

Uses for printed copy

You will find that when you first begin to use AUGMENT, you will want to print out your material quite often. The more you use the system, however, the less use you will make of the printing capabilities -- you will find the online material more easily accessible than printed material.

Of course, there are times when it is absolutely necessary to print out some document. You may use AUGMENT to prepare the text for a book or an article, but you will need printed copy for duplication. If you prepare a letter, you will want to print it before you send it (unless you send it to someone else who uses AUGMENT, in which case you can just send it to that person online).

No matter how much you use the system, you will occasionally find it necessary to get printed copy, and that is what this section is all about. It is a big section, because there are so many possibilities. AUGMENT allows users literally hundreds of options, depending on what printing device they use, what format they want, and many other things. You will find that you need to know only one or two methods to do your work, and that much of the material in this section does not apply to you. An attempt has been made to organize the material in a manner that will make it easy for you to learn exactly what you need to know.

The Print command

The AUGMENT Print command is basically very easy to use, once you are set up. Each person has a personalized set of "user profiles" which tell AUGMENT exactly how he or she usually wishes to print. Thus every user could have a different type of printer, but all would use the same printing commands. If you are a beginner, the easiest way to get started is to get someone to set up your user profiles so that they work for your particular printing device. If you do not know an experienced user who can do this for you, you will need to read through the section on user profiles that appears later (see <01191>).

automatically when you enter AUGMENT, but it can be changed to "G" when desired using the Set Viewspecs command. This pair of viewspecs controls whether the numbering (which may be either SIDs or statement numbers) is displayed at the beginning or the end of the statement. If viewspec "n" (numbers off) is in force, the G/H setting makes no difference, just like the I/J pair. To see the effect of setting viewspec "G", viewspec "m" must be turned on. All possible combinations of "I", "J", "G", "H", "m", and "n" are possible. They come in three pairs, and the setting of the I/J and G/H pairs are not visible unless "m" is in effect. Try out now a number of combinations of these viewspecs.

Printing

Uses for printed copy

You will find that when you first begin to use AUGMENT, you will want to print out your material quite often. The more you use the system, however, the less use you will make of the printing capabilities -- you will find the online material more easily accessible than printed material.

Of course, there are times when it is absolutely necessary to print out some document. You may use AUGMENT to prepare the text for a book or an article, but you will need printed copy for duplication. If you prepare a letter, you will want to print it before you send it (unless you send it to someone else who uses AUGMENT, in which case you can just send it to that person online).

No matter how much you use the system, you will occasionally find it necessary to get printed copy, and that is what this section is all about. It is a big section, because there are so many possibilities. AUGMENT allows users literally hundreds of options, depending on what printing device they use, what format they want, and many other things. You will find that you need to know only one or two methods to do your work, and that much of the material in this section does not apply to you. An attempt has been made to organize the material in a manner that will make it easy for you to learn exactly what you need to know.

The Print command

The AUGMENT Print command is basically very easy to use, once you are set up. Each person has a personalized set of "user profiles" which tell AUGMENT exactly how he or she usually wishes to print. Thus every user could have a different type of printer, but all would use the same printing commands. If you are a beginner, the easiest way to get started is to get someone to set up your user profiles so that they work for your particular printing device. If you do not know an experienced user who can do this for you, you will need to read through the section on user profiles that appears later (see <01191>).

For now, let us assume that all your user profiles are properly set up, and that you wish to print in your "usual" way.

All you need to do is type "p" for "Print" and then identify exactly what you wish to print -- a statement, branch, group, file, rest of file, or whatever. If you wish to specify viewspecs, you must type <OPT> (using the OPTION key) before typing "s" for "Statement", "b" for "Branch", and so on. After you have specified the structure to be printed, you will be asked your device type. An <OK> means your usual device, as established in your user profiles. If you are printing with all the defaults, another <OK> will end the command here.

A few simple cases of the Print command are presented below, together with a short explanation of any unusual features:

Print Branch (at) MARK (on device) <OK> <OK>

Print File (named) <OK> (on device) <OK> <OK>

An <OK> typed as the file name means this (the current) file. The name of any other file could be typed instead at this point.

Print <OPT> (printing view) x<OK> Plex (at) MARK (on device) <OK> <OK>

The "x" following the <OPT> is the viewspec with which the plex is to be printed. After the printing is complete, the viewspecs on your screen will remain as they were before the Print command was given (in other words, the "x" viewspec affects only the printed copy).

In all of the above cases, this is all that is required. Your document will be printed on your default device, whatever that may be. Of course, if your device is the workstation printer, and you have it unplugged or turned off, nothing will happen, but we were assuming that the hardware and user profiles were ready to go. A very large proportion of your printing will probably be taken care of by commands similar to those presented above.

Printing specifications

To get any combination of the specifications, simply type one after the other ("q" for "Quickformat", "c" for "Copies", and so on). As always a question mark will list out all the possibilites at any given moment. When you have finished typing specifications, type <OK>, and the command is completed.

In this section, we will discuss most of the specifications available, and you should keep in mind that not all of them apply to your device. Those discussed first are specifications that apply to any output device.

Quickformat

Normally when you print, your files will be passed through the Output Processor, which will search your file for printing directives and will produce a document formatted according to those directives. Sometimes, you want a quick format which prints more quickly and does not take the Output Processor directives into account. If you want this option, use the command word Quickformat as a printing specification.

Formatted

If you choose the Formatted specification, your file will be processed by the Output Processor before being printed. The other possibility is a Quickformat, which does nothing fancy, but prints all of the information in a reasonable way. See <01231> above. Formatted is the default.

Copies

This command word can be used to get multiple listings of the printed material. The default number is of course 1.

File

This option allows you to have the computer do all appropriate formatting, etc., but to have the final output put on a file instead of being printed on paper. This is sometimes useful when the same thing must be printed at different times, or when information must be moved to another computer which does not support AUGMENT. The file produced is not an AUGMENT file, but is rather a standard Executive sequential file.

Append

This option is similar to File above, but instead of making a new file, the information is appended to the end of an existing file.

Other workstation printer options

Please see the section below on the workstation printer queue. (See <01183>).

Different printing devices

Line printer

Workstation Printer

Terminal

The workstation printer queue

If you choose to use the printing specification "queue", AUGMENT will create a printer file with the file extension ".WPF", which stands for "workstation printer file". That file will be created in the default directory set in your user profile. The AUGMENT system default is your own directory, unless you specifically change it with the Set Profile command. Queued workstation printer files may be printed at your workstation via the Start Workstation command in Base. Alternatively, you may log in on a typewriter terminal like the Diablo printer (for letter quality printing), and use the WPCONTROL command, Start Typewriter, to print all your queued workstation printer files.

Changing your printing profile

In order to change your profile features for printing, use the Base command, "Set Profile". If you type an <OK> after issuing the command, AUGMENT will display a menu of your options. You may simply mark the appropriate feature that you wish to set. If you should accidentally mark the wrong item, you may get out of the Set Profile command by typing Command Delete, and start over. You can type Command Delete any time prior to the final confirmation in setting any profile feature.