The Table Editor for myRIO
ctable2()
V 4.0

The following describes `ctable2()`, a utility program that displays values that are stored in memory, and allows the user to change selected values. The values, with appropriate labels, appear on the LCD display. The user enters values on the keypad.

When `ctable2()` is called, it then runs continually, returning to the calling program only when ← is entered. However, other threads may use and cause to be displayed the information stored by `ctable2()`.

A table "title" is displayed on the first line of the LCD display. The table can have as many as nine numbered entries. Three of these entries are always displayed below the title. The user can scroll the entries up and down using the `UP` and `DWN` keys. Alternately, the user can cause any entry to become the top entry by entering its number.

For example, a three-entry table, shown with the third entry scrolled to the top, might look like:

```
Flow Control Table
3 BTI: ms              3.0
1 Qref:  (cc/s)        450.
2 Qact:  (cc/s)        453.
```

The user may alter an entry by scrolling it to the *top* of the list, and pressing ENTR. The display prompts for a new value of the parameter. For the example above, pressing ENTR would cause the prompt: `Enter: BTI: ms` to be displayed. The user could then enter a new value (followed by `ENTR`), causing the new value to be placed in memory and displayed.

"Edit" values and "Show" values

There are two kinds of values, called "edit" values and "show" values. Edit values are those that the user may change at will. Each edit value is presumed not to have changed since the last time it was changed (edited) by the user.

Show values are those that the user may observe more or less continually. A separate thread periodically updates the table to reflect the current show values. Show values may not be edited; each show value is presumed perhaps to have changed since the last time the table was updated. (The changes would generally be made by another thread, which would determine a new show value and place it in memory; the new value would then be displayed when the table is updated.)

Typically, edit values are system parameters set by the user, while show values are computed and change with time.

Calling ctable2()

The prototype of `ctable2()` is:

```
int ctable2(char *title, struct table *entries, int nval);
```

The `ctable2()` function is automatically linked with your code from the ME477Library. The statement: `#include "ctable2.h"` must appear in `main.c`.

When calling `ctable2()`, your program must supply appropriate values for the following arguments:

title is a string array for the table title.
Less than 20 characters.

entries is an array of structures of type `table` defined as:

```
typedef struct {
    char *e_label;  // entry label label
    int e_type;     // entry type (0-show; 1-edit)
    double value;   // value
} table;
```

Each element of the array corresponds to an entry in the table, and specifies the entry label, type (edit or show), and value of the entry. A good practice is to make the length of the labels 12 characters or less.

nval specifies the number of table entries. Again, the total number of edit and show entries must be no greater than 9.

Entering ← while the table is displayed causes `ctable2()` to terminate, returning `0` for a normal exit.

For example,

In this table entitled: `Flow Control Table`, there are two edit values that can be changed by the user (`qref` and `bti`), and one show value (`qact`).

In the main thread, the variables for the table title, and the table structure array are declared and initialized.

```
char *Table_Title ="Flow Control Table";
table my_table[] = {
    {"Qref: (cc/s)",  1, 0.   },
    {"Qact: (cc/s)",  0, 0.0  },
    {"BTI: ms     ",  1, 5.0  }
 };
```

Notice that the each element of the array `my_table` is a `struct` of type `table` containing the entry label, type, and initial value.

Finally, the table editor is called.

```
ctable2(Table_Title, my_table, 3);
```

Within the thread that uses the table values, pointers corresponding to convenient names of the table values can be declared.

```
double *qref = &((threadResource->a_table+0)->value);
double *qact = &((threadResource->a_table+1)->value);
double *bti  = &((threadResource->a_table+2)->value);
```

Then, variables may be referred to by their named pointers. For example,

```
T = *bti/1000.;
```

Note the dereferencing of the `bti` pointer.