main.cxx

```cpp
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#include <passgen.hxx>
using namespace PassGen;


void printhelp() {
    printf("APCSPCreateTaks - Random Password Generator\n\n");
    printf("\t[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");
}

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;

    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();
                break;
            case '?':
                if (optopt == 'l') {
                    printf("Error: no length specified");
                }
                return 1;
            default:
                abort();
        }
```

```
    }

    std::cout << "A: " << upAlphaFlag << " a: " << lowAlphaFlag <<
    " n: " << numFlag << " s: " << specialCharFlag << std::endl;
    if (upAlphaFlag == true) {std::cout << getUpperAlpha() << std::
    endl;}
    if (lowAlphaFlag == true) {std::cout << getLowerAlpha() << std
    ::endl;}
    if (numFlag == true) {std::cout << getNumber() << std::endl;}

    char* out = passGen(getUpperAlpha(), 12);
    std::cout << out << std::endl;

    return 0;
}
```

passgen.hxx

```
#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

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

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

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;
}

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;
}

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;

}

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
```

```
72      if (output == NULL) {return 0;} // return 0 on the failiure of
        memory allocation
73      for (int i = 0; i <= len; i++) {
74          if (i == len) {output[i] = charList[strSize(charList)];}
75          else {
76          index = std::rand()%(strSize(charList));
77          output[i] = charList[index];
78          }
79      }
80      return output;
81  }
```

utils.hxx

```
1   #ifndef UTILS
2   #define UTILS
3
4   int strSize(char* a) {
5       int out = 0;
6       int i = 0;
7       while (a[i] != 0) {
8           i++;
9           out++;
10      }
11      return out;
12  }
13
14  #endif // UTILS
```