Programming Assignment 1 (PA1) - Display Bowtie

Milestone Due:  Wednesday, April 18 @ 11:59 pm
Final Due:  Wednesday, April 25 @ 11:59 pm

<table>
<thead>
<tr>
<th>Example Input</th>
<th>Milestone Functions</th>
<th>Unit Testing</th>
<th>Extra Credit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detailed Overview</td>
<td>Post-Milestone Functions</td>
<td>README Questions</td>
<td>Turnin Summary</td>
</tr>
</tbody>
</table>

Assignment Overview

The purpose of this programming assignment is to build your knowledge of the ARM assembly language, especially branching and looping logic, calling assembly routines from within a C program, calling C functions from within assembly routines, allocating space for local variables, passing parameters and returning values, using Unix command line arguments, and learning some useful Standard C Library routines.

In this assignment you are writing an interactive program that takes in 4 inputs from the command line and prints a bowtie design to stdout. See man ascii for a map of the ASCII character set. This assignment will require appropriate error checking and reporting (as documented below).

Start early! Remember that you can and should use man in order to lookup information on specific C functions. For example, if you would like to know what type of parameters strtol() takes, or what strtol() does to errno, type man -s3 strtol. Also, take advantage of the tutors in the lab. They are there to help you learn more on your own and help you get through the course!

Grading

- **README: 10 points** - See README Requirements [here](http://cseweb.ucsd.edu/~ricko/CSE30READMEGuidelines.pdf) and questions in README Questions section
- **Compiling: 5 points** - Using our Makefile; no warnings. If what you turn in does not compile with the given Makefile, you will receive 0 points for this assignment. **NO EXCEPTIONS!**
- **Style: 20 points** - See Style Requirements [here](http://cseweb.ucsd.edu/~ricko/CSE30StyleGuidelines.pdf)
- **Correctness: 65 points**
  - **Milestone (15 points)** - Distributed across the Milestone functions
  - Make sure you have all files tracked in Git.
- **Extra Credit: 5 points** - View Extra Credit section for more information.
- **Wrong Language:** You will lose 10 points for each module in the wrong language, C vs. Assembly or vice versa.

**NOTE:** If what you turn in does not compile with given Makefile, you will receive 0 points for this assignment.
Getting Started

Follow these steps to acquire the starter files and prepare your Git repository.

Gathering Starter Files:
The first step is to gather all the appropriate files for this assignment. Connect to pi-cluster via ssh.

$ ssh cs30xyz@pi-cluster.ucsd.edu

Create and enter the pa1 working directory.

$ mkdir ~/pa1
$ cd ~/pa1

Copy the starter files from the public directory.

$ cp ~/../public/pa1StarterFiles/* ~/pa1/

Starter files provided:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pa1.h</td>
<td></td>
</tr>
<tr>
<td>test.h</td>
<td></td>
</tr>
<tr>
<td>palStrings.h</td>
<td></td>
</tr>
<tr>
<td>testInRange.c</td>
<td></td>
</tr>
<tr>
<td>Makefile</td>
<td></td>
</tr>
</tbody>
</table>

Preparing Git Repository:
You are required to use Git with this and all future programming assignments. Refer to the PA0 writeup for how to set up your local git repository.

Example Input

A sample stripped executable provided for you to try and compare your output against is available in the public directory. Note that you cannot copy it to your own directory; you can only run it using the following command (where you will also pass in the command line arguments):

$ ~/.public/paltest

NOTE:

1. The output of your program MUST match exactly as it appears in the paltest output. You need to pay attention to everything in the output, from the order of the error messages to the small things like extra newlines at the end (or beginning, or middle, or everywhere)!

2. **We are not giving you any sample outputs, instead you are provided some example inputs. You are responsible for trying out all functionality of the program; the list of example inputs is not exhaustive or complete. It is important that you fully understand how the program works and you test your final solution thoroughly against the executable.**

Example input that has error output:----------------------------------------------------------------------------------

cs30xyz@pi-cluster-001:pa1$ ./pa1
 cs30xyz@pi-cluster-001:pa1$ ./pa1 a b c d e
 cs30xyz@pi-cluster-001:pa1$ ./pa1 8a 3 4 5
 cs30xyz@pi-cluster-001:pa1$ ./pa1 9999999999999999999 a b c
 cs30xyz@pi-cluster-001:pa1$ ./pa1 8 a b c
 cs30xyz@pi-cluster-001:pa1$ ./pa1 8 ad b cc
Example input that has normal output: 

```bash
cs30xyz@pi-cluster-001:pa1$ ./pal 7 ! ! a b c
cs30xyz@pi-cluster-001:pa1$ ./pal 8 a a a
cs30xyz@pi-cluster-001:pa1$ ./pal 7 a a a
```

These next two cases are slightly different, they provide the ASCII codes as octal or hexadecimal values (which are valid input to the program). Note: We can use backslash “\” escape for using octal and hex ASCII codes as legal characters.

```bash
cs30xyz@pi-cluster-001:pa1$ ./pal 9 ! , @
cs30xyz@pi-cluster-001:pa1$ ./pal 145 a b c
```

Detailed Overview

The function prototypes for the various C and Assembly functions are as follows.

**C routines:**
```c
int main( int argc, char *argv[] );
```

**Assembly routines:**
```assembly
long isInRange( long value, long minRange, long maxRange );
long myRemainder( long dividend, long divisor );
long isEven( long value );
void printChar( char ch );
void printBowtieLine( long row, long height, char filler, char bowtie );
void displayBowtie( long height, char border, char bowtie, char filler );
```

For the Milestone, you will need to complete:

<table>
<thead>
<tr>
<th>InRange.s</th>
<th>myRemainder.s</th>
<th>isEven.s</th>
</tr>
</thead>
</table>

**Process Overview:**
The following is an explanation of the main tasks of the assignment, and how the individual functions work together to form the whole program.

This program takes 4 command line arguments:
```
$ ./pal height border_char filler_char bowtie_char
```

Explanation of Command Line Arguments:
- **height** - the height of the bowtie pattern, **not including the border**
- **border_char** - the character used to draw the border
- **filler_char** - the character used to fill out the parts that doesn’t belong to the bowtie
- **bowtie_char** - the character used to draw the bowtie
Drawing the bowtie pattern consists of the following steps:

1. Parse command line arguments in `main()`, where `isInRange()`, `myRemainder()`, and `isEven()` will help with error checking.
   a. If there are errors, print out the appropriate error messages to `stderr`.
   b. No errors, proceed to print out the bowtie pattern.

2. Draw the bowtie pattern using `displayBowtie()`, where `printBowtieLine()` will be used to print out the correct pattern for each line and `printChar()` will be used to print out the individual characters.

### Milestone Functions to be Written

Listed below are the modules to be written for the milestone.

**isInRange.s**

```c
long isInRange( long value, long minRange, long maxRange );
```

This function determines whether `value` is in the range between `minRange` (inclusive) and `maxRange` (exclusive).

**Reasons for error:**

- `minRange` is greater than or equal to `maxRange` → return -1
Return Value: -1 if an error occurred. Otherwise, return 1 if value is in range and 0 if not in range.

myRemainder.s
long myRemainder( long dividend, long divisor );

This function returns the remainder of the division between the dividend and the divisor; it should act the same as the modulus operator (%) in C (for both negative and positive numbers).

Reasons for error:
● Divisor is zero → result is undefined (we will not be checking for divide by 0)

Return Value: The remainder as a 32-bit signed integer.

isEven.s
long isEven( long value );

This function checks if value is an even number, using your myRemainder() function.

Return Value: 0 if value is odd; 1 if value is even.

Post-Milestone Functions to be Written

Listed below are the modules to be written after the milestone functions are complete.

printChar.s
void printChar( char ch );

This assembly module prints the character argument to stdout (see man -s3 printf). This is very similar to the example assembly module given below. The main difference is that printChar() just prints a single character (so think about how that might affect the format string).

Example: This example assembly module takes in a string that represents a class, and prints out a message saying that that class is your favorite class.
printFavoriteClass("CSE 30") -> "My favorite class is CSE 30"

```
.cpu cortex-a53
.syntax unified
.equ FP_OFFSET, 4
.section .rodata
fmt: .asciz "My favorite class is %s\n"
.global printFavoriteClass
.text
.align 2
```
printFavoriteClass:
    push  {fp, lr}   @ Saves registers on the stack.
    add   fp, sp, FP_OFFSET  @ Adjusts fp to be at bottom of current stack.
                           @ Uses 4 because (#regs_saved - 1) * 4.
    mov   r1, r0   @ Move string parameter to r1 as argument to printf
    ldr  r0, =fmt   @ Gets address of format string.
    bl  printf    @ Calls printf.
    sub  sp, fp, FP_OFFSET
    pop  {fp, pc}   @ Returns from function.

---

displayBowtie.s
void displayBowtie( long height, char border, char filler, char bowtie );

This assembly module will print out individual characters (via calls to printChar() and
printBowtieLine()) such that the bowtie pattern will be displayed based on the user-supplied values.
NOTE: You will have to allocate space on the stack for local variables and formal parameters.

To help you with this module, you may use these offsets
(on the right) for local variables and parameters. Material
in lecture will cover what these offsets are used for in
depth, and in future assignments you will need to
calculate these offsets yourself. Remember that you will
need to load and store from these locations.

.equ  FP_OFFSET, 4
.equ  LOCAL_OFFSET, 16
.equ  PARAM_OFFSET, 16
.equ  ROW_OFF, -8
.equ  COL_OFF, -12
.equ  INNER_OFF, -16
.equ  HEIGHT_OFF, -24
.equ  BORDER_OFF, -28
.equ  FILLER_OFF, -32
.equ  BOWTIE_OFF, -36

Here's the equivalent C version of displayBowtie:
http://cseweb.ucsd.edu/~ricko/pa1/displayBowtie.jpg

This will be a translation task for you. All of the assembly constructs you will be using will have been covered in
lecture and can be referenced in your notes. You are not limited to using the above algorithm, but part of the
purpose of this programming assignment is to learn how to write looping/conditional constructs (branches), use
the simple bl instruction to branch to subroutines with parameter passings, and perform simple arithmetic
instructions (add/sub, sdiv) in assembly.

We would encourage you to use the linked algorithm for these reasons, however we do not want to suppress
creative thinking - alternative solutions are welcome. You must use the "preferred" style of coding loops,
backwards branching logic, as detailed in class: set up an opposite logic branch to jump over the loop body
and a positive logic branch to jump backwards to the loop body. Points will be taken off for not using
backwards branching logic.
printBowtieLine.s
void printBowtieLine( long row, long height, char bowtie, char filler );

This function will be called from displayBowtie() for printing out the line of characters that make up the bowtie pattern (filler chars + bowtie chars; this function does not print out border characters).

NOTE: You will have to allocate space on the stack for local variables (used to store the values of the parameters passed in).

Here’s the equivalent C version:
http://cseweb.ucsd.edu/~ricko/pa1/printBowtieLine.jpg

main.c
int main( int argc, char *argv[] );

The main function will drive the rest of the program. It will first perform input checking by parsing the command-line arguments and checking for errors. If all inputs are valid, it will call displayBowtie(). Otherwise, it will print the corresponding error messages. Remember that all error strings have format specifiers, so be sure to add the appropriate arguments when printing error messages. Also, make sure you use your isInRange() function when checking the limits of the command line arguments.

Note that your variable names should all be in camelCase (e.g. sampleVariableName, not sample_variable_name) - see the style guidelines.

First: check that the user entered the correct number of command line arguments. If they didn’t, print the STR_USAGE and return EXIT_FAILURE right away.

Now we can parse the command line arguments in the following steps:

1. **height**: set the global variable errno to 0 (see man -s3 errno), then use strtol() to convert the bowtie height to a long (see man -s3 strtol). Check for the following errors in the order they appear below. Note that some of the error strings will need STR_HEIGHT as part of the error message.

<table>
<thead>
<tr>
<th>Error</th>
<th>How to Handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>height contains non-numerical characters</td>
<td>Use fprintf() to print the STR_ERR_NOT_INT error message. Skip to step 2.</td>
</tr>
<tr>
<td>(check endptr)</td>
<td></td>
</tr>
<tr>
<td>Error converting to long (errno was set by strtol())</td>
<td>Use snprintf() to build the error string using STR_ERR_CONVERTING (make sure the string is null-terminated). Call perror(), passing this string as a parameter. Skip to step 2.</td>
</tr>
<tr>
<td>height is not in range</td>
<td>Use fprintf() to print the STR_ERR_NUM_RANGE error message.</td>
</tr>
</tbody>
</table>
2. **border_char**: extract the first character from the border_char argument. Check for the following errors in the order they appear below. Note that both of the error strings will need `STR_BORDER_CHAR` as part of the string.

<table>
<thead>
<tr>
<th>Error</th>
<th>How to Handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not a single character (hint: <code>man -s3 strlen</code>)</td>
<td>Print the <code>STR_ERR_SINGLE_CHAR</code> error message. Skip to step 3.</td>
</tr>
<tr>
<td>char not in range (not within <code>[ASCII_MIN, ASCII_MAX]</code>)</td>
<td>Print the <code>STR_ERR_ASCII_RANGE</code> error message. Continue to step 3.</td>
</tr>
</tbody>
</table>

3. **filler_char**: extract the first character from the filler_char argument. Check for the following errors in the order they appear below. Note that both of the error strings will need `STR_FILLER_CHAR` as part of the string.

<table>
<thead>
<tr>
<th>Error</th>
<th>How to Handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not a valid single character (hint: <code>man -s3 strlen</code>)</td>
<td>Print the <code>STR_ERR_SINGLE_CHAR</code> error message. Skip to step 4.</td>
</tr>
<tr>
<td>char not within range (not within <code>[ASCII_MIN, ASCII_MAX]</code>)</td>
<td>Print the <code>STR_ERR_ASCII_RANGE</code> error message. Continue to step 4.</td>
</tr>
</tbody>
</table>

4. **bowtie_char**: extract the first character from the bowtie_char argument. Check for the following errors in the order they appear below. Note that both of the error strings will need `STR_BOWTIE_CHAR` as part of the string.

<table>
<thead>
<tr>
<th>Error</th>
<th>How to Handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not a valid single character (hint: <code>man -s3 strlen</code>)</td>
<td>Print the <code>STR_ERR_SINGLE_CHAR</code> error message. Skip to step 5.</td>
</tr>
<tr>
<td>char not within range (not within <code>[ASCII_MIN, ASCII_MAX]</code>)</td>
<td>Print the <code>STR_ERR_ASCII_RANGE</code> error message. Continue to step 5.</td>
</tr>
</tbody>
</table>

5. If any errors from above were encountered, print a newline and return `EXIT_FAILURE`. Otherwise, make sure:
   a. `border_char` and `bowtie_char` are different from each other. If they are not different, print the `STR_ERR_BORDER_BOWTIE_DIFF` error message, then print a newline, and return `EXIT_FAILURE`.
   b. `filler_char` and `bowtie_char` are different from each other. If they are not different, print the `STR_ERR_FILLER_BOWTIE_DIFF` error message, then print a newline, and return `EXIT_FAILURE`.

6. If no errors were encountered, draw the bowtie pattern by calling the `displayBowtie()` function, passing in the appropriate parameters.

**Return Value:** If errors were encountered, return `EXIT_FAILURE`. Otherwise, return `EXIT_SUCCESS`.
Unit Testing

You are provided with only one basic unit test file for the Milestone function, `isInRange()`. This file only has minimal test cases and is only meant to give you an idea of how to write your own unit test files. **You must write unit test files for each of the Milestone functions, as well as add several of your own thorough test cases to all of the unit test files. You need to add as many unit tests as necessary to cover all possible use cases of each function. You will lose points if you don’t do this!** You are responsible for making sure you thoroughly test your functions. Make sure you think about boundary cases, special cases, general cases, extreme limits, error cases, etc. as appropriate for each function. The Makefile includes the rules for compiling these tests. Keep in mind that your unit tests will not build until all required files for that specific unit test have been written. These test files will be collected with the Milestone; they must be complete before you submit your Milestone.

**Unit tests you need to complete:**
- `testisInRange.c`
- `testmyRemainder.c`
- `testisEven.c`

**To compile:**
```
$ make testisInRange
```

**To run:**
```
$ ./testisInRange
```

(Replace “testisInRange” with the appropriate file names to compile and run the other unit tests)

README Questions

(see the rest of the README requirements [here](#))

**Vim Questions**

1. What Vim command do you use to delete a single line? What about n number of lines?

2. Which Vim command would you use to copy a line? Then which command to “paste” the copied line to after the cursor?

**Git Questions**

3. What Git command is used to show the current state of your working tree?

4. What is the Git command to create a new branch, and switch into that newly created branch?

5. What is the Git command to display differences between the local version of a file and the version last committed?

**GDB Questions**

Start gdb with your pa1 executable, then set a breakpoint at `strtol`. Breakpoints are set with the `break` command (abbreviated `b`):
```
break strtol
```

Note that you may see this message:
Function "strtol" not defined.
Make breakpoint pending on future shared library load? (y or [n])
In this case just hit \texttt{y} to continue.

Run the program (in \texttt{gdb}) with the following command line args:

\begin{verbatim}
run 30Students 5 b c
\end{verbatim}

It's ok if you see an error like \texttt{"strtol.c: No such file or directory"}

Now you should be at the entry point of the Std C Lib routine \texttt{strtol} called from \texttt{main}.

6. How do you print the value of the string that is the 1st arg in \texttt{strtol}? (The value should be "30Students")

7. How do you print the \texttt{hex} value of \texttt{&endptr} that is the 2nd arg in \texttt{strtol}? (The value should be something like \texttt{0x7efff} - a high stack address - will vary)

Go to the next high level source instruction in \texttt{main}. This should be the next C instruction in \texttt{main} after the function call to \texttt{strtol}. Now we want to continue to the next source line in the current (innermost) stack frame, so type \texttt{next}. Again, if you encountered the \texttt{"strtol" not defined} error earlier or \texttt{"in strtol.c"}, you may have to enter \texttt{next} a few times until the code returns to \texttt{main}.

8. How do you print the value returned by \texttt{strtol}? (The value should be \texttt{30}) Show two ways:
   a. Using the name of the local variable you use to hold the return value
   b. Displaying the value in the register used to return the value

9. How do you print the character \texttt{endptr} is pointing to? (Should be the character 'S')

10. How do you print the entire null-terminated string \texttt{endptr} is pointing to? (Should be “Students”)

11. How do you print the decimal value of the global variable \texttt{errno} at this point? (The value should be \texttt{0})

Continue the execution of your \texttt{pa1} in \texttt{gdb}. Type \texttt{continue}, which means resuming program execution until it completes normally. You should hit another breakpoint and should see the error message:

\begin{verbatim}
height(30Students) must be an integer.
\end{verbatim}

Type \texttt{continue} again to end the program.

Run the program again (in \texttt{gdb}) with the following command line args:

\begin{verbatim}
run 11111111111111 5 b c
\end{verbatim}

You should be at the entry point of the Std C Lib routine \texttt{strtol} called from \texttt{main}. Go to the next source instruction in \texttt{main}. Type \texttt{next}. Again, if you encountered the \texttt{"strtol" not defined} error or \texttt{"in strtol.c"}, you may have to enter \texttt{next} a few times until the code returns to \texttt{main}.

It should be the source-level instruction after the call to \texttt{strtol} passing in 11111111111111 to convert to an int. Print the decimal value of \texttt{errno} at this point. The value of \texttt{errno} should be \texttt{34} now which is the value of \texttt{ERANGE}. See the man page for \texttt{errno} and section 2 intro. You can continue or quit.

\textbf{Academic Integrity Question}

12. What was your process for completing this assignment with integrity?

\textbf{Extra Credit}

Overview
There are 5 points total for extra credit on this assignment. You will make a modified version of your displayBowtie program where the number of layers of the border char is equal to the number of digits in the height command line argument. Note that in order to do this extra credit, you must have a working version of your entire PA1 assignment.

**Getting Started**
Copy your regular displayBowtie function to start on the extra credit version

```
$ cp ~/pa1/displayBowtie.s ~/pa1/displayBowtieEC.s
```

**Compiling**
Once you have written your extra credit code, you can compile the extra credit program using the following command from within your pa1 directory.
```
$ make pa1EC
```

**Sample Output:**
```
cs30xyz@pi-cluster-001:~/pa1$ ./pa1EC 11 ! . @
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!@@..................@@!!
!!@@@@..............@@@@!!
!!@@@@@@..........@@@@@@!!
!!@@@@@@@@......@@@@@@@@!!
!!@@@@@@@@@@..@@@@@@@@@@!!
!!@@@@@@@@@@@@@@@@@@@@@@!!
!!@@@@@@@@@@..@@@@@@@@@@!!
!!@@@@@@@@......@@@@@@@@!!
!!@@@@@@..........@@@@@@!!
!!@@@@..............@@@@!!
!!@@..................@@!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

**Extra Credit Assembly Functions To Be Written**
Note that while the names of files to be written are numOfDigitsEC.s and displayBowtieEC.s, the names of the functions should be numOfDigits() and displayBowtie(), respectively.

**numOfDigitsEC.s**
```
long numOfDigits( long num, long base );
```

This function counts the number of digits in num in the given base. You need to check if base is within the limits of 2 (inclusive) and 36 (exclusive) using your isInRange() function.

Note for unit testing: To type hex values, prefix the number with 0x. To type octal values prefix them with 0. C does not allow you to type values in binary, so you will need to convert binary to either decimal, octal, or hex.

For example:
```
<table>
<thead>
<tr>
<th>num</th>
<th>base</th>
<th>Number of digits</th>
</tr>
</thead>
</table>
```
<table>
<thead>
<tr>
<th>base</th>
<th>any</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>-2693</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>0x7A2</td>
<td>16</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>4 ( (8_{10} \equiv 1000_2) )</td>
</tr>
</tbody>
</table>

Error conditions:

- base is not within [2-36) (that is, inclusive of 2, exclusive of 36) → return -1

Return Value: If any errors occur, return -1. Otherwise, return the number of digits in num of the given base.

displayBowtieEC.s

```c
void displayBowtie( long height, char border, char bowtie, char filler );
```

This assembly module is very similar to your displayBowtie function, however, instead of always having 1 layer of border char around the bowtie, the number of layers of border char is the number of digits in the height (use numOfDigits() for this).

Turnin Summary

See the turnin instructions here. Your file names must match the below *exactly* otherwise our Makefile will not find your files.

Milestone Turnin:
Due: Wednesday night, April 18 @ 11:59 pm

Files required for the Milestone:

<table>
<thead>
<tr>
<th>isInRange.s</th>
<th>myRemainder.s</th>
<th>isEven.s</th>
</tr>
</thead>
<tbody>
<tr>
<td>testisInRange.c</td>
<td>testmyRemainder.c</td>
<td>testisEven.c</td>
</tr>
</tbody>
</table>

Final Turnin:
Due: Wednesday night, April 25 @ 11:59 pm

Files required for the Final Turn-in:

<table>
<thead>
<tr>
<th>isInRange.s</th>
<th>main.c</th>
<th>pal.h</th>
</tr>
</thead>
<tbody>
<tr>
<td>myRemainder.s</td>
<td>testisInRange.c</td>
<td>palStrings.h</td>
</tr>
<tr>
<td>isEven.s</td>
<td>testmyRemainder.c</td>
<td>test.h</td>
</tr>
<tr>
<td>displayBowtie.s</td>
<td>testisEven.c</td>
<td>Makefile</td>
</tr>
<tr>
<td>printChar.s</td>
<td>printBowtieLine.s</td>
<td>README</td>
</tr>
</tbody>
</table>

Extra Credit Files:

<table>
<thead>
<tr>
<th>displayBowtieEC.s</th>
<th>numOfDigitsEC.s</th>
</tr>
</thead>
</table>
If there is anything in these procedures which needs clarifying, please feel free to ask any tutor, the instructor, or post on the Piazza Discussion Board.