

Versuch i-12-a  
Programmierung von Chipkarten

Steve Moser, Markus Ganzenmüller

4. Dezember 2002

# Inhaltsverzeichnis

<b>1</b>	<b>Allgemein</b>	<b>2</b>
1.1	Einleitung . . . . .	3
1.2	Aufgabenstellung . . . . .	3
1.3	Typen von Chipkarten . . . . .	4
1.3.1	Normale Speicherkarten . . . . .	4
1.3.2	Intelligente Speicherkarten . . . . .	4
1.3.3	Prozessorchipkarten . . . . .	4
1.3.4	Superchipkarten . . . . .	5
<b>2</b>	<b>Genauere Erläuterung des Aufbaus und der Übertragung</b>	<b>6</b>
2.1	Magnetsteifenkarte . . . . .	7
2.2	Chipkarte . . . . .	7
2.3	Microprozessorkarte . . . . .	8
2.4	Kontaktlose Chipkarten . . . . .	9
2.5	Übertragungsarten . . . . .	9
2.5.1	I2C Bus . . . . .	10
2.6	Das eigene Programm . . . . .	10
2.6.1	Code . . . . .	10
2.7	Kurze Erläuterung zur Abnahme . . . . .	19
2.7.1	Probleme während der Programmerstellung . . . . .	19
2.7.2	Sicherung der Labore durch Chipkarten . . . . .	20

# **Kapitel 1**

## **Allgemein**

## 1.1 Einleitung

Chipkarten für die verschiedensten Verwendungszwecke gehören mittlerweile zu unserem alltäglichen Leben. Grundsätzlich besteht ihre Aufgabe darin, bestimmte Daten zu speichern, um diese zu einem bestimmten Zeitpunkt abrufen zu können. Beispiele für Anwendungsarten sind etwa Geldkarten, Krankenversicherungskarten, Zugangskarten oder Karten zur Zeiterfassung. Unser Praktikumsversuch beschäftigte sich unter anderem mit den Fragen nach dem Aufbau solcher Chipkarten und wie man diese programmiert, bzw. wie man Daten darauf speichern kann. Zu diesem Zweck sollte ein kleines Anwendungsprogramm geschrieben werden, welches es ermöglicht, die Informationen einer Chipkarte auszulesen, sie zu verändern und wieder zu speichern.

## 1.2 Aufgabenstellung

Thema: Format und Protokoll von externen Schnittstellen  
Rechner: G 209 Platz A4, HP4

Aufgabenstellung

=====

An der seriellen Schnittstelle ist ein Chip-Karten-Gert angeschlossen. Erstellen Sie ein Programm zum Beschreiben und Lesen von Chip-Karten.

Achtung: Fragen Sie vor einer Realisierung in Java nach, ob die notwendige Hardware bereits beschafft wurde.

## **1.3 Typen von Chipkarten**

Chipkarten unterscheiden sich nicht nur in der Art ihres Verwendungszweckes, sondern auch entscheidend durch inneren Aufbau. Im folgenden Abschnitt sollen die bekanntesten Typen genannt und in einer Einführung kurz beschrieben werden. Im nächsten Kapitel folgt eine Vertiefung in die Funktionsweise der Chipkarten.

### **1.3.1 Normale Speicherkarten**

Speicherkarten haben, wie schon ihr Name sagt, nur die Aufgabe, einfache Daten zu speichern. Auf Ihnen befindet sich nur ein kleiner Speicherchip in Form eines EEPROM/EPROM (Electrically Erasable Programmable Read Only Memory). Auf die gespeicherten Daten kann direkt und ohne spezielle Sicherheitsmaßnahmen zugegriffen werden, d.h. die Daten sind nicht besonders geschützt. Die Menge an Informationen, die gespeichert werden kann, bewegt sich meist um etwa 256 Byte. Ein Anwendungsbeispiel für diesen Kartentyp ist z.B. die Krankenversicherungskarte, welche mit einem handelsüblichen Chipkartenlese/schreib-Gerät ausgelesen und beschrieben werden kann. Während unseres Versuchs haben wir es aber tunlichst vermieden, unsere Daten zu verändern. Im Gespräch mit unserem Praktikumbetreuer Herrn Prof. Lutz kam unter anderem zur Sprache, daß eine andere Gruppe aus versehen die Daten einer solchen Karte verändert hatte, sie aber nach diversen Berechnungen glücklicherweise wiederherstellen konnte.

### **1.3.2 Intelligente Speicherkarten**

Bei intelligenten Speicherkarten kommen wie bei den normalen Speicherkarten EPROM, bzw. EEPROM Chips zum Einsatz. Jedoch sind die Daten diesmal durch eine festverdrahtete Sicherheitslogik vor dem Beschreiben gesichert und können nur durch Eingabe einer PIN-Nummer verändert werden. Auch der Lesevorgang kann wahlweise gesichert werden. Der Speicherplatz dieser Karten beträgt ca. 256 Byte bis zu einem Kilobyte.

### **1.3.3 Prozessorchipkarten**

In den letzten Jahren hat die Integrationsdichte bei bei Schaltkreisen und Prozessoren dramatisch zugenommen. Dadurch ist es mittlerweile möglich, komplette Kleinstrechner auf Chipkarten unterzubringen, welche durch direkten Kontakt oder induktiv durch das Lesegerät mit Strom versorgt werden. Neben einem Prozessor befinden sich EPROM/EEPROM, ROM (für ein kleines Betriebssystem) und RAM auf der Chipkarte. Dadurch können die Informationen der Karte durch komplexe kryptografische Algorithmen verschlüsselt und

die Karte dadurch sicherer gemacht werden. Einsatzgebiete solcher Karten sind zum Beispiel Geld- oder Identifizierungskarten. Grundsätzlich sind diese Chipkarten aber multifunktional einsetzbar, bisher aber im Vergleich zu normalen Speicherkarten sehr teuer.

### **1.3.4 Superchipkarten**

Dies wird wohl die zukünftige Kartengeneration sein, welche kontinuierlich weiterentwickelt wird. Chipkarten dieser Art besitzen neben den Funktionen einer Prozessorchipkarte zusätzlich ein eingebautes Minidisplay und sogar eine eigene Tastatur, wodurch der Benutzer z.B. PIN-Nummern direkt in die Karte und nicht in ein Terminal eingeben muß, was wiederum die Sicherheit erhöht.

Es gibt, neben den beschriebenen, noch weitere Unterscheidungsmerkmale von Chipkarten, so z.B. auf welche Art und Weise sie mit Strom versorgt werden, oder wie die Übertragung der Daten geschieht. Diese und andere Merkmale werden im nächsten Kapitel im Bezug auf einige Chipkartentypen etwas genauer erläutert.

## **Kapitel 2**

# **Genauere Erläuterung des Aufbaus und der Übertragung**

## 2.1 Magnetstreifenkarte

Auf einer Magnetstreifenkarte befinden sich 3 Magnetspuren. Die Spuren 1 und 2 sind für den Lesebetrieb gedacht. Die Spur 3 kann u.a. auch beschrieben werden. Die Speicherkapazität liegt bei ca. 1000 Bit. Zum Lesen des Magnetstreifens wird dieser von Hand oder maschinell an einem Lesekopf vorbeigezogen, wobei die Daten gelesen und elektronisch gespeichert werden.

Die Kreditkarten (ec-Karte) ist hierfür wohl das bekannteste Beispiel.

Ein großer Nachteil liegt darin, dass die Daten leicht ausgelesen und sogar verändert werden können.

Im Handel erhältliche Schreib und Lesegeräte verfügen über solch eine Möglichkeit.

## 2.2 Chipkarte

Während für Kreditkarten und ec-Karten bisher nur Magnetstreifenkarten verwendet wurden, werden für Wertkarten Chipkarten eingesetzt. Diese verfügen über eine integrierte Schaltung, die aus Elementen zur Datenübertragung, zur Speicherung von Daten und zur Verarbeitung von Daten besteht.

Die Speicherkapazität von Chipkarten ist um ein Vielfaches größer als bei Magnetstreifenkarten. Im Handel werden mittlerweile Chipkarten mit ca. 16 Kilobyte Speicher angeboten. Optische Speicherkarten allerdings besitzen eine noch höhere Speicherkapazität. Einer der wichtigsten Vorteile der Chipkarte liegt jedoch darin, daß die in ihr gespeicherten Daten gegen unerwünschten Zugriff und Manipulation geschützt werden können. Chipkarten werden nur dann akzeptiert, wenn sie eine hohe Sicherheit gegen Mißbrauch bieten können.

Folgende Punkte sollten dabei berücksichtigt werden.

- Benutzung der Karte durch Nichtberechtigte
- Auslesen von geheimen Daten
- Auslesen der Programmierung
- Ändern von Daten
- Nachahmung

Bei einer Chipkarte kann selektiv entschieden werden, welche Speicherbereiche veränderbar sind und welche es nicht sind. Die veränderbaren Bereiche können außerdem mit

einer Schutzlogik versehen werden, die nur bestimmte Veränderungen zulässt. So ist es bei der Telefonkarte mit einem Einheitenzähler erforderlich, dafür zu sorgen, daß der Zählerwert niemals durch die Besitzer der Karte erhöht werden kann, denn das würde einem Wiederaufladen der Karte entsprechen. Auch die Krankenversichertenkarte ist ins Kreuzfeuer der Kritik geraten, weil sie, je nach verwendetem Chip, von unbefugter Seite lesbar und zum Teil auch veränderbar ist. Ausführliche Anleitungen hierfür wurden bereits in Computer-Fachzeitschriften veröffentlicht. Die unzureichende Sicherheit der Krankenversichertenkarte liegt nicht an der Chipkartentechnologie, sondern daran, daß optionale Sicherheitsmerkmale für die Krankenversichertenkarte aus Kostengründen nicht gewählt wurden.

Wegen der Unterschiede in der Funktionalität, der Sicherheitslogik, aber auch im Preis, werden die Chipkarten in Speicherkarten und Mikroprozessorkarten unterteilt.

Auf einer Chipkarte sind mindestens folgende Anschlüsse zu finden:

- VCC (Versorgungsspannung)
- RST (Reset Signal)
- CLK (Taktsignal)
- GND (Masse)
- VPP (Programmierspannung)
- I/O (Datenein/ausgang)

## **2.3 Microprozessorkarte**

Die Microprozessorkarte besteht im wesentlichen aus folgenden Komponenten:

1. Microprozessor
2. RAM (Random Access Memory)
3. ROM (Read Only Memory)
4. EEPROM (Electrically Erasable Programmable Read Only Memory)
5. I/O Port

Die komplette Einheit funktioniert wie ein kleiner Computer. Im ROM befindet sich das Betriebssystem, welches bei der Herstellung eingebrannt wurde, und somit über die Lebensdauer der Karte nicht mehr veränderbar ist. Das EEPROM ist der nicht flüchtige Speicherbereich (wie eine schnelle Festplatte), wo Programm-Codes und -Daten vom Betriebssystem abgelegt werden können. Das RAM ist der flüchtige Speicherbereich. Nach dem Entfernen der Betriebsspannung gehen alle Daten verloren. Die serielle Schnittstelle besteht aus einem einzigen Register und steuert die Datenein- und -ausgabe. Mikroprozessorkarten haben den Vorteil, daß sie nach der Herstellung für vielseitige Anwendungsgebiete einsetzbar sind und man sie hierfür immer speziell programmieren kann. Desweiteren kann man über den int. Prozessor komplexe Verschlüsselungsalgorithmen ablaufen lassen, so daß der Sicherheitsaspekt um einiges besser ist als bei normalen Speicherkarten.

## 2.4 Kontaktlose Chipkarten

Kontaktlose Chipkarten funktionieren nach dem Prinzip der Induktion. D.h. es befinden sich mehrere Koppelpulen auf dem Chip, die bei Kontakt mit dem Terminal eine induktive Spannung aufbauen und somit eine Kommunikation möglich machen. Ansonsten ist das Funktionsprinzip gleich einer normalen Chipkarte. Der Vorteil hier ist, dass die Abnutzung vermindert wird. Es reicht wenn die Chipkarte (z.B im Geldbeutel) im Abstand bis zu einem Meter am Terminal vorbei geführt wird.

In vielen Wintersportgebieten ist der Einsatz solcher kontaktlosen Chipkarten ein immer beliebter Ersatz für Liftkarten.

Auch in anderen Bereichen, die eine schnelle Authorisationsprüfung erfordern, kommt dieses Medium immer mehr zum tragen.

## 2.5 Übertragungsarten

T=0 asynchron, halbduplex, byteorientiert

T=1 asynchron, halbduplex, blockorientiert

T=2 u. T=3 vollduplex

T=4 asynchron, halbduplex, byteorientiert, Erweiterung von T=0

T=5...T=13 Reserviert für zukünftige Anwendungen

T=14 für nationale Anwendungen, nicht von ISO genormt

T=15 Reserviert für zukünftige Anwendungen

## 2.5.1 I2C Bus

Der I2C-Bus wurde von der Firma Philips entwickelt und ist somit von anderen Anwendungen als einfacher und kostengünstiger Bus übernommen worden. Anwendung findet der Bus in Krankenversicherungskarten. Die Kommunikation erfolgt über 2 Leitungen (Serial Data und Serial Clock). Das Gerät, welches den Takt erzeugt (hier das Terminal), ist der Master und die Chipkarte, die den Takt übernimmt ist der Slave. Weitere Auskünfte über den I2C-Bus, bzw. Manuals erhält man auf den Internetseiten der Firma Philips, worauf an der dieser Stelle verwiesen werden soll.

## 2.6 Das eigene Programm

Unter anderem bestand unsere Aufgabe darin, ein eigenes Programm zu entwickeln, welches die Funktionen *Karte lesen* und *Karte beschreiben* erfüllt. Als Schnittstelle zum Terminal lag eine DLL (SCARD32.dll) von der Firma Towitoko zugrunde. Aus der Dokumentation der Schnittstellendefinition ging hervor wie die Karte anzusteuern ist. Unser Programm wurde mit Hilfe der MFC, Microsoft Foundation Classes, entwickelt. Wir haben uns für ein einziges Dialogfenster entschieden. Es bietet eine GUI, ist einfacher zu Handhaben, und erfüllt die Anforderungen bei weitem. Nachfolgend nun der Code, aus dem ersichtlich werden sollte wie das Terminal angesprochen wird.

### 2.6.1 Code

```
1 // chipkarteDlg.cpp : Implementierungsdatei
2 //
3
4 #include "stdafx.h"
5 #include "chipkarte.h"
6 #include "chipkarteDlg.h"
7
8
9 #ifdef _DEBUG
10 #define new DEBUG_NEW
11 #undef THIS_FILE
12 static char THIS_FILE[] = __FILE__;
13 #endif
14 //die Schnittstelle
15 typedef DWORD (__stdcall *SCardCmd)(LPDWORD Handle,
```

```

16         LPCSTR Cmd, LPINT CmdLen,
17         LPCSTR DataIn, LPINT DataInLen,
18         LPCSTR DataOut, LPINT DataOutLen);
19 // Strukt das nach belieben zur letztendlichen kommando"ubergabe
20 //gef"ullt werden kann
21 typedef struct SCARDCOMMANDstruct {
22     DWORD HMODULE;
23     char Cmd [100];
24     int CmdLen;
25     char DataIn [256];
26     int DataInLen;
27     unsigned char DataOut [256];
28     int DataOutLen;
29 } SCARDCOMMAND;
30 ///////////////VOM COMPILER VORGENERIERTE FUNKTIONEN////////////////////
31 ///////////////////////////////////////////////////////////////////
32 // CChipkarteDlg Dialogfeld
33
34 CChipkarteDlg::CChipkarteDlg(CWnd* pParent /*=NULL*/)
35     : CDialog(CChipkarteDlg::IDD, pParent)
36 {
37     //{{AFX_DATA_INIT(CChipkarteDlg)
38     // HINWEIS: Der Klassenassistent f"ugt hier Member
39     -Initialisierung ein
40     //}}AFX_DATA_INIT
41     // Beachten Sie, dass LoadIcon unter Win32 keinen nachfo
42     //lgenden DestroyIcon-Aufruf ben"otigt
43     m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
44 }
45
46 void CChipkarteDlg::DoDataExchange(CDataExchange* pDX)
47 {
48     CDialog::DoDataExchange(pDX);
49     //{{AFX_DATA_MAP(CChipkarteDlg)
50     DDX_Control(pDX, IDC_OUTBOX, m_outbox);
51     //}}AFX_DATA_MAP
52 }
53
54 BEGIN_MESSAGE_MAP(CChipkarteDlg, CDialog)
55     //{{AFX_MSG_MAP(CChipkarteDlg)
56     ON_WM_PAINT()

```

```

57         ON_WM_QUERYDRAGICON()
58         ON_BN_CLICKED(IDC_READ, OnRead)
59         //}}AFX_MSG_MAP
60 END_MESSAGE_MAP()
61
62 ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
63 // CChipkarteDlg Nachrichten-Handler
64
65 BOOL CChipkarteDlg::OnInitDialog()
66 {
67     CDialog::OnInitDialog();
68
69     // Symbol f"ur dieses Dialogfeld festlegen. Wird automatisch erledigt
70     // wenn das Hauptfenster der Anwendung kein Dialogfeld ist
71     SetIcon(m_hIcon, TRUE);           // Großes Symbol verwenden
72     SetIcon(m_hIcon, FALSE);        // Kleines Symbol verwenden
73
74     // ZU ERLEDIGEN: Hier zus"atzliche Initialisierung einf"ugen
75
76     return TRUE; // Geben Sie TRUE zur"uck, außer ein Steuerelement
77     //soll den Fokus erhalten
78 }
79
80
81 void CChipkarteDlg::OnPaint()
82 {
83     if (IsIconic())
84     {
85         CPaintDC dc(this); // Ger"atekontext f"ur Zeichnen
86
87         SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);
88
89         // Symbol in Client-Rechteck zentrieren
90         int cxIcon = GetSystemMetrics(SM_CXICON);
91         int cyIcon = GetSystemMetrics(SM_CYICON);
92         CRect rect;
93         GetClientRect(&rect);
94         int x = (rect.Width() - cxIcon + 1) / 2;
95         int y = (rect.Height() - cyIcon + 1) / 2;
96
97         // Symbol zeichnen

```

```

98         dc.DrawIcon(x, y, m_hIcon);
99     }
100    else
101    {
102        CDialog::OnPaint();
103    }
104 }
105
106 // Die Systemaufrufe fragen den Cursorform ab, die angezeigt werden
107 // w"ahrend der Benutzer
108 // das zum Symbol verkleinerte Fenster mit der Maus zieht.
109 HCURSOR CChipkarteDlg::OnQueryDragIcon()
110 {
111     return (HCURSOR) m_hIcon;
112 }
113 ////////////////////////////////////////////////////////////////////ENDE VORKOMPILATION//////////////////////////////////////////////////////////////////
114
115 //+++++++ Diese Funktion wird aufgerufen beim drucken von
116 //Button read+++++
117 void CChipkarteDlg::OnRead()
118 {
119     SCardCmd pSCardCommand = NULL;
120     SCARDCOMMAND chipkarte;
121     int response;           // R"uckgabewert des SCardServers
122     char* parser;          // Beginn des gesuchten Wertes
123     char* endparser;       // ende des gesuchten Wertes
124     int memlen;           // Gr"osse der Speicherkarte
125     int i;                // Z"ahler
126
127
128     HMODULE hSCardDLL = LoadLibrary("SCARD32.DLL");
129
130     if (hSCardDLL)
131         pSCardCommand = (SCardCmd)GetProcAddress(hSCardDLL, "SCardComand");
132     else
133         exit (-1);
134     chipkarte.HMODULE= 0; // nur eine Instanz ben"otigt, daher Handle =
135
136
137     m_outbox.AddString("DLL erfolgreich geladen...");
138

```

```

139
140 strcpy(chipkarte.Cmd, "Device,List");
141 chipkarte.CmdLen = strlen(chipkarte.Cmd);
142 chipkarte.DataInLen = 0;
143 memset(chipkarte.DataOut, 0, 256);
144 chipkarte.DataOutLen = 256;
145
146 response = pSCardCommand (      &chipkarte.HMODULE,
147                                chipkarte.Cmd,
148                                &chipkarte.CmdLen,
149                                NULL,
150                                &chipkarte.DataInLen,
151                                (const char *)&chipkarte.DataOut[0],
152                                &chipkarte.DataOutLen);
153     if (!response){
154         m_outbox.AddString("verfuegbare Terminals:");
155         m_outbox.AddString((const char *)&chipkarte.DataOut[0]);
156     }
157
158     FreeLibrary(hSCardDLL);
159 }
160
161 void CChipkarteDlg::Ausgabe (char *text)
162 {
163     int i=1;
164     CString str = "          ";
165
166     m_outbox.AddString ("Karten Text:");
167
168     for (i;i<=256;i++)
169     {
170         if( ! isprint( text[i-1] ) )
171             text[i-1] = '.';
172             str += text[i-1];
173
174             if (i%32 == 0)
175                 {
176                     //str += " ";
177                     m_outbox.AddString("");
178                     m_outbox.AddString(str);
179                     str.Empty();

```

```

180             str+="          ";
181
182         }
183     }
184 }
185
186 void CChipkarteDlg::Parser (char *text,int b)
187 {
188
189     int i = 0;
190     CString str;
191
192     for (i;i<=255;i++)
193     {
194
195         if( (text[i] == 13)&&(text[i+1] == 10) )
196         {
197             i+=2;
198             if (b == 1)
199                 m_status.AddString(str);
200             if (b == 2)
201                 m_info.AddString(str);
202
203             str.Empty();
204
205             if( (text[i] == 0)&&(text[i+1] == 0) )
206                 i=255;
207             }
208             str+=text[i];
209         }
210     }
211
212 void CChipkarteDlg::OnStore()
213 {
214     int index;
215     CString str;
216     char buffer [255];
217     int start,length;
218     start = 0;
219     int response;
220

```

```

221     SCardCmd pSCardCommand = NULL;
222     HMODULE hSCardDLL = LoadLibrary("SCARD32.DLL");
223     SCARDCOMMAND chipkarte;
224     if (hSCardDLL)
225     pSCardCommand = (SCardCmd)GetProcAddress(hSCardDLL, "SCardC
226     else
227     exit (-1);
228
229     chipkarte.HMODULE= 0; // nur eine Instanz ben"otigt, daher
230
231
232
233     //terminal auflisten
234     strcpy(chipkarte.Cmd, "Device,List");
235     chipkarte.CmdLen = strlen(chipkarte.Cmd);
236     chipkarte.DataInLen = 0;
237     memset(chipkarte.DataOut, 0, 256);
238     chipkarte.DataOutLen = 256;
239
240     response = pSCardCommand (           &chipkarte.HMODULE,
241                                     chipkarte.Cmd,
242                                     &chipkarte.CmdLen,
243                                     NULL,
244                                     NULL,
245                                     (const char *)&chipkarte.DataOut[0],
246                                     &chipkarte.DataOutLen);
247     if (!response)
248     {
249     //         if(comstatus)
250         {
251             m_status.AddString("verfuegbare Terminals:");
252             //m_status.AddString((const char *)&chipkarte.DataO
253             Parser((char*)chipkarte.DataOut,1);
254             //comstatus = false;
255         }
256     }
257
258     //Terminal ausw"ahlen
259     strcpy(chipkarte.Cmd, "Device,Select,Chipdrive Extern I");
260     chipkarte.CmdLen = strlen(chipkarte.Cmd);
261     response = pSCardCommand (           &chipkarte.HMODULE,

```

```

262                                     chipkarte.Cmd,
263                                     &chipkarte.CmdLen,
264                                     NULL,
265                                     NULL,
266                                     (const char*)&chipkarte.DataOut,
267                                     &chipkarte.DataOutLen);
268
269     if (!response){
270     m_status.AddString("Folgendes Terminal wird zum Schreiben genutzt");
271     Parser((char*)chipkarte.DataOut,1);
272     }
273
274
275     //KartenInfo
276     strcpy(chipkarte.Cmd, "Card,Info");
277     chipkarte.CmdLen = strlen(chipkarte.Cmd);
278     chipkarte.DataInLen = 0;
279     memset(chipkarte.DataOut, 0, 256);
280     chipkarte.DataOutLen = 256;
281
282     response = pSCardCommand (         &chipkarte.HMODULE,
283                                     chipkarte.Cmd,
284                                     &chipkarte.CmdLen,
285                                     NULL,
286                                     &chipkarte.DataInLen,
287                                     (const char *)&chipkarte.DataOut[0],
288                                     &chipkarte.DataOutLen);
289
290     if (!response)
291     Parser((char*)chipkarte.DataOut,2);
292
293
294     if (index = m_outbox.GetCurSel() == LB_ERR){
295
296         m_input.GetWindowText(str);
297         length = str.GetLength();
298         itoa(length,buffer,10);
299         memset(chipkarte.DataIn, 0, 256);
300         strcpy(chipkarte.Cmd, "Card,MemWrite,0,");
301         strcat(chipkarte.Cmd,buffer );
302         chipkarte.CmdLen = strlen(chipkarte.Cmd);

```

```

303         for (int b = 0;b < str.GetLength();b++)
304             chipkarte.DataIn [b]= str.GetAt(b);
305     chipkarte.DataInLen = str.GetLength();
306
307     response = pSCardCommand (        &chipkarte.HMODULE
308                                     chipkarte.Cmd,
309                                     &chipkarte.CmdLen,
310                                     (const char*)&chipkarte.DataIn,
311                                     &chipkarte.DataInLen,
312                                     NULL,
313                                     NULL);
314
315     if (!response)
316     {
317         m_status.AddString("Erfolgreich auf Karte geschrieben");
318         itoa(response,buffer,10);
319         m_status.AddString(buffer);
320     }
321     else
322     {
323         m_status.AddString("Daten konnten nicht auf Karten geschrieben");
324         itoa(response,buffer,10);
325         m_status.AddString(buffer);
326     }
327
328     }
329
330     FreeLibrary(hSCardDLL);
331 }
332
333 //Alle darstellungsfelder werden gel"oscht
334 void CChipkarteDlg::OnClearscreen()
335 {
336     m_outbox.ResetContent();
337     m_status.ResetContent();
338     m_info.ResetContent();
339     m_input.SetSel(0,-1);
340     m_input.Clear();
341 }
342
343 //Sollte eine vorauswahl getroffen werden, so kann

```

```

344 // "über diese funktion der Inhalt ausgelesen werden
345 void CChipkarteDlg::OnSelchangeOutbox()
346 {
347     CString str;
348     int i = 0;
349     int index;
350
351     index = m_outbox.GetCurSel();
352     m_outbox.GetText(index, str);
353     str.Delete(0, 12);
354     m_input.SetWindowText(str);
355     m_input.SetLimitText(32);
356
357 }

```

## 2.7 Kurze Erläuterung zur Abnahme

### 2.7.1 Probleme während der Programmerstellung

Während der Entwicklung des Programms hatten wir diverse Probleme, mit denen wir vorher nicht gerechnet hatten. Das Hauptproblem bestand darin, Daten auf Chipkarten zu verändern, bzw. neue Daten zu speichern. Die Informationen wurden nur sporadisch und zum großen Teil überhaupt nicht geschrieben, wohingegen der Lesevorgang ohne Probleme funktionierte. Auch am Tag der Abnahme bestand dieses Problem noch, welches wir aber letztendlich erfolgreich lösen konnten. Wichtig für den Schreibvorgang ist das erneute Auswählen des Terminals durch die Funktion *Device,Select* der SCARD32.DLL. Wir hatten fälschlicherweise angenommen, die Auswahl des Terminals beim Lesevorgang genüge, was aber nicht der Fall zu sein scheint. Nach der Verbesserung unserer Speicherfunktion (OnStore() ), dadurch, dass erneut die zur Verfügung stehenden Terminals angezeigt und das richtige ausgewählt wurde, funktionierte das Speichern von Daten auf den Chipkarten reibungslos.

## 2.7.2 Sicherung der Labore durch Chipkarten

Während der Abnahme bekamen wir unter anderem die Aufgabe, uns eine Sicherung der Computerlabore durch die Verwendung von Chipkarten vorzustellen, bzw. wurde uns die Frage gestellt, wie wir sie realisieren würden. Unser erster Gedanke war eine Sicherung durch gegenseitige asymmetrische Authentisierung. Das hieße, die Sicherung des Zugangs wäre in Form einer Art RSA-Verfahren erfolgt.

(siehe <http://www.fh-augsburg.de/wizard> → dva → erster versuch) Grundsätzlich ist der Gedanke, die Labore so sicher wie möglich zu machen, nicht schlecht. Ein Zugangsverfahren, wie es oben beschrieben ist, hätte aber zur Folge, dass jeder, der sich Zutritt zu einem Labor verschaffen will, zuerst die Chipkarte in ein Terminal einführen und danach eine PIN-Nummer hätte eingeben müssen, was den Vorgang umständlich und langwierig machen würde. Ferner waren wir uns nicht einig, welche Art von Daten, bzw. wieviele Daten auf der Karte gespeichert werden sollten. Im Gespräch mit Herrn Prof. Lutz stellte sich heraus, dass die sinnvollste Variante einer Zutrittssicherung eine einfache (nicht beschreibbare) Speicherkarte, auf der eine einzige Nummer gespeichert ist, wäre. Jedem Studenten würde nur eine Nummer zugeordnet, welche ihn dazu berechtigt, ein bestimmtes Labor zu betreten. Zudem ließe sich anhand dieser Nummer im Nachhinein auch feststellen, zu welchem Zeitpunkt er welches Labor betreten hat. Weiterhin ist auch der Verlust einer Karte kein großes Problem, da eben nur eine Nummer und keine Daten des jeweiligen Studenten darauf gespeichert sind. Ein etwaiger Finder kann mit der Karte nichts anfangen, wenn derjenige, der sie verloren hat, binnen möglichst kurzer Zeit den Verlust seiner Karte meldet. Die Nummer der abhandenen Karte würde einfach für den Zugang gesperrt und das Problem wäre beseitigt.

Der Versuch war sehr informativ und es war interessant, sich die verschiedenen Arten von Chipkarten, bzw. deren Funktionsweise vor Augen zu führen. Ein kleiner Wermutstropfen allerdings war, dass es nicht möglich ist, die Funktionen der SCARD32.DLL zu betrachten, um deren genauen Aufbau zu verstehen. Trotzdem sind wir mit dem Ablauf des Versuchs und mit dem Ergebnis zufrieden.