```
User: EN3005                              -at

<USERC>EN>EN3005>PROJECT.LIST

**************************************************************************************************************************
**************************************************************************************************************************


WWWWW W   W  WWW    WWW    WWW  WWWWW
W     WW  W W   W  W   W  W   W W
W     W W W     W  W   W  W   W WWWW
WWWW  W W W   WW   W W W  W W W     W
W     W W W     W  W   W  W   W     W
W     W  WW W   W  W   W  W   W W   W
WWWWW W   W  WWW    WWW    WWW   WWW



WWWW  WWWW   WWW    WWW WWWWW  WWW  WWWWW    W      WWW   WWW  WWWWW
W   W W   W W   W     W W     W   W   W      W       W   W   W   W
W   W W   W W   W     W W     W       W      W       W   W       W
WWWW  WWWW  W   W     W WWWW  W       W      W       W    WWW    W
W     W W   W   W W   W W     W       W      W       W       W   W
W     W  W  W   W W   W W     W   W   W   WW W       W   W   W   W
W     W   W  WWW   WWW  WWWWW  WWW    W   WW WWWWW  WWW   WWW    W

**************************************************************************************************************************
**************************************************************************************************************************

Label: PRT012 -form

Pathname: <USERC>EN>EN3005>PROJECT.LIST
File last modified: 86-09-25.11:13:48.Thu

Spooled:  86-09-25.12:37:56.Thu   [Spooler rev 19.4.5h]
Started:  86-09-25.12:38:56.Thu   on: PRO        by: PR3
```

SOURCE FILE: <USERC>EN>EN3005>PROJECT

```
COMPILED ON: 860925  AT: 11:12  BY: PL1G REV. 19.4.6
Options selected:
     PROJECT -b PROJECT.bin -l PROJECT.list -BIG -LCASE -NESTING -FULL_OPTIMIZE
Optimization note: Currently "-OPTimize" means "-OPTimize 2",
    "-Full_OPTimize" means "-OPTimize 3", and default is "-OPTimize 0".
Options used(* follows those that are not default):
     64V No_Allow_PREconnection BIG* Binary COPy No_DeBuG No_ERRList ERRTty
     No_EXPlist No_FRN Listing* MAp NEsting* No_OFFset OPTimize(3)* No_OVerFlow
     No_PRODuction No_RAnge SIlent(0) TIME No_STATistics Store_Owner_Field
     LCase* No_XRef

         0
         0
         0      /*
    *    0      ===============================================================================
    *    0      = UTILITY PROGRAMME FOR THE MINIMISATION OF BOOLEAN FUNCTIONS                 =
    *    0      = Written by R.C.LUCKHURST, September 1986                                    =
    *    0      = for final year BSc Electrical Engineering project, Bristol Polytechnic      =
    *    0      ===============================================================================
    *    0      */
         0
         0
         0      %REPLACE true BY '1'B, false BY '0'B;
         0      %REPLACE low BY 1, high BY 2, cost BY 3, status BY 4;
         0      %REPLACE redundant BY 1, min_cost_redundant BY 2, non_essential BY 3, min_cost_essential BY 4, essential BY 5;
         0
         0      BSc_project: PROCEDURE OPTIONS (MAIN);
         1
         1      DECLARE (BINARY, BIT, CEIL, CHARACTER, COPY, DECIMAL, INDEX,
         1              LENGTH, LOG2, MIN, SUBSTR, TRANSLATE, VERIFY) BUILTIN,
         1
         1              (num_minterms, num_dont_cares, num_terms, num_pis, num_ne_pis, num_inepi_sums,
         1              minterm (512), dont_care (512), term (1024), p_i (4,256), ne_pi (96),
         1              function_order, solution_cost) FIXED BINARY,
         1
         1              (unique_solution, new_data,
         1              epi_covers_minterm (256), pi_covers_minterm (256,256)) BIT STATIC,
         1              inepi_sum (3000) BIT (96) ALIGNED,
         1
         1              version CHARACTER (4) STATIC INITIAL ('V1.0'),
         1              continue CHARACTER (30) VARYING,
         1              pi_status (5) CHARACTER(30) VARYING STATIC INITIAL
         1              ('redundant','minimum-cost redundant','non-essential','minimum-cost essential','essential'),
         1              results_file FILE;
         1
         1
         1
         1      /*
    *    1      ###############################################################################
    *    1      #                           UTILITY PROCEDURES                                #
    *    1      ###############################################################################
    *    1      */
         1
         1
         1      /******************************************************************************
    *    1      * PROCEDURE equivalent: Returns logical equivalence between 2 integers        *
    *    1      ******************************************************************************/
         1      equivalent: PROCEDURE (x,y) RETURNS (FIXED BINARY);
         2           DECLARE (x,y) FIXED BINARY;
         2           RETURN (x & y  !  ^ x & ^ y);
```

```
        2     END equivalent;
        1
        1
        1     /**************************************************************************
   *    1     * PROCEDURE trim: Returns integer with no leading spaces                  *
   *    1     **************************************************************************/
        1     trim: PROCEDURE (value) RETURNS (CHARACTER (10) VARYING);
        2          DECLARE value FIXED BINARY;
        2          RETURN (SUBSTR(CHARACTER(value),VERIFY(CHARACTER(value),' ')));
        2     END trim;
        1
        1
        1     /**************************************************************************
   *    1     * PROCEDURE sort_data: Sorts minterms and don't cares into ascending order   *
   *    1     *                      and deletes duplicate terms and terms out of range    *
   *    1     **************************************************************************/
        1     sort_data: PROCEDURE;
        2          DECLARE (i, j, t) FIXED BINARY, (b, sorted, excess_terms, type (512)) BIT ALIGNED;
        2
        2          /* first make an all-term list */
        2          DO t = 1 TO num_minterms;
        3               term(t) = minterm(t);
        3               type(t) = true; /* ie minterm */
        3               END;
        2          DO t = 1 TO num_dont_cares;
        3               term(num_minterms + t) = dont_care(t);
        3               type(num_minterms + t) = false; /* ie dont-care */
        3               END;
        2          num_terms = num_minterms + num_dont_cares;
        2
        2          /* then sort into ascending order */
        2          excess_terms = true;
        2          DO WHILE (excess_terms);
        3               excess_terms = false; sorted = false;
        3               DO i = num_terms TO 1 BY -1 WHILE (^ (sorted ¦ excess_terms));
        4                    /* erase terms which are out of range */
        4                    IF term(i) < 0 ¦ term(i) > 255 THEN DO;
        5                         term(i) = term(num_terms);
        5                         type(i) = type(num_terms);
        5                         num_terms = num_terms - 1;
        5                         excess_terms = true;
        5                         END;
        4                    sorted = true;
        4                    DO j = 1 TO i - 1 WHILE (^ excess_terms);
        5                         IF term(j) < 0 THEN sorted = false;
        5                         /* if terms not in ascending order then swap them */
        5                         IF term(j) > term(j + 1) THEN DO;
        6                              t = term(j); term(j) = term(j + 1); term(j + 1) = t;
        6                              b = type(j); type(j) = type(j + 1); type(j + 1) = b;
        6                              sorted = false;
        6                              END;
        5                         /* erase duplicate terms and give minterm priority */
        5                         ELSE IF term(j) = term(j + 1) THEN DO;
        6                              type(j) = (type(j) ¦ type(j + 1));
        6                              term(j + 1) = term(num_terms);
        6                              type(j + 1) = type(num_terms);
        6                              num_terms = num_terms - 1;
        6                              excess_terms = true;
        6                              END;
        5                         END;
```

```
               END;
          END;

     /* now extract sorted terms back into ordered minterm & dont-care arrays */
     num_minterms = 0;  num_dont_cares = 0;
     DO t = 1 TO num_terms;
          IF type(t) THEN DO;
               num_minterms = num_minterms + 1;
               minterm(num_minterms) = term(t);
               END;
          ELSE DO;
               num_dont_cares = num_dont_cares + 1;
               dont_care(num_dont_cares) = term(t);
               END;
          END;
END sort_data;


/*
###############################################################################
#                              INPUT PROCEDURES                               #
###############################################################################
*/


/*****************************************************************************
* PROCEDURE menu_selection: Returns menu item requested: 1 - 5               *
*****************************************************************************/
menu_selection: PROCEDURE RETURNS (FIXED BINARY);
     DECLARE menu_item CHARACTER (30) VARYING, m FIXED BINARY;
     DO WHILE (true);
          GET LIST (menu_item);
          IF VERIFY(menu_item,'12345') = 0 THEN DO;
               m = BINARY(menu_item);
               IF m >= 1 & m <= 5 THEN RETURN (m);
               END;
          END;
END menu_selection;


/*****************************************************************************
* PROCEDURE continue_prompt: Stops screen scrolling                          *
*****************************************************************************/
continue_prompt: PROCEDURE;
     PUT SKIP(2) LIST ('Press RETURN to continue -->');
     GET LIST (continue);
END continue_prompt;


/*****************************************************************************
* PROCEDURE enter_data: Used to enter minterms and don't cares               *
*****************************************************************************/
enter_data: PROCEDURE;
     DECLARE action CHARACTER (30) VARYING, deleted BIT ALIGNED, (i, t) FIXED BINARY;

     get_input_list: PROCEDURE;
          DECLARE (upper, lower) FIXED BINARY,
                  input_item CHARACTER (30) VARYING,
                  illegal_entry BIT ALIGNED;

```

```
    num_terms = 0;
    illegal_entry = false;
    PUT EDIT ('Enter values in the range 0 to 255 seperated by commas or blanks or returns.',
                    'A range of values may be entered using a hyphen, e.g. 10-15.',
                    'Type E after the last entry.',
                    '--> ') (SKIP,A);
    DO WHILE (true);
         GET LIST (input_item);
         IF TRANSLATE(input_item,'E','e') = 'E' THEN RETURN;
         ELSE IF VERIFY(input_item,'-0123456789') = 0 & INDEX(input_item,'-') ^= 1 THEN DO;
               t = INDEX(input_item,'-');
               IF t = 0 THEN DO;
                    IF num_terms <= 255 THEN DO;
                         num_terms = num_terms + 1;
                         term(num_terms) = BINARY(input_item);
                         END;
                    END;
               ELSE IF t <= 5 & LENGTH(input_item) - t < 5 THEN DO;
                    upper = BINARY(SUBSTR(input_item,t + 1)); IF upper > 255 THEN upper = 255;
                    lower = BINARY(SUBSTR(input_item,1,t - 1)); IF lower > 255 THEN lower = 255;
                    IF lower > upper THEN DO; t = lower; lower = upper; upper = t; END;
                    DO t = lower TO upper WHILE (num_terms <= 255);
                         num_terms = num_terms + 1;
                         term(num_terms) = t;
                         END;
                    END;
               END;
         ELSE illegal_entry = true;
         END;
     IF illegal_entry THEN PUT SKIP LIST ('Illegal entries have been disregarded.');
END get_input_list;

DO WHILE (true);
PUT SKIP(3);
     CALL sort_data;
     CALL print_header_message(SYSPRINT);
     CALL print_input_data(SYSPRINT);
     PUT SKIP(3) EDIT ('C  = Clear data',
                        'AM = Add Minterms',
                        'DM = Delete Minterms',
                        'AD = Add Don''t cares',
                        'DD = Delete Don''t cares',
                        'E  = End data entry',
                        'Enter C/AM/DM/AD/DD/E --> ') (6(COLUMN(24),A,SKIP),SKIP,COLUMN(24),A);
     action = ' ';
     DO WHILE (VERIFY(action,'ACDEM') ^= 0);
          GET LIST (action);
          action = TRANSLATE(action,'ACDEM','acdem');
          END;
     IF action = 'E' THEN RETURN;
     IF action = 'C' THEN DO;
          num_minterms = 0;  num_dont_cares = 0;
          END;
     IF action = 'AM' | action = 'DM' | action = 'AD' | action = 'DD' THEN DO;
          CALL get_input_list;
          IF action = 'AM' THEN DO;  /* add input list to minterms */
               DO t = 1 TO num_terms;
                    minterm(num_minterms + t) = term(t);
                    END;
               num_minterms = num_minterms + num_terms;
```

```
  5                      END;
  4                 ELSE IF action = 'DM' THEN DO;   /* make minterms contained in i/p list out of range for deletion w
  5                      DO t = 1 TO num_terms;
  6                           deleted = false;
  6                           DO i = 1 TO num_minterms WHILE (^ deleted);
  7                                IF minterm(i) = term(t) THEN DO;
  8                                     minterm(i) = -1;
  8                                     deleted = true;
  8                                     END;
  7                                END;
  6                           END;
  5                      END;
  4                 ELSE IF action = 'AD' THEN DO;   /* add input list to dont-cares */
  5                      DO t = 1 TO num_terms;
  6                           dont_care(num_dont_cares + t) = term(t);
  6                           END;
  5                      num_dont_cares = num_dont_cares + num_terms;
  5                      END;
  4                 ELSE DO;  /* make dont-cares contained in i/p list out of range for deletion when sorted */
  5                      DO t = 1 TO num_terms;
  6                           deleted = false;
  6                           DO i = 1 TO num_dont_cares WHILE (^ deleted);
  7                                IF dont_care(i) = term(t) THEN DO;
  8                                     dont_care(i) = -1;
  8                                     deleted = true;
  8                                     END;
  7                                END;
  6                           END;
  5                      END;
  4                 IF num_terms > 0 THEN new_data = true;
  4                 END;
  3            END;
  2  END enter_data;
  1
  1
  1  /*
* 1  ##############################################################################
* 1  #                       MINIMISATION PROCEDURES                              #
* 1  ##############################################################################
* 1  */
  1
  1
  1  /****************************************************************************
* 1  * PROCEDURE prime_implicants:  Generates complete list of PIs               *
* 1  ****************************************************************************/
  1  prime_implicants: PROCEDURE;
  2
  2       DECLARE (i, j, term_i, term_j, i_eqv_j, vertex, p, m) FIXED BINARY,
  2               (all_vertices_contained, covered) BIT ALIGNED;
  2
  2       /* generate the prime implicants */
  2       num_pis = 0;
  2       DO i = 1 TO num_terms;
  3            DO j = num_terms TO i BY -1;
  4                 /* choose the pair (i,j) */
  4                 term_i = term(i); term_j = term(j);
  4                 /* is (i,j) a cell? */
  4                 IF (term_i & term_j) = term_i THEN DO;
  5                      /* are all the vertices of (i,j) in the function? */
  5                      i_eqv_j = equivalent(term_i,term_j);
```

```
   5                        all_vertices_contained = true; m = i + 1;
   5                        DO vertex = term_i + 1 TO term_j - 1 WHILE (all_vertices_contained);
   6                             IF (i_eqv_j & vertex) = term_i THEN DO;
   7                                  DO WHILE (term(m) < vertex); m = m + 1; END;
   7                                  all_vertices_contained = (term(m) = vertex);
   7                                  m = m + 1;
   7                                  END;
   6                             END;
   5                        IF all_vertices_contained THEN DO;
   6                             /* is (i,j) covered by an entry in the p.i. table? */
   6                             covered = false;
   6                             IF num_pis ^= 0 THEN DO p = 1 TO num_pis WHILE (^ covered);
   7                                  IF term_j <= p_i(high,p) THEN
   7                                       IF (p_i(low,p) & term_i) = p_i(low,p) THEN
   7                                            covered = ((term_j & p_i(high,p)) = term_j);
   7                                  END;
   6                             IF  ^ covered THEN DO;
   7                                  num_pis = num_pis + 1;
   7                                  p_i(low,num_pis) = term_i;
   7                                  p_i(high,num_pis) = term_j;
   7                                  END;
   6                             END;
   5                        END;
   4                   END;
   3              END;
   2
   2     END prime_implicants;
   1
   1
   1     /***********************************************************************
*  1     * PROCEDURE p_i_chart: Makes a PI chart as a bit array                 *
*  1     ***********************************************************************/
   1     p_i_chart: PROCEDURE;
   2          DECLARE (m, p) FIXED BINARY;
   2          /* generate the prime implicant chart */
   2          DO m = 1 TO num_minterms;
   3               DO p = 1 TO num_pis;
   4                    pi_covers_minterm(p,m) = ((p_i(low,p) & minterm(m)) = p_i(low,p)
   4                                            & (minterm(m) & p_i(high,p)) = minterm(m));
   4                     END;
   3                END;
   2     END p_i_chart;
   1
   1
   1     /***********************************************************************
*  1     * PROCEDURE p_i_status: Categorises PIs as essential/nonessential/redundant  *
*  1     ***********************************************************************/
   1     p_i_status: PROCEDURE;
   2          DECLARE (m, p, epi, num_covers) FIXED BINARY;
   2
   2          /* initialise all p.i. status to redundant */
   2          DO p = 1 TO num_pis; p_i(status,p) = redundant; END;
   2          /* find essential p.i.s */
   2          DO m = 1 TO num_minterms;
   3               num_covers = 0;
   3               DO p = 1 TO num_pis;
   4                    IF pi_covers_minterm(p,m) THEN DO;
   5                         epi = p;
   5                         num_covers = num_covers + 1;
   5                         END;
```

```
                    END;
               IF num_covers = 1 THEN p_i(status,epi) = essential;
               END;
          /* find minterms covered by essential p.i.s */
          num_covers = 0;
          DO m = 1 TO num_minterms;
               epi_covers_minterm(m) = false;
               DO p = 1 TO num_pis WHILE (^ epi_covers_minterm(m));
                    IF p_i(status,p) = essential & pi_covers_minterm(p,m) THEN DO;
                         epi_covers_minterm(m) = true;
                         num_covers = num_covers + 1;
                         END;
                    END;
               END;
          /* determine whether 1 solution or more */
          unique_solution = (num_covers = num_minterms);
          /* find non-essential p.i.s */
          IF ^ unique_solution THEN DO;
               DO m = 1 TO num_minterms;
                    IF ^ epi_covers_minterm(m) THEN DO p = 1 TO num_pis;
                         IF p_i(status,p) = redundant & pi_covers_minterm(p,m) THEN p_i(status,p) = non_essential;
                         END;
                    END;
               /* make a table of n.e.p.i. pointers */
               num_ne_pis = 0;
               DO p = 1 TO num_pis;
                    IF p_i(status,p) = non_essential THEN DO;
                         num_ne_pis = num_ne_pis + 1;  ne_pi(num_ne_pis) = p;
                         END;
                    END;
               END;

     END p_i_status;


     /***************************************************************************
     * PROCEDURE p_i_cost: Calculates literal costs of PIs                      *
     ***************************************************************************/
     p_i_cost: PROCEDURE;
          DECLARE (p, l, b, literals) FIXED BINARY;
          DO p = 1 TO num_pis;
               p_i(cost,p) = 0;  b = 1;
               literals = equivalent(p_i(low,p),p_i(high,p));
               DO l = 1 TO function_order;
                    IF (b & literals) ^= 0 THEN p_i(cost,p) = p_i(cost,p) + 1;
                    b = b + b;
                    END;
               END;
     END p_i_cost;


     /***************************************************************************
     * PROCEDURE irredundand_nepi_sums: Performs algebraic conversion of        *
     *                                  nonessential PI product-of-sums to      *
     *                                  sum-of-products                         *
     ***************************************************************************/
     irredundand_nepi_sums: PROCEDURE;
          DECLARE (m, p, c, s, num_umin_nepis) FIXED BINARY,
               (b, umin_nepis(256)) BIT (96) ALIGNED,
               redundant_sums BIT ALIGNED;
```

```
 2
 2        /* make an array of bit strings holding non-ess p.i. coverage of uncovered minterms */
 2        num_umin_nepis = 0;
 2        DO m = 1 TO num_minterms;
 3             IF ^ epi_covers_minterm(m) THEN DO;
 4                  num_umin_nepis = num_umin_nepis + 1;
 4                  umin_nepis(num_umin_nepis) = 0;
 4                  b = BIT(0,95) || '1'B;
 4                  DO p = 1 TO num_ne_pis;
 5                       IF pi_covers_minterm(ne_pi(p),m) THEN
 5                            umin_nepis(num_umin_nepis) = umin_nepis(num_umin_nepis) | b;
 5                       b = SUBSTR(b,2);
 5                       END;
 4                  END;
 3             END;
 2
 2        /* first pass - i.n.e.p.i. sums are those covering 1st uncovered minterm */
 2        num_inepi_sums = 0;   b = BIT(0,95) || '1'B;
 2        DO p = 1 TO num_ne_pis;
 3             /* if 1st uncovered minterm is covered by thi   n.e.p.i. then ... */
 3             IF (umin_nepis(1) & b) ^= BIT(0,96) THEN DO;
 4                  /* ... this sum is initially this n.e.p.i. */
 4                  num_inepi_sums = num_inepi_sums + 1;   inepi_sum(num_inepi_sums) = b;
 4                  END;
 3             b = SUBSTR(b,2);
 3             END;
 2
 2        /* continue by repeatedly combining with n.e.p.i. terms of succeeding minterms algebraically */
 2        DO m = 2 TO num_umin_nepis;
 3             /* initialise cover counter and n.e.p.i. pointer */
 3             c = -1;   b = BIT(0,95) || '1'B;
 3             /* add each n.e.p.i. covering this minterm successively to each sum */
 3             DO p = 1 TO num_ne_pis;
 4                  /* if this n.e.p.i. covers this minterm then ... */
 4                  IF (umin_nepis(m) & b) ^= BIT(0,96) THEN DO;
 5                       /* ... increment cover counter for this minterm ... */
 5                       c = c + 1;
 5                       /* ... step through the sums for this cover ... */
 5                       DO s = c * num_inepi_sums + 1 TO (c + 1) * num_inepi_sums;
 6                            /* ... make a copy of current sums for next cover ... */
 6                            inepi_sum(s + num_inepi_sums) = inepi_sum(s);
 6                            /* ... add this cover to the sum */
 6                            inepi_sum(s) = inepi_sum(s) | b;
 6                            END;
 5                       END;
 4                  b = SUBSTR(b,2);
 4                  END;
 3             /* calculate the new number of sums resulting from above */
 3             num_inepi_sums = (c + 1) * num_inepi_sums;
 3             /* some sums may cover others so minimise by nulling redundant sums */
 3             redundant_sums = false;
 3             DO s = 1 TO num_inepi_sums;
 4                  IF inepi_sum(s) ^= BIT(0,96) THEN DO c = 1 TO num_inepi_sums;
 5                       IF c ^= s & inepi_sum(c) ^= BIT(0,96) & (inepi_sum(s) & inepi_sum(c)) = inepi_sum(s) THEN DO;
 6                            inepi_sum(c) = BIT(0,96);
 6                            redundant_sums = true;
 6                            END;
 5                       END;
 4                  END;
 3             /* remove redundant sums and calculate the new number of sums resulting */
```

```
      3                IF redundant_sums THEN CALL remove_redundant_sums;
      3                END;
      2
      2      END irredundant_nepi_sums;
      1
      1
      1      /*************************************************************************
 *    1      * PROCEDURE minimum_cost_solution: Finds set of minimum literal cost      *
 *    1      *                                   nonessential PI sums                 *
 *    1      *************************************************************************/
      1      minimum_cost_solution: PROCEDURE;
      2           DECLARE (s, min_cost, sum_cost(1000)) FIXED BINARY,
      2                   redundant_sums BIT ALIGNED;
      2
      2           /* make a table of irredundant n.e.p.i. literal costs */
      2           DO s = 1 TO num_inepi_sums;
      3                sum_cost(s) = nonessential_cost(s);
      3                END;
      2           /* find the minimum cost */
      2           min_cost = sum_cost(1);
      2           DO s = 2 TO num_inepi_sums;
      3                IF sum_cost(s) < min_cost THEN min_cost = sum_cost(s);
      3                END;
      2           /* remove all but minimum cost sums */
      2           DO s = 1 TO num_inepi_sums;
      3                IF sum_cost(s) > min_cost THEN DO;
      4                     inepi_sum(s) = BIT(0,96);
      4                     redundant_sums = true;
      4                     END;
      3                END;
      2           IF redundant_sums THEN CALL remove_redundant_sums;
      2
      2      END minimum_cost_solution;
      1
      1
      1      /*************************************************************************
 *    1      * PROCEDURE ammend_p_i_status: Recategorises some nonessential PIs as     *
 *    1      *                               minimum cost essential/minimum cost redundant *
 *    1      *************************************************************************/
      1      ammend_p_i_status: PROCEDURE;
      2           DECLARE (ess_pis, red_pis, b) BIT (96) ALIGNED,
      2                   s FIXED BINARY;
      2
      2           /* find n.e.p.i.s common to each sum and those which have been removed */
      2           ess_pis = inepi_sum(1);
      2           red_pis = ^ ess_pis;
      2           DO s = 2 TO num_inepi_sums;
      3                ess_pis = ess_pis & inepi_sum(s);
      3                red_pis = red_pis & ^ inepi_sum(s);
      3                END;
      2           /* remove common n.e.p.i.s from the sums - these are minimum-cost essential */
      2           DO s = 1 TO num_inepi_sums;
      3                inepi_sum(s) = inepi_sum(s) & ^ ess_pis;
      3                END;
      2           /* ammend p.i. status table to show minimum-cost essential/redundant p.i.s */
      2           b = BIT(0,95) || '1'B;
      2           DO s = 1 TO num_ne_pis;
      3                IF (ess_pis & b) ^= BIT(0,96) THEN p_i(status,ne_pi(s)) = min_cost_essential;
      3                ELSE IF (red_pis & b) ^= BIT(0,96) THEN p_i(status,ne_pi(s)) = min_cost_redundant;
      3                b = SUBSTR(b,2);
```

```
     3               END;
     2
     2     END ammend_p_i_status;
     1
     1
     1     /***************************************************************************
 *   1     * PROCEDURE remove_redundant_sums: Cleans up irredundant nonessential PI      *
 *   1     *                                   sum-of-products array                    *
 *   1     ***************************************************************************/
     1     remove_redundant_sums: PROCEDURE;
     2          DECLARE (i, j) FIXED BINARY, sum_moved BIT ALIGNED;
     2
     2          DO i = 1 TO num_inepi_sums;
     3               IF inepi_sum(i) = BIT(0,96) THEN DO;
     4                    sum_moved = false;
     4                    DO j = i + 1 TO num_inepi_sums WHILE (^ sum_moved);
     5                         IF inepi_sum(j) ^= BIT(0,96) THEN DO;
     6                              inepi_sum(i) = inepi_sum(j);  inepi_sum(j) = BIT(0,96);
     6                              sum_moved = true;
     6                              END;
     5                         END;
     4                    IF ^ sum_moved THEN DO;
     5                         num_inepi_sums = i - 1;
     5                         RETURN;
     5                         END;
     4                    END;
     3               END;
     2
     2     END remove_redundant_sums;
     1
     1
     1     /***************************************************************************
 *   1     * PROCEDURE essential_cost: Returns literal cost of all essential PIs          *
 *   1     ***************************************************************************/
     1     essential_cost: PROCEDURE RETURNS (FIXED BINARY);
     2          DECLARE (p, e_cost) FIXED BINARY;
     2          e_cost = 0;
     2          DO p = 1 TO num_pis;
     3               IF p_i(status,p) > non_essential THEN e_cost = e_cost + p_i(cost,p);
     3               END;
     2          RETURN (e_cost);
     2     END essential_cost;
     1
     1
     1     /***************************************************************************
 *   1     * PROCEDURE nonessential_cost: Returns literal cost of all nonessential PIs  *
 *   1     *                              in specified sum-of-product sum              *
 *   1     ***************************************************************************/
     1     nonessential_cost: PROCEDURE (s) RETURNS (FIXED BINARY);
     2          DECLARE (s, p, ne_cost) FIXED BINARY,
     2                  b BIT (96) ALIGNED;
     2          ne_cost = 0;   b = BIT(0,95) || '1'B;
     2          DO p = 1 TO num_ne_pis;
     3               IF (inepi_sum(s) & b) ^= BIT(0,96) THEN ne_cost = ne_cost + p_i(cost,ne_pi(p));
     3               b = SUBSTR(b,2);
     3               END;
     2          RETURN (ne_cost);
     2     END nonessential_cost;
     1
     1
```

```
      1     /*************************************************************************
  *   1     * PROCEDURE run_minimisation: Performs minimisation of switching function    *
  *   1     **************************************************************************/
      1     run_minimisation: PROCEDURE;
      2          function_order = LOG2(term(num_terms)) + 1;
      2          PUT SKIP LIST ('(finding prime implicants)');
      2          CALL prime_implicants;  /* generates complete set of prime implicants */
      2          CALL p_i_chart;  /* generates array of pi coverage of minterms */
      2          CALL p_i_status;  /* gives ess/noness/red status to p.i.s & e.p.i. cover status to minterms & decides if uni
      2          CALL p_i_cost;  /* finds literal costs of p.i.s */
      2          IF ^ unique_solution THEN DO;
      3               PUT SKIP LIST ('(finding minimum cost solution)');
      3               CALL irredundant_nepi_sums;  /* generate irredundant n.e.p.i. sums to cover remaining minterms */
      3               CALL minimum_cost_solution;  /* finds lowest literal cost solutions from irredundant n.e.p.i. sums */
      3               CALL ammend_p_i_status;  /* gives min-cost-ess/min-cost-red status to n.e.p.i.s */
      3               END;
      2          solution_cost = essential_cost();
      2          IF ^ unique_solution THEN solution_cost = solution_cost + nonessential_cost(1);
      2     END run_minimisation;
      1
      1
      1     /*
  *   1     ##############################################################################
  *   1     #                           OUTPUT PROCEDURES                                #
  *   1     ##############################################################################
  *   1     */
      1
      1
      1     /*************************************************************************
  *   1     * PROCEDURE print_header_message: Prints title and version no to screen/file *
  *   1     **************************************************************************/
      1     print_header_message: PROCEDURE (f);
      2          DECLARE f FILE VARIABLE;
      2          PUT FILE (f) EDIT ('BOOLEAN MINIMISATION  ', version, COPY('=',26))
      2               (COLUMN(20), A, A, SKIP, COLUMN(20), A);
      2     END print_header_message;
      1
      1
      1     /*************************************************************************
  *   1     * PROCEDURE print_menu: Prints programme menu                                *
  *   1     **************************************************************************/
      1     print_menu: PROCEDURE;
      2          CALL print_header_message(SYSPRINT);
      2          PUT SKIP(3) LIST ('  A utility for the logical minimisation of boolean functions.');
      2          PUT SKIP(4) EDIT ('Menu',
      2                            '----',
      2                        '1.   Enter data',
      2                        '2.   Minimise',
      2                        '3.   File results',
      2                        '4.   Information',
      2                        '5.   Quit',
      2                        'Enter 1-5 --> ')
      2               (COLUMN(28), A, SKIP, COLUMN(28), A, SKIP(2), 5(COLUMN(24), A, SKIP), SKIP, COLUMN(24), A);
      2     END print_menu;
      1
      1
      1     /*************************************************************************
  *   1     * PROCEDURE print_input_data: Prints minterms and don't cares to screen/file *
  *   1     **************************************************************************/
      1     print_input_data: PROCEDURE (f);
      2          DECLARE f FILE VARIABLE, t FIXED BINARY;
      2          /* list the minterms */
      2          PUT FILE(f) SKIP(3) LIST ('Minterms:');
      2          IF num_minterms = 0 THEN PUT FILE(f) SKIP LIST ('*** none ***');
      2          ELSE PUT FILE(f) SKIP EDIT ((trim(minterm(t)) DO t = 1 TO num_minterms)) (A,X(1));
      2          /* list the dont-cares */
      2          PUT FILE(f) SKIP(3) LIST ('Don''t cares:');
      2          IF num_dont_cares = 0 THEN PUT FILE(f) SKIP LIST ('*** none ***');
      2          ELSE PUT FILE(f) SKIP EDIT ((trim(dont_care(t)) DO t = 1 TO num_dont_cares)) (A,X(1));
      2     END print_input_data;
      1
      1
      1     /*************************************************************************
  *   1     * PROCEDURE output_results: Prints minimisation results to screen/file      *
  *   1     **************************************************************************/
      1     output_results: PROCEDURE (f);
      2          DECLARE f FILE VARIABLE;
      2
      2          /* print header message */
      2          CALL print_header_message(f);
      2
      2          /*list minterms and dont-cares */
      2          CALL print_input_data(f);
      2
      2          /* print the function order */
      2          PUT FILE(f) SKIP(3) EDIT ('The function order is ',trim(function_order)) (A);
      2
      2          /* list the prime implicants and associated qualities */
      2          cell: PROCEDURE (p) RETURNS (CHARACTER (10) VARYING);
      3               DECLARE p FIXED BINARY;
      3               RETURN (trim(p_i(low,p)) ¦¦ ',' ¦¦ trim(p_i(high,p)));
      3          END cell;
      2          literals: PROCEDURE (p) RETURNS (CHARACTER (10) VARYING);
      3               DECLARE (p, l, b) FIXED BINARY, lits CHARACTER (10) VARYING;
      3               lits = '';  b = 1;
      3               DO l = 1 TO function_order;
      4                    IF (equivalent(p_i(low,p),p_i(high,p)) & b) ^= 0 THEN
      4                         IF (p_i(low,p) & b) ^= 0 THEN lits = '1' ¦¦ lits;
      4                         ELSE lits = '0' ¦¦ lits;
      4                    ELSE lits = '-' ¦¦ lits;
      4                    b = b + b;
      4                    END;
      3               RETURN (lits);
      3          END literals;
      2          BEGIN;
      3               DECLARE p FIXED BINARY;
      3               PUT FILE(f) SKIP(3) LIST ('Prime Implicants:');
      3               PUT FILE(f) SKIP EDIT (' p.i.','cell','literals','cost','status')
      3                    (A,COLUMN(11),A,COLUMN(25),A,COLUMN(40),A,COLUMN(50),A);
      3               DO p = 1 TO num_pis;
      4                    PUT FILE(f) SKIP EDIT (p,cell(p),literals(p),p_i(cost,p),pi_status(p_i(status,p)))
      4                         (F(4),COLUMN(11),A,COLUMN(25),A,COLUMN(40),F(3),COLUMN(50),A);
      4                    END;
      3               END;
      2
      2          /* abort if no minterms */
      2          IF num_minterms = 0 THEN RETURN;
      2
      2          /* print the prime implicant chart */
      2          tick: PROCEDURE (b) RETURNS (CHARACTER);
```

```
3            DECLARE b BIT;
3            IF b THEN RETURN ('*');
3            RETURN (' ');
3         END tick;
2         BEGIN;
3            DECLARE (m, b, p, num_blocks, mins_per_block) FIXED BINARY;
3            PUT FILE(f) SKIP(3) LIST ('Prime Implicant Chart:');
3            num_blocks = CEIL(DECIMAL(num_minterms) / 19.0);
3            mins_per_block = CEIL(DECIMAL(num_minterms) / DECIMAL(num_blocks));
3            DO b = 1 TO num_blocks;
4               PUT FILE(f) SKIP LIST ('      minterm -->');
4               PUT FILE(f) SKIP EDIT
4                  (' p.i. ',(minterm(m)
4                  DO m = (b - 1) * mins_per_block + 1 TO MIN(num_minterms,b * mins_per_block))) (A,F(3),18(F(4)
4               DO p = 1 TO num_pis;
5                  IF p_i(status,p) > redundant THEN PUT FILE(f) SKIP EDIT
5                     (p,(tick(pi_covers_minterm(p,m))
5                     DO m = (b - 1) * mins_per_block + 1 TO MIN(num_minterms,b * mins_per_block))) (F(4),19(X
5                  END;
4               END;
3            END;
2
2         /* print the solution */
2         BEGIN;
3            DECLARE (p, s) FIXED BINARY, or CHARACTER (2) VARYING, all_covered BIT ALIGNED, b BIT (96) ALIGNED;
3            IF unique_solution THEN PUT FILE(f) SKIP(3) LIST ('Unique Solution:');
3            ELSE PUT FILE(f) SKIP(3) LIST ('Minimum Cost Solution:');
3            PUT FILE(f) SKIP EDIT (' F = ') (A);
3            /* essentials first */
3            or = '';
3            DO p = 1 TO num_pis;
4               IF p_i(status,p) > non_essential THEN DO;
5                  PUT FILE(f) EDIT (or,trim(p)) (A);
5                  or = '+';
5                  END;
4               END;
3            /* if these do not cover all minterms then ... */
3            all_covered = true;
3            IF ^ unique_solution THEN DO s = 1 TO num_minterms WHILE (all_covered);
4               all_covered = false;
4               DO p = 1 TO num_pis WHILE (^ all_covered);
5                  IF p_i(status,p) > non_essential THEN
5                     all_covered = pi_covers_minterm(p,s);
5                  END;
4               END;
3            /* ... minimum cost nonessentials */
3            IF ^ all_covered THEN DO;
4               PUT FILE(f) EDIT (or) (A);
4               DO s = 1 TO num_inepi_sums;
5                  PUT FILE(f) EDIT ('(') (A);
5                  or = '';  b = BIT(0,95) || '1'B;
5                  DO p = 1 TO num_ne_pis;
6                     IF (inepi_sum(s) & b) ^= BIT(0,96) THEN DO;
7                        PUT FILE(f) EDIT (or,trim(ne_pi(p))) (A);
7                        or = '+';
7                        END;
6                     b = SUBSTR(b,2);
6                     END;
5                  PUT FILE(f) EDIT (')') (A);
5                  END;
```

```
      4                    PUT FILE(f) SKIP LIST ('(parenthesised expressions are alternatives)');
      4                    END;
      3               END;
      2
      2          /* print the literal cost of the solution */
      2          PUT FILE(f) SKIP(3) EDIT ('Cost = ',trim(solution_cost),' literals')  (A);
      2
      2     END output_results;
      1
      1
      1     /***********************************************************************
   *  1     * PROCEDURE print information: Prints information about the programme       *
   *  1     ***********************************************************************/
      1     print_information: PROCEDURE;
      2          PUT SKIP(2) EDIT (
      2          '  This utility determines the minimal 2-level solution for a Boolean switching',
      2          'function. This function must be fully specified as minterm and don''t care',
      2          'arrays. The maximum number of input variables is 8. Data is entered as decimal',
      2          'values in the range 0 to 255 in any order. Ranges of values may be entered by',
      2          'using a hyphen, e.g. 10-15. This data is sorted by the programme. If any value',
      2          'is specified as both a minterm and a don''t care term then it is assumed to be',
      2          'a minterm. Values which are out of range are ignored.',
      2          '  Minimisation is done by first finding the prime implicants of the function',
      2          'and then reducing the PI chart. Prime implicants are found by taking pairs of',
      2          'terms (minterms or don''t cares) and testing to see if they form a cell. If they',
      2          'do then a search is made to determine whether all the vertices of the cell are',
      2          'either minterms or don''t cares. If so the cell is tested for containment by',
      2          'any PI already found. If it is not contained then this cell is a PI. PI chart',
      2          'reduction is done using the algebraic method after removing essential PIs and',
      2          'the minterms they cover') (SKIP,A);
      2     END print_information;
      1
      1
      1     /*
   *  1     ##############################################################################
   *  1     #                              MAIN PROGRAMME                               #
   *  1     ##############################################################################
   *  1     */
      1
      1
      1     /* initialisation */
      1     num_minterms = 0;  num_dont_cares = 0;  num_terms = 0;
      1     new_data = true;
      1
      1     /* main loop */
      1     DO WHILE (true);
      2          /* get menu selection */
      2          menu:
      2          PUT SKIP(3);
      2          CALL print_menu;
      2          GO TO menu_option(menu_selection());
      2
      2          menu_option(1): /* enter data */
      2               CALL enter_data;
      2               GO TO menu;
      2
      2          menu_option(2): /* minimise */
      2               IF num_terms < 2 THEN
      2                    PUT SKIP LIST ('Insufficient data - cannot minimise.');
      2               ELSE DO;
```

```
               IF new_data THEN CALL run_minimisation;
               PUT SKIP(2);
               CALL output_results(SYSPRINT);
               new_data = false;
               CALL continue_prompt;
               END;
          GO TO menu;

     menu_option(3): /* file reults */
          IF ^ new_data THEN DO;
               OPEN FILE(results_file) TITLE ('BOOL_MIN -APPEND') LINESIZE(80) STREAM OUTPUT PRINT;
               CALL output_results(results_file);
               PUT PAGE FILE(results_file);
               CLOSE FILE(results_file);
               PUT SKIP LIST ('Results appended to file BOOL_MIN.');
               END;
          ELSE PUT SKIP LIST ('No results to file.');
          GO TO menu;

     menu_option(4): /* information */
          CALL print_information;
          CALL continue_prompt;
          GO TO menu;

     menu_option(5): /* quit */
          STOP;
     END;

END BSc_project;
```