=><=

The Alternate Source
Software Registration Form

=><=

Folks, we hope you never have a problem with our software. But, even small programs can sometimes be used in ways that authors never dream of, and most of our programs are not small. Just in case, please tell us that you're using our program. If we find a bug, we'll let you know what we found, and the proposed fix. If a fix can be applied with the PATCH command provided with most disk operating systems, we will provide the patch information and explicit "how to" instructions at no charge. At other times, we may require you to either purchase a new diskette and/or manual or to send your master disk and a postage paid, self-addressed mailer and we will provide the fix at no cost. Prices on upgrades range from $7.50 to $15.

To insure yourself of having the latest available update, please fill out your name and address below. IF this program has a registration number, it will be hand written on the actual diskette label, otherwise leave this portion blank. Transfer that number to both places provided below, clip this page on the hyphen-line, save the top half for yourself and return the bottom half to our address below. If you like, send along an extra sheet describing your system and even a little about yourself and what you're doing with your computer. We like meeting our customers, and sometimes the phone lines become clogged. If you have a problem using our product, tell us about that, too. We're still working on the Perfect Program.

Thank you for trying our products!

Respectfully,

Charley Butler
TAS Customer Service

For your records:

Program name: _____

Registration number: _____

---------- (Clip Me) ----------

Please register me as a user for this program:

Program Name: _____

My registration number is: _____

Name: _____
Address: _____
Address: _____
Address: _____

Mail to: The Alternate Source, Program Registrations, 704 North Pennsylvania Avenue, Lansing, MI 48906

North America, please include a $1 registration handling fee. Foreign customers, please include $3.
(Stamps or International Reply Coupons, OK)

TABLE OF CONTENTS

PREFACE

THE ADVENTURE SYSTEM contains programs for creating, editing and playing adventure data bases. This package requires a 48K disk or tape system.

THE ADVENTURE SYSTEM (TAS) is a tool to make programming easier. There are many concepts that have been designed into this tool that make string manipulation easy. "String manipulation" is what makes your TRS-80 appear to hang up for minutes at a time if you try to do too much.

With TAS, you are required to learn a new, unique "adventure language". Learning a new language requires a certain degree of stick-to-it-ive-ness (...not sure what language that is!). You must experiment to get comfortable with the syntax of the language. You must start thinking in => terms <= necessary to convert your ideas into adventures. Those => terms <= are the symbols of this language. As you learn more of these symbols, you will get better with the language. Remember how hard BASIC or assembly language seemed at first? (for those of you who do not know those languages -- knowledge of BASIC or assembly language is NOT required for using The Adventure System.) After a few evenings with a language, programming can be fairly easy and entertaining. End of Lesson 1.

The saying "if all else fails, read the instructions" should read "all will fail without reading the instructions" when using this software package. The "adventure language" you are about to learn has over 60 commands and conditions, and many subtle rules. Perhaps not as many as BASIC, but much more powerful than BASIC when used to write adventures. Or lessons. Or simulations. Take the language in little chunks and chew slowly. Later readings will make finer details easier to pick up.

The Adventure System has been "growing" for about three years. The manual is intended to instruct the user on everything he needs to know to write and edit adventures. You supply only the ideas.

If you have any suggestions for improvements to TAS, please send them to:

Bruce G. Hansen
c/o The Alternate Source
704 North Pennsylvania Avenue
Lansing, MI 48906

ADV/CMD is not the only driver on the market these days. However, if you desire to include ADV/CMD with your adventure data base, you should contact The Alternate Source for more information. Royalties for using ADV/CMD are a low, one-time only fee. You may also want to refer to Appendix B of this manual.

INTRODUCTION

The Adventure System DISK version contains the following programs: ADV, ADVTT and ADVEDT. All three of these programs are written in Z80 machine language.

The Adventure System TAPE version contains the programs: ADVTT and ADVEDT. Tape users must access these two programs using the Level II SYSTEM command.

In addition, both systems include three "data bases." These can be loaded into the adventure editor (ADVEDT) and played with the adventure driver (ADV). Adventures X and Y are full length adventures. We will not help you solve them. You have all the tools necessary for doing so with this package. Adventure Z is a "baby" adventure, written just to help you get started.

ADVEDT is the editing program for adventures. The editor is used to enter and modify adventures.

ADV is an adventure driver program which interprets the commands you give to the editor. ADV and ADVEDT are used to play your adventures on a disk system. ADVEDT can also be used to play your adventures on a tape system.

TAPE USERS NOTE! The ADVTT program will read in a data base saved to tape and dump out a tape version of the adventure. This tape version is a combination of the adventure driver and the data base. This program will allow anyone with a tape-based Level II TRS-80 to play your adventure. Normally, data bases saved with ADVEDT cannot be run using the Level II SYSTEM command. You must save them with the ADVTT program in order to do so.

This manual is divided into chapters, each containing a different section of ADVENTURE or using the TAS package. A summary of the chapters and appendices is given below:

Chapter 1 — Overview of TAS and adventures in general. This chapter describes what the program ADVEDT does.

Chapter 2 — Description of the ADVENTURE data base structure. This chapter will describe in detail what the different conditions and commands of ADVENTURE are. Understanding this chapter is essential before trying to write your own adventures.

Chapter 3 — ADVENTURE program instructions. This chapter describes how adventures must be entered so they will work properly with the adventure driver program (ADV).

Chapter 4 — Operating Instructions. This chapter contains the instructions for using ADVEDT and ADVTT. A suggested procedure is also given to assist you in entering an adventure.

Chapter 5 — Sample ADVENTURE. This chapter will describe a short adventure written to show how to use most of the commands and conditions. This data base is adventure "Z" on your master diskette or tape.

Chapter 6 Getting started. This chapter goes through all of the steps necessary to write an adventure: Coming up with an idea, writing the adventure, typing it into ADVEDT and debugging the adventure.

Chapter 7 ADVENTURE solving techniques. This chapter will describe how to solve any adventure written with The Adventure system.

Appendix A contains an abbreviated ADVEDT command summary. This will be highly useful when writing adventures once you are familiar with the commands and conditions.

Appendix B tells how to submit your adventures to THE ALTERNATE SOURCE for marketing purposes.

Chapter 1

OVERVIEW OF THE ADVENTURE SYSTEM

The Adventure System is a set of programs which enable you to create your own adventures. "The Adventure System" will be abbreviated frequently in this manual as, "TAS".

TAS uses a unique format for saving your adventures to tape or disk. This format allows The Adventure System "driver" program to run any adventure data base written with The Adventure System editor.

The "driver" program is analogous to the Microsoft BASIC interpreter in every TRS-80 computer. The "data base", or the actual adventure, is analogous to a BASIC program saved on disk or tape. By itself, a BASIC program saved on disk or tape can not be executed. The BASIC interpreter is required to "understand" the instructions in the BASIC program and execute them. The same type of procedure is used with TAS. Adventure data bases saved to tape or disk are not stand-alone programs, they require the adventure "driver" or interpreter.

The "driver" program of The Adventure System is called "ADV". This is the program which is run to execute a set of adventure instructions or adventure data base.

The program which allows you to edit and create your own adventure instructions (an adventure data base) is called "ADVEDT" (ADVenture EDiTor). To aid in debugging your adventures, "ADVEDT" has the adventure driver, ADV, built into it. This way you can start entering adventure instructions and easily jump to the driver see if the results of your efforts were what was expected.

Also included with TAS is the utility program "ADVTT". This program will read in an adventure data base saved to tape and write out a SYSTEM tape suitable for playing, complete with the data base "driver". The tapes written out by "ADVTT" are a combination of the driver and adventure data base. ADVTT does not save out the editor. If you want to give a copy of your adventure to a friend, this is how we request you do it, by saving the adventure with ADVTT.

For those unfamiliar with the playing style of TAS adventures, here is a brief overview. The player, or adventurer, is placed in a scenario devised by the author of the adventure. The player is given a description of the current room, a list of objects, if any, in the room and any obvious exits. Obvious exits are directions in which the player may move to further explore or exit that particular room. Obvious exits are generally doors, stairways, and the like. Sometimes an exit must be "discovered."

Two-word commands are used to move about, pick up and examine objects and generally accomplish what you want to do. These two word commands are normally a verb and a noun. For example, suppose an ax is in the room with the player. To pick up the ax, the player would type in "GET AX".

Adventures have a fairly limited vocabulary when compared to a normal human (at most 300 words). One of the major challenges of adventure is to figure how to communicate what is to be done. Generally you, are communicating with your built-in computer "robot" and providing instruction.

When an adventure is started, a typical screen display could be as follows:

```
I'm in a dark cave.  Visible items:
Coal.  Pick ax.  Rock pile.

Obvious exits are:  NORTH EAST DOWN
<------------------------------------------------------>

Welcome to the demo adventure by Bruce G. Hansen
Dedicated to all Adventure System users!!!

Tell me what to do -->
```

Everything listed above the dashed line is the "room display". "I'm in a dark cave" describes the room you're in. "Coal", "Pick ax" and "Rock pile" are visible objects in the room (there may be some others in the room that are not immediately visible - part of the adventure could be finding these "hidden" objects). "NORTH", "EAST" and "DOWN" are the obvious directions the player can move in.

The adventure driver provides a shorthand entry for moving in the standard directions (NORTH, EAST, SOUTH, WEST, UP and DOWN). Typing in just the first letter of the direction is all that is required to move in that direction. For example, typing "E" and <ENTER> would move the player to the East.

Below the dashed line is the area where the computer communicates to you through messages. The "Welcome to the demo" line is a message the driver program was instructed to send to you. "Tell me what to do --> " is the player's cue to type in a command. For example, the player could type in "GET COAL". If the adventure allows the coal to be picked up, the player may GET the coal. It will no longer be shown as a visible object when the player is carrying it.

To see what he is carrying, the player must take an inventory. Usually, an inventory can be taken by typing in "GET INVENTORY" or just "INVENTORY" or simply type "I" (no quotes) and <ENTER>. The exact format is controlled by the adventure data base. If the verb and noun "GET INVENTORY" are not in the adventure data base, then that input will not work. If the adventure data base recognizes the player response "INVENTORY" as a means for taking an inventory, then simply typing "I" and <ENTER> will display a list of items being carried. This is a special case, handled by the driver.

There are three adventures on the master disk or tape. Adventures "X" and "Y" are on your master diskette as ADVENT/DX and ADVENT/DY (or your cassette as "X" and "Y"). Adventure "X" is a "Miner's adventure" and adventure "Y" is a "Burglar's adventure". These are full sized adventures and should provide many hours of challenge and frustration. Adventure "Z" is a "Mini-venture" which is explained in Chapter 5 of this manual.

How do you write your own adventures? The concept is very simple. The basic idea is that certain commands are executed when certain conditions are met.

Suppose in an adventure you need to pick a lock with a hairpin before a door can be opened. The conditions to be met could be:

1) The player is holding a hairpin, and
2) The player is in the same room as a locked door.

If these conditions were true and the player typed in "PICK LOCK", some commands would be executed. These commands could be:

1) Remove the locked door from the room, and
2) Place an unlocked door in the room.

Remember, this was only an example, the scenario for the adventure is determined completely by the adventure's author.

The trick to writing good adventures is making the conditions subtle, but logical. For example, if you found a hair pin in one room, it would be logical to use that pin to pick a lock in another room if the door was locked. Writing an adventure is like writing a story. There is a major plot with minor conflicts encountered along the way.

Everything you need for writing adventures is supplied with TAS except ideas. Possible plots can be found from any book, movie, etc. Also, the Adventure User's Group has a newsletter devoted to TAS. The newsletter, AUGMENT, normally has some ideas for adventures in it. Information for joining the AUGGIES may be found elsewhere in this manual. The possibilities for adventures are endless!

Chapter 2

THE ADVENTURE LANGUAGE

The data base is the heart of the adventure. By changing this data base you can create different adventures. We will first tell you everything that is in a data base and then go into detail about each one. Don't even plan on learning everything in this chapter after one reading. You will probably refer to it many times for your first adventure writing project.

OVERVIEW

The data base consists of the following sections:

1) HEADER information. The header contains the number of actions, vocabulary entries, rooms, messages, objects and other variables. It is important that the header ALWAYS reflect the number of items you want in your data base (or more). This process is described later.

2) ACTION entries. There are two kinds of actions: AUTOMATIC ACTIONS and PLAYER INPUT ACTIONS. Player input action entries contain player input (verb and noun) plus conditions and commands. The automatic actions serve mainly for bookkeeping. Action entries are analogous to IF...THEN statements in BASIC.

3) VOCABULARY entries. These two lists (verbs and nouns) contain all of the words the player may use in this particular adventure.

4) MESSAGE text. These are the messages used by the adventure to communicate with the player and are controlled by the actions.

5) ROOM description. This is a list of directions for getting to other rooms (from the current room) along with a text room description.

6) OBJECT description and starting location. The description of the object determines if it is a treasure, an object which may be carried and dropped and the name of the object. The starting location tells if the object is in the current room, if it is being carried or if the object is in the storeroom.

7) TRAILER information. This contains the version number, adventure number and a checksum. It is not important to know anything about this in order to write adventures.

REFRENCING THE DATA BASE COMPONETS

The adventure data base is essentially a collection of action entries, rooms, messages, vocabulary words and objects. Each of the entries in these data base sections are numbered from zero up to the last entry. This format is used so a unique item in any of the data base sections can be viewed and/or changed. This is similar to having a BASIC program line number to uniquely reference each individual line.

For example, a list of objects in an adventure could be:

1 —> Fire breathing DRAGON!!!
2 —> A large sword
3 —> Fair maiden
4 —> Coat of mail

With the number referencing system used in TAS, the "Fire breathing DRAGON!!!" would be referred to as object 1, the "Fair maiden" as object 3, etc.

The other data base sections are numbered in the same manner. Chapter 5 has a listing of a short adventure which shows how each of the sections are numbered.

HEADER ENTRIES

The header contains the following information:

1) The number of bytes required to hold all of the text descriptions such as verbs, nouns, messages, room descriptions and object descriptions. This number includes a fixed number of characters for each verb and noun. This fixed number is the word length of this adventure plus one, times the number of vocabulary words. Also added to that value is one more than the number of characters between quotes in the messages, and room and object descriptions. You do not have to worry about this HEADER entry. It is updated automatically.

2) The highest numbered object in this particular adventure. The objects are numbered starting at zero, so the number of objects is one plus this number.

3) The highest numbered action in this particular adventure. Actions are numbered starting at zero, so the number of actions is one plus this number.

4) The highest numbered vocabulary word in this adventure. This applies to both verbs and nouns, being the larger value if they are different. Vocabulary words are numbered starting at zero, so the total number of verbs and total number of nouns is one plus this number. Some vocabulary words (GO, GET, DROP, NORTH, SOUTH, etc.) are "predefined". This is discussed later.

5) The highest numbered room in this adventure. Rooms are numbered starting at zero, but room zero is reserved as "the storeroom" so this value is the total number of rooms in which the player may move. The storeroom is explained later.

6) The maximum number of objects which may be carried. Under certain conditions, the actions can cause more than this number of objects to be carried. The player will not be able to voluntarily pick up anything unless the number of objects currently being carried is less than this number.

7) The starting room number for this adventure. This is the room number the player starts in at the beginning of the adventure.

8) The number of treasures in this adventure. When the SCORE command is issued, this number is divided by the number of treasures in the treasure room to give the percent score. If the result is 100 percent, the player wins the adventure. Treasures are "defined" as a certain kind of object. This process is described later.

9) The word length used by this adventure. This number affects the nouns and verbs. When the adventure data base is read in by the adventure driver program, all nouns and verbs are either truncated or padded to this length plus one. This value is the minimum length of verbs and nouns the player may input. Normally this value is three or four. Therefore, only the first three characters (if the word length is three) of the player's verb and noun are used.

10) The time limit. This may be used in some games to control how long the artificial light will last. If there is no artificial light, it may control the number of turns in this adventure. If the artificial light is re-filled, this value is put back in the time limit counter. The maximum time limit is 32767.

11) The highest numbered message. Messages are used by the adventure driver to communicate with the player. Messages are numbered from one, so this value is the number of messages.

12) The treasure room number. When treasures are in this room, they are considered collected. When the SCORE command is issued, the treasures in this room are summed and divided by the number of treasures for the percent score.

You must manually enter all of the items contained in the HEADER except for item one. That item is computed by the adventure editor automatically. Defining the header should be the FIRST thing done before entering an adventure. You can always increase values later, if needed. Study the maximum values for each in appendix A.

ACTION ENTRIES

ACTIONS are the heart of the adventure. Some are player input actions and others are automatic operation actions. Action entries are like IF...THEN statements in BASIC.

Action entries contain the following pieces of information: Verb, Noun, Conditions, Commands and Action Titles.

Action titles will be considered only briefly, since they are the easiest. The action titles of an action entry are used only to document the function of the action entry. They are optional and serve only as comments.

The verb and noun combination input by the player are used to select the "conditions" and "commands" to be executed. For example, if the player were to type "GO CAVE" into the adventure driver, the driver would only consider action entries with the verb, noun combination of "GO CAVE". If an action entry had the verb, noun combination of "GO HOLE" it would not be considered if the player typed in "GO CAVE" (unless HOLE and CAVE are defined as SYNONYMS, but we'll get to that).

The "conditions" of an action entry are a list of tests to be considered befor performing the "commands" of the action. Suppose the player typed in "SHOOT GUN". Before the player would be allowed to shoot a gun, a "condition" may be that he is holding onto a gun.

The "commands" of an action entry are performed if all of the "conditions" are met. If only one condition of an action entry fails, the commands of that action entry will not be performed. Suppose the player types "SHOOT GUN". An action entry is present with the verb, noun combination of "SHOOT GUN", so it is considered. Suppose the only condition to shoot the gun was that the player was holding onto a gun. If the condition(s) are true, the command(s) are executed. A command could be to display the message "Bang!!!".

Both action "conditions" and action "commands" are described in detail shortly, but first, let's look at the two kinds of actions.

PLAYER INPUT ACTIONS

If the verb of the action entry is not AUTO, then the action is a "Player Input Action". The verb, noun combination of the action must match the verb, noun combination of player's input for the action to be considered. If the noun of the action entry is "ANY", it matches any possible noun in the player's input.

For the following example, suppose the player input the verb, noun combination "EXAMINE DOOR". If an action entry had the verb, noun combination "EXAMINE DOOR", then that action entry would be considered. Likewise, if the action entry had the verb, noun combination "EXAMINE ANY", the action would be considered. A noun of "ANY" in an action entry really means "match with ANY player's noun".

When player input actions are being checked for a matching verb, noun combination, the action entries are scanned in ascending numeric order. When the verb and noun of an action entry match the player's input, the conditions are evaluated. The procedure of checking the conditions of a matched verb, noun combination is called "considering" an action entry. If all of the conditions of the action entry are true, then the commands of this action entry are performed. See how it's similar to a BASIC IF...THEN statement?

When a "true" match is found (all conditions of the action entry were found to be true), no further player input actions are evaluated on this pass. A "pass" is defined as the turn a player gets when he types in his input.

For example, suppose the player types in "CUT ROPE" and there are two action entries with that same verb, noun combination. If all of the conditions of the first "CUT ROPE" action were true (thus a "true" action or match), the second action would not be considered.

However, if a verb-noun match is made, and all of the conditions were not true, then the scanning procedure continues until either a "true" match is made, or all of the actions are checked for a matching verb, noun combination.

Using our above example, suppose the conditions of the first "CUT ROPE" action entry were not all true. The adventure driver would continue scanning the action entries for another "CUT ROPE" action. It would eventually find the second "CUT ROPE" action and consider the conditions of that action entry also.

If a verb-noun match was found in at least one action entry, but the conditions were not true in any of the matched actions, then the message --

"I can't do that . . . yet!"

is displayed after all of the action entries have been checked. If no verb-noun match was found, then the message "I'm sorry, but I don't understand what you mean" is displayed.

AUTOMATIC ACTIONS

If the verb of the action entry is "AUTO", then this is an automatic action. The automatic action entries have a variety of uses. All automatic actions are considered before the player is allowed to input his verb, noun combination.

Another way of thinking of auto actions is this: when the adventure driver is running through the automatic action entries, the computer is taking its turn. When the player inputs a verb, noun combination, the player is taking his turn.

Such things as falling asleep, checking for day/night or any other task that must be performed regardless of the player's input are candidates for automatic action entries. Chapter 5 contains a more detailed description of automatic actions.

Automatic actions are very similar to the player input actions described above. There are a few major differences however. These are:

1) The verb, noun combination of an automatic action is the verb "AUTO" and the noun is a number from 1 to 100. The noun number is the probability that this action will be considered. For example, an "AUTO 20" is an auto action which has a 20 percent chance of being considered.

2) The automatic actions must be at the beginning of the action entries - before any player input actions. If an auto action is entered after a player input action, then that auto action will never be considered. This chapter (along with chapters 3, 4, 5 and 6) has more information on the order of action entries in subsequent sections.

3) When automatic actions are being evaluated, they are all scanned regardless of how many are true or false. As you recall, when player input actions are evaluated, only the first "true" action entry is performed. With auto actions, ALL auto actions are considered regardless of how many are true or false.

ACTION ENTRY CONDITIONS

If the action is to be considered (the action's verb and noun match the player's verb and noun), up to five conditions are evaluated. If any conditions fail, the commands in the action are not performed. The commands of an action entry are explained later on in this chapter.

All condition codes have a number associated with them. This number refers to another component of the adventure. For example, "HAS 10" is a condition to test if the player is carrying (HAS) object 10. The condition codes are as follows:

PAR This condition always passes. The number included with PAR (that is, PAR 20) may be used by the commands in this entry. See the manual section following the condition description for more details on PARameters. Briefly, parameters are like DATA statements in BASIC.

HAS The condition passes if the player is carrying (HAS) the numbered object (that is, HAS 15 tests if the player is holding object 15). It fails if the object is either in the same room as the player or is in any other room.

IN/W The condition passes if the player is in the same room as the numbered object (IN/With). It fails if the player is either holding the object or the object is in any other room. Example, "IN/W 5" tests if the player is IN/With object 5.

AVL The condition passes if the numbered object is AVaiLable because the player is either carrying the object or is in the same room as the object. It fails if the object is in any other room. Example, "AVL 4" tests if object 4 is AVaiLable to the player.

IN The condition passes if the player is IN the numbered room. It fails if the player is in any other room. Example, "IN 2" tests if the player is in room 2.

-IN/W The condition passes if the numbered object is either held by the player or if the object is in any other room. It fails if the object is in the same room as the player. Example, "-IN/W 52" tests if the player is not IN/With object 52.

-HAVE The condition passes if the player is not carrying the numbered object. It fails if the player is carrying the object. Example, "-HAVE 0" tests if the player does not HAVE object 0.

-IN The condition passes if the player is not in the numbered room. The condition fails if the player is in any other room. Example, "-IN 35" tests if the player is not IN room 35.

BIT The condition passes if the numbered bit flag is set. It fails if the flag is cleared. See the description of bit flags later on for more information. Example, "BIT 3" tests if BIT flag 3 is set.

-BIT The condition passes if the numbered bit flag is cleared. It fails if the flag is set. See the description of bit flags later on for more information. Example, "-BIT 4" tests if BIT flag 4 is reset or cleared.

ANY The condition passes if the player is carrying any objects at all. It fails if the player is not carrying any objects. The number entered (that is, the 50 in the condition ANY 50) has no affect on this condition. Example, "ANY 50" tests if the player is holding ANY objects. The "50" included with the condition code "ANY" is not used and may be any legal entry. Usually it is entered as a zero ("ANY 0").

-ANY The condition passes if the player is not carrying any objects. It fails if the player is carrying any objects at all. Example, "-ANY 0" tests if the player is not holding ANY objects.

-AVL The condition passes if the numbered object is in any other room. It fails if the object is available either because it is being carried or it is in the same room as the player. Example, "-AVL 22" tests if object 22 is not AVaiLable to the player.

-RM0 The condition passes if the numbered object is not in room zero. Room zero is reserved as a storeroom. The condition fails if the object is in room zero. Example, "-RM0 20" tests if object 20 is not in RooM zero.

RM0 The condition passes if the numbered object is in room zero. The condition fails if the object is in any room other than room zero. Example, "RM0 11" tests if object 11 is in RooM zero.

CT<= The condition passes if the counter is less than or equal to the number. It fails if the counter is greater than the number. See the description of the counters later on for more information on them. Example, "CT<= 100" tests if the CounTer is less than or equal to 100.

CT> The condition passes if the counter is greater than the number. It fails if the counter is less than or equal to the number. Example, "CT> 50" tests if the CounTer is greater than 50.

ORIG The condition passes if the numbered object is in the ORIGinal (same) room it started in. It fails if the object is in any other room or is being carried. Example, "ORIG 10" tests if object 10 is in the ORIGinal room in which the adventure started.

-ORIG The condition passes if the numbered object is in any room other than its starting room or is being carried. It fails if the object is in the same room it started in. Example, "-ORIG 5" tests if object 5 is not in the ORIGinal room that it started the adventure in.

CT= This condition passes if the counter is equal to the number. It fails if the counter is not equal to the number. Example, "CT= 10" tests if the CounTer is equal to 10.

The number input with the condition must be in the range 0-1600. For example, "CT= -1" is an illegal number associated with the condition "CT=". Likewise, a "PAR 1601" is illegal.

Any action entry may have from zero (0) to five (5) conditions to be tested before its commands are executed. When the number of conditions is less than 5 (or zero to four), the unused conditions are normally set to "PAR 0". Since this "PAR" condition always passes (or tests to be true), this is the "no condition" state. Chapters 5 and 6 have more details on unused conditions in an action entry.

So far we have covered the verbs, nouns and conditions of an action entry. The last section of the action entries to be covered are the commands. If the verb and noun match the player's input (or this is an automatic action with a true probability), then the conditions are evaluated. If all of the conditions are true, then the commands are executed. The following is an explanation of the commands.

ACTION ENTRY COMMANDS

There are up to four commands per action entry. These four commands may use one or more conditions (or DATA statement elements) entered in the condition line of the same action entry. For example, if the first parameter found in the conditions was a 10 (PAR 10) and the first command which uses a parameter in the commands is a "GOTOY" command, the player would move to room 10 (GOTO 10). Commands that use parameters act like a READ statement in BASIC. They read the next item from the PAR values. The PAR values are in the "conditions" of the same action. It is possible to CONTinue actions, but you cannot pass parameters to the next action.

If a command uses one parameter, its value is represented by "Par #1" in the following command description. If the command uses two parameters, the first is represented by "Par #1" and the second by "Par #2." The parameters used by any command are skipped by later commands if they also use parameters. Again, this is exactly how BASIC handles READ...DATA statements.

For example, if the conditions contain three parameters: PAR 3, PAR 15, PAR 26, in that order, (or DATA 3,15,26 in BASIC), the first command (or READ statement) that uses a parameter will use the 3 (READ X - X will equal 3), the second command would READ the 15 and the third would READ the 26. Too many parameters in the conditions has no effect. It's like having extra DATA left over in a BASIC DATA statement.

Not having as many parameters in the conditions as the number expected by the commands will produce strange results. The adventure driver will not display an error message like BASIC's "Out of DATA", and the results will not be as expected. For example, specifying no parameters in the conditions and having a "GETX" (which is a command that uses a parameter) in the commands will produce unpredictable results.

As stated before, there may be up to four commands in an action entry. If a command is unused, it is displayed as a "-".

Commands are executed in the order in which they appear. For example, if an action entry contained the commands "GETX", "MSG5", "GOTOY" and "EXX,X", they would be executed in that order.

The following are the available commands:

0 No command or message. This is a "null" command and is the command entered when not all four possible commands are used. It is displayed as a "-".

1-99 Display message numbers 1-99.

GETX Pick up the Par #1 object unless the player is already carrying the maximum number or carry limit of objects which may be carried. The object may be in the current room or in any other room.

DROPX Drop the Par #1 object in the same room as the player. The object may be carried or in another room.

GOTOY Move the player to the Par #1 room. This command may need to be followed by a DAY/NIGHT command (described later) depending on the light status of the room.

X-RM0 This command moves the Par #1 object to room zero.

NIGHT This command sets the light/darkness bit flag (15). The room will be dark if the artificial light source is not available. The artificial light source is described in more detail in the bit flag and object sections of this chapter. This command should be followed by a DSPRM command.

DAY Clear the light/darkness bit flag (15). This should also be followed by a DSPRM command.

SETZ Set the Par #1 bit flag.

X->RM0 This command moves the Par #1 object to room zero.

CLRZ This clears the Par #1 bit flag.

DEAD This clears the light/darkness flag (makes it light), moves the player to the last room and tells him he is dead. This is the command normally used to kill the player. The HEADER holds the last room number.

X->Y Move the Par #1 object to the Par #2 room. This command will automatically display the room if the Par #1 object either entered or exited the current room.

FINI Indicate to the player that the game is over and inquire if he wants to play again.

ACTION ENTRY COMMANDS (con't)

DSPRM Display the current room. This checks the light/darkness flag and if the artificial light source is present. If it is light, the room description, visible objects and obvious exits are displayed. If it is dark, nothing is displayed (it is too dark to see) unless the artificial light source is present. A room is "light" if bit flag 15 is not set or the artificial light source is in the current room. A room is dark if bit flag 15 is set and the artificial light source is not in the current room.

SCORE Tells the player how many treasures have been stored in the treasure room and what percent of the treasures have been stored. If one hundred percent have been stored, then the winning message is displayed and the player is given the option of playing again.

INV Gives the player an INVentory or list of what objects are being carried.

SET0 This sets bit flag zero. It may be useful since no parameter from the conditions is necessary.

CLR0 Clears bit flag zero. It may be useful since no parameter from the conditions is necessary.

FILL Re-fills the artificial light source and clears bit flag 16 (indicator of light source status). Re-filling the artificial light source sets the counter containing the number of moves left before the light source runs out to the time limit of the adventure. The time limit value is specified in the HEADER section. This also picks up the artificial light source (object 9). This command should immediately be followed by a X->RM0 command where Par #1 is the unlighted artificial light source (they are two different objects). In other words, after a FILL command, the artificial light source in its unlighted state should be sent to the storeroom.

CLS This command did a clear screen in the BASIC version of ADVENTURE and does nothing in the machine language version. This command is included just for compatibility.

SAVE Saves the game to disk or tape depending on which version is being used. It writes the player's game variables such as the current room, current locations of all objects, status of all bit flags, current values of all alternate room registers and the current values of all counters.

EXX,X Exchange the room location of the Par #1 object with the room location of the Par #2 object. A DSPRM is automatically performed if either Par #1 or Par #2 objects were in the current room.

ACTION ENTRY COMMANDS (con't)

CONT This command sets a flag to allow this action entry to be CONTinued. In effect, another five conditions and/or four more commands could be performed. When all commands of the action entry containing the CONT command have been performed, the conditions of all subsequent action entries with an "AUTO 0" verb, noun combination (up to the first non "AUTO 0" verb and noun) will be evaluated. The checking procedure of the "AUTO 0" actions continues regardless if the action entry being checked is true or false. CONTinued action entries are similar to automatic action entries in this respect. For example, consider the following actions:

```
3: LIGHT TORCH  HAS 12    PAR 9    PAR 12   PAR 0    PAR 0
       EXX,X MSG5 CONT -
4: AUTO  0      PAR 1     PAR 0    PAR 0    PAR 0    PAR 0
       EXM,CT  CT-1    -     -
5: AUTO  0       CT= 0    PAR 9    PAR 12   PAR 0    PAR 0
       EXX,X   MSG6    -     -
6: AUTO  0       PAR 1    PAR 0    PAR 0    PAR 0    PAR 0
       EXM,CT  -       -     -
7: SHOOT GUN     HAS 23   IN/W 2   PAR 2    PAR 4    PAR 0
       EXX,X   MSG8    -     -
```

If the conditions of the action with the verb-noun of "LIGHT TORCH" are found to be true (action entry 3), then its commands are executed. One of its commands is a "CONT". This means that all "AUTO 0" verb-noun actions following the "LIGHT TORCH" action will be considered. In this case, there are three of these "AUTO 0" action entries. All three are considered even if none of them are true or false. For example, the third one is considered even if the second one was a "true" action entry. See how the CONTinued action entries are similar in this respect to the automatic action entries? If CONTinued actions acted like player input actions, then if the first one was true, the second and third ones would not be considered.

Note that only the three "AUTO 0" action entries would be checked. The "CONT" command stopped action entry evaluation at the first non "AUTO 0" action which was action 7 (SHOOT GUN).

AGETX Always GET the Par #1 object even if the carry limit is overflowed.

BYX->X Put the Par #1 object in the same room as the Par #2 object. If the Par #2 object is being carried, this will pick up the Par #1 object also, regardless of the carry limit. If this command changes the location of any objects in the current room, a DSPRM command is automatically executed.

CT-1 Subtract one from the counter value.

ACTION ENTRY COMMANDS (cont)

DSPCT This displays the value of the counter. No carriage return is printed after the value. This is similar to the BASIC statement "PRINT CT;".

CT<-N The sets the counter equal to the Par #1 value.

EXRMO This exchanges the current room with the room number held in alternate room register zero. This command may be used to save a player's current room for return to it later on. This command should be followed by a "GOTOY" command if alternate room register zero had not been set. Alternate room registers are explained in greater detail later on in this chapter.

EXM,CT Exchange the value of the counter and the value of the Par #1 alternate counter. There are eight counters numbered 0 to 7. When the adventure starts, these are not set to any particular value, so initialization automatic action entries should set them if they are to be used. Also, the time limit may be accessed by exchanging with alternate counter eight (8). The counters are explained in greater detail later on in this chapter.

CT+N Add the Par #1 value to the counter.

CT-N Subtract the Par #1 value from the counter.

SAYW This displays the noun (second word) input by the player without a carriage return. This is like the BASIC statement "PRINT NOUN$;".

SAYWCR This displays the noun (second word) input by the player followed by a carriage return. It is like the BASIC statement "PRINT NOUN$".

SAYCR Starts a new line on the display. It is like the BASIC statement "PRINT".

EXC,CR Exchange the value of the current room with the Par #1 alternate room register. This may be used to remember more than one room. There are six alternate room registers numbered from 0 to 5.

DELAY This command pauses for about 1 second before going on to the next command.

It may be useful to have a list of the action entry commands broken down by the number of parameters each requires. The following list does this:

NO PARameters		ONE PARameter	TWO PARmeters
0		GETX	X->Y
Messages (1-99)		DROPX	EXX,X
		GOTOY	BYX->X
NIGHT	CLS	X-RM0	
DAY	SAVE	SETZ	
DEAD	CONT	X->RM0	
FINI	CT-1	CLRZ	
DSPRM	DSPCT	AGETX	
SCORE	EXRM0	CT<-N	
INV	SAYW	EXM,CT	
SET0	SAYWCR	CT+N	
CLR0	SAYCR	CT-N	
FILL	DELAY	EXC,CR	

This list could be helpful for determining which PARameters are used by what commands in the action entries.

PARAMETER USAGE

The use of parameters is perhaps the most confusing facet of The Adventure System. New things are most easily learned when compared to something you already know. Since most people have an understanding of the BASIC READ...DATA statements, they will be used to help explain parameters.

Parameters are values defined in the action entry conditions via the "PAR" condition. They are the same as using a DATA statement in BASIC. One important difference is that each DATA statement is associated only with the action entry it appears in. For example, if some parameters are listed in the conditions of action entry #5, only the commands of action entry #5 can READ them.

The commands which require parameters act like READ statements in BASIC. The big difference is that they will only read data from the action entry they appear in. For example, if a "GETX" command is in action entry #12, it will only read a parameter (or DATA value) from action entry #12. It won't read one from action #11 or action #13 or any other action entry. An example would probably help at this point. Consider the following action:

```
SHOOT GUN   HAS 10   PAR 9    PAR 10    PAR 25     0
            EXX,X    MSG5     DROPX       -
```

The conditions in this action are "HAS 10", "PAR 9", "PAR 10", "PAR 25" and "0". The "0" signifies a "PAR 0". This shorthand is used since many action entries will probably have some unused conditions. In order to clarify the listing, "PAR 0" is displayed as "0".

If the player types "SHOOT GUN" and is holding object 10, this action entry is "true" (the parameters are always "true"). Therefore, the commands are executed.

In this case, the command "EXX,X" is done first. "EXX,X" uses two parameters (see COMMAND description above). Since the parameters are contained in the conditions, the object numbers having their locations switched by the "EXX,X" command are found there. The first parameter found in the conditions is a "PAR 9". Since "EXX,X" requires two parameters, the second parameter in the conditions, "PAR 10", is also read. The location of object 9 and object 10 are then switched by the "EXX,X" command. The BASIC equivalent would be a "DATA 9,10". A "READ X" would read the 9, a "READ Y" would read the 10.

The next command is a "MSG5". This command does not use any parameters since it simply displays message number 5. Its BASIC counterpart would be something like "PRINT A$(5)".

The next command is a "DROPX". The "DROPX" command requires a parameter so it knows which object to drop in the player's current room. The next parameter (or DATA element) defined in the conditions is a "PAR 25". So, the "DROPX" command READs the 25 and drops that object (object 25) in the player's current room.

Let's take a look at our "SHOOT GUN" action entry again:

```
SHOOT GUN   HAS 10   PAR 9    PAR 10    PAR 25    0
            EXX,X    MSG5     DROPX     -
```

Now let's list the conditions and commands of this action entry in the order they appear within the action entry:

```
HAS 10          EXX,X
PAR 9           MSG5
PAR 10          DROPX
PAR 25            -
  0
```

Now, let's eliminate all non "PAR" conditions. This leaves us with the lists:

```
PAR 9           EXX,X
PAR 10          MSG5
PAR 25          DROPX
  0               -
```

When the commands of this action entry are executed, they are performed from the top down. In this case, the first command to be executed would be an "EXX,X". As explained above, the "EXX,X" command requires two parameters from the conditions. The parameters are placed in the conditions so let's look at the list of conditions. The first two PARameters are "PAR 9" and "PAR 10". The "EXX,X" command will read the "PAR 9" and "PAR 10" and switch (or exchange) the room locations of objects 9 and 10.

The second command is a "MSG5". This command requires no parameters. It simply displays message 5 on the screen.

The third command is a "DROPX" command. As explained above, this command requires one parameter. As we refer back to our list of parameters in the conditions, we see the next one is a "PAR 25". The "DROPX" command reads the "PAR 25" value and proceeds to drop object 25 in the player's room.

The fourth command is a "-". This is an unused command and it does nothing.

Notice that the "0" or "PAR 0" parameter is not used. As stated above, not all parameters which appear in the conditions need to be used. However, there would be a problem if we ran out of conditions when there were still commands to be executed which required parameters.

The previous two examples interpreted what an action entry meant. Now let's think of a situation we want to check with an action entry and write the action entry ourselves. This action entry will use parameters to help explain their use further.

Suppose our scenario is this, we want to shoot a buffalo with a gun. After the buffalo has been shot, we will remove it from the room and replace it with a dead buffalo. The message "Got em!!" will also be displayed. The conditions for this action would be:

1) The player is holding onto a gun.
2) The player is in the same room as the buffalo.

The commands would be:

1) Remove the buffalo from the current room.
2) Drop a dead buffalo into the current room.
3) Display the message "Got em!!".

It will be necessary to define some data base sections before writing the action entry. The data base sections to be used are as follows:

Verb	Noun	Objects	Messages
SHOOT	BUFFALO	1: Gun	5: Got em!!
		2: Buffalo	
		3: Dead Buffalo	

The verb, noun combination of this action entry will be chosen to be "SHOOT BUFFALO".

The first condition should check if the player is holding onto the gun (object 1). The condition which accomplishes this is a "HAS" condition. Therefore, the condition will be "HAS 1". This condition checks to see if the player is holding onto the gun. If the player is not holding the gun (object 1), this action entry fails and it is skipped without its commands being executed.

The second condition checks if the player is in the same room as the buffalo (object 2). The condition which checks this is an "IN/W" condition. Since we want to check if the player is in with the buffalo, the condition would be "IN/W 2". So, if the player is in with the buffalo (object 2), the condition will pass. However, if the player is not in the same room as the buffalo, this condition will not pass and this action entry will be skipped without any of its commands being executed.

You may ask why we did these two conditions? The first one makes sure the player is holding onto the gun before shooting the buffalo. I chose to let the player shoot the buffalo ONLY when holding onto the gun. It doesn't make sense to let the player shoot the buffalo if he isn't carrying a gun. The second one makes sure the player is in with the buffalo before shooting it. This makes sense since we don't want the player to be able to shoot the buffalo if he's not in the same room as the buffalo.

Anyway, getting onto the commands. The first command should remove the live buffalo (object 2) from the player's current room and place it in the storeroom. The storeroom is used to store or save all objects which the player has not either found or objects we don't want the player to see yet. Since the live buffalo is dead, this is an object we don't want the player to see. The command to do this is an "X->RM0".

If you refer to the ACTION ENTRY COMMANDS section above, you will see that the "X->RM0" command uses a parameter from the conditions. This means we will have to add a "PAR 2" to the conditions. When the "X->RM0" command is executed, it will read a PARameter from the conditions. Since the object to be moved to the storeroom is object 2 (the live buffalo), the "PAR 2" must be placed in the conditions for that command to remove the proper object.

The second command will drop the dead buffalo (object 3) into the player's room. The command to drop an object into the player's room is a "DROPX". As shown above in the ACTION ENTRY COMMANDS section, the "DROPX" command requires one parameter from the conditions. Since we're writing the action entry, we will have to place the correct parameter in the conditions. In this case, it is a "PAR 3". Object 3 (the dead buffalo) is to be dropped in the player's room. When the "DROPX" command is executed, it will get the "PAR 3" from the conditions and drop that object number in the player's room.

The last command is to display the message "Got em!!". In our message data base section above, "Got em!!" is message 5. So, the command would be "MSG5".

Now our action entry is finished. Summarizing, we have:

Conditions	Commands
HAS 1	X->RM0 with a PAR 2
IN/W 2	DROPX with a PAR 3
	MSG5

Note that the first two commands have parameters associated with them. These parameters need to be placed in the conditions. To keep the order of the parameters correct, we will work from the top command down.

The first command uses a "PAR 2", so we place that parameter after the last condition listed ("IN/W 2"). The second command uses a "PAR 3", so we place that one after the first parameter (PAR 2). Our list should now be:

Conditions	Commands
HAS 1	X->RM0
IN/W 2	DROPX
PAR 2	MSG5
PAR 3	

Note that we have only used four of the five conditions possible. As stated above, unused conditions should be entered as a "PAR 0". This parameter is displayed as simply a "0".

Also note that we used only three of the four commands possible. As stated above, unused commands should be entered into the adventure editor program as a "0". The "0" command is displayed as a "-".

Putting our verb, noun, conditions and commands together, we get the following action entry:

SHOOT	BUFFALO	HAS 1	IN/W 2	PAR 2	PAR 3	0
		X->RM0	DROPX	MSG5	-	

With a little practice writing action entries becomes very simple.

BIT FLAGS

There are thirty-two bit flags available to the user, numbered from 0 to 31. These are "true/false" types flags available for use by the adventure author. They are normally used to "flag" when a certain situation has occurred. Examples are given in Chapter 5.

When the adventure is started, all bit flags are cleared. There are commands to set and clear them as well as conditions to test their status. Two bit flags are reserved by the adventure driver:

15) If this bit flag is set, it is dark outside. The room will be in darkness unless the artificial light source is available. The artificial light source is discussed in the OBJECT section of this chapter. There are two commands (DAY and NIGHT) to clear and set this bit flag.

16) When this flag is set, the artificial light source has run out. The "FILL" command will clear this flag and set the time limit to its original value.

Bit flags are tested in the conditions and set/cleared by the commands of an action entry. For example, to test if bit flag 5 is set, the condition would be "BIT 5". If bit flag 5 is to be set, a "PAR 5" would have to be in the conditions, and a "SETZ" in the commands. Moreover, if bit flag 5 was to be tested to see if it was reset, the condition would be "-BIT 5".

INITIALIZATION ACTION ENTRY USING A BIT FLAG

An example using a bit flag might be helpful at this point. When an adventure is started, usually some initialization takes place. Normally this initialization should occur only one time, right at the start of the adventure. In our example, the initialization procedure will simply display an opening message. A bit flag will be used to make sure we display the message only once.

As stated before, all bit flags are cleared at the start of an adventure. Therefore, a "-BIT" (test if bit flag is cleared) condition will always be true at the start of an adventure. We will use this convention in our initialization action.

A sample data base will need to be set up for this example. Our initialization action will be action entry 0. For the bit flag, we will use number 31. Note that bit flag 15 or 16 were not used since their use is predefined as shown above. Our sample opening message will be:

1: Welcome to the BIT FLAG example adventure!!

To test if bit flag 31 is cleared (which it will be at the start of the adventure), the condition code would be "-BIT 31". Since we want this action to be executed only once, bit flag 31 will have to be set and never cleared by the rest of the adventure. As long as bit flag 31 is set, the "-BIT 31" condition will be false, and the commands of the initialization action will not be repeated. To set bit 31, we need a "PAR 31" in the conditions and a "SETZ" instruction in the commands.

The commands of this initialization action entry will be:

1) Set bit flag 31.
2) Display the opening message (message 1).

Summarizing all of our conditions and commands, we get the action entry:

AUTO 100	-BIT 31	PAR 31	0	0	0	
	CLRZ	MSG1	-	-		

Note that the verb, noun combination of this action entry is an "AUTO 100". This is an automatic action entry that is considered 100 percent of the time. Since this is an action to be considered regardless of what the player inputs, it needs to be an auto action. Also, we want the initialization action to be considered every time. If the probability of the action was only 50 (or consider it 50 percent of the time), then there would be a 50 percent chance the opening message would not be displayed at the start of the adventure. It might be displayed after many player inputs.

The "AUTO 100" also means that this action will be considered before every player input. But if bit flag 31 is set, then the "-BIT 31" condition of this action will be false and the commands of action 1 will not be executed. This is a very common procedure for displaying opening messages in adventures.

COUNTERS

The counters are values which may be incremented, decremented, assigned values by commands as well as be tested against a number for numeric conditions. The primary counter is labeled CT. There are alternate counters which may be switched with the CT counter in order to operate on other numbers. Why would we want to switch alternate counters with CT? The only counter which may be directly acted upon is the CT value.

For example, you can not test directly if alternate counter 4 is equal to 10. First, you would have to switch the contents of alternate counter 4 and CT. Then test if CT (after the switch) is equal to 10.

When the adventure is started, CT is not assigned any particular value. There are eight alternate counters available numbered 0-7. Alternate counter 8 is the current time limit status.

Counters are very useful in adventures. Applications include counting the number of times a situation has occurred. Scott Adams uses a counter in his adventure 6 (Strange Odyssey) to keep track of the number of moves you've had a space suit on. The space suit has a limited air supply so the number of moves you've drawn air from its tanks is counted. When this value is decremented from its starting value down to zero, the player has run out of oxygen and dies.

Our example will involve a six shooter. Originally this gun has six shots. However, each subsequent shot subtracts one from a counter holding the number of shots left. Our scenario is that you shoot a Buffalo with the gun. If you have shots left in your gun when you shoot the Buffalo ("SHOOT BUFF"), you will kill it and get some "dead meat". Other actions include "EXAM GUN" which will tell you the number of shots left in the gun. A "SHOOT ANY" action is used to subtract from the shots left when you shoot something other than the Buffalo. Since at the start of the adventure all counters are not set to any predetermined value, an initialization action should set our shot count to 6. In this case we will use alternate counter 3 for our shot counter. The initialization action could be something like this:

```
AUTO  100   -BIT 1    PAR 1     PAR 3      PAR 6     0
            SETZ      EXM,CT    CT<-N      CONT
AUTO  0     PAR 3     0         0          0         0
            EXM,CT    MSG8      -          -
```

The "AUTO 100" verb, noun of the first action indicates that this action should be considered 100 percent of the time. If this is the start of the adventure, all bit flags will be cleared. If bit flag 1 is cleared, this action is executed ("-BIT 1" is the only condition which could be "false" - "PAR" conditions are always "true"). Since all bit flags are cleared at the start of an adventure, this bit will be cleared when the game is started. Bit flag 1 is subsequently set so this action will not be executed again. Since the next time the automatic actions are scanned, bit flag 1 will be set and the condition "-BIT 1" will be false so none of the commands will be executed.

The "EXM,CT" command and the associated parameter 3 exchange the CT counter value with alternate counter 3. Next, the CT value is set to 6 with the "CT<-N" command (and a parameter of 6). The CONTinue flag is set since more commands are required than are available in one action entry. The next action switches CT and alternate counter 3 back again. Then our friendly opening message 8 is displayed. Message 8 could be any message the adventure author chooses. In fact, it doesn't even need to be displayed, but most adventures start out with some sort of message. The title of the adventure and author credits are usually given.

The "EXM,CT" commands were necessary since TAS allows only testing, setting and arithmetic to be done to the CT counter. The other alternate counters (0-7) can not be set or tested directly, they must first be exchanged with the CT counter.

Before giving the rest of the actions in this example, the objects and messages used are defined:

Objects:

3	Buffalo
4	Dead meat/MEA/
5	Six shooter/GUN/

Messages

8	Welcome to example adventure!
9	GOT HIM!
10	BANG!!
11	It's a fine looking thunder stick! It has
12	shots left.
13	Click
14	No buffalo here

These objects and messages will be used in the rest of the action entries. Objects 4 and 5 may be carried since they are named objects. Named objects (notice the name between the slashes at the end of the object description) are explained in the OBJECT section of this chapter. For now, just accept the fact that these objects can be picked up and dropped without action entries to perform these functions.

The actions used in this example are:

10: SHOO	BUFF	IN/W 3	HAS 5	PAR 3	0	0
		EXM,CT	CONT	-	-	
11: AUTO	0	CT=0	0	0	0	0
		MSG13	-	-	-	
12: AUTO	0	CT>0	PAR 4	PAR 3	0	0
		MSG10	MSG9	EXX,X	CT-1	
13: AUTO	0	PAR 3	0	0	0	0
		EXM,CT	-	-	-	
14: SHOO	BUFF	-IN/W 3	0	0	0	0
		MSG14	-	-	-	

Action 10 is evaluated if the player types "SHOO BUFF". The conditions are "IN/W 3" and "HAS 5". These conditions check if the player is in the same room as the buffalo and if he is carrying the gun. If either of these conditions are false, then this "SHOO BUFF" action is skipped and the next "SHOO BUFF" or "SHOO ANY" action is considered. For now let's assume the conditions are true (the player is in with the Buffalo and is carrying the gun). The commands of action 10 will exchange counter 3 and the CT value. The CONTinue flag is also set.

Action 11 is considered next. Notice that the verb, noun combination is "AUTO 0". This action is a CONTinuation of action 10. If the CT value is equal to zero (no more shots left), then message 13 is displayed (Click). Regardless if action 11 was true or false, action 12 is also considered. This is the nature of CONTinued actions.

Action 12 checks if the CT value is greater than zero (some shots are left in the gun). If there are some shots left, messages 10 and 9 are displayed (Bang! Got him!), the location of the Buffalo and dead meat are switched (the dead meat would have originally been in the storeroom) and one is subtracted from the CT value.

The order of actions 11 and 12 was not arbitrarily selected. If the order were switched and the shot counter would have had one bullet left, subtracting one from it would set it to zero. Then action 11 would have "seen" a zero CT value and displayed the message "Click". It is intended that either one of these CONTinued actions will be true, not both. Switching their order raises the possibility that both could be true.

Action 13 is used to switch the CT value and counter 3 back again. Whenever CT and an alternate counter are switched, care should be taken to switch them back again. This is the reason that the second exchange was done on a separate action in this example. Since there were no conditions in action 13, it would always be executed if action 10 were true.

If the player typed "SHOO BUFF" and the conditions of action 10 were not met, then action 14 would be considered. The condition of action 14 is a "-IN/W 3". Therefore, if the player is not in the same room as the Buffalo, this action is true and its commands are executed. If this is the case, then message 14 is displayed (No buffalo here).

An "EXAM GUN" action could be used to find out how many shots were left in the gun. It could read:

```
20: EXAM GUN     AVL 5     PAR 3      0          0          0
                 EXM,CT    CONT       MSG11      DSPCT
21: AUTO  0      PAR 3      0         0          0          0
                 EXM,CT     MSG12     -          -
```

Action 20 is executed if the player is either holding onto the six shooter or is in the same room as the gun. If this is true, then counter 3 is exchanged with the CT value. Message 11 is displayed next (It's a fine looking thunder stick. It's got). The number of shots left is then displayed by the "DSPCT" command (CT holds the number of shots left after the EXM,CT command of action 20). The CONTinue flag is also set. Action 21 is used to exchange CT and counter 3 back again. Message 12 is also displayed (shots left).

Some action is also needed so something is displayed when the gun is shot. There are two cases involved. First, if the gun is empty, the message "Click" is displayed. Second, if the gun has some shots remaining, the message "BANG!" is displayed and one is subtracted from the shot counter. These actions could be:

```
96: SHOO ANY     HAS 5     PAR 3      0          0          0
                 EXM,CT    CONT       -          -
97: AUTO  0      CT=0      0          0          0          0
                 MSG13     -          -          -
98: AUTO  0      CT>0      0          0          0          0
                 MSG10     CT-1       -          -
99: AUTO  0      PAR 3     0          0          0          0
                 EXM,CT    -          -          -
```

Action 96 checks if the player is holding onto the gun. If he is, then CT and counter 3 values are exchanged and the CONTinue flag is set.

Action 97 is considered next. If there are no shots left, then message 13 is displayed (Click).

Action 98 checks if any shots remain. If the number of shots left is greater than zero, then message 10 is displayed (BANG!) and one is subtracted from the shot count.

Action 99 is used to exchange CT and counter 3 back again. Note that alternate counter 3 and CT were switched back again. This fact can not be over emphasized. It is extremely important to always switch the counters back. It will not cause your computer to Self-destruct, but you will almost certainly have problems with subsequent use of counters.

Notice that the "SHOO BUFF" action (actions 10 and 14 given above) preceded the "SHOO ANY" action (action 96 given above). If the "SHOO ANY" action preceded the "SHOO BUFF" action, the latter would never be executed. If the player is holding onto the gun when he types "SHOO BUFF", the "SHOO ANY" action would be found true and the "SHOO BUFF" action would never be reached (if their order was reversed). It is good practice to put the more "limited" actions before the less "limited" ones.

For more information on using the counters, see Chapter 5.

ALTERNATE ROOM REGISTERS

The value of the current room (CR) may be saved and restored by exchanging it with an alternate room register. The saved room value may be restored by performing another exchange with the same alternate room register.

Alternate room registers are another potentially confusing facet of the adventure data base format. There are currently six alternate room registers available numbered 0-5. The main purpose of the alternate room registers is to save a player's current location for recall at a later time. As a side note, don't worry much about the alternate room registers. None of the over three dozen adventures received at The Alternate Source for review have used alternate room registers. In other words, they are very esoteric commands. But the need for them may arise at some time. So if you don't understand their use right away, don't be discouraged. You can write a good adventure without using them.

As an example, suppose you had an adventure with days being accounted for (night and day). After so many moves the player will fall asleep. When the player falls asleep, he has a dream. This dream could be a hint of upcoming attractions. In this case, the dream will move the player to a "hint" room, display the hint room, then move him back to his original room.

A "GOTOY" command in the action entries with the appropriate parameter could move the player to the "hint" room, but there is no way a "GOTOY" could send him back to the original room. The room number to be returned to would change depending on which room the player was in before he had his "dream".

For example, the player could fall asleep while in room 1 or room 5. If the dream room was 10, a "GOTOY" with a "PAR 10" would send him to the dream room. However, we couldn't send the player back to the room he was in before the dream since a second "GOTOY" command would need a "PAR" room number to send him to. This could be room 1 ("PAR 1") or room 5 ("PAR 5").

In this case, the alternate room registers can come to the rescue. The player's current room number could be stored in an alternate room register before moving him to the "hint" room. The commands for using the alternate room registers are "EXRM0" and "EXC,CR". In this example, we will use alternate room register 0 thus utilizing the command "EXRM0".

The scenario to demonstrate this goes as follows: the player is currently in room number 5. He will fall asleep and be temporarily sent to room 17 (the "hint" room). After a delay (via the "DELAY" command), he will return to his original room number or room 5. For simplicity sake, bit flag 1 will be set if the player has just fallen asleep. The actions to perform this could be:

```
AUTO 100    BIT 1     PAR 1     PAR 17     0          0
            CLRZ      EXRM0     GOTOY      CONT
```

This action will test if bit flag 1 is set. If it is set, then the player has just fallen asleep. NOTE: in this example, some actions preceding this one would have set the flag if the player had fallen asleep.

Since all conditions in this action entry were true, the commands would be executed. In this case, bit flag 1 is cleared and the player's current room is exchanged with the room number currently stored in alternate room register 0.

It should be noted at this point that after the exchange, the player's room number will be a bogus value unless the alternate room register had been previously set. At this point the player is sent to room 17, or the "hint" room. The continue flag is also set. The subsequent action entry could be:

```
AUTO 0      0         0         0          0          0
            DSPRM     DELAY     EXRM0      DSPRM
```

This action will display the player's current room, which at this point is room number 17 (the "hint" room). The "DELAY" command will pause the display for about one second. After the pause, the player's room number before he fell asleep is recalled by doing another "EXRM0". This command will exchange his current room number (17) with the room number held in alternate room register 0 (room 5 at this point). The "DSPRM" command will display the player's current room which at this point is room number 5.

SETTING ALTERNATE ROOM REGISTERS

There are no obvious applications for setting the room registers but an explanation will be given anyway. Just because there are no obvious applications does not mean there aren't any. Who knows, you may be the first to come up with one!

Suppose we are currently in room number 5 and we want to set alternate room register 2 to room 10 and alternate room register 5 to room 23. The action to do this could be put in some initialization routine, such as displaying the opening message, etc. The example here assumes this case with message 8 being the opening message:

```
0: AUTO 100   -BIT 1     PAR 2     PAR 10    0         0
              EXC,CR     GOTOY     CONT      MSG8
1: AUTO  0    PAR 2      PAR 5     PAR 23    PAR 5     0
              EXC,CR     EXC,CR    GOTOY     EXC,CR
2: AUTO  0    PAR 1      0         0         0         0
              SETZ       -         -         -
```

Action 0 is the initialization action entry. When adventures are started, all bit flags are cleared. Therefore, at the start of this adventure, a "-BIT" condition will always be true. If in the same set of initialization action(s), the bit flag is set, the initialization action(s) will not be executed again.

The first "EXC,CR" with an associated parameter of 2 exchanges the player's current room with alternate room register (or arr) 2. In this case, alternate room register 2 will hold a 5 after the switch since the player's current room was 5. The "GOTO 10" ("GOTOY" command and a parameter of 10 - this shorthand will be used from this point on) sets the player's room number to 10. The "CONT" is used to tell the adventure driver program that this action entry will CONTinue on with all subsequent "AUTO 0" actions. Message 8 is displayed as some sort of opening message.

Action 1 exchanges the player's current room and alternate room register 2 again. Before the switch, alternate room register 2 held 5 and the current room was 10. The switch sets alternate room register 2 to 10 and the current room to 5. Thus, alternate room register 2 is set as we originally wanted.

Next, alternate room register 5 and the current room are exchanged. This sets alternate room register 5 to room number 5. The "GOTO 23" sets the current room to 23. The last "EXC,CR" switches alternate room register 5 and the current room back again. This second exchange sets alternate room register 5 to 23 and the current room to 5.

Our task is now completed except for one detail, the initialization bit flag must be set so this set of actions are not executed again. Action 2 accomplishes this by setting bit flag 1.

VOCABULARY entries

The vocabulary consists of a list of verb strings and noun strings. These are the words the player may use in the adventure. Synonyms are handled by beginning the word with an asterisk, which is then treated the same as the first previous word without an asterisk. Examples of primary vocabulary words and synonyms are in the following table:

Vocabulary	Synonym
GET	*TAKE
TAPE	*CASSETTE
DOG	*HOUND

A primary vocabulary word may have more than one synonym. For example, in the following list of verbs, the verb "GET" has three synonyms:

```
     #    Verb

    10:   GET
    11:   *TAKE
    12:   *GRAB
    13:   *SNATCH
```

If there is an action entry which uses the verb "GET", the player may type "GET" or any synonym of "GET" for his inputted verb to match the action entry's verb, noun combination. This is what makes synonyms powerful. For example, suppose there was an action entry with the verb, noun combination of "GET PLOW". If the player input "GRAB PLOW", the adventure driver would see the player input as "GET PLOW" since "GRAB" is a synonym of "GET". This eliminates having duplicate action entries just to allow the player to use two similar verbs.

There are really two lists of vocabulary words, one for the verbs and one for the nouns. The adventure editor (ADVEDT) references these words by number. This reference number is the position in the vocabulary list of the word in question. Notice how the short list of verbs above ("GET", "*TAKE", etc.) were numbered. They are referenced by the ADVEDT program by that number. For example, if you wanted to change verb 11 from "*TAKE" to "*LIFT", the adventure editor would be told that verb 11 is to be changed.

Some of the vocabulary entries are predefined by ADVENTURE and SHOULD NOT be changed. These predefined verbs and nouns are listed below:

Verbs

Vb# Verb

0 AUTO This is not entered by the player while playing the adventure. It is the verb entered when typing in the auto action entries which are all evaluated before a valid player input.

1 GO This is a special case for the direction nouns 1-6.

10 NOUN This is used to pick up objects if there is no action entry that applies and the noun matches the object name enclosed in slashes of an object in the current room. See the OBJECT section of this chapter for more information on the object name.

18 DROP This is used to drop objects if there is no action entry that applies and the noun matches the name enclosed in slashes of an object being carried.

Nouns

Nn# Noun

0 ANY This is not entered by the player while playing the adventure. It denotes the action entries which can match any noun (or no noun).

1 NORTH This is reserved for the first room direction entry with verb 1.

2 SOUTH This is reserved for the second room direction entry with verb 1.

3 EAST This is reserved for the third room direction entry with verb 1.

4 WEST This is reserved for the fourth room direction entry with verb 1.

5 UP This is reserved for the fifth room direction entry with verb 1.

6 DOWN This is reserved for the sixth room direction entry with verb 1.

The vocabulary words are used by the action entries (as was shown numerous times above when action entries were described). To see an actual list of vocabulary words, take a look at the adventure data base listed in Chapter 5.

The word length of the adventure affects the vocabulary words. If the word length of the adventure is four, then only the first four character of the verbs and nouns are significant. For example, the noun "BUFFALO" appears as "BUFF" to the adventure driver.

One thing to consider when writing the vocabulary words is word duplication. When the player inputs his verb, noun to the adventure driver, that program scans the vocabulary words for a match from the first word in the list to the last. Suppose the word length of an adventure is three. The nouns "KITE" and "KITCHEN" are in the noun vocabulary list. If the word length is three, both of these nouns appear to the adventure driver as "KIT". Suppose "KITE" precedes "KITCHEN" in the noun list, and the player inputs something like "GO KITCHEN". The verb, noun combination would appear to the adventure driver as "GO KIT" since the word length is three. When a scan is made of the nouns to see if "KIT" is a legal noun, the first match is made with "KITE". If the action entry which allows the player to enter a kitchen has a verb, noun combination of "GO KITCHEN", then the match will not be made. Since the noun match was with "KITE", the adventure driver program would be looking for a "GO KITE" action entry and will never "see" a "GO KITCHEN" action entry.

The solution to this potential problem is to use vocabulary words that have at most the number of word length characters in them. For example, if the word length was three, you would enter the noun "KITE" as "KIT". If both "KITE" and "KITCHEN" had been entered in the noun list as "KIT", the problem discussed above would not happen.

It is important to look at your vocabulary words for these word duplication problems. Another thing that should be noted is that two primary nouns or verbs can not have the same synonym. When a scan is made of the vocabulary and the player's verb and/or noun is a synonym, only the first occurence of the synonym will be matched. If that synonym is used for a primary noun or verb further down in the vocabulary list, it will never be matched. For example, look at the following nouns:

Noun#	Noun
10	PISTOL
11	*GUN
12	RIFLE
13	*GUN

If the player typed in "SHOOT GUN", the adventure driver would scan the nouns for a match of "GUN". The first match would be with noun 11. Since noun 11 is a synonym, the first preceding primary noun is used. In this case, it is noun 10, the "PISTOL". As you can see, the "GUN" synonym of "RIFLE" will never be matched.

The vocabulary words must be in all upper case letters. The adventure editor program, ADVEDT, will force them to upper case for you so this is not a potential problem.

ROOM entries

The room entries consist of the room number of the adjacent room(s) in the six reserved directions North, South, East, West, Up and Down plus a room description string. If the adjacent room number is zero, there is "no obvious exit" in that direction.

Rooms in The Adventure System are referenced by number. For example, a room may be described as a "Jail Cell". However, it is referenced by its numeric position in the list of rooms. When the adventure is played, the adventure driver refers to the player's room as a number. Take the following room as an example:

Room #	N	S	E	W	U	D	Room description
5:	0	2	0	15	0	0	Closet

If the player is in room 5 (the room described above), he may move SOUTH or WEST. Since a zero is the adjacent room in the N, E, U and D directions, he may not move in those directions. If the player moves SOUTH, he will move to room 2. If he moves WEST, he moves to room 15.

If the text description of the room does not begin with an asterisk ("*"), the adventure driver program will precede the room description with "You're in a"; otherwise, it will just display the description minus the asterisk. In our above example, the room description would be displayed as "You're in a Closet". However, if the room description were:

*I'm at the entrance of a large cave

The room description would be displayed as:

I'm at the entrance of a large cave

Room zero is reserved as a storeroom for objects currently not in any room. The storeroom is typically used for storing objects the player has not found yet or objects that the adventure author does not want the player to see at this time.

The player can not get to room zero by using one of the reserved directions (he can not move SOUTH, etc. into room zero). Actions usually do not permit the player to enter this room.

The last room is reserved for some sort of limbo state should the player die. This is where the player is sent with a DEAD command. It may or may not contain exits back to the other rooms. The last room number is contained in the HEADER section of the adventure.

When the adventure driver displays the room discription and there are objects in the current room, it will add ". Visible item:" after the room description. This should be considered when deciding on your room descriptions so this message is not "broken up". By "broken up", I mean part of the text being at the end of one line and the rest at the beginning of the next line. For example, consider the following room description as displayed by the adventure driver.

I'm at the fork of a road with a fence blocking a trail. Visi
ble items:

Rattlesnake.

Notice how the word "Visible" is "broken up". If room descriptions are well thought out, this "breaking up" of words will not happen.

The room description should not be ended with a period or other character of this type. The visible items message is preceded by a period, so ending a room description with a period will cause two periods to end the description.

MESSAGE entries

The messages consist of a string of characters for each message to be displayed by the action entries. Entry 0 should always be left as a null string and can not be displayed by the commands of an action entry. Recall that a message 0 was used by the action entry commands as a null or unused command. That is why message 0 can not be displayed. The messages are like the text part of BASIC PRINT statements.

As stated before, the messages are the communication medium between the adventure and the player. The highest numbered message number is 99.

Messages can be up to 255 characters. The <DOWN ARROW> key can be used to start a new line on the display. For example, consider this message:

```
WANTED:  DEAD OR ALIVE <DOWN ARROW>
    The Dalton Gang <DOWN ARROW>
 REWARD:   $5,000!! <ENTER>
```

OBJECT entries

The object entries consist of a text description of the object and the starting room number of the object. Room zero is used as the starting room for objects not found at the beginning of an adventure. A minus one (-1) is used for the starting room of an object which the player is carrying at the beginning of the adventure.

The object descriptions should begin with an asterisk if that particular object is a treasure. For example, an object description of "*GOLD NUGGET*" denotes a treasure. The SCORE action entry command checks for objects whose current location is the same as the treasure room defined in the HEADER. If an object is in the treasure room, the first character of its description is checked. If the first character is an asterisk, the object is a treasure. The number of treasures is computed by the SCORE command in this manner.

Also, if the object is to be picked up and dropped, a descriptive name for the object is enclosed in slashes at the end of the description. The word between the slashes must be the same length or smaller than the word length of the adventure. It must also be in all upper case letters. If the player's verb is "GET" or "DROP" and no other action applies, the adventure program will automatically pick up or drop the object if the player's inputted noun is the same as the object name. Take the following object as an example:

```
Obj#   Start Room   Description

 5         15        Hammer/HAM/
```

At the start of the adventure, this object is placed in room 15. The object is described as a "Hammer". Note the "/HAM/" at the end of the object description. This is the "Object Name" described above. Since this name is three characters long, the word length of this adventure would probably be three.

If the player types "GET HAM" and there are no "GET HAM" or "GET ANY" action entries, the "Hammer" is picked up by the player if the carry limit has not been exceeded and the "Hammer" is in the same room as the player. Moreover, if the player is carrying the "Hammer" and he types "DROP HAM", the "Hammer" will be dropped. These "GET" and "DROP" procedures are done automatically without the use of action entries.

The name of the object must be a noun in the list of vocabulary entries for the automatic pick up and drop feature to work. The object name must also be a primary noun, not a synonym. The purpose of this feature is to reduce the number of action entries required in an adventure. Without this pick up/drop feature, an action entry would be required to pick up and drop every object.

More than one object may have the same object "name". The adventure driver program will scan from the lowest numbered object to the highest numbered object for a "name" that matches the player's noun. If a match is made and the location of the object is the same as the player's current room, the object is picked up, provided the carry limit is not exceeded. If the location of this object is not the same as the player's room, this object is skipped and the scanning procedure continues. The following is an example of multiple object names:

Obj#	Current Room	Description
3	10	Security Badge/BADG/
4	5	Visitor's Badge/BADG/
5	3	Marshall's Badge/BADG/

Note that the object names of objects 3-5 are the same, "/BADG/". Suppose the player was currently in room 5 and was not holding any of these badges. If the player typed in "GET BADG", the automatic get procedure would be started. The first object name that matches the player's noun ("BADG") is object 3. However, object 3 is not in the player's current room (room 5) so this object is skipped.

The next object name that matches the player's noun is object 4. This object is in the player's current room (room 5) so it is picked up (provided the carry limit is not exceeded).

Suppose the player was currently holding every one of the badges. If he typed "DROP BADG", the automatic drop feature would be started. The first object being carried by the player with the name "BADG" is object 3, so this object is dropped. The next "DROP BADG" would drop object 4, etc.

An example of a treasure which can be picked up is:

FIRESTONE (cold now)/FIR/

Which can be picked up by the word "FIR" ("GET FIR"). Before the firestone is cooled, the treasure was in the storeroom and the following object was in the room:

Glowing *FIRESTONE*

Because this object does not begin with an asterisk, it is not recognized as a treasure. Also, it can not be picked up since it has no object name (name between slashes). The action that cools the firestone exchanges the locations of these two objects.

The action entries can allow objects which do not have object names to be picked up. For example, suppose there were the following two objects in an adventure:

Object#	Object description
25	Red hot poker
26	Wet towel/TOWEL/

The player can pick up and drop object 26, the towel, with no action entries since it is a "named" object. However, object 25, the red hot poker, can not be picked up without action entries because it is not "named". There may be an action entry which requires the player to be holding onto the wet towel before he can pick up the red hot poker. The action entry controlling this could be as follows:

GET POKER	IN/W 25	HAS 26	PAR 25	0	0
	GETX	-	-	-	

The "IN/W 25" condition makes sure the red hot poker is in the player's current room. The "HAS 26" condition checks to see if the player is holding onto the wet towel. If both of these conditions are true, then the command "GETX" is executed. This command will get object 25 which is the red hot poker.

When objects are displayed by the adventure driver, their description is ended by a period. Take the two objects described above for example:

I'm in a kitchen. Visible items:

Read hot poker. Wet towel.

Notice how both of these objects are followed by a period. The adventure driver program does this automatically.

ARTIFICIAL LIGHT SOURCE

Object number nine (9) is reserved as the artificial light source in its lighted state. The adventure driver program checks to see if object 9 is available when a room is in darkness (NIGHT).

If a room is in darkness (Bit flag 15 set or the NIGHT command is issued) and the artificial light source is present, then the room is displayed in its lighted state. If the artificial light source is not present in a dark room (set bit flag 15 or NIGHT command), then the only information given about the room is: "It's too dark to see!". The room description, objects and obvious exits are not displayed if the room is in darkness. If the player moves in an illegal direction in the dark (a non obvious exit direction), he is killed.

The time limit value in the HEADER section is the maximum number of moves the artificial light source will last before it runs out. This time limit counter (alternate counter 8) is automatically decremented by the adventure driver every time the player takes his turn. If the time limit is decremented to zero, the artificial light source is "snuffed out" and sent to the storeroom. Therefore, if the player was in a dark room (Bit flag 15 set or the NIGHT command has been issued), and the artificial light source goes out, the light source would be sent to the storeroom and the player would be left in the dark.

The FILL command AGETs object 9 and sets the time limit counter back to the time limit value held in the HEADER section. Examples of object 9 are a lit flashlight and a lit lamp. Chapter 5 contains the listing of an adventure which uses an artificial light source. It may be helpful to review this chapter for more information on the artificial light source.

TRAILER

The trailer information contains the version number, the adventure number and a security checksum. If the version number is 415 it will be displayed as "4.15" when the HEADER section is listed. The adventure number is simply the number identifying the adventure. The security checksum is 2 * #actions + #objects + version#. If the checksum computed by the adventure driver program does not equal the one in the adventure file, the adventure driver program will not allow the game to be played since it contains bad data.

The TRAILER is controlled and updated automatically by the adventure editor program, ADVEDT. This data base section is transparent to the adventure author and player. ADVEDT will compute and use the proper values when a data base is saved to disk or tape.

Chapter 3

ADVENTURE DRIVER INSTRUCTIONS

This chapter will give the rules governing the entering of an adventure with the adventure editor, ADVEDT. These are rules pertaining to the ADV adventure driver program, not the adventure editor (ADVEDT). Rules are broken down by data base sections such as action entries, vocabulary, etc.

Rules for the Action entries are as follows:

1) All of the automatic actions must precede player input actions. Any auto-actions not placed before the player input actions will be ignored.

2) If the action entry uses commands which require parameters, there must be parameters in the condition line. If not, the results will be unpredictable.

3) The verb and noun of an action entry must be a primary noun, not a synonym.

4) The range of values which accompany a condition code (i.e. the 10 of an "IN 10" condition) must be in the range 0 to 1600 inclusive.

5) The value which accompanies a condition code must reference a "real" item. For example, if there is a condition code like "HAS 75", there must be an object 75. The same goes for rooms, bit flags, counters and messages.

Rules for the vocabulary:

1) The predefined verbs and nouns (NORTH, GET, etc.) must remain in their preset positions. Failure to do so might cause difficulty in moving from room to room and/or carrying objects.

2) This is not a rule, but a strong suggestion. When keying in verbs and nouns with ADVEDT, make your vocabulary words the same length or shorter than the word length specified in the header. This will make unintentional duplicate words easy to find.

Duplicate nouns and verbs can be a potential problem. For example, suppose noun 10 was SHED and noun 11 was SHELF. If the word length of the adventure was three, SHED and SHELF would appear to be the same noun, SHE (the word is truncated at three characters). This is where the problem occurs. If an action entry refers to SHELF (i.e. EXAMINE SHELF), and the player types in "EXAM SHE", the adventure driver program starts scanning the list of nouns

Since the noun number the adventure driver is looking for (11) is not the same as the one it found (10), the action entry is never considered (the verb, noun combinations of the player and action entry do not match). By limiting the length of inputted nouns and verbs to the word length of the adventure, this problem will never crop up.

3) If a word is to be a synonym, it must be preceded by an asterisk and placed immediately after the primary noun or verb. For example, if GO is the verb and RUN, WALK and ENTER are to be synonyms, the list of verbs would read GO, *RUN, *WALK and *ENTER.

4) Vocabulary words must not contain imbedded asterisks. Any other character may be used in a vocabulary word. Examples are "$20", "30", etc. Vocabulay words must be upper case letters and/or special characters. Special characters are "!", "#", "$", etc.

Room rules are as follows:

1) The header contains the number of rooms, the last being used to send the player to after a DEAD command. The last room should be some sort of limbo state or something similar. A common last room text description is "lot of trouble". If the player ends up in this room, the room is described as "I'm in a lot of trouble".

2) Each room has six values associated with it. These are the room numbers which are entered on a direction command (i. e. "GO NORTH"). These values should be legal room numbers (ADVEDT won't let a bad number be entered). A zero is used if no exit is possible in that direction.

3) Room number 0 is reserved for a storeroom. Objects which have not been found, or are being hidden from the player are normally placed here.

The rules for the Objects are as follows:

1) The starting room for an object should be a valid room number. Objects not found yet or not used at the beginning of the adventure should be in room zero (the storeroom). Objects the player is carrying at the start of the adventure should have a starting room of minus one (-1).

2) An object name is placed between slashes at the end of the object description if that object is to be carried and dropped via the GET and DROP verbs (with no action entries). The name should be the same length or less than the word length of the adventure.

3) The object name must be a primary noun in the vocabulary list, not a synonym. If the object name is a synonym, the pick up and drop feature will not work for that object. If the player's noun is a synonym, the associated primary noun is found and that primary noun is used when trying to match an object name. This is why the object name must be a primary noun.

4) The first character of an object description must be an asterisk for the object to be recognized as a treasure. Otherwise, the SCORE command will not recognize the object as a treasure.

5) Object number 9 is reserved as the artificial light source in its lighted state. The time limit value held in the HEADER is decremented each time this object is not in the storeroom (room 0).

6) The object name must be all capital letters.

GENERAL ADVENTURE DRIVER RULES

The following is a description of the messages displayed and tasks performed automatically by the adventure driver.

1) If the player is in the dark and he moves in an illegal direction (i.e. NORTH when there is no obvious NORTH exit), the player is killed, the room is made light, and the player is moved to the last room number. The message "I fell and broke my neck!" is displayed.

2) If the player is in the dark and he moves in a legal direction (i.e. NORTH when there is an obvious NORTH exit), the message "It is dangerous to move in the dark" is displayed. The player is not penalized for moving in the dark as long as it's a "legal" direction.

3) If the player's room is light and he moves in an illegal direction, the message "I can't move in that direction" is displayed.

4) If the artificial light source runs out, the message "The light ran out" is displayed and the artificial light source is moved to the storeroom.

5) If the artificial light source is not in the storeroom and the number of "turns" before it runs out is between 1 and 24, the message "Light runs out in XX turns!" is displayed.

6) If the player tries to GET an object and he is already at the carry limit, the message "I'm carrying too much . . . Take inventory!" is displayed.

7) If the action entry command DEAD is issued, the message "I'm dead" is displayed.

8) If the action entry command FINI is issued, the game is ended and the message "This game is over. Play again?" is displayed.

9) If the action entry command SCORE is issued, the treasures are counted and the message "I've stored XX treasures. On a scale of 0 to 100 that rates YY" is displayed.

10) If the player stores all of the treasures in the treasure room and the SCORE command is executed, the message "Congratulations! You finished it all!" is displayed.

11) If the action entry command INV is issued, the message "I'm carrying the following:" is displayed plus a list of the objects being carried.

12) If the SAVE command is executed, the message "Saving game. Which number (0-9) ?" is displayed.

13) If the player's inputted verb and noun were both in the vocabulary (legal entries), and there is no action entry with those two words together, the message "I'm sorry, but I don't understand what you mean" is displayed. For example, "GET" is a legal verb and "NORTH" is a legal noun. If the player types in "GET NORTH" and there is no "GET NORTH" action entry, this message is displayed.

14) If the player's input matches one or more action entries but all action entries were false (none were executed), the message "I can't do that . . . yet!" is displayed.

15) If the player types "GO" with no direction, the message "Give me a direction too" is displayed.

16) If the player's inputted verb is not in the vocabulary, the message "I don't know how to "verb"something." is displayed.

17) If the player's inputted noun is not in the vocabulary, the message "I don't know what a "noun"is." is displayed.

18) If the player tries to GET an object he is already carrying, the message "I'm already carrying it" is displayed.

19) If the player tries to GET a named object, and the object is not in the player's room, the message "I don't see it here" is displayed.

20) If the player tries to GET/DROP an object which can not be picked up or dropped, the message "I't beyond my power to do that!" is displayed.

21) If the player tries to DROP an object he is not carrying, the message "I'm not carrying it" is displayed.

22) The adventure driver automatically decrements alternate counter 8 (the time limit) if the artificial light source is not in the storeroom.

23) The adventure driver automatically does a DSPRM command if an object enters or exits the current room.

24) The adventure driver automatically does a DSPRM command if the player moves to a different room via a GOTOY command.

Chapter 4

ADVEDT Instructions

This chapter contains the instructions for the adventure editor, ADVEDT, program. The differences between the tape and disk versions are noted where applicable. Each command of the editor will be covered in detail.

DISK VERSION:

Put the ADVEDT diskette in drive 0 and turn on the computer. From the DOS command mode type:

ADVEDT

TAPE VERSION:

Enter BASIC and type SYSTEM. The file name is "A". When the computer prompts you again, type "/<ENTER>".

NOTE: There is a key frequently used within ADVEDT called the "Exit" key. This key is either the <BREAK> key or the SHIFT RIGHT ARROW keys. Some DOSes do not recognize the <BREAK> key as an inputted key. To see if your DOS does, press the <BREAK> key from DOS command mode. If nothing happens, your DOS does not recognize the <BREAK> key. If this is the case with your DOS, use the SHIFT RIGHT ARROW combination. All references henseforth will use the <BREAK> key as the "exit" key. If you must use the SHIFT RIGHT ARROW combination, just remember to substitute that combination for the <BREAK> key from now on.

If any high memory drivers are to be used, they must be loaded before the ADVEDT. They must also protect themselves by setting the high memory pointer (4049H for Model I, 4411H for Model III, 40B1H for tape version) to some value below them. Failure to do this will cause "ADVEDT" to overwrite them!

To modify or just peek at an existing data base, READ it into memory and then use the ADVEDT commands to review it. To create a new adventure simply run ADVEDT and start entering data via the MODIFY command. It is normally good practice to clear memory before entering a data base via the CLEAR command.

If a data base has previously been read in during this session with ADVEDT, use the CLEAR command to zero out that data base. This will allow you to start with a "clean slate".

The ADVEDT commands and corresponding menu keys are listed below:

R	READ an adventure data base in.
W	WRITE an adventure data base out.
L	LIST the data base.
P	PRINT (hardcopy) the data base.
M	MODIFY a data base section.
I	INSERT blanks into the data base.
X	XREF: cross reference utility
C	CLEAR a data base out.
E	END ADVEDT and return to BASIC or DOS.

Most of these commands have options within them. A description of each command is given below.

Remember, while in the editor, pressing the <BREAK> key at any time (except during disk or tape I/O) will return you to the main menu.

READ command

This command will read in an adventure data base. To activate the command, press the "R" key from the main menu.

The disk version of the program has the option of reading from tape or disk. The tape version only allows input from tape. Select the appropriate device at this point. If tape is selected and you are using a Model III, the choice of fast (1500 baud) or slow (500 baud) cassette rate is given. If a disk read is specified, simply supply the adventure name and the drive number. If the drive number is not entered, the first occurrence of the file is used. The drive number is entered by typing a colon followed by the drive number AFTER the adventure name.

The adventure name may be at most two characters from 0-9 and A-Z. If two characters are entered, such as "00", the file name will be "ADVENT/D00".

Examples of good adventure names are:

Characters	Resulting file name
A:1	ADVENT/DA:1
0A	ADVENT/D0A
ZD:0	ADVENT/DZD:0

If an error occurs while reading a data base, an appropriate error message is displayed and control returns to the main menu.

If bad data somehow gets into the data base, a "*BAD SECURITY*" message is displayed. The data base will have been read in, but the accuracy of the data can not be guaranteed.

The READ command will zero out any data base currently in memory before reading in the specified data base. This is done so no information from a previously read adventure carries over.

Note that the tape version of the program does not support named adventure files. Adventures may be distinguished by writing a name on the cassette label the data base is on.

The READ command DOES NOT respect the high memory value. If an extremely large data base is read in, it may load into high memory drivers if they are present. If this happens, load ADVEDT again without the high memory driver. It is very doubtful that a data base will load in over the high memory pointer however. The only possible way of this happening is you wrote an extremely large adventure with no high memory drivers in place. Then you tried to read it back in with high memory drivers in place.

Suppose we wanted to load an adventure data base titled "A1" with the disk version of ADVEDT. The command flow would go something like this (user inputs are underlined):

```
Do you want to READ, LIST, PRINT, XREF, WRITE, MODIFY, INSERT,
CLEAR or END
R, L, P, X, W, M, I, C or E ? R
```

(Pressing the "R" key initiates the READ command)

The following message is displayed after pressing the "R" key:

```
Tape or Disk
T or D ? D
```

(Press "D" for Disk input)

The following message is displayed after pressing the "D" key:

```
Adventure name and drive # ? A1<ENTER>
```

After pressing the <ENTER> key, ADVEDT will proceed to read the adventure in. After the adventure is read in, the program returns to the main menu.

WRITE command

The WRITE command will store an adventure data base on disk or tape. With the disk version of ADVEDT, a data base can be saved on tape or disk. The tape version allows output only to a cassette recorder. If using a Model III, the BAUD rate (1500 or 500) may be specified.

If the adventure being written out was previously read in, pressing <ENTER> for the adventure name and drive number will write the adventure out with the same adventure name and drive number. This way you do not have to continuously reenter the adventure name.

The adventure name entered must be in the same format as in the READ command. If a bad adventure name is entered, an error message will be displayed and another adventure name will be requested.

Before writing the data base, ADVEDT verifies that the HEADER is holding the correct limiting values of the number of actions, messages, rooms, vocabulary and objects. For example, suppose you entered 50 messages but the HEADER said there were only 40. Writing the data base out without checking the limits would result in some lost data (messages 41-50). ADVEDT makes a check and will write out all 50 messages.

Suppose we wanted to save an adventure data base just typed in to disk. We have chosen to call this adventure "Z1". The procedure would be:

```
Do you want to READ, LIST, PRINT, XREF, WRITE, MODIFY, INSERT,
 CLEAR or END
R, L, P, X, W, M, I, C or E ? W
```

(Press "W" for the WRITE command)

```
Tape or Disk
T or D ? D
```

(Press "D" for Disk input)

The following message is displayed after pressing the "D" key:

```
Adventure name and drive # ? Z1<ENTER>
```

LIST command

This command is used to list any part of the data base on the CRT. After "L" is entered from the main menu, a LIST sub-menu is displayed. That LIST sub-menu is:

```
Which section of the data base do you want to LIST:
HEADER, ACTIONS, VOCAB, ROOMS, MESSAGES or OBJECTS
H, A, V, R, M or O ? _
```

The section to be listed is selected by pressing the first letter of its name. For example, if "A" is pressed, the Action entries are to be listed. If the <BREAK> key is pressed, the LIST sub-menu is exited and the program returns to the main menu.

After a section of the data base is selected, the lower and upper limits to be displayed are input (except for the HEADER which lists all of its variables). If the <ENTER> key is pressed as the response to this inquiry, all of that data base section will be listed. If just one number is entered, that number is taken as the lower and upper limits. What results is that only that one item will be listed.

The listing will automatically pause after a screen full of lines have been displayed. To continue the listing, press any key except <BREAK> or <CLEAR>.

The number of items in each data base section is kept in the HEADER. This value is the upper limit used by ADVEDT when the <ENTER> key is depressed on the limit inquiry (lower and upper bounds). The MODIFY command will allow input past this value without changing the HEADER value. As a result, all items of a data base section may not be reviewed on a LIST, PRINT or MODIFY if <ENTER> is pressed to the limits query. To fix this, just make sure the HEADER points to at least the highest value of the data base section in question.

To LIST Action entries 5 through 65, input the following (user inputs are underlined):

L (Press the "L" key from the main menu to enter the LIST sub-menu)

The computer will display the following:

```
Which section of the data base do you want to LIST:
Header, Actions, Vocab, Rooms, Messages or Objects
  Type:  H, A, V, R, M or O ? _
```

To select a data base section, press the first letter of that section. In our example this would be the "A" key:

A (To select the Action entries)

The computer will display:

```
  Lower Limit, Upper limit (ENTER is All) ? _
```

Now type in the limits (5 and 65):

5,65<ENTER>

The action entries will be listed starting at entry 5. After six have been listed, the display will pause. Press any key except <BREAK> or <CLEAR> to continue the listing. Pressing the <BREAK> key will return the program to the main menu.

Suppose we want to LIST the HEADER section. From the main menu, press the "L" key for LIST.

L (Select the "L" or LIST command from the main menu).

The LIST sub-menu will be displayed:

```
Which section of the data base do you want to LIST:
HEADER, ACTIONS, VOCAB, ROOMS, MESSAGES or OBJECTS
H, A, V, R, M, or O ? _
```

Press the "H" key to LIST the HEADER:

H (Select the "H" or HEADER data base section).

The HEADER will be LISTed as follows (the data presented here is just an example):

```
Adventure A   Version 1 . 22  5784 bytes free
Bytes under 16K= -3274
#OBJ #ACT #VOC #RM MAX  BEG  #TR  WLEN  TIME  #MSG TR-RM
60   224  120  50  4    23   7    4     30000 99    33
```

The first line of the HEADER list is the name of this adventure ("A"), the version number (1.22) and the bytes free. The bytes free are the number of text characters which may be added before the high memory pointer is overflowed. It is analogous to the free string space left in BASIC.

The "Bytes under 16K= -3274" line gives the number of bytes under the top of memory in a 16K machine. If the adventure is to fit in a 16K machine, this number must be greater than zero. In this case, the adventure uses 3274 bytes more than a 16K machine has.

The next line gives the number of objects (#OBJ), action entries (#ACT), vocabulary words (#VOC), rooms (#RM), carry limit (MAX), starting room number (BEG), number of treasures (#TR), word length (WLEN), time limit (TIME), messages (#MSG) and treasure room (TR-RM) of this adventure in that order. For example, the number of treasures is 7.

PRINT command:

The PRINT command will give a hardcopy listing of any one data base section or all of the data base. Pressing the "P" key while at the main menu will enter the PRINT sub-menu.

The PRINT sub-menu appears as follows:

```
Do you want to print everything or
just the Header, Actions, Vocab, Messages, Rooms or Objects
E, H, A, V, R, M or O ? _
```

The PRINT sub-menu gives the option of printing any or all of the data base. Pressing the <BREAK> key while a section is being printed will return control to the main menu and stop the printing.

The PRINT command, like the LIST command, gives options for upper and lower limits of the data base section to be printed. After pressing any of the PRINT sub-menu options, except Everything or HEADER, the limits to be printed are input. Again, pressing the <ENTER> key will cause all of that data base section to be printed. The section to be printed is selected by pressing the first character of its name.

When the "P" key is pressed from the main menu, the computer will display the following:

```
Do you want to print Everything in the data base
or just the Header, Actions, Vocab, Messages, Rooms or Objects
  Type:  E, H, A, V, R, M or O ? _
```

If any section of the data base has been entered past its limiting value (the value held in the HEADER), then all of the data will not be printed. To fix this, make sure the HEADER points to at least the highest value of the data base section being PRINTed.

If the printer is not ready when the command is started, ADVEDT will display an appropriate message and return to the main menu.

If either the "everything" or the "actions" data base section is selected, ADVEDT will ask for "Small (80) or Wide (132) column output". If you have a printer which is capable of printing on 132, columns, you may want to select the Wide column output. If the "Small (80) column output is selected, action entries will print on 2 lines for each action entry. If the "Wide (132) column output" is selected, action entries are printed one per line. If you have many action entries in an adventure, choosing the Wide (132) column ouput will save paper and speed up the printing of the adventure; you will probably find it easier to read and interpret, as well.

MODIFY command

The MODIFY command is used to edit/create an adventure. To enter the MODIFY sub-menu, press the "M" key while at the main menu.

The computer will display the following:

```
Which section do you want to MODIFY:
Header, Actions, Vocab, Rooms, Messages or Objects
  Type:  H, A, V, R, M or O ? _
```

To modify a section of the data base, simply key in the first letter of its name. If the option selected is anything other than the HEADER, an inquiry is made for the lower and upper limits of the data base section to be modified. Pressing <ENTER> here will let the user modify all elements of that section. One note however, the user can MODIFY past the limit value held in the HEADER for each data base section. If <ENTER> is pressed for the lower and upper bounds inquiry, any elements above the upper limit in the HEADER will be missed. The fix is to make sure the HEADER points to at least the last item in each data base section.

When modifying any section of the data base, pressing the <ENTER> key as the response will leave the item as is.

When an Action is modified, and all conditions and commands have been entered, an inquiry is made for "Y or N". Pressing "Y" means the modified action was correct. If "Y" is pressed, all changes to that action, if any, are stored and the next action entry is displayed until the upper limit specified earlier is reached. If the "N" key was pressed, the action entry may be modified again to correct any problems incurred while modifying it the first time.

When modifying actions, the conditions are entered with a space between the word and the number. For example, "AVL 50" and "ORIG 5" are legal inputs.

Verbs and nouns must also be entered separated by a space (VERB NOUN). Verbs and nouns input into actions must match exactly with ones found in the vocabulary data base section or an error message will be printed. For example, if "EXAMINE" was the entry in the verb vocabulary list, "EXAMINE" would have to be the verb entry in the action ("EXAM" would not work).

Suppose an action entry was to have "LIGHT TORCH" as its verb-noun. The conditions were that object 10 must be being carried (object 10 is an unlit torch) and the commands would be to switch the location of the lit torch (object 9) with the unlit one. The entry of the action would go as follows (underlined entries are input by the user):

```
M    (from the main menu to enter the MODIFY sub-menu)

Which section do you want to modify:
Header, Actions, Vocab, Rooms, Messages or Objects
  Type:  H, A, V, R, M, O or - ? A

(Select the Actions)

  Lower Limit, Upper limit (ENTER is All) ? 123
(Select Action 123)

Action 123:        Verb, Noun ? LIGHT TORCH <ENTER>
PAR 0  Cond, Value ? HAS 10
PAR 0  Cond, Value ? PAR 10
PAR 0  Cond, Value ? PAR 9
PAR 0  Cond, Value ? <ENTER>  (Pressing <ENTER> leaves it the same)
PAR 0  Cond, Value ? <ENTER>
 0  Cmd or Msg # ? EXX,X
 0  Cmd or Msg # ? <ENTER>
 0  Cmd or Msg # ? <ENTER>
 0  Cmd or Msg # ? <ENTER>
  Title ? <ENTER>
OK?  Type:  Y or N ? Y
```

See chapters 2 and 5 for more information on what the conditions and commands entered here actually do. The "PAR 0" and "0" preceding the inputs are the previous values of those entries. A "PAR 0" may be entered to delete an unnecessary condition. A "0" is entered to delete a command. When listed, a "PAR 0" will be displayed as just a "0". Since a lot of conditions in the action entries are not used, this shorthand display is used to ease make the display less cluttered. Likewise, a "0" command will be listed as a "-".

To enter a room called "CLOAK ROOM" with a NORTH exit to room number 10, the following would be entered:

```
M    (from the main menu to enter the MODIFY sub-menu)

Which section do you want to modify:
Header, Actions, Vocab, Rooms, Messages or Objects
  Type:  H, A, V, R, M or O ? R

(Select the ROOMs)

  Lower Limit, Upper limit (ENTER is All) ? 5

(Select room number 5)
```

```
Room  5:   0 N    0 S    0 E    0 W    0 U    0 D
  Room description:
N,S,E,W,U,D rooms ? 10,0,0,0,0,0
Description ? Cloak room
```

The numbers preceding the letters (N, S, E, etc.) are the previous adjacent room numbers.

When entering the adjacent room numbers, these values must be less than or equal to the number of rooms specified in the HEADER. For example, if the header says there are 10 rooms in this adventure, legal adjacent room numbers would be from 0 to 10. If you typed in a room number of 50, ADVEDT would give you an error message.

Also, when entering the adjacent room numbers, you do not have to enter all six room directions. Suppose you only wanted to change the third direction (East). You would only have to enter the first three directions. The forth through sixth room directions would be left unchanged. If the room modified above is taken as an example, you would not have to enter "10,0,0,0,0,0" as the room directions. You could simply enter just a "10". In this case, the second through sixth room directions (South through Down) would be left unchanged.

The MODIFY command, unlike the READ command, respects the high memory value. For example, if there are only 10 characters of string space left, and you try to add a string of a length greater than 10, ADVEDT will give you an out of string space message.

EDITING TEXT INPUT

At times some of the text messages in an adventure have typos or other mistakes. To correct these errors, the offending text can be simply overtyped. This means MODIFY the appropriate text string and reenter it.

For example, suppose the following message was in the data base:

Msg#	Message text
5	Look out for faling rocks!!!

The incorrect word is "faling" which should read "falling". To correct this word, we could MODIFY messge 5 and retype the message using the correct spelling of "falling". The commands would go as follows:

```
Do you want to READ, LIST, PRINT, XREF, WRITE, MODIFY, INSERT,
CLEAR or END
R, L, P, W, M, I, C or E ? M

Press "M" for MODIFY.

Which section do you want to MODIFY:
Header, Actions, Vocab, Rooms, Messages or Objects
H, A, V, R, M or O ? M

Press "M" to modify the messages.
```

```
Lower limit, upper limit (ENTER is all) ? 5

Enter a "5" since we want to MODIFY message 5.

Message 5  :  Look out for faling rock!!!
Message ?  Look out for falling rock!!!
```

Retype the message using the correct spelling of "falling".

In this example, making the correction was fairly simple since the line to be retyped was relatively short. However, if the text was long, retyping it could be very time-consuming.

ADVEDT's editing features can eliminate much of this extra typing. The editing commands are the same as those used by BASIC, so if you are familiar with those commands, you will have no trouble using them with ADVEDT.

Not every section of the data base is able to use the editor. The following data base sections support the use of the string editor:

1) Action entry titles
2) Room descriptions
3) Message descriptions
4) Object descriptions

ENTERING THE EDITOR

The string editor may be entered when ADVEDT prompts you to enter an Action title, Room descrption, Message or Object description. The prompts for each of these, which are displayed when using the MODIFY command, are as follows:

Data base section	Prompt
Action title	Title ?
Room description	Description ?
Message	Message ?
Object description	Description ?

If the first character you enter at one of these prompts is a SHIFT UP ARROW, then the editor is entered. If something other than SHIFT UP ARROW is pressed, then the standard overtype mode is invoked. The text string is replaced by whatever you enter when overtype mode is used.

EDITING COMMANDS

The following are the editing commands available (note that these commnads are the same as those of the BASIC editor):

<ENTER>

Pressing <ENTER> while in the edit mode cause ADVEDT to record all of the changes to the string you have made (if any).

nn<SPACE BAR>

While in edit mode, pressing the <SPACE BAR> advances the cursor one character to the right.

To advance the cursor more than one character at a time, type the number of characters to move over first, then press the <SPACE BAR>. For example, "15 <SPACE BAR> will move the cursor over 15 characters.

nn<BACKSPACE>

Moves the cursor to the left by "nn" spaces. If no number "nn" is entered, the cursor is backed up one space. For example, typing "3<BACKSPACE>" will move the cursor back three spaces.

(L)IST THE LINE

Pressing the "L" key causes the remainder of the string to be listed on the screen. The cursor is then brought to a new line and positioned at the first character of the string. This command is used to list the string on the display.

SHIFT/UP ARROW

The SHIFT/UP ARROW key is used to exit any of the insert-type commands listed below: I, X and H. After pressing these keys, you will still be in edit mode nad the cursor will remain at its last position.

(I)NSERT

Pressing "I" enters the insert mode. This allows you to insert text beginning at the current cursor position. Use the SHIFT/UP ARROW or <ENTER> key to exit the insert mode. Pressing SHIFT/UP ARROW exits insert mode, but leaves you in edit mode. Pressing the <ENTER> key exits both the insert mode and the edit mode.

X (EXTEND STRING)

Pressing the "X" key causes the rest of the line to be listed, moves the cursor to the end of the line and puts the editor in insert mode so you can add text to the end of the string.

(H)ACK LINE

Pressing the "H" key tells the editor to delete the remainder of the line (the current cursor position becomes the end of the line) and enter insert mode.

A (CANCEL CHANGES AND START AGAIN)

Pressing the "A" key moves the cursor back to th beginning of the string and cancels any editing changes made. For example, if you hvae added, deleted or changed something in a string and wish to cancel the changes, press the "A" key.

(O)UIT EDITING

Pressing the "O" key tells the editor to end editing and cancel all changes made in the current editing session. If you have decided to not change a line, press "O" to cancel changes and exit the edit mode.

(E)XIT EDITING

Pressing the "E" key tells the editor to end editing and save all changes made.

nn(D)ELETE

Tells the editor to delete the specified number "nn" charactes to the right of the cursor. The deleted characters will be enclosed in exclamation marks to show you which characters were affected. For example, typing "5D" will delete the five characters following the cursor.

nn(C)HANGE

Tells the editor to allow you change the specified number "nn" characters following the cursor. Pressing "C" with no number lets you change the next character. When "nn" characters have been entered, you are returned to edit mode.

nn(S)EARCHc

Tells the editor to search for the "nn"th occurrence of the character "c" and move the cursor to that position. If "nn" is not specified, the editor will search for the next occurrence of the specified character. If the character "c" is not found, the cursor is moved to the end of the line. For example, typing "2Sa" findes the second occurrence of the character "a" after the cursor.

nn(K)ILLc

Tells the editor to delete all characters up to the "nn"th occurrence of the character "c" and moves the cursor to that position.

An example of using the editor would be very helpful at this point. Let's use the example described above. If you like, you can actually input message 5 as shown above and follow along. The message appeared as follows:

Msg#	Message text
5	Look out for faling rocks!!!

Previously, we retyped the entire message to fix the misspelled word "faling". Let's use the string editor to fix the offending word. The command flow would go as follows:

Do you want to READ, LIST, PRINT, XREF, WRITE, MODIFY, INSERT,
CLEAR or END
R, L, P, X, W, M, I, C or E ? M

Press "M" for MODIFY.

```
Which section do you want to MODIFY:
Header, Actions, Vocab, Rooms, Messages or Objects
H, A, V, R, M or O ? M
```

Press "M" to modify the Messages.

```
Lower limit, upper limit (ENTER is all) ? 5
```

Enter a "5" since we want to MODIFY message 5.

```
Message 5          Look out for faling rocks!!!
Message ? <SHIFT/UP ARROW>
```

Press <SHIFT/UP ARROW> to enter the edit mode. The "Message ?" line will now be:

```
Message ? _
```

The editor is waiting for your command. We want to add an "l" to the word "faling". Before we can add this character, the cursor must be positioned to the point where the character is to be inserted. This can be done by continually pressing the <SPACE BAR> to move the cursor over.

Pressing the <SPACE BAR> once yields:

```
Message ? L_
```

The cursor moved over one space. Let's move it five more spaces over. Do this by typing "5 <SPACE BAR>". The line is now displayed as:

```
Message ? Look o_
```

NOTE: When "5 <SPACE BAR>" was typed in, nothing was displayed on the screen. This is normal for edit mode.

You can move the cursor back and forth with the <SPACE BAR> and <BACKSPACE> keys. Experiment with these keys moving the cursor forward and backward until you get the display:

```
Message ? Look out for fal_
```

We can now insert an "l" at this point to fix the typo. Enter the insert mode by pressing the "I" key. Note that the display did not change. Now type an "l". The display should be:

```
Message ? Look out for fall_
```

Press the <SHIFT/UP ARROW> keys to exit the insert mode. You can experiment with the commands available in the edit mode at this point. When you hve finished experimenting, press the <ENTER> key to save the change.

Let's look at some of the other editor commands. Suppose we want to change message 5 to "Look out for falling stones". This can easily be done. Position the cursor so it is at the position shown below:

```
Message ? Look out for falling _
```

Now press the "H" key for Hack off line. This command deletes everything after the cursor position and puts you in insert mode. Now type in the word "stones":

```
Message ? Look out for falling stones_
```

We are still in insert mode. To exit insert mode press the SHIFT/UP ARROW keys as done before. The cursor is at the end of the line now. To bring the cursor to the beginning of the line press the "L" key (List command). The display will now be:

```
Message ? Look out for falling stones
_
```

If we were to press the <ENTER> key now, the changes we have made would be saved and message 5 would be shown immediately above. Let's change the word "stones" to "boulders". Again, we must move the cursor over to the "s" in "stones" before any changes are made. We'll move the cursor over with the "S" (Search) command this time. Pressing "Ss" will give:

```
Message ? Look out for falling _
```

There are six characters in the word "stones". To delete that word we type "6D". This yields:

```
Message ? Look out for falling !stones!_
```

The exclamation marks delimit the characters deleted. To enter the word "boulders", first enter insert mode by pressing the "I" key and then type in the word:

```
Message ? Look out for falling !stones!boulders_
```

Let's save the changes this time by pressing the <ENTER> key. Message 5 will now read:

```
Look out for falling boulders
```

If you have used BASIC's line editor before, just keep in mind that all commands with ADVEDT's string editor are the same. the only difference is the way the editor is invoked. In BASIC, the editor is called up by typing:

```
EDIT .              or,
EDIT <line number>
```

With ADVEDT, press the <SHIFT/UP ARROW> keys when asked to input the string.

INSERT command:

The INSERT command will insert blank lines into certain data base sections. Press the "I" key from the main menu to enter the INSERT sub-menu.

Insertions can be made into actions, verbs and nouns. To select a section, type in its first letter. Next, indicate the number of blank lines to be inserted. Lastly, respond with the item number the blank lines are to be inserted after.

The number of blank lines to be inserted must be a positive number. If it is not, an error message is displayed and control returns to the INSERT sub-menu.

A couple of applications of INSERT are given below:

At times, the user would like to add a synonym to an existing verb or noun in the vocabulary list. If there is not a blank line for the synonym after the primary verb or noun, the vocabulary words have to be moved around to make room for the synonym. This can be a real hassle since the action entries will probably have to be modified also. For example, if verb number 20 is moved to verb number 58, then all occurrences of verb 20 in the actions will have to be changed to verb 58. The actions do not actually contain the text verb, noun combination, just their position in the vocabulary list. If the order of the vocabulary is changed, the affected action entries must also be changed. This would be done by modifying the affected actions and retyping the verb-noun combination. An easy way to find all occurrences of a verb or noun is use the XREF command (discussed below) to find which actions they are used in.

Suppose the word EXAM was in the verb list and the synonym *HIT was to be put in its place. Every action entry with the verb EXAM would now have *HIT in its place. The easy way to add the synonym would be to insert a blank line before the EXAM verb and type the synonym in. The INSERT command could be used to put one blank line before EXAM (actually the blank line would be inserted after the verb just before EXAM) so *HIT could be entered, thus saving a lot of modifying. What really happens is that the INSERT command goes through and does all of the action entry modifying described above for you. Inserting into nouns is done the same way.

There are some rules which must be followed however. No insertions may be made before verb 18 (DROP). Since an insertion before the DROP verb would move it from its predefined position, an insertion before verb 18 is not allowed.

No insertions are allowed before noun 6. Nouns 1-6 are the predefined room directions and they can not have any synonyms so no insertions may be done in them.

Another application of INSERT involves the action entries. At times, a few blank lines may be needed between two action entries. For example, maybe an action entry has to be lengthened to the point where a CONT command in the action entry must be added. If another action entry directly follows the entry to be CONTinued, at least one of them will have to be moved and retyped. The INSERT command will allow one or more blank lines to be inserted so the CONTinued action entry may be entered with no retyping.

Another use of the INSERT command on actions is for making more space for the automatic actions. The automatic action entries must precede all player input actions. If there is no more room for auto-actions, the INSERT command can make room by inserting some blank lines before the player input actions.

An example of INSERTing follows. Suppose you have the following partial noun list:

```
23:  SHELF
24:  *BOOK
25:  CASSETT
26:  DISKETT
27:  DOG
```

If you wanted to add the synonym "*TAPE" to the noun "CASSETT", the INSERT command could be used. The procedure goes as follows (all user input is underlined):

Press the "I" key from the main menu to enter the INSERT sub-menu. This message will be displayed:

```
What section of the data base do you want to insert into:
Actions, Verbs or Nouns
  Type:  A, V or N ? N
```

(The "N" was pressed to select the nouns)

The computer will respond with:

```
How many blank lines ? 1
```

(We need to insert only one blank line)

The computer will inquire:

```
After what noun # ? 25
```

After a few seconds, the insertion will be completed. After the task is finished the nouns will be listed as follows:

```
23:  SHELF
24:  *BOOK
25:  CASSETT
26:
27:  DISKETT
28:  DOG
```

Now the MODIFY command could be used to place the noun synonym "*TAPE" at noun 26.

As you can see, noun 26 (DISKETT) was moved to noun 27 (all nouns following the selected noun are moved down). The INSERT command changes all action entries affected by this operation (instead of referring to noun 26, refer to noun 27).

One warning about INSERT. When an insertion is made, the highest item in the data base section selected will be moved up in the list. The limiting value in the HEADER is not updated however. A check should be made to determine if this is the case.

XREF command:

The XREF command returns the number of every action entry a noun, verb, room, message, object, bit flag or counter appears in. To use this command, type "X" from the main menu.

The XREF sub-menu will be displayed on the screen. Press the first letter of the section the XREF is to be run on. Enter the item number for the XREF (for example, which object number). ADVEDT will not allow illegal values to be input.

Output may be routed to the screen or printer by typing an "S" for screen or "P" for printer when asked.

Finally, the limits of the actions the XREF is to be run on is entered. For example, you may want to see which actions from action entry 5 through action entry 134 reference object 34.

There are a few oddities about the XREF command. An XREF can not be done on message 0. It may also give strange results on nouns. XREF can find every occurrence of a noun number. However, no distinction is made from nouns and auto action probabilities. Just list the actions after the references are given so a distinction between the word nouns and the probabilities can be made.

When an XREF is run on the verbs, nouns or messages, a match is displayed by the message "ACTION: n" where "n" is the action number that particular item was found in.

When an XREF is run on an object, room, bit flag or counter, one of two messages will be displayed:

The first one is "CONDITION - ACTION: n". This message is displayed if the item found was referenced in the conditions of the action. For example, "BIT 30", "HAS 10" and "IN 3" are references in the conditions.

The second one is "COMMAND - ACTION: n". This message is displayed if the item found was referenced in the commands of the action. For example, "GOTOY" with the associated parameter in the conditions of "PAR 3" references room 3 or "AGETX" with a "PAR 10" in the conditions references object 10.

When displaying matches, XREF does not pause the output. Therefore, if there are more than 15 or so matches, the first ones may be hard to read. In this case, select a smaller range of actions to be scanned and do the XREF in more than one part. Another solution is to send the output to the printer.

The following table tells which conditions are checked when doing an XREF on the appropriate data base section:

OBJECTS	ROOMS	BIT FLAGS
HAS	IN	BIT
IN/W	-IN	-BIT
AVL		
-IN/W		
-HAVE		
-AVL		
-RMO		
RMO		
ORIG		
-ORIG		

The following table tells what commands are checked for when doing an XREF on the appropriate data base section:

OBJECTS	ROOMS	BIT FLAGS	COUNTERS
GETX	GOTOY	SETZ	EXM,CT
DROPX	X->Y	CLRZ	
X-RMO			
X->Y			
EXX,X			
AGETX			
BYX->X			

As you can see by the above table, the only command searched for when doing an XREF on a counter is the "EXM,CT" command. The reason for searching for only this command even though others may affect the counter (that is, CT-1) is that there is no way of checking if the other commands (such as CT-1) are affecting that particular alternate counter.

It should also be noted that for every "EXM,CT" command of a particular counter, there will almost certainly be another "EXM,CT" command to switch it back in either the same action or in one closely following the first one. This situation should be considered.

Suppose you wanted to find which actions referenced object number 25. The procedure would go as follows (user inputs are underlined):

From the main menu, press the "X" key to enter the XREF sub-menu:

```
What section of the data base do you want an XREF of:
Verbs, Nouns, Rooms, Messages, Objects, Bit flags or Counters
  Type:  V, N, R, M, O, B or C ? O

(Select an Object XREF)
```

The computer will inquire:

Object # ? 25

(Select object 25)

Screen or Printer output: Type S or P ? S

(Direct all output to the screen)

Actions scanned:
Lower Limit, Upper limit (ENTER is All) ? <ENTER>

(Scan all of the actions)

CONDITION - ACTION:6
COMMAND - ACTION:45
CONDITION - ACTION:123

The output in this example was contrived, so don't be surprised if your results are different. These messages indicate that object number 25 is referenced in the conditions of actions 6 and 123 and is referenced in the commands of action 45.

If the <ENTER> key is pressed for the range of actions scanned, the upper limit scanned will be the upper limit held in the HEADER. If the HEADER value is not high enough to encompass all actions, any actions above the HEADER value will not be checked.

CLEAR command:

Pressing the "C" key from the main menu causes the CLEAR command to be entered. This command is used to zero out a data base currently in memory.

After pressing "C" from the main menu, the program will respond with "Are you sure ? ". Typing a "Y" will zero out the data base. If you respond with a lower case "y", the data base will not be cleared (it must be an upper case "Y").

Suppose you were looking at a finished adventure and decided to start typing in a new one. This command would be used to clear out the finished adventure so you could type the new one into "cleared" memory. It's comparable to typing "NEW" in BASIC before entering a program.

END command:

Pressing the "E" key from the main menu causes the END command to be executed. The program will respond with "Are you sure ?". Typing in a "Y" (must be an upper case "Y") will cause the program to exit to DOS or BASIC (depending on which version of the program is being run - disk or tape).

If the RESET button is pressed or the editor is accidentally exited, entering the program again will leave your data base intact. However, if memory were destroyed, the data base would not be intact.

To reenter the tape version of the program, type in SYSTEM followed by "/20992".

For a disk system, simply type "ADVEDT" from DOS and your data base will be intact provided memory was not garbled.

ADVENTURE DRIVER ROUTINE

To ease the debugging of adventures, the adventure driver program, ADV, is built into ADVEDT. The driver portion of the editor is called up by pressing the <CLEAR> key.

Upon pressing the <CLEAR> key, the program will display "Play a 'SAVED' game ?". Respond with a "Y" or "N". This adventure driver allows only one saved game with the file name "SAVED" (if using the disk version of the program).

Pressing the <CLEAR> key a second time will return control to the editor with your data base in tact. This greatly increases the speed of debugging adventures.

Suppose you are finished typing in your adventure and want to play it. Pressing the <CLEAR> key will enter the adventure driver routine. Suppose after a few moves you find a problem with the adventure. Pressing the <CLEAR> key again will reenter the editor portion of ADVEDT. The fix could be made and the game played again by another press of the <CLEAR> key.

It is even possible to play the adventure up to a certain point and save the game to disk or tape. After saving the game out, the editor can be reentered by pressing the <CLEAR> key. A change could be made and the adventure driver started again by another press of the <CLEAR> key. This time say "Y" to the use a saved game question. The saved game will be read in and the adventure can be continued from the point at which it was saved.

One caution if using this method of debugging. When a game is saved, the location of all objects is written out. If the editor is reentered and the number of objects in the HEADER has changed, the saved game can not be read in again.

ADVTT UTILITY PROGRAM

Both the tape and disk versions of TAS have the utility ADVTT included. This program will read in an adventure data base saved to tape and write out a tape version of the adventure game. The tape version of the adventure game is a combination of the driver and data base. These tapes are loaded via the BASIC command SYSTEM. This program is similar to the utility program ADVTAPE, except this program reads in tape data bases, not disk data bases as ADVTAPE does.

To run the tape version of the ADVTT program, enter BASIC and type SYSTEM. The file name is "A". When you are prompted again, type "/<ENTER>". The program is self prompting.

To run the disk version of the ADVTT program, type "ADVTT" from DOS.

To play the adventure tapes created by ADVTT, go to LEVEL II BASIC if using a Model I computer or LEVEL III BASIC if using a Model III computer. Type "SYSTEM" and press the <ENTER> key. The file name of the tapes created by ADVTT is "A", so type in that file name and press <ENTER>. When prompted again, type "/<ENTER>" and the adventure will be started.

PROGRAM COMPATIBILITY

ADVEDT, ADVTT and ADV have been used with NEWDOS/80 version 2.0, TRSDOS 2.3, NEWDOS/21, DOSPLUS, LDOS, Model III TRSDOS and NEWDOS/80 version 1.0 with success. The only problem noticed was that some DOSes do not recognize the BREAK key as a legal character. Thus, in this case, the SHIFT RIGHT ARROW keyboard combination is used. Other that that, there should be no problems with any of the current popular DOSes.

The same programs work with both the Model I and Model III computers. That is, ADV, ADVTT and ADVEDT all work on both Model I and Model III computers with no modification.

ADVEDT LIMITATIONS

The following limitations are imposed on data entered via the ADVEDT program:

1) Maximum number of action entries is 500 (0-499).
2) Maximum number of vocabulary entries is 150 (0-149).
3) Maximum number of rooms is 100 (0-99).
4) Maximum number of messages is 99 (1-99).
5) Maximum number of objects is 250 (0-249).
6) Maximum characters in the description of an object, room or message is 255.
7) Maximum word length of vocabulary words is 7 characters.
8) Maximum length of action titles is 20 characters.

Another limitation of ADVEDT is that only integer numbers may be entered of the range +32767 to -32768. Typing numbers outside this range will produce strange results.

SUGGESTED ADVENTURE ENTRY WITH ADVEDT

The first step in using ADVEDT is writing the adventure on paper. It is sort of like writing a story. Just the basic idea of the adventure is needed. After the basic idea is down on paper, some of the finer details should be considered.

After the finer details are done, start writing down the rooms, vocabulary and objects on paper. After most of these are entered, start writing the actions and messages. The reason for writing the actions and messages last is so you know what objects and rooms you have to work with and can add more when needed.

After the adventure is roughly down on paper, start entering it into the ADVEDT program. The HEADER should be modified first. The values input here do not have to be exact, just approximate.

A big decision has to be made at this time. That is the word length of the adventure. The word length is significant because the length of the object names (identifier between slashes for objects to be picked up and dropped) must match this value or be shorter. After the HEADER is entered, the order of section entry does not matter (except that the actions must be entered after the vocabulary).

Now comes the time consuming part, debugging the adventure. The way to do this is check every possible thing you can think of to see if the actions are performing the way you intended them to. If they don't, take notes on which ones aren't working and continue playing your ADVENTURE. After finishing with this procedure, return to the adventure editor to fix any problems. This process takes longer than writing the adventure in most cases. But after much hard work you should have an adventure you can be proud of. Remember, The Alternate Source will market any adventures of quality. See Appendix B for more details.

You can also reference Chapter 6 - Getting Started, for more information on this subject. Chapter 6 goes through all of the steps of writing an adventure: coming up with an idea, writing the adventure, entering the adventure into ADVEDT and debugging the adventure.

SAMPLE ADVENTURE

This chapter explains, in depth, a short adventure entitled "MINI-VENTURE." It is included on your master cassette or diskette as adventure "Z". This adventure uses most of the conditions and commands available in TAS. Every significant part of the data base is explained. But first here's a listing of the data base:

```
Adventure Z   Version 1.01  14500 bytes free
Bytes under 16K= 7523
#OBJ #ACT #VOC #RM MAX  BEG  #TR  WLEN  TIME  #MSG TR-RM
14   41   22   8    5    1    1    4    999   16    7
```

ACTIONS

```
#  --  V     N     COND 1   COND 2  COND 3  COND 4  COND 5
        CMD1    CMD 2   CMD 3   CMD 4    ACTION TITLE ---
 0: AUTO   100  -BIT 1   PAR 1       0       0       0
       MSG1     SETZ       -       -     INTRO
 1: AUTO   100  -IN 2    PAR 1   PAR 4   PAR 1       0
       EXM,CT   CT<-N    EXM,CT      -    SET MUG CNTR
 2: AUTO   100  IN 2     PAR 1       0       0       0
       EXM,CT   CT-1     CONT        -    MUGGED?
 3: AUTO     0  CT= 0       0       0       0       0
       MSG2     DEAD     FINI        -
 4: AUTO     0  PAR 1       0       0       0       0
       EXM,CT     -       -       -
 5: AUTO   100  -IN 2    BIT 15      0       0       0
       DAY      DSPRM      -       -     IN LIGHT?
 6: AUTO   100  IN 2     -BIT 15     0       0       0
       NIGHT    DSPRM      -       -     IN DARK?
 7: LIGH  MATC  HAS 13   PAR 9       0       0       0
       AGETX    DSPRM    MSG3     CONT    LIGHT MATCH
 8: AUTO     0  PAR 9       0       0       0       0
       DELAY    DELAY    X->RM0   DSPRM
 9: AUTO     0      0       0       0       0       0
       MSG4       -       -       -
10: GET   KEY   IN 1     RM0 12  PAR 12      0       0
       MSG5     AGETX      -       -
11: GET   KEY   IN/W 12  PAR 12      0       0       0
       GETX     MSG5       -       -
12: DROP  KEY   HAS 12   PAR 12      0       0       0
       DROPX    MSG5       -       -
13: EXAM  KEY   AVL 12       0       0       0       0
       MSG6       -       -       -
14: GO    DOOR  IN 2     PAR 3       0       0       0
       GOTOY    MSG5       -       -
15: UNLO  DOOR  HAS 12   IN/W 3      0       0       0
       MSG7       -       -       -
16: EXAM  DOOR  IN/W 3       0       0       0       0
       MSG8     MSG9       -       -
17: EXAM  WHEE  IN 1     RM0 12      0       0       0
       MSG12      -       -       -
```

ACTIONS (con't)

```
18: GO    ELEV  IN/W 4   PAR 4     0       0       0
      GOTOY   MSG5       -       -
19: GO    ELEV  IN/W 6   PAR 4     0       0       0
      GOTOY   MSG5       -       -
20: EXAM  PANE  IN/W 5     0       0       0       0
      MSG13     -        -       -
21: PUSH  1     IN 4     PAR 2     0       0       0
      CLRZ    MSG5       -       -
22: PUSH  2     IN 4     PAR 2     0       0       0
      SETZ    MSG5       -       -
23: GO    ROOM  IN 4     -BIT 2  PAR 3     0       0
      GOTOY   MSG5       -       -
24: GO    ROOM  IN 4     BIT 2   PAR 5     0       0
      GOTOY   MSG5       -       -
25: UNLO  DOOR  IN/W 7   HAS 12    0       0       0
      MSG7      -        -       -
26: EXAM  DOOR  IN/W 7     0       0       0       0
      MSG8    MSG10      -       -
27: EXAM  DOOR  IN/W 8     0       0       0       0
      MSG8    MSG11      -       -
28: EXAM  DOOR  IN/W 10    0       0       0       0
      MSG8    MSG11      -       -
29: UNLO  DOOR  HAS 12   IN/W 8   PAR 8   PAR 10    0
      EXX,X   MSG5       -       -
30: LOCK  DOOR  HAS 12   IN/W 10  PAR 8   PAR 10    0
      EXX,X   MSG5       -       -
31: GO    DOOR  IN/W 10  PAR 7     0       0       0
      GOTOY   MSG5       -       -
32: SAVE  GAME    0        0       0       0       0
      SAVE        -        -       -
33: QUIT  ANY     0        0       0       0       0
      FINI        -        -       -
34: SCOR  ANY     0        0       0       0       0
      SCORE       -        -       -
35: INVE  ANY     0        0       0       0       0
      INV         -        -       -
36: EXAM  ANY     0        0       0       0       0
      MSG14       -        -       -
37: HELP  ANY     0        0       0       0       0
      MSG15       -        -       -
38: GO    CAR   IN 2     PAR 1     0       0       0
      GOTOY   MSG5       -       -
39: PUSH  BUTT    0        0       0       0       0
      MSG16       -        -       -
40: TURN  ANY     0        0       0       0       0
      MSG5        -        -       -
41: AUTO    0     0        0       0       0       0
          -       -        -       -
```

VOCABULARY

#	--- VERBS	NOUNS ---
0:	AUTO	ANY
1:	GO	NORTH
2:	*ENTE	SOUTH
3:	*WALK	EAST
4:	EXAM	WEST
5:	*LOOK	UP
6:	LIGH	DOWN
7:	HELP	DOOR
8:	UNLO	KEY
9:	LOCK	*KEYS
10:	GET	ROOM
11:	*TAKE	BUTT
12:	*PULL	CAR
13:	PUSH	1
14:	INVE	2
15:	QUIT	ELEV
16:	SAVE	PANE
17:	TURN	GAME
18:	DROP	WALL
19:	SCOR	WHEE
20:		MATC
21:		
22:		

ROOMS

# ---	N	S	E	W	U	D	ROOM DESCRIPTION ---
0:	0	0	0	0	0	0	Storeroom. Can't get here
1:	0	0	2	0	0	0	car with an open door
2:	0	0	0	0	0	0	*I'm on the curb
3:	0	2	0	0	0	0	hallway
4:	0	0	0	0	0	0	*I'm in an elevator next to a room
5:	0	0	6	0	0	0	hallway
6:	0	0	0	5	0	0	hallway
7:	0	0	6	0	0	0	*I'm in my apartment
8:	0	0	0	0	0	0	hospital

MESSAGES

```
#  --- MESSAGE TEXT ---
 0:
 1: Welcome to "MINI-VENTURE" by Bruce Hansen
 2: I was MUGGED!!
 3: The match flares up
 4: and goes out.
 5: OK
 6: The number "201" is stamped on one of them
 7: The key won't fit
 8: There's a plate with
 9: 101 on it
10: 200 on it
11: 201 on it
12: There's a set of keys in the ignition
13: There are two buttons marked "1" and "2"
14: I see nothing special
15: HOW?
16: Say again with which button
```

OBJECTS

```
# --- START   OBJECT DESCRIPTION ---
 0:    -1   *MY WALLET*/WALL/
 1:     1   Steering wheel
 2:     2   Apartment complex main door
 3:     3   Locked apartment door
 4:     3   Elevator
 5:     4   Elevator panel
 6:     5   Elevator
 7:     5   Locked apartment door
 8:     6   Locked apartment door
 9:     0   Lighted artificial light source
10:     0   Open apartment door
11:     7   Sign saying "LEAVE *TREASURES* HERE"
12:     0   Keys/KEY/
13:    -1   Matches/MATC/
14:     2   Car
```

WHAT DOES IT ALL MEAN?

The data base will be explained one section at a time. First the HEADER:

"Adventure Z" is the adventure name. The version # is 1.01 and the adventure leaves 14,500 bytes free. This value may vary depending on your computer's configuration and if high memory drivers are present. The bytes under 16K= 7523. This means if the adventure uses 7523 more bytes, a tape version of the adventure won't fit in a 16K tape machine. #OBJ=14 means there are 14 objects, #ACT=41 means there are 41 actions, #VOC=22 means there are 22 verbs and 22

nouns, #RM=8 means there are 8 rooms, MAX=5 means the adventurer can carry a maximum of 5 objects, BEG=1 means the player begins in room 1, #TR=1 means there is only 1 treasure, WLEN=4 means the number of significant letters in the nouns and verbs is 4, TIME=999 means the light limit is 999 moves, #MSG=16 means there are 16 messages and TR-RM=7 means the treasure room is room 7.

ACTIONS

The automatic actions must be placed before player input actions. The AUTO verb signifies an auto action. The noun is the probability of this action being considered. When the player inputs his verb, noun, not all player input actions are scanned, just until it finds a true one (if one exists). All auto actions are considered even if a previous one is true.

This description does not tell what the messages, objects and rooms are (that is, their word description) in most cases so refer to the above data base listing for that information.

```
0: AUTO 100  -BIT 1  PAR 1    0      0      0
     MSG 1    SETZ      -       -    INTRO
```

The 0: means this is ACTION 0. The "AUTO 100" means this auto action is considered 100 percent of the time. The "-BIT 1" means if bit flag 1 is cleared, this condition is true. When the adventure is started, all bit flags are cleared so on the first pass of the auto actions this condition will be true. This is useful for printing introductions. The "PAR 1" is a parameter to be passed to the commands if all conditions are met. Remember, it is the same as a "DATA 1" in BASIC. If all conditions are met, the commands are executed. In this case, message 1 would be printed and bit flag 1 would be set (SETZ). Bit flag 1 is set since the first parameter in the conditions was a 1. The "SETZ" command acted like a BASIC "READ Z" command. The word "INTRO" is an optional action title. Message 1 is the introduction message.

```
1: AUTO 100  -IN 2   PAR 1   PAR 4   PAR 1   0
    EXM,CT    CT<-N    EXM,CT    -    SET MUG CNTR
```

This is an automatic action used to set a counter. In this adventure, if the player is outside his car and not in the apartment building for 4 consecutive moves, he is mugged and killed (he loses). This action is executed 100 percent of the time (AUTO 100). The condition "-IN 2" will be true if the player is in any room other than room 2. "PAR 1", "PAR 4" and "PAR 1" are parameters used by the commands if all conditions are true. If all conditions are true, the CT (counter) value is exchanged (EXM,CT) with alternate counter 1 since the first parameter in the conditions was a 1. The command "CT<-N" will set the counter to 4 (4 is the next parameter). And finally CT is exchanged back with alternate counter 1 (EXM,CT). The reason for this switching is that ADVENTURE can only operate directly on the CT variable. ADVENTURE can operate on a maximum of 9 additional alternate counters however. "SET MUG CNTR" is the optional action title.

```
2: AUTO 100  IN 2    PAR 1   0      0      0
    EXM,CT    CT-1     CONT      -    MUGGED?
```

This action is considered 100 percent of the time. The condition "IN 2" passes if the player is in room 2. "PAR 1" is a parameter used in the commands. If all conditions are true, the commands are executed. In this case, CT is exchanged with alternate counter 1 (EXM,CT) since the first parameter found in the conditions was a 1. Then CT is decremented (CT-1). "CONT" means to continue considering all following AUTO 0's until an AUTO 1-100 or a player input action is found. In this case, the next two actions are AUTO 0's. They are as follows:

```
3: AUTO  0   CT= 0   0      0      0      0
    MSG 2     DEAD     FINI     -
4: AUTO  0   PAR 1   0      0      0      0
    EXM,CT     -        -      -
```

In action 3, CT is tested to see if it is equal to 0 (CT= 0). If it does, then message 2 (MSG 2) is printed, the player is killed (DEAD) and the game is finished (FINI). If the counter was not equal to zero, action 4 is considered. It would normally be considered regardless of the pass/no pass status of action 3. But in this case the DEAD command was issued and the player was moved to the last room and killed. The FINI command then halted the game. This also halts the auto actions. Since action 4 has no conditions it is always true and "PAR 1" is passed to the commands. In this case, CT is exchanged with counter 1 again thus putting the counters back in their original positions. There is a good reason for making these two actions separate. Two tasks need to be done, check if the counter is equal to zero and switch CT back with alternate counter 1. If these two were put in the same action and the counter (CT) did not equal zero, then the commands would not be performed. In this case, the "EXM,CT" command would not be done so the counters would not be returned to the right place.

```
5: AUTO 100  -IN 2   BIT 15   0      0      0
    DAY     DSPRM      -        -     IN LIGHT?
```

This auto action is considered 100 percent of the time. The conditions are "-IN 2" and "BIT 15". "-IN 2" is true if the player is not in room 2. "BIT 15" passes if it is dark (bit flag 15 is defined by ADVENTURE for light/dark status). If the conditions are true, then the commands "DAY" and "DSPRM" are executed. The "DAY" command makes it day and "DSPRM" displays the current room. The reason for this action is to make it day after the player is off the curb (where it is dark). The "BIT 15" condition is included so it is made DAY only when it was just NIGHT. The reason for this is that the DSPRM command makes the screen "flicker" when it is executed.

```
6: AUTO 100   IN 2    -BIT 15  0      0      0
    NIGHT     DSPRM      -        -     IN DARK?
```

This action is considered 100 percent of the time. If the player is in room 2 (IN 2) and bit flag 15 is cleared (-BIT 15) then the commands are performed. "NIGHT" makes it dark out (if the player is not holding the artificial light source) and "DSPRM" displays the room. This action makes it dark when the player is on the curb (room 2) and only does a NIGHT and DSPRM if it was previously DAY. The reason for not doing a DSPRM every time is that it causes the screen to be redrawn, which makes it look like the screen just glitched.

```
7: LIGH MATC HAS 13  PAR 9    0      0      0
     AGETX    DSPRM     MSG3   CONT LIGHT MATCH
```

This is the first player input action. If the player types in "LIGHT MATCH" then this action is considered. If even one of the conditions of this action are not met, ADVENTURE continues searching the actions for another "LIGHT MATCH". If it finds another, it considers that one also, and so on until it finds a true one. If no other "LIGHT MATCH" is found, then the message "I can't do that . . . yet!" is printed and the player is asked to respond again. The condition in this action is "HAS 13". So if the player "HAS 13" (has the matches - object 13) the commands are performed. "AGETX" will make object 9 (from PAR 9 in the conditions) be carried by the player regardless of the carry limit. "DSPRM" will make it day if it is currently light out or object 9 is available (the artificial light source). Since an "AGET 9" was just executed, the DSPRM will make it light if it was dark out or leave it light if it was light. Message 3 is then printed (MSG 3) and the CONTinue flag is set. All following AUTO 0 actions will be considered. These actions are as follows:

```
8: AUTO  0   PAR 9   0      0       0       0
     DELAY    DELAY    X->RM0   DSPRM
9: AUTO  0    0      0      0       0       0
     MSG 4     -        -       -
```

Action 8 has no conditions so its commands are executed every time action 7 is true. A "DELAY" command makes the program stall for about 1 second. After two such stalls, the "PAR 9" object is put back in RM0 (X->RM0 - PAR 9 is the first parameter from the conditions). After the artificial light source is removed from the room, another "DSPRM" is executed. This will make it dark again if it was dark before the match was lit or light if it was light before the match was lit. Action 9 is considered next. Since it has no conditions its commands are performed. In this case message 4 is printed. Since ADVENTURE found a matching player input action it does not consider any following player input actions and now goes back and checks the automatic actions (ones at the beginning of the actions).

```
10: GET  KEY  IN 1   RM0 12   PAR 12   0      0
     MSG 5    AGETX      -        -
```

If the player types in "GET KEY" this action is considered. It passes if the player is in room 1 (IN 1) and object 12 is in room 0 (RM0 12). If these conditions are true, then message 5 is printed (MSG 5) and the player is forced to pick up object 12 (AGETX - PAR 12 passed from conditions). The logic of this action is simple. The player must be in the car (IN 1) and not have already gotten the keys some time before (RM0 12) for the player to be able to get the keys.

```
11: GET  KEY  IN/W 12 PAR 12   0      0      0
     GETX     MSG 5      -        -
```

If action 10 failed for any reason then this action is considered since they both have the same verb-noun combination. In this action, the player is allowed to pick up object 12 if he is in with, but not carrying, object 12 (IN/W 12). If true, a "GETX" command is performed and message 5 is printed (MSG 5). A "GETX" checks to see if the carry limit is exceeded. If not, the player picks

up object 12 (GETX - PAR 12 from the conditions). This action would not be needed if action 10 was not included. Objects which are named (/name/ at the end of the object description) can normally be picked up and dropped without this type of action. But by having a GET action for object 12, the automatic GET feature for that particular object is disabled. In this case, GET has to be included in the actions.

```
12: DROP KEY  HAS 12  PAR 12  0     0     0
      DROPX   MSG 5     -     -
```

If the player is carrying object 12 (HAS 12) then object 12 is dropped (DROPX - PAR 12 from the conditions). Note that this action is really not required. It is the only DROP action for this object. If deleted, the automatic DROP feature would work. However, the automatic GET feature would still not work because of action 10.

```
13: EXAM KEY  AVL 12  0     0     0     0
      MSG6    -       -     -
```

If object 12 is either being carried or is in the same room as the player (AVL 12), then message 6 is printed (MSG6). This type of action is very common for such things as examining, reading, etc.

```
14: GO   DOOR IN 2    PAR 3   0     0     0
      GOTOY   MSG5    -       -
```

If the player is in room 2 (IN 2) then the player is sent to room 3 (GOTOY - PAR 3 from the conditions) and message 5 is printed (MSG5). This is a common action for GOes without a direction. For example, GO CAVE, GO TUNNEL, etc.

```
15: UNLO DOOR HAS 12  IN/W 3  0     0     0
      MSG7    -       -     -
```

If the player types "UNLOCK DOOR" this action is considered. There are three doors in this adventure and the player has a key which will open only one of them. If the player has object 12 (HAS 12), and is in the same room as object 3 (IN/W 3) then message 7 is printed.

```
16: EXAM DOOR IN/W 3  0     0     0     0
      MSG8    MSG9    -     -
```

This is a common type of EXAMINE action. A message is printed when the object is examined, provided the player is by the object. The IN/W condition is usually used for an object which can not be carried. The AVL condition is used for an object which can be carried. In this case, if the player is in with object 3 (IN/W 3), then messages 8 and 9 are printed (MSG8 and MSG9). Note that message 8 contains only the first half of the EXAM message. The second half is message 9 for this door. Message 10 and message 11 are used for the second halves of the two doors which can not be opened by the player (all three use the same first half of the EXAM message).

```
17: EXAM WHEE IN 1    RM0 12  0     0     0
      MSG12   -       -     -
```

This EXAMINE action has two conditions. If the player is in room 1 (IN 1) and object 12 is in room 0 (RM0 12) then message 12 is printed (MSG12). The logic for this action is as follows: if the player examines the wheel and the keys (object 12) are still there (RM0 12) then an appropriate message is printed. However, if the keys had been previously picked up then this EXAMINE would fail and ADVENTURE would search for another matching "EXAM WHEE" (or "EXAM ANY").

```
18: GO  ELEV  IN/W 4 PAR 4    0      0      0
     GOTOY    MSG 5     -      -
19: GO  ELEV  IN/W 6 PAR 4    0      0      0
     GOTOY    MSG 5     -      -
```

These two actions are described together since they are very similar. There are 2 objects called "ELEVATOR" in this adventure. One is on the top floor, the other on the bottom floor. These actions check if the player is in with an elevator (IN/W 4 or IN/W 6) and if so sends the player to room 4 (GOTOY - PAR 4 from the conditions).

```
20: EXAM PANE IN/W 5  0      0      0      0
     MSG13     -      -      -
```

If the player is in with object 5 (IN/W 5), then message 13 is printed (MSG13). If the player is not in with object 5, ADVENTURE searches for another EXAM PANE.

```
21: PUSH 1   IN 4    PAR 2    0      0      0
     CLRZ    MSG5      -      -
```

This action uses a bit flag. If the player is in the elevator (IN 4) then bit flag 2 is cleared (CLRZ - PAR 2 from the conditions). When this adventure was written, it was decided to use bit flag 2 cleared for floor "1" and bit flag 2 set for floor "2." Since the elevator would initially be at floor "1", this was logical because all bit flags are cleared at the start of ADVENTURE. The status of the bit flag is tested so ADVENTURE knows which floor of the apartment complex to send the player when he leaves the elevator. There are two different rooms adjacent to the elevator so this condition must be checked.

```
22: PUSH 2   IN 4    PAR 2    0      0      0
     SETZ    MSG5      -      -
```

This action is very similar to the above action except the bit flag is set, not cleared. This tells ADVENTURE to put the player on the 2nd floor when he leaves the elevator instead of the 1st floor.

```
23: GO  ROOM IN 4   -BIT 2  PAR 3   0      0
     GOTOY   MSG5      -      -
```

If the player is in room 4 (IN 4) and bit flag 2 is cleared (-BIT 2), then the player is sent to room 3 (GOTOY - PAR 3 from the conditions) and message 5 is printed (MSG 5). In words, if the player is in the elevator, and he last pushed "1", then he is sent to the 1st floor. If bit flag 2 is set, however, then this action will fail.

```
24: GO  ROOM IN 4    BIT 2   PAR 5  0      0
     GOTOY   MSG5      -      -
```

If the player is in room 4 (IN 4) and bit flag 2 is set (BIT 2), then the player is put in room 5 (GOTOY - PAR 5 from the conditions) and message 5 is printed (MSG 5). In words, if the player is in the elevator, and he last pushed "2", then he is sent to the 2nd floor exit or room 5.

```
25: UNLO DOOR IN/W 7 HAS 12   0      0      0
      MSG7      -         -      -
```

This action is the same as action 15 except this action is for a different door (object 7, not object 3).

```
26: EXAM DOOR IN/W 7  0       0      0      0
      MSG8     MSG10     -      -
27: EXAM DOOR IN/W 8  0       0      0      0
      MSG8     MSG11     -      -
28: EXAM DOOR IN/W 10 0       0      0      0
      MSG8     MSG11     -      -
```

These three actions are included together since they are very similar (the only difference is that they refer to different objects). Actions 27 and 28 really refer to the same thing. In action 27, the examine refers to a closed door in room 6. In action 28, the examine refers to the same door except it has now been opened so object 10 is in the room (open door) instead of object 8 (locked door). These actions are very similar to action 16.

```
29: UNLO DOOR HAS 12  IN/W 8  PAR 8  PAR 10  0
     EXX,X    MSG5      -      -
```

This is the only door the player can unlock. The door may be unlocked if the following conditions are true: the player is holding object 12 (HAS 12 - the keys) and he is in with object 8 (IN/W 8 - locked door). If these conditions are met, the room locations of object 8 and object 10 are exchanged (EXX,X - PAR 8 and PAR 10 from the conditions) and MSG5 is printed.

```
30: LOCK DOOR HAS 12 IN/W 10  PAR 8  PAR 10  0
     EXX,X    MSG5      -      -
```

This action is very similar to the previous one except this one locks the door. To lock the door, these conditions must be met: the player is carrying object 12 (HAS 12 - the keys) and he is in with object 10 (IN/W 10 - open door). If the conditions are met, then object 8 and object 10 are exchanged (EXX,X - PAR 8 and PAR 10 from the conditions) and MSG5 is printed.

```
31: GO  DOOR  IN/W 10 PAR 7   0      0      0
      GOTOY    MSG5      -      -
```

If the player wants to go through the door, this condition must be met: the player is in with object 10 (IN/W 10 - an open door). If that condition was met, the player is put in room 7 (GOTOY - PAR 7 from the conditions).

```
32: SAVE GAME  0      0      0      0      0
     SAVE      -         -      -
```

This action lets the player save the game. No conditions need be met so the SAVE command is always executed.

```
33: QUIT ANY   0      0      0      0      0
    FINI       -      -      -
```

This action lets the player stop the game. The "ANY" noun means that the action is considered if the player's input noun was any of the nouns in the vocabulary list. This action requires no conditions to be met and performs a "FINI" command.

```
34: SCOR ANY   0      0      0      0      0
    SCORE      -      -      -
```

This action will print the player's score. This message gives the number of treasures stored in the treasure room and the percent stored. No conditions are needed so the "SCOR" command is always performed.

```
35: INVE ANY   0      0      0      0      0
    INV        -      -      -
```

This command tells the player the name of every object he is currently carrying. The "INV" command is directly executed since no conditions are present.

```
36: EXAM ANY   0      0      0      0      0
    MSG14      -      -      -
```

This action will print message 14 (MSG14) if the object being examined was not referred to in a previous true EXAMine action. It is usually present in every adventure. The message is typically something like "I see nothing special."

```
37: HELP ANY   0      0      0      0      0
    MSG15      -      -      -
```

This is another common action. If the player types "HELP", message 15 (MSG15) is printed provided no previous HELP action was found to be true.

```
38: GO   CAR   IN 2   PAR 1   0      0      0
    GOTOY  MSG5          -      -
```

If the player is in room 2 (IN 2) then he is put in room 1 (GOTOY - PAR 1 from the conditions) and message 5 is printed (MSG5).

```
39: PUSH BUTT  0      0      0      0      0
    MSG16      -      -      -
```

If the player PUSHes BUTTon then message 16 (MSG16) is printed. Since the player is supposed to "PUSH 1" or "PUSH 2" this action is used to tell him so.

```
40: TURN ANY   0      0      0      0      0
    MSG5       -      -      -
```

If the player tries to TURN any legal object, message 5 (MSG5) is printed.

```
41: AUTO 0     0      0      0      0      0
     -         -      -      -
```

This action is not used.

VOCABULARY

The user must refer to the VOCABULARY words in the data base list for this explanation. Notice that the predefined verbs and nouns are in their proper places (AUTO, GO, GET, DROP, ANY, NORTH, SOUTH, EAST, WEST, UP and DOWN). The only thing special about the vocabulary words is the synonyms. Part of the vocabulary appears as follows:

```
0:   AUTO
1:   GO
2:   *ENTE
3:   EXAM
4:   *LOOK
```

Synonyms are required to directly follow their primary noun or verb in the list and must be preceded by an asterisk. In this case, *ENTE is a synonym of GO and *LOOK is a synonym of EXAM. There may be more than one synonym for a certain noun or verb (see verb 10 - GET). Any action entries using the nouns or verbs must refer to a primary noun or verb, not a synonym.

ROOMS

An example of two rooms follows:

```
#--  N  S  E  W  U  D  ROOM DESCRIPTION
6:   0  0  0  5  0  0  hallway
7:   0  0  6  0  0  0  *I'm in my apartment
```

The 6: and 7: are the room number (6 and 7). The next six numbers are the rooms which the player will move to if he goes in that corresponding direction. For example, if the player is in room 6 and he types "GO WEST", ADVENTURE will send him to room 5. If in room 7, typing "GO EAST" will put the player in room 6. The zeros mean the player can't go in that direction.

The ROOM DESCRIPTIONs "hallway" and "*I'm in my apartment" are examples of the default room message. For example, if the player is in room 6, the room is described as "I'm in a hallway." If in room 7, the room is described as "I'm in my apartment." Putting an asterisk before the room description disables the automatic prefix message "I'm in a."

MESSAGES

The messages are just text strings. There is nothing unusual about this data base section.

Notice that message 0 is not used. It should be null. Since message 0 can not be displayed via the action commands, typing any text for it wastes memory.

Objects

Each object has a starting room and a description. Objects which have a name, for example "Keys/KEY/" may be automatically picked up and dropped (the name is KEY in this case). If the starting room is -1 (like it is for *MY WALLET*) then the object is carried by the player when the adventure is started. Any other number is the room number in which that object can be found. A room number of zero means the object has not been found yet and is in the storeroom.

Notice that object 9 is titled "Lighted artificial light source." Object 9 is predefined by ADVENTURE as the artificial light source in its lighted condition. In this adventure, object 9 is not used (except for the "LIGHT MATC" action) so no real name is given to it.

These are the steps needed to win at this adventure:

1) GET the keys from the car (EXAMine WHEEl will tell the player if they are there).
2) Get out of the car by moving EAST.
3) To see, light a match.
4) Enter the apartment building by typing GO DOOR.
5) Type GO ELEV and PUSH 2 to go to the second floor.
6) GO ROOM to leave the elevator.
7) GO EAST to the second locked apartment door on the second floor.
8) UNLO DOOR to open the locked apartment door.
9) GO DOOR and drop the wallet (the wallet is initially carried by the player).
10) Type SCORE.

For a more challenging adventure try adventures "X" and "Y" on The Adventure System master diskette. These adventures are considerably longer than this one and will pose much more of a challenge.

Chapter 6

GETTING STARTED

You better be in a comfortable chair for this chapter. It will go through the whole process of writing an adventure. The steps are listed below:

1) Brainstorming an adventure idea.
2) Writing (Coding) the adventure.
3) Entering the adventure with ADVEDT.
4) Debugging the adventure with ADVEDT.
5) Sending in your adventure for marketing considerations.

BRAINSTORMING AN IDEA

The first step in writing an adventure is coming up with an idea. It is probably best to write an adventure on a subject you know. If you write on something you know, it will be easier to make it a full length adventure. For example, most people do not know much about quantum physics. Just think how long an adventure on that subject would be if written by the average person! However, a physicist could probably write a fairly good adventure on the subject.

Well, I will brainstorm some ideas and pick from the list. How about one of the following adventures:

1) A hijacker's adventure in a plane.
2) A spy adventure; steal the secret plans.
3) An old west adventure.
4) A jungle adventure.

I've watched enough cowboy movies and reruns of "The Wild, Wild West" to know a little more about the old west than the other three ideas, so I'll use that one.

The next step is to make a list of items or situations you might come across in your adventure. I thought the following could be found in the old west (the list is practically endless):

1) Indians
2) Winchester rifles
3) Horses
4) Six shooters
5) Ten gallon hats
6) Spurs
7) Buffalo
8) Outlaws
9) Marshall
10) Saloons
11) Lariat
12) Telegraph
13) Horse drawn carriages
14) Trading post
15) Jail
16) Gallows
17) Wanted poster
18) Rattlesnakes
19) Whiskey
20) Gold mine

This list has only twenty items in it. You could probably think of at least twenty more. You need not confine yourself to this list either. Add more to it when you come up with some ideas. The list of things you might come across is written to give a starting place for writing the adventure.

Indian Situations

The first item in our list is "Indians". The possibilities here are endless. The Indians could be hostile or friendly. They could be used as guides to the happy hunting grounds or obstacles to a box canyon. In this example, they will be friendly Indians.

Indians used "Wampum" for money. This money was usually a string of beads. In this adventure, I will make the "Wampum" a treasure. Now we must come up with some way of getting the "Wampum" from the Indians. Since these are friendly Indians, we won't steal the wampum from them. The Indians liked to trade for goods they needed. How about trading them "Whiskey" for the wampum? In this adventure, that will be the only time we use the Indians. After going through this chapter, you may want to write some of your own situations using the Indians.

Buffalo Situations

Let's take the "Buffalo" listed above next. What could you do with or get from a buffalo? The Indians used buffalo for food, utensils and clothing. In this adventure, we will make the buffalo's hide a treasure.

Now, what do we have to do to get the buffalo's hide? I would imagine the first step is to kill the buffalo. Then we would have to skin it. A Winchester rifle could be used to shoot the buffalo and a knife to skin it. That finishes the buffalo (sorry).

Trading Post Situations

How about working on some minor things now. So far, we have used a "Winchester rifle", "Whiskey" and a "Knife" as objects for the situations dreamed up. How does the player get these objects? We could start the adventure with the player holding them, but that's too easy. I'll do it this way: start the game with the player holding 100 dollars. There was a "Trading Post" listed above, so let's use it in this adventure. The "Trading Post" will have the rifle, knife and whiskey in it. The player will then buy these objects. As a side event, if the player tries to "GET" one of the objects without buying it, he will be arrested by the Marshall and thrown in jail, thus ending the adventure.

Outlaw Situations

Now we'll introduce the "Outlaws" in our adventure. Let's post a wanted poster on the jail. The poster will say something like:

```
WANTED:  Dead or alive
   The Butler Gang
REWARD:  Gold Nugget
```

One thing outlaws commonly did in the Old West was rob a bank. So we need to have a bank in our adventure. For simplicity sake, we will place the outlaws in the bank right from the start of the adventure.

Now how will we catch the outlaws? In this adventure, the outlaws will be cowards. If the player draws a pistol while the outlaws are in the bank, they will drop their guns. The marshall will come and take the outlaws to jail. You might ask "Where did the pistol come from ?". How about letting the player carry it at the start of the game?

To simplify the adventure, the pistol will be unloaded. This way we won't have to write a lot of actions regarding the shooting of the pistol. You may want to add some actions regarding the pistol after going through this chapter. In fact, it might be good practice to do that.

Reward Situations

When the outlaws are captured, the player will be given a reward. To get the reward, the player will have to go to the jail to claim it. The "Gold Nugget" reward will be our last treasure. Now we can start writing the actions for this adventure.

WRITING (CODING) THE ADVENTURE

This section will "code" the situations described above in the format required for The Adventure System. Also, the complete data base for this adventure will be listed.

The first thing you should do is decide on the word length of the adventure. In this one, I'll use a word length of three. The three means only the first three characters of the player's verb and noun are significant. For example, if the player types in "SHOOT BUFFALO", the adventure driver program will see the input as "SHO BUF".

Indian Situations Coding

The first situation we planned out above was the Indian "Wampum". To get the wampum, we need to trade for it with the whiskey. Let's start defining our objects first:

Object #	Start room	Object description
0	1	Indians
1	0	*WAMPUM*/WAM/
2	2	Whiskey/WHI/

Notice that the Indians start in room 1. Suppose the Indians started the adventure in a teepee. This teepee could be room 1. Room 1 could be described as follows:

Room #	N	S	E	W	U	D	Room Description
1	0	0	0	3	0	0	teepee

Notice that room 1 has a West exit to room 3. Room 3 will be a room on the outside of the teepee. Room 3 will have the "Teepee" displayed in it as an object. So, if you are in room 3, one of the visible items will be a "teepee". The object in room 3 and the room itself could be:

Object #	Start Room	Object Description
3	3	Teepee

Room #	N	S	E	W	U	D	Room Description
3	4	0	0	0	0	0	great plain

Note that room 4 is a North exit from room 3. Room 4 will be described later.

Notice that object 2, the "*WAMPUM*" is a named object and starts in room 0, the storeroom. Since that object is named, it can be picked up and dropped with no action entries to perform those tasks. The "*WAMPUM*" starts in the storeroom because we don't want the player to get it until he trades the whiskey for it.

Notice that the "Whiskey" starts in room 2. Room 2 will be the "Trading Post" described above. We will detail this room later.

Enough of the data base has been defined to start writing some action entries. The action entries will not be numbered at this time (like the objects and rooms were) since we do not know where in the list of actions they will be placed. After all of the action entries have been written, we can number them. Let's do our move into the teepee action first.

If the player is in room 3 ("great plain"), he will see object 3 (the "Teepee"). We want the player to be able to enter the teepee by typing "GO TEEPEE". The conditions will be that the player is in the same room as the "Teepee", or object 3. If the conditions are true, the commands will send the player into the teepee or room 1. The action entry to perform this is:

GO TEEPEE	IN/W 3	PAR 1	0	0	0
	GOTOY	-	-	-	

Now that we've given the player a way to get into the teepee, let's write an action to trade the whiskey for the wampum. The player input for this action will be "TRADE WHISKEY". The conditions will be that the player is carrying the whiskey and that he is in the teepee. If those conditions are met, we will take the whiskey away from the player and drop the wampum in the teepee. The action entry to perform this is:

TRADE WHISKEY	HAS 2	IN 1	PAR 1	PAR 2	0
	DROPX	X-RM0	-	-	

So, if the player types "TRADE WHISKEY" and he is holding object 2 (the whiskey - HAS 2) and is in the teepee (IN 1), the commands are executed. The first command will drop object 1 (wampum) in the room. The second command will send object 2 (whiskey) to the storeroom.

Buffalo Situations Coding

Now let's write the actions affecting the buffalo. The objects described in our buffalo scenario above are listed here:

Object #	Start Room	Object Description
4	3	Buffalo
5	0	Dead Buffalo
6	0	*BUFFALO HIDE*/BUF/
7	2	Winchester rifle/RIF/
8	2	Knife/KNI/

Notice that the "Buffalo", object 4, starts in room 3. Room 3 was described above as a "great plain". This is a logical place for buffalo to hang out. It just happens to be the same room that the teepee is in. There is no problem having more than one object per room, so we'll put the buffalo in room 3.

To kill the buffalo, the player's verb, noun combination should be "SHOOT BUFFALO". The conditions of this action are that the player is in the same room as the live buffalo (object 4) and that the player is holding onto the rifle (object 7). The commands of this action will be to remove the live buffalo from the room and place the dead one (object 5) there. Let's use a message in this action. How about displaying "BANG!!!" and "Got em!!!" when the player shoots the buffalo. The messages and action are:

Message #	Message
1	BANG!!!
2	Got em!!!

SHOOT BUFFALO	HAS 7	IN/W 4	PAR 4	PAR 5	0
	EXX,X	MSG1	MSG2	-	

So, if the player types "SHOOT BUFFALO" and he has object 7 (the rifle) and is in with object 4 (the live buffalo), the commands are performed. The first command will exchange the locations of object 4 (live buffalo) and object 5 (dead buffalo). We want to send the live buffalo to the storeroom and drop the dead buffalo in the room. By switching the location of the live buffalo and the dead buffalo we accomplishes this. Messages 1 and 2 are also displayed.

Continuing with our buffalo scenario, the next step is to skin it. The player's verb, noun should be "SKIN BUFFALO". The conditions for this action are that the player is holding onto the knife (object 8 - HAS 8), is in the same room as the dead buffalo (object 5 - IN/W 5) and the buffalo hide is in the storeroom (object 6 - RMO 6).

Why should the buffalo hide be in the storeroom? At the beginning of the adventure, the buffalo hide starts in the storeroom. Once the player skins the buffalo, the buffalo hide will be removed from the storeroom and dropped into the player's room. If the player tries to skin the buffalo a second time, the buffalo skin will no longer be in the storeroom and the conditions of this action will fail. Suppose the player skins the buffalo a first time and carries

the skin off to the jail. For some reason the player returns to the "great plain" and tries to skin the buffalo a second time. If the check for the buffalo hide being in the storeroom was not made, the buffalo hide could be removed from the jail and dropped in the "great plain" again. This is a common check which must be performed in many similar cases. When debugging an adventure, you should catch most of these types of conditions. When you get more experienced at writing adventures, these types of conditions are considered at the initial writing of the action.

The commands of this action will drop the buffalo hide into the player's room. This action is:

```
SKIN BUFFALO   HAS 8   IN/W 5   RMO 6   PAR 6     0
               DROPX     -         -       -
```

Trading Post Situations Coding

Our next set of actions will involve the "Trading Post". As shown before, the trading post is room 2. The trading post will have a "Counter" and a "Mean-looking storekeeper" in it. These objects are:

Object #	Start Room	Object Description
10	2	Counter
11	2	Mean-looking storekeeper

Note that object 9 was not used since it is reserved for the artificial light source. We don't use it in this adventure, but you may want to add it in later. One possibility would be to add a gold mine to the adventure that is only lit when a torch is present. This torch would be object 9.

The room description of the "Trading Post" is:

Room #	N	S	E	W	U	D	Room Description
2	0	0	5	0	0	0	Trading Post

Note the East exit to room 5. This room will be described later on.

Earlier in this chapter we decided to make the player buy the knife, whiskey and rifle. The amount of money the player has will be kept in the CT counter. When he buys one of these goods, the price of the item will be subtracted from his money.

What will the player keep his money in? How about a wallet. This wallet will be carried by the player at the start of the adventure. This object is:

Object #	Start Room	Object Description
12	-1	Wallet/WAL/

The initialization action(s) of this adventure will place the player's starting money in his wallet, or set the CT counter to 100 (for 100 dollars). For this adventure, let's make the rifle cost $50, the knife $10 and the whiskey $3. The initialization action to set the player's initial money will be written later on.

The action to tell the player how much money he has will be written now. The verb, noun for this action will be "EXAM WALLET". The condition will be that the player is carrying his wallet (object 12 - HAS 12). If this is true, we will display a message like "It has ", followed by a "DSPCT" command, and a "dollars in it" message. The messages and action entry are:

Message #	Message
3	It has
4	dollars in it

EXAM WALLET	HAS 12	0	0	0	0
	MSG3	DSPCT	MSG4	-	

Now let's write the actions which let the player buy the rifle, knife and whiskey. Earlier we said if the player tried to get any of those items without buying them first, the marshall will arrest him and the game will be over. So how do we flag when the player has bought an item? How about a bit flag. We could use one bit flag to mark (or flag) when the player has bought an item. In this case, we will let the following bit flags flag the appropriate item:

BIT FLAG	Item flagged as bought
0	Winchester rifle - object 7
1	Knife - object 8
2	Whiskey - object 2

Since the bit flags are all cleared at the start of the adventure, we will use the cleared state as the "haven't bought it yet" status. If the bit flag is set, then the appropriate object has been purchased. For example, when the player buys the rifle, bit flag 0 will be set. If the player tries to GET the rifle, and bit flag 0 is not set (meaning he has not purchased the rifle), the player will be arrested. But if the player tries to GET the rifle, and bit flag 0 is set (meaning he has bought the rifle), the player will not be arrested and will be able to pick up the gun.

The "BUY" actions will all be very similar. We will make the verb, noun combination to buy the rifle "BUY RIFLE". The conditions will be that bit flag 0 is cleared (the player hasn't bought the rifle yet), the player is holding onto his wallet (object 12 - HAS 12) and the player is in the same room as the rifle. This second condition could be written at least two ways. One would be "IN/W 7" or in with the rifle. Another would be "IN 2" which would work since the rifle starts the adventure in room 2. In this example, we'll use the "IN/W 7" condition. It does not matter which one you use though.

The commands of this action will be to subtract 50 (the price of the gun) from the CT counter and set bit flag 0 (showing that the player has bought the rifle). To let the player know the gun cost him $50, we'll display the message "That'll be $50 partner. It's yours."

Similar messages and action entries will be used to buy the knife and whiskey. The only differences between the purchases is the bit flag set, the number subtracted from the CT counter, the object of the "IN/W" condition and the message displayed. The messages and action entries to buy the rifle, knife and whiskey are:

Message #	Message
5	That'll be $50 partner. It's yours.
6	That'll be $10 partner. It's yours.
7	That'll be $3 partner. It's yours.

BUY RIFLE	IN/W 7	HAS 12	-BIT 0	PAR 0	PAR 50
	MSG5	SETZ	CT-N	-	
BUY KNIFE	IN/W 8	HAS 12	-BIT 1	PAR 1	PAR 10
	MSG6	SETZ	CT-N	-	
BUY WHISKEY	IN/W 2	HAS 12	-BIT 2	PAR 2	PAR 3
	MSG7	SETZ	CT-N	-	

Now we'll write the actions which allow the player to pick up the rifle, knife and whiskey. First, the "GET" actions which do not arrest the player will be written. These "GET" actions will check if the player is in the same room as the object and if the appropriate bit flag was set. If this was the case, the player will be allowed to pick up the object.

Let's take the knife for example. The player's verb, noun will be "GET KNIFE". The conditions will be that the player is in the room with the knife (object 8 - IN/W 8) and that bit flag 1 is set (the player has already bought the knife). If this is true, the player will be allowed to pick up the knife. When the adventure driver picks up or drops an object with the automatic pick up/drop feature, the message "OK" is displayed after the object is picked up or dropped. For the "OK" message to be displayed in this case, the action entry must display the message. The actions to pick up the rifle and whiskey are very similar to the knife action, except a different bit flag and object are checked. The "OK" message and "GET" actions read:

Message #	Message
8	OK

GET RIFLE	IN/W 7	BIT 0	PAR 7	0	0
	GETX	-	-	-	
GET KNIFE	IN/W 8	BIT 1	PAR 8	0	0
	GETX	-	-	-	
GET WHISKEY	IN/W 2	BIT 2	PAR 2	0	0
	GETX	-	-	-	

Now we will write the "GET" actions for when the player has not previously bought the item being picked up. We stated earlier that if the player tries to "GET" the rifle, knife or whiskey without buying them first, he will be arrested and thrown in jail. The actions to be written here will perform this task.

In the above actions, if the player bought the rifle, for example, bit flag 0 will be set. If the player has not yet purchased the rifle, bit flag 0 will be cleared. We will make use of that when writing these actions. The verb, noun combination of this get action will be the same, "GET RIFLE". The conditions for this action are that bit flag 0 is cleared and the player is in the room with the rifle (object 7).

The commands for this action and the whiskey and knife actions will be the same, tell the player he is arrested for stealing, send him to jail and end the adventure. Instead of putting all of these commands in all three actions, why not set a bit flag to signify that the player is to be arrested?

In a previous chapter, it was stated that after the player input actions are scanned, the automatic actions are then scanned. So, if we set a flag in the player input actions, an automatic action could be written that will check if the same flag is set. If the flag is set, the player will be told he has been arrested and will be sent to jail. For your information, setting a bit flag to signify the end of a game can be very useful if there are many ways to end the game.

For example, in this adventure the player will be arrested if he gets the rifle, knife or whiskey without buying them first. We could add a horse in this adventure. If the player got on the horse and started riding it without first buying the horse, he could be arrested and sent to jail for that also.

In this adventure, bit flag three will be set if the player "GET"s one of the objects without buying it first. The actions controlling this are:

GET RIFLE	IN/W 7	-BIT 0	PAR 3	0	0
	SETZ	-	-	-	
GET KNIFE	IN/W 8	-BIT 1	PAR 3	0	0
	SETZ	-	-	-	
GET WHISKEY	IN/W 2	-BIT 2	PAR 3	0	0
	SETZ	-	-	-	

Now we need to write an automatic action which will arrest the player. This automatic action should be checked every time the auto actions are scanned so the probability should be 100. The condition will check if bit flag 3 is set. If set, the player will be given the message "Storekeeper cries 'STOP, THIEF'!!!". We will pause the display for a second so the player can realize what has happened. Then the message "Marshall arrests me for stealing!!!" will be displayed. Then we will send the player to a room called "jail cell" and end the game. The "jail cell" will be assigned to room 6. The room and messages are:

Room #	N	S	E	W	U	D	Room Description
6	0	0	0	0	0	0	jail cell

Message #	Message
9	Storekeeper cries "STOP, THIEF!!!"
10	Marshall arrests me for stealing!!!

Notice that the jail cell has no exits to other rooms. Since the game will be ended, there is no need for exits from that room. It might make the jail more "homey" if there were a couple of objects there. We will place the two objects listed below in the jail cell:

Object #	Start Room	Object Description
13	6	Bars
14	6	Gallows being built in the street

With objects 13 and 14 in the jail cell with the player, he will know for sure he is in trouble! The automatic arrest action is:

AUTO 100	BIT 3	PAR 6	0	0	0
	MSG9	DELAY	GOTOY	CONT	
AUTO 0	0	0	0	0	0
	MSG10	DSPRM	FINI	-	

If the player has been arrested, bit flag 3 will be set. If set, this auto action will be executed. The first action will display the storekeeper message and pause for about one second. The player is then sent to room 6, the jail cell. The CONTinue flag is set next.

The second action (AUTO 0) is a continued action from the first one (AUTO 100). There are no conditions in this action so its commands are executed whenever the first action's conditions are true. The first command will display the message which tells the player the marshall arrested him. The second command displays the player's current room (DSPRM). The third command, FINIsh, ends the game.

Since these are automatic actions, they will have to be placed in front of the player input actions. We will make sure of this when the ordering of action entries is finally done.

Outlaw Situations Coding

Let's take care of those nasty outlaws now. First, we'll need to post the Wanted Poster mentioned earlier. This poster is placed on the outside wall of the jail. This means a "Jail" object must be added. This "Jail" object must be placed in some starting room. In the "Trading Post Situations Coding" section, the trading post was given an East exit to room 5. Let's define room 5 now as a "City Street". The "Jail" object is placed in room 5, or the "City Street". The "Jail" object and room 5 are:

Object #	Start Room	Object Description
15	5	Jail

Room #	N	S	E	W	U	D	Room Description
5	0	0	0	4	0	0	*I'm on a city street

Note the West exit to room number 4. Room 4 will be used to connect the player's movements together. All rooms the player will enter in an adventure must be linked together in some manner. Room 4 will be used for this:

Room #	N	S	E	W	U	D	Room Description
4	0	3	5	0	0	0	*I'm on a trail

Note the South exit to room 3 and East exit to room 5. Previously, room 3 was given a North exit to room 4. It makes sense to have a South exit from room 4 to room 3. The same thing goes for room 5. There is a West exit from room 5 to room 4. So room 4 is given an East exit to room 5.

The outlaws are to be placed in a bank. The inside and outside of the bank must be described. The outside of the bank will be an object called "Bank" placed in room 5, the city street. The inside of the bank will be a room we need to add. Inside the bank, we could find the outlaws and some people with their hands up. The "Bank" room and objects are:

Room #	N	S	E	W	U	D	Room Description
7	0	5	0	0	0	0	Bank

Object #	Start Room	Object Description
16	5	Bank
17	7	Outlaws
18	7	Townsfolk with their hands up

Note the South exit to room 5 from room 7. This is the exit the player takes to leave the "Bank".

The wanted poster mentioned earlier is to be placed on the jail. If the player examines the jail, a message will be displayed telling him there is a wanted poster there. If the player then reads the poster, he will get the poster message. Note the <DOWN ARROW>s in the message description below. You can have one message print on several lines of the screen by pressing the down arrow key where you want new lines.

The verb, noun combination of the first action is "EXAM JAIL". The condition will be that the player is in the same room as the "Jail" object. If this is true, a message like "There's a wanted poster on it" will be displayed.

The verb, noun combination of the second action is "READ POSTER". The condition of this action will also be that the player is in the same room as the "Jail" object. The command will display the "Wanted: Dead or Alive" wanted poster message. These two messages and actions are:

Message #	Message
11	There's a Wanted Poster on it
12	WANTED: DEAD OR ALIVE <DOWN ARROW> The Butler Gang <DOWN ARROW> REWARD: A Gold Nugget!

EXAM JAIL	IN/W 15	0	0	0	0
	MSG11	-	-	-	
READ POSTER	IN/W 15	0	0	0	0
	MSG12	-	-	-	

The actions to catch the dreaded "Butler Gang" are next. I'll add a twist at this point. Suppose the player enters the bank and comes upon the "Outlaws". On the next move, if the player does not draw his gun, he is shot by the "Butler Gang". Let's write that action first.

An automatic action will be used to set bit flag 4 if the player is in the bank room. If the player draws his pistol immediately after he enters the bank, the outlaws will be arrested. When arrested, the outlaws will be sent to the storeroom and an object called "Outlaws behind bars" will be placed in the jail room.

An automatic action will be used to check if the player has been shot by the outlaws. The conditions for this one are to check if bit flag 4 is set and if the outlaws (object 17) are not in room 0. If the outlaws are not in room 0, then the player has gotten the "draw" on them.

A "Jail" room must be defined also. Room 8 will be used for this room. This is where the outlaws and reward will be sent when the outlaws are arrested. The player will be able to enter the "Jail" (room 8). Therefore, there must be an exit from the jail. The outside of the jail is object 15, which starts in room 5. This means an exit from the jail (room 8) should be back to room 5 (city street). The "Jail" room is:

Room #	N	S	E	W	U	D	Room Description
8	5	0	0	0	0	0	jail

Notice the North exit back to room 5.

The verb, noun combination for the action the player uses to get the "draw" on the outlaws is "DRAW PISTOL". The conditions will check if the player has the pistol and is in the bank with the outlaws. The commands will remove the outlaws from the bank, put the outlaws behind bars in the jail room, put the reward (gold nugget) in the jail, swap some happy townspeople for the ones with their hands up and display a message about the marshall catching the outlaws. The objects, message and associated actions for this are:

Object #	Start Room	Object Description
19	-1	Pistol/PIS/
20	0	Outlaws behind bars
21	0	*GOLD NUGGET*/GOL/
22	0	Happy Townsfolk

Message #	Message
13	The outlaws dropped their guns and the marshall arrested them!

DRAW PISTOL	PAR 21	IN/W 17	PAR 17	PAR 20	PAR 8
	X-RM0	X->Y	CONT	MSG13	
AUTO 0	PAR 21	PAR 8	PAR 22	PAR 18	0
	X->Y	EXX,X	-	-	

We are going to need an automatic action which sets bit flag 4 when the player enters the bank. We will also need an automatic action to check if the player has been shot by the outlaws. Earlier, we decided if the player did not draw his pistol at first contact with the outlaws, the outlaws would shoot him.

The first action is fairly simple. The only condition is if the player is in room 7, the bank.

The second action checks if bit flag 4 is set (the player is in the bank) and if the outlaws are not in the storeroom (they are in the bank). If this is true, then a message indicating he has been shot is displayed and the game is ended. The message and action are:

Message #	Message
14	The outlaws got the drop on me! I'm dead!!

AUTO 100	BIT 4	-RM0 17	0	0	0
	MSG14	DEAD	FINI	-	
AUTO 100	IN 7	PAR 4	0	0	0
	SETZ	-	-	-	

Notice that the first action uses the "DEAD" command. This command will send the player to the last room as defined in the HEADER. Therefore, we must describe the "DEAD" room. This room will be the last entry in the room list. To be safe, we should leave a few empty positions between the last room described above and the "DEAD" room. The "DEAD" room is described as:

Room #	N	S	E	W	U	D	Room Description
12	0	0	0	0	0	0	*I'm at Boot hill (six feet under)

The order of the above automatic actions is very important. The first one must precede the second one. When the adventure is being played, the automatic actions are scanned from the lowest numbered one to the highest numbered one. After the auto actions are scanned, the player inputs his verb, noun combination. This is followed by a scan of the player input actions. The process repeats itself by the automatic actions being scanned again.

Look closely at the first action. It tests to see if bit flag 4 is set. Now look at the second action. It sets bit flag 4. If the actions are left in the order they are now, the following will happen when the player is playing the adventure. First, the player will enter the bank. Then, the automatic actions will be scanned. The first one will be false since bit flag 4 has not been set yet. However, the second action will be true (the player is in room 7) and this action will set bit flag 4. After the rest (if any) of the automatic actions are scanned, the player is asked for his verb, noun. If the player types in "DRAW PISTOL", the outlaws will give up. If the player does anything else, the outlaws will not give up and will remain in the bank (they are not placed in the storeroom). The automatic actions will be scanned again, and this time the first action will be true and the player will be shot.

If the order of these two actions were reversed, the player would never be given a chance to draw his pistol. One of the auto actions would set bit flag 4. The other action would be checked next. If bit flag 4 was set, the player is shot and the game ends. There would be no chance for the player to input a verb, noun between these two automatic actions.

For the player to claim his reward, he must return to the jail (this is where the gold nugget was placed). The gold nugget is a named object so it may be picked up and dropped with no actions.

MOVING TO AND FROM ROOMS

We have described the inside and outside of the jail and bank. Room 8 and object 15 describe the jail, room 7 and object 16 describe the bank. For the trading post, however, we have only described the inside of it with room 2. We need to place an object describing the trading post in the adventure. The jail and bank outside descriptions were placed in room 5, so we might as well be consistent and put the trading post there also. The outside of the trading post is:

Object #	Start Room	Object Description
23	5	Trading Post

So far we have given an exit from the trading post, jail and bank. But we have not given a way to enter the trading post, jail or bank. If you look at the room descriptions described so far, none of them has an exit to room 2 (the trading post) or room 7 (the bank) or room 8 (the jail). We must somehow let the player enter these rooms.

The outside of each of these rooms is described by an object in room 5. Put yourself in the player's shoes. If you saw the object "Jail" in a room, you might try to enter the room with a "GO JAIL" verb, noun combination. This is exactly what we'll do to allow the player to enter the trading post, jail and bank.

The verb, noun combination for these actions is "GO room", where "room" is "TRADE" for the trading post, "JAIL" for the jail and "BANK" for the bank. The condition for each of these actions is the same - check if the player is in room 5 (since that is where the objects describing the outsides of these rooms are). If the condition is correct, the player will be sent to the appropriate room via the "GOTOY" command. These actions are:

GO JAIL	IN 5	PAR 8	0	0	0
	GOTOY	-	-	-	
GO BANK	IN 5	PAR 7	0	0	0
	GOTOY	-	-	-	
GO TRADE	IN 5	PAR 2	0	0	0
	GOTOY	-	-	-	

USER FRIENDLY ACTIONS

There are a few actions which should be in every adventure. These will let the player save the game, get an inventory, quit the game, etc. There is usually a "HELP" action in every adventure. The "HELP" action could only respond with "Try EXAMining thing", or anything else you can dream up.

There is usually an "EXAM ANY" action also. This action is used if there is not a "specific" EXAMine action for a particular object.

These "User Friendly" actions and messages are:

Message #	Message
15	Try EXAMining things
16	I see nothing special

HELP ANY	0	0	0	0	0
	MSG15	-	-	-	
EXAM ANY	0	0	0	0	0
	MSG16	-	-	-	
SAVE GAME	0	0	0	0	0
	SAVE	-	-	-	
QUIT ANY	0	0	0	0	0
	FINI	-	-	-	
INVEN ANY	0	0	0	0	0
	INV	-	-	-	
GET INVEN	0	0	0	0	0
	INV	-	-	-	
SCORE ANY	0	0	0	0	0
	SCORE	-	-	-	

These "User Friendly" actions are usually placed at the end of the player input actions. This is especially true for the "HELP ANY" and "EXAM ANY" actions. If the "EXAM ANY" action preceded the "EXAM JAIL" action described earlier, the "EXAM JAIL" action would never be executed since the "EXAM ANY" action will always be true.

INITIALIZATION ACTION ENTRY

The initialization action entry will be executed at the very start of the adventure, and never again. In a previous chapter, it was pointed out that testing a bit flag for its cleared state and then setting the same bit flag in the action would perform this function. In this initialization action entry, we will use bit flag 31. An opening message will be displayed and the CT counter will be set to 100. CT is set to 100 since the player starts the adventure with $100. The opening message and auto action are:

Message #	Message
17	Welcome to the "Old West" adventure by Bruce G. Hansen

AUTO 100	-BIT 31	PAR 31	PAR 100	0	0
	SETZ	CT<-N	-	-	

This ends the coding of the action entries. The order of the actions is not critical except where noted. Just remember to place the automatic actions before the player input actions. This adventure is listed later in this chapter.

FINISHING UP THE CODING

So far we have described the actions, the rooms, the messages and the objects. That leaves one data base section, the vocabulary. The actions and objects written above really hold all of the vocabulary words we need. But we might want to add some synonyms to certain verbs and nouns.

For example, the verb "GET" could have the synonyms "TAKE" and "GRAB". The trading post could be described as "TRADE" or "POST".

When writing the vocabulary lists, start with the verbs and nouns in the actions. Add synonyms to them as you see fit. Do not forget to make sure the list contains the object names (name between slashes at the end of the object descriptions). The object names should be primary nouns, not synonyms. The complete list of vocabulary words (plus synonyms that I selected) is given below. Refer to that listing for more details.

OLD WEST ADVENTURE DATA BASE LISTING

The following is a listing of the Old West adventure described above. This is really a collected listing of the actions, vocabulary, rooms, messages and objects listed separately above. The only things different about this listing is that the actions and vocabulary have been ordered. The order of action entries is not generally critical, except where previously noted. Placing the automatic action entries before the player input actions is the only requirement.

You will note that some synonyms have been added to the vocabulary words. These are self-explanitory. For example, the player might use the noun "TRADE" or "POST" for the trading post.

Action Entries

Ac#	Verb	Noun	Cond 1	Cond 2	Cond 3	Cond 4	Cond 5
	Act Title		Cmd 1	Cmd 2	Cmd 3	Cmd 4	
0	: AUTO	100	BIT 3	PAR 6	0	0	0
	Player arrest		MSG9	DELAY	GOTOY	CONT	
1	: AUTO	0	0	0	0	0	0
			MSG10	DSPRM	FINI	-	
2	: AUTO	100	BIT 4	-RM0 17	0	0	0
	Outlaw kill		MSG14	DEAD	FINI	-	
3	: AUTO	100	IN 7	PAR 4	0	0	0
	In Bank?		SETZ	-	-	-	
4	: AUTO	100	-BIT 31	PAR 31	PAR 100	0	0
	Initialize		SETZ	CT<-N	-	-	
5	: GO	TEEPEE	IN/W 3	PAR 1	0	0	0
			GOTOY	-	-	-	
6	: TRADE	WHISKEY	HAS 2	PAR 1	PAR 2	0	0
			DROPX	X-RM0	-	-	

Action Entries (con't)

7 :	SHOOT	BUFFALO	HAS 7	IN/W 4	PAR 4	PAR 5	0
			EXX,X	MSG1	MSG2	-	
8 :	SKIN	BUFFALO	HAS 8	IN/W 5	RM0 6	PAR 6	0
			DROPX	-	-	-	
9 :	EXAM	WALLET	HAS 12	0	0	0	0
			MSG3	DSPCT	MSG4	-	
10 :	BUY	RIFLE	IN/W 7	HAS 12	-BIT 0	0	PAR 50
			MSG5	SETZ	CT-N	-	
11 :	BUY	KNIFE	IN/W 8	HAS 12	-BIT 1	PAR 1	PAR 10
			MSG6	SETZ	CT-N	-	
12 :	BUY	WHISKEY	IN/W 2	HAS 12	-BIT 2	PAR 2	PAR 3
			MSG7	SETZ	CT-N	-	
13 :	GET	RIFLE	IN/W 7	BIT 0	PAR 7	0	0
			GETX	-	-	-	
14 :	GET	KNIFE	IN/W 8	BIT 1	PAR 8	0	0
			GETX	-	-	-	
15 :	GET	WHISKEY	IN/W 2	BIT 2	PAR 2	0	0
			GETX	-	-	-	
16 :	GET	RIFLE	IN/W 7	-BIT 0	PAR 3	0	0
			SETZ	-	-	-	
17 :	GET	KNIFE	IN/W 8	-BIT 1	PAR 3	0	0
			SETZ	-	-	-	
18 :	GET	WHISKEY	IN/W 2	-BIT 2	PAR 3	0	0
			SETZ	-	-	-	
19 :	EXAM	JAIL	IN/W 15	0	0	0	0
			MSG11	-	-	-	
20 :	READ	POSTER	IN/W 15	0	0	0	0
			MSG12	-	-	-	
21 :	DRAW	PISTOL	HAS 19	IN/W 17	PAR 17	PAR 20	PAR 8
			X-RM0	X->Y	CONT	MSG13	
22 :	AUTO	0	PAR 21	PAR 8	PAR 22	PAR 18	0
			X->Y	EXX,X	-	-	
23 :	GO	JAIL	IN 5	PAR 8	0	0	0
			GOTOY	-	-	-	
24 :	GO	BANK	IN 5	PAR 7	0	0	0
			GOTOY	-	-	-	
25 :	GO	TRADE	IN 5	PAR 2	0	0	0
			GOTOY	-	-	-	
26 :	EXAM	ANY	0	0	0	0	0
			MSG16	-	-	-	
27 :	HELP	ANY	0	0	0	0	0
			MSG15	-	-	-	
28 :	SAVE	GAME	0	0	0	0	0
			SAVE	-	-	-	
29 :	QUIT	ANY	0	0	0	0	0
			FINI	-	-	-	
30 :	INVEN	ANY	0	0	0	0	0
			INV	-	-	-	
31 :	GET	INVEN	0	0	0	0	0
			INV	-	-	-	
32 :	SCORE	ANY	0	0	0	0	0
			SCORE	-	-	-	

Vocabulary Entries

Voc#	Verb	Noun	Voc#	Verb	Noun
0 :	AUTO	ANY	15:	SHOOT	COUNT
1 :	GO	NORTH	16:	SKIN	JAIL
2 :	*WALK	SOUTH	17:	SCORE	POSTER
3 :	*ENTER	EAST	18:	DROP	PISTOL
4 :	EXAM	WEST	19:	*LEAVE	BANK
5 :	*LOOK	UP	20:	BUY	TRADE
6 :	DRAW	DOWN	21:	*PURCH	*POST
7 :	HELP	TEEPEE	22:	READ	OUTLAW
8 :	SAVE	WHISKEY	23:		GAME
9 :	QUIT	*FIRE	24:		INVEN
10:	GET	BUFFALO	25:		WAMPUM
11:	*TAKE	*HIDE	26:		WALLET
12:	*GRAB	RIFLE	27:		GOLD
13:	INVEN	*WINC	28:		*NUGGET
14:	TRADE	KNIFE			

Room Entries

Rm#	N	S	E	W	U	D	Room Description
0 :	0	0	0	0	0	0	Storeroom. Can't get here
1 :	0	0	0	3	0	0	teepee
2 :	0	0	5	0	0	0	Trading Post
3 :	4	0	0	0	0	0	great plain
4 :	0	3	5	0	0	0	*I'm on a trail
5 :	0	0	0	4	0	0	*I'm on a city street
6 :	0	0	0	0	0	0	Jail Cell
7 :	0	5	0	0	0	0	Bank
8 :	5	0	0	0	0	0	Jail
9 :	0	0	0	0	0	0	
10:	0	0	0	0	0	0	
11:	0	0	0	0	0	0	
12:	0	0	0	0	0	0	*I'm at Boot Hill (six feet under)

Message Entries

Msg#	Message
0 :	
1 :	BANG!!!
2 :	Got em!!!
3 :	It has
4 :	dollars in it
5 :	That'll be $50 partner. It's yours.
6 :	That'll be $10 partner. It's yours.
7 :	That'll be $3 partner. It's yours.
8 :	OK
9 :	Storekeeper cries "STOP, THIEF!!!"
10:	Marshall arrests me for stealing!!

Message Entries (con't)

```
11:    There's a Wanted Poster on it
12:    WANTED:  DEAD OR ALIVE
   The Butler Gang
REWARD:  A Gold Nugget!
13:    The outlaws dropped their guns and the marshall arrested them!
14:    The outlaws got the drop on me!  I'm dead!!
15:    Try EXAMining things
16:    I see nothing special
17:    Welcome to the "Old West" adventure by Bruce G. Hansen
```

Object Entries

Obj#	Start Room	Object Description
0 :	1	Indians
1 :	0	*WAMPUM*/WAM/
2 :	2	Whiskey/WHI/
3 :	3	Teepee
4 :	3	Buffalo
5 :	0	Dead Buffalo
6 :	0	*BUFFALO HIDE*/BUF/
7 :	2	Winchester rifle/WIN/
8 :	2	Knife/KNI/
9 :	0	Lighted artificial light source
10:	2	Counter
11:	2	Mean-looking storekeeper
12:	-1	Wallet/WAL/
13:	6	Bars
14:	6	Gallows being built in the street
15:	5	Jail
16:	5	Bank
17:	7	Outlaws
18:	7	Townsfolk with their hands up
19:	-1	Pistol/PIS/
20:	0	Outlaws behind bars
21:	0	*GOLD NUGGET*/GOL/
22:	0	Happy Townsfolk
23:	5	Trading Post

If you look closely at the data base above, you will notice one data base section is not listed, the HEADER. We shall assign the HEADER values now.

So far we have determined the following HEADER values:

Number of objects = 23
Number of actions = 32
Number of vocabulary words = 28
Number of rooms = 12
Number of messages = 17
Number of treasures = 3
The Word Length = 3

Where did these values come from? Look at the data base listing. In it you will see 23 objects, 32 actions, 28 vocabulary words, 12 rooms, 17 messages and 3 treasures. We set the word length to 3 at the start of writing the adventure.

The remaining HEADER items are the carry limit, starting room, time limit and the treasure room.

Let's set the "carry limit" to 5. This means the player will not be able to carry more than 5 objects at one time. This value can be changed to a greater or smaller number later on if need be. It should not be set to less than 2 since the player is carrying two objects at the start of the adventure.

Let's start the player in room 5, or in the city street. This is arbitrary at this point and can be changed later if the need arises.

The time limit can be set to any value. This value is normally the number of moves that the artificial light source lasts. Since the light source is not used in this adventure, we do not care what this value is. In this case, we will set it to 100.

For the treasure room, I chose room number 2 or the "Trading Post". This also could be changed later. To win at this adventure, the player will have to store the treasures in room 2.

Now all of the HEADER values have been defined. In the ADVEDT format for listing the HEADER, it appears as follows:

#OBJ	#ACT	#VOC	#RM	MAX	BEG	#TR	WLEN	TIME	#MSG	TR-RM
23	32	28	12	5	5	3	3	100	17	2

The whole adventure data base has been listed. The next step is to start typing the adventure in with the adventure editor, ADVEDT.

ENTERING THE ADVENTURE WITH ADVEDT

I will assume by this point you know how to boot up your system and load the adventure editor, ADVEDT. If not, consult Chapter 4 for instructions on doing so.

The first step after loading ADVEDT is to clear out any previous adventure from memory. This is done with the CLEAR command. From the main menu of ADVEDT, the procedure goes as on following page (your inputs are underlined):

Do you want to READ, LIST, PRINT, XREF, WRITE, MODIFY, INSERT,
CLEAR or END
R, L, P, X, W, M, I, C or E ? C

(Press the "C" key for the CLEAR command)

The program will respond with:

Are you sure ? Y <ENTER>

(Press "Y" for yes and press the <ENTER> key)

After a short pause, the main menu will be displayed again. We first issued a CLEAR command to erase the adventure data base memory. If an adventure had previously been read in, we should erase it before entering the "Old West" adventure.

The next step is to set the HEADER to the values listed above. To change the HEADER, and the other data base sections, we use the MODIFY command. To set the HEADER to the appropriate values listed above, perform the following steps (again, your input is underlined):

R, L, P, X, W, M, I, C or E ? M

(Press the "M" key for MODIFY from the main menu)

ADVEDT will respond with:

Which section do you want to MODIFY:
Header, Actions, Vocab, Rooms, Messages or Objects
H, A, V, R, M or O ? H

(Press "H" to MODIFY the HEADER)

ADVEDT responds with:

```
 0 Actions Now -- How many do you want ? 32
18 Vocabulary Now -- How many do you want ? 28
 1 Rooms Now -- How many do you want ? 12
 0 Messages Now -- How many do you want ? 17
 9 Objects Now -- How many do you want ? 23
 0 Carry Limit Now -- How many do you want ? 5
 0 Start Room Now -- How many do you want ? 5
 0 Treasures Now -- How many do you want ? 3
 0 Word length Now -- How many do you want ? 3
 0 Light duration Now -- How many do you want ? 100
 0 Treasure room Now -- How many do you want ? 2
```

The values entered here were the same as those listed for the HEADER above. The value at the beginning of a line is the current value of that item. For example:

```
18 Vocabulary Now -- How many do you want ?
```

The HEADER value for the number of vocabulary words is currently "18". That means there are up to 18 verbs and 18 nouns defined at this point. If we were to type a number, such as the "28" input above, the HEADER value for the number of vocabulary words would change to that inputted number. If you were to press the "<ENTER>" key without typing in a number, the number of vocabulary words would remain at the current value or "18". The same goes for the other HEADER items.

After all of the HEADER has been modified, ADVEDT returns to the MODIFY sub-menu. The order of the data base sections to be modified is not critical, except the vocabulary words must be entered before the action entries. When an action entry is entered, you will type in a verb and a noun. ADVEDT will scan the list of vocabulary words for a match. If no match is found, you are told that the inputted verb and/or noun is not in the vocabulary. When entering this adventure data base into ADVEDT, we will MODIFY the data base sections in this order: Vocabulary, Rooms, Messages, Objects and Action entries.

To MODIFY the vocabulary, we must press the "V" key from the MODIFY sub-menu. The procedure goes as follows (your inputs are underlined):

```
Which section do you want to MODIFY:
Header, Actions, Vocab, Rooms, Messages or Objects
H, A, V, R, M or O ? V

(Press "V" for Vocabulary)

Lower limit, upper limit (ENTER is all) ? <ENTER>

(Press <ENTER> so we can MODIFY all vocabulary words)

Modify
Verbs or Nouns V or N ? V

(Press "V" to modify the verbs)

Verb 0  :  AUTO    New verb ? <ENTER>
Verb 1  :  GO      New verb ? <ENTER>
Verb 2  :          New verb ? *walk
Verb 3  :          New verb ? *enter
Verb 4  :          New verb ? exam
Verb 5  :          New verb ? *look
Verb 6  :          New verb ? draw
Verb 7  :          New verb ? help
Verb 8  :          New verb ? save
Verb 9  :          New verb ? quit
Verb 10 :  GET     New verb ? <ENTER>
Verb 11 :          New verb ? *take
Verb 12 :          New verb ? *grab
Verb 13 :          New verb ? inven
```

Verb modification (con't)

```
Verb 14 :          New verb ? trade
Verb 15 :          New verb ? shoot
Verb 16 :          New verb ? skin
Verb 17 :          New verb ? score
Verb 18 :  DROP    New verb ? <ENTER>
Verb 19 :          New verb ? *leave
Verb 20 :          New verb ? buy
Verb 21 :          New verb ? *purch
Verb 22 :          New verb ? read
Verb 23 :          New verb ? <ENTER>
Verb 24 :          New verb ? <ENTER>
Verb 25 :          New verb ? <ENTER>
Verb 26 :          New verb ? <ENTER>
Verb 27 :          New verb ? <ENTER>
Verb 28 :          New verb ? <ENTER>
```

After the 28 verbs are entered, ADVEDT returns to the MODIFY sub-menu. Note above that the verbs were input in lower case letters. The adventure driver program requires them to be in upper case letters. ADVEDT will convert them from lower case to upper case automatically. I set my keyboard to lower case (the method of doing so depends on your computer set up) since room descriptions, messages and object descriptions contain mostly lower case letters.

Now we can modify the nouns. The nouns are part of the vocabulary, so we must MODIFY that section of the data base:

```
Which section do you want to MODIFY:
Header, Actions, Vocab, Rooms, Messages or Objects
H, A, V, R, M or O ? V

(Press "V" for vocabulary again)

Lower limit, upper limit (ENTER is all) ? <ENTER>

(We want to MODIFY all of the nouns so we press <ENTER> to do so)

Modify Verbs or Nouns V or N ? N

(Press "N" to MODIFY the Nouns)

Noun 0  :   ANY      New noun ? <ENTER>
Noun 1  :   NORTH    New noun ? <ENTER>
Noun 2  :   SOUTH    New noun ? <ENTER>
Noun 3  :   EAST     New noun ? <ENTER>
Noun 4  :   WEST     New noun ? <ENTER>
Noun 5  :   UP       New noun ? <ENTER>
Noun 6  :   DOWN     New noun ? <ENTER>
Noun 7  :            New noun ? teepee
Noun 8  :            New noun ? whiskey
Noun 9  :            New noun ? *fire
Noun 10 :            New noun ? buffalo
Noun 11 :            New noun ? *hide
```

Noun modification (con't)

```
Noun 12 :          New noun ? rifle
Noun 13 :          New noun ? *winc
Noun 14 :          New noun ? knife
Noun 15 :          New noun ? count
Noun 16 :          New noun ? jail
Noun 17 :          New noun ? poster
Noun 18 :          New noun ? pistol
Noun 19 :          New noun ? bank
Noun 20 :          New noun ? trade
Noun 21 :          New noun ? *post
Noun 22 :          New noun ? outlaw
Noun 23 :          New noun ? game
Noun 24 :          New noun ? inven
Noun 25 :          New noun ? wampum
Noun 26 :          New noun ? wallet
Noun 27 :          New noun ? gold
Noun 28 :          New noun ? *nugget
```

After all of the nouns have been modified, ADVEDT returns to the MODIFY sub-menu. The next data base section to be modified is the rooms:

```
Which section do you want to MODIFY:
Header, Actions, Vocab, Rooms, Messages or Objects
H, A, V, R, M or O ? R

(Press "R" to MODIFY the rooms)

Lower limit, upper limit (ENTER is all) ? <ENTER>

(Press <ENTER> to MODIFY all of the rooms)

Room 0 : 0  N 0  S 0  E 0  W 0  U 0  D
Room description:  Storeroom, can't get here
N, S, E, W, U, D rooms ? <ENTER>
Description ? <ENTER>
```

(Note: the current values for the rest of the rooms will be left off here to save space. The room number being modified will be inside paranthesis to identify the room. Because the CLEAR command was issued above, all of their room direction data will be zeros. The room descriptions will be blanks.)

```
(room 1) N, S, E, W, U, D  rooms ? 0 0 0 3 0 0
Description ? teepee

(room 2) N, S, E, W, U, D  rooms ? 0 0 5
Description ? Trading Post

(room 3) N, S, E, W, U, D  rooms ? 4
Description ? great plain

(room 4) N, S, E, W, U, D  rooms ? 0 3 5
Description ? *I'm on a trail
```

Room modification (con't)

(room 5) N, S, E, W, U, D rooms ? 0 0 0 4
Description ? *I'm on a city street

(room 6) N, S, E, W, U, D rooms ? <ENTER>
Description ? Jail Cell

(room 7) N, S, E, W, U, D rooms ? 0 5
Description ? Bank

(room 8) N, S, E, W, U, D rooms ? 5
Description ? Jail

(room 9) N, S, E, W, U, D rooms ? <ENTER>
Description ? <ENTER>

(room 10) N, S, E, W, U, D rooms ? <ENTER>
Description ? <ENTER>

(room 11) N, S, E, W, U, D rooms ? <ENTER>
Description ? <ENTER>

(room 12) N, S, E, W, U, D rooms ? <ENTER>
Description ? *I'm at Boot Hill (six feet under)

After these rooms are modified, ADVEDT returns to the MODIFY sub-menu.

If you were following along by actually entering this adventure with ADVEDT, you probably noticed that all previous room directions were set to zeros (like room 0 above). For some of the rooms, such as room 2, we did not input all six directions for North, South, East, West, Up and Down. Let's take room 2 as an example. The following table details room 2's adjacent room directions:

	N	S	E	W	U	D
Previous room directions	0	0	0	0	0	0
Desired room directions	0	0	5	0	0	0
Room directions entered	0	0	5			

When the room directions are typed in with ADVEDT, only the number of room directions typed in are changed. For example, with room 2, we typed in only three room directions. So ADVEDT only changed the first three. We could have typed in three zeros separated by a space after the "5" room number, but we didn't have to since the West, Up and Down room numbers were already zero, the desired room direction. This can save a lot of typing.

The next section of the data base to be modified is the messages. Modifying them goes as follows:

Which section do you want to MODIFY:
Header, Actions, Vocab, Rooms, Messages or Objects
H, A, V, R, M or O ? M

(Press "M" for the messages)

```
Lower limit, upper limit (ENTER is all) ? <ENTER>
```

(Press <ENTER> because we want to modify all of the messages)

```
Message 0  :
Message ? <ENTER>
Message 1  :
Message ? BANG!!!
Message 2  :
Message ? Got em!!!
Message 3  :
Message ? It has
```

Continue to enter the messages as listed previously. The only "special" message is message 12. This message entry is:

```
Message 12 :
Message ? WANTED:  DEAD OR ALIVE<DN ARROW>
    The Butler Gang<DN ARROW>
REWARD:  A Gold Nugget!
```

The "<DN ARROW>" above signifies the "DOWN ARROW" key. Pressing this key lets you continue with the message on the beginning of the next line.

After all of the messages are entered, ADVEDT returns to the MODIFY sub-menu. The next data base section to be modified is the objects:

```
Which section do you want to MODIFY:
Header, Actions, Vocab, Rooms, Messages or Objects
H, A, V, R, M or O ? O
```

(Press "O" to MODIFY the objects)

```
Lower limit, upper limit (ENTER is all) ? <ENTER>
```

(Press <ENTER> to MODIFY all of the objects)

```
Object 0  :  0   Start
Starting room = ? 1
Description ? Indians
Object 1  :  0   Start
Starting room = ? 0
Description ? *WAMPUM*/WAM/
Object 2  :  0   Start
Starting room = ? 2
Description ? Whiskey/whi/
Object 3  :  0   Start
Starting room = ? 3
Description ? Teepee
```

Continue entering the objects as listed above. Remember to press the <ENTER> key for the starting room and description of object 9, the lighted artificial light source. Since this object is not used in this adventure, we can leave its description and starting room as is.

Notice that the object name of object 2, the whiskey, was entered in lower case ("whi") letters. ADVEDT will automatically convert the lower case "whi" to the required upper case "WHI". This feature makes typing object names easier for upper/lower case users.

After entering all of the objects, ADVEDT will return to the MODIFY sub-menu. The last section of the data base to be modified is the action entries:

```
Which section do you want to MODIFY:
Header, Actions, Vocab, Rooms, Messages or Objects
H, A, V, R, M or O ? A
(Press "A" to MODIFY the actions)

Lower limit, upper limit (ENTER is all) ? <ENTER>

ACTION 0   :   AUTO      0
VERB NOUN ? auto 100
PAR 0   COND, VALUE ? bit 3
PAR 0   COND, VALUE ? par 6
PAR 0   COND, VALUE ? <ENTER>
PAR 0   COND, VALUE ? <ENTER>
PAR 0   COND, VALUE ? <ENTER>
    -      Cmd or Msg# ? 9
    -      Cmd or Msg# ? delay
    -      Cmd or Msg# ? gotoy
    -      Cmd or Msg# ? cont

  Title ? Player arrest
Y or N ? Y
```

Continue typing in the rest of the action entries in this fashion. Notice above how the verb "auto" was entered in lower case letters. ADVEDT automatically converts the "auto" to "AUTO" for you. The same goes for the conditions and commands. Also note that the first command was entered as simply the number "9". This means "Message 9". When entering a message as a command, you enter only the message number. In this case, this was a "9".

After all of the actions have been entered, ADVEDT returns to the MODIFY sub-menu. It might be a good idea at this point to save our adventure data base to disk (or tape for tape users). First, we need to exit the MODIFY sub-menu and return to the main menu. This is done by pressing the <BREAK> key or SHIFT RIGHT ARROW keys:

```
Which section do you want to MODIFY:
Header, Actions, Vocab, Rooms, Messages or Objects
H, A, V, R, M or O ? <BREAK>

Do you want to READ, LIST, PRINT, XREF, WRITE, MODIFY, INSERT,
CLEAR or END
R, L, P, X, W, M, I, C or E ? W

(Press "W" for the WRITE command)
```

Tape or Disk ? D

(Press "D" for a disk write. Note that tape users do not have a choice, they can only write to tape.)

Adventure name and drive # ? OW:0 <ENTER>

(We will save this adventure data base out with the name "OW" on drive# 0. The drive number is optional. If writing the data base to tape, no adventure name can be specified.)

After a few seconds, ADVEDT will return to the main menu. Now we can continue the adventure writing procedure with the debugging prcedure.

DEBUGGING THE ADVENTURE WITH ADVEDT

Now that our adventure has been typed in ADVEDT, we can debug it. To debug an adventure, we must play it and try just about every conceivable thing. Since we have saved our adventure to disk, we could exit ADVEDT and play the adventure with the driver program, ADV/CMD (tape users do not have ADV/CMD). A much simplier method is to press the <CLEAR> key.

Pressing the <CLEAR> key while running ADVEDT will call up the adventure driver built into ADVEDT. Let's press the <CLEAR> key now.

You should have gotten the message:

Use old "SAVED" game ? _

If you did not get this message, run through this chapter again to see where you went wrong. Since we have not previously saved an "Old West" game in progress, we can not read one in. The WRITE command done previously saved the whole data base to disk. This inquiry lets you continue a game from some point you had saved out earlier.

We do not want to read in a "SAVED" game, so we respond with:

Use old "SAVED" game ? N <ENTER>

The screen will clear and display:

ADVEDT DRIVER ROUTINE

Press the "CLEAR" key at any time to return to the editor.

Hit ENTER to start the ADVENTURE _

This messages says if you press the <CLEAR> key will running the adventure driver, you will be returned to the editor with your adventure data base intact. This makes debugging much simplier.

Press the <ENTER> key now to start the adventure. You should see the following on the screen:

```
I'm on a city street. Visible items:

Jail. Bank. Trading Post.

Obvious exits are:  WEST
<---------------------------------------------------->


Tell we what to do --> _
```

This is the type of display you get while playing an adventure. Remember how we told the HEADER that the player was to start the adventure in room 5? If you look at the room description for room 5, you will see that it is displayed at the top of the screen, "I'm on a city street".

Also, remember that we set the starting room for object 15, (the "Jail"), object 16 (the "Bank") and object 23 (the "Trading Post") to room 5? Sure enough, we have started the adventure in room 5 and those objects are in room 5.

Now look at the room directions for room 5. The only exit from this room is to the West, which will move us to room 4. Now look at the sample adventure screen above. Notice how the only obvious exit is to the West?

All of that information is called the "Room Display" because it displays a description of the room, the objects in the room and the exits from the room. Now let's move to the West. This can be done by typing "GO WEST" or by the shorthand "W". We will use the "GO WEST" method (your inputs are underlined from now on):

```
Tell me what to do --> GO WEST
```

The screen will now have the following display:

```
I'm on a trail

Obvious exits are:   SOUTH EAST
<---------------------------------------------------->

Tell me what to do --> GO WEST
Tell me what to do --> _
```

We have just moved West from room 5 to room 4. Let's do an INVENTORY command now. This command will tell us what, if anything, we are carrying. This can be done by typing "INV" or "GET INV" or the shorthand "I". We wrote the actions to perform the "INV" and "GET INV" functions:

```
Tell me what to do --> INV
```

The screen will appear as:

```
I'm on a trail

Obvious exits are:   SOUTH EAST
<---------------------------------------------------------->


Tell me what to do --> GO WEST
Tell me what to do --> INV
I'm carrying the following:
Wallet. Pistol.
Tell me what to do --> _
```

Let's return to the city street now. We do this by moving to the East (we previously moved to the trail by moving West, so we are backtracking).

```
Tell me what to do --> E
```

The screen will be:

```
I'm on a city street. Visible items:

Jail. Bank. Trading Post.

Obvious exits are:   WEST
<---------------------------------------------------------->


Tell me what to do --> GO WEST
Tell me what to do --> INV
I'm carrying the following:
Wallet. Pistol.
Tell me what to do --> E
Tell me what to do --> _
```

To ease the reading of the display, we will ignore the previously "Tell me what to do" lines. You will notice after enough player inputs are done, that only the lines below the dashes scroll. This leaves the "Room Display" in tact.

Now we'll go into the "Trading Post". We enter this room by typing "GO TRAD" or "GO POST". The action entry which allows us to enter the trading post reads "GO TRADE". But in the nouns, we made "POST" a synonym of "TRADE", so either should work. We will enter the trading post with a "GO POST" verb, noun combination:

```
Tell me what to do --> GO POST
```

But we got back the message:

```
I'm sorry, but I don't understand what you mean.
```

This is our first "bug". There is a problem with the nouns which does not let "POST" act as a synonym of "TRADE". We will look for the specific cause in a bit. Let's play the adventure a little more to see if there are any other problems. Let's continue by going into the trading post:

Tell me what to do --> GO TRADE

The room display will be:

I'm in a Trading Post. Visible items:

Whiskey. Winchester rifle. Knife. Counter.
Mean-looking storekeeper.

Obvious exits are: EAST
<---------->

Tell me what to do --> _

When debugging an adventure, you need to test every one of your action entries. In this adventure, if the player tries to get the whiskey, rifle or knife before buying it, he is arrested. Arresting the player will end the adventure, so we will save the game at this point. Then we can see if the arrest actions work. If they do, we can restore the saved game and continue debugging from this point. The room display does not change on a "SAVE GAME" verb, noun so we won't redisplay it:

Tell me what to do --> SAVE GAME

Now we'll get the rifle without buying it first:

Tell me what to do --> GET RIFLE

The following display will be seen:

I'm in a Jail Cell. Visible items:

Bars. Gallows being built in the street.

<---------->

Tell me what to do --> SAVE GAME
Saving game.
Tell me what to do --> GET RIFLE
Storekeeper cries "STOP, THIEF!!!"
Marshall arrests me for stealing!!
This game is over. Play again ? _

It looks like the arrest action works. Now let's say we do want to play again (type "Y" and press <ENTER>). The adventure driver will ask if we want to play a saved game. This time we do:

Use old "SAVED" game ? Y

The saved game will be read in. The <ENTER> key should be pressed to start the game. Notice that the game did not start at room 5, the normal starting room of the adventure. The game is being continued from the point where we saved it above.

Now let's buy the "Winchester rifle". We do this by typing "BUY RIFLE":

```
Tell me what to do --> BUY RIFLE
```

We get the following room display:

```
I'm in a Trading Post. Visible items:

Whiskey. Winchester rifle. Knife. Counter.
Mean-looking storekeeper.

Obvious exits are:   EAST
<---------------------------------------------------->

Tell me what to do --> BUY RIFLE
That'll be $50 partner.  It's yours.
Tell me what to do --> _
```

Now that we have bought the rifle, let's try to pick it up. Hopefully, our "arrest the player" actions will let us pick it up without sending us to jail.

```
Tell me what to do --> GET RIFLE
```

The room display will be:

```
I'm in a Trading Post. Visible items:
Whiskey. Knife. Counter. Mean-looking storekeeper.

Obvious exits are:   EAST
<---------------------------------------------------->

Tell me what to do --> GET RIFLE
Tell me what to do --> _
```

Notice that the "Winchester rifle" is no longer in the room display. This is because we are carrying it. Take an inventory now, you will see the rifle there which means we are carrying it. Let's try to drop the rifle. We type:

```
Tell me what to do --> DROP RIFLE
```

But we got back the message:

```
Tell me what to do --> DROP RIFLE
It's beyond my power to do that
```

There is some sort of problem or "bug" here. We should be able to drop the rifle since it is a named object and there are no "DROP RIFLE" actions.

Now that we have a few "bugs", let's try to find what is causing them. First, we must return to the editor. This is done by pressing the <CLEAR> key. After you press the <CLEAR> key, the main menu of ADVEDT should be on the screen:

```
Do you want to READ, PRINT, XREF, WRITE, MODIFY, INSERT,
CLEAR or END
R, L, P, X, W, M, I, C or E ? _
```

We will fix the "GO POST" bug first. We need to study the list of nouns tabulated previously. As you scan through the list, take a close look at noun 17 ("POSTER").

The word length of this adventure is three, so "POSTER" is interpreted by the adventure driver to be "POS" (three letters long). When we typed in "GO POST", the "POST" was truncated to three letters also, or "POS". When the adventure driver scanned the nouns for a match of "POS", a match was made with noun 17 ("POSTER") and noun 21 "*POST" was never reached.

The easiest way to fix this "bug" is to add an action with the verb, noun combination of "GO POSTER". If the player then types "GO POST", the "POST" will be truncated to "POS" and the "GO POSTER" action will be considered. This action should have the same conditions and commands of the "GO TRADE" action. This action would be:

```
GO POSTER    IN 5    PAR 2      0        0        0
             GOTOY     -        -        -
```

Before we add this action, let's fix the "DROP RIFLE" action. Again, we refer to the "Old West" adventure listing given above. Object 7 is described as:

Obj#	Start room	Description
7	2	Winchester rifle/WIN/

The possible causes for us not being able to pick up the rifle are:

1) There is a "DROP RIFLE" action and it's conditions are not true.
2) The object name "WIN" is not in the nouns.
3) The object name "WIN" is not a primary noun.

The first case is not true. There are no action entries to drop the rifle. If there were, the automatic get/drop feature would be disabled.

We need to look at the nouns to determine if our problem is case two or case three. Look at noun 12 and noun 13. They are:

Noun#	Noun
12	RIFLE
13	*WINC

It looks like case three is our problem; the object name is not a primary noun, it is a synonym. The solution for this problem is simple, change the object name of object 7 from "/WIN/" to "/RIF/".

We can fix both of our problems now. Let's fix the "GO POST" problem first. We first need to modify the HEADER. Since we are going to add an action entry, we will have to increase the number of actions in the HEADER from 32 to 33.

From the main menu, press the "M" key (for MODIFY):

```
Do you want to READ, PRINT, XREF, WRITE, MODIFY, INSERT,
CLEAR or END
R, L, P, X, W, M, I, C or E ? M

Which section do you want to MODIFY:
Header, Actions, Vocab, Rooms, Messages or Objects
H, A, V, R, M or O ? H

32 Actions Now -- How many do you want ? 33
28 Vocabulary Now -- How many do you want ? <BREAK>
```

We changed the HEADER from 32 to 33 actions. We pressed the <BREAK> key since it was not necessary to change any of the other HEADER values. Now, reenter the MODIFY sub-menu by pressing the "M" key and modify the actions. We want to add an action entry 33:

```
Do you want to READ, PRINT, XREF, WRITE, MODIFY, INSERT,
CLEAR or END
R, L, P, X, W, M, I, C or E ? M

Which section do you want to MODIFY:
Header, Actions, Vocab, Rooms, Messages or Objects
H, A, V, R, M or O ? A
Lower limit, upper limit (ENTER is all) ? 33

ACTION 33 :   AUTO     0
VERB NOUN ? GO POSTER
PAR 0   COND, VALUE ? in 5
PAR 0   COND, VALUE ? par 2
PAR 0   COND, VALUE ? <ENTER>
PAR 0   COND, VALUE ? <ENTER>
PAR 0   COND, VALUE ? <ENTER>
    -       Cmd or Msg# ? GOTOY
    -       Cmd or Msg# ? <ENTER>
    -       Cmd or Msg# ? <ENTER>
    -       Cmd or Msg# ? <ENTER>

  Title ? <ENTER>
Y or N ? Y
```

Now the "GO POSTER" action has been added. To fix the "DROP RIFLE" program, we need to MODIFY the objects and change the object name of object 7, the rifle:

```
Which section do you want to MODIFY:
Header, Actions, Vocab, Rooms, Messages or Objects
H, A, V, R, M or O ? O
Lower limit, upper limit (ENTER is all) ? 7
Object 7  :  2    Start   Winchester rifle/WIN/
Starting room = ? <ENTER>
Description ? Winchester rifle/rif/
```

Now that we have fixed the two bugs, we should save the data base to disk again. First we must press the <BREAK> key to return to the main menu. Then we will use the WRITE command:

```
Which section do you want to MODIFY:
Header, Actions, Vocab, Rooms, Messages or Objects
H, A, V, R, M or O ? <BREAK>

Do you want to READ, LIST, PRINT, XREF, WRITE, MODIFY, INSERT,
CLEAR or END
R, L, P, X, W, M, I, C or E ? W
Tape or Disk
T or D ? D
Adventure name and drive # ? <ENTER>
```

We pressed <ENTER> for the adventure name and drive number so the data base would be written out with the same name as before. We could have manually entered the name too, but I chose not to.

Now you should continue to play the adventure to see if there are any other "bugs". You can continue to play the adventure by first pressing the <CLEAR> key to enter the adventure driver.

In the HEADER, we said that the "Treasure room" was room 2, the "Trading Post". If you go into the trading post, there is no indication that you should leave your treasures there. We should add an object to room 2 which tells the player to leave the treasures there. This object could be:

Obj#	Start room	Description
24	2	Sign saying "Leave *TREASURES* here"

Before we can add this object, the HEADER should be told that there are now 24 objects in this adventure instead of 23. The procedure for adding this object would be:

```
Do you want to READ, PRINT, XREF, WRITE, MODIFY, INSERT,
CLEAR or END
R, L, P, X, W, M, I, C or E ? M

Which section do you want to MODIFY:
Header, Actions, Vocab, Rooms, Messages or Objects
H, A, V, R, M or O ? H
33 Actions Now -- How many do you want ? <ENTER>
28 Vocabulary Now -- How many do you want ? <ENTER>
12 Rooms Now -- How many do you want ? <ENTER>
17 Messages Now -- How many do you want ? <ENTER>
23 Objects Now -- How many do you want ? 24
5 Carry limit Now -- How many do you want ? <BREAK>

Do you want to READ, PRINT, XREF, WRITE, MODIFY, INSERT,
CLEAR or END
R, L, P, X, W, M, I, C or E ? M
```

```
Which section do you want to MODIFY:
Header, Actions, Vocab, Rooms, Messages or Objects
H, A, V, R, M or O ? O
Lower limit, upper limit (ENTER is all) ? 24
Object 24 :  0     Start
Starting room = ? 2 <ENTER>
Description ? Sign saying "Leave *TREASURES* here"
```

Now that object 24 has been added, the data base should be saved out again. It is good practice to regularly save your data base out. It is very frustrating to make a lot of changes to a data base and turn off your computer without having saved the changes! You may have noticed that the opening message was not displayed when the adventure was started. This message (message 17) should be displayed by the initialization action (action 4). I'll let you fix this bug!

You can continue to play the adventure to find any other "bugs". If you want, you can add some actions to this adventure. At the beginning of this chapter we listed twenty items the player might come across in the "Old West". You could add to this adventure by writing about some of those unused items/situations.

The same procedure presented in this chapter could be used to write your own original adventures. After you have written an adventure, it really isn't that fun to play. Since you wrote it, you should be able to solve it. That leads us to the next step of adventure writing.

MARKETING YOUR ADVENTURES

The most important step here is to get the adventure as "bug" free as possible. People can get a low opinion of an adventure if they find a lot of problems with it.

After the adventure is finished, write down the steps needed to solve it. This can be a list of verb, noun combinations which must be typed in to solve the adventure.

OK, the only thing left is to get your adventure to market. Refer to Appendix B for more details.

Chapter 7

SOLVING AN ADVENTURE

This chapter will briefly describe a method for solving adventures using the ADVEDT program. To effectively use this chapter, a good understanding of TAS is recommended.

There are two basic types of adventures: mission and treasure.

In mission adventures, the object of the game is to accomplish a task. In treasure adventures, the object of the game is to collect treasures and store them in the treasure room. When all of the treasures are stored, the game is over.

SOLVING "MISSION" TYPE ADVENTURES

Mission type adventures typically end with a winning message. The first step in solving these types of adventures is to list the messages and find the winning message number. This should be an obvious message. The winning message number should be noted.

Next, do an XREF for that message number in the actions. This procedure will tell you which actions display the winning message. The number(s) of these action(s) should be noted.

Now, list the action(s) which will display the winning message. Note what the conditions are. These conditions must be true before the winning message will be displayed.

The XREF command can be used to find where the objects needed in the winning action are referenced. This will tell you how to get them if they are not simply laying in a room (where they could be picked up). If a bit flag needs to be set before the winning message is displayed, an XREF can be done on that particular bit flag to find out what must be done to set the flag.

The procedure continues in this fashion. It may take a while to get it down pat, but it is basically a simple procedure. It's really solving the adventure in reverse.

SOLVING "TREASURE" TYPE ADVENTURES

These types of adventures are very similar to mission type adventures when solving them.

The first step in solving treasure type adventures is to list the OBJECTS. Note which objects are treasures. Treasures will be the objects with an asterisk as their first descriptive character.

Treasures that have a non-zero starting room number are simply laying in a room. The only potential problem here is that some actions may need to be taken to get into the room. For example, a locked door may block the entrance of the room. By looking at the room descriptions, it can be determined if this room can be moved into from another room (for example, "GO EAST" from another room moves you into the one in question). If not, do an XREF on the room to find what conditions must be true to enter the room.

If the treasure has a starting room number of zero, then some action must take place to drop it in a room. By doing an XREF on the treasure, it can be determined what conditions must be met for the treasure to enter a room so it may be picked up.

The procedure continues for all of the treasures. Again, it's simply solving the adventure in reverse.

However, the best way to solve an adventure is to play it through. If you get stuck, look at the data base as little as possible unless you're fed up with the adventure. Remember, adventures are meant to be brain-teasers.

Apendix A

ADVENTURE COMMAND SUMMARY

CONDITIONS:

PAR	Passes a number to the commands.
HAS	True if holding the object.
IN/W	True if in same room as object (not holding it).
AVL	True if in same room or holding object.
IN	True if in room.
-IN/W	True if holding object or if object is in another room.
-HAVE	True if not holding object.
-IN	True if not in room.
BIT	True if bit flag set.
-BIT	True if bit flag cleared.
ANY	True if holding any objects.
-ANY	True if not holding any objects.
-AVL	True if object in another room.
-RM0	True if object not in room zero.
RM0	True if object in room zero.
CT<=	True if counter less than or equal to number.
CT>	True if counter greater than number.
ORIG	True if object in original starting room.
-ORIG	True if object not in original starting room.
CT=	True if counter equal to number.

COMMANDS:

GETX	Pick up object X.
DROPX	Drop object X.
GOTOY	Move player to room Y.
X->RM0	Send object X to room zero.
NIGHT	Make it night (set bit flag 15).
DAY	Make it day (clear bit flag 15).
SETZ	Set bit flag Z.
CLRZ	Clear bit flag Z.
DEAD	Tell player he's dead, make DAY, move to last room, end game.
X->Y	Send object X to room Y.
FINI	Stop game and ask for another game.
DSPRM	Display current room and account for DAY, NIGHT.
SCORE	Compute and display the score.
INV	Tell the player what he is carrying.
SET0	Set bit flag 0.
CLR0	Clear bit flag 0.
FILL	Fill artificial light source (clear bit flag 16).
SAVE	Save the game.
EXX,X	Exchange room location of object X with object X.
CONT	Continue to next action/s.
AGETX	Always get object X regardless of carry limit status.
BYX->X	Move first object X to same place as first object X.
CT-1	Decrement counter.
DSPCT	Display the counter.

COMMANDS (con't)

```
CT<-N    Set counter equal to N.
EXRM0    Exchange current room with room held in alternate room register 0.
EXM,CT   Exchange counter and alternate counter M.
CT+N     Add N to counter.
CT-N     Subtract N from counter.
SAYW     Say the player's input noun.
SAYWCR   Say the noun of the player's input noun and a carriage return.
SAYCR    Start a new line.
EXC,CR   Exchange current room with room in alternate room register C.
DELAY    Pause for about 1 second.
```

REVERSED PARAMETERS

BIT FLAGS:

15 SET=dark outside
16 SET=artificial light source has run out

VERB:

0 AUTO
1 GO
10 GET
18 DROP

NOUNS:

0 ANY
1 NORTH
2 SOUTH
3 EAST
4 WEST
5 UP
6 DOWN

ROOMS:

0 is reserved as the STOREROOM

MESSAGES:

0 must not be defined (it is not used)

OBJECTS:

9 Lighted Artificial Light Source

MAXIMUMS FOR EACH DATA BASE SECTION:

ACTIONS = 500 (0-499)
VOCABULARY = 150 (0-149)
ROOMS = 99 (1-99)
MESSAGES = 99 (1-99)
OBJECTS = 250 (0-249)
TEXT CHARACTERS = 255 MAXIMUM
ACTION TITLE CHARACTERS = 20 MAXIMUM
WORD LENGTH = 7 MAXIMUM
TIME LIMIT (MOVES) = 32767 MAXIMUM
BIT FLAGS = 32 (0-31)
COUNTERS = 8 (0-7)

Appendix B

SUBMITTING YOUR ADVENTURES FOR MARKETING

All you need to do is send a diskette or tape with the adventure data base(s) to:

THE ALTERNATE SOURCE
704 Pennsylvania Avenue
Lansing, MI 48906

Please tell us if the adventure is being submitted for printing in AUGMENT or for general distribution (or both). AUGMENT is the "Adventure Users' Group Newsletter". We are always seeking adventures for AUGMENT. Payment varies.

The Alternate Source will review the adventure for originality and general bugs. The adventure may not be acceptable because it is not original (a copy of someone else's) or is thought not to be in good taste. Or, heaven forbid, it just may not be up to standards! Would YOU buy the adventure? We recommend that you "play test" adventures thoroughly! Give a few friends a copy and let them wring out some bugs.

The diskette or tape may be returned with some suggestions for improvement. We accept no responsibility for unsolicited programs. Your media will be returned if you include a postpaid mailer with your submission. TAS also reserves the right to make simple changes to the data base to improve its play.

If your adventure is accepted, you will be notified. A contract will be sent to you upon acceptance detailing the royalty payment.

You may decide to market the adventure without going through TAS. If you so choose, remember that the adventure driver program "ADV" is copyrighted and can not be sold with your adventures unless written permission is given by the author.

Please allow at least 4 weeks for the selection process. The selection process is greatly speeded up if you include the steps needed to solve your adventure. This is not required, but it is HIGHLY recommended.

INDEX