```
(:) begin
        ?comp here 1 (;) immediate

(:) again
        ?comp 1 ?pairs [ ' branch ] literal , here - , (;) immediate

(:) until
        ?comp 1 ?pairs [ ' 0branch ] literal , here - , (;) immediate

(:) while
        ?comp 1 ?pairs [ ' 0branch ] literal , here 0 , 4 (;) immediate

(:) repeat
        ?comp 4 ?pairs [ ' branch ] literal , swap here - ,
        dup here swap - swap ! (;) immediate

(:) do
        [ ' (do) ] literal , here 3 (;) immediate

(:) loop
        3 ?pairs ?comp [ ' (loop) ] literal , here - , (;) immediate

(:) +loop
        3 ?pairs ?comp [ ' (+loop) ] literal , here - , (;) immediate

(:) compile
        ?comp R> dup 2+ >R @ , (;)

(:) [compile]
        ' dup  0= if
                ." Not in context vocabulary." cr abort
        then
                , (;) immediate

(:) (toggle)
        over over and if not and else or then (;)

(:) toggle
        over C@ swap (toggle) swap C! (;)

(:) smudge
        last 64 toggle (;)

(:) :
        ?exec 0 (:) smudge (;) immediate

(:) ;
        ?comp  0 ?pairs [ ' (;) , ] smudge (;) immediate

(:) R:
        ?exec 0 (:) (;) immediate

(:) R;
        ?comp 0 ?pairs [ ' (;) , ] (;) immediate

(:) unlink
        CONTEXT @ 2- @ CONTEXT ! definitions (;)

(:) forget
        -find 0=
        if
                ." Not in context vocabulary." cr drop
        else
                swap drop
                dup FENCE @ <
                if
                        ." In protected vocabulary." cr drop
                else
                        dup pfa 4 + CONTEXT @ = if
                                unlink ." Unlinking vocabulary." cr
                        then
                                dup DP ! pfa lfa @ CONTEXT @ ! definitions
                then
        then (;)

(:) bytes
        -1 0  here s->d d- <# #s #> type space (;)
```

```
(:) free
        bytes ." bytes of dictionary remaining." cr (;)

vocabulary tile
tile definitions
here FENCE !
free
```

```
R: s.
        empty? not
        if
                >R s. R> dup .
        then R;

: ?range
        @ > if ." Vector range error." cr abort then ;

: vector
        <builds dup , 2 * allot
        does> 2dup ?range swap 2 * + ;

: case
        ?comp 3 4 ; immediate

: of
        4 ?pairs compile over compile = compile 0branch here 0 ,
        compile drop 5 ; immediate

: endof
        5 ?pairs compile branch here 0 , swap
        2 [compile] then 4 ; immediate

: endcase
        4 ?pairs compile drop
        begin
            dup 3 = not
        while
            2 [compile] then
        repeat
        drop ; immediate

: bin.
        BASE @ swap binary u. BASE ! ;

: hex.
        BASE @ swap hex u. BASE ! ;

: dec.
        BASE @ swap decimal u. BASE ! ;

: dump
        BASE @ swap
        hex
        begin
                dup 64 + swap
                        8 0 do cr
                                dup I 8 * +
                                dup 0 4 d.r space
                                8 0 do
                                        dup I + C@
                                        3 .r
                                loop
                                drop
                        loop
                drop cr
                key BL -
        until
        drop BASE ! cr ;

variable (nothing)

: nothing
        [ here (nothing) ! ] ." Action not defined." cr abort ;

: vary
        <builds (nothing) @ ,
        does> R> drop @ >R ;

: (make)
        R> swap ! ;

: make
        ?comp compile lit ' 2+ , compile (make) ; immediate

variable struct

: structure
```

```
        IN @ <builds 0 , 0 , IN ! ' 4 + struct ! does> ;

: field
        <builds struct @ dup , 2+ dup @ , +!
        does> dup @ @ swap 2+ @ + ;

: size
        2+ @ ;

: generate
        here 2dup swap size 0 fill swap size allot ;

: expect
        TIB @ IN @ rot TIB ! inline IN ! TIB ! ;

: data
        <builds here 6 + dup , , here 0 , 1 does> ;

: end
        , 1 ?pairs here swap ! ;

: read
        dup dup @ swap 4 + @ < not if
                ." Read error." cr abort
        else
                dup @ dup @ swap 2+ rot !
        then ;

: restore
        dup 2+ @ swap ! ;


: 1@
        dup ;

: 2@
        over ;

: 3@
        rot dup >R rot rot R> ;

: 4@
        >R 3@ R> swap ;

: 5@
        >R 4@ R> swap ;

: 6@
        >R 5@ R> swap ;

: 7@
        >R 6@ R> swap ;

: 8@
        >R 7@ R> swap ;
```

```
decimal

vocabulary basic        basic definitions
vocabulary logic
vocabulary out
vocabulary in

in seal                 out seal                logic seal

basic definitions

: (version)
        ." 1.02" ;

." Simple BASIC Version " (version) cr
." Copyright 1990 St. Andrews University." cr
." Written By Steven James." cr cr
." Loading Compiler..." cr

: create \ Forth 79 Version of create.
        <builds does>  ;

100 constant size       \ The size of the symbol table.

create symbol           \ Create the symbol table.
        size 4 * allot

: initsymbol \ --. Initialise the symbol table.
        symbol 2+ symbol ! ;

variable line#          \ The most recently defined line number.

: report \ n --. Display the information about line n.
        stdin? not if
                ." Compiling -->" . cr
        else
                drop
        then ;

: line \ n --. Update the most recently defined line number.
        dup 0 <> if
                dup line# ! report
        else
                drop
        then ;

: errorinline \ --. Display the standard error message and abort.
        space ." in line" space line# @ . abort ;

: ?finished \ n addr -- f. Check if addr contains n, or addr points to
             \ the bottom of the symbol table.
        dup symbol @ = >R @ = R> or ;

: lookup \ n -- addr. Look up n in the symbol table and return an
          \ associated address, or false if not present.
        symbol 2+
        begin
                2dup ?finished not
        while
                4 +
        repeat
                swap drop dup symbol @ = if drop 0 then ;

: symbol! \ addr n --. Store n and addr in the next free element in
          \ in the symbol table.
        symbol @ ! symbol @ 2+ ! 4 symbol +! ;

: >symbol \ addr n --. Insert n and addr into the symbol table only if
          \ n is not preset, otherwise generarte an error message
          \ and abort.
        dup dup line lookup 0= if
                dup 0= if
                        drop drop
                else
                        symbol!
                then
        else
                cr ." Line" space . ." has already been defined." abort
        then ;
```

```
: symbol> \ n -- addr. Retrieve the addr associated with n from the
          \ symbol table.
        dup lookup dup 0= if
                drop cr ." Line" space . ." is not defined" abort
        else
                2+ @ swap drop
        then ;

variable last;          \ The last occurence of ';'
variable last,          \ The last occurence of ','

: tidy \ --. Remove any free occurences of ';' or ',' from the
       \ compilation stack.
        here last; @ = if               \ Dangling ';' ?
                2drop 2drop             \ Remove ';'
        else
                here last, @ = if       \ Dangling ',' ?
                        2drop           \ Remove ','
                then
        then ;

variable #(             \ The depth of parentheses.

: (
        10 #( +! ; immediate

: )
        -10 #( +! #( @ 0< if
                cr ." Unmatched )" errorinline
        then ; immediate


: defer \ addr' n' addr n -- addr'' n''. Compile into the dictionary all
        \ those addresses with a precedence greater or equal to n.
        #( @ +
        begin
                2over swap drop over < not
        while
                2swap drop ,
        repeat ;

: nextcfa \ -- cfa. Return the cfa of the next word in the input
          \ stream, without advancing the IP.
        IN @ ' swap IN ! ;

: precedence \ n -- cccc. Re-compile cccc with a precedence value of n.
        nextcfa create , , immediate does> dup @ swap 2+ @ defer ;

: endexpr \ --. Compile in any remaing precedence values.
        tidy 0 1 defer 2drop #( @ if
                cr ." Syntax error" errorinline
        then ;

: nul \ --. Compile a null line number into the dictionary.
        0 , 0 , ;

: statement \ --. Insert the previously compiled line number into the
            \ symbol table along with the current value of the
            \ dictionary pointer.
        here 2- @ >R -4 allot endexpr here R> >symbol ;

variable context

: address \ --. Force the compiler to treat all variables as addresses.
        1 context ! ;

: value \ --. Force the compiler to treat all variables as values.
        0 context ! ;

: ?address \ -- f. Returns true if the current context is 1.
        context @ 1 = ;

: integer \ --. Declare a BASIC variable.
        variable immediate
        does> [compile] literal ?address if
                value
        else
                [ ' @ ] literal ,
```

```
        then ;

: jump \ addr --. Jump to addr.
        R> drop >R ;

: 0jump \ f addr --. Jump to addr if false.
        swap 0= if R> drop >R else drop then ;

: jumps \ addr --. Jump to the subroutine at addr.
        >R ;

: rts \ --. Return from a subroutine call.
        R> drop ;

: litnumber \ --- cccc. Compile the following words as a literal.
        BL word number drop [compile] literal ;

: lnnumber \ -- cccc. Compile the following word as a line number.
        litnumber [ ' symbol> ] literal , ;

: (goto) \ -- n. Compile in a jump to line n.
        lnnumber [ ' jump ] literal , ;

: (gosub) \ -- n. Compile in a jump to subroutine al line n.
        lnnumber [ ' jumps ] literal , ;

: (return) \ --. Compile in a return from subroutine.
        [ ' rts ] literal , ;

: (then) \ --. Compile in a 0jump to the next line.
        [ ' lit ] literal , symbol @ 2+ ,
        [ ' @ ] literal , [ ' 0jump ] literal , ;

: 1 \ -- 1. Place the initial increment of 1 onto the satck.
        1 ;

: (to) \ -- n addr. Generate a FOR..NEXT frame.
        here [compile] literal [ ' 1 ] literal 1 ;

: inc \ addr n m var -- addr n m var. Increment the variable @ var by m.
        2dup +! ;

: ?to \ addr n m var -- addr f. Has var reached the n limit ?
        swap 0< if
                @ >
        else
                @ <
        then ;

: (next) \ addr n m var --. Increment the variable var and check that it is
         \ smaller than m, if so then jump to addr otherwise clear the
         \ stack.
        inc ?to if
                drop
        else
                R> drop >R
        then ;

variable (rnd)          here (rnd) !    \ Initialise seed.

: random \ --.
        (rnd) @ 31421 * 6927 + dup (rnd) ! ;

: rnd \ u1 -- u2.
        random u* swap drop ;

6 precedence rnd

: numin \ addr --. Accepts numeric input and stores in addr.
        begin
                ." ? " inline BL word number dup
                1 <> if
                        ." Invalid input." cr swap drop
                then
        1 = until swap ! ;

: inkey \ -- n. Scan the keyboard and return the ASCII value of the key.
        key ;
```

```
8 constant width        \ The width of the integer display field.

: ,. \ n --. Display n at the right-hand end of width spaces.
        width .r space ;

: ;. \ n --. Display n with no following space.
        0 .r ;

: (;print)
        [ ' cr ] literal 1 [ ' ;. ] literal 1 here last; ! ;

: (,print)
        [ ' cr ] literal 1 [ ' ,. ] literal 1 here last, ! ;

: (input)
        [ ' numin ] literal 1 ;

in definitions

R: "
        [compile] ." R; immediate

R: ,
        endexpr address (input) R; immediate

R: ;
        R; immediate

basic

out definitions

R: "
        2drop [compile] ." R; immediate

R: ,
        endexpr -2 allot value (,print) R; immediate

R: ;
        endexpr -2 allot value (;print) R; immediate

basic definitions

R: let          statement address basic R;                          immediate
R: goto         statement (goto) basic R;                           immediate
R: gosub        statement (gosub) basic R;                          immediate
R: return       statement (return) basic R;                         immediate
R: if           statement value logic R;                            immediate
R: then         endexpr (then) address nul R;                       immediate
R: :            nul R;                                              immediate
R: for          [compile] let R;                                    immediate
R: to           endexpr (to) basic R;                               immediate
R: step         endexpr -2 allot basic R;                           immediate
R: next         statement [ ' (next) ] literal 1
                address basic R;                                    immediate
R: input        statement address in (input) R;                     immediate
R: print        statement value out (,print) R;                     immediate
R: stop         statement (return) basic R;                         immediate
R: end          statement 2drop [compile] R; basic R;               immediate

logic definitions

R: <> \ n m -- f. True if n is not equal to m.
        = not R;

R: <= \ n m -- f. True if n is smaller than or equal to m.
        > not R;

R: >= \ n m -- f. True if n is greater then or equal to m.
        < not R;

2 precedence <>         2 precedence <=         2 precedence >=
2 precedence =          2 precedence >          2 precedence <
1 precedence and        1 precedence or

basic definitions

R: = \ addr n --. Assign n to the variable at address addr.
        swap ! R;
```

```
R: ** \ n m -- n. Raise n to the power of m ( n ^ m ).
        1 swap 1 do over * loop * R;

6 precedence abs
5 precedence **
4 precedence *  4 precedence /  4 precedence */
3 precedence +  3 precedence -
1 precedence =

R: program \ -- cccc. Declare a BASIC program.
        cr initsymbol address 0 #( ! [compile] R: basic 0 0 R;
```

```
tile definitions
decimal

200 constant asize
200 constant bsize

variable asp
variable bsp

variable aframe
variable bframe

variable gdp
variable glast

variable #vars
variable #locals

here  asize 2 * allot constant abase
here  bsize 2 * allot constant bbase

: a0
        abase asp ! ;

: b0
        bbase bsp ! ;

a0 asp @ aframe !
b0 bsp @ bframe !

: ?afull
        asize 2 * asp @ > ;

: ?bfull
        bsize 2 * bsp @ > ;

: ?aempty
        asp @ abase = ;

: ?bempty
        bsp @ bbase = ;

: >a
        ?afull if
                ." Alpha stack overflow." cr abort
        else
                asp @ ! 2 asp +!
        then ;

: a>
        ?aempty if
                ." Alpha stack underflow." cr abort
        else
                -2 asp +! asp @ @
        then ;

: >b
        ?bfull if
                ." Beta stack overflow." cr abort
        else
                bsp @ ! 2 bsp +!
        then ;

: b>
        ?bempty if
                ." Beta stack underflow." cr abort
        else
                -2 bsp +! bsp @ @
        then ;

: a@
        1- 2 * aframe @ + @ ;

: b
        1- 2 * bframe @ + ;

: <closure
        aframe @ >a asp @ aframe !
        bframe @ >b bsp @ bframe ! ;
```

```
: closure>
        aframe @ asp ! a> aframe !
        bframe @ bsp ! b> bframe ! ;


: <local
        here gdp !
        last glast !
        400 DP +! ;

: local>
        gdp @ DP ! ;

: remove
        glast @ CONTEXT @ ! ;

: stk>a
        #vars @ dup 1 > if
                 1 do compile >a loop
        else
                drop then ;

: 0>b
        #locals @ dup 1 > if
                 1 do compile lit  0 , compile >b loop
        else
                drop then ;

: def
        [compile] R: <local 1 #vars ! 1 #locals ! ; immediate

: {
        local> compile <closure stk>a 0>b ; immediate

: }
        compile closure> [compile] R; remove ; immediate

: var
        <builds #vars dup @ ,  ++ immediate
        does> @ compile lit #vars @ , compile lit ,
        compile - compile a@ ; immediate

: local
        <builds #locals dup @ , ++ immediate
        does> @ compile lit , compile b ; immediate

: (
        ?comp -2 allot here @ 5 ; immediate

: )
        ?comp 5 ?pairs , ; immediate
```

```
: cfa->nfa
        2+ nfa ;

: cfa.
        OUT @ C/L > if cr then cfa->nfa id. space ;

: dolit
        dup @ cfa. 2+ dup @ . 2+ ;

: do(.")
        dup @ cfa. 2+ dup count 34 emit type 34 emit space dup C@ + 1+ ;

: docolon
        begin
                dup @ [ ' lit ] literal = if dolit else
                dup @ [ ' 0branch ] literal = if dolit else
                dup @ [ ' branch ] literal = if dolit else
                dup @ [ ' (loop) ] literal = if dolit else
                dup @ [ ' (+loop) ] literal = if dolit else
                dup @ [ ' (.") ] literal = if do(.") else
                dup @ cfa. 2+ then then then then then then
        dup @ 122  =
        until drop ;

: doword
        dup pfa cfa @ 120 = if
                58 emit space dup id. space pfa cfa
                2+ docolon 59 emit
        then ;

: unthread
        -find if
                0 OUT ! doword
        else
                ." Not in context vocabulary." cr
        then drop ;
```