

main.cxx

```
1 #include <iostream>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <unistd.h>
5 #include <string>
6 #include <cstring>
7 #include <passgen.hxx>
8 using namespace PassGen;
9
10
11 void printhelp() {
12     printf("APCSPCreateTaks - Random Password Generator\n\n");
13     printf("\t[Arguments]:\n\n");
14     printf("\t-A : include upper case alphabets in password\n\n");
15     printf("\t-a : include lower case alphabets in password\n\n");
16     printf("\t-n : include numbers in password\n\n");
17     printf("\t-s : include special characters in password\n\n");
18 }
19
20 int main(int argc, char** argv) {
21     // flags for program arguments
22     bool upAlphaFlag = false;
23     bool lowAlphaFlag = false;
24     bool numFlag = false;
25     bool specialCharFlag = false;
26     int length = -1;
27     int arg;
28
29     std::string input;
30
31     while ((arg = getopt (argc, argv, "Aanshl:")) != -1) {
32         switch (arg) {
33             case 'A':
34                 upAlphaFlag = true;
35                 break;
36             case 'a':
37                 lowAlphaFlag = true;
38                 break;
39             case 'n':
40                 numFlag = true;
41                 break;
42             case 's':
43                 specialCharFlag = true;
44                 break;
45             case 'l':
46                 length = std::atoi(optarg);
47                 break;
48             case 'h':
49                 printhelp();
50                 break;
51             case '?':
52                 if (optopt == 'l') {
53                     printf("Error: no length specified");
54                 }
55                 return 1;
56         }
57     }
```

```

56         default:
57             abort();
58     }
59 }
60
61 std::cout << "A: " << upAlphaFlag << " a: " << lowAlphaFlag <<
62 " n: " << numFlag << " s: " << specialCharFlag << std::endl;
63 if (upAlphaFlag == true) {input += getUpperAlpha();}
64 if (lowAlphaFlag == true) {input += getLowerAlpha();}
65 if (numFlag == true) {input += getNumber();}
66 if (specialCharFlag == true) {input += getSpecialChars();}
67
68 char *cInput = new char[input.length() + 1];
69 strcpy(cInput, input.c_str());
70 char *out = passGen(cInput, length);
71 std::cout << out << std::endl;
72
73 return 0;
74 }

```

passgen.hxx

```

1 #ifndef PASSGEN
2 #define PASSGEN
3
4 #include <stdlib.h>
5 #include <stdio.h>
6
7 namespace PassGen {
8     char* getLowerAlpha();
9     char* getUpperAlpha();
10    char* getNumber();
11    char* getSpecialChars();
12    char* passGen(char* charList, int len);
13 }
14
15 #endif // PASSGEN

```

passgen.cxx

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 #include <utils.hxx>
6 #include <passgen.hxx>
7
8 // getSpecialChars - get the lower case alphabets
9 // void : takes nothing
10 // return (char*) : the string with all lower case alphabets in
11 // standard ASCII
12 char* PassGen::getLowerAlpha() {
13     char* output = new char[26]; // 26 letters
14     if (output == NULL) {return 0;} // check if memory allocation
15     is failed
16     int offset = 97; // 97th letter in ASCII (a)

```

```

15     for (int i = 0; i < 26; i++) {
16         output[i] = offset + i;
17     }
18     return output;
19 }
20
21 // getUpperAlpha - get the upper case alphabets
22 // void : takes nothing
23 // return (char*) : the string with all upper case alphabets in
    standard ASCII
24 char* PassGen::getUpperAlpha() {
25     char* output = new char[26]; // 26 letters
26     if (output == NULL) {return 0;} // check if memory allocation
    is failed
27     int offset = 65; // 65th letter in ASCII (A)
28     for (int i = 0; i < 26; i++) {
29         output[i] = offset + i;
30     }
31     return output;
32 }
33
34 // getSpecialChars - get the numbers
35 // void : takes nothing
36 // return (char*) : the string with all numbers in standard ASCII
37 char* PassGen::getNumber() {
38     char* output = new char[10]; // 10 letters
39     if (output == NULL) {return 0;} // check if memory allocation
    is failed
40     int offset = 48; // 48th letter in ASCII (0)
41     for (int i = 0; i < 10; i++) {
42         output[i] = offset + i;
43     }
44     return output;
45 }
46 }
47
48 // getSpecialChars - get the special characters
49 // void : takes nothing
50 // return (char*) : the string with all special characters in
    standard ASCII
51 char* PassGen::getSpecialChars() {
52     char* output = new char[42]; // 42 symbols
53     if (output == NULL) {return 0;} // check if memory allocation
    is failed
54     int offset = 33; // 33rd letter in ASCII (!)
55     int listOffset = 0;
56     int i;
57     // ASCII range of 33 - 64 (32 symbols)
58     for (i = 0; i < 32; i++) {
59         output[i] = offset + i;
60     }
61     listOffset = 32;
62     offset = 91;
63     // ASCII range of 91 - 96 (6 symbols)
64     for (i = 0; i < 6; i++) {
65         output[i + listOffset] = offset + i;
66     }

```

```

67     listOffset = 38;
68     offset = 123;
69     // ASCII range of 123 - 126 (4 symbols)
70     for (i = 0; i < 4; i++) {
71         output[i + listOffset] = offset + i;
72     }
73     return output;
74 }
75
76 // passGen - Password Generator
77 // charList (char*) : list of char to be used in password
78 // generation
79 // len (int) : length of password
80 // return (char*) : the generated password
81 char* PassGen::passGen(char *charList, const int len) {
82     std::srand(time(nullptr));
83     unsigned int index;
84     char* output = new char[len+1]; // length of password + 1
85     terminating char
86     if (output == NULL) {return 0;} // return 0 on the failiure of
87     memory allocation
88     for (int i = 0; i <= len; i++) {
89         if (i == len) {output[i] = charList[strSize(charList)];}
90         else {
91             index = std::rand()%(strSize(charList));
92             output[i] = charList[index];
93         }
94     }
95     return output;
96 }

```

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

```