卒 業 研 究 報 告

題　　目

VC++を用いた
入退室管理システムの開発

指 導 教 員

八田　章光

報 告 者

中元　雅美

平成　14年　2月　5日

高知工科大学 電子・光システム工学科

目次

第 1 章

序論

一昔前は、深夜や外出時に鍵をかける家庭は多くはなかった。しかし、現在では在宅でありながらも閉鍵は必須となり、防犯カメラを設置する家庭も増加している。自分の身は自分で守らなくてはならない社会となった。

一方情報の電子化やネットワーク化により、企業情報や個人情報の管理が一層重要となっている。そのためセキュリティーシステムは現代社会において欠かせないものとなっている。

本校、本学科では現在、研究室への入室は指紋認証により行われている。認証システムは学科ごとに異なっている。指紋認証は認証セキュリティーシステムにおいて個人的属性認証にあたり、高水準セキュリティーとして幅広く利用されている。

しかし、多くの学生が入室の際に費やされる時間に苛立ちを覚えているのが現状である。そこで、もっと簡単に、かつセキュリティー面においても高水準であるシステムを開発することを考えた。

だが、セキュリティー面での水準の高さを追求していると、やはり使いやすさが損なわれてしまう。しかし、今回はまず使いやすさに重点をおきパスワードによる簡便な認証システムを開発することにした。利用者が簡単に使用できるのは、現在パソコンのログインなどに利用している学籍番号とパスワードの入力による入退室が最適だと考えた。

本研究の目的は、最新の認証セキュリティーシステムについて学ぶとともに、VisualC++を用いて、簡便なユーザーのパスワード認証とログファイルの保存を行うシステムを開発することである。

第 2 章

セキュリティーシステム

現代社会では、セキュリティーといってもいろんな種類のものがある。鍵や雨戸を閉めることもセキュリティーなら、電子情報を暗号化して情報を漏らさないようにすることもセキュリティーである。

本章ではセキュリティーシステムについて色々な観点から学ぶことを目的とした。

2.1　認証セキュリティーシステムの必要性

情報技術は社会の変化を加速させる性格を持っている。スピード化、ボーダレス化、オープン化といった社会経済に影響をもたらす情報技術の特性は、情報の戦略的活用を必須とし、世界市場の中で競争激化相互依存をもたらす。

現在では、特にマルチメディアはインターネットのもたらす影響が強いといえるだろう。ネットワーク情報は匿名性、時間・距離の超越性、場の不特定性、無痕跡性といった性質がある。このような現在において、重要とされているのが高水準な認証セキュリティーシステムである。

2.2　認証セキュリティーシステムの分類

従来、セキュリティーを実現するシステムはパスワードによるものがほとんどであった。このシステムは、パスワードは本人しか知らない情報であることを根拠にして認証を行っていた。しかし、実際にはパスワードを読み取られ、他人になりすまされる危険性が高い上に、パスワードを忘れてしまえば情報にアクセスすることさえできなくなってしまうという欠点があった。また、最近

ではパスワードを利用するシステムが増えている。これらに同一のパスワードをあてていては情報が読み取られる危険性が高くなってしまう。

したがってそれぞれのシステムごとにパスワードを変えるため、利用者は複数の異なったパスワードを記憶しておかなくてはならず、負担は増えるばかりである。

これらの問題を解決する認証セキュリティーシステムにおいて高水準であると注目されたものが、バイオメトリクス(生物学的特徴)を用いたシステムである。

バイオメトリクス・システムは個人の身体的特徴を用いて認証を行うシステムであり、すでに様々なものが実用化されている。バイオメトリクス・システムには指紋・顔形・網膜パターン・虹彩パターン・声紋照合技術などが現在までに開発されている。原理としては、一つの身体的特徴を一つの数字に変換し、この数字にもとづいてディジタルにエンコードされた識別情報を作る。二人以上の人間で同じ身体的特徴を持つことはありえないため、識別された情報の数字とアルゴリズムにで形成された個人プロフィールは、ただ一人、その本人にしか当てはまらないことになる。

バイオメトリクス・システムの利点は、第一に忘れたり、置いてきたりすることがない。第二に偽造や本人になりすますことがほぼ不可能である。第三に終生不変なものであることがあげられる。

現在の認証セキュリティーシステム例のいくつかを表.1に示す。

表.1　セキュリティーシステム

種類	内容	技術
利用者認証	知識認証	パスワード
〃	所有物認証	磁気カード・IC カード
〃	個人的属性認証	指紋・声紋・網膜パターン照合 etc(バイオメトリクス類)
メッセージ認証	改ざんの検出	サーファーフィードバック法
ディジタル署名	発信元の保証	一方向性関数・公開鍵暗号の応用

知識認証ではパスワードによる認証になる。現在で最も多く使用されている技術の一つといえるだろう。しかし、メモ書きした紙、パスワードの打ち込みでの解読、本人からの聞き出しなどで人の手にわたってしまう危険性が高い。また、長期間同じパスワードを使用することで読み取られやすくなってしまう。

個人的属性認証などは偽造することは困難であり、セキュリティーシステム

において高水準である。しかし、個人の微妙な特徴の差を識別することで照合が行われるため、認証に時間を取られるとことが難点である。現在、高知工科大学、電子・光システム工学科ではこの指紋認証システムによるセキュリティーが施されている。

所有物認証は主に IC カードにより認証されるシステムである。そのため IC カードそのものがなくては認証にはいたれず、また複製が可能なことからセキュリティー面においては低水準とされている。そのため、実際には知識認証や個人的属性認証などと組み合わせて利用されることが多い。本校ではこの方法で休日・夜間の管理を行っている。

今回の研究では、知識認証でのパスワードを利用する。今回の研究では研究室の利用者管理を目的としている。そのため、利用者はほぼ固定されている。現在、研究室の入室には指紋照合によって行われているが、指紋を認証されずらい人も多々存在するため、入室までに多くの時間を費やしてしまう状態にある。本研究はその入室までの時間をどれだけ短縮できるかというのも目的としている。

2.2　最新のバイオメトリクス・システム

バイオメトリクス・システムには指紋・顔形・網膜パターン・虹彩パターン・声紋照合技術などが現在までに開発されている。特に最近では、技術の向上により指紋照合製品はかなり小型化が進み、一般にも広く利用されるようになっている。また、パソコン接続型及びパソコン内蔵型の装置についても、既に多数の会社により製品化されている。認証に要する時間も約 0.3 秒と極めて短いことから、有力な端末・本人間認証技術として広く普及することが期待されている。

だが、指紋認証には問題となる点も多い。指先の表面の模様であるため、擦り減ったり傷があったりすると認証されづらい。指先を頻繁に使う職業や趣味を持つ人には使い勝手が悪い。また、指を押し付ける強さによっても認証がされないこともある。また、指紋の多くはパターンにより分類できるのだが、ごく稀にまったく分類に入らない指紋を持つ人物がいるが、その人物はまったく認証されない。どのような場合でもある程度の認証ができなければ利用者の使い勝手が悪くなってしまう。(参考文献１)

顔形照合技術はあまり馴染みがない。これは正面から顔型を画像で取り込み、登録画像データと照合する技術である。非接触での本人確認が可能であり、利用者の心理的負担が少ないというメリットがある。しかし、顔画像データを取

り込む際に、顔の向きや距離等の厳密性が要求されることや、照合対象物の性質上、精度にやや問題がある。このため、一般での実用化には更なる研究を必要としている。現在、顔写真データを圧縮して IC カードに記憶させるという認証技術の開発が進められている。なお、顔形による認証時間も約 1 秒以内と極めて短い。(参考文献２)

網膜パターン照合技術は、網膜に対し赤外線を直径 5mm程度照射することにより網膜の血管パターンの個人差を計測し、認証する。発光強度は自然界にも存在するもので微弱であり安全とされている。識別能力は高く、放射線施設などの重要管理施設における入退室管理装置として現実に活用されている。しかし、装置自体が高価であり、複数の利用者が同じところに目を押し付けるため、衛生上好まれない。そのため一般向けの事例は少ない。(参考文献１)

虹彩パターン照合技術は、目の瞳孔の縮小・拡大を調整する網膜組織、虹彩の個人固有の文様を識別し、個人を認識する。その精度は極めて高く、誤認認識率は 120 万分の 1 と極めて低い。認証時にはカメラの前に立つだけでよい非接触型であるため利用者にも拒まれることはない。しかし、システムが比較的高価であり、エラーも起きやすい。(参考文献３)

声紋照合システムは声の波形に表れる特徴を抽出し、その個人差を識別するものである。声を出すことは人間にとって自然であり、利用するにあたって不快感はないものの、体調やノイズに左右され、認識されないこともある。また、登録者の声をテープに録音し不正使用される可能性がある。(参考文献４)

現在、新しく静脈パターン認証方式とＤＮＡ認証方式のシステムが製品化されている。

静脈パターン認証は、手の甲にある静脈のパターンを認証に用いる。しかし、まだ一般的ではない。製作会社によると汚れや汗などがあっても認識可能だとされているが、今のところ製品化されてはいるものの、その信頼性においてははっきりとしたことがわかっていない。しかし、非接触型であるため、利用者は抵抗なく使用できるだろう。(参考文献５)

DNA は細胞の遺伝子に含まれる塩基配列情報で、理論上は全人類を識別できる。しかし、DNA 全体は膨大な情報量であり、管理や識別は容易ではない。また、個人の身体的特徴や遺伝病に関する情報が含まれているため、重大なプライバシー問題を含む。このため、DNA すべての抽出と登録に対しては利用者が拒否する可能性が高い。そこで、個人識別用の DNA 情報は、多重ＳＴＲ（Short Tandem Repeat）配列法という方法により生成する DNA-ID を使用する。STR はいわゆる遺伝子領域以外の部分に存在する DNA であり、病気や人体の構造には無関係である。それでいて個人の識別には十分な情報が含まれていることが理論と実験で証明されている。まだ具体的な製品はなく、開発段階にある。

(参考文献６)

参考文献

１)Security & Trust ホームページ
http://www.atmarkit.co.jp/fsecurity/index.html

２)東芝　e-ソリューション社　ホームページ
http://www3.toshiba.co.jp/ccc/index1_j.htm

３)沖電気株式会社　ホームページ
http://www.oki.com/jp/

４)企業アニモ　ホームページ
http://www.animo.co.jp/index.html

５)東西電気産業株式会社　ホームページ
http://www.tozaidensan.co.jp/index.htm

６)株式会社　NTT データ　ホームページ
http://www.nttdata.co.jp/index.html

第 3 章

プログラム言語

本研究ではマイクロソフト社が開発した Microsoft VisualC++ Development System Version6.0(以下 VisualC++)を利用して、簡便な認証、ログファイルシステムを Windows アプリケーションとして作成した。

3.1 C 言語

今回、プログラムを作成するために使用する言語は VisualC++である。VisualC++は C 言語の後継言語と言われている。VisualC++を使用するにあたり、まず C 言語を学んだ。

本格的アプリケーションの開発に使用されるプログラム言語のひとつが C 言語である。C 言語は 1969 年に Martin. Richards が開発した BCPL 言語を Kenneth.L.Tompson が拡張・開発した B 言語から Dennis.M.Ritchie によって 1972 年に開発された。もともとは Unix システムを記述するために開発された言語である。(参考文献 1)

Unix を簡単にいえば、タイムシェアリングシステムにより 1 台のパソコンを複数人で同時に使用可能にした OS である。

タイムシェアリングシステムとは、同一の計算機を複数の使用者で同時に使用し、なおかつ各々の利用者があたかも自分が計算機を専有して使っているかのように見える処理方式である。ここでは、お互いの処理内容はまったく関係ない独立なものになっている

3.2　C 言語と VC++言語の違い

C 言語の特徴にはデータ演算子が多く、簡潔明瞭に効率のよいプログラムが書けること、ユーザー関数の定義により言語拡張が可能であること、また異なるシステムでも移植が容易であることがあげられる。移植が容易であるのはアセンブリ言語に近い記述力をもち複雑なプログラミングが可能なことにより実現している。計算機を動かすためには計算機が直接解釈できる機械語命令を読ませる必要があるが、人間が書いたプログラムをコンパイル(翻訳あるいは編集)して機械語命令に変換するのがコンパイラで、人間が書いたプログラムをアセンブル(直訳あるいは組み立て)して機械語命令に変換するのがアセンブラーである。

アセンブリ言語は、記述できる命令自体は機械語命令と 1 対 1 に対応するもので、ジャンプ命令の飛び先やデータ領域などのアドレスを記号で書いておくことができる言語である。(参考文献２)

VisualC++の特徴は面倒な作業はすべて付属のツールがサポートしてくれ、しかも定型処理部分の(ほぼ)自動的なコーティングも行ってくれる点である。また、Windows のために設計されたライブラリのおかげで、GUI(Graphical User Interface)を利用した独特のスタイルのプログラムが、簡単に、かつ簡潔に記述できる点である。

3.3　オブジェクト指向

プログラミング言語には 2 種類の型がある。C 言語に代表される「手続き指向型言語」と VC++に代表される「オブジェクト指向型言語」である。

手続き型指向とは、どのような作業を、どのような手順で行うかをプログラミング時の基本的考え方とするものである。C 言語以外には Fotran、Cobol、Basic などがあげられる。(参考文献３)

オブジェクト指向とは、処理の対象はどのようなものから構成され、それは何を指定することによって、どのような働きをするかを記述することを、プログラミング時の基本的考え方とすることである。VC++以外には Java Script などがあげられる。

オブジェクト指向言語の概念は一般的に内部構造をもち、普通オブジェクト指向言語は以下の 3 つの機能を実装していなくてはならない。下記にその機能を示す。

① **カプセル化**（encapsulation）
② **クラスの継承**(inheritance)
③ **多態性**(polymorphism)　　(参考文献４)

カプセル化とは、情報(プロパティ)や振る舞い(メソッド)をグループ化し、オブジェクトへ格納して情報を隠蔽し、更に振る舞いの具体的な内部処理も隠蔽することである。

簡単にいえば、データとそれに対する手続きを一体化し、データへのアクセスは決められた手続きしか許されないようにすることである。このカプセル化したものをオブジェクトと呼ぶ。例えば TV の受信機などがそうである。見る側はスイッチの操作のみを許され、どうやって TV が受信されるかなどはわからない。ただ、スイッチを押すことで TV が見られることだけがわかっている。

クラスの継承とは、すでに作成されているクラスの機能(メソッド、プロパティ)を新たに作成するクラスに引き継ぐことである。簡単にいえば、クラス定義の際、他のクラスの定義や手続きを引き継ぐことである。例えば、「インフルエンザウィルスに感染して 3 日間寝込んだ」という表現は間違いではないが、「コンピュータウィルスに感染して 3 日間寝込んだ」という表現は間違いである。

これは、コンピュータウィルスは人間が感染するものではない、ということをその人に継承されていなかったためである。(参考文献６)

多擬性とは、クラスの継承をする場合に、子クラス内で親クラスで定義されているメソッドを上書きすることである。簡単にいうと、同じ名前のメッサージを送っても、受け手によって適切な手続きが呼び出されることである。例えば、「今何時ですか？」と聞かれ、「午後 2 時です」と答えられることだ。このことから、今、自分はどの場所に存在しているのかを理解することができる。(参考文献５)

3.4　GUI（グラフィカル・ユーザ・インターフェース）

Windows95 の爆発的な流行とともに、日本でもグラフィック画面をベースとするパソコン操作環境が一般のものとして定着した。日ごろパソコンのデスクトップ上に並べられたマルチウィンドウ・ボタン・メニュー・スライダー・アイコンそしてごみ箱といったオブジェクトに対し、マウスを操作することでメッセージを送っている。これが GUI である。GUI は直感的でわかりやすい。卓上で仕事をするイメージをそのまま表現している。(参考文献６)

内部に存在するデータや手続きが抽象化されたオブジェクトとなっていくわけだが、ユーザーにとってみれば見えているのは実環境のシミュレーションなので、感覚や慣れによって操作することが可能なのである。

参考文献

１)発行所：技術評論社　　著者：塚越一雄
「はじめての C 言語　完全入門」

２)発行所：技術評論社　　著者：株式会社アンク
「最新　これからはじめる　VisualC++」

３)発行所：共立出版株式会社　　著者：森澤良彰　栗野俊一
「オブジェクト指向プログラミング入門　C から C++へ」

４)発行所：森北出版株式会社　　著者：安良勝也
「一晩で理解するオブジェクト指向　C から C++へ」

５)発行所：技術評論社　　著者：塚越一雄
「決定版　はじめての C++」

６)発行所：工学図書株式会社　　著者：杉原敏夫
「C++とオブジェクト指向」

７)発行所：アスキー出版局　　著者：桜田幸嗣　田口景介
「VisualC++　プログラミング入門」

第 4 章

入退室管理システム

第 2 章で述べたように､セキュリティーとして高水準とされるのはバイオメトリクスを用いたシステムである。しかし、バイオメトリクスを用いたシステムには共通する欠点があった。

バイオメトリクス・システムはほとんどといって良いほど誤認照合がない。認証がスムーズに行われれば、約 1 秒ほどで入室が可能になる。しかし、精密な照合による認証システムであるため、エラーがでやすいのである。

本学科では研究室への入室は指紋照合システムにより行われている。実際に研究室内に入るとき、指紋照合がうまくいかず悪戦苦闘した覚えのある人は大勢いるだろう。

エラーがでる主な理由は、セル面の汚れや自らの指先の汚れにより認証が困難になる場合が多い。また、稀なケースでは、個人によっては照合成功率が極めて低い指紋の持ち主がいる。この指紋を持つ人物は全く認証が行われない。

日常で頻繁に利用する場所に、思うように入ることが出来なくては利用者にストレスを与えてしまう。こうした背景から利用者にも管理者にも使いやすさを重視したセキュリティーシステムの開発を行った。

4.1　概要

このシステムは研究室などの利用者に限りある共有スペースの管理と、入室時間の短縮を目的としている。利用者に限りがあるため、あらかじめ名簿を作成することが可能で、名簿以外の利用者については許可を必要とする。

画面上のダイアログボックスの入力欄に学籍番号と暗証番号を入力することで扉は開錠され、自動的に入室した時間が記録されることを目的として開発した。実際には扉の開閉の制御までは行わないものとする。

メニューから入室を選択する。選択されると学籍番号・パスワードを入力するダイアログボックスが表示される。ここに自分の学籍番号とパスワードを入力し、OK ボタンを押す。学籍番号とパスワードが認証されると入室が可能になる。入室が完了すると、学籍番号と入室時間がテキストファイルに保存される。

退室時も同様であるが、学籍番号のみをダイアログボックス内に表示させ、それを選択することで退室時間の記録が行われるようにする。

入室時に行われる動作フローチャート(図.１)を下記に示す。

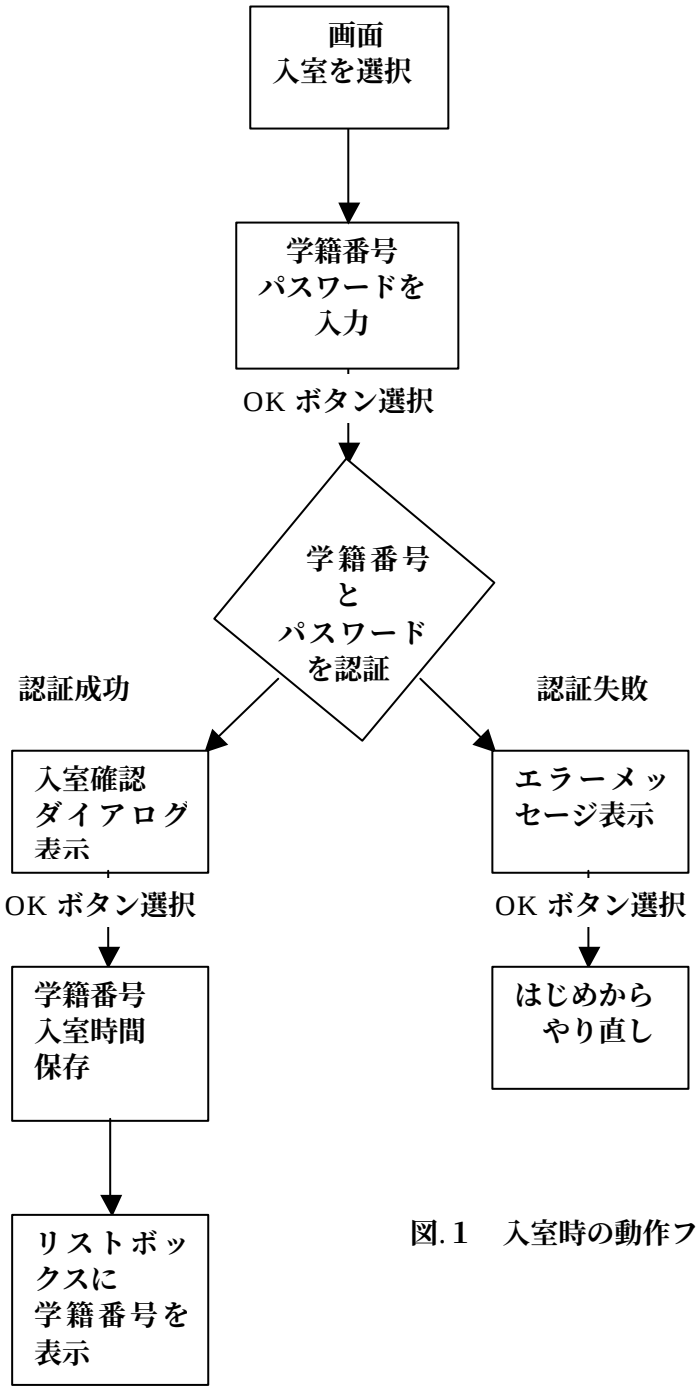

図.１　入室時の動作フローチャート

次に、退室時の動作フローチャート(図.２)を下記に示す。

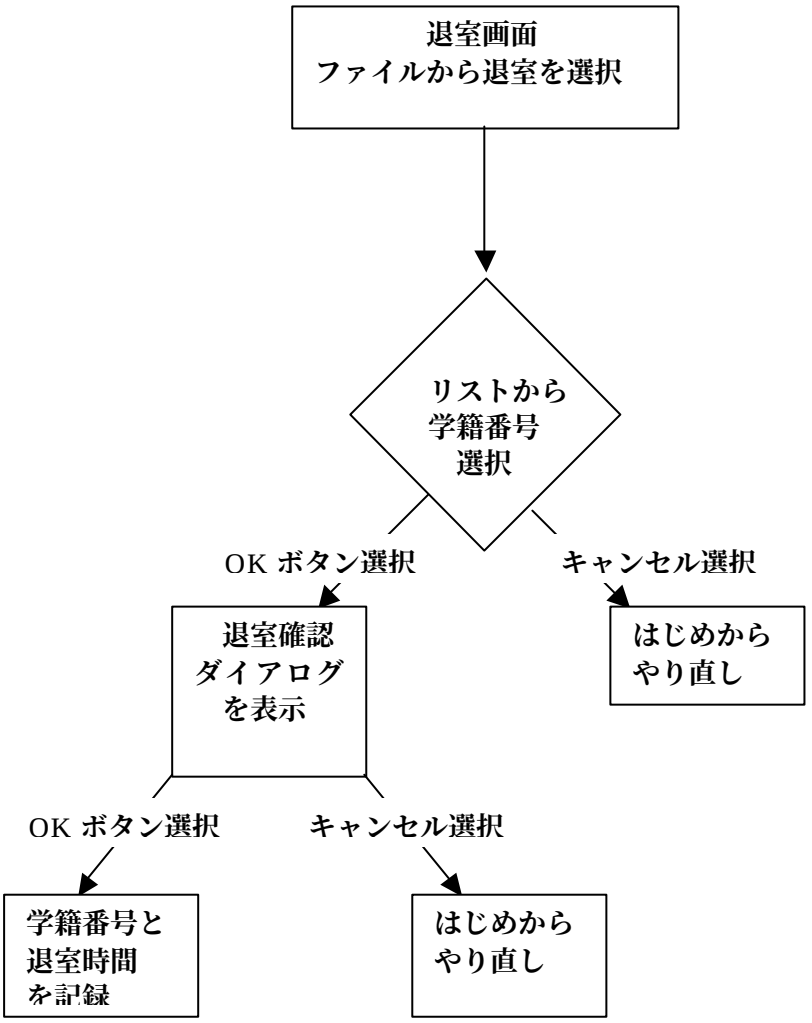

図.２　退室時の動作フローチャート

4.2　入室画面

ファイルメニューの入室を選択する。下の図.３はその画面の拡大図である。すると、学籍番号とパスワードを入力するダイアログボックスが表示される。

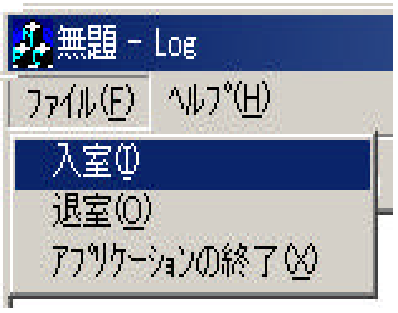

図.３メニュー画面

ダイアログボックスの表示方法には２種類の方法がある。モーダルとモードレスだ。今回、ダイアログボックスの表示方法にはモーダルを利用した。

モーダルとは、ダイアログボックスが閉じられるまで、他のウィンドウをアクティブにできないものである。

モードレスは、ダイアログボックスを閉じなくても他のウィンドウをアクティブにできるものである。

今回、モーダルを使用したのは、入室の際に他のウィンドウをアクティブにしておく必要はないと考えたからである。

図.４に入室用ダイアログボックスを示す。

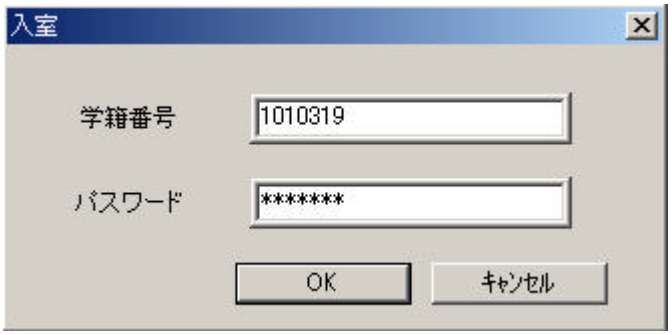

図.４　入室用ダイアログボックス

ダイアログボックス内を、学籍番号とパスワードの文字表示はスタティックテキストで、入力スペースはエディットボックスで作成した。

学籍番号とパスワードを入力し、入力が正しければ「入室します」と表示されたダイアログボックスが表れる。入力が間違っていれば、「学籍番号またはパ

スワードが違います」と書かれたダイアログボックスが表示される。OK ボタンを押すと入室完了である。しかし、今回の本論文ではパスワードと学籍番号の認証を実行することができなかった。本論文でのプログラムは、学籍番号とパスワードが同じ文字や数字であったとき認証完了のダイアログボックスが表示されるようになっている。

退室用ダイアログボックスも入室用ダイアログボックスと同様に、文字入力部分にはエディットボックスを用いた。

退室の際には利用者が特定されているため、現在入室している利用者の学籍番号をそのまま表示させることにした。学籍番号を複数表示させるためにリストボックスを用いた。

リストボックス内にある学籍番号を選択し、OK ボタンを押すと「退室します」と書かれたダイアログボックスが表示される。OK ボタンを押すと退室完了である。

図.５は退室用ダイアログボックスである。

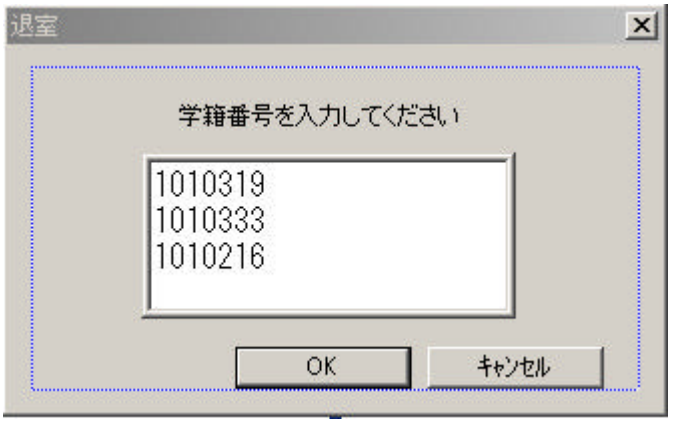

図.５　退室用ダイアログボックス

図.５のリストボックス内のある学籍番号はイメージ上のものである。リストボックス内に、入室が確認されると学籍番号が表示されるのだが、今回その動作を成功させることはできなかった。

リストボックス内に学籍番号を表示させたままにしておくのはセキュリティー上あまり良くないのだが、コントロールを使うクラスの学習の一環としてこの方法を選択した。

4.2　時間記録とその集計

記録はテキストファイルに保存される。入室時は、入力がされ OK ボタンをクリックした時間を記録する。このとき学籍番号も表示される。このとき、テキストファイルにどのように記録されるかを図.6 に示す。

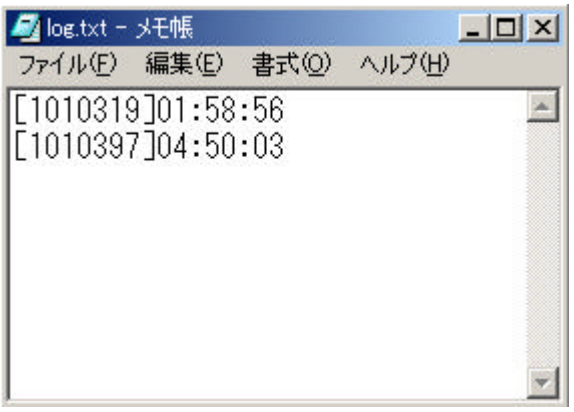

図.6　入室記録テキストファイル

なお、入力後の OK ボタン選択で時間と入力された学籍番号が記録されるため、不正に入室しようと適当な学籍番号でパスワードを入力され続けても、全てその入力値は記録されていくこととなる。

退室時の学籍番号と時間の記録は別のテキストファイルに記録さることにした。理由は、データを集計を行うのに Excel に記入したいためである。
そのデータ集計例を表.2 に示す。

表.2　データ集計例

学籍番号	入室	退室	入室	退室
1010111	09：00	12：10	13：45	18：00
1010222	10：35	11：45		
1010333				
1020111	10：00	11：25		
1020222	09：20	11：00	12：55	19：10
1020333				
1030111	07：50	10：30	13：10	15：00
1030222				
1030333	14：50	16：25		

データ集計は日ごと、月ごとに行うものとする。これをグラフ化し、個人の利用頻度や不正アクセスがなかったかなどを検討することができる。

4.3　応用

このシステムの応用としては、装置使用者記録があげられる。

本学科では実験に装置を利用することはあたりまえである。その電気使用料金も多額である。当然だが、研究室ごとに使用する装置や使用時間は異なる。しかし、電気使用量は研究室ごとに分かれておらず、どの研究室がどれだけ使用したかがわからない。

しかし、システムを応用することで、装置使用者の記録を採ることができる。

使用者は装置使用時に学籍番号とパスワードを入力する。入力が確認されると学籍番号と入力時刻が記録される。このとき、使用終了の入力が行われるまでの間、画面上に使用者の名前が表示されるようにする。こうすることで、使用者が周りにもわかり、また、電源を入れたままの放置防止に役立つと考えたからである。

使用終了の入力が行われると使用時間の算出を行うようにする。これは一日ごとに集計し、データを保存しておく。

このように装置使用者、使用時間を記録し、装置の使用者所属の研究室別に電力使用量を算出することで、電気代の予算を研究室に公平に配分することができる。

参考文献

これらの参考文献はプログラムコードを作成する際に参考にさせて頂きました。

１)発行所：工学図書株式会社　　著者：杉原敏夫
「C++とオブジェクト指向」

２)発行所：アスキー出版局　　著者：桜田幸嗣／田口景介
「VisualC++　プログラミング入門」

３)発行所：技術評論社　　著者：塚越一雄
「決定版　はじめての C++」

４)発行所：技術評論社　　著者：株式会社アンク
「最新　これからはじめる　VisualC++」

５)発行所：株式会社翔泳社　　著者：山本信雄　著
「プログラミング学習シリーズ　VisualC++①
はじめての Windows プログラミング」

６)発行所：株式会社翔泳社　　著者：山本信雄　著
「プログラミング学習シリーズ　VisualC++②
はじめてのオブジェクト指向プログラミング」

７)発行所：株式会社翔泳社　　著者：山本信雄　著
「プログラミング学習シリーズ　VisualC++③
はじめての MFC プログラミング」

第 5 章

結論

本論文での最終的な入退室管理システムプログラムは、学籍番号とパスワードが同じであるときのみ認証が行われ、テキストファイルに番号と時刻が表示されるまでとなった。最終的目標である認証システムまでに達成できなかった。

しかし、本研究を通じて、C 言語と VC++言語の違いや基本的考え方を学ぶことができた。

リストボックスへの学籍番号の表示をすることができなかった。コード自体にエラーは発生しないが、実行する際にデバッグでエラーが起きてしまい、これを解決できないままとなってしまった。

認証は、学籍番号とパスワードが違ったときのみエラーを出すのみにとどまってしまった。学籍番号、パスワードを格納したファイルにリンクさせることができなかった。

今後の課題として、リストボックスの操作方法、アルゴリズムの復習、そして ODBC 及び DAO の操作方法を学ぶことでこのシステムを完成させることができる。

謝辞

今回の論文を書くにあたって、指導教員である八田章光助教授をはじめ、同研究室の方々から数々のご指導・ご助言を頂いたことに深く感謝し、また論文を書くにあたり参考とさせて頂いた著者の方々に対しても感謝の意を表したい。最後に、たくさんの方々から頂いた助言を今後の更なる発展に繋げていきたい。

参考資料

入退室管理システムの学籍番号と時間の入室時の記録を実現したプログラムを以下に記す。

＜Log.h : LOG アプリケーションのメイン ヘッダー ファイル＞

```
// CLogApp:
// このクラスの動作の定義に関しては Log.cpp ファイルを参照
class CLogApp : public CWinApp
{
public:
        CLogApp();

// オーバーライド
        // ClassWizard は仮想関数のオーバーライドを生成
        //{{AFX_VIRTUAL(CLogApp)
        public:
        virtual BOOL InitInstance();
        //}}AFX_VIRTUAL

// インプリメンテーション
        //{{AFX_MSG(CLogApp)
        afx_msg void OnAppAbout();
        // メモ - ClassWizard はこの位置にメンバ関数を追加または削除
                //   この位置に生成されるコードを編集禁止
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

＜LogDoc.h : CLogDoc クラスの宣言およびインターフェイスの定義＞

```
class CLogDoc : public CDocument
{
protected: // シリアライズ機能のみから作成します。
        CLogDoc();
        DECLARE_DYNCREATE(CLogDoc)

// アトリビュート
public:

// オペレーション
public:

//オーバーライド
        // ClassWizard は仮想関数のオーバーライドを生成
        //{{AFX_VIRTUAL(CLogDoc)
        public:
        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);
        //}}AFX_VIRTUAL

// インプリメンテーション
public:
        virtual ~CLogDoc();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// 生成されたメッセージ マップ関数
protected:
        //{{AFX_MSG(CLogDoc)
                // メモ - ClassWizard はこの位置にメンバ関数を追加または削除
                // する。この位置に生成されるコードを編集禁止
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

＜CLoginDialog ダイアログ＞

```
class CLoginDialog : public CDialog
{
// コンストラクション
public:
	CLoginDialog(CWnd* pParent = NULL);   // 標準のコンストラクタ

// ダイアログ データ
	//{{AFX_DATA(CLoginDialog)
	enum { IDD = IDD_LOGIN };
	CStringm_Id;
	CStringm_Passwd;
	//}}AFX_DATA

// オーバーライド
	// ClassWizard は仮想関数のオーバーライドを生成します。
	//{{AFX_VIRTUAL(CLoginDialog)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);
                                                    // DDX/DDV サポート
	//}}AFX_VIRTUAL

// インプリメンテーション
protected:

	// 生成されたメッセージ マップ関数
	//{{AFX_MSG(CLoginDialog)
		// メモ: ClassWizard はこの位置にメンバ関数を追加
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
```

＜CLogoutDialog ダイアログ＞

```
class CLogoutDialog : public CDialog
{
// コンストラクション
public:
        CLogoutDialog(CWnd* pParent = NULL);   // 標準のコンストラクタ

// ダイアログ データ
        //{{AFX_DATA(CLogoutDialog)
        enum { IDD = IDD_LOGOUT };
        CListBox        m_List;
        //}}AFX_DATA


// オーバーライド
        // ClassWizard は仮想関数のオーバーライドを生成
        //{{AFX_VIRTUAL(CLogoutDialog)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);
                                                // DDX/DDV サポート
        //}}AFX_VIRTUAL

// インプリメンテーション
protected:

        // 生成されたメッセージ マップ関数
        //{{AFX_MSG(CLogoutDialog)
                // メモ: ClassWizard はこの位置にメンバ関数を追加
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

＜LogView.h : CLogView クラスの宣言およびインターフェイスの定義＞

```
class CLogData : public CObject
{
public:
        CString m_id;
        CTime m_time;
        CLogData(){};
        CLogData(const
CString&id,CTimetime =CTime::GetCurrentTime()){m_id=id,m_time=time;}
};

class CLogArray : public CObArray
{
public:
        CString m_id;
        CTime m_time;
        void RemoveAll();
        int Add(CLogData* pData);
        void Load();
        void Save();
};


class CLogView : public CView
{
protected: // シリアライズ機能のみから作成
        CLogView();
        DECLARE_DYNCREATE(CLogView)

// アトリビュート
public:
        CLogDoc* GetDocument();

// オペレーション
public:

// オーバーライド
        // ClassWizard は仮想関数のオーバーライドを生成
        //{{AFX_VIRTUAL(CLogView)
        public:
        virtual void OnDraw(CDC* pDC);  // このビューを描画する際にオーバーラ
                                                イド
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        protected:
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
        //}}AFX_VIRTUAL
```

```
// インプリメンテーション
public:
        CLogArray m_logAr;
        virtual ~CLogView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// 生成されたメッセージ マップ関数
protected:
        //{{AFX_MSG(CLogView)
        afx_msg void OnMlogin();
        afx_msg void OnMlogout();
        afx_msg void OnTimer(UINT nIDEvent);
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // LogView.cpp ファイルがデバッグ環境の時使用
inline CLogDoc* CLogView::GetDocument()
   { return (CLogDoc*)m_pDocument; }
#endif
```

＜MainFrm.h : CMainFrame クラスの宣言およびインターフェイスの定義＞

```
class CMainFrame : public CFrameWnd
{
protected: // シリアライズ機能のみから作成
        CMainFrame();
        DECLARE_DYNCREATE(CMainFrame)

// アトリビュート
public:

// オペレーション
public:

// オーバーライド
        // ClassWizard は仮想関数のオーバーライドを生成
        //{{AFX_VIRTUAL(CMainFrame)
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        //}}AFX_VIRTUAL

// インプリメンテーション
public:
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // コントロール バー用メンバ
        CStatusBar    m_wndStatusBar;
        CToolBar      m_wndToolBar;

// 生成されたメッセージ マップ関数
protected:
        //{{AFX_MSG(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
                // メモ - ClassWizard はこの位置にメンバ関数を追加または削除
                //        この位置に生成されるコードを編集禁止
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

＜CLogView クラスの構築/消滅＞

```
CLogView::CLogView()
{
        // TODO: この場所に構築用のコードを
}

CLogView::~CLogView()
{
}

BOOL CLogView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: この位置で CREATESTRUCT cs を修正して Window クラスまたは
    スタイルを修正する
    //ログデータの読み込み
    m_logAr.Load();

    return CView::PreCreateWindow(cs);
}
```

＜時計の表示＞

```
void CLogView::OnDraw(CDC* pDC)
{
	CLogDoc* pDoc = GetDocument();
	ASSERT_VALID(pDoc);
	// TODO: この場所にネイティブ データ用の描画コードを追加します。
	CFont *oldFont,newFont;
	CTime ct;
	CString cs;

	//新しいフォントを設定する
	newFont.CreatePointFont(500,"Times New Roman");
	oldFont = pDC->SelectObject(&newFont);

	//現在時刻の取得
	ct = ct.GetCurrentTime();
	cs = ct.Format("%H:%M:%S");

	//現在時刻の表示
	pDC->TextOut(250,250,cs);

	//元のフォントに戻す
	newFont.DeleteObject();
	pDC->SelectObject(oldFont);

	//タイマーをセットします
	static BOOL fTimer = FALSE;
	if (fTimer == FALSE)
	{
		SetTimer(123,500,NULL);
		fTimer = TRUE;
	}
}
```

＜CLogView クラスのメッセージ ハンドラ＞

```
void CLogView::OnMlogin()
{
	// TODO: この位置にコマンド ハンドラ用のコードを追加する
	//Static を使用することで cld は静的オブジェクトと宣言されている
	static CLoginDialog cld;

	int iRet;

	//ダイアログボックスの表示
	iRet = cld.DoModal();

	//パスワードのチェック
	if (iRet == IDOK)
	{
		{
			CLogData* pData = new CLogData(cld.m_Id);
			m_logAr.Add(pData);
			m_logAr.Save();
		}

		if(cld.m_Id !=cld.m_Passwd)
		{
			AfxMessageBox("学籍番号またはパスワードが違います！");
			cld.m_Id ="";
			cld.m_Passwd ="";
		}
		else //認証されたとき
		{
			AfxMessageBox("入室します！");
			cld.m_Id ="";
			cld.m_Passwd ="";

		}
	}
}

void CLogView::OnMlogout()
{
	// TODO: この位置にコマンド ハンドラ用のコードを追加してください
	static CLogoutDialog cld;

	int iRet;

	//ダイアログボックスの表示
	iRet = cld.DoModal();
```

```
	//ここから先に List へ学籍番号が入る予定だった
	//でも実際には Debug 方面でエラーがでる
	//表示もされない
	CString str="1010319¥n 1020111¥n 1030111";
	 m_listBox.AddString(str);

}

void CLogView::OnTimer(UINT nIDEvent)
{
   // TODO: この位置にメッセージ ハンドラ用のコードを追加するかまたはデフォル
   トの処理を呼び出し
	if (nIDEvent == 123)
	{
		InvalidateRect(NULL);
	}

	CView::OnTimer(nIDEvent);
}

int CLogArray::Add(CLogData* pData)
{
	if(GetSize()>10)
	{
	delete GetAt(0); //まずメモリ領域を削除
	RemoveAt(0);	//次に、配列の上で削除。これでインデックスが一つずれる
	}
	return CObArray::Add((CObject*)pData);
}

void CLogArray::Save()
{
	CFile file;
	if(file.Open("log.log",CFile::modeCreate | CFile::modeWrite))
	{
		CArchive ar(&file,CArchive::store,4096);

			ar<<GetSize();
		for(int i=0;i<GetSize();i++)
		{
			CLogData* pData=(CLogData*)GetAt(i);
			ar<<pData->m_id;
			ar<<pData->m_time;
		}
		ar.Close();
		file.Close();
	}
	file.Abort();
```

```
	CStdioFile file2;
	if(file2.Open("log.txt",CFile::modeCreate | CFile::modeWrite))
	{
		for(int i=0;i<GetSize();i++)
		{
			CLogData* pData=(CLogData*)GetAt(i);
			CString str;

	str.Format("[%s]%s¥n",pData->m_id,pData->m_time.Format("%H:%M:%S"
));
			file2.WriteString(str);
		}
		file2.Close();
	}
	file2.Abort();

}

void CLogArray::Load()
{
	CFile file;
	if(file.Open("log.log",CFile::modeRead))
	{
		CArchive ar(&file,CArchive::load,4096);

		int num;
		ar >> num;
		for(int i=0;i<num;i++)
		{
			CLogData* pData=new CLogData;
			ar>>pData->m_id;
			ar>>pData->m_time;
			Add(pData);
		}
		ar.Close();
		file.Close();
	}
	file.Abort();

	CStdioFile file2;
	if(file2.Open("log.txt",CFile::modeCreate | CFile::modeWrite))
	{
		for(int i=0;i<GetSize();i++)
		{
			CLogData* pData=(CLogData*)GetAt(i);
			CString str;

	str.Format("[%s]%s¥n",pData->m_id,pData->m_time.Format("%H:%M:%S"
));
```

```
                        file2.WriteString(str);
                }
                file2.Close();
        }
        file2.Abort();
}

void CLogArray::RemoveAll()
{
        for(int i=0;i<GetSize();i++)
                delete GetAt(i);
        CObArray::RemoveAll();
}
```