the AARDVARK JOURNAL

october 1980 vol. I , no. 4

**** IN THIS ISSUE ****

THE "BEGINNERS' CORNER" KINDA GOT OUT OF HAND AND CONSISTS OF SEVERAL PAGES ON SUBSCRIPTED VARIABLES. DOWN NEAR THE END OF THE ARTICLE THERE IS EVEN SOME INFORMATION THAT MIGHT BE OF INTEREST TO THE MORE EXPERIENCED PROGRAMMERS. IF WE DO THIS AGAIN, WE'LL HAVE TO STOP CALLING IT A "CORNER". MOST OF THE REST OF THE ISSUE IS ON MEMORY SAVING TECHNIQUES, BUT WE ALSO HAVE SOME NEAT ARTICLES SPECIALLY FOR DISK PEOPLE. THE ISSUE CONTAINS A METHOD OF MOVING THE DIRECTORY OFF TRACK 12 AND A PROGRAM TO RENAME BEXEC*.

WE ONLY HAVE ONE PROGRAM THIS MONTH (ASIDE FROM THE DISK STUFF), AS I WANTED TO SQUEEZE IN THE LONG ARTICLE ON SUBSCRIPTED VARIABLES. I'LL TRY TO MAKE UP FOR THAT NEXT MONTH WITH A FEW EXTRA PROGRAMS BUT - I AM GOING TO STOP PLANNING MORE THAN ONE ISSUE AHEAD. THE CHECKBOOK PROGRAM I PROMISED IN ISSUE #2 AND NEVER FOUND SPACE TO PUBLISH WILL NEVER SEE LIGHT OF DAY. ON LONGER USEAGE, IT TURNED OUT TO BE ONLY TWICE AS HARD TO USE AS A TI CALCULATOR FOR THE SAME JOB. THAT WAS AN IMPROVEMENT OVER MOST I HAVE SEEN, BUT NOT GOOD ENOUGH TO PUBLISH.

WE HAVE SOME CORRECTIONS AND ADDITIONS TO THE MIRACLE SCREEN CLEAR IN THE "LETTERS TO THE EDITOR".

THIS MONTHS' PROGRAM

IN WRITING "ASTEROID RACE", I HAD IN MIND DEMONSTRATING THAT A SHORT AND SIMPLE PROGRAM COULD BE FUN. I THINK THAT I SUCCEEDED. IT LOOKS RATHER LONG IN PRINT, BUT MOST OF IT IS REMARKS.

I DON'T EXPECT ANYONE TO RUN IT LONG THE WAY IT WAS WRITTEN. IT IS A NATURAL FOR EASY EXPANSION FOR THE KIDS. PLAY WITH THE NUMBER OF ASTEROIDS, THE TIMING, AND THE SCORING TO START WITH. IMPROVING THE CANNON, ADDING A SECOND SHIP AND IMPROVING THE CONTROLS COULD PROVIDE A FEW HOURS OF FUN. THE CANNON IS, IN PARTICULAR, USELESS. I JUST LEFT IT IN TO PROVIDE A STARTING POINT FOR IMPROVEMENT.

IT IS FUN JUST AS IT IS AND TYPES IN A HURRY IF YOU LEAVE THE REMS OUT. AS WITH ALL MY PROGRAMS, THE REMS ARE REMOVABLE.

THE THINGS TO WATCH IN THIS PROGRAM ARE THE USE OF FLAGS TO INDICATE MISSLE MOVEMENT, THE WAY TWO LINES (180-190) WILL CHANGE IT FROM C2/4/8 TO C1P, AND THE INTERMIXING OF PRINT AND POKE GRAPHICS.

SAVING MEMORY

WARNING:HERESY

ELSEWHERE IN THIS EDITION WE PLAN TO INCLUDE AN ARTICLE BY BOB RETELLE ON THE TECHNIQUES HE USES IN HIS PROGRAMS TO SAVE SPACE. WHILE IT IS A GOOD ARTICLE, AND I WOULD OF COURSE RECOMMEND THAT YOU READ IT CAREFULLY, I THINK THE TOPIC IS IMPORTANT ENOUGH TO WARRANT EVEN FURTHER ATTENTION.

DESPITE THE FACT THAT SYSTEMS ARE GETTING LARGER, MEMORY REMAINS A CRITICAL PROBLEM FOR THE MINI-SYSTEM. A YEAR AGO THE STANDARD SYSTEM WAS ONLY 4K IN SIZE. IT NOW IS APPROXIMATELY 8K AND DISK SYSTEMS USUALLY RUN 12 TO 14K OF USABLE SPACE. UNFORTUNATELY, MOST COMPUTER CLASSES ARE TAUGHT BY AND MOST COMPUTER BOOKS ARE WRITTEN BY UNIVERSITY PROFESSORS WHO WORK ON MEGABYTE TIMESHARING SYSTEMS. THEREFORE, LITTLE HELP IS AVAILABLE FOR THE COMPUTERIST FACING THE REAL EVERYDAY PROBLEM THAT HE HAS ALL THE MEMORY THAT HE CAN AFFORD NOT ALL OF WHAT GIGANTIC UNIVERSITY CAN AFFORD.

BEFORE WE BEGIN, LET'S INTERJECT A WORD OF CHEER. WHILE WE ARE CONTINUOUSLY PUSHED BY LACK OF MEMORY SIZE, WE ACTUALLY HAVE A LOT MORE MEMORY THAN WAS COMMON EVEN ON MAIN FRAME SYSTEMS 10 - 15 YEARS AGO. THOSE OF YOU WHO HAVE BEEN AROUND A WHILE WILL REMEMBER THAT NOT TOO LONG AGO WE RAN LARGE BUSINESSES AND LARGE BOOKKEEPING PROGRAMS ON SYSTEMS THAT HAD WELL UNDER 4K OF ACCESSABLE RAM. THE SYSTEMS THAT WE CURRENTLY HAVE IN OUR HOMES AND AT OUR BUSINESSES THEN ARE CAPABLE OF RATHER EXTENSIVE WORK. HOWEVER, WE NEED TO IMPLEMENT THEM WITH THE GREATEST POSSIBLE EFFICIENCY EVEN IN THIS DAY OF SLIGHTLY CHEAPER MEMORY.

THERE ARE SEVERAL NEW AND POPULAR CONCEPTS BEING PUSHED BY EVERYONE FROM WAYNE GREEN TO GOD WHICH APPEAR AN ANATHEMA FOR THE SMALL COMPUTERIST. AMONG THE WORST CONCEPTS ARE "TOP DOWN" PROGRAMMING AND "STRUCTURED" PROGRAMMING. STRUCTURED PROGRAMMING IS A NICE IDEA IF YOU HAVE MEGABYTE OR SO OF MEMORY AT HAND BUT IT IS BASICALLY OR PRIMARILY A WAY TO HAVE TO AVOID DOCUMENTING A PROGRAM. FOR THE SUPERVISORS WHO PUSH FOR IT IN THEIR ESTABLISHMENTS, IT IS MAINLY A WAY TO AVOID THE RESPONSIBILITY OF MAKING A PROGRAMMER WRITE DOWN WHAT HE DID. IT IS EASIER TO IMPOSE AN OVERALL SYSTEM THAN TO INSIST THAT THE PROGRAMMER DOCUMENT TODAY THE CODE THAT HE WROTE THIS MORNING. THE ONLY PARTICULAR VALUE THAT SUCH CONCEPTS HAVE IS THE PEOPLE WHO FOLLOW NEED NOT LEARN TO READ PROGRAMMING AND THOSE WHO PRECEDE NEED NOT LEARN TO FLOW CHART OR WRITE IT DOWN. ANOTHER CONCEPT IS STILL BEING PUSHED BY A LOT OF PEOPLE WHICH IS PURE FOOLISHNESS. THE WHOLE CONCEPT OF 'STYLE IN BASIC'. I REPEAT HERE, AS I HAVE SAID BEFORE, THAT THE BEST WAY TO HELP THE ENERGY CRISIS IS TO BURN THOSE BLASTED BOOKS OF STYLE. AS A COMPUTERIST, YOU ARE FACED WITH THE DESIRES OF A NUMBER OF PEOPLE. YOUR CHILDREN WANT AS MANY DRAGONS AS POSSIBLE IN THEIR GAME, THE PAYROLL DEPARTMENT WANTS AS MANY EMPLOYEES AS POSSIBLE IN THE PROGRAM AND ACCOUNTING WANTS TO PUT IN AS MANY RECORDS AS THEY CAN WITHOUT HAVING TO ACCESS THE DISK. NONE OF THEM COULD CARE LESS WHAT YOUR PROGRAM LOOKS LIKE WHEN IT IS PRINTED OUT.

LET'S BEGIN WITH THE OBVIOUS, BUT STILL IGNORED, CONCEPTS. I WOULDN'T REPEAT THIS FIRST ONE EXCEPT THAT THE PROGRAMS THAT I HAVE RECEIVED THIS WEEK SHOW THAT PEOPLE STILL DON'T LISTEN EXTREMELY WELL. THE FIRST BASIC SKILL OF A COMPUTERIST WHO IS GOING TO READ AND WRITE IN THE BASIC LANGUAGE IS THAT HE LEARN TO READ AND WRITE WITHOUT A LOT OF EXTRANIOUS AND UNNECESSARY SPACES. I STILL RECEIVE PROGRAMS WHICH ARE AS MUCH AS THIRTY PERCENT WASTED SPACE, THAT WASTED SPACE BEING PRIMARILY UNNEEDED SPACES BETWEEN BASIC WORDS. TO LEARN TO WRITE BASIC WITHOUT SPACES TAKES SOMETHING ON THE ORDER OF A DAY OR TWO OF PRACTICE. IT STILL AMAZES ME THAT THERE ARE COMPUTERISTS OUT THERE WHO WRITE COMPLETE PROGRAMS WHO HAVE NOT YET LEARNED THAT SIMPLE SKILL.

BOB RETELLE POINTS OUT IN HIS ARTICLE IN THIS ISSUE THE VALUE OF USING MULTIPLE STATEMENTS PER LINE. I'LL LET BOB COVER THAT IN DETAIL, BUT WILL REMIND YOU AT THIS POINT THAT LINE NUMBERS ARE VERY EXPENSIVE.

NOW THAT I HAVE SOUNDED OFF ABOUT MY FAVORITE TOPICS (AGAIN), WE CAN GET DOWN TO REAL WORK.

THE FIRST MAJOR TOPIC TO COVER IS REMARKS. AS WE HAVE SAID IN PREVIOUS ISSUES, REMARKS ARE HIGHLY VALUABLE. ANY PROGRAMMER WHO WRITES A PROGRAM AND DOES NOT REMARK IT QUALIFIES FOR THE "DAMNED FOOL OF THE WEEK" AWARD. HOWEVER, REMARKS IN A BASIC PROGRAM MUST ALWAYS BE AVAILABLE BUT NEED NOT ALWAYS BE LOADED. AS WE'VE POINTED OUT BEFORE, REMARKS SHOULDN'T BE ADDRESSED. THAT IS, THERE SHOULD NEVER BE A 'GOTO1000' WHERE LINE 1000 IS A REMARK. A GOOD COMPUTERIST, WHETHER HE IS A HOBBIST OR A PROFESSIONAL, WHETHER HE WORKS FOR HIS OWN PLEASURE OR FOR SOMEBODY ELSE'S BUSINESS, SHOULD ALWAYS MAINTAIN A NOTEBOOK WHEREIN HE MAINTAINS LISTINGS OF ALL THE

"THE AARDVARK JOURNAL" IS PUBLISHED SIX TIMES YEARLY BY RODGER OLSEN, AARDVARK TECHNICAL SERVICES, 1690 BOLTON, WALLED LAKE, MI 48088. SUBSCRIPTION RATE: $9.00 PER YEAR. BULK RATE POSTAGE (PERMIT #5) PAID AT WALLED LAKE, MI 48088

PROGRAMS HE WRITES. IT IS NOT NECESSARY, IF YOU DON'T HAVE A PRINTER, TO KEEP AN EXACT LISTING. IT IS HOWEVER IMPORTANT TO HAVE A LIST OF REMARKS FOR EACH PROGRAM AVAILABLE ON HARD COPY AWAY FROM THE PROGRAM ITSELF. OTHERWISE YOU WILL SOON HAVE FORGOTTEN WHAT THE VARIOUS VARIABLES ARE FOR AND WHAT THE SUBROUTINES DO AND WHERE THEY ARE. THE IMPORTANT THING IS THAT YOU PUT THE REMARKS IN IN SUCH A WAY AS THEY CAN BE EASILY REMOVED SO THAT YOU CAN SAVE ALL THAT RUN TIME SPACE IF YOU NEED IT.

THERE ARE SOME IMPORTANT HABITS TO PICK UP THAT CONSISTANTLY SAVE A LOT OF BYTES IN EVERY PROGRAM. THE FIRST GOOD HABIT YOU NEED TO GET INTO IS TO USE AS FEW VARIABLE NAMES AS POSSIBLE. STOP AND THINK FOR A SECOND WHAT IT TAKES FOR BASIC TO USE A NAME. IF YOU USE THE VARIABLE X IN 'FORX=1TO10' IN THE FIRST LINE OF THE PROGRAM AND THEN NEVER USE 'X' AGAIN, BASIC STILL HAS TO STORE THE LAST VALUE THAT IT GAVE X JUST IN CASE YOU SHOULD WANT TO CALL IT UP AGAIN. THAT MEANS THAT EVERY TIME YOU USE A VARIABLE NAME, THAT VARIABLE NAME MUST BE STORED WITH A RELATED VALUE IN A TABLE OF VARIABLES. THEREFORE, YOU DON'T USE MORE NAMES THAN YOU HAVE TO USE. IT'S A GOOD HABIT TO GET INTO TO PICK A COUPLE OF VARIABLES THAT YOU ALWAYS USE FOR TEMPORARY COUNTERS SINCE THERE WILL PROBABLY BE FREQUENT TIMES IN YOUR PROGRAM WHEN YOU WILL WANT TO USE A VARIABLE TO DO NOTHING MORE THAN COUNT A LOOP OR TEMPORARILY STORE A VALUE THAT YOU ARE NOT GOING TO USE AGAIN LATER ON. FOR INSTANCE, IF YOU HAVE A TIMING LOOP 'FORX=1TO100:NEXT' THAT WAS JUST DESIGNED TO SLOW DOWN THE PROGRAM A LITTLE, YOU DON'T REALLY CARE WHAT THE VALUE OF X IS WHEN THE LOOP IS DONE. ALL YOU CARE ABOUT IS THE FACT THAT THERE WAS A DELAY IN THE PROGRAM. IF YOU'RE INPUTING FIVE NAMES IN A LOOP AND YOU SET UP A LOOP 100FORX=1TO5: INPUTA$(X):NEXT YOU AGAIN DON'T CARE WHAT THE VALUE OF X TURNS OUT TO BE. YOU KNEW IT WAS GOING TO BE 5, BUT WHAT YOU WANTED WAS THE FIVE ITEMS YOU INPUT. THEREFORE, YOU SHOULD ALWAYS HAVE A COUPLE OF VARIABLES YOU USE WHENEVER YOU NEED A TEMPORARY COUNTER AND REUSE THEM. THOSE OF YOU WHO HAVE LOOKED AT THE AARDVARK PROGRAMS MAY HAVE NOTICED THAT I TEND TO USE X AND Y AS MY TEMPORARY COUNTERS IN EVERY PROGRAM. BY ESTABLISHING THE HABIT FAIRLY EARLY IN MY PROGRAMMING CAREER, I FIND IT EASY TO READ MY OWN PROGRAMS SINCE I ALWAYS KNOW WHEN I SEE AN X OR A Y THAT IT IS A TEMPORARY VALUE THAT ISN'T GOING TO BE STORED OR KEPT FOR ANYTHING. JUST TO BE DIFFERENT, IN ONE OF THE PROGRAMS WE PUBLISHED THIS MONTH, I USED 'I' AS THE TEMPORARY COUNTER. YOU WILL NOTICE IN THE 'ASTROID BELT' PROGRAM THAT THE VARIABLE 'I' APPEARS OVER AND OVER AGAIN. IT IS USED WHEREVER I DON'T PLAN TO KEEP OR STORE A VALUE. I USED IT FOR A COUNTER AND I USED IT TO PASS VALUES BACK AND FORTH BETWEEN VARIOUS SUBROUTINES. IF I HAD USED A DIFFERENT VARIABLE NAME FOR EACH ONE OF THOSE COUNTERS I WOULD HAVE PROBABLY USED ANOTHER 100 TO 150 BYTES IN THE PROGRAM.

YOU SHOULD ALSO LOOK FOR VARIABLE NAMES IN THE PROGRAM THAT YOU ONLY NEED FOR A LITTLE WHILE. SOMETIMES YOU NEED TO KEEP A VARIABLE VALUE ONLY AS LONG AS YOU ARE RUNNING A SUBROUTINE, BUT WHEN THE SUBROUTINE IS OVER YOU COULDN'T CARE LESS WHAT IT IS. IN THAT CASE YOU CAN USE THE SAME VARIABLE NAMES IN SEVERAL DIFFERENT PARTS OF THE PROGRAM AND STILL NOT RUN INTO ANY CONFLICT. NOW, OF COURSE, IT IS IMPORTANT, IF YOU ARE GOING TO DO THIS, THAT YOU COMMENT YOUR PROGRAM EXTREMELY WELL. IF YOU USE SC AS A TEMPORARY COUNTER TO DO THE SCREEN CLEAR IN LINE 10 AND LATER ON USE IT TO MEAN SCORE, MAKE SURE YOU KEEP SOME REMARKS AROUND SOMEPLACE TO REMIND YOURSELF OF THAT. OTHERWISE YOU'LL END UP ALMIGHTY CONFUSED. ONCE YOU'VE MASTERED THAT GOOD HABIT, WE'VE GOT A FEW MORE FOR YOU.

IF YOU ARE GOING TO USE A FIVE OR SIX DIGIT NUMBER AND YOU ARE GOING TO USE IT MORE THAN ONCE IN A PROGRAM, YOU CAN OFTEN SAVE SPACE BY USING A VARIABLE NAME INSTEAD. THAT'S BECAUSE OF THE WAY BASIC STORES INFORMATION IN A BASIC LINE IN TOKENIZED FORM. WHILE THE BASIC COMMANDS SUCH AS 'THEN', 'PRINT' AND SO ON ARE ENCODED, BOTH THE VARIABLE NAMES AND NUMBERS ARE STORED IN ASCII FORM. THEREFORE, THE NUMBER 52348 TAKES FIVE BYTES IN MEMORY. HOWEVER, THE VARIBLE NAME 'S' IN THE SAME LINE TAKES ONLY ONE BYTE OF MEMORY. NOW IT TAKES ABOUT TWELVE BYTES TO STORE AND INITIALIZE THE TERM 'S'. THEREFORE, YOU WOULDN'T USE IT IF YOU WERE ONLY GOING TO USE THE VALUE ONCE. BUT IF IT IS GOING TO APPEAR SEVERAL TIMES, YOU ARE SAVING FOUR BYTES EVERY TIME YOU USE THE VARIABLE NAME RATHER THAN THE NUMBER AND SPEEDING UP THE PROGRAM TO BOOT. THEREFORE, IF YOU ARE GOING TO USE A LOT OF BIG NUMBERS, MAKE THEM VARIABLE NAMES INSTEAD. IT ALSO HAS THE SIDE ADVANTAGE OF SOMETIMES ALLOWING YOU TO PUT MORE INFORMATION ON A LINE, SINCE YOU ARE USING FEWER SPACES ON THAT LINE, AND THEREFORE, POSSIBLY CUTTING DOWN ON THE NUMBER OF LINES NEEDED TO HOLD THE PROGRAM, SAVING YOU THAT FAMOUS FIVE BYTES PER LINE THAT YOU SAVE BY COMBINING LINES.

THAT SOMEHOW, AND FOR NO GOOD REASON, LEADS ME TO THINK OF TWO ITEMS OF STYLE THAT SEEM TO AFFLICT MOST COMPUTERISTS. IT SEEMS THAT WHEN PEOPLE BEGIN TO WRITE COMPUTER PROGRAMS, THEY ARE SO ENTHRALLED WITH THE IDEA THAT SOMEONE WILL ACTUALLY READ WHAT THEY HAVE WRITTEN ON THE SCREEN THAT THEY WRITE THEMSEVES DARN NEAR TO DEATH. FOR INSTANCE, THEY WILL OFTEN USE THE PHRASE

'INPUT"DO YOU WISH INSTRUCTIONS (YES OR NO)" FOR AN INPUT WHEN THE PHRASE 'INPUT"INSTRUCTIONS?"' WOULD PROBABLY GET THE IDEA ACROSS JUST AS WELL. PARTICULARLY WHEN YOU ARE WORKING WITH A SMALL SYSTEM AND LIMITED MEMORY, YOU TALK LIKE THE TRADITIONAL NEW ENGLANDER - USE AS FEW WORDS AS POSSIBLE AS EVERY SINGLE CHARACTER USED IN A PRINT STATEMENT IS ONE MORE FULL BYTE OF MEMORY LOST IRRETRIEVABLY.

THE SECOND ITEM OF VERY BAD STYLE WHICH SEEMS TO AFFLICT BOTH NEW AND OLD PROGRAMMERS IS THE CONTINUAL CHECKING BASED ON THE ASSUMPTION THAT THE USER OF THE PROGRAM IS A TOTAL IDIOT WHO DOES NOT SPEAK ENGLISH OR ANY OTHER LANGUAGE PARTICULARLY WELL. A NEW PROGRAMMER, AND EVEN SOME EXPERIENCED ONES, WILL OFTEN DO SOMETHING LIKE ASK FOR A "YES" OR "NO" ANSWER TO A QUESTION AND THEN CHECK FOR A "YES" AND HAVE ANOTHER LINE CHECK FOR A 'NO" AND HAVE YET ANOTHER LINE THAT SAYS IF YOU DIDN'T GET A 'YES' AND DIDN'T GET A "NO", GO BACK AND ASK THE QUESTION AGAIN. I SUPPOSE THEY'RE TRYING TO COVER THE POSSIBILITY THAT THE USER MAY HAVE PUT IN A PHRASE SUCH AS "WAIT A MINUTE, I'M STILL THINKING ABOUT THAT". HOWEVER, SUCH ANSWERS TO SIMPLE QUESTIONS AREN'T VERY LIKELY. IT WOULD SEEM, THEREFORE RATIONAL, THAT IN MOST CASES YOU WOULD ASSUME THE MOST COMMON ANSWER, CHECK FOR THE LEAST LIKELY ANSWER AND THEN BRANCH. I'M NOT SAYING THAT YOU SHOULDN'T GIVE A USER A CHANCE TO VERIFY A RESPONSE, PARTICULARLY WHERE THE RESPONSES ARE PERHAPS COMPLEX OR THERE IS A LARGE CHANCE OF ERROR, HOWEVER, THE AMOUNT OF CHECKING THAT GOES ON FOR SIMPLE QUESTIONS SUCH AS 'DO YOU WANT INSTRUCTIONS' IS OFTEN RIDICULOUS.

THAT PRETTY MUCH COVERS THE SIMPLE TRICKS INVOLVED IN SAVING MEMORY. THERE ARE OTHER TRICKS, SOME OF WHICH WE HAVE COVERED ELSEWHERE IN THE JOURNALS AND WILL NOT REPEAT HERE. FOR INSTANCE, JOURNAL #1 CONTAINED AN ARTICLE ON STRINGS THAT SHOWED WAYS OF SAVING SPACE BY STORING INTEGER DATA IN STRING ARRAYS RATHER THAN SUBSCRIPTED ARRAYS. THERE IS ALSO AN ARTICLE ELSEWHERE IN THIS JOURNAL ON HOW TO USE SUBSCRIPTED ARRAYS TO SAVE PROGRAMMING STEPS. HOWEVER, THERE IS NOT A LOT MORE THAT CAN BE DONE WITH SIMPLE TRICKS. AT THIS POINT WE HAVE TO TALK ABOUT A LITTLE MORE DIFFICULT CONCEPT OF THINKING ABOUT AND ORGANIZING A PROGRAM.

I GUESS I SHOULD ALSO POINT OUT THAT WE ARE NOW TALKING ABOUT CONSIDERABLE WORK IN WRITING A PROGRAM. ONE OF THE MAJOR REASONS FOR PROGRAMS RUNNING MUCH TOO LONG IS THAT AUTHORS ,WHETHER AMATEUR OR PROFESSIONAL OFTEN ARE UNWILLING TO PUT ANY PROFESSIONAL DEGREE OF EFFORT IN ON THEIR PROGRAMS. VERY FEW PROGRAMS ARE GOING TO COME OUT POLISHED AND FINISHED BY THE END OF THE FIRST WRITING. TO DO AN EFFICIENT PROGRAM, AND TO KEEP MEMORY USAGE DOWN, YOU ARE GOING TO HAVE TO BE WILLING TO WRITE AND TO REWRITE SECTIONS OF THE PROGRAM OR ENTIRE PROGRAMS TWO OR THREE TIMES. I MIGHT SUGGEST, WHILE WE DON'T OFTEN PUSH OUR PRODUCTS IN THE JOURNAL, THAT YOU SHOULD HAVE A TEXT EDITOR OF SOME SORT TO DO THE FINAL WORK ON THE PROGRAM. OTHERWISE THE AMOUNT OF EFFORT REQUIRED IN REWRITING AND COMBINING LINES MIGHT DISCOURAGE YOU FROM ACTUALLY PUTTING THE POLISH ON A PROGRAM AND FITTING IT INTO AN APPROPRIATE MEMORY SIZE. ONCE YOU'VE FINISHED YOUR PROGRAM AND YOU HAVE IT RUNNING, TAKE A GOOD CLOSE LOOK AT IT. LOOK FOR PLACES WHERE YOU CAN USE LOOPS TO DO REPETIVE WORK WHERE YOU ARE NOW USING DISCRETE POKE STATEMENTS OR DISCRETE INPUT STATEMENTS. LOOK CAREFULLY FOR PLACES WHERE YOU CAN USE 'ON----GOTO' RATHER THAN 'IF---THEN' TYPE STATEMENTS. IF YOU HAVE TO MAKE ONE OF A NUMBER OF DIFFERENT JUMPS OR CALL ONE OF A NUMBER OF DIFFERENT SUBROUTINES DEPENDING ON WHAT HAPPENED TO THE VARIABLE, USE THE 'ON----GOTO' STATEMENT. IT TAKES A LOT OF MEMORY TO HAVE A SEPARATE LINE FOR EACH POSSIBILITY. WITH THE MEMORY SAVINGS INHERENT IN MAKING ANY LARGE NUMBER OF BRANCHES, IT IS OFTEN POSSIBLE TO CONDITION THE VARIABLE SO THAT IT FALLS INTO THE RIGHT RANGE. FOR INSTANCE, IF YOU ARE GOING TO GO TO ONE SUBROUTINE IF THE SCORE IS 100 AND ANOTHER IF IT IS 200 AND A THIRD IF IT IS 300, YOU CAN MAKE A STATEMENT LIKE "ONX/100GOTO1,2,3". IT'S CHEAPER IN MEMORY SIZE AND TIME TO DIVIDE THE VARIABLE OR SUBTRACT FROM IT OR WHATEVER YOU HAVE TO DO TO GET IT DOWN TO A SIMPLE SEQUENCE RATHER THAN HAVE A LOT OF "IF" STATEMENTS IN MANY, MANY CASES.

WHEN YOU READ THE PROGRAM, THERE ARE A COUPLE OTHER THINGS YOU CAN LOOK FOR, SPECIFICALLY, WHICH COULD GIVE YOU HINTS TO PLACES YOU COULD SAVE BYTES. FOR INSTANCE, MOST COMPUTERISTS ARE AWARE THAT IF YOU HAVE MORE THAN ONE IDENTICAL LINE IN A PROGRAM, IT IS BEST TO SET IT UP AS A SUBROUTINE, PUT IT IN ONCE AND ACCESS IT WHENEVER YOU NEED IT. THERE IS ALSO AT LEAST ONE STRUCTURE LEFT OVER FROM THE EARLY DAYS OF BASIC WHICH SEEMS TO AFFLICT A LOT OF PROGRAMMERS. THAT IS THE CONCEPT OF SKIPPING A LINE WITH AN "IF" STATEMENT. FOR EXAMPLE, ON THE OLD BASICS, IF STATEMENTS WERE OFTEN LIMITED TO "IF(SOMETHING)THENGOTO(A LINE NUMBER), AND IN THOSE BASICS YOU HAD TO BRANCH AROUND A CONDITION IN ORDER TO AVOID IT WITH AN IF STATEMENT. FOR INSTANCE, IF I WANTED TO MULTIPLY A VARIABLE BY 1.4 ONLY IF IT WAS OVER 40, I WOULD HAVE WRITTEN IN THE OLD DAYS:

```
100 IFA<41THEN120
110 A=A*1.5
120 (THE NEXT LINE)
```

OUR BASIC ALLOWS US, HOWEVER, TO PUT LONG STATEMENTS AFTER AN IF CLAUSE AND HAVE THEM EXECUTED. THEREFORE, IN OSI BASIC, THE LINE SHOULD BE WRITTEN

```
100 IFA>40THENA=A*1.5
```

BY PUTTING THE PHRASE ON THE END OF THE IF STATEMENT, WE SAVE A LINE NUMBER AND A JUMP. SINCE THE OLD CONSTRUCTION WAS VERY COMMON FOR A LONG TIME AND STILL APPEARS IN PLACES LIKE CREATIVE COMPUTING PROGRAM LISTINGS, A LOT OF NEW PROGRAMMERS ARE STILL DOING IT. A GOOD HINT IS THAT WHENEVER YOU SEE AN IF STATEMENT THAT DOES NOTHING ELSE THAN SKIP A LINE, YOU ARE PROBABLY WASTING SOME BYTES.

YOU SHOULD ALSO READ CAREFULLY THE SUGGESTIONS GIVEN LAST MONTH ON SAVING TIME WITH "IF" STATEMENTS AND THE OTHER SUGGESTIONS GIVEN IN THE ARTICLE ON SPEED. MOST OF THE SUGGESTIONS GIVEN TO SPEED UP PROGRAMS WORK AS WELL TO CUT DOWN ON MEMORY SIZE. A FAST PROGRAM AND A MEMORY EFFICIENT ONE NORMALLY USE THE SAME TECHNIQUES.

ONE FINAL WORD. THERE IS NO SUBSTITUTE FOR ORIGINAL THOUGHT. I RECENTLY RECEIVED A RATHER CUTE LITTLE ROCKET BATTLE PROGRAM THAT DREW A FAIRLY COMPLEX PLAYING FIELD ACROSS THE BOTTOM OF THE SCREEN. IT DREW THE LANDSCAPE WITH A SERIES OF DATA STATEMENTS LIKE THIS.

```
100 FOR X=1TOMANY:READY:READPLACE:POKEPL,Y:NEXT
100DATA53247,161,53440,267,53660,67 ETC.
```

THE ETC. WENT ON FOR ABOUT 60 NUMBERS.

THE PROBLEM WAS THAT THE PROGRAM WOULD NOT FIT IN 4K. THE DATA STATEMENTS WERE USING 10 BYTES FOR EACH POINT BY STORING A 5 DIGIT PLACE, A THREE DIGIT POKE VALUE, AND A COUPLE OF COMMAS.

I THOUGHT ABOUT THE PROBLEM AND CAME UP WITH A NEW FORMAT. I STORED ALL OF THE INCREMENTS TO MOVE IN THE 8 COMMON DIRECTIONS IN AN ARRAY. I THEN INPUT A STARTING POSITION FOR THE DRAWING AND USED 4 BYTE CODES TO CODE THE MOVES AND POKES. IT WORKED THIS WAY. ASSUMING THAT YOU WANTED TO MOVE IN DIRECTION 1 (LEFT), AND WANTED TO POKE A 161 IN THE SPOT YOU REACHED, YOU MADE THE 1 THE FIRST NUMBER IN THE DATA ENTRY AND THE 161 THE NEXT THREE BYTES. TO DRAW A PICTURE, YOU STARTED AT THE STARTING POINT, READ A DATA BYTE, MOVED IN THE DIRECTION THE FIRST DIGIT POINTED AND THEN POKED THE REMAINDER IN THE SPOT YOU GOT TO. I DIDN'T LOSE ANYTHING BY SETTING UP THE MOVEMENT ARRAY, AS I NEEDED IT TO MOVE CHARACTERS AROUND THE SCREEN LATER AND ENDED UP SAVING A COUPLE OF HUNDRED BYTES - LIKE THIS -

```
100 FORX=1TO8:READM(X):NEXT REM FREE. I HAD TO DO IT ANYWAY
110DATA1,-63,-64,-65,-1,65,65,63  REM THE 8 TREK DIRECTIONS FOR C2/4
190ST=ANY PLACE ON THE SCREEN.
200FORX=1TOMANY:READD:DI=INT(D/1000):ST=ST+M(DI)
210PO=D-(DI*1000):POKEST,PO:NEXT
220DATA1161,2000,2160,ETC. REM ONE 4 BYTE ENTRY PER POKE REPLACES A 5 BYTE AND A 3 BYTE ENTRY.
```

PROGRAM WOULD MOVE IN DIRECTION 1, POKE A 161, MOVE IN DIRECTION 2 AND POKE A 0, MOVE IN DIRECTION 2 AGAIN AND POKE A 160.

```
230REM ST=STARTING PLACE, DI=DIRECTION, PO=VALUE TO BE POKED.
```

THAT IS NOT AN EXACT COPY OUT OF THE PROGRAM, BUT I THINK IT WILL RUN AS PRINTED HERE AND IT SHOWS ONE WAY I SAVED A LOT OF MEMORY BY THINKING ABOUT HOW TO STORE INFORMATION. AN EXPANDED VERSION OF THAT ROUTINE WILL PROBABLY ALLOW US TO PUBLISH A LANDER TYPE GAME SOON WHICH WILL HAVE SEVERAL TERRAINS WHICH WILL CHANGE AS YOU GET CLOSER, AND WHICH WILL STILL FIT IN 8K.

THIS, THEN, IS THE FINAL RULE. AFTER YOU MASTER THOSE SIMPLE GOOD HABITS FOR EFFICIENT PROGRAMMING, THINK ABOUT WHAT THE PROGRAM HAS TO DO AND RETHINK IT, AND RETHINK IT. THIS IS, AFTER ALL, A HOBBY AND A PROFESSION THAT REWARDS THOUGHT.

** OSI DEMONSTRATES NEW SOUND I.Q. BOARD **

AT A RECENT COMPUTER SHOW, OSI DEMONSTRATED A NEW VOICE I.Q. BOARD MATED TO A NEW EXTENDED ELIZA PROGRAM. THE SIMULATION OF INTELLIGENCE WAS UNCANNY. THE COMPUTER WOULD ASK SEVERAL QUESTIONS (VERBALLY), ACCEPT SPOKEN ANSWERS, AND ADJUST THE SUBSEQUENT QUESTIONS TO REFLECT THE INTELLIGENCE OF THE USER. THE ESTIMATED I.Q., THOUGH INACURRATE, WAS DISPLAYED ON A SCREEN OUT OF THE USERS VIEW.

WE SAW THREE DEMONSTRATIONS. AN OSI ENGINEER WAS ASKED TO FACE THE MACHINE. IN ABOUT TWO MINUTES IT FLASHED "I.Q. 155" AND DREW HIM INTO TALKING ABOUT THE EFFECTS OF REAGAN'S CANDIDACY ON THE ARMS RACE. THE SECOND SUBJECT WAS AN APPLE PROGRAMMER. THE SYSTEM FLASHED "I.Q. 105" AND BEGAN TO ASK QUESTIONS ABOUT SPORTS. (HE MISSED THEM.)

THE LAST SUBJECT WAS AN TRS-80 DEALER FROM LESSER PLAINS GEORGIA. EVENTUALLY THE SYSTEM FLASHED "I.Q. 76" AND BROKE OUT WITH A LOUD "BREAKER, BREAKER, GOOD BUDDY. WHAT'S YOUR 10-40. COME BACK WONTCHA, COME BACK THERE."

BEGINNERS' CORNER

SUBSCRIPTED VARIABLES

THE THREE MOST IMPORTANT AND SIMPLEST CONCEPTS WHICH A PROGRAMMER MUST MASTER ARE THE USE OF NAMED VARIABLES, THE USE OF LOOPS, AND THE USE OF SUBSCRIPTED VARIABLES (ARRAYS). JUDGING FROM THE PROGRAMS WHICH ARE SUBMITTED FOR PUBLICATION, THE THIRD ITEM IN THAT LIST SEEMS TO GIVE NEW COMPUTERISTS THE MOST TROUBLE.

IN ITS SIMPLEST FORM, A SUBSCRIPTED VARIABLE IS USUALLY DESCRIBED AS A SERIES OF NUMBERED BOXES. EACH BOX REPRESENTS A DISCRETE VARIABLE. HOWEVER, THE BOXES ARE ALL NUMBERED SO THE COMPUTER CAN MOVE FROM ONE BOX TO THE NEXT UNDER PROGRAM CONTROL. SINCE THERE MAY BE SEVERAL SETS OF THESE BOXES, EACH SET OF BOXES (ARRAY) IS GIVEN A NAME TO IDENTIFY THE SET. THE ARRAYS CAN BE EITHER NUMERICAL VARIABLES OR STRING INFORMATION AND YOU MAY THEREFORE FIND EITHER A(X) OR A$(X) IN A PROGRAM. THE VARIABLE NAME DEFINES WHICH ARRAY YOU ARE USING, AND THE NUMBER IN THE PARANTHESIS DEFINES WHICH VARIABLE OUT OF THAT ARRAY YOU ARE GOING TO USE AT THIS TIME. FOR INSTANCE, A(5) REFERS TO THE FIFTH VARIABLE NAMED 'A'. A$(3) MEANS THE THIRD STRING IN THE SET OF BOXES CALLED 'A$'. BY THE WAY THAT IS USUALLY PRONOUNCED 'A STRING SUB THREE'.

WHAT MAKES A SUBSCRIPTED VARIABLE SPECIAL IS THAT THE COMPUTER CAN WALK THROUGH THE ARRAY OR JUMP THROUGH IT BY FIGURING LOGICALLY WHICH VARIABLE IT SHOULD BE WORKING WITH AT ANY ONE TIME. FOR INSTANCE, A COMPUTER COULD NOT TELL THAT THE VARIABLE 'TOTAL' SHOULD FOLLOW THE VARIABLE 'PAY'. IT CAN, HOWEVER, DO AN ARITHMETIC OPERATION AND CAN THEREFORE FIGURE THAT A(3) FOLLOWS A(2), AND THAT A(5) FOLLOWS A(4). THAT ALLOWS US TO USE THE SAME PROGRAM LINES TO MANIPULATE A LOT OF DIFFERENT VARIABLES.

BEFORE WE EXPLAIN HOW TO USE A SUBSCRIPTED VARIABLE, PERHAPS WE SHOULD STOP FOR ONE QUICK AND SIMPLE DEMONSTRATION OF WHAT MAKES THEM UNIQUE. PICTURE YOURSELF WRITING A PROGRAM TO FIGURE THE GROSS PAY FOR TEN EMPLOYEES OF A SMALL COMPANY. LETS ASSUME THAT EACH EMPLOYEE MAKES $6.50 AN HOUR. YOU COULD ASSIGN EACH EMPLOYEE'S HOURS A DIFFERENT VARIABLE NAME. YOU COULD FOR INSTANCE CALL THEM A0, A1, A2, ...A9. YOU WOULD THEN HAVE TO WRITE TEN DIFFERENT PROGRAM LINES TO FIGURE THE TEN DIFFERENT PAYS. YOU WOULD HAVE TO WRITE-

```
100PAY=A0*6.5:?PAY
110 PAY=A1*6.5:?PAY
```

AND SO ON FOR TEN DIFFERENT EMPLOYEES.

IT WOULD BE NICE IF THE COMPUTER COULD SIMPLY STEP THROUGH ALL TEN EMPLOYEES AND PICK THE NEXT ONE AS IT NEEDED IT. TO DO THAT WE WOULD STILL GIVE EACH EMPLOYEE'S HOURS A DIFFERENT VARIABLE NAME BUT WE WOULD MAKE IT A SUBSCRIPTED VARIABLE. THE FIRST EMPLOYEE WOULD BE A(0), THE SECOND A(1) AND SO ON TO A(9). WE CAN THEN USE A SIMPLE LOOP TO STEP THROUGH ALL TEN EMPLOYEES.

```
100FORX=0TO9:PAY=A(X)*6.5:?PAY:NEXT
```

NOW WE ONLY NEED THAT ONE PROGRAM LINE BECAUSE THE COMPUTER WILL INCREMENT X EACH TIME THROUGH THE LOOP AND THEREFORE EACH TIME PICK THE NEXT EMPLOYEE IN LINE.

BEFORE WE GO ON TO DISCUSS THE OTHER USES OF THE VARIABLE, LETS LOOK AT WHAT YOU NEED TO DO TO USE AN ARRAY. THE FIRST THING YOU HAVE TO DO IS TO SET UP SPACE IN BASIC TO HOLD THE ARRAY. SINCE AN ARRAY TAKES A LOT OF SPACE, BASIC NORMALLY NEEDS TO PLAN FOR THE SIZE OF THE ARRAY BEFORE IT OPENS UP THE MEMORY. THAT'S THE REASON FOR THE DIMENSION STATEMENTS AT THE BEGINNING OF PROGRAMS. THE DIM STATEMENT TELLS THE COMPUTER HOW MUCH SPACE TO SET ASIDE TO EVENTUALLY HOLD ELEMENTS IN AN ARRAY. NOW YOU DON'T NEED TO DIMENSION ALL OF THE ARRAYS. IF BASIC SEES A SUBSCRIPTED VARIABLE THAT HAS NOT BEEN DIMENSIONED, IT WILL AUTOMATICALLY ASSUME THAT YOU HAVE DIMENSIONED IT FOR 10. ANYTHING HIGHER MUST BE IN A DIM STATEMENT. IF YOU TRY TO USE A VARIABLE WITH A HIGHER NUMBER THAN THE ARRAY IS DIMENSIONED FOR, YOU WILL GET A FUNCTION CALL

ERROR. THE REASON WHY THE DIMENSION STATEMENTS ARE NORMALLY AT THE BEGINNING OF A PROGRAM IS THAT AN ARRAY MAY NOT BE DIMENSIONED TWICE. IF IT HASN'T HAD A DIM STATEMENT, THE FIRST TIME BASIC SEES A DIMENSIONED VARIABLE IT WILL ESTABLISH IT AS DIMENSION 10. IF YOU THEN PUT IN A DIM STATEMENT FOR THAT VARIABLE, YOU WILL GET A 'DD' OR DOUBLE DIMENSION ERROR. THEREFORE, WE NORMALLY PUT THE DIMENSION STATEMENTS AT THE BEGINNING OF A PROGRAM.

YOU WILL OFTEN READ OR HEAR THE PHRASE "ESTABLISH AN ARRAY". THE MEANING OF THAT TERM IS FAIRLY FLEXIBLE AND PRETTY MUCH JUST COMES DOWN TO WE DECIDE TO USE AN ARRAY TO HOLD SOMETHING. WE MIGHT IMPLEMENT THAT BY ADDING A DIMENSION STATEMENT OR, IF IT IS GOING TO BE A VERY SMALL ARRAY, SIMPLY USING A SUBSCRIPTED VARIABLE IN A PROGRAM. BUT "ESTABLISHING AN ARRAY" IS SIMPLY ANOTHER WAY TO SAY THAT WE HAVE DECIDED TO USE AN ARRAY TO HOLD A CERTAIN CLASS OF VARIABLES.

USING AN ARRAY IS FAIRLY SIMPLE BECAUSE BASIC GIVES YOU A LOT OF TOOLS TO DO IT WITH. YOU SIMPLY GIVE THE NAME OF THE VARIABLE ARRAY AND THEN PUT THE NUMBER OF THE ELEMENT YOU WANT IN PARENTHESIS. HOWEVER, YOU CAN SPECIFY THE NUMBER IN PARENTHESIS BY PUTTING IN AN ACTUAL NUMBER, OR BY PUTTING IN A VARIABLE SUCH AS 'X' OR BY PUTTING IN AN EXPRESSION WHICH BASIC WILL EVALUATE, SUCH AS A(X+Y). FOR INSTANCE, IF X = 1 AND Y = 4, THEN A(X) IS THE FIRST ELEMENT IN THE ARRAY, A(Y) CALLS THE FOURTH ELEMENT IN THE ARRAY, AND A(X+Y) WOULD GET YOU THE FIFTH ELEMENT IN THE ARRAY. THE FACT THAT BASIC CAN CALCULATE WHICH VARIABLE YOU WANT IS REALLY THE SECRET TO THE POWER OF THE SUBSCRIPTED ARRAY.

ALRIGHT, NOW WHAT CAN WE DO WITH IT? WE'VE ALREADY SEEN IN THE FIRST SIMPLE EXAMPLE THAT WE CAN USE AN ARRAY TO GET BASIC TO DO THE SAME THING TO A LOT OF DIFFERENT VARIABLES. IT IS ALSO VERY POWERFUL WHEN WE HAVE TO RELATE TWO COLUMNS OF FIGURES OR TWO GROUPS OF VARIABLES IN A CERTAIN WAY. FOR INSTANCE, ASSUME THAT WE ARE TRYING TO FIGURE THE TOTAL PAY FOR EACH OF TEN EMPLOYEES AND THIS TIME EACH OF THE EMPLOYEES HAS A DIFFERENT HOURLY RATE. WE WANT THE COMPUTER TO TAKE THE FIRST MAN'S HOURS AND MULTIPLY IT BY THE FIRST MAN'S HOURLY RATE AND THEN TAKE THE SECOND MAN'S HOURS AND MULTIPLY IT BY THE SECOND MAN'S HOURLY RATE, AND SO ON FOR TEN EMPLOYEES. RATHER THAN WRITE TEN DIFFERENT LINES OF PROGRAMMING, WE CAN USE TWO ARRAYS. IN THE FIRST ARRAY, WE'LL PUT THE EMPLOYEE'S HOURS NUMBERED 0 THROUGH 9, SO THAT A(0) IS THE HOURS THE FIRST MAN WORKED. THEN IN THE SECOND ARRAY WE'LL PUT THEIR PAY RATES SUCH THAT P(0) IS THE PAY RATE FOR THE FIRST MAN, P(1) IS THE RATE FOR THE SECOND MAN, AND SO ON FOR THE TEN EMPLOYEES. ONE LINE OF PROGRAMMING THEN WILL MULTIPLY THE APPROPRIATE FIGURES FROM EACH COLUMN TOGETHER.

```
100 FORX=0TO9:PAY=A(X)*P(X):?PAY:NEXT
```

OF COURSE, IN REAL LIFE, WE'D PROBABLY HAVE A NUMBER OF DIFFERENT PROGRAM LINES HERE FIGURING STUFF LIKE SOCIAL SECURITY, TAXES, BENEFITS AND SO ON FOR EACH EMPLOYEE. BUT THE IMPORTANT THING IS THAT WE WOULD WRITE THAT PROGRAM SEGEMENT ONLY ONCE, AND THEN BY CHANGING THE VALUE OF X, BE ABLE TO APPLY IT TO ALL TEN EMPLOYEES WITHOUT ANY OTHER CHANGES.

THAT'S THE SECOND WAY THAT ARRAYS ARE HANDY. WHEN WE HAVE A LOT OF THINGS IN COLUMN A THAT WE WANT TO RELATE TO COLUMN B, FOR INSTANCE IF YOU WANT TO MULTIPLY A WHOLE BUNCH OF STOCKS BY THEIR YEILD, OR A BUNCH OF LOANS BY THEIR INTEREST RATE, WE WOULD PUT ONE VARIABLE IN ONE ARRAY AND THE SECOND VARIABLE IN ANOTHER ARRAY AND THEN STEP THROUGH AND MULTIPLY.

THE NEXT THING THAT ARRAYS ARE HANDY FOR IS WHEN YOU WANT BASIC TO PICK OUT A VARIABLE FOR YOU. FOR INSTANCE, ASSUME THAT YOU ARE DESIGNING A PIN BALL GAME AND THAT YOU HAVE SIX BUMPERS IN THE PIN BALL GAME. WHEN YOU HIT A BUMBPER YOU MAKE B EQUAL TO THE NUMBER OF THAT BUMPER. ASSUME FURTHER THAT EACH BUMPER GIVES A DIFFERENT SCORE. MOST NEW COMPUTERISTS ADD THE SCORE UP BY GOING THROUGH A BUNCH OF 'IF' STATEMENTS TO PICK OUT THE RIGHT INCREMENT. IT LOOKS LIKE THIS:

```
100 IFB=1THENSC=SC+1
110 IFB=2THENSC=SC+2
120 IFB=3THENSC=SC+3
```

AND SO ON FOR THE NUMBER OF BUMPERS. IT WOULD BE EASIER AND FASTER IF BASIC COULD JUST PICK OUT WHICH SCORE TO PUT WITH WHICH BUMBPER. FOR THAT YOU USE A SUBSCRIPTED ARRAY. IN THIS CASE WE WOULD SET UP AN ARRAY OF SIX ELEMENTS LONG AND EACH ELEMENT WOULD BE THE SCORE ADDED BY THAT BUMBPER TO THE TOTAL SCORE. FOR INSTANCE, S(2) WOULD BE THE POINTS THAT BUMPER TWO WAS WORTH. THEN ONE LINE WOULD INCREMENT THE SCORE.

```
100 SC=SC+S(B)
```

BY SIMPLY LOOKING AT WHAT B WAS THE COMPUTER COULD PICK OUT THE VALUE TO BE ADDED TO THE SCORE RATHER THAN DOING A WHOLE BUNCH OF IF'S TO SORT THE WHOLE THING OUT.

THAT THEN IS THE THIRD USE OF ARRAYS. WHENEVER YOU WANT BASIC TO PICK ONE VARIABLE OUT OF A

BATCH SUCH AS TO PICK WHICH VARIABLE WILL BE INCREMENTED OR WHICH VALUE WILL BE ADDED TO A VARIABLE, YOU USE A SUBSCRIPTED ARRAY SO THAT BASIC CAN CALCULATE ITS WAY THROUGH.

YOUR BASIC IS MORE POWERFUL THAN OSI TOLD YOU IN THE MANUAL. YOU HAVE THE ABILITY TO USE WHAT THEY CALL 'MULTIDIMENSIONED ARRAYS'. THE MOST COMMON MULTIDIMENSIONED ARRAY IS A TWO DIMENSIONAL ARRAY. A TWO DIMENSIONAL ARRAY IS NORMALLY PICTURED AS BEING SET UP SORT OF LIKE A CHECKER BOARD WHERE YOU NUMBER ALL THE ROWS AND THEN NUMBER ALL THE COLUMNS. YOU THEN SELECT WHICH BOX TO USE BY TELLING THE COMPUTER WHICH ROW DOWN AND WHICH COLUMN OVER YOU WANT. THAT MEANS THAT YOU HAVE TWO NUMBERS OR TWO EXPRESSIONS IN PARENTHESIS SEPARATED BY A COMMA. FOR INSTANCE, S(3,2) MEANS THAT YOU WANT THE THIRD ROW DOWN, SECOND COLUMN OVER 'S'. THERE ARE A LOT OF USES FOR TWO OR THREE DIMENSIONAL ARRAYS, BUT LET ME GIVE YOU A SIMPLE EXAMPLE TO SHOW HOW THEY WORK. LET'S ASSUME THAT I HAVE A MAILING LIST OF 30 ADDRESSES STORED IN MY PROGRAM. EACH ADDRESS HAS FIVE LINES: NAME, STREET NUMBER, CITY, STATE, ZIP CODE AND COUNTRY. I COULD SET UP AN ARRAY THAT WAS DIMENSIONED FOR (20,5). THAT WOULD GIVE ME AN ARRAY WITH TWENTY ELEMENTS IN IT AND EACH ELEMENT WOULD HAVE FIVE SUBELEMENTS IN IT. THEREFORE, IF I WANTED THE STREET NUMBER ON THE TENTH ADDRESS, I COULD ASK THE COMPUTER TO PRINT S$(10,2) AND GET THE SECOND ELEMENT IN THE TENTH ROW DOWN. NOW I HAVE A WHOLE BUNCH OF THINGS I CAN GET TO WITH THE SAME SET OF NUMBERS. ONE OF THE SIMPLIST USES FOR IT WOULD BE TO PRINT A MAILING LIST. I COULD PRINT A LIST OF ALL TWENTY NAMES AND ALL TWENTY ADDRESSES WITH ONE PROGRAM LINE.

```
100 FORX=1TO20:FORY=1TO5:?S$(X,Y):NEXTY,X
```

THAT WOULD STEP THROUGH ALL TWENTY ADDRESSES, IN EACH ADDRESS STEP THROUGH ALL FIVE LINES AND THEN GO ON TO THE NEXT ONE. IN FACT, WE USE A SIMILAR TECHNIQUE IN OUR MAILING LIST PROGRAM.

YOU MAY HAVE NOTICED THAT THE SAME LINES COULD BE USED TO INPUT ALL THE ADDRESSES WITH ONLY ONE OR TWO LINES. LIST FOR THE C1P.

I AM SURE THAT IF YOU WATCH THE PROGRAMS THAT YOU TYPE IN OR USE, YOU WILL SEE A LOT OF OTHER USES FOR SUBSCRIPTED VARIABLES. THERE ARE A NUMBER OF THINGS YOU NEED TO REMEMBER WHEN YOU USE THEM.

THE FIRST THING TO REMEMBER IS THAT SEVERAL VARIABLES CAN BE DIMENSIONED WITH THE SAME STATEMENT.

```
100DIMA(12),A$(34),D(19) ETC.
```

YOU SHOULD ALSO REMEMBER THAT A SUBSCRIPTED VARIABLE CAN HAVE THE SAME NAME AS A REGULAR VARIABLE AND BASIC WILL KNOW THE DIFFERENCE. A(X) IS NOT THE SAME VARIABLE AS 'A'.

ALSO KEEP IN MIND THAT WHEN YOU DIMENSION A VARIABLE YOU HAVE ONE MORE SLOT IN THAT VARIABLE ARRAY THAN THE NUMBER YOU DIMENSIONED FOR. MOST PEOPLE TEND TO FORGET THAT 0 IS THE FIRST NUMBER AS FAR AS A COMPUTER IS CONCERNED. THEREFORE, IF YOU DIMENSION 'A' FOR 10, YOU ACTUALLY HAVE ELEVEN SLOTS - A(0) THROUGH A(10). REMEMBERING THAT WILL SAVE YOU MEMORY.

THE NEXT THING TO REMEMBER IS THAT IT TAKES ABOUT SIX BYTES FOR EVERY ELEMENT IN AN ARRAY WHETHER IT IS USED OR NOT. IF YOU NEVER HAVE AN A(7) BUT YOU DIMENSIONED FOR IT, A(7) WILL TAKE SIX BYTES EVEN IF IT SITS THERE EMPTY. THEREFORE, DON'T DIMENSION FOR AN ARRAY BIGGER THAN YOU'RE ACTUALLY GOING TO USE.

ALSO REMEMBER THAT YOU CAN USE EITHER NUMERICAL VARIABLES OR STRINGS IN ARRAYS, BUT THE STRINGS DON'T WORK EXACTLY RIGHT IN MOST OSI BASICS. IN ALL OSI BASICS, THE FRE(X) ROUTINE WILL STOP WORKING AS SOON AS YOU HAVE A DIMENSIONED STRING VARIABLE. THE BASIC SIMPLY CAN'T FIGURE IT OUT AND GETS LOST. IN ROM BASIC, YOU HAVE A BUG IN THE BASIC THAT WILL SOMETIMES FREEZE UP THE MACHINE IF YOU SUBSCRIPT A VARIABLE FOR MORE THAN TWENTY OR SO AND DO STRING MANIPULATIONS.

IF YOU ARE GOING TO INITIALIZE AN ARRAY, THE USUSAL WAY TO DO IT IS TO USE DATA STATEMENTS. FOR INSTANCE, IF WE ARE GOING TO READ TEN DIFFERENT VALUES INTO VARIABLE A(X) IT IS USUALLY DONE THIS WAY:

```
100 FOR X=0TO9:READA(X):NEXT
110DATA0,1,2,3,4,5,6,7,8,9
```

THAT'S USUALLY THE MOST EFFICIENT WAY TO GET DATA INTO AN ARRAY.

YOU MAY OCCASSIONALLY BECOME CONFUSED BECAUSE PROGRAMMERS SOMETIMES USE THE TERM 'SUBSCRIPTED ARRAY', 'SUBSCRIPTED VARIABLE ' AND 'ARRAY' INTERCHANGABLY. DON'T LET IT CONFUSE YOU - THOSE TERMS ARE ACTUALLY INTERCHANGEABLE. TECHNICALLY, A SUBSCRIPTED VARIABLE IS THE VARIABLE NAME AND THE PAIR OF PARENTHESIS THAT APPEARS IN THE TEXT. THE ARRAY IS THE SET OF

BOXES THAT THE VARIABLES ARE STORED IN. HOWEVER, IN CONVERSATION, MOST COMPUTERISTS AREN'T THAT ACCURATE AND SIMPLY THROW OUT WHICHEVER TERM COMES TO MIND FIRST.

*** NEW FROM AARDVARK ***

AS USUAL, LOTS OF NEW GAMES AND STUFF. BUT THE BIG NEWS IS THAT THE STUFF WE PROMISED LAST MONTH IS NOW IN AND IS CHEAPER THAN WE PLANNED.

WE HAVE NOT ONE, BUT THREE NEW ROM MONITORS.

THE C1S IS FOR THE C1P ONLY. IT HAS ALL THE FEATURES THAT WE ADVERTISED LAST MONTH AND A FEW MORE BESIDES. IT HAS FULL EDIT FEATURES, SOFTWARE SELECTABLE SCROLL WINDOWS, A SOFTWARE SELECT FOR OSI OR TYPEWRITER KEYBOARD, BELL SUPPORT, A SOFTWARE SELECT FOR THE OLD MONITOR ROUTINES, AND A COUPLE OF OTHER FEATURES. WE ARE SELLING IT FOR $39.95 AS AN INTRODUCTORY OFFER.

THE C1E IS AVAILABLE FOR BOTH THE C1P AND THE C2/4/8 ROM MACHINES. IT HAS ALL THE STUFF WE JUST LISTED FOR THE C1S AND ADDS AN EXTENDED MACHINE CODE MONITOR FOR THE MACHINE CODE ENTHUSIASTS. IT HAS BREAKPOINT INSTALLATION, MACHINE CODE PROGRAM SAVE AND LOAD, MEMORY MOVE, AND HEX DUMP UTILITIES IN IT. IT ALSO HAS THE DISK BOOTSTRAP, BUT CANNOT PROVIDE EDIT ROUTINES WHILE RUNNING 65D BASIC.

WE DROPPED THE INTRO PRICE ON THAT ALSO - TO $59.95.

THE PC BOARD HOUSE HAS PROMISED TO DELIVER THE MEMORY BOARDS BEFORE THIS ISSUE GOES OUT. WE HAGGLED A LOT AND WERE ABLE TO GET A BETTER PRICE THAN I PLANNED.
THE NEW PRICE FOR THE 8K RAM BOARD WITH TWO PARALLEL PORTS IS $29.95 FOR THE BARE BOARD.

HATE TO DEAL WITH A FACELESS COMPANY? HERES AN ALMOST HONEST AND MOSTLY ACCURATE DESCIPTION OF AARDVARK.

THE COMPANY IS ABOUT THREE YEARS OLD, A VIRTUAL ANCIENT IN THIS FIELD. IT STARTED AS A HOBBY AND BECAME A FULL TIME BUSINESS A LITTLE OVER A YEAR AGO. (ACCORDING TO MY WIFE, TWO FULL TIME BUSINESSES, ACCORDING TO MY COMPETITORS, I'M JUST WASTING MY TIME.)

EVER NOTICE THOSE FUNNY NOISES IN THE BACKGROUND WHEN YOU CALL? EVER WONDER WHY WE ANSWER THE PHONE AT ODD HOURS? WHEN IT CAME TIME TO OPEN THE FULL TIME BUSINESS, I DIDN'T LIKE THE IDEA OF A LONG DRIVE TO WORK, SO WE ADDED THE OFFICE AND SHOP ONTO THE HOUSE. THAT WAY WE GOT THE ROOM FOR THE PROFESSIONAL SIZED OFFICE COMPLETE WITH SEVEN COMPUTERS, PRINTERS, EQUIPMENT, SHIPPING OFFICE AND THE SHORTEST WALK TO WORK POSSIBLE. SAVES A LOT ON GAS.

THERE ARE FIVE PEOPLE WORKING HERE NOW, BUT THERE ARE ONLY THREE THAT YOU ARE LIABLE TO GET ON THE PHONE.

I DO THE PROGRAMMING, ANSWER QUESTIONS (YES, AARDVARK ANSWERS QUESTIONS ON THE PHONE!), WRITE MOST OF THE JOURNAL AND TRY TO LOOK BUSY. IF YOU WANT ME, CALL AFTER NOON. NORMALLY I STUMBLE IN ABOUT NOON, ANSWER QUESTIONS AND HANDLE BUSINESS UNTIL ABOUT 6:00. AFTER A BREAK, I COME BACK AND PROGRAM FROM ABOUT 10:30 TO 4:00 A.M. I'M USUALLY CHEERFULL IF YOU CALL ANYTIME EXCEPT DURING THE PROGRAMMING HOURS. I USE THE NIGHT HOURS SPECIFICALLY TO PROGRAM WITHOUT PHONE INTERRUPTIONS.

JANE (MY SPOUSE) AND JUDY OFFICIALLY WORK FROM 9:00 AM TO 3:00 PM. THEY HANDLE ALL THE SHIPPING AND BOOKWORK AND CAN TELL YOU A LOT MORE ABOUT WHERE YOUR ORDER IS THAN I CAN. JANE IS RAPIDLY TURNING INTO A BETTER THAN AVERAGE PROGRAMMER AND CAN HANDLE A LOT OF QUESTIONS. JUDY AND I ARE COMPLETE OPPOSITES. SHE KNOWS WHERE EVERYTHING IS. ON DAYS THAT SHE IS OFF, I HAVE BEEN KNOWN TO LOSE THE C8DF. IF YOU TALK TO EITHER ONE ON THE PHONE - BE NICE - THEY WERE HIRED FOR THEIR LOOKS.

THE REST OF THE EMPLOYEES FILE, PACK ORDERS, COLLATE BOOKS, DO THE BOOKS, AND DO NOT ANSWER THE PHONE.

MEMORY SAVING TECHNIQUES

BY BOB RETELLE

IN THE LAST ISSUE, RODGER SHOWED YOU HOW TO MAKE YOUR BASIC PROGRAMS RUN AS FAST AS POSSIBLE. THIS ARTICLE WILL ATTEMPT TO SHOW YOU HOW TO MAKE THE BEST USE OF THE MEMORY IN YOUR SYSTEM. ANYONE WITH 32K OR 48K MAY TURN THE PAGE. THE BASIC OSI SYSTEM WITH 8K OF RAM IS QUITE USEABLE

FOR MOST PROGRAMMING. THERE WILL BE TIMES HOWEVER, WHEN THE PROJECT YOU'RE WORKING ON JUST WON'T FIT. THAT'S WHEN YOU SHOULD TRY SOME OF THE TECHNIQUES PRESENTED HERE.

IF YOU'VE BEEN READING THE AARDVARK JOURNAL AND AARDVARK DATA SHEETS, YOU SHOULD ALREADY BE FAMILIAR WITH A FEW OF THE MORE BASIC TECHNIQUES FOR SAVING MEMORY. THE FIRST AND SIMPLIST IS TO ELIMINATE ALL SPACES FROM YOUR PROGRAMS. THE SLIGHT INCREASE IN READABILITY IS MORE THAN OFFSET BY THE MEMORY THEY TAKE UP. REMEMBER THAT EACH SPACE BURNS UP ONE BYTE OF MEMORY. NOT ONLY THAT, BUT EXTRA SPACES WILL FORCE YOU TO USE A LOT OF EXTRA PROGRAM LINES AND EACH EXTRA LINE COSTS YOU FIVE BYTES OVERHEAD JUST FOR THE LINE NUMBER, NEXT LINE POINTER, AND END OF LINE MARKER. THUS, WE COME TO THE MOST IMPORTANT TECHNIQUE. USE AS FEW LINES AS POSSIBLE. AS RODGER HAS SAID, "REMEMBER THE COLON!!" BY PUTTING AS MANY STATEMENTS AS POSSIBLE ON EACH LINE, YOU SAVE THOSE FIVE BYTES OF OVERHEAD. WHILE FIVE BYTES DOESN'T SOUND LIKE MUCH, IT ADDS UP QUICKLY. IT SHOULD GO WITHOUT SAYING THAT REMS SHOULD BE AVOIDED LIKE THE PLAGUE! EVERY BYTE USED BY A REM IS WASTED AS FAR AS THE EXECUTION OF YOUR PROGRAM IS CONCERNED. IF YOU FEEL YOU NEED TO DOCUMENT YOUR PROGRAM, EITHER MAKE A LOT OF NOTES AS YOU GO ALONG OR BUY A BIGGER MEMORY. BASIC "STYLE" AND "STRUCTURED" PROGRAMMING ARE NICE FOR MACHINES WITH HUGE MEMORIES; BUT ARE JUST NOT COMPATIBLE WITH AN 8K MICROS. ONE SIDE BENEFIT OF USING THE ABOVE PROGRAMMING TECHNIQUES IS THAT YOUR PROGRAM WILL RUN SLIGHTLY FASTER!

NOW WE'LL GET TO SOME OF THE MORE ADVANCED WAYS OF SQUEEZING YOUR PROGRAM. FIRST, LET'S THINK ABOUT LINE NUMBERS A MINUTE. BASIC STORES EACH PROGRAM LINE'S NUMBER AS A TWO BYTE HEXADECIMAL NUMBER. THAT MEANS THAT IT MAKES NO DIFFERENCE WHETHER A LINE'S NUMBER IS 0 OR 64123, IT STILL TAKES UP TWO BYTES. IN THAT RESPECT, THE LINE NUMBER DOESN'T AFFECT MEMORY USAGE. HOWEVER, IN THE TEXT OF THE PROGRAM, BASIC STORES ALL NUMBERS IT SEES AS ASCII CODES, SO A LINE CONTAINING 'GOTO10560' WILL TAKE THREE MORE BYTES THAN 'GOTO10'. AGAIN,

THREE BYTES DOESN'T SEEM SIGNIFICANT, BUT IT ADDS UP TOO. FOR AN EXAMPLE OF THIS, GET OUT YOUR 'TIME TREK' DOCUMENTATION. IF YOU DON'T HAVE A 'TIME TREK', GO GET ONE. WE'LL WAIT..... OK? NOW IN 'TIME TREK', I WAS FACED WITH TAKING AN 8K PROGRAM AND ADDING FULL GRAPHICS SCREENS AND READOUTS AND ADDING REAL TIME ACTION AND STILL FITTING IT INTO 8K. IT TURNED OUT THAT THERE WERE SEVERAL SUBROUTINES WHICH WERE CALLED REPEATEDLY THROUGHOUT THE PROGRAM. NOW A 'STRUCTURED' PROGRAM WOULD HAVE ALL THE SUBROUTINES GROUPED NEAR THE END OF THE PROGRAM. (ACTUALLY THAT IS THE WAY I USUALLY TRY TO PROGRAM BECAUSE IT DOES HELP TO FIND A LOST SUBROUTINE.)

IN THIS CASE THOUGH, PUTTING THESE FEW RECURRENT SUBROUTINES AT THE END WOULD HAVE CALLED FOR LINE NUMBERS AROUND 8000, OR FOUR BYTES EACH. WHAT I ENDED UP DOING WAS TO RENUMBER THEM WITH SINGLE DIGIT NUMBERS (2-9) AND HAVE THE FIRST PROGRAM LINE JUMP AROUND THEM. THE SCREEN CLEAR, PSEUDO-PRINT AT, TIME DELAY AND OTHERS ALL HAVE A ONE BYTE LINE NUMBER. WITHOUT GOING THROUGH THE WHOLE PROGRAM AND COUNTING EACH INSTANCE, I CAN SAY THAT THE SAVINGS OF MEMORY WAS SIGNIFICANT. IF YOUR SUBROUTINES ARE TOO LONG TO FIT ON ONE OR TWO LINES, YOU CAN STILL TAKE ADVANTAGE OF THIS TRICK. JUST HAVE THE MAIN PROGRAM 'GOTO' OR 'GOSUB', SAY, LINE 2. LINE 2 WILL THEN SAY '2 GOTO(YOUR SUBROUTINE). WHILE THIS WON'T SAVE AS MUCH AS ACTUALLY HAVING YOUR SUBROUTINE AT LINE 2, THERE WILL BE SOME SAVINGS. ANOTHER EXAMPLE IS THE ADVENTURES I WROTE FOR AARDVARK. THE MAIN ENTRY POINT INTO THE 'COMMAND?' LOOP WAS AROUND LINE 150 (3 BYTES). THIS LINE WAS JUMPED TO FROM SO MANY LOCATIONS THAT I CHANGED IT TO LINE 9 (1 BYTE). THE SAVINGS IN THIS CASE WERE EVEN MORE SIGNIFICANT. THIS TECHNIQUE WILL BE MOST EFFECTIVE OF COURSE WHEN YOU HAVE ONE OR MORE LINES WHICH ARE CALLED SEVERAL TIMES. YOU CAN STILL REALIZE AN OVERALL SAVINGS EVEN IF THERE ARE NO SPECIFICALLY REPEATED SUBROUTINES OR LINES BY RENUMBERING THE WHOLE PROGRAM TO KEEP THE LINE NUMBERS LESS THAN 1000. IN MOST CASES, THIS WILL MEAN LOSING THAT PRETTY 'STEPS OF 10' LOOK, BUT ONCE AGAIN, JUST THINK OF THE SAVINGS. USUALLY NUMBERING IN STEPS OF 5 WILL DO THE TRICK. AGAIN, EACH NEW LINE NUMBER WILL SAVE ONE BYTE EACH TIME IT IS REFERENCED, BUT THE TOTAL COULD EASILY BE OVER 100 BYTES!

ANOTHER WAY TO PICK UP SOME BYTES IS BY LOOKING HARD AT THE VARIABLES YOU USE. IT'S GREAT TO USE DOUBLE LETTER VARIABLES LIKE SC FOR SCORE AND AL FOR ALIEN LOCATION WHEN YOU'VE GOT PLENTY OF MEMORY. IT MAKES IT EASY TO READ AND ALL THAT. WHEN YOU'RE SQUEEZING FOR BYTES THOUGH, SOMETHING'S GOT TO GO. I USUALLY TRY TO KEEP THE VARIABLE NAMES AS CLOSE A POSSIBLE TO THEIR FUNCTIONS, EVEN TO THE POINT OF USING DOUBLE LETTER VARIABLES WHEN I'M STARTING A NEW PROGRAM. WHEN IT STARTS GETTING TIGHT THOUGH, I LIST OUT THE VARIABLES ON PAPER USING EITHER MY

NOTES OR THE VARIABLE TABLE MAKER AND REASSIGN A SINGLE LETTER TO EACH OF THEM. THEN BY RUNNING THE SEARCH PROGRAM, I FIND AND CHANGE EACH TO ITS NEW SINGLE LETTER VARIABLE NAME. OF COURSE, IF THERE ARE MORE THAN 26 VARIABLES, SOME OF THEM WILL HAVE TO HAVE MORE THAN ONE LETTER. IN THAT CASE, WHAT I HAVE TO DO IS TO ASSIGN THE SINGLE LETTERS TO THOSE VARIABLES WHICH OCCUR MOST OFTEN. IT SOUNDS LIKE A LOT OF WORK AND IT IS, BUT IF YOU CONSIDER THAT A LONG PROGRAM WILL HAVE SEVERAL HUNDRED REFERENCES TO VARIABLES, THIS ONE IDEA CAN SAVE YOU A COUPLE OF HUNDRED BYTES. IN A REVISION OF ONE PROGRAM, I MANAGED TO SAVE OVER 95 BYTES BY CHANGING ONE VARIABLE ALONE! OF COURSE, IT SAVES A LOT OF TIME AND WORK IF YOU WRITE THE PROGRAM WITH SINGLE LETTER VARIABLES IN THE FIRST PLACE.

ONE LAST THING TO BE AWARE OF IS THE WAY SOME UTILITY PROGRAMS MAY TREAT YOUR PROGRAM. FOR INSTANCE, SOME EDITORS MAY LEAVE EXTRA BLANKS AT THE END OF A PROGRAM LINE AFTER EDITING IT. RENUMBERERS GENERALLY PUT A SHORTER LINE NUMBER IN THE PROGRAM TEXT BY POKING A BLANK IN AT THE END OF THE NEW NUMBER. THESE CAN BE VERY HARD TO FIND, ESPECIALLY THE BLANKS AT THE ENDS. THE SOLUTION IS ALWAYS TO RUN THE PACKER PROGRAM AS THE LAST STEP IN SQUEEZING A PROGRAM. IT WILL TAKE OUT ANY STRAY BLANKS REMAINING HERE AND THERE. IT'S A REAL TIME SAVER, AS ARE ALL OF THE UTILITIES MENTIONED ABOVE. I SUPPOSE YOU COULD DO IT ALL BY HAND, BUT YOUR PROGRAM WOULD PROBABLY BE OBSOLETE BY THE TIME YOU FINISHED IT.

IT MAY NOT SEEM WORTH WHILE TO GO THROUGH ALL THESE PROGRAMMING GYMNNASTICS JUST TO SAVE 1 OR 2 BYTES AT A TIME, BUT TAKEN TOGETHER, 100 HERE AND 200 THERE, YOU MAY END UP WITH 1/2K MORE SPACE THAN YOU STARTED OUT WITH, WHICH JUST MIGHT MEAN THE DIFFERENCE BETWEEN A TIGHT FITTING, BUT RUNNING, PROGRAM AND ONE WHICH JUST WON'T FIT.

THERE ARE A COUPLE MORE TECHNIQUES FOR SAVING MEMORY WHICH ARE JUST TOO COMPLEX TO GO INTO NOW, INVOLVING STORAGE OF NUMERIC VARIABLES IN DISCRETE MEMORY LOCATIONS INSTEAD OF IN SUBSCRIPTED ARRAYS AND METHODS OF RETRIEVING THEM. I'M HOPING TO BE ABLE TO USE THIS METHOD TO SQUEEZE C4 'TIME TREK' ENOUGH TO ADD COLOR AND SOUND TO THE 8K VERSION. THE USE OF STRINGS TO REPRESENT NUMERIC DATA IS ANOTHER MEMORY SAVING TECHNIQUE WHICH WOULD BE THE SUBJECT OF ANOTHER ENTIRE ARTICLE.

HOPEFULLY, THE METHODS OUTLINED HERE WILL GIVE YOU A START IN PUTTING YOUR OVER-LENGTH PROGRAMS ON A BYTE-FREE DIET.

ONE LAST ITEM.....

FOUND! 207 BYTES!

WHEN I STARTED TO LEARN MACHINE LANGUAGE, I SOON DISCOVERED THAT I COULDN'T PUT MY USR ROUTINES IN GOOD OLD PAGE 2 AT THE SAME TIME AS THE C1 CURSOR! PUTTING THEM AT THE TOP OF MEMORY WORKED OF COURSE, BUT THAT TOOK UP RAM FROM MY BASIC PROGRAM. THEN, WHILE PERUSING MY COPY OF AARDVARK'S 'FIRST BOOK OF OSI' (BY WILLIAMS & DORNER), I FOUND THE ANSWER. DOWN ON PAGE 1 OF YOUR COMPUTER'S MEMORY ARE TWO LOCATIONS WHICH ARE SUPPOSED TO CONTAIN THE NMI (NON-MASKABLE-INTERRUPT) AND IRQ (INTERRUPT REQUEST) SERVICING ROUTINES. ASSUMING YOU'RE NOT USING THE NMI FOR A REAL-TIME CLOCK OR THE LIKE, THERE ARE 207 BYTES IN ONE BLOCK FROM $0130 TO $01FF (THAT'S DECIMAL 304 TO 511). THE USR(X) VECTOR SHOULD BE SET UP WITH POKE11,48 AND POKE12,1. YOU CAN NOW PUT A USR ROUTINE ON PAGE 1 AND USE THE C1 CURSOR TO EDIT THE BASIC PROGRAM, OR HAVE TWO USR ROUTINES OR ONE VERY LONG USR ROUTINE OR ?????

*** MOVING THE DIRECTORY OFF TRACK 12 ***

ONE OF THE MOST DAMNABLE ANNOYING THINGS THAT THE OSI PROGRAMMERS DID WAS TO PUT THE DIRECTORY TRACK RIGHT IN THE MIDDLE OF THE 5" DISK. INSTEAD OF GETTING 39 TRACKS OF FILE SPACE, YOU GET 10 ON ONE SIDE OF THE DIRECTORY AND 26 ON THE OTHER.

FORTUNATELY, CHUCK SCOTT IS AROUND. HE'S THE FELLOW WHO WROTE THE SINGLE DISK COPIER AND THE DISK CATALOGER. HE STOPPED IN THE OFFICE LAST WEEK TO PICK UP A COPY OF THE SOURCE CODE FOR OS65D AND CALLED BACK THE NEXT DAY WITH THIS SET OF INSTRUCTIONS. WITH THESE, YOU CAN MOVE THE DIRECTORY AND THE OVERLAYS THAT LAY ON TRACK 12 TO ANY OTHER TRACK. WE ARE GOING TO USE TRACK 39 AS AN EXAMPLE.

THE FIRST THING TO DO IS TO SET UP A DISK WITH A DIRECTORY, CREATE AND DELETE UTILITIES ON IT SET UP TO CALL TRACK 39 (OR WHATEVER TRACK YOU ARE GOING TO USE) INSTEAD OF TRACK 12. (I.E. WHEREVER IT SAYS DISK!"CA 12,....." CHANGE IT TO DISK!" CA 39,....."). DO THE SAME FOR THE SAVE FUNCTIONS.

THAT IS SO THAT YOU WON'T DO UP AN ENTIRE DISK AND THEN FIND OUT THAT YOU HAVE NO WAY TO USE IT.

BOOT UP AND EXECUTE THE FOLLOWING COMMANDS - 5" DISK ONLY.

```
EXIT
EM (CALL EXTENDED MONITOR)
!CA 4A00=01,1
   4DC4 /12 39 REM TYPE  4DC4 CR. 12 WILL APPEAR, TYPE THE 39
!SA 01,1=4A00/8
!CA 4A00=05,1
   513A /12  CHANGE TO 39
!"CA 4200=06,1
  42B8 / 12  CHANGE TO 39
!SA 06,1=4200/1   NOTE-ONE SECTOR ONLY.
!CA 4200=12,1  REM MOVE DIRECTORY INFORMATION TO NEW TRACK
!SA 39,1=4200/1
!CA 4200=12,2
!SA 39,2=4200/1
!CA 4200=12,3  REM MOVE OVERLAYS
!SA 39,3=4200/I
!CA 4200=12,4
!SA 39,4=4200/1
EXIT
BASIC
```

IF YOU ARE USING THE AARDVARK BEXEC*, YOU NEED ONLY CHANGE THE 12s TO 39s IN LINES 190, 200, 370, 380, 490, 500, 560, 570, 610, 620, AND 630 TO HAVE A READY TO GO SYSTEM. IF YOU HAVE A SUPERDISK TEXT EDITOR, THAT WILL TAKE ABOUT 10 MINUTES. THE REST OF YOU JUST TYPE AWAY.

=== SPEAKING OF TYPING ===

I HATE TYPING IN "BEXEC*" CONTINOUSLY, IT IS SIX SCATTERED AND UNCOMFORTABLE KEYSTROKES. SINCE I HAVE THE DIRECTORY, CHANGE, DELETE, AND ZERO ON THE BEXEC* TRACK, I TYPE IT ALL TOO OFTEN.

I PREFER A NAME LIKE "E" . ONE EASY KEYSTROKE. SO I CHANGED OVER THE 5" DISKS. THIS IS HOW IT WORKS. YOU CAN ENTER ANY NAME UP TO 6 CHARACTERS AND IT WILL BE "BEXEC*" FROM THEN ON.

REMEMBER TO CREATE NEW FILE AND STORE YOUR CURRENT "BEXEC*" UNDER THE NEW NAME BEFORE YOU RUN THIS. OTHERWISE THE SYSTEM WILL GET CONFUSED AND NOT BE ABLE TO BOOT THE DISK.

```
100 INPUT"NEW NAME";B$
110 IFLEN(B$)<6THENB$=B$+" ":GOTO110
120 DISK!"CA 5000=01,1
130 FORX=1TO6:POKE21544+X,ASC(MID$(B$,X)):NEXT
140 DISK!"SA 01,1=5000/8":RUNB$ REM LAST PHRASE (RUNB$) OPTIONAL.
```

LETTERS TO THE EDITOR

BILLY SMITH, RICHMOND, KY

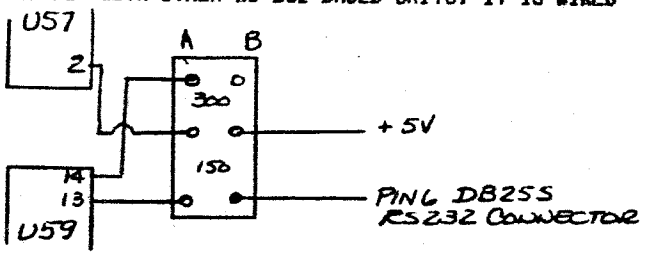
HERE'S A METHOD OF CONNECTING THE C1P WITH RS232 INTERFACE TO A KSR-37 TELETYPE.
PARTS REQUIRED: DPDT SWITCH AND HOOK UP WIRE.

1) LOCATE AND CUT TRACE GOING TO U57, PIN 2. SOLDER WIRE FROM U57, PIN 2 TO A SIDE OF CENTER POLE. SOLDER WIRE FROM U59, PIN 14 TO 'A' SIDE OF UPPER POLE. SOLDER WIRE FROM U59, PIN 13 TO 'A' SIDE OF LOWER POLE. (ALLOWS SELECTION OF EITHER 300 OR 150 BAUD. PLACEMENT OF SWITCH CAN BE AT OWNER'S DISCRESSION, BUT SHORT LEADS ARE DESIRABLE.

2) SOLDER WIRE FROM PIN 6 OF DB25S CONNECTOR TO 'B' SIDE LOWER POLE. SOLDER WIRE FROM 'B' SIDE CENTER POLE TO +5V TERMINAL OF POWER SUPPLY.

3) INSURE PATCH CABLE USED CONNECTS RS-232 OUT OF C1-P TO RS-232 IN ON TELETYPE.

SUMMARY: THE KSR-37 TELETYPE OPERATES AT 110 TO 150 BAUD. THE RS-232 INTERFACE ACCEPTS A +5V TO GROUND LOGIC LEVELS. PIN 6 MUST BE HELD AT +5V LEVEL TO INDICATE "DATA SET READY". SINCE 5V ON PIN 6 MIGHT INTERFER WITH OTHER RS-232 BASED UNITS, IT IS WIRED THROUGH ONLY IN THE 150 BAUD POSITION



FROM W.H. WITTE
FORT WAYNE IN. 46815

THE "GREAT NEW SCREEN CLEAR" IN THE AUGUST ISSUE IS REALLY GREAT BUT IT WORKS ON SOME MACHINE OTHER THAN MY C4P. (PRESUMABLY C1P?).
TO MAKE THE IDEA WORK ON THE C4P, YOU NEED TO CHANGE THE SO THAT S$ DOESN'T GET TOO LONG BEFORE THE SCREEN IS CLEARED.

THIS WORKS ON THE C4P WITH ROM BASIC:

```
100A=PEEK(129):B=PEEK(130):POKE129,192:POKE130,215:S$=" ":FORS=1TO62
110S$=S$+" ":NEXT:POKE129,A:POKE130,B:RETURN
```

THE SAME IDEA WORKS GREAT TO CLEAR THE COLOR SCREEN THIS WAY:

```
100A=PEEK(129):B=PEEK(130):POKE129,255:POKE130,231
110INPUT"COLOR";R$:S$=R$
120FORS=1TO62:S$=S$+R$:NEXT
```

THE COLOR NUMBERS FROM 1 - 16 ARE INPUT WITH THE LETTERS FOUR BITS WIDE AND THE LAST FOUR BITS OF THE ASCII CODE FOR THESE LETTERS PROVIDE THE RIGHT COMBINATIONS. LETTER

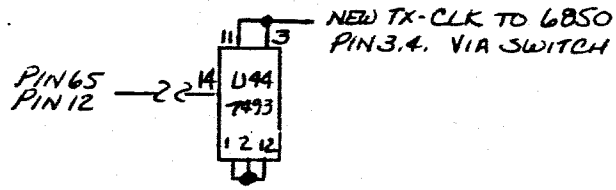
(YEP , WE BLEW THAT ONE THREE WAYS - AND GOT A LOT OF LETTERS ABOUT IT. THIS WAS THE MOST SUCCINCT AND CLEAR EXAMPLE. I WAS IN SUCH A HURRY TO ADD THAT AT THE LAST MINUTE THAT I DIDN'T THINK ABOUT THE FACT THAT THE C2/4/8 SCREENS WOULD TAKE A DIFFERINT STRING COMBINATION FOR THE 2K SCREEN.
SECOND OMMISSION WAS THE LACK OF RESEARCH ON ON WHERE THE STRING POINTERS WERE ON DISK SYSTEMS. IT TURNS OUT THAT THEY ARE ONE BYTE LOWER SO 65D USERS WILL HAVE TO PEEK AND POKE LOCATIONS 128 AND 129 RATHER THAN 129 AND 130.
THIRD OMMISION WAS NOT PAYING ATTENTION TO OUR OWN FIRST BOOK OF OSI. THE TECHNIQUE IS IN THERE. I READ IT AND MISSED IT - AND HEARD FROM JIM WILLIAMS ABOUT THE OMMISION. SORRY JIM.)

D. VALENTINE, NEW YORK, NY

.....A PARALLEL PORT PROBLEM ON A 500 CPU BOARD RUN IN C2-8P'S. WE HAVE BEEN DRIVING A LINE PRINTER ON A C3 WITH THE 510 BOARD PIA "B" SIDE FOR SEVERAL YEARS USING A MACHINE CODE ROUTINE. RECENTLY WE UNSUCCESSFULLY ATTEMPTED TO RUN THAT LINE PRINTER OFF THE C2 PIA. ON THE 500 BOARDS (WE HAVE A SPARE) THE PORT WILL OUTPUT ONLY WHEN INITIALIZED WITH A BASIC ROUTINE. THE MACHINE CODE ONE WILL NOT WORK. OBVIOUSLY, BECAUSE IT WORKS IN BASIC, THE WIRING IS CORRECT. IT IS NOT A PIA CHIP PROBLEM (ANY OF THE 6421'S WE HAVE WILL WORK ON THE 510 BOARD, BUT NOT THE 500'S) ANY HELP OR SUGGESTIONS WOULD BE APPRECIATED.

GEORGE MILLER, ROYAL OAK, MI

TO RUN THE 6850 CHIP AT 110 BAUD ON A C1, DO THE FOLLOWING:
1) U65, PIN 12 "HS" TO A 7493 CHIP (I USED LOCATION U44)
2) RUN 7493 OUTPUT THROUGH A SWITHC AND USE IT AS TX CLK
THIS WILL RUN THE 6850 AT ABOUT 107 WHICH WORKS JUST FINE FOR A TTY.



RICHARD MORRIS, ROCKPORT, IN

IN #2 AARDVARK JOURNAL, THERE WAS DISCUSSION ABOUT RECORD SIZE IN RANDOM ACCESS FILES. THE LIST OF POKES TO 12024 AND 12076 WERE GIVEN FOR BOTH 5" AND 8" DISKS. SOME OF THE NUMBERS GIVEN WILL CAUSE THE SYSTEM TO NOT WORK PERFECTLY FOR RECORDS PAST THE FIRST TRACK OF A DATA FILE. ON OUR C4P-MF 5" SYSTEM, LOCATIONS 12042 AND 12076 BOTH CONTAIN 35 AFTER A PROGRAM IS LOADED INTO THE WORKSPACE. WHEN A RANDOM LILE IS OPENED, THOSE LOCATIONS CONTAIN 16 AND 7 RESPECTIVELY. IF THE LOCATIONS ARE POKED WITH SOMETHING ESLE BEFORE A FILE IS OPENED, 16 AND 7 WILL APPEAR ANYWAY WHEN A RANDOM FILE IS OPENED. THESE NUMBERS ALLOW 128 BYTE RECORDS. 12076 AND 12042 NUMBERS MUST BE POKED WITH THE NUMBERS AFTER OPENING THE FILE TO CHANGE THE RECORD LENGTH.

BYTES PER RECORD	POKE 12976	POKE 12042
16	4	128
32	5	64
64	6	32
128	7	16
256	8	8
512	9	4
1024	10	2
2048	11	1

THE POKE TO 12076 SETS THE RECORD LENGTH AS SHOWN ON THE CHART. THE POKE TO 12042 SEEMS TO HELP THE OPERATING SYSTEM FIND THE LOCATION OF THE RECORD. USING THE VALUES SHOWN IN THE ARTICLE CAUSES SOME INTERESTING THINGS TO HAPPEN. IT IS ONLY ABLE TO STORE RECORDS ON THE FIRST TRACK AND THE FIRST RECORD ON THE SECOND TRACK. IT THEN SOMEHOW BLOWS THE PROGRAM.

THE PROPER NUMBERS TO POKE INTO 12042 ARE LOCATED IN THE CHART. IT MAY NOT BE A COINCIDENCE THAT THE POKE 12042 NUMBER FOR ANY GIVEN RECORD LENGHT IS THE SAME AS THE NUMBER OF RECORDS PER TRACK. IF A SMALLER NUMBER IS USED, IT WILL STORE ONLY THAT NUMBER OF RECORDS IN EACH TRACK. FOR INSTANCE, BY POKING 4 INTO 12076 AND 100 IN 12042 (SHOULD BE 128) 100 16 BYTE RECORDS ARE PRINTED ON EACH TRACK. RECORDS 0 TO 99 ON THE FIRST TRACK, 100 TO 199 ON THE SECOND TRACK AND SO ON. THE CHART CAN BE EXTENDED BUT IT CAUSES A MINOR PROBLEM.

BYTES PER RECORD	POKE 12042	POKE 12076	%TRACK USED
2	255	1	24%
4	255	2	49%
8	255	3	97%

THE LARGEST NUMBER THAT CAN BE POKED INTO A LOCATION IS 255. QUITE A BIT OF UNUSED SPACE IS LEFT ON EACH TRACK WITH 2, 4, AND 8 BYTES PER RECORD SINCE ONLY 255 RECORDS CAN BE STORED ON EACH TRACK. THE CHART SHOWS THE PERCENTAGE OF EACH TRACK THE MACHINE COULD USE. USING THE ABOVE POKES, WE CAN NOW VARY THE RECORD SIZE FROM 2 BYTES TO 2K BYTES. WE DID NOT FIND A WAY TO HAVE THE SYSTEM PROVIDE A RECORD LENGTH OF MORE THAN ONE TRACK.

** WANTED **

A USED, CHEAP, AND WORKING C1P OR SUPERBOAD, AND A C2P. WE NEED SOME SPARES FOR TESTING MODIFICATIONS. CONTACT AARDVARK.

****

NOTE:: THE AARDVARK JOURNAL WILL CARRY ADDS FOR USED OR NEW EQUIPMENT FOR THE PALTRY SUM OF $2.00 PER 85 CHARACTER LINE.

```
ASTEROID RACE
CONTROLS - LEFT SHIFT IS SHIP DOWN
ESC SHIP UP
RIGHT SHIFT SHIP FORWARD
REPEAT IS USELESS CANNON
YOU MUST GET TO THE RIGHT SIDE OF THE SC
REEN. YOU HAVE 5 SHIPS.


 100 GOSUB680:FORX=1TO10:PRINT:NEXT:PRIN
T"ASTEROID RACE"
 105 FORX=1TO200:NEXT:REM COPYRIGHT 1980
 BY AARDVARK - BY RODGER OLSEN
 110 REM SCREEN CLEAR
 120 GOSUB680:M=0
 130 REM C2/4/8 VARIABLE VALUES: C=UPPER
 CORNER: E=LOWER CORNER:
 140 REM L=LINE LENGTH: SH=SHIP DISPLAY
CHARACTER:K=KEYBOARD
 150 REM L2=2 LINE MOVE VALUE.:F=FLAG FO
R MISSLE
 160 C=53376:E=55104:P=55040:F=1
 170 L=64:SH=237:K=57100:L2=62:TB=L2
 180 IFPEEK(57088)<129THEN200
 190 L=32:TB=22:E=54082:P=54051:V=1:C=C+
4
 200 INPUT"DIFFICULTY (1-5)";D:IFD>5THEN
D=5
 210 REM T=TIME DELAY :D=DIFFICULTY LEVE
L
 220 T=150*(5-D):D=D*1.5:GOSUB680
 230 REM FULL SCREEN WITH ASTEROIDS:PUT
SHIP ON SCREEN (S=SHIP POSITION
 240 FORX=1TO75:POKEC+2000*RND(8),226:NE
XT
 250 S=C+1+INT(20*RND(8)+4)*L:IFPEEK(S)<
>32THEN250
 260 REM BLINK THE SHIP FOR IDENTIFICATI
ON.
 270 FORX=1TO40:POKES,183+X:POKES,237:NE
XT:M=0
 280 REM POKE IN NEW ASTEROIDS: NUMBER D
ETERMINED BY DIFFICULTY LEVEL.
 290 FORX=1TOD:POKEP+(L-3)*RND(8)+2,226:
NEXT
 300 REM PEEK KEYBOARD. INVERT VALUE IF
SYSTEM IS C1P
 305 REM IF SHIP STOPS BETWEEN MOVES 305
M=0
 310 FORQ=1TO2:POKES,237:I=PEEK(K):IFVTH
ENI=255-I
 320 IFI=5THENM=L+1
 330 IFI=129THENIFFTHENF=0:N=S+1:POKEN,9
1
 340 IFI=33THENM=-(2*L)+1
 350 IFI=3THENM=1
 360 REM ANY MOVE GETS A SCORE.
 370 IFMTHENSC=SC+1
 380 REM MOVE SHIP IF NOT OUT OF BOUNDS.
 390 POKES,32:IFS+M>CANDS+M<ETHENS=S+M
 400 I=PEEK(S):IFI<>32THENI=S:GOTO520
 410 REM SEE IF WE HIT ANYTHING.
 420 IFNTHENPOKEN,32
 430 REM PRINT SIDE BARRIER (MOVES EVERY
THING UP). SEE IF WE HIT THE SH
 440 PRINTTAB(TB)CHR$(93):I=PEEK(S):IFI<
>32THENI=S:GOTO520
 450 POKES,SH:FORX=1TOT:NEXT
 460 REM IF NO SHELL MOVING, DELAY A MOM
ENT-AND THEN START AGAIN.
 470 IFFTHENFORX=1TO10:NEXT:GOTO290
 480 REM MOVE THE SHELL THREE TIMES AND
SEE IF YOU HIT ANYTHING.
 490 FORI=1TO3:IFPEEK(N+1+L+1)<>32THENI=
N+L+1:F=1:N=0:GOTO520
 500 N=N+1+L:NEXT:POKEN,124
 510 GOTO290
 520 IFPEEK(S)=93THEN630
 530 Y=I-3-3*L:POKEI,32:FORX=1TO6
 540 POKEY+RND(8)*6+INT((RND(8)*6))*L,12
3+RND(8)*30
 550 NEXT:POKEY+3*L+1,32:IFI<>5THEN450
 560 PRINTTU" SHIPS GONE!!";:FORX=1TO900
:NEXT:FORX=1TO22
 570 POKEP+X+L,32:NEXT
 580 TU=TU+1:IFTU<6THENGOSUB680:GOTO230
 590 FORX=1TO10:PRINT:NEXT:PRINT"FIVE SH
IPS DOWN ":PRINT"SCORE :"SC
 600 INPUT"TRY AGAIN";A$:IFASC(A$)=89THE
NSC=0:TU=1:F=1:GOTO200
 610 RUN"BEXEC*
 620 REM IF NOT DISK THEN 610STOP
 630 FORX=1TO32:POKEI,X:NEXT:PRINT"BONUS
! BONUS!
 640 FORX=1TO999:NEXT:SC=SC+LX
 650 FORX=1TO999:NEXT:SC=SC+50:PRINT" *
* SCORE "SC" * *":FORX=1TO999:NEXT
 660 GOSUB680:GOTO230
 680 A=PEEK(128):B=PEEK(129):POKE128,0:P
OKE129,212:S$=" "
 690 FORI=1TO7:S$=S$+S$+" ":NEXT:POKE128
,A:POKE129,B:RETURN
 700 REM INSTANT SCREEN CLEAR FOR C1MP:
SEE LETTERS TO EDITOR FOR C2/4
 710 VERSIONS. LAST MONTHS JOURNAL FOR C
1P VERSION.
```

****

WIN FREE TEXT EDITING FOR YOUR SYSTEM

****

ONE OF OUR SOOPER DOOPER NEW ROMS OR A SUPERDISK CAN BE YOURS ABSOLUTELY FREE!!!. WE ARE GOING TO OFFER A PRIZE FOR THE BEST ARTICLE, SUGGESTION, OR LETTER PUBLISHED NEXT MONTH. THE PRIZE DEPENDS ON WHAT SYSTEM YOU HAVE. IT WILL BE A C1S ROM (FULL EDIT AND ALL THAT STUFF) FOR A C1P, A C1E (SAME BUT BETTER) IF YOU HAVE A C2/4/8 BASIC IN ROM MACHINE, OR A SUPERDISK & SUPERCOPY PROGRAM IF YOU HAVE A DISK SYSTEM. IF YOU HAVE ALREADY PURCHASED ONE BEFORE THE NEXT ISSUE, THE PRIZE WILL BE A REFUND.

SECOND PRIZE WILL BE AS USUAL, GIFT CERTIFICATES FOR AARDVARK MERCHANDISE TO ALL WHOSE CONTRIBUTIONS ARE PRINTED.

****

AARDVARK
TECHNICAL SERVICES
1690 Bolton, Walled Lake, MI 48088