main.cxx

```cpp
#include <cxxopts.hpp> // command line argument parser library written by jarro2783 (https://github.com/jarro2783/cxxopts)
#include <ui.hxx>

auto main(int argc, char** argv) -> int{
    // flags for program arguments
    bool upAlphaFlag = false;
    bool lowAlphaFlag = false;
    bool numFlag = false;
    bool specialCharFlag = false;
    bool guiFlag = false;

    // other variable change by the flags
    unsigned int length = 0;
    int state = -1; // execution state (-1 for init value)
    bool flagEnabled = false;

    // specifies options
    cxxopts::Options options("APCSP Create Task", "Random Password Generator");
    options.add_options()
        ("A,upper", "Include upper case alphabets")
        ("a,lower", "Include lower case alphabets")
        ("n,number","Include numbers")
        ("s,special", "Include special characters")
        ("g,gui", "Run the GUI version of program, other passed arguments are ignored")
        ("h,help", "Print help")
        ("l,length", "Set the length of password", cxxopts::value<int>())
    ;

    // parse options from argv with fail safe
    cxxopts::ParseResult result;
    try {
        result = options.parse(argc, argv);
    } catch (const cxxopts::exceptions::parsing &e) {
        printHelp();
        std::cerr << e.what() << "\n"; // show error
        return FAIL;
    }

    // sets flags and option value from options(argc and argv)
    // help flag (show help doc and exit)
    if (result.count("help")) {
        printHelp();
        return SUCCESS;
    }

    // GUI flag
    if (result.count("gui")) guiFlag = true;

    // char flags
    if (result.count("upper")) upAlphaFlag = true;
    if (result.count("lower")) lowAlphaFlag = true;
```

```
    if (result.count("number")) numFlag = true;
    if (result.count("special")) specialCharFlag = true;
    flagEnabled = checkFlags(upAlphaFlag, lowAlphaFlag, numFlag,
    specialCharFlag);

    // length option
    if (result.count("length") && !guiFlag && flagEnabled) {
        length = result["length"].as<int>();
    } else if (!guiFlag && flagEnabled) { // print out error when
    no character flag is enabled but only length option specified
        printHelp();
        printLine("Error: no length argument");
        return FAIL;
    }

    // run GUI version if -g option is present
    if (guiFlag) {
        state = rungui();
    } else if (length > 0 && flagEnabled){ // otherwise run CUI
    version
        state = runcui(length, upAlphaFlag, lowAlphaFlag, numFlag,
    specialCharFlag);
    } else { // if no argument specified, report error and set
    execution state 1 (failure)
        printHelp();
        printLine("Error: No argument specified");
        state = FAIL;
    }
    return state;
}
```

passgen.hxx

```
#ifndef PASSGEN_HXX
#define PASSGEN_HXX

#include <cstdlib>
#include <ctime>
#include <exception>

namespace PassGen {
    /**
    * @brief Get the list of upper alphabet letters
    * @return char* - List of upper alphabet letters
    */
    auto getUpperAlpha() -> char*;

    /**
    * @brief Get the list of lower alphabet letters
    * @return char* - List of lower alphabet letters
    */
    auto getLowerAlpha() -> char*;

    /**
    * @brief Get the list of numbers
    * @return char* - List of numbers
    */
```

```
25      auto getNumber() -> char*;
26
27      /**
28      * @brief Get the list of special characters
29      * @return char* - List of special characters
30      */
31      auto getSpecialChars() -> char*;
32
33      /**
34      * @brief generate the password from list of characters and
        specified length
35      * @param charList list of characters (pure C string)
36      * @param len length of password to be generated (int)
37      * @return (char*) - generated password
38      */
39      auto passGen(const char *charList, const unsigned int& len) ->
        char*;
40
41      // exceptions for the functions in PassGen namespace
42      namespace exceptions {
43          // base exception class
44          class exception : public std::exception {
45              public:
46                  // constructor (sets m_message from msg argument)
47                  explicit exception(char* msg) {m_message = msg;};
48                  /**
49                  * @brief show the content of the error message
50                  * @return (char*) pure C string of error message
51                  */
52                  auto what() -> char* {return m_message;};
53              private:
54                  char* m_message = nullptr; // error message (init
        with nullptr)
55          };
56
57          // exception to be thrown when something is failed to
        allocate its memory
58          class memoryAllocationFailiure : public exception {
59              public:
60                  explicit memoryAllocationFailiure() : exception((
        char*)"Couldn't allocate memory space!") {};
61                  // call construct from the base class supplied with
         specified error message
62          };
63
64          // exception to be thrown when there is no content in
        charList (PassGen::passgen())
65          class noCharList : public exception {
66              public:
67                  explicit noCharList() : exception((char*)"No
        character list specified!") {};
68                  // call construct from the base class supplied with
         specified error message
69          };
70
71          // exception to be used when unknown error occured
72          class unknownError : public exception {
```

```
            public:
                explicit unknownError() : exception((char*)"Unknown Error Caught : Mark me 0 :(") {};
                // call construct from the base class supplied with specified error message
            };
    } // PassGen::exceptions
} // PassGen

#endif // PASSGEN_HXX
```

passgen.cxx

```
#include <utils.hxx>
#include <passgen.hxx>

auto PassGen::getLowerAlpha() -> char* {
    const int numOfLetters = 26;
    char* output = nullptr; // initialize pointer
    output = (char*)malloc(numOfLetters * sizeof(char) + 1); // allocate memory for 26 letters and a terminate character
    if (output == nullptr) {throw PassGen::exceptions::memoryAllocationFailiure(); return nullptr;} // check if memory allocation is failed
    const int offset = 97; // 97th letter in ASCII (a)
    // adds 26 letters (a-z)
    for (int i = 0; i < numOfLetters; i++) {
        output[i] = (char)(offset + i);
    }
    output[numOfLetters] = '\0'; // add terminate character at end
    return output;
}

auto PassGen::getUpperAlpha() -> char* {
    const int numOfLetters = 26;
    char* output = nullptr; // initialize pointer
    output = (char*)malloc(numOfLetters * sizeof(char) + 1); // allocate memory for 26 letters + terminate character
    if (output == nullptr) {throw PassGen::exceptions::memoryAllocationFailiure(); return nullptr;} // check if memory allocation is failed
    const int offset = 65; // 65th letter in ASCII (A)
    // adds 26 letters (A-Z)
    for (int i = 0; i < numOfLetters; i++) {
        output[i] = (char)(offset + i);
    }
    output[numOfLetters] = '\0'; // add terminate character at end
    return output;
}

auto PassGen::getNumber() -> char* {
    const int numOfLetters = 10;
    char* output = nullptr; // initialize pointer
    output = (char*)malloc((numOfLetters) * sizeof(char) + 1);
    if (output == nullptr) {throw PassGen::exceptions::memoryAllocationFailiure(); return nullptr;} // check if memory allocation is failed
```

```
    const int offset = 48; // 48th letter in ASCII (0)
    // adds 10 letters (0-9)
    for (int i = 0; i < numOfLetters; i++) {
        output[i] = (char)(offset + i);
    }
    output[numOfLetters] = '\0'; // add terminate character at end
    return output;

}

auto PassGen::getSpecialChars() -> char* {
    const int numOfLetters = 31; // 31 symbols
    char* output = nullptr; // initialize pointer
    output = (char*)malloc((numOfLetters) * sizeof(char) + 1);
    if (output == nullptr) {throw PassGen::exceptions::
    memoryAllocationFailiure(); return nullptr;} // check if memory
     allocation is failed

    int ind = 0; // index in the output list

    // loop config and range exclusion config
    const int start = 33; // loop through ASCII #33
    const int end = 127; // to #127

    const int numStart = 48; // ASCII range that represents number
    (#48
    const int numEnd = 57; // to #57)

    const int upperStart = 65; // ASCII range that represents upper
     case alphabets (#65
    const int upperEnd = 90; // to #90)

    const int lowerStart = 97; // ASCII range that represents lower
     case alphabets (#97
    const int lowerEnd = 122; // to #122)

    // adds special characters to output
    for (int i = start; i < end; i++) {
        if ((numStart <= i && i <= numEnd) || (upperStart <= i && i
     <= upperEnd) || (lowerStart <= i && i <= lowerEnd)) {
            continue; // skip at the number, upper case and lower
    case alphabets
        }
        output[ind] = (char)(i);
        ind++;
    }
    output[numOfLetters] = '\0'; // add terminate character at end
    return output;
}

auto PassGen::passGen(const char *charList, const unsigned int& len
    ) -> char* {
    if (strSize(charList) == 0) {throw PassGen::exceptions::
    noCharList(); return nullptr;}

    std::srand(time(nullptr)); // set random seed to current time

```

```
85      unsigned int randomCharPos = 0; // position of charList which
        will be randomly selected
86      const char termChar = '\0';
87      const char backSlash = '\\';
88      const int charListSize = strSize(charList);
89      char currentLetter = 0;
90      char previousLetter = 0;
91
92      char* output = new char[len+1]; // length of password + 1
        terminating char
93      // return null pointer on the failiure of memory allocation
94      if (output == nullptr) {throw PassGen::exceptions::
        memoryAllocationFailiure(); return nullptr;}
95
96      while (strSize(output) != len) { // to make sure output is in
        desired length
97      for (int i = 0; i <= len; i++) {
98          if (i == len) {output[i] = termChar;} // ends with
        terminating char
99          else { // adds other chars otherwise
100
101         // adds random character from charList
102         randomCharPos = std::rand()%charListSize;
103         output[i] = charList[randomCharPos];
104
105         // set current and previous letter for check
106         currentLetter = output[i];
107         previousLetter = output[i-1];
108
109         // checks escape sequences which causes issues
110         while (previousLetter == backSlash &&
111         ((currentLetter == 'a') || // '\a'
112         (currentLetter == 'b') || // '\b'
113         (currentLetter == 'c') || // '\c'
114         (currentLetter == 'n') || // '\n'
115         (currentLetter == 'f') || // '\f'
116         (currentLetter == 'r') || // '\r'
117         (currentLetter == 't') || // '\t'
118         (currentLetter == 'U') || // '\U'
119         (currentLetter == 'u') || // '\u'
120         (currentLetter == 'v') || // '\v'
121         (currentLetter == 'x') // '\x'
122         )) {
123             // regenerate random letter
124             randomCharPos = std::rand()%charListSize;
125             output[i] = charList[randomCharPos];
126
127             // re-set current letter
128             currentLetter = output[i];
129         }}
130     }}
131     return output;
132 }
```

utils.hxx

```
1 #ifndef UTILS_HXX
```

```
#define UTILS_HXX

#include <iostream>

/**
* @brief return the size(length) of string (pure C char list)
* @param str pure C string to be measured
* @return (int) - size(length) of string
*/
inline auto strSize(const char *str) -> int {
    int out = 0;
    int index = 0;
    while (str[index] != 0) {
        index++;
        out++;
    }
    return out;
}

/**
* @brief print the passed in argument
* @param object takes any type of input that is able to stdout to
    the console
* @return (void) - console output of the object
*/
template<class T>
inline auto printLine(T object) -> void {
    std::cout << object << std::endl;
}

/**
* @brief prints the help for the console application
* @return (void) - console output of help document
*/
inline auto printHelp() -> void {
    printLine("APCSPCreateTask - Random Password Generator\n");
    printLine("[Usage]: APCSPCreateTask [-A -a -n -s -g] -l <length
    >\n");
    printLine("[Options]:\n");
    printLine("\t-A, --upper : include upper case alphabets in
    password\n");
    printLine("\t-a, --lower : include lower case alphabets in
    password\n");
    printLine("\t-n, --number : include numbers in password\n");
    printLine("\t-s, --special  : include special characters in
    password\n");
    printLine("\t-l, --length <number> : set the length of the
    password\n");
    printLine("\t-g, --gui : run in GUI regardless of the previous
    options\n");
    printLine("\t-h, --help : print this help\n");
}

/**
* @brief checks if any flag is enabled
* @param upper boolean flag for upper case letters
* @param lower boolean flag for lower case letters
```

```
* @param num boolean flag for numbers
* @param special boolean flag for special characters
* @return (bool) - returns true if one of any flag is enabled
*/
inline auto checkFlags(bool upper, bool lower, bool num, bool special) -> bool {
    bool isEnabled = (upper || lower || num || special);
    return isEnabled;
}

#endif // UTILS_HXX
```

ui.hxx

```
#ifndef UI_HXX
#define UI_HXX

// enumeration of states of program
enum PROGRAMSTATE : int {
    SUCCESS, // 0
    FAIL // 1
};

#include <string>
#include <cstring>
#include <utils.hxx>
#include <passgen.hxx>
#include <gtkui.hxx>

/**
* @brief runs the CUI version of program
* @param len length of the password (int)
* @param upper boolean flag for upper case alphabets
* @param lower boolean flag for lower case alphabets
* @param num boolean flag for numbers
* @param special boolean flag for special characters
* @return (int) - execution state of program
*/
inline auto runcui(const unsigned int& len, const bool& upper, const bool& lower, const bool& num, const bool& special) -> int {
    // appends letters to input
    std::string input;
    try {
        if (upper) {input += PassGen::getUpperAlpha();}
        if (lower) {input += PassGen::getLowerAlpha();}
        if (num) {input += PassGen::getNumber();}
        if (special) {input += PassGen::getSpecialChars();}
    }
    // print out error when any of getChars threw exception
    catch (PassGen::exceptions::memoryAllocationFailiure &err) {
        printLine(err.what());
        return FAIL;
    }

    // converts to std::string to pure C string
    char *cInput = new char[input.length() + 1];
```

```
    strcpy(cInput, input.c_str());

    // generate password, report error and exit with status of 1 (failiure) if any caught
    char *out = nullptr;
    try {
        out = PassGen::passGen(cInput, len);
    }
    // print out error when PassGen::passGen threw exception
    catch (PassGen::exceptions::memoryAllocationFailiure &err) {
        printLine(err.what());
        return FAIL;
    }
    catch (PassGen::exceptions::noCharList &err) {
        printLine(err.what());
        return FAIL;
    }
    catch (...) {
        PassGen::exceptions::exception error = PassGen::exceptions::unknownError();
        printLine(error.what());
        return FAIL;
    }

    // prints password, and exit with status of 0 (success)
    std::cout << out << std::endl;
    return SUCCESS;
}

/**
* @brief runs the GUI version of program
* @return (int) - execution state of program
*/
inline auto rungui() -> int {
    // run the GTK application
    auto app = Gtk::Application::create("apcsp.passgen");
    return app->make_window_and_run<PassGenUI>(0,nullptr); // run GTK app with no arguments
}

#endif // UI_HXX
```

gtkui.hxx

```
#ifndef GTKUI_HXX
#define GTKUI_HXX

#include <gtkmm.h> // GTK GUI Library (C++ wrapper)
#include <passgen.hxx>

class PassGenUI : public Gtk::Window
{
    public:
        PassGenUI(); // constructor
        ~PassGenUI() override;// destructor

        /**
```

```
        * @brief The button event for m_generate_button
        * @return (void) Generate password, set to m_output_buffer, and show to the user
        */
        auto on_generate_button_clicked() -> void; // button event

        /**
        * @brief Show error dialog with exceptions and message
        * @param err PassGen::exceptions::exception exception to be reported
        * @param extraMsg Glib::ustring Extra message to be shown along side the reported exception, set empty by default
        * @return (void) Show the dialog
        */
        auto showErrorDialog(PassGen::exceptions::exception &err, Glib::ustring extraMsg) -> void;

    private:
        const int winHeight = 480;
        const int winWidth = 640;
        const int widgetMargin = 10;
        const int maxLength = 8192;

        // checkboxes
        Gtk::CheckButton m_upper_check, m_lower_check, m_num_check, m_special_chars_check;

        // boxes (containers)
        Gtk::Box m_char_checks, m_output_box, m_main_box;

        // title of the program
        Gtk::Label m_title;

        Gtk::Button m_generate_button;

        // length input
        Gtk::SpinButton m_num_input;

        // sets range for m_num_input (0-maxLength)
        Glib::RefPtr<Gtk::Adjustment> m_num_input_adj = Gtk::Adjustment::create(0, 0, maxLength);

        // Privides CSS to the GTK widget
        Glib::RefPtr<Gtk::CssProvider> m_output_style = Gtk::CssProvider::create();

        // Text area for output
        Gtk::ScrolledWindow m_output_scroll;
        Gtk::TextView m_output;

        // text buffer for m_output
        Glib::RefPtr<Gtk::TextBuffer> m_output_buffer = Gtk::TextBuffer::create();
};

#endif // GTKUI_HXX
```

gtkui.cxx

```cpp
#include "passgen.hxx"
#include "utils.hxx"
#include <gtkui.hxx>

PassGenUI::PassGenUI():
    // initialize widgets
    m_generate_button("Generate"),
    m_main_box(Gtk::Orientation::VERTICAL,widgetMargin),
    m_char_checks(Gtk::Orientation::VERTICAL,widgetMargin),
    m_output_box(Gtk::Orientation::VERTICAL, widgetMargin),
    m_upper_check("Include Upper Case Letters"),
    m_lower_check("Include Lower Case Letters"),
    m_num_check("Include Numbers"),
    m_special_chars_check("Include Special Characters"),
    m_title("Password Generator")
{
    // set window props
    set_title("AP CSP Create Task - Password Generator");
    set_default_size(winWidth,winHeight);
    // link the button event to the function
    m_generate_button.signal_clicked().connect(sigc::mem_fun(*this,
     &PassGenUI::on_generate_button_clicked));

    // populate the widgets and other boxes in main box
    set_child(m_main_box);
    m_main_box.set_margin(widgetMargin);
    m_num_input.set_adjustment(m_num_input_adj); // set the range
    of m_num_input widget can handle
    m_main_box.append(m_title);
    m_main_box.append(m_char_checks);
    m_main_box.append(m_num_input);
    m_main_box.append(m_generate_button);
    m_main_box.append(m_output_box);

    // populate the widgets in character configuration section
    m_char_checks.append(m_upper_check);
    m_char_checks.append(m_lower_check);
    m_char_checks.append(m_num_check);
    m_char_checks.append(m_special_chars_check);

    // populate the widgets in ouput section
    m_output_scroll.set_child(m_output);
    m_output_scroll.set_expand();
    // set the style of m_output using properties and CSS
    m_output.set_editable(false);
    m_output.set_monospace(true);
    m_output.set_cursor_visible(false);
    m_output_style->load_from_data("#m_output {font-size: 14pt;}");
    m_output.set_name("m_output");
    m_output.get_style_context()->add_provider(m_output_style, 1);
    m_output.set_wrap_mode(Gtk::WrapMode::CHAR);
    // add widget to the box
    m_output_box.append(m_output_scroll);
}

PassGenUI::~PassGenUI() = default;
```

```
55
56 auto PassGenUI::showErrorDialog(PassGen::exceptions::exception &err
       , Glib::ustring extraMsg="") -> void {
57     Glib::RefPtr<Gtk::AlertDialog> m_Alert = Gtk::AlertDialog::
       create();
58     m_Alert->set_message("Error!");
59     m_Alert->set_detail((Glib::ustring)err.what() + "\n" + extraMsg
       );
60     m_Alert->show(*this);
61 }
62
63 auto PassGenUI::on_generate_button_clicked() -> void {
64     // appends letters to input according to the flags
65     std::string input;
66     try {
67         if (m_upper_check.get_active()) {input += PassGen::
       getUpperAlpha();}
68         if (m_lower_check.get_active()) {input += PassGen::
       getLowerAlpha();}
69         if (m_num_check.get_active()) {input += PassGen::getNumber
       ();}
70         if (m_special_chars_check.get_active()) {input += PassGen::
       getSpecialChars();}
71     }
72     // show error dialog when any of getChars threw exception
73     catch (PassGen::exceptions::memoryAllocationFailiure &err) {
74         showErrorDialog(err, "Please free some memory.");
75         return;
76     }
77
78     // convert std::string to pure C string
79     char* cInput = new char[input.length() + 1];
80     strcpy(cInput, input.c_str());
81
82     // get the length of password to be generated
83     int len = m_num_input.get_value_as_int();
84
85     // generate password
86     char* passwd = nullptr;
87     try {
88         passwd = PassGen::passGen(cInput, len);
89     }
90     // Show error dialog when PassGen::passGen threw exceptions
91     catch (PassGen::exceptions::memoryAllocationFailiure &err) {
92         showErrorDialog(err, "Please free some memory.");
93         return;
94     }
95     catch (PassGen::exceptions::noCharList &err) {
96         showErrorDialog(err, "Please check the form.");
97         return;
98     }
99     catch (...) {
100        PassGen::exceptions::exception error = PassGen::exceptions
       ::unknownError();
101        showErrorDialog(error);
102        return;
103    }
```

```
104
105
106     // Show the passwd by setting text and buffer
107     Glib::ustring output = Glib::convert(passwd, "UTF-8", "ISO
        -8859-1"); // convert to appropriate type and encoding of text
108     m_output_buffer->set_text(output);
109     m_output.set_buffer(m_output_buffer);
110 }
```