**“Base64.h”**

**(Because of different format code may appear different)**

#ifndef BASE64\_H

#define BASE64\_H

#include <vector>

#include <string>

// our encryption will use the base64 algorithm

namespace Base64

{

std::string base64\_encode(const std::string &);

// 3 lines to scramble base64 encoding

const std::string &SALT1 = "LM::TB::BB";

const std::string &SALT2 = "\_:/\_77";

const std::string &SALT3 = "line=boostedC++";

// triple encode with base64 with slight modifications (arbitrary)

std::string EncryptB64(std::string s)

{

s = SALT1 + s + SALT2 + SALT3;

s = base64\_encode(s);

s.insert(7, SALT3);

s += SALT1;

s = base64\_encode(s);

s = SALT2 + SALT3 + s + SALT1;

s = base64\_encode(s);

s.insert(1, "Ls");

s.insert(7, "A");

return s;

}

const std::string &BASE64\_CODES = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";

std::string base64\_encode(const std::string &s)

{

std::string ret;

int val = 0; // index we use map to table

int bits = -6; // used to represent number of bits in a subgroup (of six)

const unsigned int b63 = 0x3F; // decimal value is 63

for (const auto &c : s) // auto when type is unknown, c is assign every character from s string

{

val = (val << 8) + c; // left binary shift by 8, same as val \* 2^8 + c

bits += 8; // add 8 to numbers of bits when extracting info (octets)

while (bits >= 0)

{

ret.push\_back(BASE64\_CODES[(val >> bits) & b63]); // does binary right shift + binary AND comparison

bits -= 6;

}

}

if (bits > -6) // means at least one character has been insert

ret.push\_back(BASE64\_CODES[((val << 8) >> (bits + 8)) & b63]);

while (ret.size() % 4)

ret.push\_back('=');

return ret;

}

}

#endif // BASE64\_H