Acorn
Software Products, Inc.
634 North Carolina Ave., S.E., Washington, D.C., 20003 · (202) 544-4259

STRUCTURED BASIC TRANSLATOR

by

Gene Bellinger

Published by Acorn Software Products, Inc.

TABLE OF CONTENTS

INTRODUCTION

Down on the BASIC idea? Tired of attempting to make program modifications without being foiled by line numbers and GOTO's? Have you managed to forget how portions of your programs work because you left out the REMARK statements to conserve memory and speed up execution? If these and other drawbacks of BASIC keep you from getting this done, then SBT can provide some relief!

SBT is not a programming language; it is simply a utility which allows you to write structured programs using PROCEDURES, CALLS, CASE-CALLS, IF-THEN-ELSE, WHILE and UNTIL structures with no line numbers and no GOTO's. A structured program is written using an editor (one provided) and placed in a disk file, or files, which SBT will convert into an executable BASIC program with line numbers and GOTO's as necessary.

SBT is relatively small, i.e. less than 20K including BASIC in its original version, and fast; i.e. SBT will translate its own source code in less than 4 minutes. SBT is fast so you would not hesitate to alter or modify the source code because it took forever to retranslate it.

CONVENTIONS

SBT is a line-oriented translator that understands only three types of lines: STRUCTURE ELEMENTS; COMMENTS; and VALID BASIC STATEMENTS, in that order. Lines may be indented or offset using spaces or tabs which will be removed before the line is processed by SBT. Thus any further reference to the word 'LINE' is meant to imply a character string with the offset characters removed.

STRUCTURE ELEMENTS

All structure elements begin with a '%' character. These lines alone are processed by SBT. The type checking performed on the characters following the '%' is just sufficient to determine uniqueness of the structure element but the full name should be used:

%PROC	%ELSE
%CALL	%ELSEIF
%WHILE	%END
%UNTIL	%ON
%IF	

The structure elements %PROC, %CALL, %WHILE, %UNTIL, %IF, %ELSEIF and %ON all have arguments (to be explained shortly) which must be separated from the structure element by one or more spaces.

COMMENTS

Any line beginning with a non-alphabetic character other than a '%' character is a comment line. These lines are not processed. This allows you a wide variety of comment conventions as desired. These comments serve to document the 'SOURCE' code for later references and have no effect on the final executable BASIC program. There is an exception: a line beginning with 'REM' will be treated as a legal BASIC statement and end up as a REMark statement in the final BASIC program; however, the abbreviation ' will be considered a SBT remark statement and will not be processed.

BASIC STATEMENTS

Any line which is not a structure element and not a comment is assumed to be a valid BASIC statement and is passed to the resultant BASIC program. Note that SBT assumes validity of these statements with no checking performed; it simply assigns a line number as required and outputs the statement.

THE STRUCTURES

PROCEDURE STRUCTURES

A program is composed of one or more procedures, each of which performs a function or some portion of a function. A procedure structure begins with a procedure statement of the form '%PROC' followed by a procedure name of 1 to 8 alpha-numeric characters, with the first being alphabetic. Remember also that one or more spaces must follow '%PROC'.

A procedure ends with a '%END' statement, and all lines from the procedure statement to the end statement are considered to be in the procedure. Since procedures cannot be nested and all other structures also terminate with '%END' statements, a good convention to use is to end the procedure with a '%END-PROC' or '%END-PROC-NAME'. Since '%END' is the extend of the type checking performed by SBT, any helpful information may be included.

Procedures are invoked or executed by using a '%CALL NAME' statement, where 'NAME' is the name of the procedure to be executed, e.g. %CALL TEST1. Although procedures cannot be nested, they may be recursive, i.e. you can %CALL a procedure from within itself without any resulting confusion (to the program).

CASE-CALL STRUCTURE

The CASE-CALL is a one line pseudo-structure which performs a conditional call of one of several procedures depending on the value of its argument. The CASE-CALL is of the form:

```
%ON (ARGUMENT) CALL PROC1,PROC2,...,PROCn
```

where PROC1, PROC2, ..., PROCn are the actual procedure names desired, and (argument) is some numeric expression. The CASE-CALL translates to:

```
ON (argument) GOSUB N1,N2,..,Nn
```

where the Ni's are the actual line numbers where the desired procedures begin as determined by SBT.

```
/* SAMPLE FOR PROC, CALL, CASE-CALL
PRINT "IN MAIN CALLING TEST"
%CALL TEST
PRINT "RETURN FROM TEST"
/* GENERATE A RANDOM CALL TO ONE OR TWO
I=INT(2*RND(1))+1
%ON (I) CALL ONE,TWO
PRINT "END SAMPLE"
/* AN END FOR THE MAIN PROGRAM
END
```

```
%PROC TEST
    PRINT "IN TEST"
    IF A=2 THEN RETURN
    A=A+1
    /* A RECURSIVE CALL
    %CALL TEST
    /* AN END FOR TEST, NO RETURN NEEDED
%END-PROC TEST

%PROC ONE
    PRINT "IN ONE"
    %CALL TWO
%END-PROC ONE

%PROC TWO
    PRINT "IN TWO"
%END-PROC TWO
```

The above sample program translates to produce:

```
1 PRINT "IN MAIN CALLING TEST"
2 GOSUB 500
3 PRINT "RETURN FROM TEST"
4 I=INT(2*RND(1))+1
5 ON (I) GOSUB 1000,1500
6 PRINT "END SAMPLE"
7 END
500 PRINT "IN TEST"
501 IF A=2 THEN RETURN
502 A=A+1
503 GOSUB 500
504 RETURN
1000 PRINT "IN ONE"
1001 GOSUB 1500
1002 RETURN
1500 PRINT "IN TWO"
1501 RETURN
```

LOOP STRUCTURES

SBT supports two repeat or loop structures %WHILE and %UNTIL. %WHILE loops test at the beginning of the loop and %UNTIL loops test at the end of the loop. The FOR-NEXT construct may be used for loop control as desired as it is passed to basic as any other valid basic statement with no processing performed on it.

WHILE STRUCTURE

The %WHILE structure is of the form:

```
%WHILE conditional-statement
    sequence-of-statements
%END-WHILE
```

where the conditional-statement is some test to be performed, e.g. I<4, B=A, etc. The conditional is evaluated at the beginning of the loop and the loop repeats "WHILE" the conditional is true. Thus the sequence-of-statements may be executed zero (0) times. Note, the '%END-WHILE' need only be a '%END', but is used in this form as a programmer aid to remember what this particular %END is terminating.

```
/* WHILE EXAMPLE
/* REMOVE SPACES FROM FRONT OF STRING

%WHILE LEFT$(S$,1)=" "
    S$=MID$(S$,2)
%END-WHILE
```

which translates to produce:

```
1 IF NOT (LEFT$(S$,1)=" ") THEN 4
2 S$=MID$(S$,2)
3 GOTO 1
4
```

UNTIL STRUCTURE

The %UNTIL structure is of the form:

```
%UNTIL conditional-statement
    sequence-of-statements
%END-UNTIL
```

The conditional-statement is evaluated at the end of the loop and the loop repeats "UNTIL" the condition is true. Thus the sequence-of-statements of a %UNTIL structure must be executed at least once.

```
/* UNTIL EXAMPLE

S=0:C=0
%UNTIL S>1
    C=C+1
    S=S+RND(1)
%END-UNTIL
```

which translates to produce:

```
1 S=0:C=0
2 C=C+1
3 S=S+RND(1)
4 IF NOT (S>1) THEN 2
5
```

IF-THEN-ELSE STRUCTURE

This structure is a combination of the IF, IF-ELSE, and CASE structures of other languages all rolled into one neat package with a multitude of forms as follows:

```
%IF conditional-statement
    sequence-of-statements
%END-IF

%IF conditional-statement
    sequence-of-statements-1
%ELSE
    sequence-of-statements-2
%END-IF

%IF conditional-statement-1
    sequence-of-statements-1
%ELSEIF conditional-statement-2
    sequence-of-statements-2
%ELSEIF conditional-statement-3
    sequence-of-statements-3
%ELSE
    sequence-of-statements-4
%END-IF
```

The following points should be noted carefully. The sequence of statements are mutually exclusive in all forms, i.e. only one of the many will be executed. There may be as many %ELSEIF constructs as desired in a single structure, but if the %ELSE construct is used it may only appear once, but it need not appear at all. The following form is valid:

```
%IF conditional-statement-1
    sequence-of-statements-1
%ELSEIF conditional-statement-2
    sequence-of-statements-2
```

```
%END-IF

/* IF-THEN-ELSE EXAMPLE

%IF A>0
    B=10
%ELSEIF A<0
    B=20
%ELSE
    B=30
%END-IF
```

which translates to produce:

```
1 IF NOT (A>0) THEN 4
2 B=10
3 GOTO 8
4 IF NOT (A<0) THEN 7
5 B=20
6 GOTO 8
7 B=30
8
```

NESTING OF IF, WHILE, AND UNTIL STRUCTURES

These structures may appear inside one another as desired to some indeterminant level of nesting. Since SBT doesn't keep track of level nesting as such, it is difficult to define a maximum limit. If you should encounter a stack overflow on S$(), i.e. S>50, then increase the dimensions of S$(E) and S$(E,1) to something greater than its initial setting of 50 and the problem should go away.

ERROR HANDLING ROUTINES

In the TRS-80 environment, control is given to the programmer to capture and handle errors when they occur. It will be noted by those who use these routines that SBT will not handle these routines since the statement must read 'ON ERROR GOTO'. Since SBT 'SOURCE' has no line numbers, this instruction has nowhere to go. Therefore, it cannot be used.

SBT implements another version of the %ON routine, and that is:

```
%ON ERROR
 sequence-of-statements
%END-ERROR
```

Care must be taken in coding this block. Since BASIC wants to see this statement early in the program, usually at the beginning, be careful not to use a %CALL statement in the sequence-of-statements block if the first %PROC has not been encountered or, if done this way, issue a %CALL to the main PROC after the %END-ERROR statement followed by a BASIC END. One other reminder - a CLEAR XXXX statement will reset the "ON ERROR GOTO" to 0!!

```
/* ON ERROR EXAMPLE

%PROC MAIN
    %CALL INITIALIZE
    %CALL MAJOR
    %CALL END-IT
    END
%END-PROC

%PROC INITIALIZE
    %ON ERROR
        %IF ERR/2+1=53
            PRINT "FILE ";F$;"NOT FOUND "
            RESUME
        %ELSE
            ON ERROR GOTO 0
        %END-IF
    %END-ERROR
%END-PROC
```

PROCEDURE ENVIRONMENT

This is probably by far the most difficult concept of the entire operation.

MAIN PROGRAM FILE

The program that SBT translates may reside in one or more disk files. When SBT is run it initiates a prompt and the user inputs the initial or first program file name. The file must be saved with extension '/SBS', but as instructed by the prompt the file name is entered without the extension. If you use the editor supplied with this program, files will automatically be saved with the correct extension saving typing. Anyway, SBT reads and translates this file, the first part of which must be the main program. (The main program may or may not be a procedure, as the user desires).

As the main program file is being translated, SBT stores all encountered procedure names in a table. Procedure names may be encountered in %CALL or %ON statements before the procedures are actually encountered for translation. SBT assumes that a procedure will be translated into a sequence of line numbers beginning at 500*name indes in procedure table, so GOSUB line numbers are generated accordingly. If a procedure is encountered during the translation of the main program file, it is translated with a sequence of line numbers as dictated by the procedure table. Also a key is set so SBT may later know that this procedure has been found and translated.

EXTERNAL PROCEDURES AND FILES

Upon completion of translating the main program file SBT selects the first procedure name in the table that has not been found and translated and assumes it to be in a disk file with the same name and an extension '/SBS'. SBT then opens this file and begins translation. This file must contain a procedure with the same name as the file name, but it may also contain other procedures, which will be translated as encountered. The procedures in this file need not be in any specific order but, it is good practice to have the first procedure the same as the file name.

Upon completion of processing this file, SBT again looks in the procedure table for the next unresolved procedure name and repeats the preceding process.

At present SBT is limited to 50 procedures but can be expanded by increasing the dimenstions of P$() and P(). The maximum value that may be used is 130 because of the scheme SBT uses for the generation of line numbers where procedures begin. The method

used also limits procedures to a maximum of 500 executable lines, but this is not considered to be restrictive.

NOT TO WORRY

SBT creates a BASIC program file with the main program file name and the extension '/BAS'. If you do a DOS list of this file before loading it into BASIC, you will find that the line numbers are out of order. SBT was written using one of the supposedly trivial features of BASIC, that when it loads an ASCII copy of a program, it checks line numbers and puts them in ascending order. This feature allowed SBT to be written as a one-pass translator and is also responsible for the flexibility of procedure placement in source files.

THE UNPLANNED BONUS

The BASIC program created by SBT is anything but structured. It makes an attempt to optimize the generated code by not generating any GOTO's which transfer to other GOTO's or RETURN's. A GOTO that would transfer control to a RETURN is substituted by a RETURN and a GOTO which would transfer control to another GOTO substitutes the line number of the second GOTO into the first. This feature significantly decreases the execution time of the result code.

VERSION INFORMATION

SBT has been implemented on a number of systems. Currently these include the HEATHKIT H8 under HEATH EXTENDED BASIC and MICROSOFT BASIC; TRS-80; OSI; and MEMORITE. Version #3.0 identified the first release for the TRS-80 and updates will be provided to registered users.

ONE WORD OF CAUTION

If you are using a program that requires the clearing of string space the CLEAR XXXX statement must appear as the first statement in the program. This should be coded before any other statement. If you inadvertently place a clear statement inside a subroutine, the program will terminate on an error when the RETURN is encountered.

RUNNING THE PROGRAM

From BASIC run "SBT". A file will be asked for that contains the program to be translated. Enter the required file name. The program will supply information to the user as to the file being used and the PROC being translated.

If you want to try SBT, use the text for SBT itself or the text for the editor. The resultant '/BAS' file will not destroy any functional programs.

After a program has been translated load the '/BAS' file. This will orient the line numbers correctly. Now save the program using a different file name. To conserve disk space, the '/BAS' file may be killed.

TEXT EDITOR

The supplied text editor is written in SBT. It is designed to be used in conjunction with the SBT program since all files are saved with the required extension '/SBS'. Therefore, if used for other purposes, the extension must be changed or the user must remember it is there.

Line editing is limited to the replacement of a line, but additional changes could be implemented with simple routines by the users. The program is self documenting and very easy to run.

ANOTHER WORD OF CAUTION

In all TRS-80 programs which use string space, TRS-80 BASIC will occasionally need to consolidate unused string space. This will appear as a "FREEZE UP" of the equipment for as long as a minute or more. There is nothing wrong with your machine and nothing can be done to remedy the situation except be patient.