pottery Design Plan

WO1 Clayton E. Williams

June 2023

Project Structure

In the attempt to begin to build SOPs for future projects, the project will be broken down as such:

```
.
├── data
│   └── Pottery.txt
├── doc
│   ├── design.md
│   ├── potter.1
│   ├── pottery.pdf
│   ├── writeup.adoc
│   └── writeup.pdf
├── include
├── Makefile
├── README.md
├── scratch.c
├── src
│   └── potter.c
└── test
```

`data` - external files that are read as input to the program

`doc` - documentation for the project, such as design plan, man page, writeup, and test plan

`include` - custom header files for various C source code files

`src` - C source code files for the project

`test` - C source code files for unit testing

Objects Needed

The best data structure to use will be of a struct that includes the size, capacity, array of student names, and array of student times to finish the project:

```
typedef struct student {
        int size;
        int capacity;
        char **stu_name;
        int *stu_time;
} student;
```

This provides the ability to pass the struct pointer to various functions and easily maintain its current size and determine when it needs to be resized.

Functions Needed

The basic functions needed will be:

```
get_file_size();
clean();
create_struct();
reallocate_struct();
get_user_input();
create_student();
grade_students();
print_students();
```

`get_file_size` - Identifies number of lines in the data file to allocate arrays to exact length.

`clean` - run at the end of program or at exit to free allocated memory.

`create_struct` - initializes struct.

`reallocate_struct` - resize struct.

`get_user_input` - take user input from command line for manual student entry.

`create_student` - adds student name and time to the respective arrays.

`grade_students` - iterate over students to identify optimal index to begin grading.

`print_students` - print the students who failed based on index returned from `grade_students`.

Project Flow

1. Define struct and write function prototypes
2. Define main as `int main (int argc, char* argv[])` and use argc to determine if a file name was passed to the command line.
3. If a file name is provided, attempt to open (exit if error). Invoke `get_file_size()` to determine number of lines in the program, to malloc student member variables of that size.
4. If no file name is provided, invoke `get_user_input()` that will take user input, line by line.

5. Whether student data is read line by line from command line or provided file name, they are both treated as char pointers taht are passed to `create_student()`.
6. `create_student()` will have logic to call `reallocate_struct()` to increase the size of the struct arrays based on number of students entered from command line. `create_students()` will parse the string and subsequently add to the struct.
7. When the file or user input is exhausted, the struct is fd to `grade_students()`. The general logical flow is to iterate over from 0 to size, and determine if a students time is >= size * 5. After that, the loop needs to run from 1 to size, and 0 to 1, and so on. Each call of the loop will increment a counter of students failed if conditional is met and compare to subsequent loops to find the iteration with the most failed students and save that index.
8. After grading is complete, the index with most failed is passed to `print_students()` that will iterate in a similar manner, this time, printing the students who failed.
9. After this is complete, invoke the `clean()` function to free all allocated heap memory.