CODE FOR PROBLEM 14 AND 15

Problem 14

```
#include <iostream>
#include <limits>
#include <cmath>
using namespace std;

class expFunction {
private:
        double my_power(double x, int n);
public:
        expFunction(){};
        ~expFunction(){};

        double my_exp(double x);
};

double expFunction::my_power(double x, int n) {
        if (n == 0) {
                return 1.0;
        }

        if (n%2 == 0) {
                return my_power(x, n/2) * my_power(x, n/2);
        } else {
                return x * my_power(x, n/2) * my_power(x, n/2);
        }
}

double expFunction::my_exp(double x)
{
        if (x < 0) {
                return 1.0/my_exp(-x);
        }

        // Round up x when x is large so that
        // e^x = 1 + x + ... + x^n/n! + O(x^n) converges faster.
        int roundup = ceil(x);
        double x_modified = x/roundup;
        double result = 1.0;
        double TaylorExpansionTerm = x_modified;
        int n = 1;
        while (TaylorExpansionTerm > numeric_limits<double>::min()) {
                result += TaylorExpansionTerm;
                TaylorExpansionTerm *= (x_modified/++n);
        }

        return my_power(result, roundup);
}


int main(int argc, char const *argv[]) {
        expFunction soln;

        double power;
        cout << "Input the power: " << endl;
        cin >> power;

        cout << "e^" << power << " = " << soln.my_exp(power) << endl;

        return 0;
}
```

Problem 15

```
#include <iostream>
#include <string>
#include <vector>
#include <ctime>
using namespace std;

class strMatch {
private:
        vector<int> next;
        void GetNext(const string& str);
public:
        strMatch(){};
        ~strMatch(){};

        bool strStr(const string& haystack, const string& needle);
        bool strStrKMP(const string& haystack, const string& needle);
};

// Brute force: time O(m*n), space O(1)
bool strMatch::strStr(const string& haystack, const string& needle) {
        if (needle.empty()) {
                return true;
        }

        for (int i = 0; i < haystack.size(); i++) {
                if (haystack[i] == needle[0]) {
                        bool match = true;
                        for (int j = 0; j < needle.size(); j++) {
                                if (haystack[i+j] != needle[j]) {
                                        match = false;
                                        break;
                                }
                        }

                        if (match) {
                                return true;
                        }
                }
        }

        return false;
}
```

```
// KMP: time O(m + n), space O(n)
void strMatch::GetNext(const string& str) {
        next.push_back(-1);
        int i = -1;
        int j = 0;

        while (j < str.size() - 1) {
                //str[i] - prefix str[j] - suffix
                if (i == -1 || str[j] == str[i]) {
                        i++;
                        j++;

                        if (str[j] != str[i]) {
                                next.push_back(i);
                        } else {
                                next.push_back(next[i]);
                        }

                } else {
                        i = next[i];
                }
        }

        return;
}

bool strMatch::strStrKMP(const string& haystack, const string& needle) {
        GetNext(needle);

        int i, j;
        int haystackLen = haystack.size();
        int needleLen = needle.size();

        for (i = 0, j = 0; i < haystackLen && j < needleLen; ) {
                // currently, match!
                if (j == -1 || haystack[i] == needle[j]) {
                        i++;
                        j++;
                } else {
                        // currently, NOT match..
                        j = next[j];
                }
        }

        if (j == needle.size()) {
                return true;
        } else {
                return false;
        }
}

int main(int argc, char const *argv[]) {
        strMatch soln;

        string haystack;
        cout << "Input haystack: ";
        getline(cin, haystack);

        string needle;
        cout << "Input needle: ";
        getline(cin, needle);

        clock_t now = clock();
        cout << "Brute force: " << soln.strStr(haystack, needle) << endl;
        clock_t after = clock();
        cout << "Brute force run-time: " << (after - now) /
        (double)(CLOCKS_PER_SEC / 1000) << " ms" << endl;

        now = clock();
        cout << "KMP: " << soln.strStrKMP(haystack, needle) << endl;
        after = clock();
        cout << "KMP run-time: " << (after - now) /
        (double)(CLOCKS_PER_SEC / 1000) << " ms" << endl;

        return 0;
}
```