main.cxx

```cpp
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string>
#include <cstring>
#include <passgen.hxx>
using namespace PassGen;


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");
}

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;
}

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;

    std::string input;

    while ((arg = getopt (argc, argv, "Aanshl:")) != -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 if (isprint(optopt)){
                    printf("Error: Unknown option: -%c\n", optopt);
                } else {
                    printf("Error: Unknown value: -%c\n", optopt);
                }
                return 1;
            default:
                printhelp();
                abort();
        }
    }

    if (length < 0 && checkFlags(upAlphaFlag, lowAlphaFlag, numFlag
    , specialCharFlag) == false) {
        printhelp();
        printf("\nError: No option specified\n");
        return 1;
    }

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

    if (checkFlags(upAlphaFlag, lowAlphaFlag, numFlag,
    specialCharFlag) == false) {
        printhelp();
        printf("\nError: No character flag(s) specified\n");
        return 1;
    }

    if (upAlphaFlag == true) {input += getUpperAlpha();}
    if (lowAlphaFlag == true) {input += getLowerAlpha();}
    if (numFlag == true) {input += getNumber();}
    if (specialCharFlag == true) {input += getSpecialChars();}

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

    return 0;
}
```

passgen.hxx

```h
#ifndef PASSGEN
#define PASSGEN

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

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

#endif // PASSGEN
```

passgen.cxx

```cpp
#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[42]; // 42 symbols
    if (output == NULL) {return 0;} // check if memory allocation is failed
    int offset = 33; // 33rd letter in ASCII (!)
    int listOffset = 0;
    int i;
    // ASCII range of 33 - 64 (32 symbols)
    for (i = 0; i < 32; i++) {
        output[i] = offset + i;
    }
    listOffset = 32;
    offset = 91;
    // ASCII range of 91 - 96 (6 symbols)
    for (i = 0; i < 6; i++) {
        output[i + listOffset] = offset + i;
    }
    listOffset = 38;
    offset = 123;
    // ASCII range of 123 - 126 (4 symbols)
    for (i = 0; i < 4; i++) {
        output[i + listOffset] = offset + i;
    }
    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;
    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
```

```
    for (int i = 0; i <= len; i++) {
        if (i == len) {output[i] = charList[strSize(charList)];}
        else {
        index = std::rand()%(strSize(charList));
        output[i] = charList[index];
        }
    }
    return output;
}
```

utils.hxx

```
#ifndef UTILS
#define UTILS

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

#endif // UTILS
```