TINY

A Loop Restructuring Research Tool

Michael Wolfe
Oregon Graduate Institute of Science and Technology
19600 NW von Neumann Drive
Beaverton, OR 97006 USA
tel: 503-690-1153
fax: 503-690-1029
email: mwolfe@cse.ogi.edu

December 1990

1. Keys to Remember

TINY is designed around a character menu interface (for the nonce); to choose a menu item, move the menu bar between menu items by using the space or backspace key, then typing the return key when the desired choice is highlighted. A quicker method is to type the leading character of the desired choice; uppercase or lowercase may be used.

Three keys always (almost) have special meaning.

Q at any menu should quit TINY and take you back to the shell prompt.

X at any menu should escape or exit out of this menu. Alternatively, the 'escape' key has the same effect.

^L A control-L will refresh the screen. Occasional junk gets by the Curses package and will not be corrected by ^L.

2. Starting TINY

The screen display shows positions via highlighting, though the screen dumps in this documentation don't show that highlighting. To save space, multiple blank lines in the screen dumps are deleted.

When starting up TINY with no file argument, you get the screen:

```
Tiny Tool [as of December 1990]
*Browse  File   Parse  Restor  System  Trans  Write  Msgs   Quit
```

The first 22 lines (on a standard 24 line display) are the main window; the next to last line is a message line, and the bottom line is the current menu. The message would give the file name of the current program, if one was chosen. The main menu has 8 choices; choices can be made from any menu by using the space bar and backspace key to move the menu position (shown with highlighting and the asterisk) to the desired choice and hitting the return key, or (faster) by typing the leading character of the choice. Here, for instance, typing 'Q' will quit TINY. All menus in TINY have the Msgs, Quit and Xcape options (except for the main menu, which has no Xcape). Choosing Quit from any menu will quit TINY completely and immediately (in the absence of bugs). Choosing Msgs from any menu will display the most recent saved messages in the main window. Choosing Xcape from any menu returns to the previous menu; Xcape may also be chosen by typing the 'Escape' key on your keyboard.

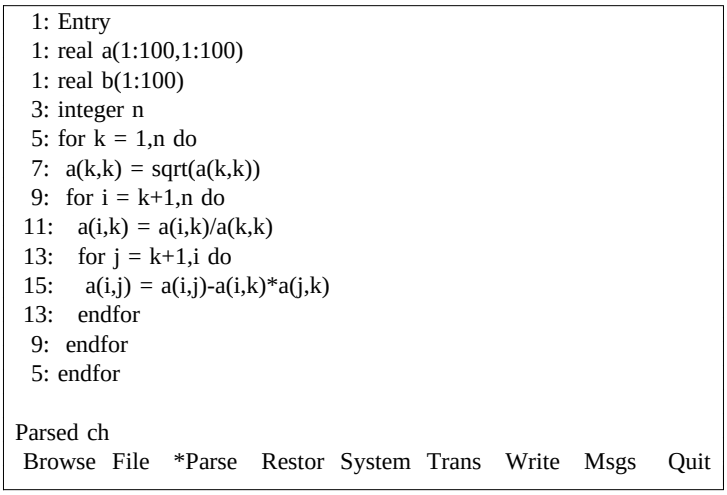
To read in a program from a file after starting up TINY, we choose the Parse option; TINY then asks for the name of the file we wish to parse:

```
Tiny Tool [as of December 1990]
File:
```

We then respond with the name of a file name:

```
Tiny Tool [as of December 1990]
File: ch
```

and this file is read in and displayed:



A quicker way to start up TINY on a file is to give the file name on the command line:

```
% t ch
```

which will start up by first parsing and displaying the program.

3. Transformations

TINY currently implements 8 elementary loop restructuring transformations, with plans for several more to be added. The list is:

bumping

circulation

distribution

interchanging

negation (or reversal)

parallelization

skewing

vectorization (simple)

Each of these is explained here with examples.

3.1. Loop Bumping

Bumping is simple adjusting loop lower and upper limits by adding (or subtracting) a constant integer. This is occasionally used to make lower or upper limits of two loops match exactly, for instance to satisfy the strict requirements of loop interchanging for non-tightly nested loops. A more sophisticated tool would notice the need for bumping automatically. An example is given under loop interchanging.

3.2. Loop Circulation

Circulation is a generalization of loop interchanging; it is equivalent to interchanging a loop inside of (or outside of) multiple inner (outer) loops in a single step. For example, take the smoothing program:

```
  1: Entry
  1: real a(1:100,1:100,1:100)
  3: integer n
  5: for k = 2,n do
  7:  for i = 2,n do
  9:   for j = 2,n do
 11:    a(k,i,j) = a(k,i-1,j)+a(k,i,j-1)+a(k,i,j+1)+a(k,i+1,j)+a(k-1,i,j)+a(k+1,
i,j)
  9:   endfor
  7:  endfor
  5: endfor

Parsed wave3a
 Browse  File   *Parse   Restor  System  Trans   Write   Msgs    Quit
```

The outermost 'k' loop can be 'innermosted', or interchanged to the innermost position, or 'intercirculated' inside of the 'j' loop in one step by choosing the Circ menu item, then choosing the circulate inside of the 'j' loop:

```
  1: Entry
  1: real a(1:100,1:100,1:100)
  3: integer n
  7: for i = 2,n do
  9:  for j = 2,n do
  5:   for k = 2,n do
 11:    a(k,i,j) = a(k,i-1,j)+a(k,i,j-1)+a(k,i,j+1)+a(k,i+1,j)+a(k-1,i,j)+a(k+1,
i,j)
  5:   endfor
  9:  endfor
  7: endfor

Intercirculating loop k inside of j
 Browse  DD      Loop   *Restr   See     Undo    Var     Msgs    Quit    Xcape
```

In contrast, outercirculating the 'j' loop (of the original program) to the outermost position, in one step, would produce:

```
  1: Entry
  1: real a(1:100,1:100,1:100)
  3: integer n
  9: for j = 2,n do
  5:  for k = 2,n do
  7:   for i = 2,n do
 11:    a(k,i,j) = a(k,i-1,j)+a(k,i,j-1)+a(k,i,j+1)+a(k,i+1,j)+a(k-1,i,j)+a(k+1,
i,j)
  7:   endfor
  5:  endfor
  9: endfor

Outercirculating loop j outside of k
 Browse  DD      Loop   *Restr   See     Undo    Var     Msgs    Quit    Xcape
```

The data dependence tests for intercirculation and outercirculation are explained in two recent papers: Utpal Banerjee, "A Theory of Loop Permutations" (which appears in the Springer-Verlag monograph *Languages and Compilers for Parallel Computing*, Gelernter, Nicolau and Padua (eds.), 1990) and Michael Wolfe, "Data Dependence and Program Restructuring" (to appear in *The Journal of Supercomputing* in late 1990 or early 1991). After these transformations, dependence direction or distance vectors are modified to account for the new loop ordering.

3.3. Loop Distribution

Loop distribution is a well-known transformation which is often used to distribute an outer loop around non-tightly nested inner loops. As an example, we use a Cholesky decomposition program:

```
  1: Entry
  1: real a(1:100,1:100)
  1: real b(1:100)
  3: integer n
  5: for k = 1,n do
  7:  a(k,k) = sqrt(a(k,k))
  9:  for i = k+1,n do
 11:   a(i,k) = a(i,k)/a(k,k)
 13:   for j = k+1,i do
 15:    a(i,j) = a(i,j)-a(i,k)*a(j,k)
 13:   endfor
  9:  endfor
  5: endfor

Parsed ch
 Bump   Circ  *Dist   Inter  Neg    Par    Skew   Vec    Msgs   Quit   Xcape
```

After distribution, the loop looks like:

```
  1: Entry
  1: real a(1:100,1:100)
  1: real b(1:100)
  3: integer n
  5: for k = 1,n do
  7:  a(k,k) = sqrt(a(k,k))
  9:  for i = k+1,n do
 11:   a(i,k) = a(i,k)/a(k,k)
  9:  endfor
  9:  for i = k+1,n do
 13:   for j = k+1,i do
 15:    a(i,j) = a(i,j)-a(i,k)*a(j,k)
 13:   endfor
  9:  endfor
  5: endfor

Distributing loop i
 Browse  DD     Loop   *Restr  See    Undo   Var    Msgs   Quit   Xcape
```

The dependence test for loop distribution involves finding dependence cycles, and keeping all dependence cycles in a single distributed loop.

3.4. Loop Interchanging

In TINY, choosing Interchanging will interchange the current loop with its immediate outer loop. Taking the distributed loops from the loop distribution example, we can move to the 'j' loop and interchange it with the 'i' loop to get:

```
  1: Entry
  1: real a(1:100,1:100)
  1: real b(1:100)
  3: integer n
  5: for k = 1,n do
  7:  a(k,k) = sqrt(a(k,k))
  9:  for i = k+1,n do
 11:   a(i,k) = a(i,k)/a(k,k)
  9:  endfor
 13:  for j = k+1,n do
  9:   for i = max(k+1,j),n do
 15:    a(i,j) = a(i,j)-a(i,k)*a(j,k)
  9:   endfor
 13:  endfor
  5: endfor

Interchanging loops i and j
 Browse  DD      Loop   *Restr   See     Undo     Var     Msgs     Quit     Xcape
```

Note how the triangular loop limits have been modified; some future version of TINY will simplify the loop limits of the 'i' loop to be simply 'j,n', eliminating the 'max' when unnecessary. Loop interchanging is legal if there are no (<,>) dependence relations, and the dependence direction and distance vector elements for the interchanged loops are also interchanged.

Non-tightly nested loops can also be interchanged. For instance, we can take the above example (after interchanging the 'i' and 'j' loops) and choose to interchange the 'j' loop outwards. This produces the result:

```
  1: Entry
  1: real a(1:100,1:100)
  1: real b(1:100)
  3: integer n
 13: for j = 1,n do
  5:  for k = 1,j-1 do
  9:   for i = max(k+1,j),n do
 15:    a(i,j) = a(i,j)-a(i,k)*a(j,k)
  9:   endfor
  5:  endfor
  7:  a(j,j) = sqrt(a(j,j))
  9:  for i = j+1,n do
 11:   a(i,j) = a(i,j)/a(j,j)
  9:  endfor
 13: endfor

Interchanging loops k and j
 Browse  DD      Loop   *Restr   See     Undo     Var     Msgs     Quit     Xcape
```

Note that lines 4 through 6 have been moved from above the inner loop to below the inner loop, and that the loop index 'k' has been replaced by 'j'. Currently, interchanging nontightly nested loops requires the loop limits to be perfectly square or triangular; while other loops limits could be handled by adding 'if' statements or adjusting the loop limits automatically, this current restriction sometimes requires 'bumping' a loop limit. For instance, suppose we take the original Cholesky decomposition and instead of distributing the 'i' loop, we interchange the 'i' loop outwards, producing:

```
  1: Entry
  1: real a(1:100,1:100)
  1: real b(1:100)
  3: integer n
  9: for i = 1,n do
  5:  for k = 1,i-1 do
 11:   a(i,k) = a(i,k)/a(k,k)
 13:   for j = k+1,i do
 15:    a(i,j) = a(i,j)-a(i,k)*a(j,k)
 13:   endfor
  5:  endfor
  7:  a(i,i) = sqrt(a(i,i))
  9: endfor

Interchanging loops k and i
 Browse  DD     Loop   *Restr   See     Undo    Var     Msgs    Quit    Xcape
```

Now suppose we want to interchange the 'j' and 'k' loops. If we try to distribute the 'k' loop, we get the message:

```
Can't distribute
 Browse  DD     Loop   *Restr   See     Undo    Var     Msgs    Quit    Xcape
```

since there is a dependence cycle involving lines 6 and 8. If we try to interchange the 'j' loop outwards, we get the message:

```
Imperfect loop interchanging requires square/triangular loops
 Browse  DD     Loop   *Restr   See     Undo    Var     Msgs    Quit    Xcape
```

The loop limits can be made triangular by subtracting one from the 'j' loop limits (or adding one to the 'k' loop limits). We can choose to bump the 'j' loop limits by choosing the 'Bump' menu item at the 'j' loop:

```
  1: Entry
  1: real a(1:100,1:100)
  1: real b(1:100)
  3: integer n
  9: for i = 1,n do
  5:  for k = 1,i-1 do
 11:   a(i,k) = a(i,k)/a(k,k)
 13:   for j = k+1,i do
 15:    a(i,j) = a(i,j)-a(i,k)*a(j,k)
 13:   endfor
  5:  endfor
  7:  a(i,i) = sqrt(a(i,i))
  9: endfor

Imperfect loop interchanging requires square/triangular loops
*Bump   Circ   Dist   Inter  Neg    Par    Skew   Vec    Msgs   Quit   Xcape
```

TINY then asks what constant to add to the loop limits:

```
Interchanging loops k and i
Bump by how much:
```

We answer with '-1', which then produces the program:

```
  1: Entry
  1: real a(1:100,1:100)
  1: real b(1:100)
  3: integer n
  9: for i = 1,n do
  5:  for k = 1,i-1 do
 11:   a(i,k) = a(i,k)/a(k,k)
 13:   for j = k+1-1,i-1 do
 15:    a(i,j+1) = a(i,j+1)-a(i,k)*a(j+1,k)
 13:   endfor
  5:  endfor
  7:  a(i,i) = sqrt(a(i,i))
  9: endfor

Bump loop j by -1
 Browse  DD      Loop   *Restr   See     Undo    Var     Msgs    Quit    Xcape
```

Notice that within the loop, 'j' is replaced by 'j+1'; (also notice that no expression simplification is done). Now the two loops can be successfully interchanged:

```
  1: Entry
  1: real a(1:100,1:100)
  1: real b(1:100)
  3: integer n
  9: for i = 1,n do
 13:  for j = 1,i-1 do
 11:   a(i,j) = a(i,j)/a(j,j)
  5:   for k = 1,j do
 15:    a(i,j+1) = a(i,j+1)-a(i,k)*a(j+1,k)
  5:   endfor
 13:  endfor
  7:  a(i,i) = sqrt(a(i,i))
  9: endfor

Interchanging loops k and j
 Browse  DD      Loop   *Restr   See     Undo    Var     Msgs    Quit    Xcape
```

3.5. Loop Negation

Loop negation (also called loop reversal) involves running the loop backward. TINY shows a negated loop by negating and switching the lower and upper limits of a loop, and negating the loop index within the body of the loop. Negating the 'j' loop in the first interchanging example gives:

```
  1: Entry
  1: real a(1:100,1:100)
  1: real b(1:100)
  3: integer n
  5: for k = 1,n do
  7:  a(k,k) = sqrt(a(k,k))
  9:  for i = k+1,n do
 11:   a(i,k) = a(i,k)/a(k,k)
  9:  endfor
 13:  for j = -n,-(k+1) do
  9:   for i = max(k+1,-j),n do
 15:    a(i,-j) = a(i,-j)-a(i,k)*a(-j,k)
  9:   endfor
 13:  endfor
  5: endfor

Negate loop j
 Browse  DD      Loop   *Restr   See     Undo     Var     Msgs     Quit     Xcape
```

Loop negation is legal if the loop carries no dependence relations, and the dependence graph is modified to negate any dependence distance or directions.

3.6. Loop Parallelization

A loop can be parallelized as long as it carries no data dependence relations. Unlike other transformations, this cannot be 'Undone', nor will it show up in the 'Restore' display. In the previous example, the loops that can be parallelized are:

```
  1: Entry
  1: real a(1:100,1:100)
  1: real b(1:100)
  3: integer n
  5: for k = 1,n do
  7:  a(k,k) = sqrt(a(k,k))
  9:  doall i = k+1,n do
 11:   a(i,k) = a(i,k)/a(k,k)
  9:  endfor
 13:  doall j = -n,-(k+1) do
  9:   doall i = max(k+1,-j),n do
 15:    a(i,-j) = a(i,-j)-a(i,k)*a(-j,k)
  9:   endfor
 13:  endfor
  5: endfor

Parallelize loop i
 Browse  DD      Loop   *Restr   See     Undo     Var     Msgs     Quit     Xcape
```

3.7. Loop Skewing

Forward (reverse) loop skewing involves adding (subtracting) an outer loop index to the lower and upper limits for an inner loop. Loop skewing is always legal, but modifies the direction and distance vectors by adding the outer loop elements to the inner loop element. Thus, a (<,=) direction is changed to a (<,<) direction; this means that after loop interchanging, the dependence relation will be carried by the outer loop, allowing parallel execution of the inner loop. This is useful in a smoothing algorithm, as in the circulation example. Forward skewing the inner 'j' loop with respect to the 'k' loop gives:

```
  1: Entry
  1: real a(1:100,1:100,1:100)
  3: integer n
  5: for k = 2,n do
  7:  for i = 2,n do
  9:   for j = 2+k,n+k do
 11:    a(k,i,j-k) = a(k,i-1,j-k)+a(k,i,j-k-1)+a(k,i,j-k+1)+a(k,i+1,j-k)+a(k-1,i
,j-k)+a(k+1,i,j-k)
  9:   endfor
  7:  endfor
  5: endfor

Forward skew loop j with respect to k
 Browse  DD      Loop   *Restr   See     Undo    Var     Msgs    Quit    Xcape
```

Forward skewing the 'j' loop again with respect to the 'i' loop gives:

```
  1: Entry
  1: real a(1:100,1:100,1:100)
  3: integer n
  5: for k = 2,n do
  7:  for i = 2,n do
  9:   for j = 2+k+i,n+k+i do
 11:    a(k,i,j-i-k) = a(k,i-1,j-i-k)+a(k,i,j-i-k-1)+a(k,i,j-i-k+1)+a(k,i+1,j-i-
k)+a(k-1,i,j-i-k)+a(k+1,i,j-i-k)
  9:   endfor
  7:  endfor
  5: endfor

Forward skew loop j with respect to i
 Browse  DD      Loop   *Restr   See     Undo    Var     Msgs    Quit    Xcape
```

Now we see the advantage of loop skewing, by circulating the 'j' loop all the way outside:

```
  1: Entry
  1: real a(1:100,1:100,1:100)
  3: integer n
  9: for j = 2+2+2,n+n+n do
  5:  for k = max(2,j-(n+n)),min(n,j-(2+2)) do
  7:   for i = max(2,j-(n+k)),min(n,j-(2+k)) do
 11:    a(k,i,j-i-k) = a(k,i-1,j-i-k)+a(k,i,j-i-k-1)+a(k,i,j-i-k+1)+a(k,i+1,j-i-
k)+a(k-1,i,j-i-k)+a(k+1,i,j-i-k)
  7:   endfor
  5:  endfor
  9: endfor

Outercirculating loop j outside of k
 Browse  DD      Loop   *Restr   See     Undo    Var     Msgs    Quit    Xcape
```

And now we can parallelize the inner 'k' and 'i' loops:

```
  1: Entry
  1: real a(1:100,1:100,1:100)
  3: integer n
  9: for j = 2+2+2,n+n+n do
  5:  doall k = max(2,j-(n+n)),min(n,j-(2+2)) do
  7:   doall i = max(2,j-(n+k)),min(n,j-(2+k)) do
 11:    a(k,i,j-i-k) = a(k,i-1,j-i-k)+a(k,i,j-i-k-1)+a(k,i,j-i-k+1)+a(k,i+1,j-i-
k)+a(k-1,i,j-i-k)+a(k+1,i,j-i-k)
  7:   endfor
  5:  endfor
  9: endfor

Parallelize loop i
 Browse  DD      Loop   *Restr   See     Undo    Var     Msgs    Quit    Xcape
```

3.8. Vectorization

Vectorization is legal when there are no loop-carried dependence cycles in the inner loop. Vectorization in TINY works only for trivial cases, as it does not even recognize simple reduction operators; non-inner loops cannot be vectorized, though vectorization within an outer parallel loop is legal. Vectorization does not change the dependence graph at all. As an example, taking the program from the Skewing section, we can vectorize the inner loop to get:

```
  1: Entry
  1: real a(1:100,1:100,1:100)
  3: integer n
  9: for j = 2+2+2,n+n+n do
  5:  doall k = max(2,j-(n+n)),min(n,j-(2+2)) do
  7:   forall i = max(2,j-(n+k)),min(n,j-(2+k)) do
 11:    a(k,i,j-i-k) = a(k,i-1,j-i-k)+a(k,i,j-i-k-1)+a(k,i,j-i-k+1)+a(k,i+1,j-i-
k)+a(k-1,i,j-i-k)+a(k+1,i,j-i-k)
  7:   endfor
  5:  endfor
  9: endfor

Parallelize loop i
 Browse  DD      Loop   *Restr   See     Undo    Var     Msgs    Quit    Xcape
```

4. TINY Language

The TINY language is very simple. It comprises scalar and array variables, loops, assignments and block-structured IF statements. The language BNF is:

```
program          ::= stlist

stlist       ::= statement [';' statement]...

statement    ::= integerdecl
             ::= realdecl
             ::= constdecl
             ::= assignment
             ::= loop
             ::= if


constdecl    ::= 'const' constassignment [',' constassignment]...

constassignment   ::= ID '=' expression

integerdecl  ::= 'integer' vardecllist
realdecl     ::= 'real' vardecllist

vardecllist  ::= vardecl [',' vardecl]...

vardecl          ::= ID
             ::= ID '(' [expression ':'] expression
                 [',' [expression ':'] expression]... ')'


assignment   ::= lhs '=' expression

lhs          ::= ID
             ::= ID '(' expression [',' expression]... ')'

loop         ::= ['for' | 'doall']
                 ID '=' expression ['to' | ',' | ':'] expression
                 [ ['by' | ',' | ':'] expression]
                 'do' stlist 'endfor'

if           ::= 'if' expression 'then' stlist ['else' stlist] 'endif'
```

```
expression  ::= ID
            ::= ID '(' expression [',' expression]... ')'
            ::= INTCONST
            ::= FLOATCONST
            ::= expression '+' expression
            ::= expression '-' expression
            ::= expression '*' expression
            ::= expression '/' expression
            ::= expression '**' expression
            ::= '-' expression
            ::= '+' expression
            ::= '(' expression ')'
            ::= expression '<' expression
            ::= expression '<=' expression
            ::= expression '=' expression
            ::= expression '<>' expression
            ::= expression '>' expression
            ::= expression '>=' expression
            ::= expression 'mod' expression
            ::= expression 'max' expression
            ::= expression 'min' expression
            ::= 'sqrt' '(' expression ')'
            ::= 'floor' '(' expression '/' expression ')'
            ::= 'ceiling' '(' expression '/' expression ')'
            ::= 'max' '(' expression [',' expression]... ')'
            ::= 'min' '(' expression [',' expression]... ')'
```

5. Sample Session

Start out by firing up TINY with the command line:

```
% t
```

TINY will return with the display:

```
Tiny Tool [as of December 1990]
*Browse  File    Parse   Restor  System  Trans   Write   Msgs    Quit
```

Go to the File menu by typing 'f', getting the list of files:

```
/ogc1/staff/mwolfe/tiny
Announcement  doc/          fix/          source/       test/
dif           doc.log       make.out      t*            todo

Tiny Tool [as of December 1990]
*Down    Edit    Newdir  Redo    Sh      Up      Msgs    Quit    Xcape
```

To move to the 'test' subdirectory, choose 'Down' by typing 'd', getting the prompt:

```
Tiny Tool [as of December 1990]
Directory:
```

Respond by typing 'test', following by the return key; TINY then displays the files in the 'test' subdirectory:

```
/ogc1/staff/mwolfe/tiny/test
ave3        dd.4        doc2.log    lu          parenb      wave2
ch          dd.t1       dynamic     ludecomp    rev         wave3
dd.1        dd.t2       example.1   paren       rn1         wave3a
dd.2        dd.t3       example.2   paren2      wave        wave3b
dd.3        doc.log     example.3   parena      wave.8wa

Tiny Tool [as of December 1990]
*Down    Edit    Newdir  Redo    Sh      Up      Msgs    Quit    Xcape
```

To check the contents of a file, type 'E' to edit the file; TINY prompts for the file name, so a response of 'ch' gets:

```
real a(100,100), b(100)
integer n
for k = 1 to n do
a(k,k) = sqrt(a(k,k))
for i = k+1 to n do
a(i,k) = a(i,k)/a(k,k)
for j = k+1 to i do
a(i,j) = a(i,j)-a(i,k)*a(j,k)
endfor
endfor
endfor
~
~
"ch" 11 lines, 191 characters
```

Exiting the editor returns to the same File menu. Return to the Main menu by typing the 'escape' key:

```
Tiny Tool [as of December 1990]
 Browse *File    Parse   Restor  System  Trans   Write   Msgs    Quit
```

Notice that the default menu selection when returning from a submenu is left at the most recent choice; to enter the 'File' menu again, type 'return'. To parse one of those files, so type 'p':

```
Tiny Tool [as of December 1990]
File:
```

Respond with the file name 'decomp.cholesky', followed by 'return':

```
  1: Entry
  1: real a(1:100,1:100)
  1: real b(1:100)
  2: integer n
  3: for k = 1,n do
  4:  a(k,k) = sqrt(a(k,k))
  5:  for i = k+1,n do
  6:   a(i,k) = a(i,k)/a(k,k)
  7:   for j = k+1,i do
  8:    a(i,j) = a(i,j)-a(i,k)*a(j,k)
  7:   endfor
  5:  endfor
  3: endfor

Parsed ch
 Browse  File   *Parse   Restor  System  Trans   Write   Msgs    Quit
```

To explore some interactive restructuring, go to the Browse menu by typing 'B':

```
  1: Entry
  1: real a(1:100,1:100)
  1: real b(1:100)
  2: integer n
  3: for k = 1,n do
  4:  a(k,k) = sqrt(a(k,k))
  5:  for i = k+1,n do
  6:   a(i,k) = a(i,k)/a(k,k)
  7:   for j = k+1,i do
  8:    a(i,j) = a(i,j)-a(i,k)*a(j,k)
  7:   endfor
  5:  endfor
  3: endfor

Parsed ch
*Browse  DD     Loop    Restr   See     Undo    Var     Msgs    Quit    Xcape
```

Notice that when entering a new menu, the default menu item is always the first choice, not necessarily the one most often chosen. Notice that in the Browse menu, there is always a current position in the file; initially the current position is the 'Entry' node, at the top of the file. To look at the data dependence relations in the program, enter the DD Browse Menu by typing 'D':

```
  1: Entry
  1: real a(1:100,1:100)
  1: real b(1:100)
  2: integer n
  3: for k = 1,n do
  4:  a(k,k) = sqrt(a(k,k))
  5:  for i = k+1,n do
  6:   a(i,k) = a(i,k)/a(k,k)
  7:   for j = k+1,i do
  8:    a(i,j) = a(i,j)-a(i,k)*a(j,k)
  7:   endfor
  5:  endfor
  3: endfor

No DD successors.
*Cycle  Goto  Next  Pred  Succ  Var   Write  Msgs  Quit  Xcape
```

The message tells that the current position has no DD successors; this is not surprising since the current position is the Entry node. (Some future version of TINY may link upwardly-exposed uses to the Entry node.) Two ways to browse the DD graph are given. The most useful way is to cycle through all the dependences in the program by using the Cycle menu choice; this moves the current position to the next variable reference that has data dependence successors and displays that dependence relation on the message line. It will display each dependence successor from that variable reference, then move on to the next reference. Typing 'C' (or just return, since Cycle is the default menu item) gives the display:

```
  1: Entry
  1: real a(1:100,1:100)
  1: real b(1:100)
  2: integer n
  3: for k = 1,n do
  4:  a(k,k) = sqrt(a(k,k))
  5:  for i = k+1,n do
  6:   a(i,k) = a(i,k)/a(k,k)
  7:   for j = k+1,i do
  8:    a(i,j) = a(i,j)-a(i,k)*a(j,k)
  7:   endfor
  5:  endfor
  3: endfor

flow dependence 4: --> 6:(=)  (0)
*Cycle  Goto  Next  Pred  Succ  Var   Write  Msgs  Quit  Xcape
```

Notice the highlighting in lines 4 and 6 showing the variable references in question. The message line shows the kind of dependence (flow, anti, output), the line numbers involved, the direction vector and the distance vector (unknown directions or distances appear as "*"). Typing 'C' again gives the next dependence relation:

```
  1: Entry
  1: real a(1:100,1:100)
  1: real b(1:100)
  2: integer n
  3: for k = 1,n do
  4:  a(k,k) = sqrt(a(k,k))
  5:  for i = k+1,n do
  6:   a(i,k) = a(i,k)/a(k,k)
  7:   for j = k+1,i do
  8:    a(i,j) = a(i,j)-a(i,k)*a(j,k)
  7:   endfor
  5:  endfor
  3: endfor

flow dependence 6: --> 8:(=,=)  (0,0)
*Cycle  Goto  Next  Pred  Succ  Var  Write  Msgs  Quit  Xcape
```

and so on.

The other method to browse the dependence relations interactively is to use the Var menu option to move the current position to the next variable reference. The first predecessor or successor of that variable reference may be displayed by using the Pred or Succ menu items, and Next will show the next predecessor or successor, as appropriate. Goto will move the current position to the dependence predecessor or successor reference shown on the message line.

In any case, all these messages are saved in the message buffer, and may be displayed by typing the 'M' key:

```
Parsed ch
No DD successors.
flow dependence 4: --> 6:(=)  (0)
flow dependence 6: --> 8:(=,=)  (0,0)
flow dependence 6: --> 8:(=,<=)  (0,*)
anti dependence 8: --> 4:(<)  (*)
anti dependence 8: --> 6:(<,=)  (*,0)
anti dependence 8: --> 8:(<,=,=)  (*,0,0)
flow dependence 8: --> 4:(<)  (*)
output dependence 8: --> 4:(<)  (*)
flow dependence 8: --> 6:(<,=)  (*,0)
flow dependence 8: --> 6:(<,<)  (*,*)
output dependence 8: --> 6:(<,=)  (*,0)
flow dependence 8: --> 8:(<,=,=)  (*,0,0)
flow dependence 8: --> 8:(<,=,<)  (*,0,*)
flow dependence 8: --> 8:(<,<=,<)  (*,*,*)
output dependence 8: --> 8:(<,=,=)  (*,0,0)
No DD successors.

No DD successors.
*Xcape  Quit
```

The dependence relations can all be written to a file by choosing the Write menu option; TINY prompts for the file to which to write the dependence information. The file for this program would look like:

```
flow dependence:  4 -->  6 (=)  (0)     a(k,k) --> a(k,k)
flow dependence:  6 -->  8 (=,=)  (0,0)       a(i,k) --> a(i,k)
flow dependence:  6 -->  8 (=,<=)  (0,*)      a(i,k) --> a(j,k)
anti dependence:  8 -->  4  (<)  (*)     a(i,j) --> a(k,k)
anti dependence:  8 -->  6  (<,=)  (*,0)       a(i,j) --> a(i,k)
anti dependence:  8 -->  8  (<,=,=)  (*,0,0)  a(i,j) --> a(i,j)
flow dependence:  8 -->  4 (<)  (*)     a(i,j) --> a(k,k)
output dependence:  8 -->  4     (<)  (*)     a(i,j) --> a(k,k)
flow dependence:  8 -->  6 (<,=)  (*,0)       a(i,j) --> a(i,k)
flow dependence:  8 -->  6 (<,<)  (*,*)       a(i,j) --> a(k,k)
output dependence:  8 -->  6     (<,=)  (*,0)       a(i,j) --> a(i,k)
flow dependence:  8 -->  8 (<,=,=)  (*,0,0)  a(i,j) --> a(i,j)
flow dependence:  8 -->  8 (<,=,<)  (*,0,*)  a(i,j) --> a(i,k)
flow dependence:  8 -->  8 (<,<=,<)  (*,*,*)a(i,j) --> a(j,k)
output dependence:  8 -->  8     (<,=,=)  (*,0,0)  a(i,j) --> a(i,j)
```

Now to do some transformations. Type 'X' from the DD Browse Menu to get to the Browse menu:

```
  1: Entry
  1: real a(1:100,1:100)
  1: real b(1:100)
  2: integer n
  3: for k = 1,n do
  4:  a(k,k) = sqrt(a(k,k))
  5:  for i = k+1,n do
  6:   a(i,k) = a(i,k)/a(k,k)
  7:   for j = k+1,i do
  8:    a(i,j) = a(i,j)-a(i,k)*a(j,k)
  7:   endfor
  5:  endfor
  3: endfor

No DD successors.
 Browse *DD      Loop    Restr   See     Undo    Var     Msgs    Quit    Xcape
```

Currently the only transformations supported are loop transformations. To perform a loop transformation, move the current position to the loop to be transformed. Typing 'L' twice will move the current position from Entry to the first 'for' loop, then to the second 'for' loop:

```
  1: Entry
  1: real a(1:100,1:100)
  1: real b(1:100)
  2: integer n
  3: for k = 1,n do
  4:  a(k,k) = sqrt(a(k,k))
  5:  for i = k+1,n do
  6:   a(i,k) = a(i,k)/a(k,k)
  7:   for j = k+1,i do
  8:    a(i,j) = a(i,j)-a(i,k)*a(j,k)
  7:   endfor
  5:  endfor
  3: endfor

No DD successors.
 Browse  DD     *Loop    Restr   See     Undo    Var     Msgs    Quit    Xcape
```

Choose the Restructure menu by typing 'R':

```
  1: Entry
  1: real a(1:100,1:100)
  1: real b(1:100)
  2: integer n
  3: for k = 1,n do
  4:  a(k,k) = sqrt(a(k,k))
  5:  for i = k+1,n do
  6:   a(i,k) = a(i,k)/a(k,k)
  7:   for j = k+1,i do
  8:    a(i,j) = a(i,j)-a(i,k)*a(j,k)
  7:   endfor
  5:  endfor
  3: endfor

No DD successors.
*Bump   Circ   Dist   Inter  Neg    Par    Skew   Vec    Msgs   Quit   Xcape
```

The choices are loop bumping, loop circulation (generalized loop interchanging for tightly nested loops), loop distribution, loop interchanging (for tightly or non-tightly nested loops), loop negation, loop parallelization, and loop skewing (along with the standard MQX menu choices). Distribute loop 'i' by typing 'D':

```
  1: Entry
  1: real a(1:100,1:100)
  1: real b(1:100)
  2: integer n
  3: for k = 1,n do
  4:  a(k,k) = sqrt(a(k,k))
  5:  for i = k+1,n do
  6:   a(i,k) = a(i,k)/a(k,k)
  5:  endfor
  5:  for i = k+1,n do
  7:   for j = k+1,i do
  8:    a(i,j) = a(i,j)-a(i,k)*a(j,k)
  7:   endfor
  5:  endfor
  3: endfor

Distributing loop i
 Browse  DD     Loop   *Restr   See    Undo   Var    Msgs   Quit   Xcape
```

Now type 'L' to move to the 'j' loop, and go to the restructuring menu again:

```
  1: Entry
  1: real a(1:100,1:100)
  1: real b(1:100)
  2: integer n
  3: for k = 1,n do
  4:  a(k,k) = sqrt(a(k,k))
  5:  for i = k+1,n do
  6:   a(i,k) = a(i,k)/a(k,k)
  5:  endfor
  5:  for i = k+1,n do
  7:   for j = k+1,i do
  8:    a(i,j) = a(i,j)-a(i,k)*a(j,k)
  7:   endfor
  5:  endfor
  3: endfor

Distributing loop i
*Bump   Circ   Dist   Inter  Neg    Par    Skew   Vec    Msgs   Quit   Xcape
```

This time, let's see what happens when the 'j' loop is negated by typing 'N':

```
  1: Entry
  1: real a(1:100,1:100)
  1: real b(1:100)
  2: integer n
  3: for k = 1,n do
  4:  a(k,k) = sqrt(a(k,k))
  5:  for i = k+1,n do
  6:   a(i,k) = a(i,k)/a(k,k)
  5:  endfor
  5:  for i = k+1,n do
  7:   for j = -i,-(k+1) do
  8:    a(i,-j) = a(i,-j)-a(i,k)*a(-j,k)
  7:   endfor
  5:  endfor
  3: endfor

Negate loop j
 Browse  DD      Loop   *Restr   See     Undo    Var     Msgs    Quit    Xcape
```

This negates and switches the loop limits, and negates the index variable inside the range of the loop. It also affects the dependence relations of the loop. But if this turns out to be uninteresting, and the original version of the program with the forward loop is desired. One could re-negate the loop, but it is simpler to undo the negation by typing 'U' for Undo:

```
  1: Entry
  1: real a(1:100,1:100)
  1: real b(1:100)
  2: integer n
  3: for k = 1,n do
  4:  a(k,k) = sqrt(a(k,k))
  5:  for i = k+1,n do
  6:   a(i,k) = a(i,k)/a(k,k)
  5:  endfor
  5:  for i = k+1,n do
  7:   for j = k+1,i do
  8:    a(i,j) = a(i,j)-a(i,k)*a(j,k)
  7:   endfor
  5:  endfor
  3: endfor

Negate loop j
 Browse  DD      Loop    Restr   See    *Undo    Var     Msgs    Quit    Xcape
```

Notice that the current position is moved back to the Entry node. Again move to the 'j' loop (typing 'L') and go to the Restructure menu, choosing instead the Intchg (interchange) menu item. This interchanges the 'i' and 'j' loops:

```
  1: Entry
  1: real a(1:100,1:100)
  1: real b(1:100)
  2: integer n
  3: for k = 1,n do
  4:  a(k,k) = sqrt(a(k,k))
  5:  for i = k+1,n do
  6:   a(i,k) = a(i,k)/a(k,k)
  5:  endfor
  7:  for j = k+1,n do
  5:   for i = max(k+1,j),n do
  8:    a(i,j) = a(i,j)-a(i,k)*a(j,k)
  5:   endfor
  7:  endfor
  3: endfor

Interchanging loops i and j
 Browse  DD      Loop   *Restr   See     Undo    Var     Msgs    Quit    Xcape
```

Parallelize the j loop by going to the Restructure menu and typing P for Parallelize, giving:

```
  1: Entry
  1: real a(1:100,1:100)
  1: real b(1:100)
  2: integer n
  3: for k = 1,n do
  4:  a(k,k) = sqrt(a(k,k))
  5:  for i = k+1,n do
  6:   a(i,k) = a(i,k)/a(k,k)
  5:  endfor
  7:  doall j = k+1,n do
  5:   for i = max(k+1,j),n do
  8:    a(i,j) = a(i,j)-a(i,k)*a(j,k)
  5:   endfor
  7:  endfor
  3: endfor

Parallelize loop j
 Browse  DD     Loop   *Restr   See     Undo    Var     Msgs    Quit    Xcape
```

Now type X to escape to the main menu, and type W to write this version of the program to a file. The file will contain the program:

```
real a(1:100,1:100)
real b(1:100)
integer n
for k = 1,n do
a(k,k) = sqrt(a(k,k))
for i = k+1,n do
a(i,k) = a(i,k)/a(k,k)
endfor
doall j = k+1,n do
for i = max(k+1,j),n do
a(i,j) = a(i,j)-a(i,k)*a(j,k)
endfor
endfor
endfor
```

To work with the wave program, type ’P’ at the main menu (to parse a new program) and type ’wave’ in response to the prompt for the filename. The new program is displayed:

```
  1: Entry
  1: real a(1:100,1:100)
  3: integer n
  5: for k = 1,100 do
  7:  for i = 2,n-1 do
  9:   for j = 2,n-1 do
 11:    a(i,j) = (a(i-1,j)+a(i,j-1)+a(i+1,j)+a(i,j+1))/4
  9:   endfor
  7:  endfor
  5: endfor

Parsed wave
 Browse  File   *Parse   Restor  System  Trans   Write   Msgs    Quit
```

To try to parallelize the ’j’ loop, enter the ’Browse’ menu (by typing ’b’) and move to the ’j’ loop by using the ’L’ key:

```
  1: Entry
  1: real a(1:100,1:100)
  3: integer n
  5: for k = 1,100 do
  7:  for i = 2,n-1 do
  9:   for j = 2,n-1 do
 11:    a(i,j) = (a(i-1,j)+a(i,j-1)+a(i+1,j)+a(i,j+1))/4
  9:   endfor
  7:  endfor
  5: endfor

Parsed wave
 Browse  DD    *Loop    Restr   See     Undo    Var     Msgs    Quit    Xcape
```

Parallelize the 'j' loop by typing 'R' and choosing the Parallelize menu item:

```
  1: Entry
  1: real a(1:100,1:100)
  3: integer n
  5: for k = 1,100 do
  7:  for i = 2,n-1 do
  9:   for j = 2,n-1 do
 11:    a(i,j) = (a(i-1,j)+a(i,j-1)+a(i+1,j)+a(i,j+1))/4
  9:   endfor
  7:  endfor
  5: endfor

Parsed wave
 Bump   Circ   Dist   Inter  Neg   *Par    Skew   Vec    Msgs   Quit   Xcape
```

Unfortunately, parallel execution of the 'j' loop would be illegal; TINY responds by telling about the offending data dependence relations:

```
  1: Entry
  1: real a(1:100,1:100)
  3: integer n
  5: for k = 1,100 do
  7:  for i = 2,n-1 do
  9:   for j = 2,n-1 do
 11:    a(i,j) = (a(i-1,j)+a(i,j-1)+a(i+1,j)+a(i,j+1))/4
  9:   endfor
  7:  endfor
  5: endfor

flow dependence 11: --> 11:(<=,=,<)  (*,0,1)
*Accept    Next      Override  Msgs      Quit      Xcape
```

From this menu, you can accept the facts of life, or look at the next dependence relation, or override all these dependence relations (perform the transformation anyway). Typing 'N' to see the next dependence relation gives:

```
  1: Entry
  1: real a(1:100,1:100)
  3: integer n
  5: for k = 1,100 do
  7:  for i = 2,n-1 do
  9:   for j = 2,n-1 do
 11:    a(i,j) = (a(i-1,j)+a(i,j-1)+a(i+1,j)+a(i,j+1))/4
  9:   endfor
  7:  endfor
  5: endfor

anti dependence 11: --> 11:(<=,=,<)  (*,0,1)
 Accept  *Next      Override  Msgs      Quit      Xcape
```

Any transformation which would violate a data dependence relation gets to this menu; typing 'A' to accept these relations (or 'x' to escape) returns to the restructuring menu, without performing the transformation:

```
  1: Entry
  1: real a(1:100,1:100)
  3: integer n
  5: for k = 1,100 do
  7:  for i = 2,n-1 do
  9:   for j = 2,n-1 do
 11:    a(i,j) = (a(i-1,j)+a(i,j-1)+a(i+1,j)+a(i,j+1))/4
  9:   endfor
  7:  endfor
  5: endfor

Data Dependence prevents loop parallelization
 Browse  DD     Loop  *Restr  See     Undo    Var     Msgs    Quit    Xcape
```

In order to run one of 'i' or 'j' in parallel, since they both carry dependence relations, one of the loops must be skewed. Move to the 'j' loop and enter the restructuring menu again:

```
  1: Entry
  1: real a(1:100,1:100)
  3: integer n
  5: for k = 1,100 do
  7:  for i = 2,n-1 do
  9:   for j = 2,n-1 do
 11:    a(i,j) = (a(i-1,j)+a(i,j-1)+a(i+1,j)+a(i,j+1))/4
  9:   endfor
  7:  endfor
  5: endfor

Data Dependence prevents loop parallelization
 Bump  Circ  Dist  Inter  Neg   Par   *Skew  Vec   Msgs   Quit   Xcape
```

Choose 'Skew', to get to the skew menu. Loop skewing with respect to outer loops is always legal; this menu allows me to choose the loop with respect to which to skew, though the skewing factor is always one:

```
  1: Entry
  1: real a(1:100,1:100)
  3: integer n
  5: for k = 1,100 do
  7:  for i = 2,n-1 do
  9:   for j = 2,n-1 do
 11:    a(i,j) = (a(i-1,j)+a(i,j-1)+a(i+1,j)+a(i,j+1))/4
  9:   endfor
  7:  endfor
  5: endfor

Data Dependence prevents loop parallelization
 Out       *Forward    Reverse    By_Factor  Msgs       Quit       Xcape
```

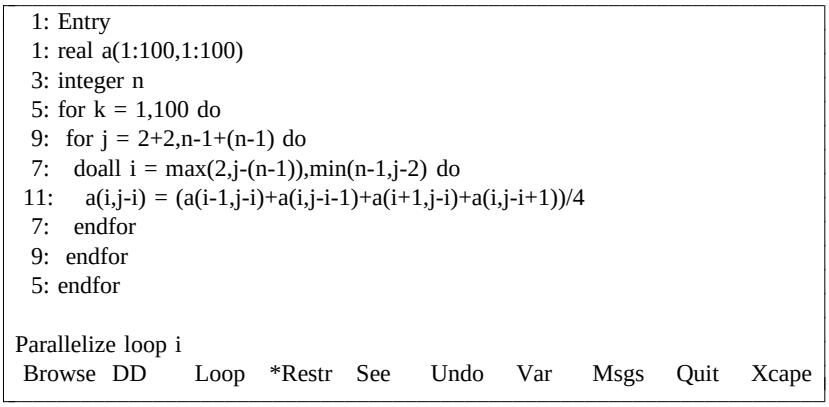
Choosing 'Out' here moves the marker to the next outer loop; choosing (-1 factor)skewing with respect to the marked loop; choosing By_Factor will skew by any constant integer factor. To choose forward skewing with respect to the 'i' loop, type 'f':

```
  1: Entry
  1: real a(1:100,1:100)
  3: integer n
  5: for k = 1,100 do
  7:  for i = 2,n-1 do
  9:   for j = 2+i,n-1+i do
 11:    a(i,j-i) = (a(i-1,j-i)+a(i,j-i-1)+a(i+1,j-i)+a(i,j-i+1))/4
  9:   endfor
  7:  endfor
  5: endfor

Forward skew loop j with respect to i
 Browse  DD      Loop   *Restr   See     Undo    Var     Msgs    Quit    Xcape
```

Forward loop skewing simply adds the outer loop index to the lower and upper limit expressions (adds 'i' to the limits of 'j'), and subtracts the outer loop index from the inner loop index within the body of the loop. The other effect is on the data dependence relations, changing a (<,=) to a (<,<). Now, interchanging the 'j' and 'i' loops gives:

```
  1: Entry
  1: real a(1:100,1:100)
  3: integer n
  5: for k = 1,100 do
  9:  for j = 2+2,n-1+(n-1) do
  7:   for i = max(2,j-(n-1)),min(n-1,j-2) do
 11:    a(i,j-i) = (a(i-1,j-i)+a(i,j-i-1)+a(i+1,j-i)+a(i,j-i+1))/4
  7:   endfor
  9:  endfor
  5: endfor

Interchanging loops i and j
 Browse  DD      Loop   *Restr   See     Undo    Var     Msgs    Quit    Xcape
```

This is the 'wavefront' formulation of the loop. Now, parallelizing the 'i' loop is legal:



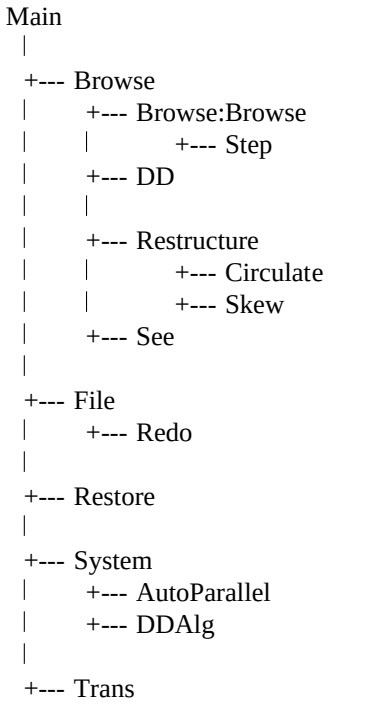
This version of the program can be written to a file by escaping to the main menu and choosing 'Write'.

6. Menu Descriptions

The various menus are described here. The main menu is first, and the remaining menus are listed alphabetically. The menus, along with their parentage, are:

Main	from command line
AutoParallel	from System menu
Browse	from Main Menu
Browse:Browse	from Browse Menu
Circulate	from Restructure Menu
DD Browse	from Browse Menu
DD Algorithm	from System Menu
DD Prevents	from restructuring transformation
File	from Main Menu
Find	from See or Step Menu
Msgs	from any menu
Redo	from File Menu
Restore	from Main Menu
Restructure	from Browse Menu
See	from Browse Menu
Skew	from Restructure Menu
Step	from Browse:Browse Menu
System	from Main Menu
Trans	from Main Menu

A pictorial diagram of the menu lineage is:



Again, typing 'M' at any menu will get to the Msgs menu, displaying recent messages from TINY; typing 'Q' from any menu will quit TINY immediately; typing 'X' (or the escape key) from any menu except the Main menu will exit (escape) to the parent menu.

6.1. Main Menu

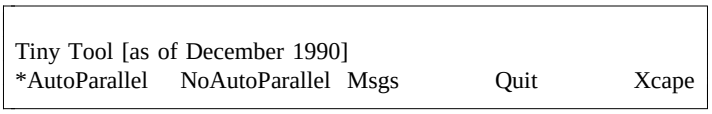
```
Tiny Tool [as of December 1990]
*Browse File  Parse  Restor System Trans  Write  Msgs   Quit
```

From the main menu, there are the following eight choices:

Browse	Go to the Browse menu to create new program version via interactive restructuring or to browse the data structures.
File	Go to the File menu to move around directories, edit files, start command shells, etc.
Parse	Choose an initial file (if no file was given on the command line) or choose a different file to parse and then browse. TINY will ask for the file name to parse.
Restor	Go to the Restore menu to restore an old program version.
System	Go to the System menu to change dependence decision algorithm or verify data structures.
Trans	Go to the Translate menu to change the display language, or to write out an assembler file.
Write	Write the current program version to a file; TINY will ask for the name of the file to write.
Msgs	Go to the Msgs menu and display the most recent messages.
Quit	Quit TINY.

6.2. AutoParallel Menu

This menu lets you decide whether or not TINY should attempt to parallelize every loop after each transformation. This is a simple way to see how your transformations have affected the parallelism in the program. Initially this option is disabled; you can enter this menu from the System menu.



AutoParallelEnable automatic parallelization.

NoAutoParallel
Disable automatic parallelization.

6.3. Browse Menu

This is the menu where you will probably spend lots of time. When you enter the Browse Menu, TINY highlights a 'current position' in the program. This will generally be a loop or a variable reference. When you first enter this mode, the current position will be at the entry.

```
  1: Entry
  1: real a(1:100,1:100,1:100)
  3: integer n
  5: for k = 2,n do
  7:  for i = 2,n do
  9:   for j = 2,n do
 11:    a(k,i,j) = a(k,i-1,j)+a(k,i,j-1)+a(k,i,j+1)+a(k,i+1,j)+a(k-1,i,j)+a(k+1,
i,j)
  9:   endfor
  7:  endfor
  5: endfor

Parsed wave3a
*Browse  DD     Loop    Restr   See     Undo    Var     Msgs    Quit    Xcape
```

From this menu you can browse around the data structures, move the current position to another loop or variable, and perform or undo restructuring transformations.

Browse — Go to the Browse:Browse menu (all right, it's a stupid name).

DD — Go to the DD Browse Menu to examine dependence relations.

Loop — Move the current position to the next loop.

Restr — Go to the Restructure Menu, to perform a restructuring transformation on this loop; the current position must be a loop.

See — Traverse the abstract syntax tree (AST) data structure interactively, by going to the See menu.

Undo — Undo the most recent restructuring transformation. See the discussion under 'Restore Menu' to see how this is implemented.

Var — Move the current position to the next variable reference. Note that due to the data structure for assignments, the right hand side expressions are visited 'before' the left hand side.

6.4. Browse:Browse Menu

This is a simple way to traverse the abstract syntax tree (AST) in detail, by moving to sibling, parent or child tree nodes one at a time. As in the Browse menu, the current position is highlighted.

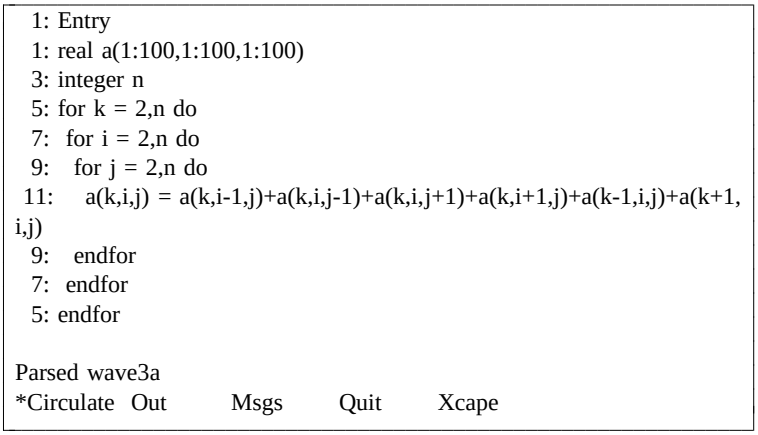
```
  1: Entry
  1: real a(1:100,1:100,1:100)
  3: integer n
  5: for k = 2,n do
  7:  for i = 2,n do
  9:   for j = 2,n do
 11:    a(k,i,j) = a(k,i-1,j)+a(k,i,j-1)+a(k,i,j+1)+a(k,i+1,j)+a(k-1,i,j)+a(k+1,
i,j)
  9:   endfor
  7:  endfor
  5: endfor

Parsed wave3a
*Back  Decl  Jump  Loop  Next  Step  Var   Msgs  Quit  Xcap
```

The various menu choices move the current position around:

Back	Move the current position to the last position.
Decl	If the current position is on a variable reference, move the current position to the declaration of that variable.
Jump	Move to the next non-trivial operator or variable reference.
Loop	Move the current position to the next loop.
Next	If the current position is on a variable reference, move the current position to the next reference of that variable.
Step	Go to the Step Menu.
Var	Move the current position to the next variable reference.

6.5. Circulate Menu

When you want to circulate a loop ordering, this menu allows you to choose what type of circulation you want. Intercirculation is a circulation of the current loop to a position inside of some inner loop; outercirculation is a circulation of the current loop to a position outside of some enclosing loop. The Circulate Menu display highlights the loop inside or outside of which the current loop can be circulated. The Out menu choice highlights the next possible choice, and the Circulate menu choice enables circulation to inside of or outside of the highlighted loop.

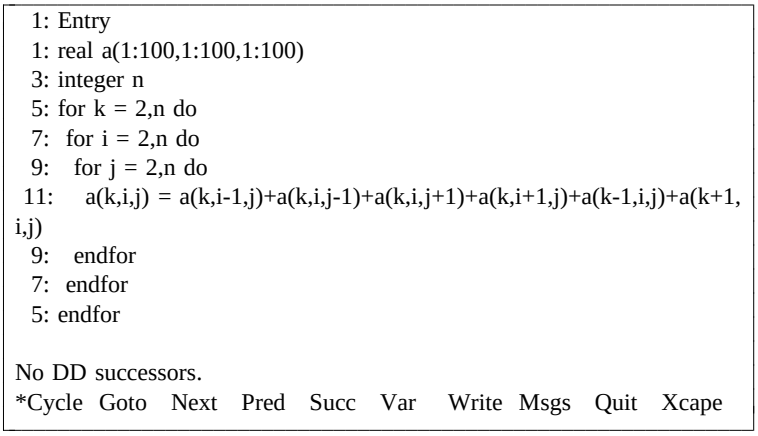


Circulate attempt to circulate the current loop inside of or outside of the highlighted loop.

Out move the highlight to the next tightly-nested outer loop.

6.6. DD Browse Menu

At this menu you can inspect the data dependence relations in the program.



The message line will display the first dependence relation from the current position; if there are no dependence relations (as, for instance, for non-variable reference nodes) then this will also be displayed. The kind of dependence, line numbers, direction vector and distance vector are displayed; direction or distance vector elements which are unknown are displayed as asterisks. By default, this will display dependence successors of the current node, and both source and target of the dependence relation are highlighted.

Cycle perhaps the most useful menu choice, this cycles through all the dependence relations in the program, moving the current position as necessary.

Goto move the current position to the 'other end' of the dependence relation being displayed.

Next display the next dependence successor or predecessor of the current node.

Pred display dependence predecessors of the current node.

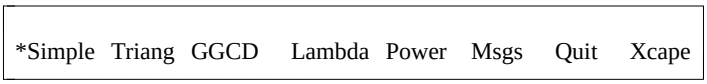
Succ display dependence successors of the current node.

Var move the current position to the next variable reference.

Write prompts for a file name, and write the dependence relations to that file.

6.7. DD Algorithm Menu

This menu lets the user choose what decision algorithm to use to solve the subscript dependence equation. This option will not take effect until the next program is parsed (using the Parse item from the Main Menu).



Simple — Use a simple set of tests, including an exact test when only a single loop index variable appears in a subscript (to get dependence distances), and the GCD and simple Banerjee's Inequalities otherwise (to get dependence directions). These tests are applied subscript-by-subscript.

Triang — As above, except use triangular Banerjee's Inequalities instead of the simple Inequalities.

GGCD — Use Banerjee's Generalized GCD simultaneous subscript test; this gives dependence distances if fixed.

Lambda — Use an implementation of the Lambda Test (see Li, Yew and Zhu's paper "Data Dependence Analysis on Multi-Dimensional Array References" in the 1989 ACM Int'l Conf. on Supercomputing proceedings, or Grunwald's paper "Data Dependence Analysis: The Lambda Test Revisited" in the 1990 Int'l Conf. on Parallel Processing proceedings).

Power — Use Banerjee's Generalized GCD test, extended by a different search for an empty solution space by modified Fourier-Motzgin search.

6.8. DD Prevents Menu

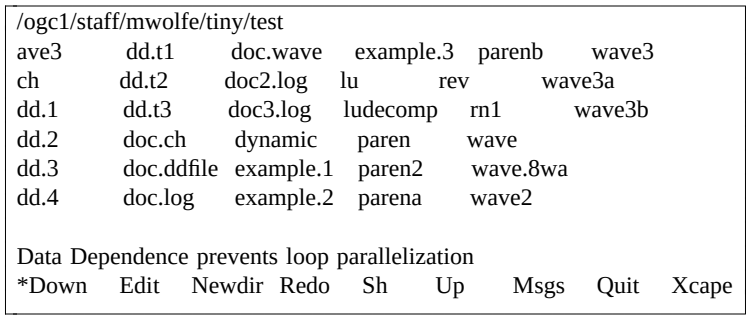
When data dependence relations prevent application of a restructuring transformation, those dependence relations are displayed. The user can view all the relations (using the Next menu option), Accept the restriction, or Override the dependence relations. Note: Choosing Override will blindly apply the transformation; the modified dependence graph after the transformation will probably no longer be valid. In the example shown, the user tried to parallelize the 'k' loop. Since the 'k' loop carries a dependence relation (actually, two dependence relations), parallel execution is not allowed.

```
  1: Entry
  1: real a(1:100,1:100,1:100)
  3: integer n
  5: for k = 2,n do
  7:  for i = 2,n do
  9:   for j = 2,n do
 11:    a(k,i,j) = a(k,i-1,j)+a(k,i,j-1)+a(k,i,j+1)+a(k,i+1,j)+a(k-1,i,j)+a(k+1,
i,j)
  9:   endfor
  7:  endfor
  5: endfor

flow dependence 11: --> 11:(=,=,<)  (0,0,1)
*Accept    Next      Override  Msgs      Quit      Xcape
```

Accept	accept this dependence restriction; do not apply the transformation.
Next	display the next data dependence relations that prevents this dependence.
Override	override the dependence restrictions and apply the transformation anyway.

6.9. File Menu

The file menu appears with a listing of the file names in the current directory. Subdirectory names are shown with a '/', and executable file names are shown with a '*', much as the command 'ls -F' does; the current directory name is shown at the top of the file listing:



Down — Move down to a subdirectory; TINY will prompt for the subdirectory name to which to change. Currently the whole subdirectory name must be typed.

Edit — Edit a file; currently, the editor to use is hard-coded in TINY as 'vi'.

Newdir — Make a new directory; TINY will prompt for the name of the subdirectory name to create.

Redo — Go to the Redo (Restart) menu, from which stopped subprocesses can be restarted. Currently, the only processes that can be stopped are editor subprocesses.

Sh — Start up a command shell; currently, the shell to use is hard-coded in TINY as 'csh'.

Up — Move Up to the parent menu (a la 'cd ..').

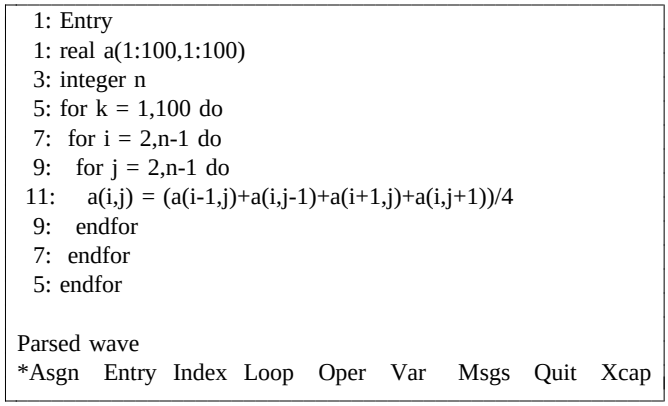
Msgs — Go to the Message display and menu.

Quit — Quit TINY.

Xcape — Return to the Main Menu.

6.10. Find Menu

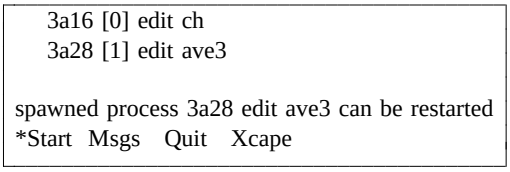
From the See or Step Menu, this menu moves the current position to the next operator of the specified type:



Asgn	move the current position to the next assignment operator.
Entry	move the current position to the entry node.
Index	move the current position to the next loop index reference.
Loop	move the current position to the next loop.
Oper	move the current position to the next operator.
Var	move the current position to the next variable reference.

6.11. Redo Menu

A list of stopped processes is given.



Any one process can be restarted by choosing the menu item Start, and typing the number (0-9, in square brackets) corresponding to that process. Currently only 9 stopped processes are saved, and only editor processes may be stopped.

Start Start one of the stopped processes.

6.12. Restore Menu

This interface is all new since the last version. When you restructure programs, or parse new programs, as long as you do not quit TINY, the data structures for each version of each program is saved. Using the Restore Menu, you can return to any previously parsed or restructured version of a program in this TINY session. The Restore display shows the chain of restructuring transformations taken to get to the 'current' version of the program:

```
File: wave3a
 Original Program
  Skew loop j with respect to i
   Skew loop j with respect to k
    Circulate loop j outside of loop k

Outercirculating loop j outside of k
*Child   Parent  Next    Prev    Msgs    Quit    Xcape
```

This display says that the original program was read from file 'wave3a', then skewed the 'j' loop with respect to 'i' and 'k', then circulated (interchanged) the 'j' loop to outside the 'k' loop. To return to a previous version, choose Parent from the menu, which will effectively 'undo' the bottom transformation; this is exactly how the 'Undo' menu item in the Browse Menu is implemented. That transformation can be effectively reapplied by choosing 'Child' from the menu. If there are 'children' programs or derived programs from the current one, they are displayed with number identifiers:

```
File: wave3a
 Original Program
  Skew loop j with respect to i
   Skew loop j with respect to k
    0: Circulate loop j outside of loop k

Outercirculating loop j outside of k
 Child  *Parent  Next    Prev    Msgs    Quit    Xcape
```

Choosing Child will cause Tiny to ask which child you want, and you are expected to respond with a digit, 0 through 9.

Child — Choose a child version of the program as the current version, effectively 'reapplying' a transformation.

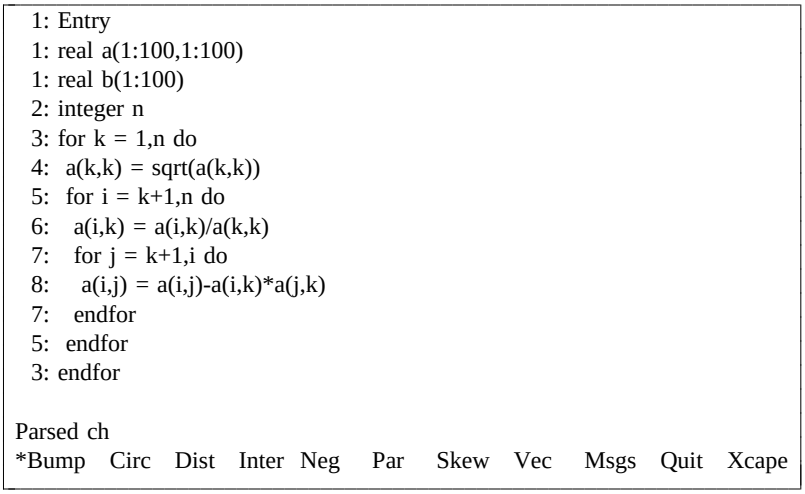
Parent — Choose the parent version of the program as the current version, effectively 'undoing' a transformation.

Next — Choose the next version of the program, essentially the next child of the parent. For a top-level program, this goes to a 'later-parsed' program.

Prev — Choose the previous version of the program, essentially the previous child of the parent. For a top-level program, this goes to an 'earlier-parsed' program.

6.13. Restructure Menu

Here you choose one of several restructuring transformations to perform on the current loop. The current position must be a loop.



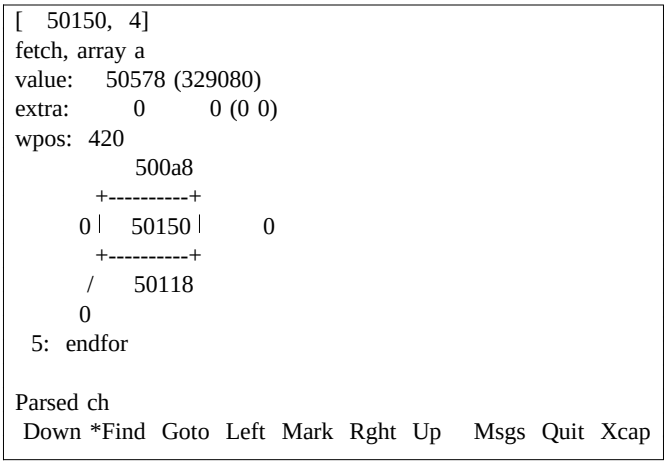
Most transformations have dependence tests which must be satisfied for the transformation to be legal. They also may modify the dependence relations, such as changing the dependence direction or distance. The restructuring transformations allowed are:

Bump	bump a loop by adding a signed integer constant to both lower and upper limits.
Circ	circulate the current loop inside or outside of a nest of tightly-nested loops.
Dist	distribute the current loop.
Inter	interchange the current loop with its immediately surrounding tightly nested surrounding loop.
Neg	negate (reverse) the current loop.
Par	parallelize the current loop.
Skew	skew the current loop with respect to an outer loop.
Vec	vectorize the current loop, if it is an inner loop and has no dependence cycles (change to a 'forall').

Transformations in the works are loop rotation, tiling and vectorization.

6.14. See Menu

This allows you to see a 'binary dump' of each abstract syntax tree node, and to move around the AST:



The binary dump includes the hexadecimal address of the node, with its line number (in square brackets), the node operator, its value in hex and decimal, the extra nodes and window position, and graphical display of the parent, child, sibling and link relationships. The menu choices are the same as for the Step Menu.

Down	Move the current position to its first child.
Find	Go to the Find Menu, to move the current position to the next operator of a particular type.
Goto	Move the current position to one of the 26 previously marked positions.
Left	Move the current position to its left sibling.
Mark	Mark the current position as one of 26 saved positions; TINY will prompt for a single-letter position name, [a-z].
Rght	Move the current position to its right sibling.
Up	Move the current position to its parent.

6.15. Skew Menu

When you want to skew a loop, this menu allows you to choose what type of skewing (forward or reverse) and the loop with respect to which you want to skew. Forward skewing means skewing with a factor of +1, and reverse skewing is with a factor of -1. The Circulate Menu display highlights the outer loop with respect to which the skewing will be done. The Out menu choice highlights the next possible choice, and the Forward or Reverse menu choice enables the appropriate kind of skewing.

```
  1: Entry
  1: real a(1:100,1:100,1:100)
  3: integer n
  5: for k = 2,n do
  7:  for i = 2,n do
  9:   for j = 2,n do
 11:    a(k,i,j) = a(k,i-1,j)+a(k,i,j-1)+a(k,i,j+1)+a(k,i+1,j)+a(k-1,i,j)+a(k+1,
i,j)
  9:   endfor
  7:  endfor
  5: endfor

No saved program in that direction
*Out        Forward    Reverse    By_Factor  Msgs       Quit       Xcape
```

Out	Move the highlight to the next outer loop.
Forward	Enable loop skewing by factor of +1.
Reverse	Enable loop skewing by factor of -1.
By_Factor	Prompts for a (optionally signed) constant integer skewing factor; enables skewing by that factor.

6.16. Step Menu

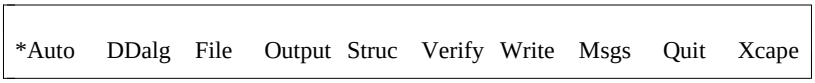
This is reached from the Browse:Browse Menu, and allows detailed traversal of the data structures.

```
  1: Entry
  1: real a(1:100,1:100,1:100)
  3: integer n
  5: for k = 2,n do
  7:  for i = 2,n do
  9:   for j = 2,n do
 11:    a(k,i,j) = a(k,i-1,j)+a(k,i,j-1)+a(k,i,j+1)+a(k,i+1,j)+a(k-1,i,j)+a(k+1,
i,j)
  9:   endfor
  7:  endfor
  5: endfor

No saved program in that direction
*Down  Find  Goto  Left  Mark  Rght  Up    Msgs  Quit  Xcap
```

Down	Move the current position to its first child.
Find	Go to the Find Menu, to move the current position to the next operator of a particular type.
Goto	Move the current position to one of the 26 previously marked positions.
Left	Move the current position to its left sibling.
Mark	Mark the current position as one of 26 saved positions; TINY will prompt for a single-letter position name, [a-z].
Rght	Move the current position to its right sibling.
Up	Move the current position to its parent.

6.17. System Menu

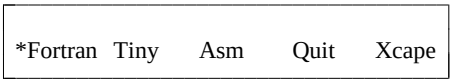
The system menu lets the user change certain special options.



Auto	Go to the AutoParallel Menu to decide whether or not to autoparallelize every loop after each transformation.
DDalg	Go to the DD Algorithm Menu to change the DD decision algorithm.
File	Prompts for a file name, then reopens 'debug' as that file.
Output	Reopens 'debug' as standard output.
Struc	Dump the abstract syntax tree (AST) data structure to the current 'debug' file.
Verify	Verifies that the abstract syntax tree (AST) has no bogus pointers. The verifier reports any data structure inconsistencies. This is useful when debugging new transformations.
Write	Writes the program to the current 'debug' file. This is useful when 'debug' is another file.

6.18. Trans Menu

This menu lets you choose to view the program in a Fortran syntax as opposed to TINY syntax, or to compile the program into Alliant assembler code.



Fortran — This option lets you view the program using Fortran syntax for loops. You may have to type ^L (control-L) to get the Fortran to show up. Here, Alliant directives are used to show parallel and vector loops. If Fortran is chosen, then when the program is written out, it will also use Fortran syntax. Note that Tiny does NOT have a Fortran parser, so it will not accept this syntax as input. Also note that the output may need modifications to be compiled and executed, since the Tiny language has no procedure header statements and the like.

Tiny — This option lets you view the program in the default Tiny syntax.

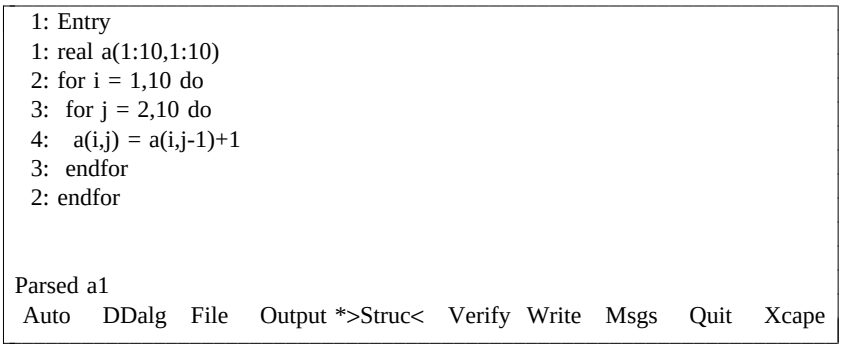
Asm — This option will "compile" the program, as it has been transformed, into Alliant FX/8 assembler code.

7. AST Dump Information

A sample 'dump' is given here. For the program:

```
real a(10,10)
for i = 1 to 10 do
 for j = 2 to 10 do
  a(i,j) = a(i,j-1) + 1
 endfor
endfor
```

the interactive display is:



The dump (to the 'debug' file, whether it be opened to standard output, the default, or to a file) would be:

```
.[   35424]        0=Child,     353f0=Next, entry
.                  0=Parnt,         0=Prev,         0=Val
.[   353f0]    353bc=Child,     3521c=Next, declare
.                  0=Parnt,    35424=Prev,    21404=Val
..[   353bc]    35388=Child,    35320=Next, bounds
..               353f0=Parnt,         0=Prev,         0=Val
...[   35388]        0=Child,    35354=Next, constant
...              353bc=Parnt,         0=Prev,         1=Val
...[   35354]        0=Child,         0=Next, constant
...              353bc=Parnt,    35388=Prev,         a=Val
..[   35320]    352ec=Child,         0=Next, bounds
..               353f0=Parnt,    353bc=Prev,         0=Val
...[   352ec]        0=Child,    352b8=Next, constant
...              35320=Parnt,         0=Prev,         1=Val
...[   352b8]        0=Child,         0=Next, constant
...              35320=Parnt,    352ec=Prev,         a=Val
.[   3521c]    351e8=Child,         0=Next, do
.                  0=Parnt,    353f0=Prev,         1=Val
..[   351e8]    3514c=Child,    35284=Next, dolimit
..               3521c=Parnt,         0=Prev,    21444=Val
...[   3514c]    35118=Child,         0=Next, do
...              351e8=Parnt,         0=Prev,         2=Val
....[   35118]    34edc=Child,    351b4=Next, dolimit
....             3514c=Parnt,         0=Prev,    21484=Val
.....[   34edc]    34ea8=Child,    34e40=Next, stmtnumber
.....            35118=Parnt,         0=Prev,         1=Val
......[   34ea8]        0=Child,    34e74=Next, index
......             34edc=Parnt,         0=Prev,    351e8=Val
......[   34e74]        0=Child,         0=Next, index
```

```
......          34edc=Parnt,   34ea8=Prev,   35118=Val
.....[  34e40]   34f10=Child,      0=Next, assign
.....          35118=Parnt,   34edc=Prev,      0=Val
......[  34f10]   35048=Child,   350e4=Next, add
......          34e40=Parnt,      0=Prev,      0=Val
.......[  35048]   35014=Child,   34f44=Next, fetch_array
.......          34f10=Parnt,      0=Prev,   353f0=Val
........[  35014]      0=Child,   34f78=Next, index
........          35048=Parnt,      0=Prev,   351e8=Val
........[  34f78]   34fe0=Child,      0=Next, subtract
........          35048=Parnt,   35014=Prev,      0=Val
.........[  34fe0]      0=Child,   34fac=Next, index
.........          34f78=Parnt,      0=Prev,   35118=Val
.........[  34fac]      0=Child,      0=Next, constant
.........          34f78=Parnt,   34fe0=Prev,      1=Val
.......[  34f44]      0=Child,      0=Next, constant
.......          34f10=Parnt,   35048=Prev,      1=Val
......[  350e4]   350b0=Child,      0=Next, store
......          34e40=Parnt,   34f10=Prev,   353f0=Val
.......[  350b0]      0=Child,   3507c=Next, index
.......          350e4=Parnt,      0=Prev,   351e8=Val
.......[  3507c]      0=Child,      0=Next, index
.......          350e4=Parnt,   350b0=Prev,   35118=Val
....[  351b4]      0=Child,   35180=Next, constant
....          3514c=Parnt,   35118=Prev,      2=Val
....[  35180]      0=Child,      0=Next, constant
....          3514c=Parnt,   351b4=Prev,      a=Val
..[  35284]      0=Child,   35250=Next, constant
..          3521c=Parnt,   351e8=Prev,      1=Val
..[  35250]      0=Child,      0=Next, constant
..          3521c=Parnt,   35284=Prev,      a=Val
```

The number in brackets is the hexadecimal address of that AST entry. The Child, Next, Parnt and Prev numbers are the hexadecimal addresses of the AST entries pointed to by the Child, Next, Parent and Previous pointers. The Node Operator is given in text, and the Value is given is hex.

8. Installation and Distribution

TINY was designed to be relatively portable, but it's pretty rough; in many cases, a choice between elegance, portability and simplicity was made in favor of ease of implementation. It has been installed on many a Unix system using native compilers and the Gnu 'gcc', and on an IBM PC-clone using the Turbo C++ compiler. The entire design and implementation is geared toward supporting a research effort into elementary program restructuring, not toward developing an industrial-strength product. It is written in ANSI 'C' (bleah), with special hooks to be able to compile it on compilers without ANSI function headers (just about the only ANSI C features used). TINY uses a character-based interface, using the 'Curses' character windowing package to manage the screen (bleah) under Unix, and uses the Turbo C screen addressing routines on a PC. I would like to install TINY on an X-window interface, but that awaits some other interested graduate student.

In the best of cases, a simple 'make t' do all the compiles and links. Some modifications may be necessary.

I will gladly accept bug reports, suggestions or enhancements, but I cannot promise that anything will get fixed, except that I'll do my best, given the time and resources at my command.

The source or object code of TINY may be freely redistributed and reused as you see fit. Have fun.

Table of Contents