#include "Roman.h"

#include <boost/regex.hpp>

#include <boost/algorithm/string.hpp>

#include <boost/tokenizer.hpp>

#include <boost/lexical\_cast.hpp>

#include <fstream>

#include <vector>

#include <string>

using namespace std;

using namespace boost::algorithm;

void Roman::intergalaxy\_transaction(string& request)

{

//print the input

cout<<"INPUT: "<<request<<endl;

//Read the file and get the list of input string

vector<string> input\_list = get\_input\_list();

//This credit list will store the key/value of Gold/Silver/Iron credit values

map<string,float> credit\_list = get\_credit\_list(input\_list);

//Generate output

string output\_str = get\_output(request,credit\_list);

//print result

cout<<"OUTPUT: "<<output\_str<<endl;

cout<<"============================================="<<endl;

}

map<string,float> Roman::get\_credit\_list(vector<string> &input\_list)

{

//This base list will store key/value of glob=I,prok=V,pish=X and tegj=L

//map<string,string> base\_list;

//Final list

map<string,float> credit\_list;

//Apply rules

static const boost::regex rule1("^(glob|prok|pish|tegj){1}.\* is{1} (I|V|X|L){1}$");

static const boost::regex rule2("^(glob|prok|pish|tegj){1}.\* (glob|prok|pish|tegj){1}.\* (Silver|Gold|Iron){1} .\* Credits$");

//Build dict

for(int index=0;index<input\_list.size();index++)

{

string input\_str =input\_list.at(index);

if ( true == regex\_match(input\_str,rule1))

{

string key= input\_str.substr(0,4);

string value = input\_str.substr(input\_str.size()-1,input\_str.size());

base\_list[key] = value;

}

else if(true == regex\_match(input\_str, rule2))

{

float credit\_value = 0.0;

if(input\_str.find("Gold") != -1)

{

credit\_value = get\_credit\_value(base\_list,input\_str);

credit\_list["Gold"] = credit\_value;

}

else if(input\_str.find("Silver") != -1)

{

credit\_value = get\_credit\_value(base\_list,input\_str);

credit\_list["Silver"] = credit\_value;

}

else if(input\_str.find("Iron") != -1)

{

credit\_value = get\_credit\_value(base\_list,input\_str);

credit\_list["Iron"] = credit\_value;

}

else

{

cout<<"Gold/Silver/Iron only currently allowed"<<endl;

}

}

else

{

cout<<"String Not Matched For Input, Please refer the format \""<<input\_str<<"\""<<endl;

}

}

return credit\_list;

}

string Roman::get\_output(string &request,map<string,float> &credit\_list)

{

//Output Format Rules

static const boost::regex rule1("^((how much is)?) ((pish|tegj|glob|prok|\\?)\\s?){0,}$");

static const boost::regex rule2("^((how many Credits is)?) ((pish|tegj|glob|prok)\\s?){0,}((Silver|Gold|Iron)\\s?){1}\\?$");

string final\_output;

if ( true == regex\_match(request,rule1))

{

string output\_str = get\_result(request);

final\_output = output\_str + "is " + boost::lexical\_cast<std::string>(process\_tokens(output\_str));

}

else if(true == regex\_match(request,rule2))

{

string output\_str = get\_result(request);

float credit\_val = credit\_list[find\_credit\_name(output\_str)] \* process\_tokens(output\_str);

final\_output = output\_str + " is " + boost::lexical\_cast<std::string>(credit\_val) + " Credits";

}

else

{

final\_output = "I have no idea what you are talking about";

}

return final\_output;

}

string Roman::find\_credit\_name(string &credit\_name)

{

trim(credit\_name);

int found = credit\_name.find\_last\_of(" ");

return credit\_name.substr(found+1);

}

int Roman::process\_tokens(string &string\_tokens)

{

typedef boost::tokenizer<boost::char\_separator<char> > tokenizer;

boost::char\_separator<char> sep(" ");

tokenizer tok(string\_tokens,sep);

string temp\_str;

for(tokenizer::iterator beg=tok.begin(); beg!=tok.end();++beg)

{

string t = string(\*beg);

temp\_str += base\_list[t];

}

return convert\_roman\_to\_number(temp\_str);

}

string Roman::get\_result(string &request)

{

int start = request.find(" is ");

int end = request.find("?");

string output\_str = request.substr(start+3,(end-(start+3)));

return output\_str;

}

float Roman::get\_credit\_value(map<string,string> &base\_list,string &input\_str)

{

string roman\_value = base\_list[input\_str.substr(0,4)] + base\_list[input\_str.substr(5,4)];

int decimal\_value = convert\_roman\_to\_number(roman\_value);

int start = input\_str.find(" is ");

int end = input\_str.find(" Credits");

string credit\_value\_str = input\_str.substr(start+3,(end-(start+3)));

return (atof(credit\_value\_str.c\_str())/decimal\_value);

}

int Roman::convert\_roman\_to\_number(string& input\_str)

{

//Init the map

map<char,int> maplist = init\_map();

//Trim the input;

trim(input\_str);

//Check the given Rules

apply\_rules(input\_str);

//Convert Roman to number

return romanToNumber(input\_str,maplist);

}

map<char,int> Roman::init\_map()

{

map<char,int> mlist;

mlist['I'] =1;

mlist['V'] =5;

mlist['X'] =10;

mlist['L'] =50;

mlist['C'] =100;

mlist['D'] =500;

mlist['M'] =1000;

return mlist;

}

bool Roman::apply\_rules(const string& str)

{

//Rule1: "I", "X", "C", and "M" 3 times allowed

//Rule2: "D", "L", and "V" shouldnt be repeated

static const boost::regex e("^M{0,3}(CM|CD|D?C{0,3})(XC|XL|L?X{0,3})(IX|IV|V?I{0,3})$");

if ( false == regex\_match(str, e))

{

throw "Invalid Roman Input";

}

}

int Roman::romanToNumber(const string& input\_str,map<char,int> &maplist)

{

int output\_num = 0;

for(int i=0,j=1;j<=input\_str.length();i++,j++)

{

if(maplist[input\_str[i]] >= maplist[input\_str[j]])

{

output\_num += maplist[input\_str[i]];

}

else if(maplist[input\_str[i]]<maplist[input\_str[j]])

{

output\_num += maplist[input\_str[j]] - maplist[input\_str[i]];

i++;j++;

}

}

return output\_num;

}

vector<string> Roman::get\_input\_list()

{

vector<string> list;

ifstream file("InputFile.txt");

string line;

while(getline(file, line))

{

list.push\_back(line);

}

file.close();

return list;

}