```
:::::::::::::: stationnames ::::::::::::::
#!/bin/sh
#
# stationnames - list names of stations
#

        cut -d";" -f5 sched | cut -d"=" -f2 | grep '[a-z]' | sort -u



:::::::::::::: stationnames.cgi ::::::::::::::
#!/bin/sh
# stationnames.cgi
#    outputs list of station names as plain text
#

        echo "Content-type: text/plain"
        echo ""

        ./stationnames | pr -3 -a -t



:::::::::::::: CommuterRail.html ::::::::::::::
<html>
<head>
       <title>Commuter Rail</title>
       <style type='text/css'>
              A:link { color: darkblue; text-decoration: none; }
              A:visited { color: darkblue;  text-decoration: none; }
              A:hover { text-decoration: underline; color: red; }
       </style>
</head>

<body style='background-color: silver; color: black;
              font-family: sans-serif'>
  <div style='font-weight: bold'>Train Schedule Information</div>

  <div style='margin-left: 30px'>
    <p><a href='train-times.html'>Find train times for a station</a></p>
    <p><a href='trainsched.html'>Find schedule for a train</a></p>
    <p><a href='stationnames.cgi'>List of stations</a></p>
  </div>
</body>
</html>
```

```
:::::::::::::: train-times2.html ::::::::::::::
<html>

<!-- version 2 of the train-times html form
     This one uses a radio button to enforce legal values for
     direction -->

<body>
Find train times for a station
<p>
<form action="train-times.cgi" method="get">
    Station: <input type="text" size="25" name="station">
    <br>
    Direction: <input type=’radio’ name=’dir’ value=’i’>Inbound
               <input type=’radio’ name=’dir’ value=’o’>Outbound
    <br>
    Day:        <input type=’radio’ name=’when’ value=’m-f’>Weekday
                <input type=’radio’ name=’when’ value=’sa’>Saturday
                <input type=’radio’ name=’when’ value=’su’>Sunday
    <br>
    <input type="submit">
</form>
<body></html>


:::::::::::::: train-times.cgi ::::::::::::::
#!/bin/sh
# processing script for train-times.html

    eval ‘./qryparse‘
    echo "Content-type: text/plain"
    echo ""

    echo "Train times for $station direction $dir on $when"
    ./train-times-args "$station" "$dir" "$when"



:::::::::::::: train-times-args ::::::::::::::
#!/bin/sh
#
# train-times-args
#    purpose: list train times for a station
#      usage: train-times-args stationname direction [day]
#      where: direction is "i" or "o", day is "m-f" or "sa" or "su"
#
        # 1st - check there are two arguments or more
        if test $# -lt 2
        then
                echo "usage: train-times-args station direction [day]"
                exit 1
        fi

        STATION=$1
        DIR=$2
        grep "stn=$STATION" sched | grep "dir=$DIR" | grep "day=$3"
```

```
:::::::::::::: trainsched2.html ::::::::::::::
<html>
<body>
<form action='trainsched2.cgi'>
        <p>Train number please?
        <input type='text' name='trainnumber' size='10'>
        </p> <p>
        Capitalize Names: <input type='radio' name='caps' value='y'>Yes
                          <input type='radio' name='caps' value='n'>No
        </p><p>
        <input type='submit'>
       </p>
</form>
</body>
</html>

:::::::::::::: trainsched2.cgi ::::::::::::::
#!/bin/sh
#
# connector for getting train schedule
# this one uses and if then else to capitalize or not
#
        eval `./qryparse`

        echo "Content-type: text/plain"
        echo ""

        if test "$caps" = "y"
        then
                ./trainsched2 $trainnumber | ./capitalize | expand -8
        else
                ./trainsched2 $trainnumber | expand -8
        fi

:::::::::::::: trainsched2 ::::::::::::::
#!/bin/sh
#
# trainsched2
#   usage: trainsched trainnum
#  action: list all entries for that train and format them some
#

        if test $# != 1
        then
                echo "usage: trainsched trainnumber"
                exit 1
        fi
        grep "TR=$1" sched | cut -d";" -f4,5 | sort -t= -k2n | ./semi2tab3 -t

:::::::::::::: array-func-demo.c ::::::::::::::
#include <stdio.h>

/* array-func-demo.c
 * shows how an int is passed by value, but an array is passed by reference
 */
main()
{
        int     n = 4;
        int     a[10] = { 1, 2, 3, 4, 5 };

        printf("n is %d, a is %d, %d, %d, ... \n", n, a[0],a[1],a[2]);
        func(n,a);
        printf("n is %d, a is %d, %d, %d, ... \n", n, a[0],a[1],a[2]);
}
func(int x, int l[10])
{
        printf("x is %d, l is %d, %d, %d, ... \n", x, l[0],l[1],l[2]);
        x = 100;
        l[0] = 36; l[1] = 49; l[2] = 64;
        printf("x is %d, l is %d, %d, %d, ... \n", x, l[0],l[1],l[2]);
}
```

```
:::::::::::::: splitline.c ::::::::::::::
#include        <stdio.h>

/* :ts=8 */
/*
 * splitline.c
 *
 *    purpose: show one way of splitting a line of text into subsections
 *      input: lines of colon-separated data
 *     output: list format of same data
 *     method: split a line of chars into an array of lines of chars
 *
 *      shows: strings, arrays, functions, loops,
 *
 *      Try setting these values to short values and exceed the limits to see
 *
 *      a) how C organizes memory
 *      b) how C doesn't care if you do dumb things
 *      c) why error checking is not just something you talk about
 */

#define LINELEN         200
#define MAXFIELDS       10
#define MAXFLDLEN       40

#define DELIM           ';'

int
split_line( char orig[], char fields[MAXFIELDS][MAXFLDLEN] )
/*
 * purpose: parse a line of DELIM-separated items into an array of strings
 *    args: a line of DELIM-separted items and an array of char arrays
 *  method: loop through orig copying and copy to target array.  Change
 *          to next row at each colon
 * returns: number of fields found in line
 *    note: no error checking.  See what you can do to break it
 */
{
        int     src_pos,                /* position in source array     */
                cur_fld,                /* item in fields[] array       */
                dest_pos;               /* position in dest array       */

        src_pos = cur_fld = dest_pos = 0;

        while( 1 )
        {
                /*
                 * if end of record, then terminate current string
                 * and advance to next row in fields[] table
                 */

                if ( orig[src_pos] == DELIM || orig[src_pos] == '\0' )

                {
                        fields[cur_fld][dest_pos] = '\0';
                        cur_fld++;
                        dest_pos = 0;
                        if ( orig[src_pos] == '\0' )
                                break;
                }
                else

                /*
                 * not end of record, so copy the char from the source
                 * to the current char position in the current string
                 */
                {
                        fields[cur_fld][dest_pos] = orig[src_pos];
                        dest_pos++;
                }
                src_pos++;
        }
        return cur_fld;
}
```

```
:::::::::::::: fix_sched_order.c ::::::::::::::
#include        <stdio.h>

/*
 *  fixorder.c
 *
 *      purpose: process sched file to ensure consistent field order
 *        input: 'sched file' data
 *       output: same data, but each line has TR, dir, day, TI, stn, Line
 *       method: parse line, output fields in correct order
 *         uses: splitline.c
 */

#define LINELEN         200
#define MAXFIELDS       10
#define MAXFLDLEN       40
#define DELIM           ';'

#define NUM_SCHED_FIELDS        6

int splitline(char [], char [MAXFIELDS][MAXFLDLEN] );

main()
{
        char    input[LINELEN];
        char    data[MAXFIELDS][MAXFLDLEN];

        while( fgets(input, LINELEN, stdin ) )          /* read a line */
        {
                input[ strlen(input) - 1 ] = '\0';      /* trim off newline */

                if ( split_line( input, data ) != NUM_SCHED_FIELDS )
                {
                        fprintf(stderr, "incorrect data: %s\n", input );
                        exit(1);
                }

                fix_order( data , NUM_SCHED_FIELDS );
        }
}

int fix_order( char data[MAXFIELDS][MAXFLDLEN] , int num_rows )
/*
 * print out fields in order: LN, FN, CL, AD
 */
{
        printfld( "TR=",   data , num_rows ); putchar(DELIM);
        printfld( "dir=",  data , num_rows ); putchar(DELIM);
        printfld( "day=",  data , num_rows ); putchar(DELIM);
        printfld( "TI=",   data , num_rows ); putchar(DELIM);
        printfld( "stn=",  data , num_rows ); putchar(DELIM);
        printfld( "Line=", data , num_rows ); putchar('\n');
}

printfld( char tag[], char table[MAXFIELDS][MAXFLDLEN] , int num )
/*
 * find line with tag in table and print that line
 */
{
        int     row;
        int     len = strlen( tag );

        for( row = 0 ; row<num ; row++ )
        {
                if ( strncmp( tag, table[row], len ) == 0 )
                {
                        printf("%s", table[row] );
                        break;
                }
        }
}
```

```
:::::::::::::: readln.c ::::::::::::::
#include       <stdio.h>

/*
 * readln(char buf[], int len, char delim)
 *
 * A function to read from stdin one line.
 *    args: buf[]   an array for the chars
 *          len     size of array
 *          delim   read until one of these
 *  action: read from stdin until len-1 chars
 *          or delim appear.  Put a '\0' in
 *          the array.
 * returns: 0 on EOF, else len+1
 *    Note: stops at EOF
 *    Note: unlike fgets, deletes the delim
 */

int readln(char buf[], int len, char delim)
{
       int     i = 0;
       int     rv = 0;
       int     c;

       /* loop while still space && more chars */
       while( i<len-1 && (c = getchar()) != EOF ){
              rv++;
              if ( c == delim )
                     break;
              buf[i++] = c;
       }
       buf[i] = '\0';
       return rv;
}
:::::::::::::: empties.c ::::::::::::::
#include       <stdio.h>
/*
 *     empties.c
 *             assume lines look like
 *                    XX=stuff;YY=morestuff;..;ZZ=yetmorestuff
 *             print lines that have one or more empty fields
 *             compile with readln.c
 *             Returns 0 for no empties, 1 for found some
 */
#define LINESIZE       512
#define TRUE           1
#define FALSE          0
#define DELIM          ';'
int has_empty(char []);

int main()
{
       char    line[LINESIZE];              /* an array of characters */
       int     rv = 0;               /* passed back to shell         */

       while ( readln( line, LINESIZE, '\n') != 0 )
              if ( has_empty(line) == TRUE ){
                     puts(line);
                     rv = 1;
              }
       return rv;
}
int has_empty(char string[])
/*
 *     looks through the string to see if any fields are empty.
 *     this means that the character after an = is a colon or a newline
 *     return TRUE if any empties are found, else returns FALSE
 */
{
       int     i;             /* use this for indexing */

       for(i=0; string[i] != '\0' ; i++)
              if ( string[i]=='=' && (string[i+1]==DELIM || string[i+1]=='\0'))
                     return TRUE;
       return FALSE;
}
```