main.cxx

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string>
#include <utils.hxx>
#include <ui.hxx>


int main(int argc, char** argv) {
    // flags for program arguments
    bool upAlphaFlag = false;
    bool lowAlphaFlag = false;
    bool numFlag = false;
    bool specialCharFlag = false;
    int length = -1;
    int arg;
    int state = 0;

    while ((arg = getopt (argc, argv, ":lAansh")) != -1) {
        switch (arg) {
            case 'A':
                upAlphaFlag = true;
                break;
            case 'a':
                lowAlphaFlag = true;
                break;
            case 'n':
                numFlag = true;
                break;
            case 's':
                specialCharFlag = true;
                break;
            case 'l':
                length = std::atoi(optarg);
                break;
            case 'h':
                printhelp();
                return 0;
            case '?':
                if (optopt == 'l') {
                    printf("Error: No length specified\n");
                } else {
                    printhelp();
                    printf("Error: Unknown Option: %c", optopt);
                }
                return 1;
            default:
                printhelp();
                abort();
        }
    }

    if (length < 0 && checkFlags(upAlphaFlag, lowAlphaFlag, numFlag
    , specialCharFlag) == false) {
        state = rungui(argc, argv);
    } else {
        state = runcui(length, upAlphaFlag, lowAlphaFlag, numFlag,
    specialCharFlag);
    }
    return state;
}
```

passgen.hxx

```cpp
#ifndef PASSGEN
#define PASSGEN

#include <stdlib.h>
#include <stdio.h>

namespace PassGen {
    char* getUpperAlpha();
    char* getLowerAlpha();
    char* getNumber();
    char* getSpecialChars();
    char* passGen(char* charList, int len);
}

#endif // PASSGEN
```

passgen.cxx

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

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

// getSpecialChars - get the lower case alphabets
// void : takes nothing
// return (char*) : the string with all lower case alphabets in standard ASCII
char* PassGen::getLowerAlpha() {
    char* output = new char[26]; // 26 letters
    if (output == NULL) {return 0;} // check if memory allocation is failed
    int offset = 97; // 97th letter in ASCII (a)
    for (int i = 0; i < 26; i++) {
        output[i] = offset + i;
    }
    return output;
}

// getUpperAlpha - get the upper case alphabets
// void : takes nothing
// return (char*) : the string with all upper case alphabets in standard ASCII
char* PassGen::getUpperAlpha() {
    char* output = new char[26]; // 26 letters
    if (output == NULL) {return 0;} // check if memory allocation is failed
    int offset = 65; // 65th letter in ASCII (A)
    for (int i = 0; i < 26; i++) {
        output[i] = offset + i;
    }
    return output;
}

// getSpecialChars - get the numbers
// void : takes nothing
// return (char*) : the string with all numbers in standard ASCII
char* PassGen::getNumber() {
    char* output = new char[10]; // 10 letters
    if (output == NULL) {return 0;} // check if memory allocation is failed
    int offset = 48; // 48th letter in ASCII (0)
    for (int i = 0; i < 10; i++) {
        output[i] = offset + i;
    }
    return output;

}

// getSpecialChars - get the special characters
// void : takes nothing
// return (char*) : the string with all special characters in
```

```
    standard ASCII
char* PassGen::getSpecialChars() {
    char* output = new char[14+7+6+4]; // 31 symbols
    if (output == NULL) {return 0;} // check if memory allocation is failed
    int ind = 0;
    for (int i = 33; i < 127; i++) {
        if ((i >= 48 && i <= 57) || (i >= 65 && i <= 90) || (i >= 97 && i <= 122)) {
            continue;
        }
        output[ind] = i;
        ind++;
    }
    return output;
}

// passGen - Password Generator
// charList (char*) : list of char to be used in password generation
// len (int) : length of password
// return (char*) : the generated password
char* PassGen::passGen(char *charList, const int len) {
    std::srand(time(nullptr));
    unsigned int index;
    const char termChar = '\0';
    const char backSlash = '\\';
    char* output = new char[len+1]; // length of password + 1 terminating char
    if (output == NULL) {return 0;} // return 0 on the failiure of memory allocation
    while (strSize(output) != len) {
    for (int i = 0; i <= len; i++) {
        if (i == len) {output[i] = termChar;}
        else {
        index = std::rand()%(strSize(charList));
        output[i] = charList[index];
        char currentLetter = output[i];
        char previousLetter = output[i-1];
        // checks escape sequences which causes issues
        while (
            (previousLetter == backSlash && currentLetter == 'n') ||
            (previousLetter == backSlash && currentLetter == 'a') ||
            (previousLetter == backSlash && currentLetter == 'b') ||
            (previousLetter == backSlash && currentLetter == 'r') ||
            (previousLetter == backSlash && currentLetter == 't') ||
            (previousLetter == backSlash && currentLetter == 'v') ||
            (previousLetter == backSlash && currentLetter == 'f') ||
            (previousLetter == backSlash && currentLetter == 'u') ||
            (previousLetter == backSlash && currentLetter == 'U')
    ||
            (previousLetter == backSlash && currentLetter == 'x')
    ||
            (previousLetter == backSlash && currentLetter == 'c'))
    {
                printf("ESC warn: %c\n", currentLetter);
                index = std::rand()%(strSize(charList));
                output[i] = charList[index];
                currentLetter = output[i];
            }
        }
    }
    }
    return output;
}
```

utils.hxx

```
#ifndef UTILS
#define UTILS

#include <stdlib.h>

inline int strSize(char* a) {
    int out = 0;
    int i = 0;
    while (a[i] != 0) {
        i++;
        out++;
    }
    return out;
}

inline void printhelp() {
    printf("APCSPCreateTask - Random Password Generator\n\n");
    printf("[Usage]: APCSPCreateTask [-A -a -n -s] -l <length>\n\n"
    );
    printf("[Arguments]:\n\n");
    printf("\t-A : include upper case alphabets in password\n\n");
    printf("\t-a : include lower case alphabets in password\n\n");
    printf("\t-n : include numbers in password\n\n");
    printf("\t-s : include special characters in password\n\n");
    printf("\t-l <number> : set the length of the password\n\n");
    printf("\t-h : print this help\n\n");
}

inline bool checkFlags(bool up, bool low, bool num, bool spec) {
    int count = 0;
    if (up == true) {count++;}
    if (low == true) {count++;}
    if (num == true) {count++;}
    if (spec == true) {count++;}
    return count > 0 ? true : false;
}

#endif // UTILS
```

ui.hxx

```cpp
#ifndef UI
#define UI

#include <iostream>
#include <string>
#include <cstring>
#include <utils.hxx>
#include <passgen.hxx>
#include <gtkui.hxx>
using namespace PassGen;

int runcui(int len, bool up, bool low, bool num, bool special) {

    std::string input;

    if (len < 0) {
        printhelp();
        printf("\nError: Length not specified\n");
        return 1;
    }

    if (checkFlags(up, low, num, special) == false) {
        printhelp();
        printf("\nError: No character flag(s) specified\n");
        return 1;
    }

    if (up == true) {input += getUpperAlpha();}
    if (low == true) {input += getLowerAlpha();}
    if (num == true) {input += getNumber();}
    if (special == true) {input += getSpecialChars();}

    char *cInput = new char[input.length() + 1];
    strcpy(cInput, input.c_str());
    char *out = passGen(cInput, len);
    std::cout << out << std::endl;
    return 0;
}

int rungui(int argc, char** argv) {
    auto app = Gtk::Application::create("io.apcsp.passgen");
    return app->make_window_and_run<PassGenUI>(argc, argv);
}

#endif // UI
```

gtkui.hxx

```cpp
#ifndef GTKUI
#define GTKUI

#include <gtkmm.h>
#include <passgen.hxx>
#include <string>
#include <cstring>

class PassGenUI : public Gtk::Window
{
    public:
        PassGenUI();
        ~PassGenUI();
        void on_generate_button_clicked();
    private:
        Gtk::CheckButton m_upper_check, m_lower_check, m_num_check, m_special_chars_check;
        Gtk::Box m_char_checks, m_output_box, m_main_box;
        Gtk::Label m_title;
        Gtk::Button m_generate_button;
        Gtk::SpinButton m_num_input;
        Glib::RefPtr<Gtk::Adjustment> m_num_input_adj = Gtk::Adjustment::create(0, 0, 8192);
        Gtk::ScrolledWindow m_output_scroll;
        Gtk::TextView m_output;
        Glib::RefPtr<Gtk::TextBuffer> m_output_buffer = Gtk::TextBuffer::create();
};

#endif // GTKUI
```

gtkui.cxx

```cpp
#include <gtkui.hxx>
#include <iostream>

PassGenUI::PassGenUI():
m_generate_button("Generate"),
m_main_box(Gtk::Orientation::VERTICAL,10),
m_char_checks(Gtk::Orientation::VERTICAL,10),
m_output_box(Gtk::Orientation::VERTICAL, 10),
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_title("AP CSP Create Task - Password Generator");
    set_default_size(600,400);
    m_generate_button.signal_clicked().connect(sigc::mem_fun(*this,
     &PassGenUI::on_generate_button_clicked));

    // populate the widgets
    set_child(m_main_box);
    m_main_box.set_margin(10);
    m_num_input.set_adjustment(m_num_input_adj);
    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);

    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);

    m_output_scroll.set_child(m_output);
    m_output_scroll.set_expand();
    m_output.set_editable(false);
    m_output.set_wrap_mode(Gtk::WrapMode::CHAR);
    m_output_box.append(m_output_scroll);
    m_output.set_buffer(m_output_buffer);
}

PassGenUI::~PassGenUI() {
}

void PassGenUI::on_generate_button_clicked() {
    std::string input;
    if (m_upper_check.get_active() == true) {input += PassGen::
    getUpperAlpha();}
    if (m_lower_check.get_active() == true) {input += PassGen::
    getLowerAlpha();}
    if (m_num_check.get_active() == true) {input += PassGen::
    getNumber();}
    if (m_special_chars_check.get_active() == true) {input +=
    PassGen::getSpecialChars();}
    char* cInput = new char[input.length() + 1];
    strcpy(cInput, input.c_str());
    int len = m_num_input.get_value_as_int();
    Glib::ustring output = Glib::convert(PassGen::passGen(cInput,
    len), "UTF-8", "ISO-8859-1");
    m_output_buffer->set_text(output);
    m_output.set_buffer(m_output_buffer);
}
```