```cpp
#include "Menu.h"//importing header file

// defining the constructor
Menu::Menu() {}

// main menu member function
void Menu::mainMenu()
{
        Menu::clear_screen(); //clears screen before menu call
        cout << endl;
        cout << "***********************************************" << endl;
        cout << "* Welcome to the Encryption/Decryption Machine *" << endl;
        cout << "*                                             *" << endl;
        cout << "* Please select an option from the menu below  *" << endl;
        cout << "***********************************************" << endl;
        cout << endl;

        // Prompt user for main menu selection
        cout << "1) Encrypt text" << endl;
        cout << "2) Decrypt text" << endl;
        cout << "3) Exit" << endl;
        cout << "-> ";
        cin >> mainChoice;
        if(!cin) {        // if input does not match data type
                cin.clear(); // clears error flags
                cin.ignore(999,'\n'); // ignore all input up until new line
        }
        switch (mainChoice) {
                case 1:
                        Menu::encryptMenu(); // call encryption main menu
                        break;
                case 2:
                        Menu::decryptMenu(); // call decryption main menu
                        break;
                case 3:                 // exit program
                        return;
                default:                // handles out of range requests
                        cout << endl;
                        cout << "Invalid Choice" <<
                                        "\nPlease Choose Again" << flush << endl;
                        Menu::sleep();  // call sleep function to review error prior to mainmenu call (clearing screen
)
                        Menu::mainMenu();  // return to main menu
        }
}
void Menu::encryptMenu()  // main encryption menu
{
        Menu::clear_screen();  // call clearscreen
        cout << endl;
        cout << "*************************************************************" << endl;
        cout << "*                      Encryption                           *" << endl;
        cout << "*                                                           *" << endl;
        cout << "* Please choose an encryption method from the menu below *" << endl;
        cout << "*************************************************************" << endl;
        cout << endl;

        // prompt user for encryption selection
        cout << "1) Substitution Cipher" << endl;
        cout << "2) Transposition Cipher" << endl;
        cout << "3) Return to Main Menu" << endl;
        cout << "-> ";
        cin >> encryptChoice;
        if(!cin) {                      // check for invalid data types
                cin.clear();                            // clears error flags
                cin.ignore(999,'\n');           // ignore input buffer to newline
        }
        switch (encryptChoice) {
                case 1:
                        Menu::enSubMenu();      // call substitution encryption submenu
                        break;
                case 2:
                        Menu::enCaeMenu();      // call transposition encryption submenu
                        break;
                case 3:
                        Menu::mainMenu();       // return to main menu
                        break;
                default:                                // handle all input out of range
                        cout << endl;
                        cout << "Invalid Choice" <<
                                        "\nPlease Choose Again" << endl;
                        Menu::sleep();                          // call sleep function to delay return to encrypt menu

                        Menu::encryptMenu();
                        break;
        }
}
```

```
// Main decryption menu
void Menu::decryptMenu()
{
        Menu::clear_screen();                   // call clear screen
        cout << endl;
        cout << "*********************************************************" << endl;
        cout << "*                      Decryption                       *" << endl;
        cout << "*                                                       *" << endl;
        cout << "* Please choose a decryption method from the menu below  *" << endl;
        cout << "*********************************************************" << endl;
        cout << endl;

        // prompt user for menu selection
        cout << "1) Substitution Cipher" << endl;
        cout << "2) Transposition Cipher" << endl;
        cout << "3) I don't know which decryption method was used" << endl;
        cout << "4) Return to Main Menu" << endl;
        cout << "-> ";
        cin >> decryptChoice;
        if(!cin) {                                      // check for invalid data types
                cin.clear();                            // clears error flags
                cin.ignore(999,'\n');           // ignore input buffer to newline
        }
        switch (decryptChoice) {
                case 1:
                        Menu::deSubMenu();              // call substitution decryption submenu
                        break;
                case 2:
                        Menu::deCaeMenu();              // call transposition decryption submenu
                        break;
                case 3:
                        Menu::bfMenu();         // call bruteforce decryption submenu
                        break;
                case 4:
                        Menu::mainMenu();               // return to main menu
                        break;
                default:                                // handle all input out of range
                        cout << endl;
                        cout << "Invalid Choice" <<
                                        "\nPlease Choose Again" << endl;
                        Menu::sleep();                  // call sleep function to delay return to decrypt menu
                        Menu::decryptMenu();
                        break;
        }
}

// Substitution encryption function
string Menu::subEncrypt(string cleartext) {

        string ciphertext;

        for (int i{ 0 }; i <= cleartext.length(); i++) {        // for char in cleartext
                for (int j{ 0 }; j < 26; j++) {                 // for char in alpha
                        if (cleartext[i] == ' ') {                      // check for whitespace character
                                ciphertext += ' ';
                                break;
                        }
                        if (toupper(cleartext[i]) == toupper(alpha[j])) {       // translate cleartext char to upper &
 match w/ upper char
                                if (isupper(cleartext[i])) {                                    // if cleartext char i
s uppercase
                                        ciphertext += toupper(subalpha[j]);                     // add respective uppe
rcase substitution char to cipher
                                        break;
                                }
                                else {                                                          // if char is lowercas
e
                                        ciphertext += subalpha[j];              // add respective lowercase substituti
on char to cipher
                                        break;
                                }
                        }
                }
        }
        return ciphertext;                      // return ciphertext to calling function
}
// Transposition encryption function
string Menu::caesarEncrypt(string cleartext, int shift) {

        string ciphertext;

        for (int i{ 0 }; i <= cleartext.length(); i++) {        // for char in cleartext
                for (int j{ 0 }; j < 26; j++) {                 // for char in ciphertext
                        if (cleartext[i] == ' ') {                      // if cleartext char is whitespace
                                ciphertext += ' ';                              // add whitespace to ciphertext
                                break;
```

```
166:                         }
167:                         if (toupper(cleartext[i]) == toupper(alpha[j])) {       // translate cleartext char to upper &
 match w/ upper alpha char
168:                                 if (isupper(cleartext[i])) {            // if cleartext char is upper
169:                                         caesarindex = (j + shift) % 26;         // shift index
170:                                         ciphertext += toupper(alpha[caesarindex]);      // add shifted uppercase char
to cipher text
171:                                         break;
172:                                 }
173:                                 else {                                                          // if cleartext char i
s lower
174:                                         caesarindex = (j + shift) % 26;         // shift index
175:                                         ciphertext += alpha[caesarindex];       // add shifted lowercase char to ciphe
r text
176:                                         break;
177:                                 }
178:                         }
179:                 }
180:         }
181:         return ciphertext;              // return ciphertext to calling function
182: }
183: // Substitution encryption menu
184: void Menu::enSubMenu()
185: {
186:         Menu::clear_screen();   // call clear screen
187:         cout << endl;
188:         cout << "************************************" << endl;
189:         cout << "*            Encryption            *" << endl;
190:         cout << "*                                  *" << endl;
191:         cout << "*        Substitution Cipher       *" << endl;
192:         cout << "* Please select from the menu below *" << endl;
193:         cout << "************************************" << endl;
194:         cout << endl;
195:
196:         // prompt user for menu selection
197:         cout << "1) Enter the text to be encrypted" << endl;
198:         cout << "2) Return to the Encryption Menu" << endl;
199:         cout << "-> ";
200:         cin >> enSubChoice;
201:         if(!cin) {                              // if input does not match data type
202:                 cin.clear();            // clears error flags
203:                 cin.ignore(999,'\n');           // ignore input buffer to newline
204:         }
205:         switch(enSubChoice) {
206:                 case 1: {
207:                         cout << "\nEnter Clear Text" << endl;           // prompt user for cleartext

208:                         cin.ignore();                   // clear input buffer
209:                         getline(cin, cleartext);        // store input
210:                         // return substitution encrypted text
211:                         cout << "Encrypted message: " << Menu::subEncrypt(cleartext) << endl;
212:                         cout << endl;
213:                         Menu::superMenu();              // call main submenu
214:                         break;
215:                 }
216:                 case 2:
217:                         Menu::encryptMenu();    // return to encryption menu
218:                         break;
219:                 default:
220:                         cout << endl;           // check for input out of range
221:                         cout << "Invalid Choice " <<
222:                                 "\nPlease Choose Again" << endl;
223:                         Menu::sleep();          // add error delay prior to calling top of function
224:                         Menu::enSubMenu();              // call top of function
225:                         break;
226:         }
227: }
228: // Transposition encryption submenu
229: void Menu::enCaeMenu()
230: {
231:         Menu::clear_screen();   // clear screen
232:         cout << endl;
233:         cout << "************************************" << endl;
234:         cout << "*            Encryption            *" << endl;
235:         cout << "*                                  *" << endl;
236:         cout << "*       Transposition Cipher       *" << endl;
237:         cout << "* Please select from the menu below *" << endl;
238:         cout << "************************************" << endl;
239:         cout << endl;
240:
241:         // prompt user for menu selection
242:         cout << "1) Enter the text to be encrypted" << endl;
243:         cout << "2) Return to the Encryption Menu" << endl;
244:         cout << "-> ";
245:         cin >> enCaeChoice;
246:         if(!cin) {                              // if input does not match data type
```

```
  247:                 cin.clear();            // clears error flags
  248:                 cin.ignore(999,'\n');   // ignore input buffer to newline
  249:         }
  250:         switch(enCaeChoice) {
  251:                 case 1: {
  252:                         cout << "\nEnter Clear Text" << endl;   // prompt user for cleartext
  253:                         cin.ignore();                           // clear input buffer
  254:                         getline(cin, cleartext);        // save cleartext input
  255:                         // return caesar encrypted cleartext
  256:                         cout << "Encrypted message: " << Menu::caesarEncrypt(cleartext) << endl;
  257:                         cout << endl;
  258:                         Menu::superMenu();              // call main submenu
  259:                         break;
  260:                 }
  261:                 case 2:
  262:                         Menu::encryptMenu();    // return to top of encryption menu
  263:                         break;
  264:                 default:                        // if input is out of range
  265:                         cout << endl;
  266:                         cout << "Invalid Choice " <<
  267:                                         "\nPlease Choose Again" << endl;
  268:                         Menu::sleep();                  // delay for error message return to top of menu
  269:                         Menu::enCaeMenu();              // call top of menu
  270:                         break;
  271:         }
  272: }
  273: // Substitution decryption function
  274: string Menu::subDecrypt(string cleartext) {
  275:
  276:         string ciphertext;
  277:
  278:         for (int i{ 0 }; i < cleartext.length(); i++) {   // for char in ciphertext
  279:                 for (int j{ 0 }; j < 26; j++) {         // for char in subalpha
  280:                         if (cleartext[i] == ' ') {              // if char is whitespace
  281:                                 ciphertext += ' ';                      // add whitespace to 'cleartext'
  282:                                 break;
  283:                         }
  284:                         // if upper 'ciphertex' char matches upper subalpha char
  285:                         if (toupper(cleartext[i]) == toupper(subalpha[j])) {
  286:                                 if (isupper(cleartext[i])) {            // check if ciphertext char is upper
  287:                                         ciphertext += toupper(alpha[j]);        // add respective upper alpha char to
'cleartext'
  288:                                         break;
  289:                                 }
  290:                                 else {                                  // if ciphertext char is lower
  291:                                         ciphertext += alpha[j];         // add respective lower alpha char to 'clearte
xt'
  292:                                         break;
  293:                                 }
  294:                         }
  295:                 }
  296:         }
  297:         return ciphertext;                      // return cleartext to calling function
  298: }
  299: // Transposition decryption function
  300: string Menu::caesarDecrypt(string cleartext, int shift) {
  301:
  302:         string ciphertext;
  303:
  304:         for (int i{ 0 }; i < cleartext.length(); i++) {         // for char in ciphertext
  305:                 for (int j{ 0 }; j < 26; j++) {                 // for char in alpha
  306:                         if (cleartext[i] == ' ') {                      // if ciphertext char is whitespace
  307:                                 ciphertext += ' ';                              // add whitespace to ciphertext
  308:                                 break;
  309:                         }
  310:                         // if upper cipher matches upper alpha
  311:                         if (toupper(cleartext[i])== toupper(alpha[j])) {
  312:                                 if (isupper(cleartext[i])) {                            // if cipher char is upper
  313:                                         caesarindex = (26 + j - shift) % 26;    // subtract index shift
  314:                                         // add respective shifted upper char to 'cleartext'
  315:                                         ciphertext += toupper(alpha[caesarindex]);
  316:                                         break;
  317:                                 }
  318:                                 else {                                                                  // if
cipher char is lower
  319:                                         caesarindex = (26 + j - shift) % 26;    // subtract index shift
  320:                                         ciphertext += alpha[caesarindex];               // add respective shifted lowe
r char to 'cleartext'
  321:                                         break;
  322:                                 }
  323:                         }
  324:                 }
  325:         }
  326:         return ciphertext;                              // return 'cleartext' to calling function
  327: }
  328: // Brute force function - returns all 26 shifted values and substitution decrypt attempt
```

```
329: void Menu::bruteforce(string ciphertext) {
330:
331:         cout << endl;
332:         cout << left << setw(35) << "Caesar Brute Force" << "Substitution Cipher" << endl;
333:         for (int shift{ 1 }; shift <= 26; shift++) {    // for each of 26 shifts
334:                 if (shift == 1) {                                                       // if first iteration
335:                         // include first transposition offset
336:                         cout << left << setw(35) << Menu::caesarEncrypt(ciphertext, shift);
337:                         // as well as substitution decryption attempt
338:                         cout << Menu::subDecrypt(ciphertext) << endl;
339:                         continue;
340:                 }
341:                 // return remaining transpotition offsets
342:                 cout << left << setw(35) << Menu::caesarEncrypt(ciphertext, shift) << endl;
343:         }
344: }
345: // Substitution decrypt submenu
346: void Menu::deSubMenu()
347: {
348:         Menu::clear_screen();                   // clear screen
349:         cout << endl;
350:         cout << "************************************" << endl;
351:         cout << "*            Decryption            *" << endl;
352:         cout << "*                                  *" << endl;
353:         cout << "*        Substitution Cipher       *" << endl;
354:         cout << "* Please select from the menu below *" << endl;
355:         cout << "************************************" << endl;
356:         cout << endl;
357:
358:         // prompt user for menu selection
359:         cout << "1) Enter the text to be decrypted" << endl;
360:         cout << "2) Return to the Decryption Menu" << endl;
361:         cout << "-> ";
362:         cin >> deSubChoice;
363:         if(!cin) {                                              // if input invalid data type
364:                 cin.clear();                                    // clears error flags
365:                 cin.ignore(999,'\n');           // ignore input buffer to newline
366:         }
367:         switch(deSubChoice) {
368:                 case 1: {
369:                         cout << "\nEnter Clear Text" << endl;  // prompt user for ciphertext
370:                         cin.ignore();                                                   // clear input buffer
371:                         getline(cin, cleartext);                                // save ciphertext
372:                         // return substitution decrypted cipher text
373:                         cout << "Decrypted message: " << Menu::subDecrypt(cleartext) << endl;
374:                         cout << endl;
375:                         Menu::superMenu();                      // call main submenu
376:                         break;
377:                 }
378:                 case 2:
379:                         Menu::decryptMenu();            // return to decrypt menu
380:                         break;
381:                 default:                                                        // if input is out of range
382:                         cout << endl;
383:                         cout << "Invalid Choice" <<
384:                                         "\nPlease Choose Again" << endl;
385:                         Menu::sleep();                          // delay return to top for error message
386:                         Menu::deSubMenu();                      // return to top of function
387:                         break;
388:         }
389: }
390: // Transposition decryption menu
391: void Menu::deCaeMenu()
392: {
393:         Menu::clear_screen();                   // clear screen
394:         int choice7{0};
395:         cout << endl;
396:         cout << "************************************" << endl;
397:         cout << "*            Decryption            *" << endl;
398:         cout << "*                                  *" << endl;
399:         cout << "*       Transposition Cipher       *" << endl;
400:         cout << "* Please select from the menu below *" << endl;
401:         cout << "************************************" << endl;
402:         cout << endl;
403:
404:         // prompt user for menu selection
405:         cout << "1) Enter the text to be decrypted" << endl;
406:         cout << "2) Return to the decryption Menu" << endl;
407:         cout << "-> ";
408:         cin >> deCaeChoice;
409:         if(!cin) {                                      // if input does not match data type
410:                 cin.clear();                    // clears error flags
411:                 cin.ignore(999,'\n');   // ignore input buffer to newline
412:         }
413:         switch(deCaeChoice)     {
414:                 case 1: {
```

```
415:                         cout << "\nEnter Clear Text" << endl; // prompt user for ciphertext
416:                         cin.ignore();                                           // clear input buffer
417:                         getline(cin, cleartext);                          // save ciphertext
418:                         // return decrypted transposition ciphertext
419:                         cout << "Decrypted message: " << Menu::caesarDecrypt(cleartext) << endl;
420:                         cout << endl;
421:                         Menu::superMenu();                                  // call main submenu
422:                         break;
423:                 }
424:                 case 2:
425:                         Menu::decryptMenu();                    // return to decyrption menu
426:                         break;
427:                 default:                                                        // handle input out of range
428:                         cout << endl;
429:                         cout << "Invalid Choice" <<
430:                                         "\nPlease Choose Again" << endl;
431:                         Menu::sleep();                                  // delay return to top for error message
432:                         Menu::deCaeMenu();                              // return to top of function
433:                         break;
434:         }
435: }
436: // Brute-force submenu
437: void Menu::bfMenu()
438: {
439:         Menu::clear_screen();                                   // clear screen
440:         cout << endl;
441:         cout << "************************************" << endl;
442:         cout << "*            Decryption            *" << endl;
443:         cout << "*                                  *" << endl;
444:         cout << "*          Unknown Cipher          *" << endl;
445:         cout << "* Please select from the menu below *" << endl;
446:         cout << "************************************" << endl;
447:         cout << endl;
448:
449:         // prompt user for menu selection
450:         cout << "1) Attempt to decrypt using a bruteforce method" << endl;
451:         cout << "2) Return to the decryption Menu" << endl;
452:         cout << "-> ";
453:         cin >> bfChoice;
454:         if(!cin) {                                      // if input does not match data type
455:                 cin.clear();                    // clears error flags
456:                 cin.ignore(999,'\n');   // ignore input buffer to newline
457:         }
458:         switch(bfChoice)        {
459:                 // choice 1
460:                 case 1: {
461:                         cout << "\nEnter Clear Text" << endl;   // prompt user for unknown cipher text
462:                         cin.ignore();                                                   // clear input buffer
463:                         getline(cin, cleartext);                // save ciphertext
464:                         Menu::bruteforce(cleartext);    // return transposition array / sub decrypt attempt
465:                         cout << endl;
466:                         Menu::superMenu();                      // return to main submenu
467:                         break;
468:                 }
469:                 // choice 2
470:                 case 2:
471:                         Menu::decryptMenu();            // return to decryption menu
472:                         break;
473:                 default:                                                        // handle input out of range
474:                         cout << endl;
475:                         cout << "Invalid Choice" <<
476:                                         "\nPlease Choose Agagin" << endl;
477:                         Menu::sleep();                  // delay for error message
478:                         Menu::bfMenu();                 // return to top of function
479:                         break;
480:         }
481: }
482: // Program main submenu
483: void Menu::superMenu()
484: {
485:         cout << endl;
486:         cout << "***************************" << endl;
487:         cout << "*   Please choose where   *" << endl;
488:         cout << "* you'd like to return to *" << endl;
489:         cout << "*   from the menu below   *" << endl;
490:         cout << "***************************" << endl;
491:         cout << endl;
492:
493:         // prompt user for menu selection
494:         cout << "1) Encryption Menu" << endl;
495:         cout << "2) Decryption Menu" << endl;
496:         cout << "3) Main Menu" << endl;
497:         cout << "-> ";
498:         cin >> superChoice;
499:         if(!cin) {                                              // if input does not match data type
500:                 cin.clear();                            // clears error flags
```

```
501:                 cin.ignore(999,'\n');           // ignore input buffer to newline
502:         }
503:         switch (superChoice) {
504:         // choice 1
505:         case 1:
506:                 Menu::encryptMenu();            // return to encryption menu
507:                 break;
508:         // choice 2
509:         case 2:
510:                 Menu::decryptMenu();            // return to decryption menu
511:                 break;
512:         // choice 3
513:         case 3:
514:                 Menu::mainMenu();                       // return to program main menu
515:                 break;
516:         default:                                                // handling input out of range
517:                 cout << endl;
518:                 cout << "Invalid Choice" <<
519:                                 "\nPlease Choose Again" << endl;
520:                 Menu::sleep();                          // delay for error message
521:                 Menu::superMenu();                      // return to top of menu
522:                 break;
523:         }
524: }
525:
526: // clear screen function
527: void Menu::clear_screen() {
528:         #ifdef WINDOWS                                  // if architecture matches windows
529:             std::system("cls");                 // return system 'cls'
530:         #elif _WIN32                                    // if 32-bit win architecture
531:                 std::system("cls");                     // return system 'cls'
532:         #elif _WIN64                                    // if 64-bit win architecture
533:                 std::system("cls");                     // return system 'cls'
534:         #else                                                   // assume POSIX
535:             std::system("clear");               // return system 'clear'
536:         #endif
537: }
538:
539: // menu sleep (delay) function
540: void Menu::sleep() {
541:     using namespace std::this_thread;           // sleep_for, sleep_until
542:     using namespace std::chrono_literals;       // ns, us, ms, s, h, etc.
543:     using std::chrono::system_clock;
544:
545:     sleep_for(1s);                                                      // call sleep for 1 second
546: }
547:
548: // end implementation file
```