名簿管理プログラム
—独自コマンドを添えて—

氏名: 中嶋空偉 (NAKAJIMA, Sorai)
学生番号: 09B23549

出題日: 2024年6月12日
提出日: 2024年6月19日
締切日: 2024年7月24日

1　概要

構造体と配列を用いて，大量のデータを扱うC言語プログラムの基本を学ぶ．その過程で，標準入力から「ID, 氏名, 誕生日, 住所, 備考」からなる情報を管理する名簿管理プログラムを作成する．このプログラムでは，上の形式で入力するとデータが保存される．また，%から始まるコマンドを入力すると，内容を表示，プログラムの停止，保存数の参照，探索，ファイルへの書き込み，読み込み，データの整列などができる．

本レポートでは，探索，ファイルへの書き込み，読み込み，データの整列のコマンドを実装していく．

演習中に取り組んだ課題として，以下の課題4から課題7についての内容を報告する．

課題1　名簿管理プログラムの要件や仕様を踏まえたプログラム作成方針の検討
課題2　コマンド実装と考察(a)%Fコマンド
課題3　コマンド実装と考察(b)%Rコマンドと%Wコマンド
課題4　コマンド実装と考察(a)%Sコマンド
課題5　コマンド実装と考察(a)%独自コマンド
課題6　プログラム全体に対する考察
課題7　発展課題の実装

また，考察課題として，以下の観点での考察をおこない，??章にまとめた．

考察1　メモリ上のデータ配置と構造体のサイズに関する考察（??節)
考察2　エラー処理に関する考察（??節）

2　プログラムの作成方針

本演習で作成したプログラムが満たすべき要件と仕様として，「(1) 基本要件」と「(2) 基本仕様」を示す．

(1) 基本要件

1. プログラムは，その実行中，少なくとも10000件の名簿データをメモリ中に保持できるようにすること．
2. 名簿データは，「ID，氏名，誕生日，住所，備考」を一つのデータとして扱えるようにすること．
3. プログラムとしての動作や名簿データの管理のために，以下の機能を持つコマンドを実装すること．
   （a） プログラムの正常な終了
   （b） 登録された名簿データのデータ数表示
   （c） 名簿データの全数表示，および，部分表示
4. 標準入力からのユーザ入力を通して，データ登録やデータ管理の操作を可能とすること．
5. 標準出力にはコマンドの実行結果のみを出力すること．

(2) 基本仕様

1. 名簿データは，コンマ区切りの文字列（CSV入力と呼ぶ）で表されるものとし，Listing ??に示したようなテキストデータを処理できるようにする．
2. コマンドは，%で始まる文字列（コマンド入力と呼ぶ）とし，表??にあげたコマンドをすべて実装する
3. 1つの名簿データは，C言語の構造体 (`struct`) を用いて，構造を持ったデータとしてプログラム中に定義し，利用する．
4. 全名簿データは，”何らかのデータ構造”を用いて，メモリ中に保持できるようにする．
5. コマンドの実行結果以外の出力は，標準エラー出力に出力する．

2.1　プログラムの全体像

2.2　名簿管理プログラムの実装方針

3　プログラムの実装と考察

3.1　コマンドの実装と考察(a)%Fコマンド

3.1.1　プログラムの説明

この関数が呼び出されると、与えられた引数を使ってprofile_data_store配列内のプロファイルを検索する.

- 引数 引数として文字列ポインタwordを受け取る。このwordが探索する際のキーワードとなる.
- 動き
  1. profile_data_store配列を一つずつ見ていって、各プロファイルをチェックしていく.
  2. 各プロファイルに対して以下のチェックを行う.:
     （a） プロファイルIDをsprintfを使って文字列に変換し、strcatを使ってwordと比較する．IDが一致した場合、そのプロファイルの情報を出力する.
     （b） wordとプロファイルのコメント、名前、住所を上と同じように比較し、いずれかが一致した場合、そのプロファイルの情報を出力する.
     （c） プロファイルの誕生日を文字列に変換し、wordと同じように比較する．誕生日が一致した場合、そのプロファイルの情報を出力する.

3.1.2　プログラムの動作確認

3.1.3　実装にあたっての考察

この関数の実装にあたってatoiについてエラーが多く起こってしまったのでなぜうまくいかなかったのかについて考察する.

- 変換方法: atoiは文字列が数字でない場合，いくつかの動作を行う．例えば，12345abcが入力されたとき12345と数字以外のものがくるまで変換して返す．abcが入力されたときは，変換できずに0が返される．数字の0が返されたのか，文字列を変換した結果返されたのがが分からないため，エラーの原因になりえる.
- 比較方法: 文字列と整数の比較を行う場合，変換がうまくいかなかった場合，その後に影響する場合もある．特に、文字列である場合、エラーが起こる.

いづれかの理由でエラーがおこってしまう．そのため今回はsprintfを使用して整数値を文字列に合わせるほうが確実だと考え，そちらを採用した.

3.2 コマンドの実装と考察(b)%Rコマンドと%Wコマンド

3.2.1 %Rコマンドの説明

この関数が呼び出されると，与えられたファイルからデータを読み取り，プロファイルを作成し，profile_data_store配列に格納する．

- ファイルのオープン：ファイルの名前が引数として渡されると，fopen関数により引数と一致するファイルが読み取りモード"r"で開かれる．開くのに失敗したときエラーとして出力される．
- ファイルからのデータ読み込み： textttfget関数を使ってfileからlineに一行ずつ格納する．その際，subst関数で改行文字をnull文字に変える．不要な改行をなくす．
- プロファイルの作成と格納： profile_data_itemがMAX_PROFILES以上であればエラーを返す．それ未満であれば， profile型の構造体を定義し，profileに格納していく関数new_profileに渡す．その際，profile_data_itemを1増やす．作成できなかった場合はエラーを返す．
- ファイルのクローズ：データの読み込み後にfclose関数によってファイルを閉じる．

3.2.2 %Wコマンドの説明

この関数が呼び出されると，引数によって与えられたファイルに現在保持しているデータを書き込む，

- ファイルのオープン：ファイルの名前が引数として渡されると，fopen関数により引数と一致するファイルが書き込みモード"r"で開かれる．開くのに失敗したときエラーとして出力される．
- ファイルへのデータ読み込み： profile_data_itemに基づいて/textttforループを実行する．それぞれのデータを構造体のポインタに格納する．その後，csv形式に合ったデータをfprintf関数によってファイルに保存していく．
- ファイルのクローズ：データの書き込み後にfclose関数によってファイルを閉じる．

3.2.3 プログラムの動作確認

3.2.4 実装にあたっての考察

この関数の実装にあたってget_line関数を変更するべきか，それとも新しくすべきか考えたので考察していく． get_line関数を標準入力とファイルからの読み込みに対してどちらも対応した改良版のコードはこの後記載している．

- まとめることのメリット： get_line関数をまとめることによって同じような関数を2回定義しなくてよいので簡潔なコードになる．また，エラーや修正が必要な改善が求められたときに修正箇所が少なくて済む．

- まとめることのデメリット：標準入力のときのみ追加したい事項やファイルのときのみ追加したいエラーなどがあった場合に実装しづらい．また，%Rコマンドの時にのみしか必要ないにもかかわらず，多く使用されるであろう標準入力のときにもif(file == NULL)という条件分岐が発生するため，処理速度に影響があると推測できる．

get_line関数をまとめることによって簡潔さなどが向上する反面，実際は時間のかかる処理ではないため，影響はないかもしれないが，1万という大きなデータを扱う上では大きな差になってしまうかもしれない．

```
 1:       int get_line(char *line, int size, FILE *file) {
 2:     if (file == NULL) {
 3:         if (fgets(line, 1024, stdin) != NULL) {
 4:             subst(line, '\n', '\0');
 5:             return 1;
 6:         }
 7:     } else {
 8:         if (fgets(line, 1024, file) != NULL) {
 9:             subst(line, '\n', '\0');
10:             return 1;
11:         }
12:     }
13:     return 0;
14: }
```

3.3 コマンドの実装と考察(c)%Sコマンド

3.3.1 プログラムの説明

この関数は、cmd_sortが呼び出されると，int型の引数columnによって id，名前，誕生日，住所，備考について並び替える．なお，並び替えた後の表示の機能はない．このソートはバブルソートのアルゴリズムに基づいている．

- 引数 引数としてint型の引数columnを受け取る。このcolumnがソートする際のキーワードとなる．
- 動き
  1. swap_profile関数:メモリ効率のために構造体のポインタを通してprofileを交換する．
  2. switch文によりcolumnの値に応じてそれぞれソートを行う．
  3. バブルソートを行っていく，要素同士を比較して入れ替える関数をそれぞれの要素に対して用意している．

     idのとき，それぞれをsprintfによって文字列に変換する．strcmpにより比較する．この値が正の時，swap関数によって入れ替える．

     birthdayのとき，それぞれをsprintfによって年，月，日付をつなげて文字列に変換する． strcmpにより比較する．この値が正の時，swap関数によって入れ替える．

     name,address,commentのとき,strcmpにより比較する．この値が正の時，swap関数によって入れ替える．

3.3.2 プログラムの動作確認

3.3.3 実装にあたっての考察

この関数の実装にあたって計算量がどのようになっているのか気になったので各関数の計算量，改善案について考察する．

- 計算量： `swap_profile`関数は構造体のコピーが3回ある．コピー操作はO(1)なので全体はO(1)． `sort_id`関数などの比較して並び替える関数は与えられたデータを文字列に変換して比較し，必要であれば`swap_profile`関数を呼び出す．それぞれの操作はO(1)なので全体の操作はO(1)．`cmd_sort`関数はそれぞれの分岐に対して`for`ループにより N回ループの中でN回ループが実行される．その中で計算量$O(1)$の関数が実行される．よって全体の計算量は$O(N^2)$となる
- 改善案1：クイックソートやマージソートなどのソート法に変える．これらのソート法は平均計算量が$N \log N$なので計算量を減らすことができる．
- 改善案2：比較する際の文字列の変換を減らす．現状，比較する際にすべてのデータを文字列に変換しているが，idやbirthdayは数値のまま比較することで計算量は変換するために必要な分だけ計算量は減る．

3.4 コマンドの実装と考察(d)独自コマンド

3.4.1 %Dコマンドの説明

この関数が呼び出されると，`id`が一致するものを探し，その`profile`データを削除する．

- 探索： idが引数として渡されると，引数と一致するidをもつプロファイルが1から順に探索される．
- メモリの解放： `free`関数を用いて，探索で見つかったプロファイルの`comment`フィールドのメモリを解放する．メモリの解放がないとのちにメモリの圧迫により，エラーや動作が重くなる原因になる．
- profileの削除：見つかったプロファイルがあった場所にその次のプロファイルを格納していく．`for`ループによって一つずつずらしてプロファイルを削除する．最後に`profile_data_items`を1減らす．

3.4.2 プログラムの動作確認

3.4.3 実装にあたっての考察

この関数を実装するうえで計算時間が多くなってしまっているのではないかと考えたので考察する．

- 計算量：
- 改善案1：二分探索法の実装

4　プログラム全体の考察

4.1　メモリ上のデータ配置と構造体のサイズに関する考察

4.2　エラー処理に関する考察

5　発展課題

6　感想

7　作成したプログラムのソースコード

作成したプログラムを以下に添付する．なお，1章に示した課題については，??章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:
 3: int main()
 4: {
 5:     char s[4] = {'A', 'B', 'C', '\0'};
 6:
 7:     printf("s = %s\n", s);
 8:
 9:     return 0;
10: }
```

参考文献

[1] 平田富雄，アルゴリズムとデータ構造，森北出版，1990.

[2] 著者名，書名，出版社，発行年.

[3] WWWページタイトル，URL，アクセス日.