```
int create_socket(int port)
{
    SOCKET s = 0;
    //struct sockaddr_in addr;

    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
    {
        printf("WSAStartup()fail:%d\n", GetLastError());
        return -1;
    }

    /*addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    addr.sin_addr.s_addr = htonl(INADDR_ANY);*/

    s = socket(AF_INET, SOCK_STREAM, 0);
    if (s < 0) {
        perror("Unable to create socket");
        exit(EXIT_FAILURE);
    }

    return s;
}

SSL_CTX* create_context()
{
    const SSL_METHOD* method;
    SSL_CTX* ctx;

    method = TLS_client_method();

    ctx = SSL_CTX_new(method);
    if (!ctx) {
        perror("Unable to create SSL context");
        ERR_print_errors_fp(stderr);
        exit(EXIT_FAILURE);
    }

    return ctx;
}

void configure_context(SSL_CTX* ctx)
{
    /* Set the key and cert */
    if (SSL_CTX_use_certificate_file(ctx, "cert_test.pem", SSL_FILETYPE_PEM) <= 0) {
        ERR_print_errors_fp(stderr);
        exit(EXIT_FAILURE);
    }

    if (SSL_CTX_use_PrivateKey_file(ctx, "key_test.pem", SSL_FILETYPE_PEM) <= 0) {
        ERR_print_errors_fp(stderr);
        exit(EXIT_FAILURE);
    }
}

void createCertificate()
{
    EVP_PKEY* pkey;
    pkey = EVP_PKEY_new();

    RSA* rsa;
    rsa = RSA_generate_key(
        2048,   /* number of bits for the key - 2048 is a sensible value */
        RSA_F4, /* exponent - RSA_F4 is defined as 0x10001L */
```

```
    NULL,   /* callback - can be NULL if we aren't displaying progress */
    NULL    /* callback argument - not needed in this case */
);

EVP_PKEY_assign_RSA(pkey, rsa);

X509* x509;
x509 = X509_new();

ASN1_INTEGER_set(X509_get_serialNumber(x509), 1);

X509_gmtime_adj(X509_get_notBefore(x509), 0);
X509_gmtime_adj(X509_get_notAfter(x509), 31536000L);

X509_set_pubkey(x509, pkey);

auto name = X509_get_subject_name(x509);

int ret = X509_NAME_add_entry_by_txt(name, "C", MBSTRING_ASC,
    (unsigned char*)"CA", -1, -1, 0);
std::cout << ret << std::endl;
ret = X509_NAME_add_entry_by_txt(name, "O", MBSTRING_ASC,
    (unsigned char*)"MyCompany Inc.", -1, -1, 0);
std::cout << ret << std::endl;
ret = X509_NAME_add_entry_by_txt(name, "CN", MBSTRING_ASC,
    (unsigned char*)"localhost", -1, -1, 0);
std::cout << ret << std::endl;

ret = X509_set_issuer_name(x509, name);
std::cout << ret << std::endl;

ret = X509_sign(x509, pkey, EVP_sha1());
std::cout << ret << std::endl;

ret = X509_verify(x509, pkey);
std::cout << ret << std::endl;

/* BIO* f = BIO_new(BIO_s_mem());
 PEM_write_bio_X509(f, x509);
 size_t pri_len = BIO_pending(f);
 char* private_key_char = (char*)malloc(pri_len + 1);
 BIO_read(f, private_key_char, pri_len);
 private_key_char[pri_len] = '\0';*/

 //BIO* bio_file = NULL;

 //bio_file = BIO_new_file("AAAAAAA.pem", "w");
 //if (bio_file == NULL) {
 //    ret = -1;
 //}
 //ret = PEM_write_bio_X509(bio_file, x509);
 //if (ret != 1) {
 //    ret = -1;
 //}
 //BIO_free(bio_file);


BIO* w = NULL;
w = BIO_new_file("key_test.pem", "wb");
PEM_write_bio_PrivateKey(
    w,                  /* write the key to the file we've opened */
    pkey,               /* our key from earlier */
    NULL, /* default cipher for encrypting the key on disk */
    NULL,       /* passphrase required for decrypting the key on disk */
    0,                 /* length of the passphrase string */
```

```
        NULL,               /* callback for requesting a password */
        NULL                /* data to pass to the callback */
    );
    BIO_free(w);

    BIO* f = NULL;
    f = BIO_new_file("cert_test.pem", "wb");
    PEM_write_bio_X509(
        f,   /* write the certificate to the file we've opened */
        x509 /* our certificate */
    );
    BIO_free(f);
}

int main(int argc, char** argv)
{
    //createCertificate();
    int sock;
    SSL_CTX* ctx;

    ctx = create_context();

    configure_context(ctx);

    sock = create_socket(4433);

    struct sockaddr_in addr;
    addr.sin_family = AF_INET;
    addr.sin_port = htons(4433);
    addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    int ret = connect(sock, (struct sockaddr*)&addr, sizeof(addr));
    if (ret < 0) {
        perror("Unable to connect");
        exit(EXIT_FAILURE);
    }

    SSL* ssl;
    ssl = SSL_new(ctx);
    SSL_set_fd(ssl, sock);

    if (SSL_connect(ssl) <= 0) {
        ERR_print_errors_fp(stderr);
    }
    else {
        char buf[1024];
        ret = SSL_read(ssl, buf, strlen(buf));
        buf[ret] = '\0';

        std::cout << buf << std::endl;

        SSL_write(ssl, "test2", strlen("test2"));
    }
}
```