```
Script started on Thu 07 Dec 2017 05:08:56 PM CST
\033]0;p_stach@mars:~/frame\007[p_stach@mars frame]$ pwd
/home/students/p_stach/frame
\033]0;p_stach@mars:~/frame\007[p_stach@mars frame]$ cat main.ifn\033[K\033[K\033[Knfo
ame: Patrick Stach

Class: CSC122-001

Program: Project - "A box upon ye"

Levels Attempted:
This assignment is (Level 5).
Add (Level 1.5) to add a choice of frame types: single line, double line, or
shaded
Add (Level 1) to make a nice menu to set up these options and include in
the menu save and load options. This option configuration should also be
automatically read at program startup time \227 if it exists.
5+1.5+1= Level 7.5

Program Description:
This program takes user input either through keyboard of a phrase, and given
desired framing character, frame style and centering, will either
display the framed phrase or output it to a file
\033]0;p_stach@mars:~/frame\007[p_stach@mars frame]$ cat c\033[K\033[Kconfig.txt
x
0
C
\033]0;p_stach@mars:~/frame\007[p_stach@mars frame]$ cat data.txt
"Please give me an A in the class -Patrick"
\033]0;p_stach@mars:~/frame\007[p_stach@mars frame]$ cat frame.cpp
#include "frame.h"
#include <string>
#include <vector>
#include <iomanip>
using namespace std;

ostream& operator<<(ostream& out, const Frame& f) // output overload operator
{
    typedef std::vector<std::string>::size_type vct_str_type;
    typedef std::string::size_type str_type;
    //middle of frame
    vct_str_type height = f.word_list.size();
    str_type width = 0;
    for (vct_str_type pos = 0; pos < height; pos++) //finds width of frame
    {
        str_type tmp_width = f.word_list[pos].length();
        if (tmp_width > width)
        {
            width = tmp_width;
        }
    }
    switch (f.style)
    {
    case ’0’: //use character
    {

        if (f.align == ’L’) // Left align
        {
            out << string(width + 4, f.frame_ch) << endl;
            for (str_type i = 0; i < height; i++)
            {
                out << f.frame_ch << ’ ’ << setw(width)<<left << f.word_list[i];
                out << ’ ’ << f.frame_ch << endl;
            }
            out << string(width + 4, f.frame_ch) << endl;
        }
        else if (f.align == ’C’) // Center allign
        {
            out << string(width + 4, f.frame_ch) << endl;
            size_t padding_L, padding_R;
            for (str_type i = 0; i < height; i++)
            {
                //calculation for centering
                size_t word_length = f.word_list[i].length();
                padding_L = (width - word_length) / 2;
                padding_R = width - word_length - padding_L;
                out << f.frame_ch << ’ ’ << string(padding_L, ’ ’);
                out << f.word_list[i] << string(padding_R, ’ ’);
                out << ’ ’ << f.frame_ch << endl;
            }
            out << string(width + 4, f.frame_ch) << endl;
        }
        else if (f.align == ’R’) //right align
        {
            out << string(width + 4, f.frame_ch) << endl;
            for (str_type i = 0; i < height; i++)
            {
                out << f.frame_ch << ’ ’ << setw(width) << right << f.word_list[i];
                out << ’ ’ << f.frame_ch << endl;
            }
            out << string(width + 4, f.frame_ch) << endl;
        }
        else //error
        {
            cerr << "Error: invalid option for alignment." << endl;
        }

    }
    break;
    case ’1’: // Single line
    {
        if (f.align == ’L’) //left align
        {
            out << ’+’ << string(width + 2, ’-’) << ’+’ << endl;
            for (str_type i = 0; i < height; i++)
            {
                out << ’|’ << ’ ’ << setw(width) << left << f.word_list[i];
                out << ’ ’ << ’|’ << endl;
            }
            out << ’+’ << string(width + 2, ’-’) << ’+’ << endl;

        }
        else if (f.align == ’C’) // center align
        {
            out << ’+’ << string(width + 2, ’-’) << ’+’ << endl;
            size_t padding_L, padding_R, word_length;
            for (str_type i = 0; i < height; i++)
            {
                //calculation for center align
                word_length = f.word_list[i].length();
                padding_L = (width - word_length) / 2;
                padding_R = width - word_length - padding_L;
                out << ’|’ << ’ ’ << string(padding_L, ’ ’);
                out << f.word_list[i] << string(padding_R, ’ ’);
                out << ’ ’ << ’|’ << endl;
            }
            out << ’+’ << string(width + 2, ’-’) << ’+’ << endl;
        }
        if (f.align == 'R') //right align
        {
            out << '+' << string(width + 2, '-') << '+' << endl;
            for (str_type i = 0; i < height; i++)
            {
                out << '|' << ' ' << setw(width) << right << f.word_list[i];
                out << ' ' << '|' << endl;
            }
            out << '+' << string(width + 2, '-') << '+' << endl;

        }
        else //error
        {
            cerr << "Error: invalid option for alignment." << endl;
        }

    }
    break;
    case '2': // Single line, shaded
    {
        if (f.align == 'L') //left align
        {
            out << '+' << string(width + 2, '-') << '+' << endl;
            for (str_type i = 0; i < height; i++)
            {
                out << '|' << ' ' << setw(width) << left << f.word_list[i];
                out << ' ' << "|*" << endl;
            }
            out << '+' << string(width + 2, '-') << "+*" << endl;
            out << ' ' << string(width + 4, '*') << endl;

        }
        else if (f.align == 'C') //center align
        {
            out << '+' << string(width + 2, '-') << '+' << endl;
            size_t padding_L, padding_R, word_length;
            for (str_type i = 0; i < height; i++)
            {
                //center align calculation
                word_length = f.word_list[i].length();
                padding_L = (width - word_length) / 2;
                padding_R = width - word_length - padding_L;
                out << '|' << ' ' << string(padding_L, ' ');
                out << f.word_list[i] << string(padding_R, ' ');
                out << ' ' << "|*" << endl;
            }
            out << '+' << string(width + 2, '-') << '+' << endl;
            out << ' ' << string(width + 4, '*') << endl;

        }
        else if (f.align == 'R') //right align
        {
            out << '+' << string(width + 2, '-') << '+' << endl;
            for (str_type i = 0; i < height; i++)
            {
                out << '|' << ' ' << setw(width) << right << f.word_list[i];
                out << ' ' << "|*" << endl;
            }
            out << '+' << string(width + 2, '-') << "+*" << endl;
            out << ' ' << string(width + 4, '*') << endl;
        }
        else
        {
            cerr << "Error: invalid option for alignment." << endl;
        }

    }
    break;
    case '3': // Double line
    {
        if (f.align == 'L') //left align
        {
            out << '+' << string(width + 4, '=') << '+' << endl;
            for (str_type i = 0; i < height; i++)
            {
                out << "||" << ' ' << setw(width) << left << f.word_list[i];
                out << ' ' << "||" << endl;
            }
            out << '+' << string(width + 4, '=') << '+' << endl;

        }
        else if (f.align == 'C') //center align
        {
            out << '+' << string(width + 4, '=') << '+' << endl;
            size_t padding_L, padding_R, word_length;
            for (str_type i = 0; i < height; i++)
            {
                //calculation for center align
                word_length = f.word_list[i].length();
                padding_L = (width - word_length) / 2;
                padding_R = width - word_length - padding_L;
                out << "||" << ' ' << string(padding_L, ' ');
                out << f.word_list[i] << string(padding_R, ' ');
                out << ' ' << "||" << endl;
            }
            out << '+' << string(width + 4, '=') << '+' << endl;

        }
        else if (f.align == 'R') //right align
        {
            out << '+' << string(width + 4, '=') << '+' << endl;
            for (str_type i = 0; i < height; i++)
            {
                out << "||" << ' ' << setw(width) << right << f.word_list[i];
                out << ' ' << "||" << endl;
            }
            out << '+' << string(width + 4, '=') << '+' << endl;
        }

    }
    break;
    case '4': // Double line, shaded
    {
        if (f.align == 'L') //left align
        {
            out << '+' << string(width + 4, '=') << '+' << endl;
            for (str_type i = 0; i < height; i++)
            {
                out << "||" << ' ' << setw(width) << left << f.word_list[i];
                out << ' ' << "||#" << endl;
            }
            out << '+' << string(width + 4, '=') << "+#" << endl;
            out << ' ' << string(width + 6, '#');

        }
        else if (f.align == 'C') // center align
        {
            out << '+' << string(width + 4, '=') << '+' << endl;
            size_t padding_L, padding_R, word_length;
            for (str_type i = 0; i < height; i++)
            {
                //calculation for center align
                word_length = f.word_list[i].length();
                padding_L = (width - word_length) / 2;
                padding_R = width - word_length - padding_L;
                out << "||" << ' ' << string(padding_L, ' ');
                out << f.word_list[i] << string(padding_R, ' ');
                out << ' ' << "||#" << endl;
            }
            out << '+' << string(width + 4, '=') << "+#" << endl;
            out << ' ' << string(width + 6, '#');

        }
        else if (f.align == 'R') //right align
        {
            out << '+' << string(width + 4, '=') << '+' << endl;
            for (str_type i = 0; i < height; i++)
            {
                out << "||" << ' ' << setw(width) << right << f.word_list[i];
                out << ' ' << "||#" << endl;
            }
            out << '+' << string(width + 4, '=') << "+#" << endl;
            out << ' ' << string(width + 6, '#');
        }

    }
    break;
    default:
    {
        cerr << "Error: invalid option for style." << endl;
    }
    }

    return out;
}

//inputs phrase
istream& operator>>(istream& in, Frame& f)
{
    bool get_first = false;
    char first;
    while (!get_first) // ignores garbage input until
    {
        in >> first;
        if (first == '"')
        {
            get_first = true;
        }
    }
    bool done = false;
    string str_tmp;
    char char_tmp;
    while (!done) // vector input
    {
        if (in.peek() == ' ') // finds space
        {
            in >> char_tmp; // takes in last letter before space
            if (!str_tmp.empty()) // something in string
            {
                f.word_list.push_back(str_tmp);
                str_tmp = char_tmp;
            }
        }
        else if (in.peek() == '\"') // Finds end of parenthesis
        {
            in >> char_tmp; //gobble quotes
            if (!str_tmp.empty()) // something in string
            {
                f.word_list.push_back(str_tmp);
                str_tmp = "";
            }
            done = true;
        }
        else //finds character that is not space or "
        {
            in >> char_tmp; // ins char
            str_tmp = str_tmp + char_tmp;
        }
    }

    return in;
}
\033]0;p_stach@mars:~/frame\007[p_stach@mars frame]$ cat frame.h
#ifndef FRAME_H
#define FRAME_H

#include <string>
#include <iostream>
#include <vector>

class Frame
{

    std::vector<std::string> word_list;
    char frame_ch;
    char style;// 0=shaded, 1 = single, 2 = double
    char align; // L= Left, C = Center, R = Right
public:
    Frame() : frame_ch('|'), style(1), align('L') {}
    void print_word_list() const
    {
        typedef std::vector<std::string>::size_type vct_str_type;
        for (vct_str_type pos = 0; pos < word_list.size(); pos++)
        {
            std::cout << word_list[pos] << std::endl;
        }
    }

    friend std::ostream& operator<<(std::ostream& out, const Frame& f);
    friend std::istream& operator>>(std::istream& in, Frame& f);
    void input_config(std::istream& in)
    {
        in >> frame_ch >> style >> align;
    }
    void output_config(std::ostream& out)
    {
        out << frame_ch << std::endl <<style << std::endl << align << std::endl;
    }

};

#endif
\033]0;p_stach@mars:~/frame\007[p_stach@mars frame]$ cat main.cpp
#include <iostream>
#include <string>
```

```
#include <climits>
#include "frame.h"
#include <fstream>
using namespace std;

const string divider = string(60, ’-’) + ’\n’;

const string menu = // Menu
"Enter (separated by space or newline):\n"
"* Framing character\n"
"* Style (choices below)\n"
"    0) Use character\n"
"    1) Single line frame\n"
"    2) Single line frame (shaded)\n"
"    3) Double line frame\n"
"    4) Double line frame (shaded)\n"
"* (R)ight, (L)eft or (C)enter justified\n";

bool true_or_false(const string inputs);

int main()
{
    bool done = false;
    bool loaded = false;
    string config = "config.txt";
    cout << "Attempting to load config.txt." << endl;
    ifstream conf_in;
    conf_in.open(config.c_str());
    if (conf_in)
    {
        cout << "Loaded config file" << endl;
    }
    else
    {
        cout << "No config file found" << endl;
    }
    ifstream input;
    ofstream output;
    string filename;
    while (!done)
    {
        cout << divider;
        Frame f;
        // User chose to read from file
        if (true_or_false("Read phrase from file? Y/N: "))
        {

            cout  << "What is the name of the input file?  ";
            getline(cin, filename);
            input.open(filename.c_str());
            while (!input) // open failed
            {
                input.close();
                input.clear();
                cout << divider << "File does not exist.\nEnter file name: ";
                getline(cin, filename);
                input.open(filename.c_str());
            }
            cout << "’" << filename << "’ selected as input file." << endl;
            input >> f;
            if (conf_in)
            {
                if (true_or_false("Use initial config file? (Y/N): "))
                {
                    f.input_config(conf_in);
                }
                else
                {
                    cout << menu;
                    f.input_config(cin);
                    if (true_or_false("Save config file? (Y/N): "))
                    {
                        if (conf_in)
                        {
                            conf_in.close();
                            ofstream conf_out;
                            conf_out.open(config.c_str());
                            f.output_config(conf_out);
                            conf_out.close();
                            conf_in.open(config.c_str());
                        }
                        else
                        {
                            ofstream conf_out;
                            conf_out.open(config.c_str());
                            f.output_config(conf_out);
                            conf_out.close();
                        }
                    }
                }
            }
            else
            {
                cout << menu;
                f.input_config(cin);
                if (true_or_false("Save config file? (Y/N): "))
                {
                    if (conf_in)
                    {
                        conf_in.close();
                        ofstream conf_out;
                        conf_out.open(config.c_str());
                        f.output_config(conf_out);
                        conf_out.close();
                        conf_in.open(config.c_str());
                    }
                    else
                    {
                        ofstream conf_out;
                        conf_out.open(config.c_str());
                        f.output_config(conf_out);
                        conf_out.close();
                    }
                }
            }

            if (true_or_false("Output frame to file (Y/N): "))
            {
                cout << divider << "Enter a new output file name:  ";
                getline(cin, filename);
                output.open(filename.c_str());
                cout << "’" << filename << "’ selected as output file." << endl;
                output << f;
                output.close();
            }
            else
            {
                cout << f << endl;
            }
            input.close();
        }
        // User chose to read from keyboard
        else
        {
            cout << "Input phrase in quotes (\"\"): ";
            cin >> f; // input
            if (conf_in)
            {
                if (true_or_false("Use initial config file? (Y/N): "))
                {
                    f.input_config(conf_in);
                }
                else
                {
                    cout << menu;
                    f.input_config(cin);
                    if (true_or_false("Save config file? (Y/N): "))
                    {
                        if (conf_in)
                        {
                            conf_in.close();
                            ofstream conf_out;
                            conf_out.open(config.c_str());
                            f.output_config(conf_out);
                            conf_out.close();
                            conf_in.open(config.c_str());
                        }
                        else
                        {
                            ofstream conf_out;
                            conf_out.open(config.c_str());
                            f.output_config(conf_out);
                            conf_out.close();
                        }
                    }
                }
            }
            if (true_or_false("Output frame to file (Y/N): "))
            {
                cout << divider << "Enter a new output file name:  ";
                getline(cin, filename);
                output.open(filename.c_str());
                cout << "'" << filename << "' selected as output file." << endl;
                output << f;
                output.close();
            }
            else
            {
                cout << f << endl;
            }

        }
        done = true_or_false("Exit? Y/N: "); // Asks to exit
    }
    conf_in.close();
    return 0;
}

//returns true when user enters 'Y' or 'y', false for 'N' or 'n'
bool true_or_false(const string prompt)
{
    bool valid_input = false;
    char input;
    bool choice;
    while (!valid_input)
    {
        cout << prompt;
        cin >> input;
        cin.clear();
        cin.ignore(INT_MAX, '\n');
        switch (input)
        {
        case 'Y': case 'y':
        {
            choice = true;
            valid_input = true;
        }
        break;
        case 'N': case 'n':
        {
            choice = false;
            valid_input = true;
        }
        break;
        default:
        {
            cout << "Invalid input.\n";
        }
        }
    }
    return choice;
}
\033]0;p_stach@mars:~/frame\007[p_stach@mars frame]$ CPP main frame
frame.cpp...
main.cpp***
In file included from frame.cpp:1:
frame.h:11:   instantiated from here
frame.h:11:   instantiated from here
frame.h:11:   instantiated from here
frame.h: In constructor 'Frame::Frame()':
frame.h:16: warning: 'Frame::word_list' should be initialized in the
member initialization list
frame.h:20:   instantiated from here
frame.cpp:289:   instantiated from here
In file included from main.cpp:4:
frame.h:11:   instantiated from here
frame.h:11:   instantiated from here
frame.h:11:   instantiated from here
frame.h: In constructor 'Frame::Frame()':
frame.h:16: warning: 'Frame::word_list' should be initialized in the
member initialization list
main.cpp: In function 'int main()':
main.cpp:26: warning: unused variable 'loaded'
frame.h:20:   instantiated from here


\033]0;p_stach@mars:~/frame\007[p_stach@mars frame]$ ./main.out
Attempting to load config.txt.
Loaded config file
------------------------------------------------------------
Read phrase from file? Y/N: y
What is the name of the input file?  data.txt
'data.txt' selected as input file.
Use initial config file? (Y/N): y
Output frame to file (Y/N): y
------------------------------------------------------------
```

```
Enter a new output file name:  output.txt
’output.txt’ selected as output file.
Exit? Y/N: y
\033]0;p_stach@mars:~/frame\007[p_stach@mars frame]$ cat output.txt
xxxxxxxxxxxx
x  Please  x
x   give   x
x    me    x
x    an    x
x    A     x
x    in    x
x   the    x
x  class   x
x -Patrick x
xxxxxxxxxxxx
\033]0;p_stach@mars:~/frame\007[p_stach@mars frame]$ cat output.txt
xxxxxxxxxxxx
x  Please  x
x   give   x
x    me    x
x    an    x
x    A     x
x    in    x
x   the    x
x  class   x
x -Patrick x
xxxxxxxxxxxx
\033]0;p_stach@mars:~/frame\007[p_stach@mars frame]$ \033[4P./main.xtout
Attempting to load config.txt.
Loaded config file
-----------------------------------------------------------
Read phrase from file? Y/N: n
Input phrase in quotes (""): "Hello Mr. James please give me andAä"
Use initial config file? (Y/N): n
Enter (separated by space or newline):
* Framing character
* Style (choices below)
    0) Use character
    1) Single line frame
    2) Single line frame (shaded)
    3) Double line frame
    4) Double line frame (shaded)
* (R)ight, (L)eft or (C)enter justified
x 0 R
Save config file? (Y/N): y
Output frame to file (Y/N): n
xxxxxxxxxx
x  Hello x
x    Mr. x
x  James x
x please x
x   give x
x     me x
x     an x
x      A x
xxxxxxxxxx

Exit? Y/N: y
\033]0;p_stach@mars:~/frame\007[p_stach@mars frame]$ cat config.txt
x
0
R
\033]0;p_stach@mars:~/frame\007[p_stach@mars frame]$ \033[4P./main.xtout
Attempting to load config.txt.
```

```
Loaded config file
-----------------------------------------------------------
Read phrase from file? Y/N: n
Input phrase in quotes (""): "I would love to get an A"
Use initial config file? (Y/N): n
Enter (separated by space or newline):
* Framing character
* Style (choices below)
    0) Use character
    1) Single line frame
    2) Single line frame (shaded)
    3) Double line frame
    4) Double line frame (shaded)
* (R)ight, (L)eft or (C)enter justified
0 1 L
Save config file? (Y/N): n
Output frame to file (Y/N): n
+-------+
| I     |
| would |
| love  |
| to    |
| get   |
| an    |
| A     |
+-------+
Error: invalid option for alignment.

Exit? Y/N: n
-----------------------------------------------------------
Read phrase from file? Y/N: n
Input phrase in quotes (""): "Plzz gimme A"
Use initial config file? (Y/N): n
Enter (separated by space or newline):
* Framing character
* Style (choices below)
    0) Use character
    1) Single line frame
    2) Single line frame (shaded)
    3) Double line frame
    4) Double line frame (shaded)
* (R)ight, (L)eft or (C)enter justified
2 2 C
Save config file? (Y/N): n
Output frame to file (Y/N): n
+-------+
| Plzz  |*
| gimme |*
|   A   |*
+-------+
 *********

Exit? Y/N: n
-----------------------------------------------------------
Read phrase from file? Y/N: n
Input phrase in quotes (""): "ayyyy gimme that A plz"
Use initial config file? (Y/N): n
Enter (separated by space or newline):
* Framing character
* Style (choices below)
    0) Use character
    1) Single line frame
    2) Single line frame (shaded)
    3) Double line frame
    4) Double line frame (shaded)
* (R)ight, (L)eft or (C)enter justified
0 3 R
Save config file? (Y/N): n
Output frame to file (Y/N): n
+=========+
|| ayyyy ||
|| gimme ||
||  that ||
||     A ||
||   plz ||
+=========+

Exit? Y/N: n
------------------------------------------------------------
Read phrase from file? Y/N: n
Input phrase in quotes (""): "pleaase i am begging you "for that
"
e

"Please I am begging you"

duyy]


[1]+  Stopped                 ./main.out
\033]0;p_stach@mars:~/frame\007[p_stach@mars frame]$ ./main.out
Attempting to load config.txt.
Loaded config file
------------------------------------------------------------
Read phrase from file? Y/N: n
Input phrase in quotes (""): n
"Hello there buddy"
Use initial config file? (Y/N): n
Enter (separated by space or newline):
* Framing character
* Style (choices below)
    0) Use character
    1) Single line frame
    2) Single line frame (shaded)
    3) Double line frame
    4) Double line frame (shaded)
* (R)ight, (L)eft or (C)enter justified
k 4 L
Save config file? (Y/N): n
Output frame to file (Y/N): n
+=========+
|| Hello ||#
|| there ||#
|| buddy ||#
+=========+#
 ###########
Exit? Y/N: y
\033]0;p_stach@mars:~/frame\007[p_stach@mars frame]$ exit
exit
There are stopped jobs.
\033]0;p_stach@mars:~/frame\007[p_stach@mars frame]$ exit
exit

Script done on Thu 07 Dec 2017 05:18:13 PM CST
```