TODO: TaskManager™

Nicholas Rubio

```
#include <iostream>
#include <string>
using namespace std;

#include <fstream>
#include <filesystem>
#include "menu.h"
#include "welcomeMessage.h"
#include "secret.h"
```

```
//
//  menu.h
//  TaskManager-Final
//

#ifndef MENU_H // prevents multiple inclusions
#define MENU_H

#include <iostream>
using namespace std;

inline void menu(){
    cout << "\t▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀\n\t▶\t\t\t\t\t\t\t\t\t\t   ◀\n";
    cout << "\t▶\t\t\t\t  ▶ MENU ◀     \t\t\t   ◀\n\t▶\t\t\t\t\t\t\t\t\t\t   ◀\n"
    << "\t▶\t1. Welcome Message\t\t\t\t   ◀\n\t▶\t\t\t\t\t\t\t\t\t\t   ◀\n"
    << "\t▶\t2. Create Task\t\t\t\t\t   ◀\n"
    << "\t▶\t3. Rename Task\t\t\t\t\t   ◀\n"
    << "\t▶\t4. Reset Task Description\t\t\t   ◀\n\t▶\t\t\t\t\t\t\t\t\t\t   ◀\n"
    << "\t▶\t5. Reset Task Priority Level\t\t   ◀\n\t▶\t\t\t\t\t\t\t\t\t\t   ◀\n"

    << "\t▶\t6. Delete Task\t\t\t\t\t   ◀\n\t▶\t\t\t\t\t\t\t\t\t\t   ◀\n"
    << "\t▶\t7. Display All Task Names\t\t\t   ◀\n"
    << "\t▶\t8. Search Name (Get all info)(BinSrch) ◀\n\t▶\t\t\t\t\t\t\t\t\t\t   ◀\n"
    << "\t▶\t9. Save session\t\t\t\t\t   ◀\n\t▶\t\t\t\t\t\t\t\t\t\t   ◀\n"
    << "\t▶\t10. Quit\t\t\t\t\t\t   ◀\n\t▶\t\t\t\t\t\t\t\t\t\t   ◀\n";
}
#endif
```

```
//
//  welcomeMessage.h
//  TaskManager-Final
//

#ifndef WELCOMEMESSAGE_H // prevents multiple inclusions
#define WELCOMEMESSAGE_H

#include <iostream>
using namespace std;

inline void welcomeMessage(){
    cout << "\n\t\t\t\t\t\t\tWelcome to TaskManager™!\n\n"
            << "In this console-based to-do-list, you can store and manage up to FIFTY tasks at a time!\n\n"
            << "As a user, your privileges include:\n\n"
            << "• Creating a task\n"
            << "• Deleting a task\n"
            << "• Renaming a task\n"
            << "• Redescribing a task\n"
            << "• Resetting a task's priorty level\n\n"
            << "Features include:\n\n"
            << "• Display all tasks\n"
            << "• sortTasksByName algorithm + binarySearch algorithm\n"
            << "• Save/load TaskManager™ sessions\n\n"
            << "*** Be advised: Every session with TaskManager™ can be both saved and loaded to a text file for your convenience!"
            << "\n\n                                                                                \n" << endl;
}
#endif
```

```
//
//  secret.h
//  TaskManager-Final
//

#ifndef SECRET_H
#define SECRET_H

#include <iostream>
using namespace std;

inline void secret(){
    cout << "\n\t\t\t\t\t\t\t\t\t\t\t^•ﻌ•^ฅ"
         << "\n\tSecret (͡° ͜ʖ ͡°)\t\t\t╬APPY"
         << "\n\t\t\t\t\t\t\t\tBIRT╬DAY!\n"
         << "\n                                                                                \n\n";

}
#endif

/*
 _______________________
/ \                      \
|   | I wish I could use  |
\_  |    this cool paper  |
   | thing, but it's ann- |
   | oying to put in cout |
   |    statements :p     |
   |                      |
   |        - Nicholas R. |
   |   ___________________|__
   |  /                     /
   \_/_____________________/
 */
```

```cpp
class Task {
public:
    string taskName = "No title";
    string description = "No description";
    short priorityLevel = 0;

    Task(){};

    Task(string tn, string d, short pl){
        taskName = tn;
        description = d;
        priorityLevel = pl;
        cout << "\n▶▶▶▶ Task(string, string, short) created successfully ◀◀◀◀" << endl;
    }

    // The createTask method creates a new Task by collecting valid input from the user.
    // It ensures that the title, description, and priority level meet specific constraints.
    void createTask(){

        string name, desc;
        short pri = 0;

        cout << "\n\t\t\t\t\t▶ NEW TASK ◀ \n" << endl;
        cout << "\t   ▶▶▶ Please enter the following data ◀◀◀\n" << endl;

        while (name.length() > 20 || name.empty()) {
            cout << "\nEnter Title (20 Characters or Less): ";
            getline(cin, name);

            if (name.length() > 20) {
                cout << "\n\t\t  ▶▶▶ ERROR: TITLE MUST BE 20 CHARACTERS OR LESS ◀◀◀\n\n";
            } else if (name.empty()) {
                cout << "\n\t\t  ▶▶▶ ERROR: TITLE CANNOT BE EMPTY ◀◀◀\n\n";
            }
        }

        while (desc.length() > 75 || desc.empty()) {
            cout << "\nEnter Description (75 Characters or Less): ";
            getline(cin, desc);

            if (desc.length() > 75) {
                cout << "\n\t\t  ▶▶▶ ERROR: DESCRIPTION MUST BE 75 CHARACTERS OR LESS ◀◀◀\n\n";
            } else if (desc.empty()) {
                cout << "\n\t\t  ▶▶▶ ERROR: DESCRIPTION CANNOT BE EMPTY ◀◀◀\n\n";
            }
        }

        while (pri >= 11 || pri <= 0){
            cout << "\nPriorty level (1-10): ";
            cin >> pri;
            cin.ignore();

            if (pri >= 11 || pri <= 0){
                cout << "\n▶▶▶ ERROR: PRIORITY LEVEL MUST BE WITHIN RANGE (1-10) ◀◀◀\n\n";
            }
        }
```

```
    // Once all input is valid, set the member variables of the Task.
    taskName = name;
    description = desc;
    priorityLevel = pri;

    // Confirm successful creation of the task.
    cout << "\n\n\t\t   ▶▶▶ TASK SUCCESSFULLY CREATED ◀◀◀\n\n"
    << "                                                    \n" << endl;
}

string getTaskName()const{return taskName;}    // getter for Task Name
string getTaskDesc()const{return description;} // getter for Task Description
short getTaskPri()const{return priorityLevel;} // getter for Task Priority Level
```

```cpp
    void renameTask(){
        string newName;
        string oldName = getTaskName();
        while (true) {
            cout << "\n\tEnter new name: ";
            getline(cin, newName);
            if (newName.length() > 20) {
                cout << "\n\t\t  ▶▶▶ ERROR: TITLE MUST BE 20 CHARACTERS OR LESS ◀◀◀\n\n";           // length check + null/empty check
            } else if (newName.empty()) {
                cout << "\n\t\t  ▶▶▶ ERROR: TITLE CANNOT BE EMPTY ◀◀◀\n\n";
            } else {
                break;
            }
        }
        // Set and confirm for user
        taskName = newName;
        cout << "\n\t\t  ▶▶▶ TASK NAME SUCCESSFULLY CHANGED ◀◀◀\n\n"
        << "\t\t\t• New = " << taskName << "\t\t• Previous = " << oldName
        << "\n\n[redacted]\n" << endl;
    }

    // The renameDesc method overwrites a preexisting Task description by collecting valid input from the user.
    // It ensures that the new description meets specific constraints before overwriting.
    void renameDesc(){
        string newDesc;
        string oldDesc = getTaskDesc();
        while (true) {
            cout << "\n\tEnter new Description: ";
            getline(cin, newDesc);
            if (newDesc.length() > 75) {
                cout << "\n\t\t  ▶▶▶ ERROR: DESCRIPTION MUST BE 75 CHARACTERS OR LESS ◀◀◀\n\n";  // length check + null/empty check
            } else if (newDesc.empty()) {
                cout << "\n\t\t  ▶▶▶ ERROR: DESCRIPTION CANNOT BE EMPTY ◀◀◀\n\n";
            } else {
                break;
            }
        }
        // Set and confirm for the user
        description = newDesc;
        cout << "\n\n\t\t  ▶▶▶ TASK DESCRIPTION SUCCESSFULLY CHANGED ◀◀◀\n\n"
        << "\t New = " << newDesc << "\n\n\t Previous = " << oldDesc
        << "\n\n[redacted]\n" << endl;

    }

    // The resetPriLvl method resets a preexisting Task Priority Level by collecting valid input from the user.
    // It ensures that the new priority meets specific constraints before overwriting.
    void resetPriLvl(){
        short newPriLvl = 0;
        short oldPriLvl = getTaskPri();

        while (newPriLvl < 1 || newPriLvl > 10) {
            cout << "\n\tEnter new Priority level (1-10): ";
            cin >> newPriLvl;
            cin.ignore();

            if (newPriLvl < 1 || newPriLvl > 10) {
                cout << "\n\t ▶▶▶ PRIORITY LEVEL MUST BE WITHIN RANGE (1-10) ◀◀◀\n";
            }
        }
        // Set and confirm for the user
        priorityLevel = newPriLvl;
        cout << "\n\t\t  ▶▶▶ PRIORITY LEVEL SUCCESSFULLY CHANGED ◀◀◀\n\n"
        << "\t\t\t     New = " << newPriLvl << "\t  Previous = " << oldPriLvl
        << "\n\n[redacted]\n" << endl;
    }
};
```

```
// ------------------------------------------------------------------------------------

// Algorithm for searching a Task by title in a SORTED array of Task objects.
int binarySearch(Task arr[], int size, string target) {
    int low = 0;
    int high = size - 1;
    while (low <= high) {
        int mid = (low + high) / 2;
        if (arr[mid].getTaskName() == target) {
            return mid;
        } else if (arr[mid].getTaskName() < target) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return -1;
}

// Algorithm for SORTING the array of Task objects (needed for binarySearch)
void sortTasksByName(Task arr[], int size) {
    for (int i = 0; i < size - 1; ++i) {
        for (int j = 0; j < size - i - 1; ++j) {
            if (arr[j].getTaskName() > arr[j + 1].getTaskName()) {
                swap(arr[j], arr[j + 1]);
            }
        }
    }
}

// ------------------------------------------------------------------------------------
```

```cpp
int main(){

    const short MAX = 50;
    Task taskList[MAX];

    short taskCount = 0;
    short num = 0;
    short ask = 0;

    cout << "[redacted]" << endl; // Added solid blocks for more obvious section seperation
    cout << "\n\tLoad a previous file? (0/1): ";
    cin >> ask;
    cin.ignore();


    // The start of 'main' is a while loop, that is prolonged for the user as long as neccessary to enter a file name correctly
    // They can break the loop from the start or retry as many times as needed
    if(ask == 1){
        while (true){ // Start of while loop
            string filename = "";
            cout << "\n\t*** Enter the name of the file: ";
            getline(cin, filename);

            ifstream inFile(filename);

            if(!inFile){
                cout << "\n\t▶▶▶ ERROR: FILE NOT FOUND ◀◀◀\n";
                cout << "\n\tTry again? (0/1): ";
                cin >> num;
                cin.ignore();
                if (num == 0){ // This is where the user can break out of the loop if they can't find a file to reload
                    cout << "\n[redacted]\n" << endl;
                    break;
                }

            } else {
                string name, desc, line;
                short pri;
                while (getline(inFile, line)) {
                    if (line.find("Name: ") == 0) {
                        name = line.substr(6);
                    } else if (line.find("Description: ") == 0) {
                        desc = line.substr(13);
                    } else if (line.find("Priority: ") == 0) {
                        pri = stoi(line.substr(10));
                        taskList[taskCount] = Task(name, desc, pri);
                        taskCount++;
                    }
                }
                // File is closed upon successful retrievel of data + confirmation for user is posted
                inFile.close();
                cout << "\n\t▶▶▶ Successfully loaded (" << taskCount << ") tasks from " << filename << " ◀◀◀\n";
                cout << "\n[redacted]\n" << endl;
                break;
            }
        }
    } else {
        cout << "\n[redacted]\n" << endl;

    }
```

```
// This is where the user will spend most of their time.
// The menu will ALWAYS be recalled until the user has quit the program
// The menu is what directs the user to the program's functionality + save feature for future sessions
while (true){
    menu();

    cout << "\t▶\t*** Enter a number: ";
    cin >> num;
    cin.ignore();
    cout << "\t▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀" << endl;
    while (num <= 0 || num >= 12){
        cout << "\n\t\t▶▶▶ ERROR: ENTRY MUST BE 1-10 ◀◀◀\n";
        cout << "\n\t\t*** Enter a number: ";
        cin >> num;
        cin.ignore();
        cout << "\n\t▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀▶◀" << endl;
    }
    cout << "\n                                                                                                    " << endl;
```

```
// switch cases are used to speed up processing time immensely
// only runs what's called for, completely ignores the rest of the code in all other cases
switch (num){
    case 1:
        welcomeMessage(); // provides a breakdown of TaskManager™'s features and functionality
        break;



        // Checks if the Task array still has space for adding another Task. Throws error message if false
    case 2:
        if (taskCount < MAX) {
            taskList[taskCount].createTask();
            taskCount++;
        } else {
            cout << "\n\t\t  ▶▶▶ ERROR: TASK LIST MAXED OUT ◀◀◀\n\n"
            << "                                                    \n" << endl;
        }
        break;


        // Checks if Task Array has at least 1 Task Object to rename.
        // Ensures the user enters a valid selection. Otherwise sends user back to Menu
        // Provides a small menu for the user to make a selection for renaming
    case 3:
        if (taskCount == 0) {
            cout << "\n\t\t  ▶▶▶ ERROR: NO TASK TO RENAME ◀◀◀\n\n"
            << "\n\n                                                    \n" << endl;
        } else {
            cout << "\n\t\t\t\t\t▶ CURRENT TASKS ◀ \n" << endl;
            for (int i = 0; i < taskCount; ++i) {
                cout << "\t" << (i + 1) << ". "
                << taskList[i].getTaskName() << endl;
            }

            int renameIndex;
            cout << "\n\tEnter the number of the task to rename: ";
            cin >> renameIndex;
            cin.ignore();

            if (renameIndex < 1 || renameIndex > taskCount) {
                cout << "\n\t\t  ▶▶▶ ERROR: INVALID TASK NUMBER ◀◀◀\n"
                << "\n                                                    \n" << endl;
            } else {
                taskList[renameIndex - 1].renameTask();
            }
        }
        break;
```

```
    // Checks if Task Array has at least 1 Task Object to update the description of.
    // Ensures the user enters a valid selection. Otherwise sends user back to Menu
    // Provides a small menu for the user to make a selection for updating
case 4:
    if (taskCount == 0) {
        cout << "\n\t\t  ▶▶▶ ERROR: NO TASKS TO UPDATE DESCRIPTION ◀◀◀\n"
        << "\n\n                                                                    \n" << endl;
    } else {
        cout << "\n\t\t\t\t\t\t▶ CURRENT TASKS ◀ \n" << endl;
        for (int i = 0; i < taskCount; ++i) {
            cout << "\t" << (i + 1) << ". "
            << taskList[i].getTaskName() << endl;
        }

        int descIndex;
        cout << "\n\tEnter the number of the task to reset description: ";
        cin >> descIndex;
        cin.ignore();

        if (descIndex < 1 || descIndex > taskCount) {
            cout << "\n\t\t  ▶▶▶ ERROR: INVALID TASK NUMBER ◀◀◀\n"
            << "\n                                                                    \n" << endl;
        } else {
            taskList[descIndex - 1].renameDesc();
        }
    }
    break;


    // Checks if Task Array has at least 1 Task Object to update the priority of.
    // Ensures the user enters a valid selection. Otherwise sends user back to Menu
    // Provides a small menu for the user to make a selection for updating
case 5:
    if (taskCount == 0) {
        cout << "\n\t▶▶▶ ERROR: NO TASKS PRESENT ◀◀◀"
        << "\n\n                                                                    \n" << endl;
    } else {
        cout << "\n\t\t\t\t\t\t▶ CURRENT TASKS ◀ \n" << endl;
        for (int i = 0; i < taskCount; ++i) {
            cout << "\t" << (i + 1) << ". "
            << taskList[i].getTaskName() << endl;
        }

        int priIndex;
        cout << "\n\tEnter the number of the task to update Priority Level: ";
        cin >> priIndex;
        cin.ignore();

        if (priIndex < 1 || priIndex > taskCount) {
            cout << "\n\t ▶▶▶ ERROR: INVALID TASK NUMBER ◀◀◀\n"
            << "\n                                                                    \n" << endl;
        } else {
            taskList[priIndex - 1].resetPriLvl();
        }
    }
    break;
```

```
    // Checks if Task Array has at least 1 Task Object to delete
    // Ensures the user enters a valid selection. Otherwise sends user back to Menu
    // Provides a small menu for the user to make a selection for deleting
case 6:
    if (taskCount == 0) {
        cout << "\n\t\t  ▶▶▶ ERROR: TASK LIST IS EMPTY ◀◀◀\n\n"
        << "\n\n[redacted]\n" << endl;
    } else {
        cout << "\n\t\t\t\t\t▶ CURRENT TASKS ◀ \n" << endl;
        for (int i = 0; i < taskCount; ++i) {
            cout << "\t" << (i + 1) << ". "
            << taskList[i].getTaskName() << endl;
        }

        int delIndex;
        cout << "\n\tEnter the number of the task to delete: ";
        cin >> delIndex;
        cin.ignore();

        if (delIndex < 1 || delIndex > taskCount) {
            cout << "\n\t\t  ▶▶▶ ERROR: INVALID TASK NUMBER ◀◀◀\n"
            << "\n[redacted]\n" << endl;
        } else {
            for (int i = delIndex - 1; i < taskCount - 1; ++i) {
                taskList[i] = taskList[i + 1];
            }
            taskList[taskCount - 1] = Task();
            taskCount--;

            cout << "\n\t\t  ▶▶▶ TASK DELETED SUCCESSFULLY ◀◀◀\n"
            << "\n[redacted]\n" << endl;
        }
    }
    break;


    // Checks if Task Array has at least 1 Task Object to display
    // Sends user back to Menu if there are 0 Task objects
    // Displays the name of all tasks currently present in the Task Array
case 7:
    if (taskCount == 0) {
        cout << "\n\t\t  ▶▶▶ ERROR: TASK LIST IS EMPTY ◀◀◀\n\n"
        << "\n\n[redacted]\n" << endl;
    } else {
        cout << "\n\t\t\t\t\t▶ CURRENT TASKS ◀ \n" << endl;
        for (int i = 0; i < taskCount; ++i) {
            cout << "\t" << (i + 1) << ". "
            << taskList[i].getTaskName() << "\n";
        }
        cout << "\n[redacted]\n" << endl;
    }
    break;
```

```
    // Checks if Task Array has at least 1 Task Object to search the name of and retrieve info for
    // Sends user back to Menu if there are 0 Task objects
    // Displays the name, description, and priority level of the task searched for
case 8:
    if (taskCount == 0) {
        cout << "\n\t\t  ▶▶▶ ERROR: TASK LIST IS EMPTY ◀◀◀\n\n"
        << "\n\n                                                                    \n" << endl;
    } else {
        string target;
        cout << "\n\tEnter the task name to search for: ";
        getline(cin, target);

        sortTasksByName(taskList, taskCount); // Ensuring that name is sorted 1st and foremeost

        int result = binarySearch(taskList, taskCount, target);
        if (result != -1) {
            cout << "\n\t\t  ▶▶▶ TASK FOUND ◀◀◀\n\n"
            << "\t• Task: " << taskList[result].getTaskName()
            << "\n\t• Description: " << taskList[result].getTaskDesc()
            << "\n\t• Priority Level: " << taskList[result].getTaskPri()
            << "\n\n                                                                    \n" << endl;
        } else {
            cout << "\n\t\t\t  ▶▶▶ TASK NOT FOUND ◀◀◀\n"
            << "\n                                                                    \n" << endl;
        }
    }
    break;
```

```
    // Checks if Task Array has at least 1 Task Object to save the data of
    // Sends user back to Menu if there are 0 Task objects
    // Saves the name, description, and priority level of the task/s in a customizable file name
    // Can overwrite currently selected file
    // if there are any issues opening and writing to file, error message is posted & user is sent back to menu
    // Offers the user the file path if desired, and displays confirmation of save
case 9:
    if (taskCount == 0) {
        cout << "\n\t\t  ▶▶▶ ERROR: NO TASKS TO SAVE ◀◀◀\n\n"
        << "\n\n                                                    \n" << endl;
    } else {
        string filename;
        cout << "\n\tEnter filename to save to (Ex: tasks.txt): ";
        getline(cin, filename);

        ofstream fout(filename);

        if (!fout) {
            cout << "\n\t\t  ▶▶▶ ERROR: COULD NOT CREATE FILE ◀◀◀\n\n"
            << "\n                                                    \n" << endl;
        } else {
            for (int i = 0; i < taskCount; ++i) {
                fout << "Task " << (i + 1) << ":\n";
                fout << "Name: " << taskList[i].getTaskName() << "\n";
                fout << "Description: " << taskList[i].getTaskDesc() << "\n";
                fout << "Priority: " << taskList[i].getTaskPri() << "\n";
                fout << "--------------------------\n";
            }
            fout.close();

            short choice = 0;
            cout << "\t*** Show file path? (0/1): ";
            cin >> choice;
            cin.ignore();

            if (choice == 1){
                filesystem::path currentPath = filesystem::current_path();
                string pathString = currentPath.string();
                cout << "\n\t*** " << pathString << endl;
            }

            cout << "\n\t\t  ▶▶▶ TASKS SAVED TO: " << filename << " ◀◀◀\n"
            << "\n                                                    \n" << endl;
        }
    }
    break;
```

```
            // Ends program and displays confirmation to user
        case 10:
            cout << "\n\t\t  ▶▶▶ SUCCESSFULLY ENDED TaskManager™ ◀◀◀\n\n"
            << "                                                                \n" << endl;
            return 0;


            // secret
        case 11:
            secret();
            break;

            // default case if anything breaks
        default:
            cout << "\n\t\t  ▶▶▶ ERROR: INVALID OPTION ◀◀◀\n\n";
    }
  }
}
```

Steps for running the GUI:

1.) cd /Users/nicholasrubio/TaskManagerGUI
2.) Enter: ./TaskManagerGUI

TaskManager™'s GUI is only capable of:

- Creating a new Task
- Naming the Task
- Describing the Task
- Setting the Priority Level of the Task
- Saving the session

* It's important to note that despite TaskManager™' GUI limitation, IT STILL follows the constraints set fourth by the source code from TaskManager™ main.cpp file. Meaning, it still:

- Checks character length constraint of name
- Checks character length constraint of description
- Ensures priorityLevel is within range (1–10)
- Null checking is built in
- Warning/Error messages are displayed to user

```
#include <wx/wx.h>
#include "Task.h" // <- Task.h is a copy/paste of the Task class from TaskManager™

#include <vector> // Using vectors here to dynamically change the size of taskList

std::vector<Task> taskList;

class MyApp : public wxApp {
public:
    virtual bool OnInit();
};

class MyFrame : public wxFrame {
public:
    MyFrame(const wxString& title);

private:
    wxTextCtrl* titleInput;
    wxTextCtrl* descInput;
    wxTextCtrl* priInput;

    void OnCreateTask(wxCommandEvent& event);
};

wxIMPLEMENT_APP(MyApp);

bool MyApp::OnInit() {
    MyFrame *frame = new MyFrame("TaskManager™ GUI");// Added my Program's name on the window
    frame->Show(true);
    return true;
}
```

```
MyFrame::MyFrame(const wxString& title)
    : wxFrame(NULL, wxID_ANY, title, wxDefaultPosition, wxSize(400, 300))
{
    wxPanel* panel = new wxPanel(this, wxID_ANY);

    wxStaticText* titleLabel = new wxStaticText(panel, wxID_ANY, "Title (max 20):", wxPoint(20, 20));      // Set Title
    titleInput = new wxTextCtrl(panel, wxID_ANY, "", wxPoint(150, 20), wxSize(200, -1));

    wxStaticText* descLabel = new wxStaticText(panel, wxID_ANY, "Description (max 75):", wxPoint(20, 60)); // Set Description
    descInput = new wxTextCtrl(panel, wxID_ANY, "", wxPoint(150, 60), wxSize(200, -1));

    wxStaticText* priLabel = new wxStaticText(panel, wxID_ANY, "Priority (1-10):", wxPoint(20, 100));      // Set Priority
    priInput = new wxTextCtrl(panel, wxID_ANY, "", wxPoint(150, 100), wxSize(200, -1));

    wxButton* createButton = new wxButton(panel, wxID_ANY, "Create Task", wxPoint(150, 150));              // Create Task Button
    createButton->Bind(wxEVT_BUTTON, &MyFrame::OnCreateTask, this);
}
```

```
void MyFrame::OnCreateTask(wxCommandEvent& event) {
    wxString title = titleInput->GetValue();
    wxString desc = descInput->GetValue();
    wxString priStr = priInput->GetValue();

    if (title.IsEmpty() || title.Length() > 20) {                                   // Error checker for Title
        wxMessageBox("Title must be 1-20 characters.", "Error", wxOK | wxICON_ERROR);
        return;
    }
    if (desc.IsEmpty() || desc.Length() > 75) {                                     // Error checker for Description
        wxMessageBox("Description must be 1-75 characters.", "Error", wxOK | wxICON_ERROR);
        return;
    }

    long priority;
    if (!priStr.ToLong(&priority) || priority < 1 || priority > 10) {               // Error checker for Priority
        wxMessageBox("Priority must be a number from 1-10.", "Error", wxOK | wxICON_ERROR);
        return;
    }


    Task t(std::string(title.mb_str()), std::string(desc.mb_str()), (short)priority);       // Task Constructor
    taskList.push_back(t);                                                                  // taskList grows by one(1)

    wxMessageBox("Task created successfully!", "Success", wxOK | wxICON_INFORMATION);       // Confirmation message


    titleInput->Clear(); // clear title input
    descInput->Clear();  // clear description input
    priInput->Clear();   // clear priority input
}
```


THANK YOU