情報工学実験C 実験レポート
クライアントサーバモデルを基にした
ネットワークプログラミング

氏名: 島谷　隼生 (Shimatani Toshiki)
学生番号: 09428526

出題日: 2018 年 12 月 04 日
提出日: 2019 年 01 月 29 日
締切日: 2019 年 01 月 29 日

1　クライアントサーバモデルとは

この章では，本実験の最終課題であるクライアントサーバモデルに基づくプログラムの作成にあたって，基礎となるクライアントサーバモデルの詳細を説明する．

1.1　クライアントサーバモデルとは

クライアント・サーバモデルとは，プログラムがそれぞれクライアント，サーバと呼ばれる 2 つの部分に機能を分割する，コンピュータネットワークにおけるソフトウェア構造の一つである．クライアントプログラムでは，他のサーバプログラムから提供されるサービスを使用する機能を，サーバプログラムでは，クライアントプログラムに対してサービスを提供する機能を持つ．

1.2　通信の仕組み

クライアントサーバモデルでは，機能を分割しているため，互いの処理の結果をネットワークを通じて交換する必要がある．そのため，クライアントプログラムとサーバプログラムでは，要求メッセージと応答メッセージという形でデータのやりとりを行っている．今回は TCP/IP 通信をメインにプログラムを作成するため，TCP/IP での送受信関数を利用してメッセージの交換を実現している．

1.3　クライアントとサーバのそれぞれにおける処理の流れ

以下に大まかな処理のフローチャートを示す．

1. 基本仕様の概要
   (a) 標準入力から「ID，氏名，生年月日，住所，備考」からなるコンマ区切り形式 (CSV 形式) の名簿データを受け付けて，それらをメモリ中に登録する機能を持つ．CSV 形式の例を以下に示す．

図 1: クライアントサーバのフローチャート

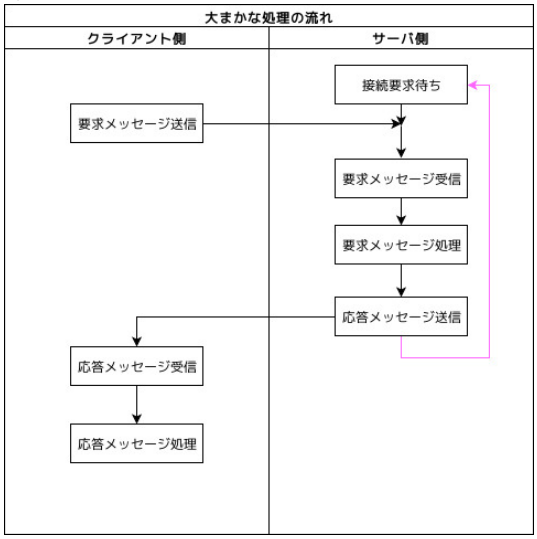

表 1: 実装するコマンド

コマンド	意味	備考
%Q(q)	プログラムの終了 (Quit)	
%C(c)	登録項目数の出力 (Check)	
%P(p) n	CSV の先頭から n 番目を抜き出して表示 (Print)	n:0 →全件,n:負→後から-n 件
%R(r) file	file から読込み (Read)	
%W(w) file	file から書出し (Write)	
%F(f) word	検索結果を表示 (Find)	%P と同じ形式で表示
%S n	CSV の n 番目の項目で整列 (Sort)	表示はしない

```
09428900,Takahashi Kazuyuki,1977-04-27,Saitama,male
09428901,Honma Mitsuru,1972-08-25,Hokkaidou,male
09428902,Ogura Shinsuke,1976-07-23,Kanagawa,male
09428903,Shibata saki,1968-03-16,Hyougo,female
:
```

(b) 標準入力から%で始まるコマンドを受け付けて，ファイルを入出力したり，登録してあるデータを操作する機能を持つ．実装する方針のコマンドを表 1 に示す．

2. 機能や性能に関わる要件

(a) 名簿データは配列などを用いて少なくとも 10000 件のデータを登録できるようにする．今回のプログラムでは，構造体 struct profile の配列 profile_data_store[] を宣言して，10000 件のデータを格納できるようにする．

(b) 名簿データは構造体 struct profile および構造体 struct data を利用して，構造を持ったデータとしてプログラム中に定義して利用する．実装すべきデータ構造は表 2 である．表中の n bytes とは，n バイトの char 型配列を意味する．

(c) コマンドの実行結果出力に不必要なエラーや警告は標準エラー出力に出力する．

また，本レポートでは以下の考察課題について考察をおこなった．

表 2: 名簿データ

ID	氏名	生年月日	住所	備考データ
32 bit 整数	70 bytes	`struct date`	70 bytes	任意長

1. 不足機能についての考察
2. エラー処理についての考察
3. 新規コマンドの実装
4. 既存コマンドの改良
5. 構造体のサイズ
6. 本課題の要件に対する考察
7. コマンドの拡張
8. テキスト形式とバイナリ形式

2 プログラムの作成方針

クライアントプログラムとサーバプログラムのそれぞれにおける作成方針を以下に示す.

2.1 クライアントプログラム

クライアントプログラムは，おおよそ以下の部分から構成することにした．それぞれについて作成方針を立てる.

1. プロセス間通信の前処理部（2.1.1 節）
2. 要求メッセージ送信部（2.1.2 節）
3. 応答メッセージ受信部（2.1.3 節）
4. クライアント側コマンド処理部（2.1.4 節）

2.1.1 プロセス間通信の前処理部

“前処理部” はクライアントサーバ間でメッセージのやりとりを行うにあたり必要な処理を行う部分である．本実験で作成した名簿管理プログラムの

```
struct date {
  int y;
  int m;
  int d;
};

struct profile {
  int id;
  char name[70];
  struct date birthday;
  char home[70];
  char *commment;
};

struct profile_data_store[10000];
```

ここで，コメントについてはポインタを用いて宣言している．これによって，長さを任意長にすることができる．

2.1.2 解析部

“解析部” は標準入力から得られたデータの読み込みを行う箇所である．しかし，このままでは，わかりにくい．そこで，段階的詳細化の考え方に基づいてさらなる詳細化をおこない，下記の (a) から (e) のように分割することにする．

(a) EOF が来るまで，標準入力から文字の配列 `char line[]` に 1 行分を読み込む．

(b) `line` の 1 文字目が，’%’ ならば，2 文字目をコマンド名，4 文字目以降をその引数として，決定されたコマンドを実行する．

(c) さもなくば `line` を CSV とみなし’,’ を区切りとして 5 つの文字列に分割する．

(d) 分割してできた 5 つの文字列を変換部に渡し構造体に代入する．

(e) 次の行を読み込む

ここで，(b) は “コマンド文字に応じて分岐” と詳細化することもできるが，今回はそのままとしておく．また，(d) で扱う文字列は char 型配列として処理するため，解析部に続く変換では型変換に注意する必要がある．

2.1.3 変換部

“変換部” は分割された CSV データを項目毎に型変換し，対応する構造体メンバに保存する部分である．メンバに様々な型を用いているため，適切な代入の使い分けが必要となる．

文字列は関数 `new_profile` を用いて代入する．数値の場合，関数 `atoi` を用いて文字列を数値化してから代入する．構造体 `struct date` であるメンバ `birthday` については更に分割してから代入する．

なお，構造体への代入については，関数 `split` を用いることで容易に実装することができる．例えば，`"2014-10-25"`のような文字列を読み込んで，’-’ にごとに分割しつつ格納するという処理は，csv データを分割しつつ格納する処理と同じ処理である．従って，区切り文字が CSV の’,’ とは異なる’-’ になること以外は同様に記述できるはずである．

2.1.4 各種コマンド 実現部

“各種コマンド実現部” は構造体の配列に格納したデータに対する処理をおこなう部分である．このレポートでは，具体的には `%Q(Quit),%C(Check),%P(Print),%R(Read),%W(Write),%F(Find),%S(Sort)`，その他の追加コマンドを実装している．また，コマンドの文字については，大文字と小文字の双方で反応する仕様としている．

終了 (`%Q`) は `exit` 関数を用いてプログラムを終了させればよい．

チェック (`%C`) は `new_profile` 関数中でカウントされた `profile_data_ntimes` を `printf` 関数で表示すればよい．

表示（`%P n`）は `printf` 関数で各項目毎に表示すればよい．ただし，`n`が負であるとき，後ろから`-n` 件表示することに注意が必要である．

読み込み (%R file) は get_line 関数を拡張して引数にファイルポインタを加えた上でで，main 関数中の処理にファイルの開閉処理を加えることで実装できる．このとき，拡張前後でプログラムに支障がないかテストが必要である．

書き込み (%W file) は%P コマンドに手を加えることで実装可能である．具体的には，%P コマンド中の範囲指定を取り除き，出力の仕方を CSV 形式に調整し，ファイルの開閉処理を加えればよい．

検索 (%F word) も%P コマンドに手を加えることで実装可能である．%W コマンドと同様に範囲指定は取り除くが，今度は入力 (word) された文字列と構造体メンバが一致するか調べる必要がある．文字列，数，構造体のそれぞれを入力文字列と比較するが，今回は数と構造体をともに文字列に変換する方針をとる．その詳細は実装過程における考察において述べる．

ソート (%S n) は今回，最も単純なバブルソートを用いることにする．引数の n の値によってソートの判定に用いる項目が異なるが，二要素間を入れ替えるかどうかを判定する compare_profile 関数中にて switch 文を用いることで解決できる．また，入れ替えを行うときはポインタで操作することに注意する．

その他の追加コマンドの方針等ついては 6.3 節にて説明する．

2.2 サーバプログラム

1. プロセス間通信の前処理部（2.1.1 節）
2. 要求メッセージ受信部（2.1.2 節）
3. 応答メッセージ送信部（2.1.3 節）
4. クライアント側コマンド処理部（2.1.4 節）

3 プログラムおよびその説明

プログラムリストは 8 節に添付している．プログラムは全部で 573 行からなる．以下では，前節の作成方針における分類に基づいて，プログラムの主な構造について説明する．

3.1 宣言部 (11 行目から 56 行目)

前節で示したように，宣言部ではプログラム中で使用する構造体や配列，加えてグローバル変数を宣言する．date 構造体や profile 構造体はデータに関する個別の情報をそれぞれ格納するために，配列 profile_data_store はデータ全体を格納するために用いる．グローバル変数 profile_data_nitems は現在までに何件のデータを格納しているかを各関数で共有するために用いる．save はデータをファイルに保存したかどうかを示す．詳しくは，5.4 節で説明する．また，今後使う関数も 28–56 行目でプロトタイプ宣言をしておく．

3.2 解析部 (56 行目から 175 行目)

49–55 行目は main 関数であり，作成方針で説明した解析部の動作におおよそ相当する．ただし (c) の 5 つの文字列に分割する部分は，解析部の main() 関数では実現せず，変換部である new_profile 関数中で split を呼出すことにしている．57–111 行目は main 関数内で用いる関数

とそれに関係する関数郡である．(a) は　get_line 関数, (b) は parse_line 関数，exec_command 関数によって実現している．

解析部の関数に用いているので，文字列操作関数 subst() 関数をこの部分に含めている．subst() 関数は 78–96 行目で宣言している．

subst は，str が指す文字列中の c1 文字を c2 に置き換える．プログラム中では，入力文字列中の末尾に付く改行文字をヌル文字で置き換えるために使用している．

3.3　変換部 (177 行目から 240 行目)

177–223 行目は変換部の主となる new_profile 関数である．225–237 行目は split 関数である．解析部ではなく変換部で呼び出した理由については，考察にて後述する．

split は str が指す文字列を区切文字 c で分割し，分割した各々の文字列を指す複数のポインタからなる配列を返す関数である．プログラム中では，CSV を ’,’ で分割し，分割後の各文字列を返すのに使用されている．また，“2004-05-10” のような日付を表す文字列を ‘-’ で分割して，struct date を生成する際にも使用している．

3.4　コマンド 実現部 (242 行目から 576 行目)

242 行目–469 行目は各種コマンド関数であり，472 行目–576 行目は各種コマンド関数に用いる関数郡である．できる限り一つの関数に複数の処理が重ならないように意識した．特にソートに関する関数は条件分岐が多くなったので，条件部分も関数として作成した．コマンドに関する説明は次節にて行う．

4　プログラムの使用法

本プログラムは名簿データを管理するためのプログラムである．CSV 形式のデータと % で始まるコマンドを標準入力から受け付け，処理結果を標準出力または%W(w) コマンド ,%BW(bw) コマンドで指定されたファイルに出力する．入力形式の詳細については，概要の節を参照のこと．

プログラムは，一般的な UNIX で用いることを意図している　gcc でコンパイルした後，標準入力からファイルを与える，もしくは手入力で CSV 形式でデータを入力していく．

```
\$ gcc -o program1 program1.c
\$ ./program1 < csvdata.csv

\$ gcc -o program1 program1.c
\$ ./program1
\$ 09428900,Takahashi Kazuyuki,1977-04-27,3,Saitama,male
```

プログラムの出力結果としては CSV データの各項目を読みやすい形式で出力する．例えば，下記の cvsdata.csv に対して，

```
09428900,Takahashi Kazuyuki,1977-04-27,Saitama,male
09428901,Honma Mitsuru,1972-08-25,Hokkaidou,male
%C
%P
%P 1
%Q
```

以下のような出力を得る.

```
  2 profile(s)

  Id    : 94285900
  Name  : Takahashi Kazuyuki
  Birth : 1977-04-27
  Addr  : Saitama
  Com.  : male

  Id    : 94285901
  Name  : Honma Mitsuru
  Birth : 1972-08-25
  Addr  : Hokkaido
  Com.  : male

  Id    : 94285900
  Name  : Takahashi Kazuyuki
  Birth : 1977-04-27
  Addr  : Saitama
  Com.  : male
```

入力された%Cは，これまでの入力データが何件登録されたかということを示し，%P は 入力したデータを全件表示することを示している．また，%P 1は入力したデータを先頭から 1 件 (負の数だと後ろから) 表示することを示し，%Qはプログラムを終了することを示す.

上の例では%Q，%C，%P コマンドの説明を行った．以下では基本方針の残りの%R，%W，%F，%S コマンドの説明を行う．以下のような sample.csv,sample2.csv に対して,

```
(sample.csv)
   %R sample2.csv
(sample2.csv)
   %C
   09428900,Takahashi Kazuyuki,1977-04-27,Saitama,male
   09428901,Honma Mitsuru,1972-08-25,Hokkaidou,male
   %C
   09428902,Nakamura Hiroki,1975-09-04,Nagano,male
   %W sample.csv
   %S 3
   %P
   %F Nagano
```

以下のような出力を得る.

```
   0 profile(s)
   2 profile(s)

   Id    : 94285901
   Name  : Honma Mitsuru
   Birth : 1972-08-25
   Addr  : Hokkaido
   Com.  : male

   Id    : 94285902
   Name  : Nakamura Hiroki
   Birth : 1975-09-04
   Addr  : Nagano
   Com.  : male

   Id    : 94285900
```

```
Name  : Takahashi Kazuyuki
Birth : 1977-04-27
Addr  : Saitama
Com.  : male

Id    : 94285902
Name  : Nakamura Hiroki
Birth : 1975-09-04
Addr  : Nagano
Com.  : male
```

出力後，sample.csv は以下のように書き換えられている．

```
09428900,Takahashi Kazuyuki,1977-04-27,Saitama,male
09428901,Honma Mitsuru,1972-08-25,Hokkaidou,male
09428902,Nakamura Hiroki,1975-09-04,Nagano,male
```

入力された`%R sample2.csv` は指定されたファイル (この場合，sample2.csv) を読み込むことを示し，`%W sample.csv` は指定されたファイル (この場合，sample.csv) に書き込みをすることを示す．また，`%S 3` はこれまでの入力データを 3 番目の項目 (生年月日) でソート (バブルソート) することを示しており，`%F Nagano` は入力されたワード (この場合，Nagano) と一致する項目をもつデータを表示する．これら以外にもコマンドはあるが，それらについては 6.3 節で説明する．

5 作成過程における考察

2 節で述べた実装方針に基づいて，3 節ではその実装をおこなった．しかし，実装にあたっては実装方針の再検討が必要になる場合があった．本節では，名簿管理プログラムの作成過程において検討した内容，および，考察した内容について述べる．

5.1 宣言部についての考察

宣言部については，概ね方針通りに実装することができた．また，3 節でも述べたようにグローバル変数を使用した．これは，グローバル変数を使用することで処理するデータを関数間で共有することが容易にできるためである．ただ，グローバル変数は他人がプログラムに変更を加えるときの障害になる可能性を持つことを留意しておく必要がある．今回は，自分以外の人が手を加えることを想定していないため，遠慮なく用いている．また，`profile` 構造体のメンバ `id` に 8 桁の制限をかけている．これは 9 桁となると int 型の範囲を超えてしまうためである．実際に制限をかけているプログラムは変換部の `new_profile` 関数中にて実装している．

5.2 解析部についての考察

解析部の作成方針として読み込んだ行がコマンドでなければ 5 つの文字列に分解するという方針としたため，分解後の文字列が数字であろうとなかろうと，char 型配列であることに注意が必要である．なぜなら宣言部にて宣言した構造体のメンバに，int 型のメンバがものが含まれているため，続く変換部にて構造体のそのまま代入ができない．これを解決するにはキャストすることも考えられるが，今回は `atoi` 関数を用いることで対処した．なぜならば，キャストした文字列は数字であることを想定しているが，もし文字であった場合の挙動が予測できないので，文字であった場合の動作が保証されている `atoi` 関数を選択した．

5.3 変換部についての考察

変換部についても，概ね方針通りに実装することができた．だが，入力された CSV 形式のデータの要素数 (分割数) が少ない場合，その入力を不正なものとして処理して無効な入力としている．これについては，後のエラー処理に関する考察で説明する．また，date 構造体に格納するときの分割処理においては，要素数 (分割数) が規定の値 (今回は年，月，日のため 3) 出ない場合，同様のエラー処理を行っている．このため，正しい入力の仕方をしなければ，延々とエラー出力が続くようになっている．そのため，標準エラー出力によって正しい入力の仕方を表示するようにしている．また，変換部にて文字列分割処理を行うこととした理由は，このような方針とすることで new_profile 関数に渡す引数が，１つの文字列のみでよくなり，関数内の処理を行いやすくできるためである．

5.4 コマンド 実現部についての考察

コマンド 実現部についても，概ね方針通りに実装した．%F コマンドについては入力された文字列と構造体メンバが一致しているかどうかの判定に strcmp 関数を使用しているため，部分一致の場合は検索に引っかからない．今回はサンプルプログラムと同様の挙動を意識したため，このような仕様となっているが，部分一致の場合も反応するためには独自の比較関数を作成して，比較する実装方針とするほうがよいだろう．また，%F コマンドの比較操作部分で登録されているデータのメンバをすべて文字列へと変換する方針とした理由は，入力されたデータを変化させることなく続けて各メンバとの比較を行うことができるからである．また，入力データを各メンバに合わせて変換する方針とした場合，入力されたデータがどのメンバと比較したいのかを判断できなければ型変換が困難となることも理由の一つである．また，%Q コマンドについてであるが，データをファイルに保存 (書き込み) する前に，誤って終了してしまう可能性も考えられる．そのため，ファイルに書き込むコマンドを使用していない場合，本当にプログラムを終了してよいかどうかを確認するようにしている．グローバル変数で宣言した save はファイルに書き込むコマンドを使用したかどうかを 0,1 で判断するために使用している．

6 結果に関する考察

演習課題のプログラムについて仕様と要件をいずれも満たしていることをプログラムの説明および使用法における実行結果例によって示した．ここでは，概要で挙げた以下の項目について考察を述べる．

1. 不足機能についての考察
2. エラー処理についての考察
3. 新規コマンドの実装
4. 既存コマンドの改良
5. 構造体のサイズ
6. 本課題の要件に対する考察
7. コマンドの拡張
8. テキスト形式とバイナリ形式

6.1 不足機能についての考察

考えられる不足機能としては，部分一致でも反応する新たな%F コマンド，登録データの削除機能，修正機能，実装してあるコマンドを説明する機能などが考えられる．それぞれの理由を説明する．まず，部分一致でも反応する新たな%F コマンドについてだが，今回実装した%F コマンドでは完全一致させる必要があり，不便である．また，探したいデータについての情報が不確かな場合に見つけることが困難になるという問題点もある．次に，登録データの削除機能についてである．本プログラムでは，不正な入力は登録されないような仕様としたが，正しい入力形式で入力されたデータの要素に間違いがあった場合はファイル書き出しを行い，プログラム終了後に emacs などを用いて書き出したファイルの編集を行って対処しなければならない．これでは大変不便である．そのため，プログラム実行中に削除できる機能が必要であると考えられる．修正機能についても同様である．また，削除，修正するデータを指定するために，該当するデータが先頭から何番目かを調べる機能も必要であると考えられる．実装してあるコマンドを説明する機能は，いわゆるヘルプ機能である．プログラムを用いる人がプログラムの使用方法を完全に知っている可能性は高くないため，それを補う必要がある．そのための機能である．

6.2 エラー処理についての考察

6.2.1 CSV データ処理中のエラー処理

CSV データ中に，不正なデータが含まれていた場合の処理について考察する．エラーが含まれていた場合は，以下のような対処が考えられる．

(1) エラーのあった行を指摘して，無視する
この方法は，一回の入力で，できるだけ多くのエラーを発見できるため，通常はこの方法が好ましい．しかし，エラーのあった状態からの復帰を行う必要があるためプログラムが複雑になる．

(2) エラーのあった行を指摘して，終了する
この方法は，入力中に 1 つのエラーを発見することしかできない．しかし，エラーのあった入力をデータを無視してしまうと以降のデータ入力の正当性チェックにも影響がでるような場合には，この方法を採らざるを得ないこともある．

エラーのあった行を指摘せず，終了または無視するという方法も考えられるが，正常終了との区別が付かないため実用的でない．

今回は，エラーのあった行を指摘して，無視する方法がよいと考えた．理由は，ファイル読み込みなどで大量の CSV データを読み込む際に，一つのエラー入力でそれ以降の入力が中止されてしまうと，データの登録にかかる時間が大幅に増えてしまう．そうであるならば，誤った入力を無視してしまったほうが効率は良くなる．ただ，その場合はファイル読み込みの際にエラーが何行目で発生したかを表示させるべきと考えられる．

6.2.2 データを保存する前にプログラムを終了してしまう際のエラー処理

ユーザが登録した名簿データをファイルに保存 (%W コマンドを使用) する前にプログラムを終了してしまう可能性は十分に考えられる．その対処としては，保存を行っていない場合でプログラムを終了しようとする際に，%W コマンド，%BW を使用していないという警告をだすことや，一時

保存ファイルを作成することが考えられる．どちらが適しているか考えると，警告を出してユーザに確認をとる方がユーザに気づかせることもできることから，前者の方がてきしているだろう．今回は%Q コマンドでそれを実装している．

6.3 新規コマンドの実装

6.1 節を踏まえて，新規コマンドとして登録されたデータを修正をコマンド%M n を実装した．これはメモリ中のデータを指定し，CSV 形式でデータを入力し直すことでデータの修正を行うものである．引数 n は先頭から n 番目のデータを修正することを意味している．n が登録件数よりも大きい場合，または負の数である場合は，エラー出力を行う．また，登録されたデータを削除するコマンド%D n も実装した．引数に関する説明や仕様は%M n コマンドと同様である．また，登録されているデータが先頭から何番目かを調べる機能については，%F コマンド中の処理でカウントされているためそれを利用して新たに%f コマンドを実装した．機能は%F コマンドと全く変わらないが，検索したデータが先頭から何番目かを表示する機能を追加している．また，詳細は 6.8 節にて述べるが，データをバイナリ形式でファイルに入出力するコマンド%BW word,%BR word も実装している．

6.4 既存コマンドの改良

%W コマンドは利用する際に毎回ファイル名を指定する必要があり，不便である．改良案として，同じファイルに続けて書き込みを行う際には引数に'/' を用いることで，同じファイル名を繰り返し入力することを避ける方法が挙げられる．'/' とした理由は，'/' は linux においてファイル名として使用できない文字であるから，ファイル名を引数とする%W コマンドに適していると考えたからである．また%S コマンドのソートの方法として，本プログラムではバブルソートを用いているが，この方法は効率が悪いと言われている．よって，より効率化をはかるためには，クイックソートを用いることが挙げられる．さらに，クイックソートの軸を選ぶ際にいくつかの要素（３つ程度）をとり，その平均を軸としたり，ソートが行われていない残りの要素数が少なくなったときに挿入法に切り替えることでさらなる効率化をはかることができる．今回はそういった工夫は行わなかったが，ソートコマンドの改良として，クイックソートを実装した．%S と大文字としたときはバブルソートを，%s と小文字にしたときはクイックソートを実行するようにしている．また，5.4 節でも述べたが，%Q コマンドに関する改良 (ファイルを書き込みしていないときの確認) を行っている．

6.5 構造体のサイズ

以下のプログラムをｇｃｃでコンパイルし，実行した．

```
 struct profile a;
printf("size1..%d\n",sizeof(a));
printf("size2..%d\n",sizeof(a.id));
printf("size3..%d\n",sizeof(a.name));
printf("size4..%d\n",sizeof(a.birthday));
printf("size5..%d\n",sizeof(a.home));
printf("size6..%d\n",sizeof(a.comment));
printf("1..%p\n",&a);
printf("2..%p\n",&a.id);
printf("3..%p\n",&a.name);
printf("4..%p\n",&a.birthday);
```

```
printf("5..%p\n",&a.home);
printf("6..%p\n",&a.comment);
```

以下の結果を得た.

```
size1..164
size2..4
size3..70
size4..12
size5..70
size6..4
1..0xbfeff84c
2..0xbfeff84c
3..0xbfeff850
4..0xbfeff898
5..0xbfeff8a4
6..0xbfeff8ec
```

これを見ると構造体のメンバのサイズの合計が 160，構造体のサイズが 164 となっており一致していない．それぞれのアドレスを確認してみると，メンバ name とメンバ birthday，メンバ home とメンバ comment の間に隙間が空いていることが確認できる．これはハードウェア (CPU) の都合によって，型によっては配置できるアドレスに制限があることや，もしくは配置することができても効率が悪くなるような CPU があり．そのような場合に，コンパイラが適当に境界調整 (アラインメント) 行い，構造体に適切なパティング (詰め物) が挿入されるからである．また，アラインメントが構造体の末尾に入ることもあり、そのような場合は構造体に sizeof 演算子を適用すると，末尾のアラインメントを含めたサイズが返される．このため，構造体のメンバのサイズの合計と構造体のサイズが異なるのであろうと考えられる．

6.6 本課題の要件に対する考察

本課題の要件のうち，コマンドが 1 文字であるのは，直感的ではない上に，コマンドの数が増えた場合の拡張性に乏しい．また，コマンドの後のスペースが 1 個のみしか許されないというのは制限が強い．しかしプログラムの作成が容易になるというメリットがある．また，課題ではコマンドはすべて大文字で記載されていたが，それにこだわる必要性は薄いと考えたため，小文字でも，大文字と同様の機能を行える仕様としている．また，学校名や所在地が 70byte までとした理由が不明である，10000 件ものデータを格納できるようにと定めているのであるから，データは節約すべきだろう．少なくとも学校名は 40–50byte でも問題はないだろう．

6.7 コマンドの拡張

1 文字だけではなく，2 文字のコマンドを受け付ける機能を実装するには，まず，1 文字コマンドか，2 文字コマンドのどちらかを判断するが必要がある．これを実装するには 2 つの方法が考えられる．1 つ目は読み込んだ入力文字列を `split` 関数を用いて，分解し，分解された文字列の最初の文字列の文字数で 1 文字か 2 文字を判定する方法，2 つ目は，読み込んだ入力文字列の 2 文字目と 4 文字目以降の文字列だけではなく，3 文字目も読み込み，それが空白文字またはナル文字か否かで判定する方法である．今回は，後者を選択した．理由としては，後者の方がより少ない処理でコマンドの文字数判定が行えるからである．

実装の手順は，まず exec_command 関数へ渡す引数を拡張して 3 文字目を渡すようにし，条件文を用いて，3 文字目が空白かナル文字ならば 1 文字コマンド，それ以外ならば 2 文字コマンドを実行するようにする．2 文字コマンドを実行する手段としては，switch 文を二重で使った．これにより，switch 文を同様に増やすことで新たな 2 文字コマンドが実装しやすくなっている．その代わり，exec_command 関数が長くなるという欠点がある．また，1 文字コマンドのときは，4 文字目以降の文字列を引数として渡していたが，2 文字コマンドの場合は +1 をして，5 文字目以降の文字列を渡す必要があることに注意する．

実際に動作するかの確認として%R コマンドと同じ機能を持つ%TR コマンドを実装した．結果は問題なく動作した．また，このコマンドは%R コマンドを流用しているので，新規コマンドには加えていない．

6.8 テキスト形式とバイナリ形式

バイナリ形式とは，テキスト形式と対比される用語で，CPU がそのまま扱えるデータ形式のことである．また，これはテキスト形式と異なり，人間がそのまま読めない形式となっている．さらに，バイナリ形式でデータを保存する場合，効率が良くなる場合がある．例えば，’112’ というデータを扱う場合，バイナリ形式で扱えば’70’ の 1 バイト分で済むこととなる．ただ，これは整数を多く取り扱う場合に限ってであり，可変長である文字列が多い場合，テキスト形式の方が効率がよくなる．今回は，可変長である文字列は comment のみであるため，バイナリ形式で効率がよくなる可能性は小さくない．

c 言語を用いてデータをバイナリ形式でファイルに入出力するに当たって，使用できる関数は fwrite 関数，fread 関数の 2 つのみである．今回は，この 2 つを主として実装を行っていった．実装していく中で大きな問題点となったことが，comment がポインタであることである．ポインタを含む構造体を出力し，次に読み込んだ時，そのポインタが利用できないからである．もし利用しようとすれば，すぐに SEGV となる．そのため，利用する際はポインタを初期化しなければならず，その際ポインタ内のデータは読み込めない．そのため，可変長文字列をバイナリ形式のファイルを通じてやりとりすることはできないだろう (もしかしたらそれを実現する方法があるかも知れないが，見つけ出すこと，考え出すことはできなかった)．そのため，comment を要素数の決まった (今回は講義で配布された sample.csv を参考に要素数を 100 とした) 配列を用いて，strncpy 関数を用いてバイナリ形式ファイルに出力した．そのため，comment が 100 文字に制限されてしまっている．また，for 文を用いてループさせて，すべてのデータを出力させているが，ループごとに配列を初期化する必要がある．そうしなければ，前回のループ時の要素数が今回のループ時の要素数より多かった場合に，不正な値が出力されてしまう．バイナリ形式ファイルから入力を得るには，出力と逆の手順をすればよい．

7 作成したプログラム

作成したプログラムを以下に添付する．与えられた課題については，節で示したようにすべて正常に動作したことを付記しておく．

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #define MAX_LINE_LEN 1024 /*1 行に読み込める最大文字数*/
6  #define MAX_STR_LEN 70 /*構造体メンバ name と home の最大文字数*/
7  #define MAX 5 /*split 関数での最大分割数*/
```

```
 8  #define PDS profile_data_store
 9  #define PDN profile_data_nitems
10
11  struct date {
12    int y;
13    int m;
14    int d;
15  };
16
17  struct profile {
18    int id;
19    char name[70];
20    struct date birthday;
21    char home[70];
22    char *comment;
23  };
24
25  struct profile profile_data_store[10000]; /*10000 件のデータまで登録可能*/
26  int profile_data_nitems = 0; /*登録したデータ数*/
27  int save = 0; /* データを保存したかどうかの判定に使用 */
28  /*************************************************************************/
29  int get_line(FILE *fp,char *line);
30  int subst(char *str, char c1, char c2);
31  void parse_line(char *line);
32  void exec_command(char cmd1,char cmd2, char *param);
33  void new_profile(char *line);
34  int split(char *str, char *ret[], char sep, int max);
35  void cmd_quit();                 /************************************/
36  void cmd_check();
37  void cmd_print(int n);
38  void cmd_read(char *file);
39  void cmd_write(char *file);      /*             コマンド関数           */
40  void cmd_find(char *word);
41  void cmd_bsort(int n);
42  void cmd_qsort(int n);
43  void cmd_modify(int n);
44  void cmd_delete(int n);
45  void cmd_find2(char *word);
46  void cmd_Bwrite(char *line);
47  void cmd_Bread(char *line);     /************************************/
48  void print_profile(struct profile *p);
49  void fprint_profile_csv(FILE *fp, struct profile *p);
50  char *date_to_string(char buf[], struct date *date);
51  int comparsion (char *word, int i);/*        コマンド関数に用いる関数     */
52  void b_sort(struct profile *p, int left, int right, int column);
53  int compare_profile(struct profile *p1, struct profile *p2, int column);
54  void swap (struct profile *a, struct profile *b);
55  void q_sort(struct profile *p, int left, int right, int column);
56  /*************************************************************************/
57  int main() {
58    char line[MAX_LINE_LEN + 1];
59    while (get_line(stdin, line)) {
60      parse_line(line);
61     }
62    cmd_quit();
63    return 0;
64  }
65
66  int get_line(FILE *fp, char *line)
67  {
68    if (fgets(line, MAX_LINE_LEN + 1, fp) == '\0')
69      {
```

```
70        return 0;
71      }
72
73    subst(line, '\n', '\0');
74
75    return 1;
76  }
77
78  int subst(char *str, char c1, char c2) {
79    char *s;
80    int i, x, n;
81
82    s = str;
83    while (*s != '\0') {
84      s++;
85    }
86    x = s - str; /* strの文字数 */
87    s = str;
88    n=0; /*入れ替えを行った回数 */
89    for(i=0;i<x+1;i++) {
90      if (*(s+i)== c1) {
91        *(s+i) = c2;
92        n++;
93      }
94    }
95    return n;
96  }
97
98  void parse_line(char *line) {
99    if (*line == '%') {
100     exec_command(line[1], line[2], &line[3]);
101   } else {
102     if (profile_data_nitems < 10000 ) {
103       new_profile(line) ;
104     }
105   }
106 }
107
108 void exec_command(char cmd1, char cmd2, char *param) {
109
110   if(cmd2 == ' ' || cmd2 == '\0') {
111     /************************** 1文字コマンド **************************/
112     switch (cmd1) {
113     case 'Q' :
114     case 'q' : cmd_quit();  break;
115     case 'C' :
116     case 'c' : cmd_check(); break;
117     case 'P' :
118     case 'p' : cmd_print(atoi(param)); break;
119     case 'R' :
120     case 'r' : cmd_read(param);  break;
121     case 'W' :
122     case 'w' : cmd_write(param); break;
123     case 'F' : cmd_find(param); break;
124     case 'f' : cmd_find2(param);  break;
125     case 'S' : cmd_bsort(atoi(param));  break;
126     case 's' : cmd_qsort(atoi(param));  break;
127     case 'M' :
128     case 'm' : cmd_modify(atoi(param)); break;
129     case 'D' :
130     case 'd' : cmd_delete(atoi(param)); break;
131     default:
```

```
132       fprintf(stderr, "%%%c command is undefined \n", cmd1);
133       break;
134     }
135     /*********************************************************************/
136   } else
137     /************************** 2文字コマンド ***************************/
138     switch (cmd1) {
139     case 'T' :
140     case 't' :
141       switch (cmd2) {
142       case 'R' :
143       case 'r' : cmd_read(param+1); break;
144       default:
145         fprintf(stderr, "%%%c%c command is undefined \n", cmd1, cmd2);
146       break;
147       } break;
148     case 'B' :
149     case 'b' :
150       switch (cmd2) {
151       case 'R' :
152       case 'r' : cmd_Bread(param+1); break;
153       case 'W' :
154       case 'w' : cmd_Bwrite(param+1); break;
155       case 'S' :
156       case 's' : cmd_bsort(atoi(param+1));  break;
157       default:
158         fprintf(stderr, "%%%c%c command is undefined \n", cmd1, cmd2);
159         break;
160       } break;
161     case 'Q' :
162     case 'q' :
163       switch (cmd2) {
164       case 'S' :
165       case 's' : cmd_qsort(atoi(param+1)); break;
166       default:
167         fprintf(stderr, "%%%c%c command is undefined \n", cmd1, cmd2);
168       break;
169       } break;
170     default:
171       fprintf(stderr, "%%%c%c command is undefined \n", cmd1, cmd2);
172       break;
173     }
174     /*********************************************************************/
175 }
176
177 void new_profile(char *line) {
178
179   char *ret[5];
180   char *ret2[3];
181   int cnt1, cnt2;
182
183   cnt1 = split(line, ret, ',', MAX);
184   if (cnt1 == 5){
185     if (strlen(ret[0]) > 8) {
186       fprintf(stderr,"ID max digits are 8\n");
187     }
188     PDS[PDN].id = atoi(ret[0]);
189     strncpy(PDS[PDN].name, ret[1], MAX_STR_LEN);
190     PDS[PDN].name[MAX_STR_LEN] = '\0';
191     cnt2 = split(ret[2], ret2, '-', 3);
192     if (cnt2 == 3) {
193       PDS[PDN].birthday.y = atoi(ret2[0]);
```

```
194       PDS[PDN].birthday.m = atoi(ret2[1]);
195       PDS[PDN].birthday.d = atoi(ret2[2]);
196       if (PDS[PDN].birthday.y > 9999 ||
197           PDS[PDN].birthday.y < 0    ||
198           PDS[PDN].birthday.m > 12   ||
199           PDS[PDN].birthday.m < 1    ||
200           PDS[PDN].birthday.d < 1    ||
201           PDS[PDN].birthday.d > 31) {
202         fprintf(stderr,"inputed birthday data is inappropriate\n");
203         cnt2 = 0;
204       }
205     } else if (cnt2 == 2 || cnt2 == 1) {
206       fprintf(stderr,"inputed birthday data is inappropriate\n");
207     }
208     strncpy(PDS[PDN].home, ret[3], MAX_STR_LEN);
209     PDS[PDN].home[MAX_STR_LEN] ='\0';
210     PDS[PDN].comment = (char*)malloc(sizeof(char*)*(strlen(ret[4])+1));
211     strcpy(PDS[PDN].comment, ret[4]);
212     profile_data_nitems++;
213     if (cnt2 != 3 || strlen(ret[0]) > 8) {
214       profile_data_nitems--;
215       if (cnt2 == 2 || cnt2 == 1 ) {
216       fprintf(stderr,"correct birthday form example: 1000-10-10\n");
217       }
218     }
219   } else {
220     fprintf(stderr,"error: this input is wrong form\n");
221     fprintf(stderr,"correct form : (ID),(name),(birthday),(home),(comment)\n");
222   }
223 }
224
225 int split(char *str, char *ret[], char sep, int max)
226 {
227   int cnt = 0;
228
229   *(ret + (cnt++)) = str;
230
231   while (*str && cnt < max) {
232     if (*str == sep){
233       *str = '\0';
234       *(ret + (cnt++)) = str + 1;
235     }
236     str++;
237   }
238
239   return cnt;
240 }
241
242 void cmd_quit() {
243   int c;
244   if (save == 0) {
245     printf("you don't save date\n");
246     printf("May I end this program? y or n\n");
247     while((c = getchar()) == 'y' ||(c = getchar()) == 'n'){
248       if (c = 'y') {
249       exit(0);
250       } else if (c = 'n') {
251     } else
252       printf("inputed character is wrong\n");
253     }
254   } else
255     exit(0);
```

```
256
257
258 }
259
260 void cmd_check() {
261   printf("%d profile(s)\n",profile_data_nitems);
262 }
263
264 void cmd_print(int n) {
265   int i;
266   if (n > 0) {
267     if ( n > profile_data_nitems ) {
268       n = profile_data_nitems;
269     }
270     for( i = 0 ; i < n ; i++) {
271       print_profile(PDS + i);
272     }
273   } else if (n == 0) {
274     for( i = 0 ; i < profile_data_nitems ; i++) {
275       print_profile(PDS + i);
276     }
277   } else if (n < 0 ) {
278     if ( (-1) * n > profile_data_nitems) {
279       n = (-1) * profile_data_nitems;
280     }
281     for( i = profile_data_nitems + n ; i < profile_data_nitems ; i++) {
282       print_profile(PDS + i);
283     }
284   }
285 }
286
287 void cmd_read(char *file) {
288   FILE *fp;
289   char line[MAX_LINE_LEN + 1];
290
291   fp = fopen(file, "r");
292
293   if (fp == NULL) {
294     fprintf(stderr,"Could not open file: $s\n", file);
295   }
296   while (get_line(fp, line)) {
297     parse_line(line);
298   }
299
300   fclose(fp);
301 }
302
303 void cmd_write(char *file) {
304   int i;
305   FILE *fp;
306
307   fp = fopen(file, "w");
308
309   if (fp == NULL) {
310     fprintf(stderr,"Could not open file: $s\n", file);
311   }
312   for (i = 0; i < profile_data_nitems; i++) {
313     fprint_profile_csv(fp, PDS+i);
314   }
315   fclose(fp);
316
317   if (save == 0) {
```

```
318       save = 1;
319     }
320   }
321
322   void cmd_find(char *word) {
323     int i;
324     for (i = 0; i < profile_data_nitems; i++) {
325       if (comparsion(word, i) == 1) {
326         print_profile(&profile_data_store[i]);
327       }
328     }
329   }
330
331   void cmd_bsort(int n) {
332     if(1 <= n || n <= 5) {
333     b_sort(PDS, 0, profile_data_nitems - 1, n);
334     } else
335       fprintf(stderr,"this argument is wrong. plese input 1,2,3,4,5\n");
336   }
337
338   void cmd_qsort(int n) {
339     if(1 <= n || n <= 5) {
340     q_sort(PDS, 0, profile_data_nitems - 1, n);
341     } else
342       fprintf(stderr,"this argument is wrong. plese input 1,2,3,4,5\n");
343   }
344
345   void cmd_modify(int n) {
346     if (n-1 < PDN && n > 0) {
347     char line[MAX_LINE_LEN + 1];
348     get_line(stdin, line);
349
350     char *ret[5];
351     char *ret2[3];
352     int cnt1, cnt2;
353
354     cnt1 = split(line, ret, ',', MAX);
355     if (cnt1 == 5){
356       if (strlen(ret[0]) > 8) {
357         fprintf(stderr,"ID max digits are 8\n");
358       }
359       PDS[n-1].id = atoi(ret[0]);
360       strncpy(PDS[n-1].name, ret[1], MAX_STR_LEN);
361       PDS[n-1].name[MAX_STR_LEN] = '\0';
362       cnt2 = split(ret[2], ret2, '-', 3);
363       if (cnt2 == 3) {
364         PDS[n-1].birthday.y = atoi(ret2[0]);
365         PDS[n-1].birthday.m = atoi(ret2[1]);
366         PDS[n-1].birthday.d = atoi(ret2[2]);
367         if (PDS[n-1].birthday.y > 9999 ||
368             PDS[n-1].birthday.y < 0    ||
369             PDS[n-1].birthday.m > 12   ||
370             PDS[n-1].birthday.m < 1    ||
371             PDS[n-1].birthday.d < 1    ||
372             PDS[n-1].birthday.d > 31) {
373           fprintf(stderr,"inputed birthday data is inappropriate\n");
374           cnt2 = 0;
375         }
376       } else if (cnt2 == 2 || cnt2 == 1) {
377         fprintf(stderr,"inputed birthday data is inappropriate\n");
378       }
379       strncpy(PDS[n-1].home, ret[3], MAX_STR_LEN);
```

```
380       PDS[n-1].home[MAX_STR_LEN] ='\0';
381       PDS[n-1].comment = (char*)malloc(sizeof(char)*strlen(ret[4])+1);
382       strcpy(PDS[n-1].comment, ret[4]);
383         if (cnt2 == 2 || cnt2 == 1 ) {
384         fprintf(stderr,"correct birthday form example: 1000-10-10\n");
385         }
386     } else {
387       fprintf(stderr,"error: this input is wrong form\n");
388       fprintf(stderr,"correct form : (ID),(name),(birthday),(home),(comment)\n");
389     }
390     } else
391       fprintf(stderr,"error: your inputed argument is not correct\n");
392   }
393
394   void cmd_delete(int n) {
395     int i;
396     if (n-1 < PDN && n > 0) {
397     printf("Number %d date deleted\n", n);
398
399     for(i = 0;i < PDN - (n-1);i++) {
400       swap(&PDS[n-1], &PDS[n]);
401       n++;
402     }
403     PDN--;
404     } else
405       fprintf(stderr,"error: your inputed argument is not correct\n");
406   }
407
408   void cmd_find2(char *word) {
409     int i;
410
411     for (i = 0; i < profile_data_nitems; i++) {
412       if (comparsion(word, i) == 1) {
413         print_profile(&profile_data_store[i]);
414         printf("this data is number %d from start\n",i+1);
415       }
416     }
417   }
418
419   void cmd_Bwrite(char *line) {
420     int length; /*profile_date_store の comment の文字数を保存する変数*/
421     int i;
422     int p; /*comment を格納する配列を初期化することに使用する変数*/
423     char a[100+1]; /*comment を格納する配列　100 文字まで格納可能*/
424     FILE *fpw = fopen(line, "wb");
425
426     fwrite(&PDN, sizeof(int), 1, fpw);
427     for(i=0;i<PDN;i++){
428     length = strlen(PDS[i].comment);
429     fwrite(&length, sizeof(int), 1, fpw);
430     fwrite(&PDS[i], sizeof(PDS[i]), 1, fpw);
431     for(p=0;p<101;p++) { /*配列を初期化*/
432       a[p] = '\0';
433     }
434     if(length > 100) {
435       length = 100;
436     }
437     strncpy(a, PDS[i].comment, length);
438     a[100] = '\0';
439     fwrite(&a, sizeof(char), 100+1, fpw);
440     }
441
```

```
442   fclose(fpw);
443
444 }
445
446 void cmd_Bread(char *line) {
447   int length; /*profile_date_store の comment の文字数を保存する変数*/
448   int i;
449   int bdn; /*読み込んだバイナリファイル中のデータ数*/
450   FILE *fpr = fopen(line, "rb");
451   char a[100+1]; /*comment を格納する配列　100 文字を想定*/
452
453   fread(&bdn, sizeof(int), 1, fpr);
454   bdn = bdn + PDN;
455   for(i=PDN;i<bdn;i++) {
456   fread(&length, sizeof(int), 1, fpr);
457   fread(&PDS[i], sizeof(PDS[i]), 1, fpr);
458   PDS[i].comment = NULL;
459   fread(&a, sizeof(char), 100+1, fpr);
460   PDS[i].comment = (char*)malloc(sizeof(char*)*length+1);
461   strcpy(PDS[i].comment, a);
462   PDN++;
463   }
464   fclose(fpr);
465
466   if (save == 0) { /*Quit コマンドで使用*/
467     save = 1;
468   }
469 }
470
471
472 void print_profile(struct profile *p) {
473   printf("Id    : %d\n",p->id);
474   printf("Name  : %s\n",p->name);
475   printf("Birth : %04d-%02d-%02d\n",p->birthday.y,p->birthday.m,p->birthday.d);
476   printf("Addr  : %s\n",p->home);
477   printf("Com.  : %s\n",p->comment);
478   printf("\n");
479 }
480
481 void fprint_profile_csv(FILE *fp, struct profile *p) {
482   fprintf(fp,"%d,",p->id);
483   fprintf(fp,"%s,",p->name);
484   fprintf(fp,"%d-%d-%d,",p->birthday.y,p->birthday.m,p->birthday.d);
485   fprintf(fp,"%s,",p->home);
486   fprintf(fp,"%s\n",p->comment);
487
488 }
489
490 char *date_to_string(char buf[], struct date *date)
491 {
492   sprintf(buf, "%04d-%02d-%02d", date->y, date->m, date->d);
493   return buf;
494 }
495
496 int comparsion (char *word, int i) {
497   char buf[8 + 1]; /* ID は 8 桁と制限しているため 8+1 としている*/
498   char birthday_str[10 + 1]; /*birthday は"-"を含めて 10 桁になるよう制限しているた
め*/
499
500   sprintf(buf, "%d", PDS[i].id);
501   if (strcmp(PDS[i].name, word)    == 0  ||
502       strcmp(PDS[i].home, word)    == 0  ||
```

```
503       strcmp(PDS[i].comment, word) == 0  ||
504       strcmp(buf, word)         == 0  ||
505       strcmp(date_to_string(birthday_str, &PDS[i].birthday), word) ==0) {
506     return 1;
507   } else
508     return 0;
509 }
510
511 int compare_profile(struct profile *p1, struct profile *p2, int column) {
512   switch (column) {
513   case 1:
514     return p1->id - p2->id;
515   case 2:
516     return strcmp(p1->name, p2->name);
517   case 3:
518     if (p1->birthday.y != p2->birthday.y) return p1->birthday.y - p2->birthday.y;
519     if (p1->birthday.m != p2->birthday.m) return p1->birthday.m - p2->birthday.m;
520     return p1->birthday.d - p2->birthday.d;
521   case 4:
522     return strcmp(p1->home, p2->home);
523   case 5:
524     return strcmp(p1->comment, p2->comment);
525
526
527   }
528 }
529
530 void b_sort(struct profile *p, int left, int right, int column) {
531   int i, j;
532
533   for (i = left; i <= right; i++) {
534     for (j = left; j <= right - 1; j++) {
535       if ((compare_profile(p+j, p+j+1, column)) > 0) {
536           swap(&p[j], &p[j + 1]);
537       }
538     }
539   }
540 }
541
542 void swap(struct profile *a, struct profile *b) {
543   struct profile temp;
544
545   temp = *a;
546   *a = *b;
547   *b = temp;
548 }
549
550
551
552 void q_sort(struct profile *p, int left, int right, int column) {
553   int i,j;
554   int pivot;
555   pivot = (left + right) / 2;
556
557   i = left;
558   j = right;
559
560   while(1) {
561     while ((compare_profile(p+pivot, p+i, column)) > 0) {
562       i++;
563     }
564     while ((compare_profile(p+j, p+pivot, column)) > 0) {
```

```
565       j--;
566     }
567       if (i <= j) {
568         break;
569       }
570     swap(&p[i], &p[j]);
571     i++;
572     j--;
573   }
574     q_sort(p,left, j, column);
575     q_sort(p, i, right, column);
576 }
```