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;}


    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 <passgen.hxx>

char* PassGen::getLowerAlpha() {
    char* output = (char*)malloc(26+1); // 26 letters + 1 terminating char
    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;
    }
    output[26] = '\0'; // terminating char
    return output;
}

char* PassGen::getUpperAlpha() {
    char* output = (char*)malloc(26+1); // 26 letters + 1 terminating char
    if (output == NULL) {return 0;} // check if memory allocation is failed
```

```
20      int offset = 65; // 65th letter in ASCII (A)
21      for (int i = 0; i < 26; i++) {
22          output[i] = offset + i;
23      }
24      output[26] = '\0';
25      return output;
26  }
27
28  char* PassGen::getNumber() {
29      char* output = (char*)malloc(10+1); // 10 letters + 1
        terminating char
30      if (output == NULL) {return 0;} // check if memory allocation
        is failed
31      int offset = 48; // 48th letter in ASCII (0)
32      for (int i = 0; i < 10; i++) {
33          output[i] = offset + i;
34      }
35      output[10] = '\0';
36      return output;
37
38  }
39
40  char* PassGen::getSpecialChars() {
41      char* output = (char*)malloc(42+1); // 42 symbols + 1
        terminating char
42      if (output == NULL) {return 0;} // check if memory allocation
        is failed
43      int offset = 33; // 33rd letter in ASCII (!)
44      int listOffset = 0;
45      int i;
46      // ASCII range of 33 - 64 (32 symbols)
47      for (i = 0; i < 32; i++) {
48          output[i] = offset + i;
49      }
50      listOffset = 32;
51      offset = 91;
52      // ASCII range of 91 - 96 (6 symbols)
53      for (i = 0; i < 6; i++) {
54          output[i + listOffset] = offset + i;
55      }
56      listOffset = 38;
57      offset = 123;
58      // ASCII range of 123 - 126 (4 symbols)
59      for (i = 0; i < 4; i++) {
60          output[i + listOffset] = offset + i;
61      }
62      output[42] = '\0';
63      return output;
64  }
65
66  // passGen - Password Generator
67  // charList (char*) : list of char to be used in password
        generation
68  // len (int) : length of password
69  // return (char*) : the generated password
70  char* PassGen::passGen(char* charList, int len) {
71      char* output = (char*)malloc(len+1); // length of password + 1
```

```
    terminating char
    if (output == NULL) {return 0;} // return -1 on the failiure of
     memory allocation
    for (int i = 0; i < len; i++) {
        output[i] = charList[i];
    }
    return output;
}
```