A PROGRAM TO PLAY CONTRACT BRIDGE

by

GAY LORAN CARLEY

S.B. Massachusetts Institute of Technology (1961)

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June, 1962

Signature of Author

Department of Electrical Engineering, May 25, 1962

Certified by

Thesis Supervisor

Accepted By

Chairman, Departmental Committee on Graduate Students
DISCLAIMER NOTICE

Due to the condition of the original material, there are unavoidable flaws in this reproduction. We have made every effort possible to provide you with the best copy available.

Thank you.

The following pages were not included in the original document submitted to the MIT Libraries.

This is the most complete copy available.

Page 45
This thesis is a report on the second phase of an effort by the author to develop a skillful bridge-play program for the IBM 7090 computer. The first phase consisted of a study and some programming of the bidding, whereas this phase was more oriented toward the play.

Two important things have been accomplished. (1) A program which plays acceptable, but not outstanding bridge has been developed. (2) A large complex of bridge-oriented subroutines now exists, forming a "language" with which new strategy ideas may be quickly turned into a working program. It is hoped that the framework developed may be of use to future bridge programmers.

In the present program the bidding is quite weak. This is because an extremely sophisticated bidding program is well under way, and it was deemed undesirable to spend much effort on an intermediate version. The weak bidding system is able to count Goren points and make elementary partnership considerations.

The present play routine runs some 3000 FAP instructions and plays quite respectably. It is essentially a long list of rules-of-thumb. A typical rule: Defender, can follow suit, third card to trick, second card high, probability the trick can be won in this hand more than .5 -- play lowest such winner.

The programming has been kept very flexible so that a major change in strategy could be made with a minimum of wasted or duplicated work. This has resulted in more than 60 subroutines, only a few of which involve any commitment to strategy. Subroutines range from a short routine which reverses the accumulator end-for-end to an elaborate routine which computes a priori probabilities. Because the development of these routines represents the major effort on this project, and because they may be of use to some other bridge programmer, a short description of each is given herein.

Since the program has grown to substantial length, considerable thought and experimentation have been given the debugging problem. Resulting techniques have improved greatly the effectiveness of debugging.

Thesis Supervisor: Claude Shannon
Title: Donner Professor of Science
ACKNOWLEDGEMENT

Experimental work on this project was done with the facilities of the MIT Computation Center, under problem number H1403. The project was supported in part by funds from the International Business Machines Corporation.
TABLE OF CONTENTS

Title page
Abstract
Acknowledgement
Table of Contents

I. The Game of Bridge
II. The Present System for Bidding
III. The Present System for Play
IV. An Advanced Bidding Program
V. Some Thoughts on an Advanced Play Program
VI. Time Sharing Applied to Bridge
VII. The Organization of the Program
VIII. A Typical Subroutine
IX. Experiments in Debugging
X. A List of All Subroutines with a Brief Description of Each

Bibliography
I. THE GAME OF BRIDGE

Bridge is played by four players with an ordinary playing deck of 52 cards. Players sitting opposite each other at the table are partners.

The 52 cards are dealt 13 to each player. The game begins with the "bidding", a coded conversation among the players explained below. By the end of the bidding one player has been established as "declarer" and one suit possibly picked as "trump". The second phase of the game, the "play", is opened by the player to the left of the declarer by exposing one of his 13 cards. After this move, called the "opening lead", the player opposite declarer exposes all 13 of his cards and retires from the game. His cards, the "dummy", are henceforth played by his partner, declarer. Play then proceeds clockwise around the table for the 2nd, 3rd, and 4th moves of the play (the fourth move being made by declarer). This completes the first of thirteen "tricks". Each player to a trick must play a card of the same suit as the lead. If he has no such card, he may play any card he has.

The winner of a trick is the player who plays the highest card of the led suit, or, if any trump cards have been played to the trick, the highest card of the trump-suit. Cards rank A K Q J 10 9 8 7 6 5 4 3 2. The winner of a given trick, which may be any of the four players, becomes the leader to the next trick and may lead any of his remaining cards. This continues through the thirteenth trick when all players are out of cards. The total number of tricks won by declarer
and dummy is counted. This number is then converted monotonically to a point score for declarer's team. The exact form of the conversion depends on the bidding.

The bidding: The bidding follows the dealing of the cards and proceeds the play. One player (defined as North throughout this program) starts the bidding by calling (saying) one of 38 possible bids. The player to his left then makes a call and bidding proceeds clockwise around the table from 4 to about 250 bids. (typically 8) The bidding is terminated when the call "pass" is heard three times in succession. The 38 legal bids are "pass", "double", "redouble" and an ordered set of 35 consisting of five suits at seven levels. The five suits are, in order, Clubs, Diamonds, Hearts, Spades, and No Trump, and the levels are one through seven. The level dominates the suit, putting "five Clubs" above "two Spades". Each of these 35 ordered bids is legal only if it or a higher bid has not been called previously. "Double" is legal only if the last non-pass call was one of the 35 and was called by an opponent. "Redouble" is legal only if the last non-pass bid was "double" and made by an opponent. "Pass" is legal at any time.

The last of the 35 ordered bids heard before three passes in succession is the "contract". The player who made this call, or his partner (which ever mentioned the suit first) becomes declarer. The suit becomes trump. The level and suit, as well as any "double" or "redouble" made after this bid, determine the monotonic curve. The zero of this curve is between the level +5 and the level +6.
tricks.

This entire process, bidding and play, which takes humans about 10 minutes, is repeated a number of times until a "rubber" is complete. (The number of deals in a rubber may range from two up). At the end of the rubber, the side with the largest number of points wins the rubber.

II. THE PRESENT SYSTEM FOR BIDDING

Here are the rules used by the present bidding routine, LEV1B, routine EX. From the point of view of a bridge enthusiast, they are exceptionally crude. Because of the nature of the game, fortunately, the inferior level of the machine's bidding will not always be apparent to the other players. The ability level is about that of a person who has played a dozen or so hands of bridge and has little interest in the game.

Definitions:

1. **Card Value**: Ace = 4, King=3, Queen=2, Jack=1, others=0
2. **Distribution Value**: void (no cards in a suit)=3,
   Singleton=2, doubleton=1, other length=0.
3. **Hand Value**: total card values plus total distribution values.
4. **Suit value**: total card values in suit plus suit length.
5. **Assumed partner's hand value**: (if two categories apply, use highest)

   A. Partner opened the bidding
   B. Partner raised your suit

   14
   10
C. Partner opened the bidding and raised your suit  
D. Partner mentioned a suit other than one you 
   previously mentioned  
E. Partner has called "double"  
F. Partner opened the bidding and later mentioned 
   a suit other than one you previously mentioned  
G. Partner has constantly passed.

6. Point to level correspondence table

<table>
<thead>
<tr>
<th>Partnership Value</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less than 20</td>
<td>None</td>
</tr>
<tr>
<td>21-22</td>
<td>1</td>
</tr>
<tr>
<td>23-25</td>
<td>2</td>
</tr>
<tr>
<td>26-28</td>
<td>3</td>
</tr>
<tr>
<td>29-30</td>
<td>4</td>
</tr>
<tr>
<td>31-32</td>
<td>5</td>
</tr>
<tr>
<td>33-36</td>
<td>6</td>
</tr>
<tr>
<td>More than 36</td>
<td>7</td>
</tr>
</tbody>
</table>

The algorithm:  

1. Opening bid (no non-pass bids yet in auction)  

Your hand value is  

a) Less than 13 pass  
    bid 1 of highest value suit (not NT) 

b) 13-15  
    bid "One No Trump"  

c) 16-18  
    bid 2 of highest value suit (not NT)  

d) 19-21  
    bid "Two No Trump"  

e) 22-24
2. Partner bid one of your two best suits when he last spoke. Your hand value is
   a) less than 8   pass
   b) 8-12         raise him 1
   c) more than 12 raise him 2

3. The last non-pass bid was an opponent's bid in one of your two best suits. Your hand value is
   a) more than 11 "double"
   b) less than 12 "pass"

4. Other than 1, 2, or 3. Add this hand's value to assumed partner's hand value. Then pick a suit at random weighted by the suit values.
   a) cannot raise suit without getting above correspondence table. "pass"
   b) otherwise raise as little as possible.

III. THE PRESENT SYSTEM FOR PLAY

This is the algorithm used by the present playing routine, LEV1P, routine EP. Its playing level is about that of the average bridge player.

All references to probabilities refer to a priori probabilities and do not reflect any strategy used in closed hands, but do reflect the current history of the play.

If a given category does not apply, the machine proceeds
to the next category of the same level. If it does apply, the machine proceeds to the subcategories beneath it.

After the card has been selected by these rules, all equivalent cards in the hand are generated. Two cards are equivalent if (1) they are the same suit and (2) there exists no card between them which is either in another hand or played to the current trick. Then, except on the opening lead when this process isn't used, one of the equivalent moves is picked at random and that is the final decision. Such equivalent moves are completely equivalent operationally, but may (except moves by dummy) convey different information to the other players.

The algorithm:

I. Opening lead:

A. Pick suit (then go on to B for card)
   a) pick longest suit unless tie.
   b) pick highest point-count suit with more than two cards.
   c) pick randomly any suit with more than three cards.

B. Pick card in that suit
   a) If the suit is led by A K Q, pick King.
   b) If the suit is led by n, n-1, and n-2(or n<3) pick card n.
   c) pick the fourth highest, if there is one.
   d) Pick the lowest card in the suit.
II. Leads other than the opening lead:

A. Defenders

a) partner has bid a suit (other than no trump) and he has a chance of winning this trick. Lead the lowest card of partner's first bid suit.

b) There exists a card with a probability more than .4 of holding this trick. Lead highest card of suit containing card with highest such probability.

c) This hand holds the second, third, or more in sequence of the cards out in some suit. Lead the lowest card in the longest such suit. (random if tie)

d) This hand has trumps. Lead lowest trump.

e) Lead lowest card of longest suit without violating protection in that suit. Protection consists of 2nd X or 3rd X X. (more than one such suit: random)

f) Violate 3xx protection before 2x, then random.

B. Declarer (or dummy)

a) Trick can be won in a non-trump suit in other hand with probability greater than .6. Lead lowest card of shortest such suit. (tie-random)

b) Trick can be won in a non-trump suit in this hand with probability greater than .5 and opposite hand is void in suit. Lead highest card from longest such suit.
c) Partnership can definitely win a trump trick. Lead up (with lowest trump) if possible, otherwise lead lowest possible trump winner in this hand.

d) An Ace is outstanding in trump or a suit with more than one card on a side with trumps left. Lead lowest card of that suit.

e) Choose randomly from the lowest cards of each suit.

III. It is possible to follow suit.

A. Second card to trick being decided

a) An honor (AKQJ 10) was led.

i. This hand can top the lead. Play lowest card higher than the lead.

ii. Play lowest card.

b) A non-honor was led

i. Declarer (or dummy)

1) The trick can be won in opposite hand. Play lowest card of suit.

2) With probability more than .7 the trick can be won in this hand. Play highest card of suit.

3) Play the lowest card, if any, higher than the card led.

4) Play lowest card of suit.

ii. Defenders

1) It is possible that trick can be won in this hand. Play lowest such possible winner.
2) Play the lowest card, if any, other than an honor, higher than the card led.

3) Play lowest card of suit.

B. Third card to trick

   a) Trick won by partner with probability more than .7. Play lowest card of suit.
   b) Declarer (or dummy)
      i. The trick is definitely won by the second card. Play lowest card of suit.
      ii. There is one and only one card in the defenders' hands such that all higher cards are in this hand or the opposite hand, at least one in this hand, and this hand contains the next lowest card (ignoring those in opposite hand). Play this next lowest card.
      iii. This hand can win this trick with probability greater than .7. Play highest such winner.
      iv. Play 2nd highest card. (If only one, play it.)

C. Last card to trick

   a) Partnership has already won trick. Play lowest card in suit.
   b) Partnership cannot win this trick. Play lowest card in suit.
   c) Declarer: play highest card which will win trick.
   d) Defender: Play lowest card which will win trick.
IV. It is not possible to follow suit.

The decision here is whether to trump or not. The following rules either specify a particular trump card or "side suit". If it is "side suit", the rules following this section are used for the choice of the specific card. If the hand is void in trump, or there is no trump, the machine transfers to this section immediately.

A. Second card to trick, declarer or dummy.

a) Left-hand opponent definitely can trump.

i. Both hands have a trump higher than his highest possible trump and this hand has fewer or equal trumps than opposite hand. Play "side suit".

ii. This hand has a trump higher than his highest possible trump. Play lowest such trump.

iii. This hand has fewer (not equal) trumps than opposite hand. Play lowest trumps.

iv. Play "Side suit."

b) Left-hand opponent may not be able to trump.

i. Probability of being overtrumped by him less than .3 and this hand has fewer trumps than opposite. Play lowest such trump.

ii. Play "side Suit".

B. Second card to trick, defenders

a) A trump has a chance of winning. Play lowest such trump.
b) Play "side suit."

C. Third card to trick

a) Declarer: (or dummy)

i. Second card to trick is high

1) Probability is greater than .7 this hand can win by trumping. Play lowest such trump.

2) Play "side suit".

ii. Probability is greater than .2 lead may be trumped or topped by left opponent.

1) He is known out in led suit. Play lowest trump which definitely can beat him or lowest trump which will force him to use his highest trump.

2) Play lowest trump.

iii. Play "side suit".

b) Defenders

i. Partner high

1) Chance of partner's holding trick is greater than .5. Play "side suit."

2) Chance of successful trump by this hand is greater than .5. Play lowest such trump.

3) Play "side suit".

ii. Chance of successful trump by this hand is greater than .5. Play lowest such trump.

iii. Play "side suit".
D. Last card to trick.
   a) Partnership high. Play side suit.
   b) This hand can win trick by trumping. Play lowest such trump.
   c) Play "side suit."

"Side Suit"

A. This hand possesses a singleton or doubleton of some side suit which has none of the highest 1/3 cards out. Play lowest card of that suit. (More than onesuch suit: play doubleton first, then random.)

B. Play lowest card of longest suit not runable. More than one such suit: play lower point count suits first.

A runnable suit is a suit which contains in this hand (declarer include opposite hand) the highest n cards out and all opponents can be exhausted of the suit (probability more than .5) after n rounds. Also, if this condition will occur after some number of rounds.

C. All suits runable. Violate higher order runnable suits first.
IV. AN ADVANCED BIDDING PROGRAM

A quite sophisticated bidding system is now in the late stages of development. This bidding system (to be administered by BID, routine 2) is built around six strategy-oriented subroutines. Two of them are dependent on the particular bidding language used. (this program uses Goren bidding throughout). The other four involve a commitment to a basic strategy but may be used with any bidding system, such as the Italian system.

The basic strategy of the advanced system is
(a) generate plausible bids, (b) evaluate each with respect to the four functions a bid should perform, and (c) choose the bid with the best overall evaluation. Those four functions are: 1) providing useful information to one's partner, 2) eliciting useful information from one's partner, 3) providing useless or deceptive information to one's opponents and 4) setting the level of the bidding.

In more detail: First a (bidding system dependent) routine is called which to form, based on the bidding, a priori probabilities, and the known hand, an accurate picture of the concealed hands. Hands are evaluated along a list of "interesting" dimensions such as: number of aces, evenness of distribution, length in Clubs, etc. Each quantity and an estimate of its uncertainty is computed.

Using this data as input, four evaluation routines are entered for each plausible bid to calculate a numerical evaluation (in units of "expected score gain") the value of the bid along each of the four lines. These evaluations
are made with decision structures and ad hoc formulas similar in philosophy to the bidding and play algorithms given herein.

By continually improving the quality of these subunits it should be possible to evolve a program of expert bidding skill.

V. SOME THOUGHTS ON AN ADVANCED PLAYING PROGRAM.

The real mark of the expert bridge player is in the play. Although the problem is much better defined in the play than in the bidding, prospects for an outstanding play program in the near future are not nearly so encouraging.

Considerable work (thought) has been given to the development of an expert play program, but the results to date have been discouraging. The whole rationale behind working on games is the hope that they may provide a bite-size stepping stone to more general results but often the game problem seems equivalent rather than preliminary to solving the whole artificial intelligence problem.

The approach used was mainly one of reading what the experts had to say, analyzing in great detail specific bridge hands (a process that takes hours per hand), comparing the analyses to the one or two paragraph analyses of the same hands by bridge experts, and analyzing the analysis to see how to automate it. Since similar things have been tried in chess and checkers, here are some comparisons between bridge and these games:

The blow-by-blow move tree analysis which is the heart
of chess is present in bridge but not at all central. Bridge analyses are more "closed" than chess analyses. A chess analysis, right up to the end-game, is usually directed to intermediate or sub-goals and state-of-the-board advantage. A bridge analysis, even one made early in the play, usually covers all 13 tricks. Positional advantage exists in bridge (such things as entries), but is less important than in chess. Frequently in the play there are as many as 4 or 5 moves whose order and time of play is clearly immaterial. A similar phenomenon does not occur in chess, where a strong "amplification" effect keeps any two moves from being really equivalent. Chess and bridge do share the "brilliant" move, the highly surprising or deceptively minor move which turns the course of the game. The player who makes the move usually has no idea of why he thought of it and may even present a rationalization which is invalid on close scrutiny. Except for the brilliant move, a bridge analysis usually consists of reducing the problem to a handful of overall strategies and then looking carefully into each one.

Certainly much more work needs to be done before a program at this level can even be started.

VI TIME SHARING APPLIED TO BRIDGE

This program can simultaneously play two independent games of bridge. The necessity for this is one of economics. A hand of bridge requires about 10 minutes, a sizable amount of time on a modern machine. When the machine is playing only one of the four table positions (this program can play any number
of positions), it is spending about 75% of its time waiting for other players to move. Unlike human beings who think about their next decision in advance, this program can only consider a move when it's its turn.

The program has plenty it could be doing during these idle minutes. There is always a backlog of "book games" to be played from tape. The answer, then, is to have the machine pick up and put down the off-line game between moves in the on-line game.

The simplest way to do this is to have two identical programs inside the machine. Although programming languages make it easy to do this, there just isn't enough space in the machine's memory for two bridge programs. (One is starting to be a squeeze.) However, registers containing fixed information (including program) need not appear twice. There are about 10 instructions for every register of actual "memory", hence only 10% more memory is really required.

It would be possible to achieve this result using ordinary programming (by storing a list of all temporary memory locations) but it is much more efficient and fool-proof to locate all temporary storage in a central area. (Temporary storage at the higher levels of the program's hierarchy must not be in this area). Then, when it is time to change games, this area and another just like it are switched bodily, a process requiring milliseconds. (The machine's active registers, 170 bits, must also be restored).

Programming in this convention (all storage in COMMON) has proved exceptionally easy because of the following
devices available on the 7090.

1. Indexing — whereby an address may be modified by an active register.

2. Indirect addressing — whereby a computed address may be used without actually modifying an instruction.

3. The EXEC instruction — whereby a computed instruction may be "executed" without actually transferring control to the place where the instruction is stored.

In the earlier days of computers one of the important reasons for storing programs in memory was that they could then be operated on as data. It is interesting to note that this feature is now entirely dispensable.

VII THE ORGANIZATION OF THE PROGRAM

Because there are no final answers to the mechanized bridge problem, and because of the size of the program, it has been written as a growing and flexible experimental tool rather than a monolithic creation which will suddenly become "done" at some instant. As an experimental tool, it should prove of extreme use to anyone wishing to test an idea for playing bridge by machine. Not only are all the bookkeeping, card dealing, bridge input-output, etc. routines available but also a library of bridge-oriented manipulating routines forming a "language" in which a
a particular strategy may be programmed. Anyone planning to use these facilities should become familiar with them before he does any actual programming. The following pages should convey a fairly accurate picture of what is available and what constraints are imposed on any future additions to the program. For more detailed information, consult the author.

One of the diseases of the programming art is the tendency at every level to fabricate a large number of "conventions". During the development of this program a number of conventions were experimented with. Some became artificial and were abandoned. However, some have proved extremely useful and are explained below.

1. Status. Another user of this program should become a second programmer under Computation Center problem number M1403.

2. Master programs. All routines, even master control routines, should be assembled as subroutines. This facilitates comparing the performance of several systems or incorporating several systems into a more general master control routine. The nominal "main program" should always consist of an extremely short routine which transfers control to the master program desired. This same convention is applied lower in the tree where the routine PLAY consists of a single instruction calling the current play strategy routine.
The following conventions are all illustrated in the sample routine which follows this section.

3. The few registers in the beginning and end of a routine follow a very rigid format giving certain vital information about the routine. These registers are looked for and used by special core-scanning programs and an error may result if they are incorrect.

4. Inter-run memory. The Computation Center provides no convenient vehicle by which a routine may communicate with itself from one run to another. Since this is desirable in a bridge program, programs have been written which provide this vehicle in a very convenient form. The ultimate vehicle is binary punched cards. At the end of each routine the programmer makes a list of all inter-run memory registers. (In the sample program, -BJCALS- is such a register). The contents of these registers will be automatically restored before each run in spite of relocation, possible changes in the routine or changes to the list.

5. Common storage. Because of time sharing, it is vital that no register in the body of the program be changed during running. All changed registers are to be in COMMON. A certain amount of COMMON is allotted to each routine (as much as it requires) and all registers in that area are named starting with the routine's two-letter code. ("BJ" in example). The first register must be named xxHOL4 and the last (lowest in core) xxEX. A routine
A routine must never operate on the common area of another routine except for registers designated as "input" to the other routine. A routine must never operate on its own input registers.

6. Hold and restore. All active registers of the machine (the three index registers, the 38 bit accumulator, the MQ and the SI) must be left unchanged by a routine, unless one of these registers is designated as "output" for the routine.

7. Codes. Of course a complete set of codes has been developed and it is mandatory that all routines be compatible with these codes. Codes include: a 2 bit code for suits and table positions (Clubs and North are 00), a six bit code for cards and bids (101101 = Heart King; 101011 = "Five Spades"), a three-register format for a hand and a 42 register format for a complete hand (deal) history.

VIII A TYPICAL SUBROUTINE

So that the reader may have some idea what is being talked about, a quite typical routine listing is included.
THIS ROUTINE MANIPULATES A SET OF 13 OR FEWER CARDS STORED IN
STANDARD FORMAT. STANDARD FORMAT IS THREE REGISTERS, 6 CARDS IN
THE FIRST TWO REGISTERS IN CORE AND THE 13TH CARD IN THE 3RD
REGISTER BITS 0-5.

ON INPUT THE LOCATION OF THE FIRST OF THESE 3 REGISTERS IS STORED
IN -BJHAN- AND A SINGLE CARD IS STORED IN BITS 0-2 OF -BJCD-.

ON CALL TO *CDIN* THAT SINGLE CARD IS INSERTED INTO THE SEQUENCE OF
13 CARDS.

ON CALL TO *CDOUT*, THAT CARD IS REMOVED FROM THE SEQUENCE OF 13
CARDS IF IT IS PRESENT. BLANK (OCT. 00) IN -BJCD- WILL CAUSE NO
CARD TO BE ADDED OR REMOVED. NO MATTER WHICH CALL IS USED;
HOWEVER, THE 13 CARDS WILL BE PROPERLY ARRANGED ON EXIT. PROPER
ARRANGEMENT IS HIGH CARDS FIRST AND BLANKS AT THE END.

ALL DUPLICATES ARE REMOVED IN A PROPER ARRANGEMENT.

THE REGISTER -BJIND- MAY ALSO BE OF SOME INTEREST. EACH BIT

* SIGNALS SOME CONDITION AS FOLLOWS=
* BIT 35 CARD TO BE ADDED BY *CDIN* IS ALREADY PRESENT.
* BIT 34 THE CARD SPECIFIED IN *CDOUT* WAS FOUND AND REMOVED.
* BIT 33 *CDIN* WAS CALLED AND THE RESULT CAME TO 14 CARDS.
* THE LOWEST ONE, WHICH WAS NOT THE ONE IN -BJCD-;
* WAS DROPPED.
* BIT 32 THE CARD SPECIFIED BY *CDIN* WAS PUT IN PLACE.
* IF THIS BIT IS 0 AFTER CALL TO *CDIN*, IT MEANS EITHER
* NO-CARD (BLANK) WAS SPECIFIED OR THERE WERE ALREADY
* 13 CARDS AND THE ONE IN -BJCD- WOULD MAKE THE 14TH
* AND LOWEST;
* BIT 31 MULTIPLE SCANS OF THE 13 CARDS WERE REQUIRED. THIS
* MEANS THAT THERE WAS AT LEAST ONE INVERSION OF
* ORDER IN THE ORIGINAL 13 CARDS;
* BIT 30 1 FOR *CDIN* 0 FOR *CDOUT*;
* BIT 29 THERE WAS AT LEAST ONE DUPLICATE CARD IN THE
* ORIGINAL 13. (ALL DUPLICATES ARE CORRECTED).

HISTORY OF THIS ROUTINE=
* 4/27/62 WRITTEN
* 4/28/62 PROOFOREAD
* 5/1/62 KEYPUNCHEO.
* 5/21/62 NEATIFIED SLIGHTLY FOR INCLUSION IN M.S. THESIS

ROUTINES CALLED = *NONE*

ENTRY CDIN
ENTRY CDOUT
ENTRY BEGBJ
ENTRY ENDBJ
BEGBJ VFD 02170513745, 15/ENDBJ
BCI 1*BJBBBJ
BCI 1*CDIN
BCI 1*CDOUT
PZE BJHOL4*BJEX
<table>
<thead>
<tr>
<th>CDIN</th>
<th>SXD</th>
<th>BZHOL4×4</th>
</tr>
</thead>
<tbody>
<tr>
<td>SXD</td>
<td>BZHOL2×2</td>
<td></td>
</tr>
<tr>
<td>SXD</td>
<td>BZHOL1×1</td>
<td></td>
</tr>
<tr>
<td>STQ</td>
<td>BZHOLQ</td>
<td></td>
</tr>
<tr>
<td>SLW</td>
<td>BZHOLA</td>
<td></td>
</tr>
<tr>
<td>ARS</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>STO</td>
<td>BZHOLA-1</td>
<td></td>
</tr>
<tr>
<td>STI</td>
<td>BZHOL1</td>
<td></td>
</tr>
<tr>
<td>LDI</td>
<td>=040</td>
<td></td>
</tr>
<tr>
<td>STI</td>
<td>BZIND</td>
<td></td>
</tr>
<tr>
<td>TRA</td>
<td>INIT</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CDOUT</th>
<th>SXD</th>
<th>BZHOL4×4</th>
</tr>
</thead>
<tbody>
<tr>
<td>SXD</td>
<td>BZHOL2×2</td>
<td></td>
</tr>
<tr>
<td>SXD</td>
<td>BZHOL1×1</td>
<td></td>
</tr>
<tr>
<td>STQ</td>
<td>BZHOLQ</td>
<td></td>
</tr>
<tr>
<td>SLW</td>
<td>BZHOLA</td>
<td></td>
</tr>
<tr>
<td>ARS</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>STO</td>
<td>BZHOLA-1</td>
<td></td>
</tr>
<tr>
<td>STI</td>
<td>BZHOL1</td>
<td></td>
</tr>
<tr>
<td>STZ</td>
<td>BZIND</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INIT</th>
<th>CAL</th>
<th>BZHAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANA</td>
<td>=0777777</td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>=0100003</td>
<td></td>
</tr>
<tr>
<td>SLW</td>
<td>BZIA1</td>
<td></td>
</tr>
<tr>
<td>CAL</td>
<td>BZCALS</td>
<td></td>
</tr>
</tbody>
</table>

COUNT OF THE NUMBER OF CALLS TO THIS ROUTINE.

*ADD =1
| SLW  | BZCALS |
| CAL  | BZCD  |

ANA =07700000000000 DROP ANY TRASH
| SLW  | BZCD2  |

A
| STZ  | BZH0   |
| STZ  | BZSIG  |
| AXT   | 3×1    |
| CAL   | =077777777777 |

A
| SLW  | BZH1   |
| AXT   | 36×2   |

M
| TRA  | C      |
| TXL   | D×1×1  |
| TIX   | **3×2×6 |
| AXT   | 36×2   |

C
| TIX   | **1×1×1 |
| CAL   | BZIA1  |
| ALS   | 36×2   |
| LDI   | BZIND  |

C2
| ANA   | =07700000000000 |
| RFT   | 21       |

| TRA   | B      |
| NZT   | BZH0  |

B
| TRA   | L      |
| LAS   | BZH1  |
| TRA   | H      |
| TRA   | G      |

F
| SLW   | BZH1  |
| NZT   | BZH0  |
| TRA   | M2    |

BITS 31 AND 35, NON-FIRST SCAN OR EQUAL CARD FOUND ON *CDIN*
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LUQ</td>
<td>BJHO</td>
</tr>
<tr>
<td>ORA</td>
<td>=1</td>
</tr>
<tr>
<td>XCL</td>
<td></td>
</tr>
<tr>
<td>STQ</td>
<td>BJHO</td>
</tr>
<tr>
<td>ANA</td>
<td>=07700000000000</td>
</tr>
<tr>
<td>M2</td>
<td></td>
</tr>
<tr>
<td>PAI</td>
<td></td>
</tr>
<tr>
<td>LDQ</td>
<td>BJIA1</td>
</tr>
<tr>
<td>LGL</td>
<td>36*2</td>
</tr>
<tr>
<td>XCL</td>
<td></td>
</tr>
<tr>
<td>ANA</td>
<td>=00077777777777</td>
</tr>
<tr>
<td>OAI</td>
<td></td>
</tr>
<tr>
<td>PIA</td>
<td></td>
</tr>
<tr>
<td>XCL</td>
<td></td>
</tr>
<tr>
<td>LGR</td>
<td>36*2</td>
</tr>
<tr>
<td>STQ</td>
<td>BJIA1</td>
</tr>
<tr>
<td>G</td>
<td></td>
</tr>
<tr>
<td>TRA</td>
<td>M</td>
</tr>
<tr>
<td>TZE</td>
<td>M</td>
</tr>
<tr>
<td>PXA</td>
<td>0</td>
</tr>
<tr>
<td>LDI</td>
<td>=0100</td>
</tr>
<tr>
<td>OIS</td>
<td>BJIND</td>
</tr>
<tr>
<td>TRA</td>
<td>F</td>
</tr>
<tr>
<td>H</td>
<td></td>
</tr>
<tr>
<td>STL</td>
<td>BJSIG</td>
</tr>
<tr>
<td>ZET</td>
<td>BJHO</td>
</tr>
<tr>
<td>TRA</td>
<td>M</td>
</tr>
<tr>
<td>SXA</td>
<td>BJIX*1</td>
</tr>
<tr>
<td>SAD</td>
<td>BJIX*2</td>
</tr>
<tr>
<td>TXI</td>
<td>++12*6</td>
</tr>
<tr>
<td>TXL</td>
<td>++32*36</td>
</tr>
<tr>
<td>AXT</td>
<td>6*2</td>
</tr>
<tr>
<td>TXI</td>
<td>++1*1</td>
</tr>
<tr>
<td>PAI</td>
<td></td>
</tr>
<tr>
<td>LDQ</td>
<td>BJIA1</td>
</tr>
<tr>
<td>LGL</td>
<td>36*2</td>
</tr>
<tr>
<td>XCL</td>
<td></td>
</tr>
<tr>
<td>ANA</td>
<td>=077777777777</td>
</tr>
<tr>
<td>OAI</td>
<td></td>
</tr>
<tr>
<td>PIA</td>
<td></td>
</tr>
<tr>
<td>XCL</td>
<td></td>
</tr>
<tr>
<td>LGR</td>
<td>36*2</td>
</tr>
<tr>
<td>CAL</td>
<td>BJIA1</td>
</tr>
<tr>
<td>ALS</td>
<td>36*2</td>
</tr>
<tr>
<td>ANA</td>
<td>=07700000000000</td>
</tr>
<tr>
<td>STQ</td>
<td>BJIA1</td>
</tr>
<tr>
<td>LXA</td>
<td>BJIX*1</td>
</tr>
<tr>
<td>LXD</td>
<td>BJIX*2</td>
</tr>
<tr>
<td>L</td>
<td></td>
</tr>
<tr>
<td>TRA</td>
<td>F</td>
</tr>
<tr>
<td>LAS</td>
<td>BJCD2</td>
</tr>
<tr>
<td>TRA</td>
<td>B</td>
</tr>
<tr>
<td>TRA</td>
<td>1</td>
</tr>
<tr>
<td>J</td>
<td></td>
</tr>
<tr>
<td>RNT</td>
<td>40</td>
</tr>
<tr>
<td>TRA</td>
<td>B</td>
</tr>
<tr>
<td>LDI</td>
<td>BJCD2</td>
</tr>
<tr>
<td>SIR</td>
<td>1</td>
</tr>
<tr>
<td>STI</td>
<td>BJHO</td>
</tr>
<tr>
<td>LDI</td>
<td>=010</td>
</tr>
<tr>
<td>OIS</td>
<td>BJIND</td>
</tr>
<tr>
<td>TRA</td>
<td>F</td>
</tr>
<tr>
<td>D</td>
<td></td>
</tr>
<tr>
<td>CAL</td>
<td>BJHO</td>
</tr>
</tbody>
</table>
* A COMMON DECK IS REQUIRED WITH THIS ROUTINE = BJ

* THIS IS THE COMMON AREA FOR ROUTINE #BJ* *COMMON*

BJHOL4 COMMON 1
BJHOL2 COMMON 1
BJHOL1 COMMON 1
BJHOLQ COMMON 1
BJHOLA COMMON 2
BJHOLI COMMON 1
BJHAN COMMON 1
BJIND COMMON 1
BJIX COMMON 1
BJIA1 COMMON 1
BJCALS COMMON 1
BJCD COMMON 1
BJCD2 COMMON 1
BJH0 COMMON 1
BJSIG COMMON 1
BJHI COMMON 1
BJEX COMMON 1

** NOTE TO THESIS READER== THESE COMMON REFERENCES BECAME INCORPORATED WITH THE COMMON FOR ALL OTHER ROUTINES INTO A LARGE *COMMON DECK* WHICH IS ASSEMBLED WITH EVERY ROUTINE.**
IX EXPERIMENTS IN DEBUGGING

Debugging is the major problem in programming. Unless care is taken, the time spent turning a written program into a working program may be many times the time spent on the original. During the course of this project several approaches to debugging have been experimented with. The result of this experience has been a probable doubling of debugging efficiency. Below are some approaches which have been considered and found ineffective:

1. **Debugging as a second order problem.**

   This general attitude results in programming masterpieces that take forever to get working. A misplaced comma quickly eclipses the most elegant of programs. The programmer should think not just in terms of the finished product, but rather in terms of the entire process from first notion to final result.

2. **The brute force approach.**

   This consists of saying: "Let the machine do the work, it's not the number of runs but rather the number of man-hours that counts." As soon as a routine was written, it was punched and run without proofreading. When an error showed up, it was corrected (without looking for other errors) and the routine rerun. The idea was that a large number of routines would be in process at any time and only a few minutes of human effort would be required per error. This whole approach turned out to be grossly in error. There were two main problems. First, bookkeeping and handling
became excessive. It takes at least 10 minutes just for the card handling to submit a run and at least a half hour to locate a demonstrated error. Since around 10% of the instructions on the first draft of a routine are in error, this procedure can quickly use up many man-hours. Second, as work on routines becomes spread out over time, it is hard to remember all the motivations and techniques peculiar to each routine. The reviewing process can be lengthy and can introduce errors of its own.

3. Automated debugging.

Debugging of a routine consists of trying it out in various situations. Could this process be effectively automated? One plan is this: when each routine was written, the programmer would also make up some practice problems for it, in a specified format. An automatic debugging routine (routine J) then enters the routine with each such problem and traces the activity of the unproven routine. The has some possibilities, but so far the debugging program has just become one more thing to go wrong and the format is too rigid to handle a wide variety of routines.

Present Approach

The present approach is a combination of several ideas.

1. Test and dummy routines. For each routine, a short companion routine is written to put it through its paces. This is much more flexible than listing practice problems. In order that debugging may proceed in parallel at different levels, dummy routines are written for any lower (called) routine not yet working.
2. **Bookkeeping.** With many routines in process at the same time, disorganization can quickly set in. The simple device of keeping a notebook with an external (black box) description of each routine, a record of its debugging history, and its current debugging status, completely solves this problem. Filing listings by date instead of routine (the number of runs on a given date is more nearly constant), and keeping an index of listings with a short explanation of what each means makes a large volume of computer output into a finger-tip library of debugging history.

3. **Choice of subroutine size.** The natural tendency is to make subroutines complete units in terms of the overall program. Instead, however, it is better to make routines approximately fixed size (about 200 instructions seems best). This results in somewhat shorter subroutines. Splitting a large routine in two will usually double the speed of debugging since both halves can be debugged in parallel. Of course, the routine must not be too small or side effects, such as debugging the test routine, will cancel any gain.

4. **Proofreading.** By proofreading a routine carefully as soon as it is written, while original motivations are still in mind, the proofreading process is much more efficient. During the course of the debugging this rigorous proofreading is repeated several times; each time trying to find all errors, not just the ones that have shown up in runs.

5. **Programming techniques.** Unless there is a good reason for it, tricky or subtle programming should be avoided. Such programming may save a few registers or microseconds, but
it considerably hampers debugging. Often the simplest of changes can turn a hard-to-follow routine into one that reads fluently. This is very important in debugging.

X. A LIST OF ALL ROUTINES

Here is a list of all subroutines in this program. To save space, only the briefest description is given. For more details, the author should be consulted. They are grouped by categories. The two-letter routine codes are approximately chronological.

1. General input-output routines.

STH, SCH, TSH, FIL, RTN, MLR, TSB, TSB, RLR, CSN, FIN routine N

During input-output a time sharing interrupt might cause an error. These routines buffer the Computation Center's corresponding routines.

MEMORY, KEEP routine AT

This routine automates inter-run memory. At the end of a run it scans core to find information designated as "inter-run memory" and punches it out. At the beginning of the next run it reads the information back into place.

PRINT routine K

This routine makes it possible to print a message without counting characters or setting up a message area. By making it easy to program comments, it encourages their use, valuable in debugging.

NOTE, NOTEQ routine AW
NOTE

This routine prints very short (18 characters or less) messages, making it practical to comment much more often than PRINT would allow. It also prints the time of each message to a 60th of a second and will print the contents of the MQ as the 13-18th characters if desired.

NOTEQ

This routine prints very short (18 characters or less) messages, making it practical to comment much more often than PRINT would allow. It also prints the time of each message to a 60th of a second and will print the contents of the MQ as the 13-18th characters if desired.

PRINT

This routine prints the time in tenths of minutes, hours, 60ths of a second since the start of a run, and the date.

MQ

This routine prints the time in tenths of minutes, hours, 60ths of a second since the start of a run, and the date.

M1403

A short routine which punches binary cards saying "M1403", used to identify punched output of this program.

HLAMAP

This routine does exactly the same thing as the Computation Center's STOMAP, after which it is plagiarized, except that it uses M1403 input-output facilities (routine N).

2. Debugging routines:

TITRAL, LOC, MATCH, KEYS, HALT

This routine, which is entered every 60th of a second (using interrupts from the interval timer), makes available several interesting features (some of which are found on other computers as hardware). It can interrupt if the machine gets in a loop (after a specified length of time), if the machine halts, if certain forbidden registers are touched or regions of memory entered, or if the panel keys are altered. This allows a calling routine to keep complete
control over a called routine. A complicated priority list and algorithm allows this routine to be used at all levels of the hierarchy without interaction. If an interrupt occurs, it is possible to restart the interrupted routine at a later time.

**DEBUG**

This routine supplies unproven routines with practice problems. By using routine I, it is virtually impossible for the unproven routine to seize control. Therefore a number of routines can be debugged in one run without interaction.

**DUMHIS**

This routine generates dummy bridge histories for testing bridge routines. It has a stored "book game" and can simulate any point in this game.

**SIGNAL**

Often, when using test routines, something goes wrong before the entire test is complete. This routine makes it very easy to tell how far things got.

**FLOW**

As large structures of routine become working, it is desirable to know the sequence of calls and time spent in each routine. This routine, normally the single instruction TRA 1,4, is called by all large routines immediately after the SXD xxHOL4,4 instruction. If it is activated, FLOW prints the time to a 60th of a second, the name of the called routine, the name of the calling routine, and the point from which the call was made.
STATE

This routine prints the contents of all the machine's active registers: the instruction counter, 32 bit AC, 36 bit MQ, 36 bit SI, and the three 15 bit index registers. In spite of the simplicity of the idea, this routine has proved very valuable in debugging.

CHDUMP,DIF

CHDUMP dumps in various formats the contents of up to 1000 registers. On subsequent calls to DIF only those registers which have changed since the last dump are dumped. This allows a large number of registers to be "watched" without excessive volume in dumps.

3. Hand manipulating routines

PSUEDO

This routine deals four arranged bridge hands using pseudo-random number generator RANNO, routine AX.

CDIN,CDOUT

This routine adds or removes a given card from a list of 13, and arranges such a list in proper order.

JUSTS

This routine left justifies a given suit from a list of 13 cards, removing all other suits.

SCANH

This routine scans a list of 13 cards at high speed, delivering a single card on each call.
4. Hand function routines:

**DIST**

This routine takes a list of 13 cards and counts the number of cards in each suit.

**GENRY**

This routine, part of the advanced bidding package, looks at a given hand and evaluates it along a series of dimensions of interest in the acution, such as evenness of distribution.

**POINT**

This routine looks at a list of 13 cards and counts the high-card points in each suit. (Ace=4, King=3, Queen=2, Jack=1, Ace and 10=5)

**CDFRS**

This routine looks at a given hand and sees whether a given card is present.

5. Play manipulation routines.

**LEVEL**

This is the level one play strategy administrating routine. It automates the play algorithm given in this thesis.

**SCANN**

This routine scans the play history of a given deal at high speed. At each call, one move (card) is delivered in the order of actual play. The player making the particular move and the leader to the trick are also included in the output.
This routine computes a priori probabilities. Given a logical expression (presently limited to six items), the routine computes the probability of that expression, ignoring all factors of strategy. Items allowed are the location of a particular card, the length of a particular suit in a particular hand, or the upper or lower limit on that length. Connectives "and", "or", and "not" are allowed, but no parentheses.

This routine, which calls routine BG, computes the a priori probability that a given player can top a given card or can trump it.

This routine determines the location, or possible locations, of a given card.

This routine locates and retrieves the current trick in a given deal history.

Given a player and a suit, this routine determines the bounds on the length of the suit in the player's hand.

This routine computes a number of interesting 1 bit functions of the hand (deal) history, such as "have Spades been mentioned in the bidding?" or "Can this hand follow suit?".
OUT

This routine generates a list of all cards in a given suit that are out. A card is "out" if it has not yet been played or has been played to the current trick.

6. Bidding routines

LEVEL
This routine administrates the level one bidding strategy. It automates the bidding algorithm given in this thesis.

SCAN
This routine delivers, one by one, all the calls which were made in the auction in the order they were heard.

BID
This routine administers the advanced bidding system.

GOREN
This routine generates one to eight plausible bids, for use in the advanced bidding system.

NEROG
This routine estimates from the bidding the contents of closed hands, along certain interesting dimensions. It is part of the advanced bidding system.

BOXIV
This routine evaluates a hypothetical bid on how much good (or harm) it does because of information communicated to the machine's partner. It is part of the advanced bidding system.

BOXV
This routine evaluates a hypothetical bid on how
much good (or harm) it does because of information it might cause the machine's partner to give. It is part of the advanced bidding system.

**BOXVI**

This routine evaluates a hypothetical bid on how much good or harm it does because of the level to which it raises the auction. It is part of the advanced bidding system.

**TE** (trick estimator)

This routine, part of the advanced bidding system, estimates how many tricks the machine and its partner could win at a given contract.

7. Bridge oriented input-output routines

**LEVLM**

This is the master control routine for the first on-line bridge playing program. No time-sharing is used. Routine BP and BX make the strategy decisions for the play and bidding respectively.

**BIDTR**

This routine translates bids from the internal bid code (6 bits) to BCD for eventual human consumption.

**CDTR**

This routine translates cards from the internal card code to BCD for eventual human consumption.

**MOUT**

This routine communicates moves to the people in the room. The CRT and on-line printer are used.
READ

This routine reads in off-line book games. When a card is read, the routine decides what the card is asking or telling the machine, and then the master control routine, just above this, is able to call the appropriate routine to take the appropriate action.

HCARD

This routine punches "hand cards". Its input is a hand stored in internal language and its output is a BCD card with the hand punched in appropriate format. These cards are for either human or machine consumption. (they can be read by routine S)

TABLE

This routine prints out a complete deal history, translating from the internal code to an attractive printout.

8. Non-bridge oriented manipulating routines.

SWITCH

This specially written high-speed routine can reverse the 36 bit logical AC end for end in 29 machine cycles.

RANBIT, RANNO

This routine is the random number generator for the entire program. By incorporating a list of quantum-mechanically generated true random numbers, and using a more sophisticated algorithm, it is considerably more random than random number generators available at the Computation Center.
COUNT

This routine, specially coded using lists, counts the number of 1 bits in the 72 bit AC-MQ in 36 machine cycles.

JUST, JUST61

Routine JUST shifts the 72 bit logical AC-MQ left until AC position P is one, and then stores the number of shifts required. (all zeros stores 72). By using special techniques this takes place in 29 machine cycles. JUST61 shifts left the 72 bit logical AC-MQ six bits at a time until six ones are in positions 0-5 of the AC and stores the number of shifts required. (72 if there is no such group of ones present.)

REV

This fast routine reverses a specified group of registers in core.


BCT

This routine translates from binary to binary-coded-trinary. Trinary is used in the packer routines to achieve variable length numbers. (using 11 as an end mark)

LEGAL

Attempting to print a six bit code other than the 48 legal IBM codes will cause difficulties in the printer.

This routine replaces all illegal 6-bit groups in the MQ with blanks.

DCONV
This routine translates numbers from binary to BCD in core. Unfortunately, Computation Center routines which make this translation are quite inflexible and of little use outside of a given format.

This routine converts from BCD to binary in core. It is useful in extracting fields from cards whose format is not known to the internal program.

This routine converts from BCD to binary in core. It is useful in extracting fields from cards whose format is not known to the internal program.

This routine converts between the 12 bit and 6 bit BCD card column codes. Illegal 12 bit codes are checked for.


The following routines are part of a system for packing BCD FAP language cards as efficiently as practical onto 960 bit binary cards. Variable length coding is used throughout, with binary columns and strings of blanks treated as special cases.

This routine administers the packing process.
UNPACK

This routine administrates the unpacking process.

CODE

This routine generates variable length code words for BCD characters, using a list of 64 probabilities.

DECODE

This routine locates and translates the first complete code word in an open-ended string of bits.

FACOUT

This routine arranges variable length code words into binary, x cards, adding sync. information to each card.

PACIN

This routine unwraps a packed card into a string of binary bits, feeding bits as fast as they are consumed by the variable-length decoder.

BCDOUT

This routine forms BCD characters, 12 bit binary columns, and blank-counts into integral Hollerith cards.

BCDIN

This routine takes BCD cards as input and delivers the 5 bit code for each column. Binary columns and strings of blanks are recognized and properly handled for coding into packed form.

11. ML403 Assembler

It is becoming increasingly desirable to perform certain operations on a program before it is seen by the FAP processor. These two routines are the start of a system of pre-assembler routines.
LEVI A routine BD
This is the master control routine for the level one assembler. It essentially is an input-output and bookkeeping routine for TRANS.

TRANS routine AY
The most pressing problem in the pre-assembly category is that the COMMON deck, containing all temporary storage registers, is becoming so large (about 600 cards) it often dwarfs the routines it is attached to. TRANS will automatically generate this deck on tape, saving a great deal of cards and card handling.

12. Miscellaneous
SCORE routine A
This routine, given the contract and the number of tricks won by declarer, computes the standard bridge score.

UPDATE routine BV
This routine takes a given move and adds it to the history of a hand. It makes a complete check for move legality.

SSS (pseudo sense switch) routine C
Computation Center routines are unfortunately geared to a very restricted concept of computer usage. One manifestation of this is that the choice of on-line versus off-line input-output can only be made by the operator at the console. This routine scans memory looking for Computation Center routines and pre-emps them so that this decision may be made internally by bridge program executive routines.
DISCLAIMER NOTICE

MISSING PAGE

p. 45
Bibliography

7. Fortran Assembly Program (FAI) for the IBM 709/7090. Form J28-6098-1, 1961
8. Bridge columns from numerous newspapers.
START routine AS

This routine is called at the beginning of every run. It reads in the inter-run memory (via routine AT), calls the Computation Center's "SETUP" and performs any other initialization that is or may become part of the bridge program.

STOP routine AU

This routine is called at the end of every run. It reads out the inter-run memory (via AT), calls the Computation Center monitor, and performs any other termination operations that are or may become part of the bridge program.