# プログラミング演習I第8回レポート

学籍番号：2364902

名前：キム　ギュソク

## (1) 課題番号と課題内容

### A. 基本課題 1

i. データファイルから読み込んだ学生のデータを、指定されたフォーマットで表示するプログラムに次の処理を追加する。構造体配列とデータ数を引数とする。単純選択整列のアルゴリズムを用いた整列関数（得点で降順）を作成する。整列した結果を表示する

(2) フローチャートまたは疑似言語によるアルゴリズムの記述

A. 基本課題 1

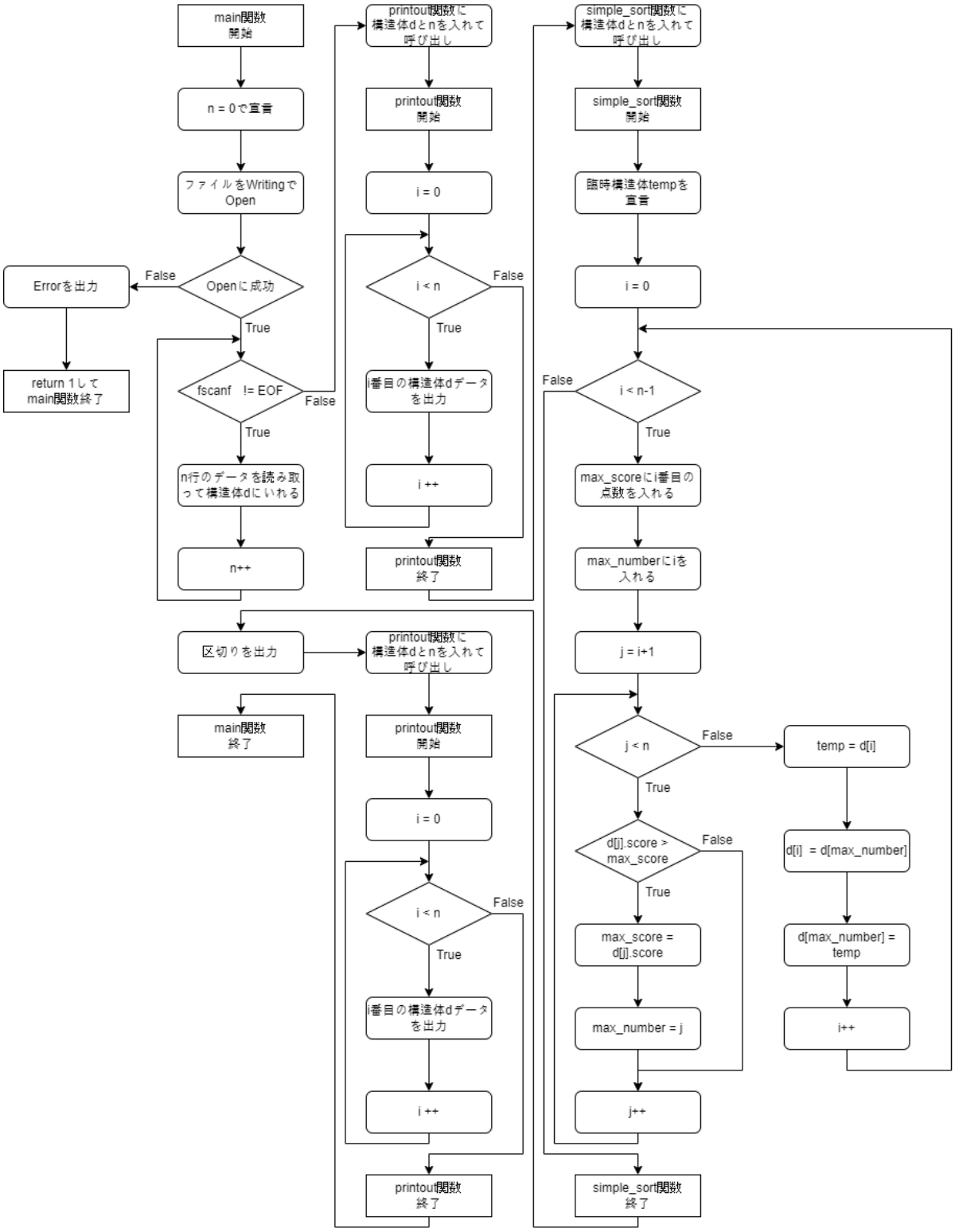


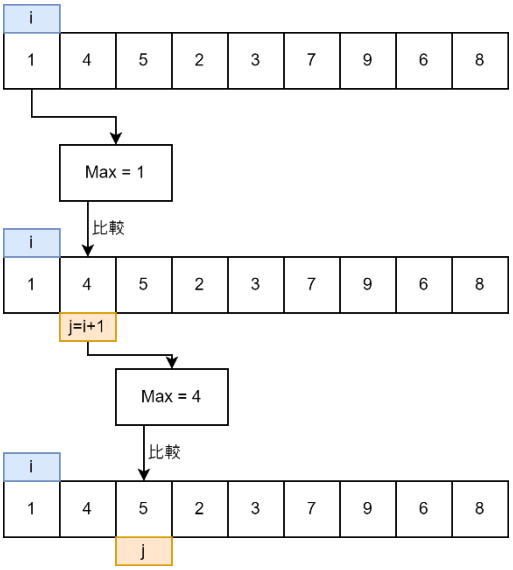

**図 1. フローチャート**

## (3) アルゴリズムが正しいことの説明

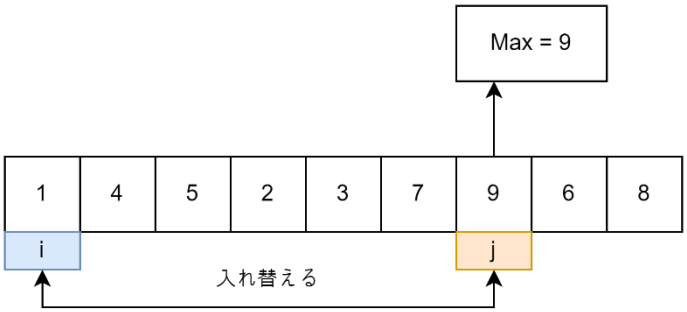
### A. 基本課題 1

#### i. 正当性

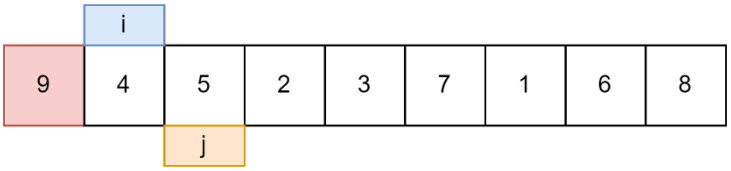
1. 単純選択整列は降順の場合配列の中で比較したい対象が一番大きいものを選択して一番前に置くことで整列するアルゴリズムである。例えば、[1, 4, 5, 2, 3, 7, 9, 6, 8]の配列が与えられたとする。




このような方法で最大値を求めることができる




よって、i番目の数字とj番目のデータを入れ替えると




このように最大値が左に次第に入ると

このように大きい順に整列できる。このような方法を simple_sort 関数で行っている。特に、simple_sort関数では数字だけを入れ替えるだけではなく構造体を入れ替えることで名前と学籍番号と成績が同時に変えるように設定することで元のデータの形式は維持しながら降順に整列することができる。よって、このアルゴリズムは正当であると考えることができる。

ii. 停止性

1. このアルゴリズムでは繰り返し文をmain関数のデータの数を数えるための繰り返し文とprintout関数のiの繰り返しとsimple_sort関数の外側繰り返し文と内側繰り返し文総合四つ存在する。
   A. main関数
      i. fscanfの場合ファイルを読み取ってファイルの最後の部分にたどり着くとEOF（End Of File）である-1を読み取る。よって、終わらないファイルは存在しないので必ずファイルの最後にたどり着いてEOFを読み取ったときに繰り返し文を終了するように条件を付けることで正しく停止するように設計されている。
   B. printout関数
      i. この繰り返し文では全てのデータを出力するための繰り返し文なのでn回繰り返す必要がある。よって、インデクスとしてiを設定して増やしながらi ＜ nという条件を付けることでn回繰り返して正しく停止する。
   C. simple_sort関数
      i. 外側繰り返し
         1. n-1まで大きい順に整列するとn番目のデータは自動的に一番小さいデータになるのでn-1まで繰り返す必要がある。よって、インデクスとしてiを設定して i ＜ n-1という条件を付けることでn-1インデクスまで整列して停止するように設計されている。
      ii. 内側繰り返し
         1. i番目に入る最大値を求めるためにi+1からnまでに比較することで最大値を求めることができる。よって、i+1からnまで繰り返すように設計する必要がある。よって、インデクスjをi+1として設定してj ＜ nまでの条件を付けることでi+1からnまで繰り返すことができる。よって、正しい結果が得られた時に停止する。

# (4) ソース・プログラムの説明

## A. 基本課題 1

### i. ソースコードの説明

1. まず構造体studentを学籍番号と名前と点数として宣言する。その後、各関数のプロトタイプを宣言する。main関数でデータの数を表す n を 0 で初期化して宣言して、引数としてもらったファイルの名前をreadingであるrでポインタとしてfpにいれる。そして、もしファイルを開くのに失敗したときにわかるようにif文でエラーメッセージを出力して1をreturnするようにする。その後、student構造体の構造体配列をd最大データ数分宣言する。その後、ファイルが終わるまでに構造体にデータを入れながら n を増やしてデータの数を数えながらファイルのデータを構造体に入れる。その後、予め宣言したprintout関数に構造体配列とデータの数を入れて呼び出す。printout関数はfor文を利用して全てのデータを出力する関数である。simple_sort関数を呼び出す前に整列する前と後を確認するために予め出力してsimple_sort関数に構造体配列とデータ数を入れて呼び出す。simple_sort関数では構造体の順番を入れ替えるための臨時構造体tempを宣言してfor文を利用してi番目のデータをmax_scoreに入れ、iをmax_numberに入れる。その後、二重ループとしてfor文でi+1から一つ一つ比較するのでjにi+1を入れてnまで繰り返す。そして、if文を利用して大きい場合max_scoreとmax_numberに最大値の点数とインデクスを入れて最大値と最大値のインデックスを求める。その後、tempを利用してi番目のデータとmax_number番目のデータを入れ替えて整列する。その後、もう一回printout関数を呼び出して整列されたデータを出力してプログラムは終了する。

## (5) 考察

### A. 基本課題 1

i. 単純選択整列の場合外側繰り返しのn-1を繰り返す度に最大値を求めるためにN-1, N-2, N-3, … , 3, 2, 1回の比較演算が必要である。よって、

$$T(n) = (n-1) + (n-2) + (n-3) + \cdots + 2 + 1 = \frac{(n-1) \times n}{2}$$

$$O(n) = n^2$$

よって、計算にnこのデータがある時に$n^2$の計算が必要である。この計三回数はとても非効率的である。最近のビッグデータの時代にはたくさんのデータを扱うことが多い。例えば、１万のデータを単純選択整列すると10000×10000すると100,000,000の計算が必要である。もし一つの計算が1秒かかると仮定すると、100,000,000秒約３年が必要である。このような単純に整列するだけの作業に3年も使うことはとても非効率である。よって、データが多くなった時には他の計算回数が少ないクイックソートを利用することがより効率的だと考える。クイックソートの場合n個のデータに対してかかる計算回数は$nlog_2 n$である。もし同じく１万のデータを整列すると考えると、約1.5日で整列が終わる。3年と1.5日の差はとても大きいと考えられる。つまり、データ数が少なく単純な考えで整列したいときには単純選択整列の方がいいと思う。しかし、データ数が少しでも大きくなるとより効率的なアルゴリズムを使う方がいいと思う。

## (6) 感想

A. 授業で覚えた最大値を求める方法を利用することでよく理解することができた。また、人が直観的やっている整列をプログラムとして作ることを考えることで日常的に考えることをプログラムとして活用できるかを考えるようになった。また、色々なソートの種類を調べながらアルゴリズムの効果を理解することができた。