Brandon Swanson
Week 2 Assignment Report
CS 165 Summer 2014

UNDERSTANDING

This week had two major learning components of strings and functions, but I would say that the themes are Encapsulation and Handling Unexpected Input.

The exercise component this week had many problems that involved ensuring specific types of input were received from the user. The problem of format names (exercise 2) would not be that difficult of a task, if all the incoming data was uniform. I sketched out a design for this problem (see below) and the bulk of the problem came down to eliminating possible deviations from acceptable input. I wrote down a series of test cases and outputs that also defined the range of acceptable input. then the problem simply became iterating over input - check for validity until it was a workable string inputed.

Encapsulation was also a heavy theme of the exercise components. Identifying the most fundamental parts of a task, and then deciphering the necessary information (parameters) to achieve that task. I would say that this skill and program practice are the most valuable benefit to using functions. If a program has non-parametered functions (referencing global values) that only solve a problem for that specific program then they have sort of only accomplished better readability. Conversely functions that operate on the most abstracted level, with well defined input and output conditions that are independent of the global context can be ported to many other problems.

This principle of encapsulation is well demonstrated by the three classes of static functions included in this weeks programs (swansonStrings, swansonInput, swansonUtils). The myFuntions.h which contains these functions declarations is #include-d in both my exercise components program, and my word guesses game. I am able to use these functions indiscriminate of the programs data system or the problem being solved. For example the function swansonInput::GetString once and for all (probably not) solves the problem of retrieving strings of input from the user, included error checking. And many of my other functions invoke this one. Granted in the future I might find more robust error check, but a refactor of this method (if the entry and exit conditions don't change) will carry that benefit through to every function/program that invokes it.

DESIGN

see sketches below. After having not read the assignment for a day and only having a vague memory of the specifications I drew up a design for the word guessing game. It was missing a few needed sets of data, and the exact nature of the output at each guess had to change.

I tried something slightly different this time concerning design, I usually begin by sketching out all my computational functions, choosing their parameters and return types, and test each one in isolation, then once each is correctly function i string them together into an entire program. this time i worked more from the outside inward by designing the gameplay loop first and calling not yet implemented functions. I coded dummy functions with hardcoded returns, and then checked that the gameplay loop worked. this helped me discover what the return types and paramaters of these functions should be.

TESTING

I used many different testing methods for this assignment. I tested my formulas on larger scopes and smaller scopes. I walked through my code step by step describing what each step would do and rubber ducked with other student on IRC, this was for testing against logic errors, one of wich i found on my first iteration for problem 7.

later I tested that this algorithm was working properly by adding very verbose cout statements containing the number generated, the numbers already generated, and then indicators of progressions through loops and conditions, followed by output (see below)

for the exercise components I tried to write a few exception cases before beging each one so that I could design against them and then test using them. for example what happens with only one name given or with 4 names given on the formatting names exercise

REFLECTION

This week I spent a lot of time and effort recreating functions that are already in the c++ library, such as the functions in cctype for isAlpha and isNum. I suppose these were a good learning exercise but I have since spent time trying to familiarize myself with c++ library functions to avoid this in the future.

this week for a number of different problems that involved storing a changing amount of items, (revealed characters, letters in word) I used array in favor of vectors. I was able to be certain of the maximum amount of items. The tricky and error prone part is that each function called needed to include the number of items in the array (not the size of the array, but the amount of items actually added to it, for this round). there was a danger of accessing items that were placed there in previous rounds, if these counter variables were not properly initialized. I think i might switch to vectors for this kind of storage, while they might have their own downsides, their convenience methods (size, clear) will help me avoid logic errors.

ACCOMPLISHMENTS

- I decided to make a string.h file to contain all the output strings so that they could easily be edited in one location, and my code would be easier to read
- extensive static functions that will be useful in assignments to come, especially the input functions
- learned how to compile programs with multiple source files

Word

Vars

CONST: MaxGuess
MaxWordLength

---

guessesMade String[]

numGuessAttempt

LettersCorrect bool[] or list<int>

Guess

>> Word

>> Guess

<< Hint

! Guess = Word ||
> MAX-G

Input

* Only one word
* Letters only
* no Spaces
* Max Word Length
* Is A Word?

Impossible?

Methods

AllLetters()
SeperateWords()
getString()

Contains()

Brandon _ Swanson

Brandon Drew Swanson
Brandon D. Swanson

Input

Swanson, Brandon D. Output

Str input
First, middle, Last
middleinit

get string
Chek Valid
- contains letters and " " " , "
- num words 2 or 3

~~format~~ Minitil + "."

rearange

List char " " " . "

bool
all lettersand (string, list)
string[]
~~Seperate~~ (string)
"tokenize"
string
getValidnam ()
{bool is letters(str)
bool 2 or 3 names(str)
}

Tests

~~Brandon~~
~~Bran_don_Swan~~_son
Brandon5 Sw!anson
Bran d. Swanson
Bran d

Problem 2
Rearange Name

insertion

void insert ([int], & int[], numfilled)

{

for (int i 0, < numfilled, ++i)

{ if (int in < [ ])

for (v = numfilled +1, < i, --)

Break; [v] = [i]

}

[i] = int in

}

range -2 = 5

3 = 7

5 = 7

4 = 6

0 0 0 0 0 0 0

range 13

0 0 0 0 0 0 0 0 0 0 0 0 0

35, 1

77 - - - -

inc = 0

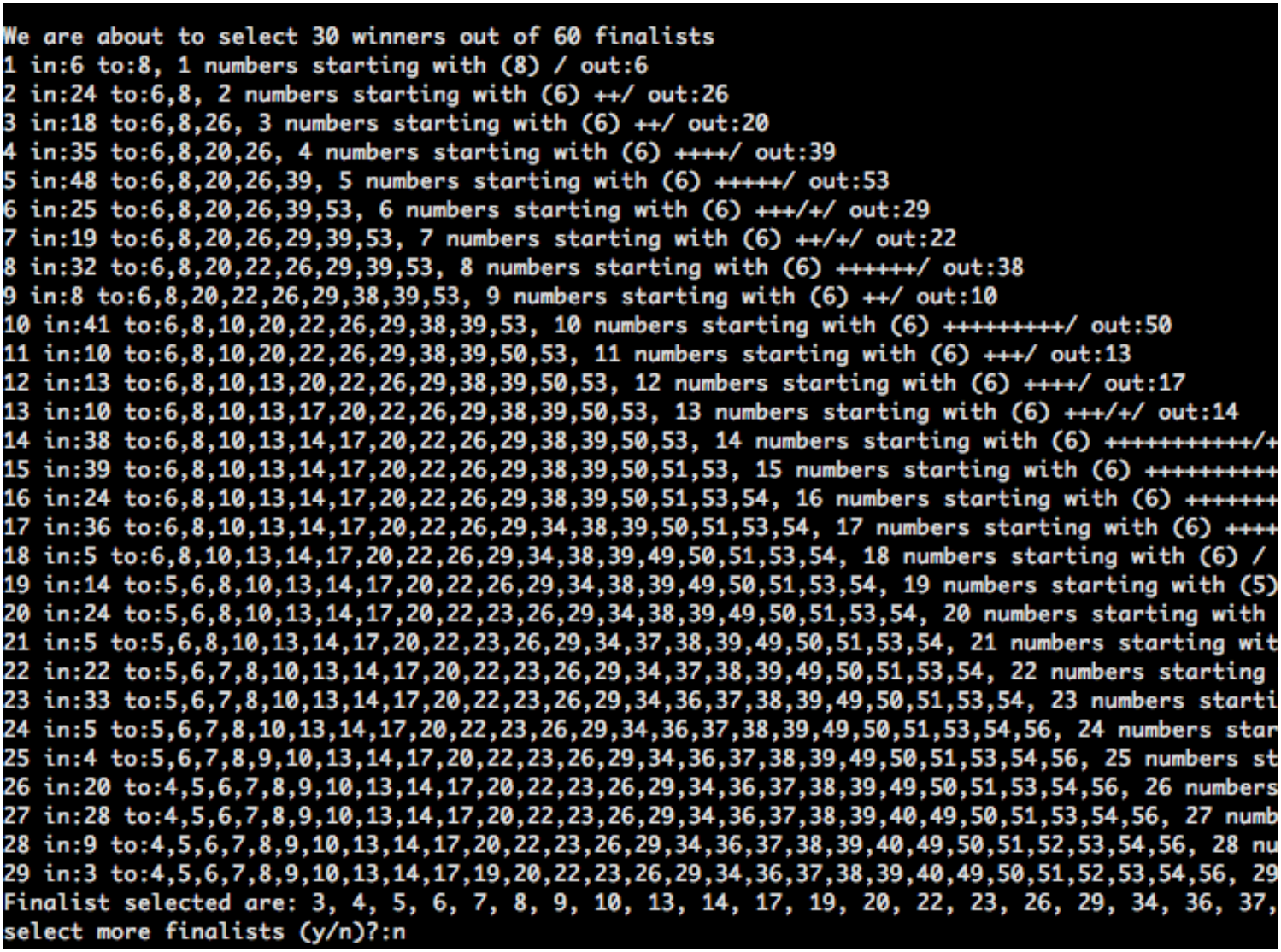
do {

while (!empty && front < selection)

{ inc + 1

} select + = inc

while { !empty && front < select }



Testing mapping numbers debug output

source code for produced output on previous page

```
for (int i = 1; i < NUM_WINNERS; ++i) {
        nextSelection = swansonUtil::randomInRange(1,range);
        range--;

        selectionsList.clear();

        //cout << endl << i << " in:" << nextSelection << " to:"; //debug
        for (int j = 0; j < i; j++) {
                selectionsList.push_back(finalistsSelected[j]);
                //cout << finalistsSelected[j] << ","; //debug
                //cout << selectionsList.back() << ","; //debug
        }

        //cout << " " << selectionsList.size() << " numbers starting with (" << selectionsList.front() << ") ";
        do{
                int increment = 0;
                while ( !selectionsList.empty() && selectionsList.front() <= nextSelection){
                        increment++;
                        selectionsList.pop_front();
                        //cout << "+"; //debug
                }
                //cout << "/"; //debug
                nextSelection += increment;
        } while ( !selectionsList.empty() && selectionsList.front() <= nextSelection);

        //cout << " out:" << nextSelection; //debug

        //finalistsSelected[i] = nextSelection;
        insertElement(nextSelection,finalistsSelected,i);
}
```