第７章．探索のアルゴリズム

【学習のねらい】

① 探索（検索）を行う基本的なアルゴリズム（線形探索、2 分探索）を学習し、その処理の流れを理解する。
② 探索アルゴリズムの効率について考察する。
③ 探索アルゴリズムを応用したプログラムを学習する。

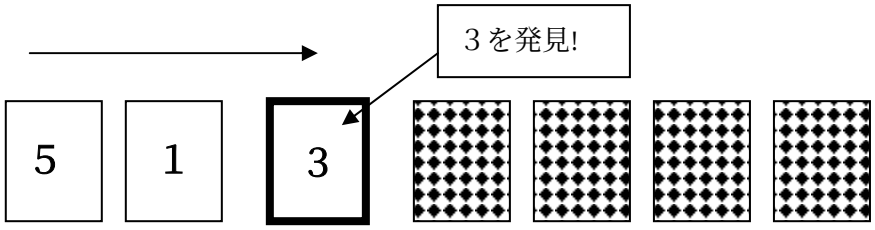
第６章で学習した整列（ソート）処理と並んでポピュラーな処理として、探索処理があります。検索と言ってもよいです。これは、あるデータ群から目的のデータと合致するものを見つける、と言う処理です。やはり、ソート処理と並んで応用範囲が広いため様々なアルゴリズムが研究されています。特に、キーワードを指定して該当する Web ページを検索する機能は、最近急速に進歩しました。皆も日常的に使っているはずです。これも探索アルゴリズムの応用に他なりません。今や検索の速さや効率を競い合い、そのアルゴリズムは企業秘密になっている程です。

本章では、探索に関する、最も基本的なアルゴリズムを学習します。最先端のアルゴリズムをいきなり理解することは困難ですが、ここで学習するアルゴリズムは極めて分かりやすい基本アルゴリズムです。しかし、探索の基本はここにあります。前章と並んでアルゴリズム学習のヤマ場です。じっくりと学習しましょう。

７－１　線形探索

まず、最も単純な線形探索から学習を始めましょう。これは、データの先頭から順番に目的のデータと符合するかどうかをチェックして行く方法です。簡単な具体例で考えましょう。

今、数字が記入されたカードが何枚かテーブルの上に（伏せて）置かれているものとします。そして例として、この中に数字の「３」が含まれているかどうかを確かめる、という場面を想定しましょう。皆はどうしますか・・・？わざわざ考えるまでもなく、端から順番にめくって行ってその数字が３かどうかを確かめて行くでしょう。例えば端から 3 番目をめくった時点で３を見つけたらこの処理は終了です。



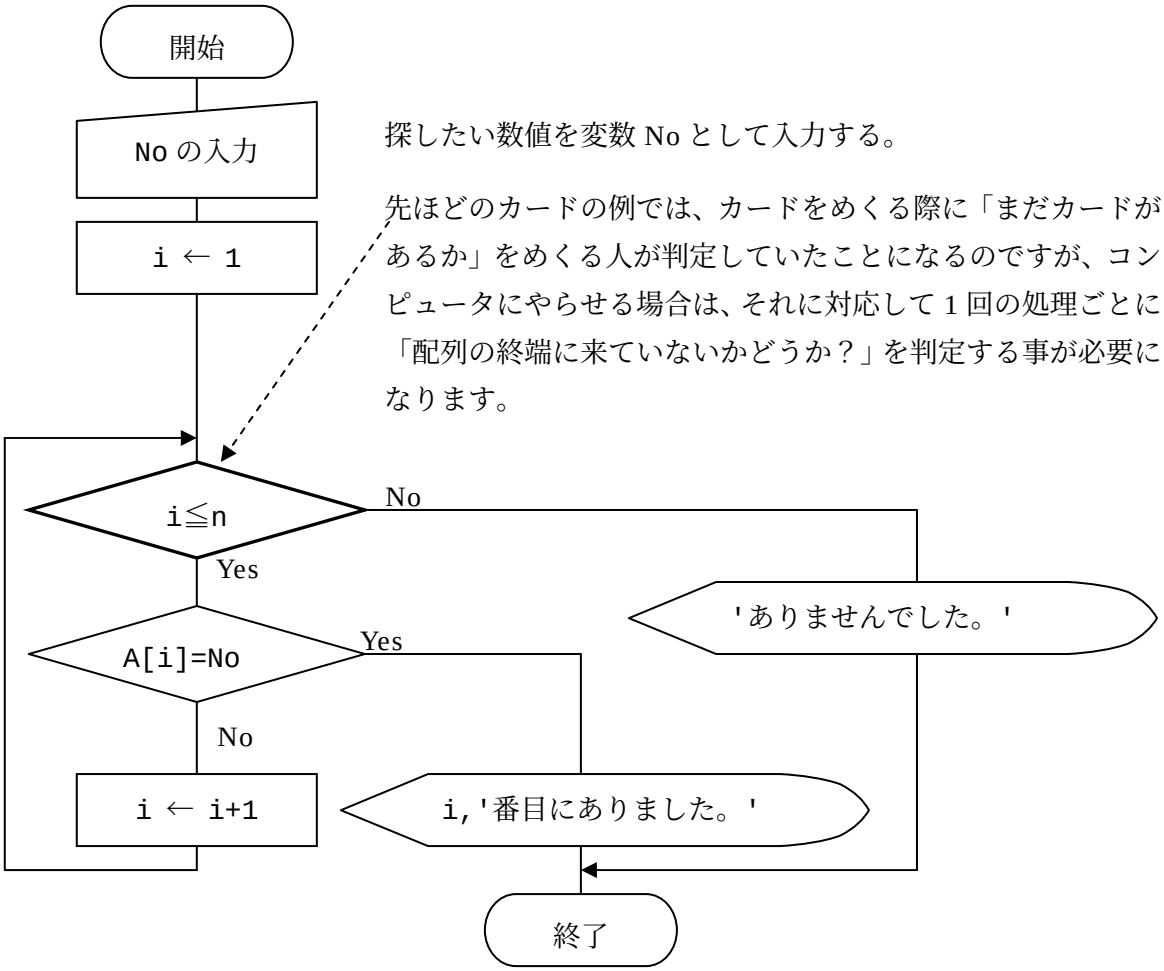
この、いわば”わざわざ考えるまでもない”単刀直入な探索方法が**線形探索法**なのです。

さて、それではこの探索処理をコンピュータにやらせることを考えましょう。いつも通

り、コンピュータが扱いやすいように数字が入っているデータは配列にします。そこで、次のように問題を定式化しましょう。

「今、A[1]～A[n]の大きさｎの配列に、任意の数値が保管されている。この中に、指定した数値が入っているかどうか、入っていればそれは何番目のデータか？」を求めるものとします。そのアルゴリズムは次の通りとなります。

＜線形探索のアルゴリズム＞



探したい数値を変数 No として入力する。

先ほどのカードの例では、カードをめくる際に「まだカードがあるか」をめくる人が判定していたことになるのですが、コンピュータにやらせる場合は、それに対応して 1 回の処理ごとに「配列の終端に来ていないかどうか？」を判定する事が必要になります。

この流れ図から明らかな様に、線形探索法では、1 回のデータ照合を行うに当たって

①　データが終端に来ていないかどうか（まだデータがあるかどうか）

②　探しているデータと一致しているか

の 2 回の判定（変数比較）を行うことになります。ですから、もし該当するデータがなかった場合、ｎ個のデータと照合することになりますから、２×ｎ回の比較を行うことになります。

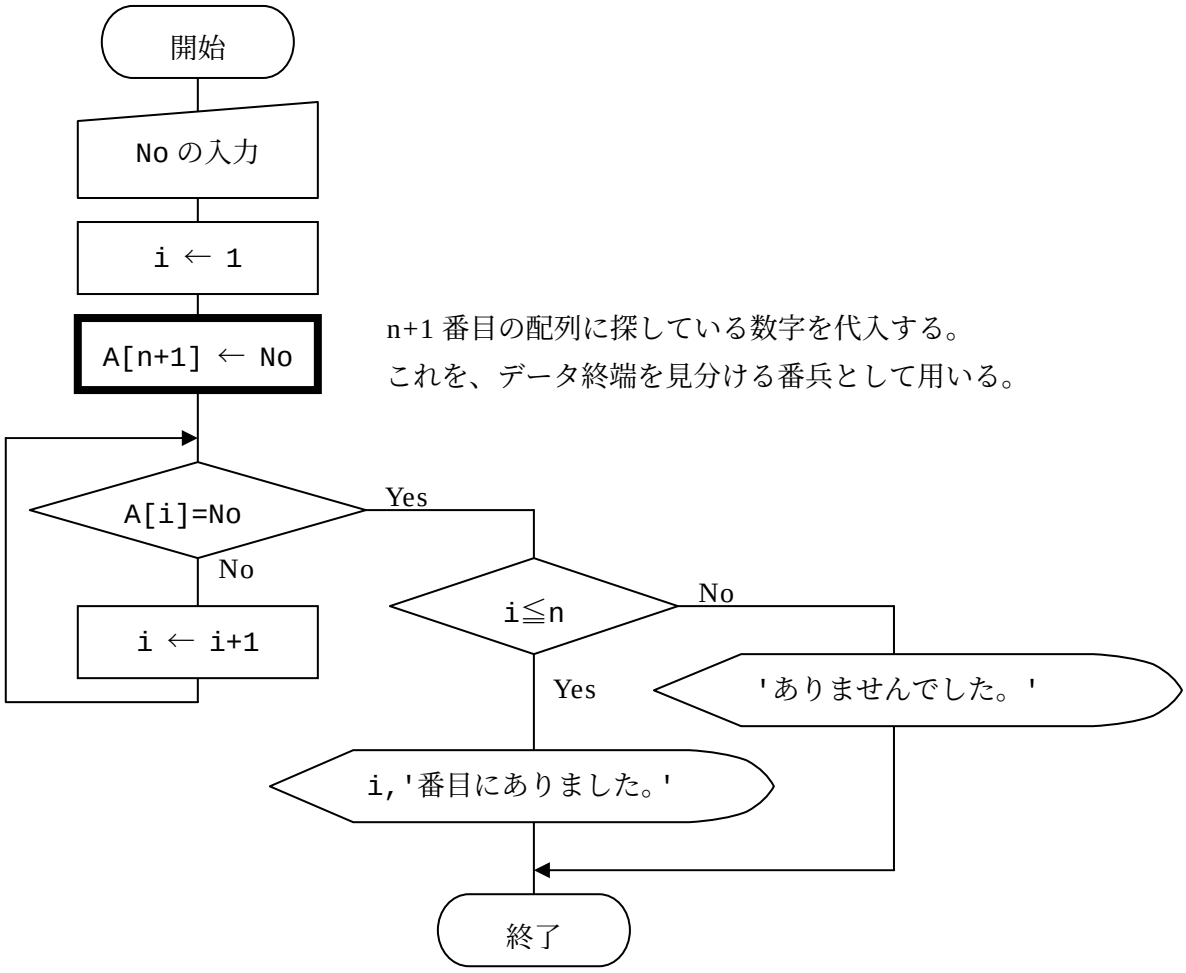
でも、②の比較を 1 回毎に行うのは仕方がないとしても、毎回①の比較を行うのは、何か無駄なことをしている様な気がしませんか・・・？

そこで、編み出されたのが「番兵法」という改善方法です。番兵とは見張りのことで、何ともユニークな名前ですが、アルゴリズム論の世界では立派な専門用語になっています。それでは、その番兵とはいったい何なのか・・・？　種明かしをすると、「最後の次のデータ、つまり A[n+1]番目のデータを用意し、そこに探している数値を代入してそれを（データの終端を監視する）番兵として用いる。」ということになります。この番兵法を用いた場合には、

① 必ず、探している数字が見つかることになります。
② 問題は、見つかったのが何番目のデータかということです。
③ それがｎ番目以内 {A[1]～A[n]} であれば、最初のｎ個のデータ中にあったことになり、n+1 番であれば、「なかった。」ということになります。
④ データの最後に番兵がくっつくことで、「データが終端に来ていないかどうか」を毎回のデータ照合毎に判定する必要がなくなります。

以上を念頭において、以下の番兵法のアルゴリズムをみてみましょう。

＜番兵法＞



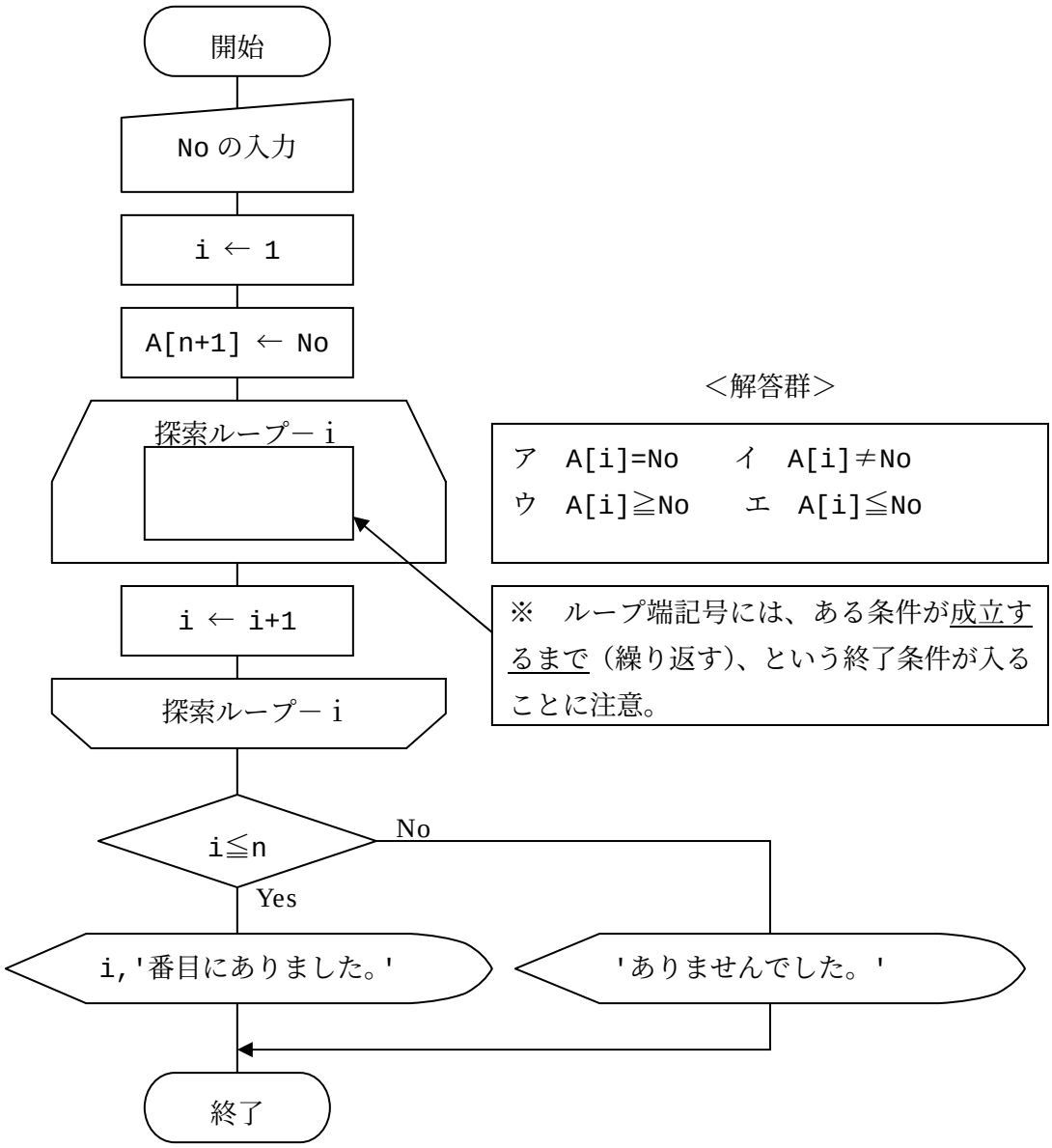
流れ図から分かるとおり、データの照合に当たって「データが終端に来ていないかどうか（まだデータがあるかどうか)」の判定（比較）は行う必要はなくなりました。なぜなら、最初のデータに探索対象の数字が含まれていなくても、n+1 番目より先に処理が進むことがなくなったからです。そして、見つかった後に、「その要素番号がｎ以下であるかどうか」を最後に 1 回判定するだけで、最初のデータ中に探索している数字が含まれていたかどうかを判定することができるようになりました。この ”からくり ”を理解できたでしょうか？

【基礎課題 7-1】

データ数がｎ個ある場合、その中に探索するデータがなかった場合（つもり最も処理に手間がかかった場合)、線形探索では 2ｎ回の比較（判定）を行いました。それでは、番兵法では何回比較（判定）を行うことになるでしょうか？

【基礎課題 7-2】

上の番兵法の流れ図は、ある条件が成り立つと繰り返しループから抜け出る構造をしています。これを、ループ端記号を用いて記述すると次の通りとなります。空欄に入る式を解答群から選択してください。



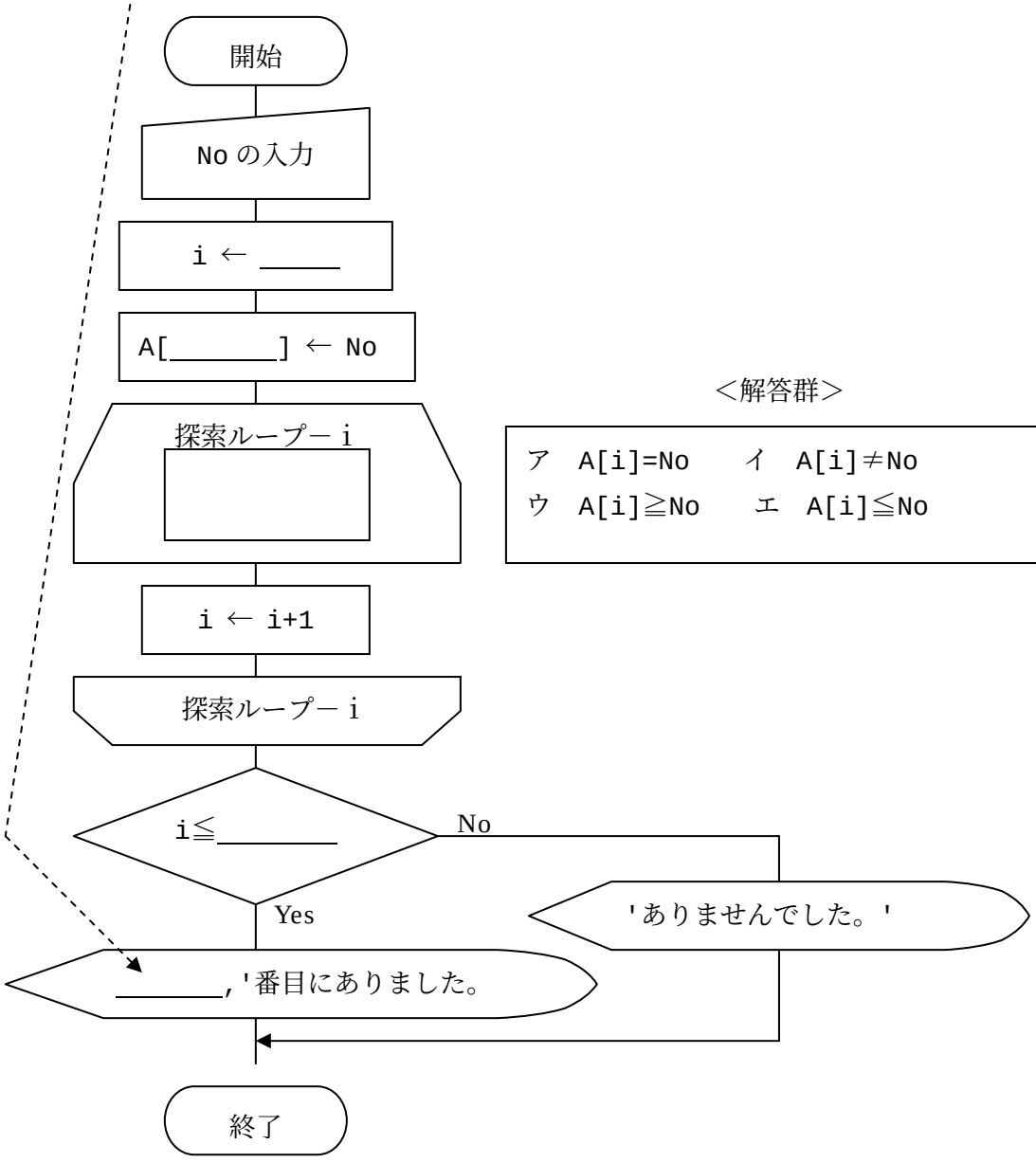
＜解答群＞

ア　A[i]=No　　イ　A[i]≠No
ウ　A[i]≧No　　エ　A[i]≦No

※　ループ端記号には、ある条件が成立するまで（繰り返す）、という終了条件が入ることに注意。

【基礎課題 7-3】

上の流れ図を実際の Java プログラムに適用する際には、例によって配列要素が 0 番目から始まることを考慮する必要があります。そこで、問題を「配列 A[0]～A[n-1]に、任意の数値が保管されている。この中に、指定した数値が入っているかどうか、入っていればそれは何番目のデータか？」を求めるものと設定しましょう。この場合の番兵法の流れ図は前ページからどのように修正すれば良いでしょうか？下の下線部に適切な式あるいは数値を埋めて下さい。空欄については、前ページ同様、解答群から適切な式を選択してください。なお、”何番目か”を数える際には、通常通り（0 番目ではなく）１番目から数えるものとします。



＜解答群＞

ア　A[i]=No　　イ　A[i]≠No
ウ　A[i]≧No　　エ　A[i]≦No

【基礎課題 7-4】

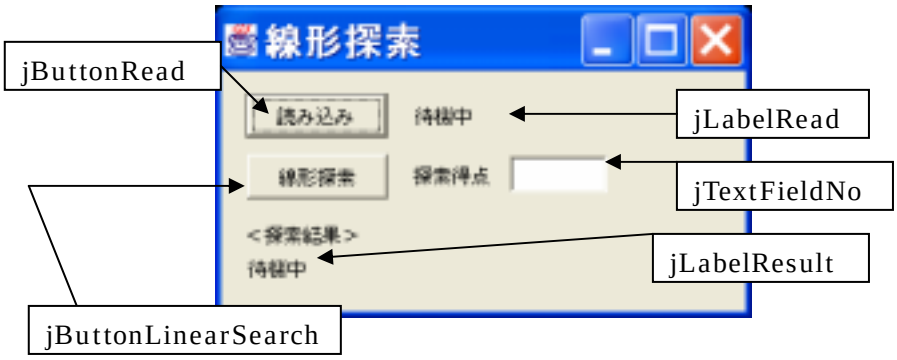
線形探索法（番兵法）を、実際の問題に応用してみましょう。簡単な練習として【基礎課題 6-1】で用いた「tokuten.txt」ファイルから得点を読み込み、指定した得点をとった人が（上から）何番目の人かを表示するプログラムを作成することにします。

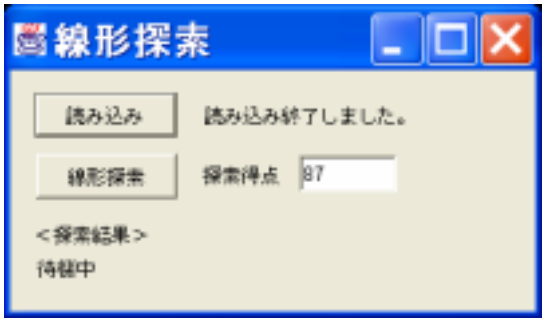
作成するプログラムの動作内容は次の通りです。

＜tokuten.txt＞

```
55
63
39
87
48
70
35
77
59
44
```

プログラムを起動すると次の画面が現れます。

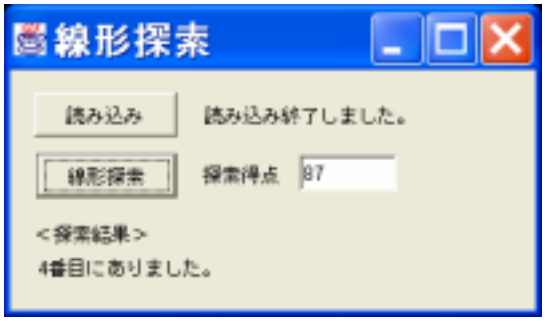




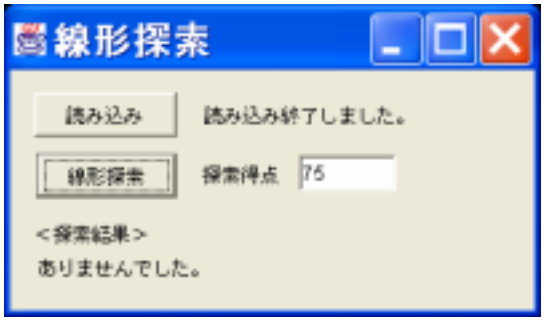
［読み込み］ボタンをクリックして入力ファイル「tokuten.txt」を指定します。ここまでは、【基礎課題 6-1】と全く同じです。

データ入力終了後、探索得点欄に探したい得点を入力します。その後［線形探索］ボタンをクリックすると・・・

該当する得点が存在する場合、上から何番目にあるかを表示します。



該当する得点が存在しなかった場合、なかった旨の表示をします。



それではプログラムを作成しましょう。まず、［読み込み］ボタンのイベントハンドラは【基礎課題 6-1】と全く同じです。p.92 の 1 および 2 の通り記述して下さい。そして［線形探索］ボタンのイベントハンドラは次のようになります。p.110 の流れ図を参考にして、空欄①～④を埋めて下さい。

＜［線形探索］ボタン＞

```
void jButtonLinearSearch_actionPerformed(ActionEvent e) {
  // 探索得点を変数 No に代入する。
  int No=Integer.parseInt(jTextFieldNo.getText());
  //以下は線形探索の処理
  int i=0;
  Tokuten[ ① ]=No; //番兵の定義

  while ( ② ) {
    i++;
  }
  if(i<= ③ ) {
    jLabelResult.setText( ④ +"番目にありました。");
  }
  else {
    jLabelResult.setText("ありませんでした。");
  }
}
```

※　while 文では、( )内の条件が成立している間繰り返す、という意味になります。一方流れ図のループ端記号による表記では、指定した条件が成立するまで繰り返す、という意味である点に注意して下さい。つまり、②に入る条件式は、p.110 の流れ図のそれと逆になります。よく分からない人は、6-3 節 p.101 の説明を読み返して下さい。

作成後プログラムを実行し、正しく動作するか確認して下さい。

７－２　２分探索

線形探索法は、例え番兵法を用いるとしても、データ数に比例して、データ照合のための比較回数が増えました。例えばデータ数が 10 倍になったら 10 倍計算時間がかかるということです。そのため、大きなデータ数の時には、大変効率が悪くなります。一方、データ数が 10 倍になっても比較回数はわずかしか増えない”うまい手”があります。それがここで学習する 2 分探索法です。まずは論より証拠。7-1 節で説明したカードの例でその処理の流れをみてみましょう。

今、10 枚のカード A[1]～A[10]があって、裏に数字が書かれています。これが下の様に昇順にソートされている場合を考えます。

A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
1	3	7	12	13	14	20	22	30	41

さて、例として数字「30」がこの中にどこに含まれているかどうかを探索する場合を考えましょう。

①　「30」を真ん中のカード、つまり 5 番目のカードと比較します。

1	3	7	12	13	14	20	22	30	41

30 は A[5]より大きいので、探索候補は、それ以降となります。つまり、A[6]～A[10]に探索範囲は絞られます。

探索範囲

1	3	7	12	13	14	20	22	30	41

②　今度は新たな探索範囲の真ん中である A[8]と 30 を照合します。

1	3	7	12	13	14	20	22	30	41

やはり、30 は A[8]より大きいので、候補はそれ以降、つまり A[9]か A[10]のいずれかに絞られます。

探索範囲

1	3	7	12	13	14	20	22	30	41

③　そこで、A[9]と 30 を照合します。

1	3	7	12	13	14	20	22	30	41

その結果、30 が A[9]にあることが分かりました。

この様に、ソートしたデータを用いると、1 回のデータの照合でその次の探索範囲を半分に絞ることができ、大変効率が良くなります（これに対して線形探索では、1 回の照合毎に、探索範囲は一つ少なくなるだけだった事を思い出してください）。これが、2 分探索法です。改めて言うまでもなく、探索範囲が常に 2 分される、つまり半分になることからこう命名されたものです。

【基礎課題 7-5】

2 分探索法で、上の例の様に 10 個のデータを探索対象とした場合、探索範囲は

１０→５→２→１

となるため、最高 4 回のデータ照合（比較）で探索が終了します（上の例で５の半分は 2.5 ですが、小数点以下は切り捨てます）。では、20 個の場合はどうでしょうか？確かめてみると

２０→１０→５→２→１

となるので最高 5 回で終了です。つまりデータが 2 倍になっても最大比較回数は 1 回しか増えません。これが 2 分探索法の特徴（そしてメリット）です。

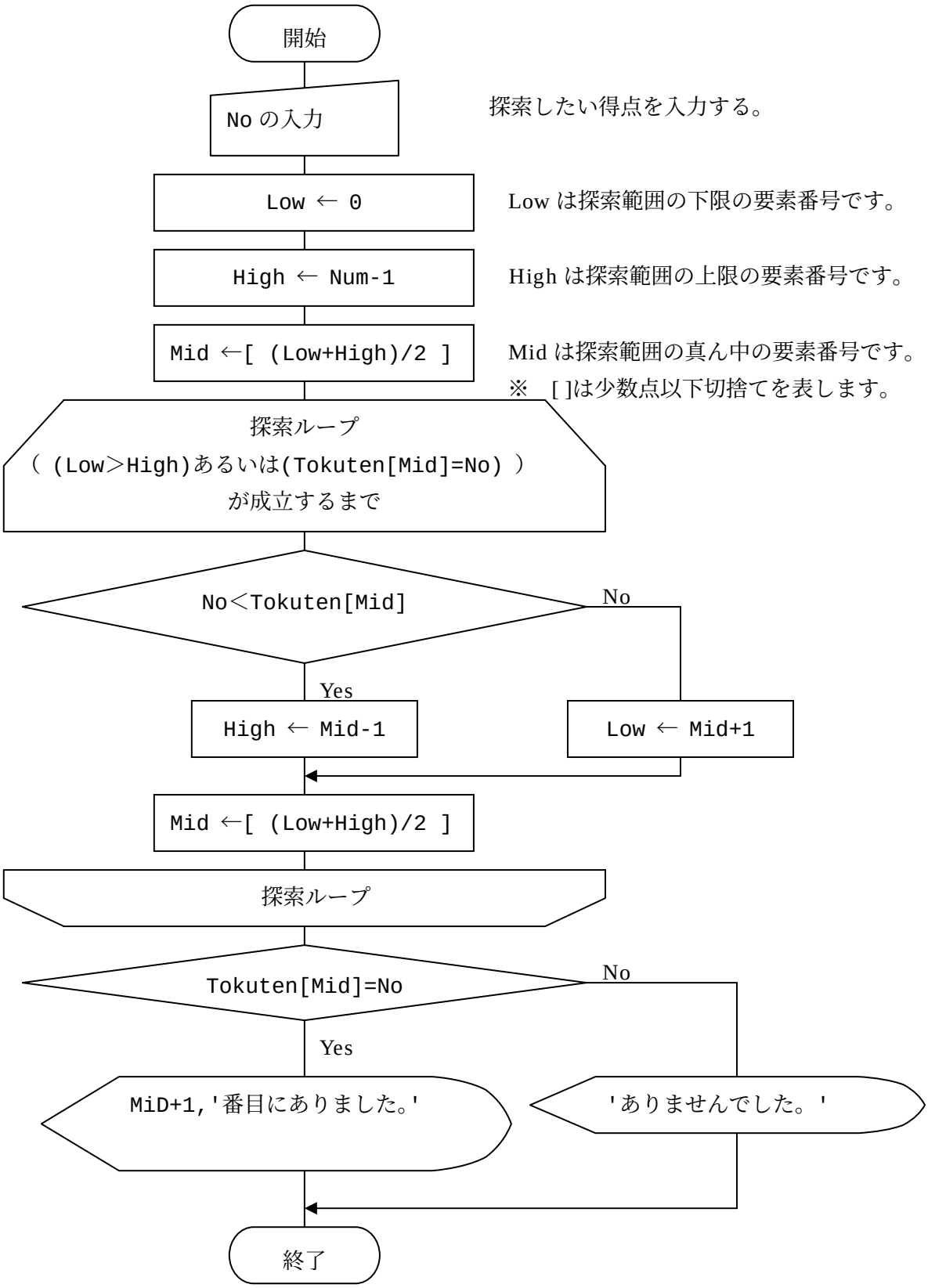
それでは、確認のためにデータ数ｎ個の時に 2 分探索で必要になる最大比較回数Ｎを実際に求めてみましょう。以下の表の空欄を埋めてください。

データ数ｎ	2	3	2^2(=4)	5	2^3(=8)	9	2^4(=16)	17
最大比較回数 N	2		3		4		5	

さて、2 分探索法の処理の流れは理解できたと思います。そこで、これを【基礎課題 7-4】と同様の問題に応用してみましょう。ただし、今度は Tokuten[0]～Tokuten[Num-1]に入っている得点は昇順にソートされているものとします。このときの処理のアルゴリズムは、次ページのようになります。じっくりと確認してください。

そして、処理の流れを確認できたら【基礎課題 7-6】へ進んでください。

＜2 分探索法＞


開始
No の入力
探索したい得点を入力する。
Low ← 0
Low は探索範囲の下限の要素番号です。
High ← Num-1
High は探索範囲の上限の要素番号です。
Mid ←[(Low+High)/2]
Mid は探索範囲の真ん中の要素番号です。
※　[ ]は少数点以下切捨てを表します。
探索ループ
（ (Low＞High)あるいは(Tokuten[Mid]=No) ）
が成立するまで
No＜Tokuten[Mid]
No
Yes
High ← Mid-1
Low ← Mid+1
Mid ←[(Low+High)/2]
探索ループ
Tokuten[Mid]=No
No
Yes
MiD+1,'番目にありました。'
'ありませんでした。'
終了

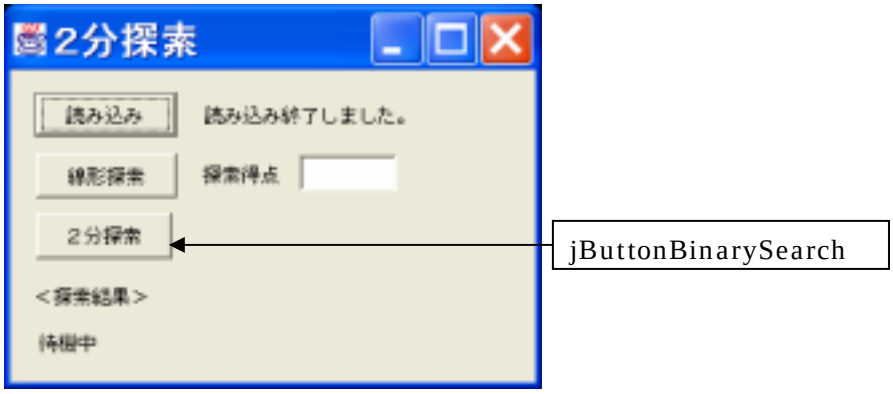
【基礎課題 7-6】

HP の該当部分から、ある科目の得点が昇順にソートされて記録されているファイル「tokuten2.txt」をダウンロードして下さい。そして【基礎課題 7-4】と全く同じ探索を、今度は 2 分探索法を用いて行うようにしましょう。

まず、【基礎課題 7-4】のプログラムを開き、次のように［2 分探索］ボタンを追加して下さい。

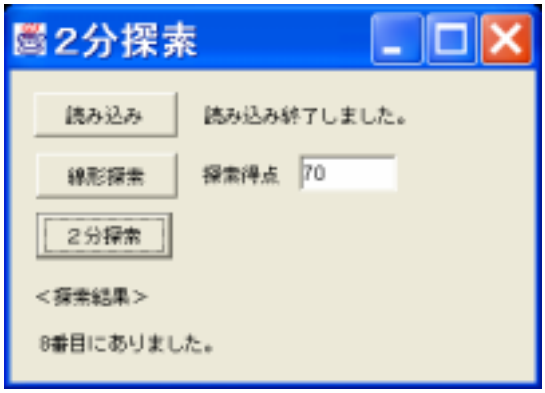
＜tokuten2.txt＞

35
39
44
48
55
59
63
70
77
87

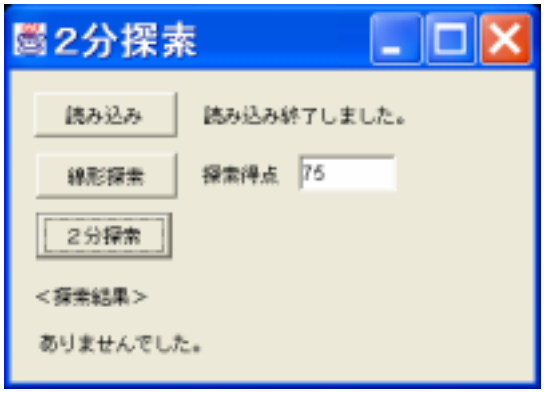


動作内容は、【基礎課題 7-4】と同様で、次の通りです。まず［読み込み］ボタンをクリックし、入力ファイルとして上の「tokuten2.txt」を指定します。その後、探索したい得点を探索得点欄に入力し［2 分探索］ボタンをクリックすると・・・

指定した得点が上から何番目にあるかを表示します。



該当する得点がない場合は、「ありませんでした。」と表示します。



それでは、プログラムを作成しましょう。新たに記述するのは、［2 分探索］ボタンのイベントハンドラのみです。次ページの空欄を埋めてプログラムを完成させて下さい。よく分からない場合は、p.115 の流れ図をよく見て下さい。

＜［2分探索］＞

```
void jButtonBinarySearch_actionPerformed(ActionEvent e) {
  int No=Integer.parseInt(jTextFieldNo.getText());
  //2分探索の処理
  int Low=0;
  int High=Num-1;
  int Mid=(Low+High)/2;
  while (  (Low<=High) && (Tokuten[Mid]!=No) ) {
    if (No<Tokuten[Mid]) {
      
    }
    else {
      
    }
    Mid=(Low+High)/2;
  }
  if(Tokuten[Mid]==No) {
    jLabelResult.setText((Mid+1)+"番目にありました。");
  }
  else {
    jLabelResult.setText("ありませんでした。");
  }
}
```

作成したら実行し、動作を確認して下さい。

７－３　アルゴリズムの効率

ｎ個のデータを探索する場合、データ照合に要する最大比較回数は、(【基礎課題 7-1】で求めた通り）線形探索法（番兵法）では、n+1 回でした。ｎが大きくなると「+1」は重要ではなくなるのでｎとしておきましょう。

それでは､2 分探索では何回になるでしょうか？すぐには求めにくいですね｡手がかりは、【基礎課題 7-5】で説明した通り、データ数ｎが 2 倍になると最大比較回数 N が一つ増えるということです。さらに、同じく【基礎課題 7-5】で気づいたと思いますが、

$2^x \leqq n < 2^{x+1}$　の時　→　$N = x+1$　　（１）

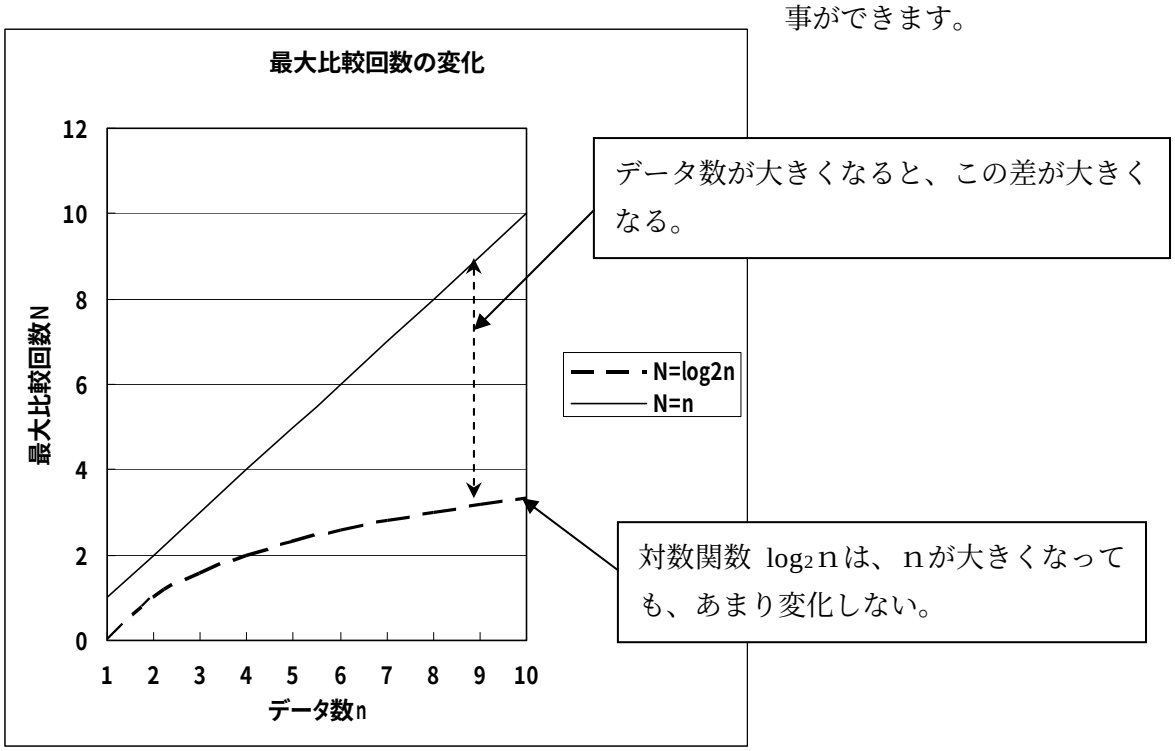
という関係があります。ここまで分かればしめたものです。あとはｎが決まったときのｘを求めればよいのです。結論を言うと、ｘは対数関数の定義より

$2^x = n$ の時　→　$x = \log_2 n$　　（２）

と表されます。対数関数に慣れていない人は戸惑うかもしれませんが、とにかく定義だと思ってください。関数の詳細は全く気にする必要はありません。ともかく、（２）で求めたｘを（１）に代入すると

$N = x + 1 = [\log_2 n] + 1$　　※［ ］は小数点以下切捨てを意味します。

と表されます。つまり、2 分探索の最大比較回数 N はデータ数ｎの増大と共に $\log_2 n$ の割合で増えて行くのです。ここで下のグラフをみてください。この対数関数は、ｎよりもかなりゆっくりと増加することが分かります。このため、データ数が大きくなると 2 分探索の方が決定的に有利になります。例えばデータ数が 1000 の場合、2 分探索法の N は 10 にしかならず、線形探索より 100 倍効率が良いことになります。このように、アルゴリズムの効率は、データ数が増大したときにどれだけ処理に要する手間が増えるか、で判断する事ができます。



７－４　応用問題

【基礎課題 7-A】

線形探索法（番兵法）を、もう少し実用的な問題に応用してみましょう。

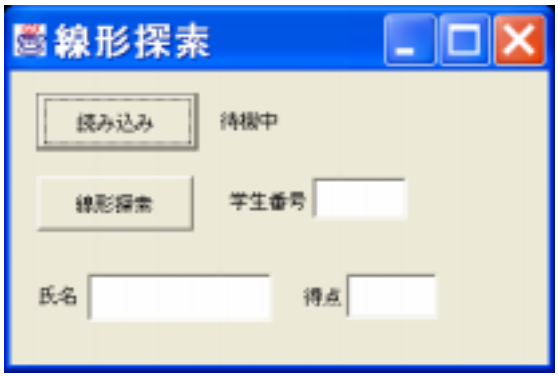
右のファイル「score4.txt」を HP の該当部からダウンロードしてください。このファイルからデータを読み込んで、指定した学生番号の学生の成績を表示させるプログラムを作成して下さい。

作成するプログラムの動作内容は次の通りです。

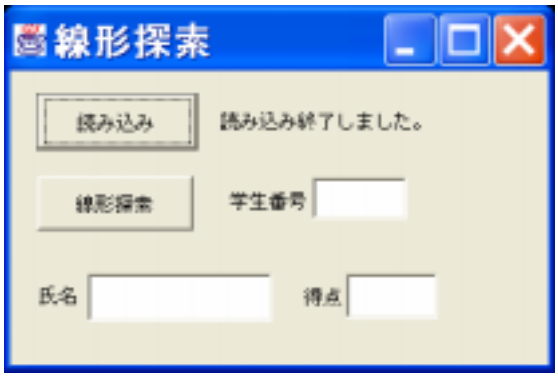
学生番号　氏名　　得点

```
151,花形　満,55
111,轟　次郎,63
101,金田正太郎,39
166,鮎原こずえ,87
134,浅丘ユミ,48
122,矢吹　丈,70
149,東　八郎,35
105,星　明子,77
167,諸星　団,59
153,伊達直人,44
```

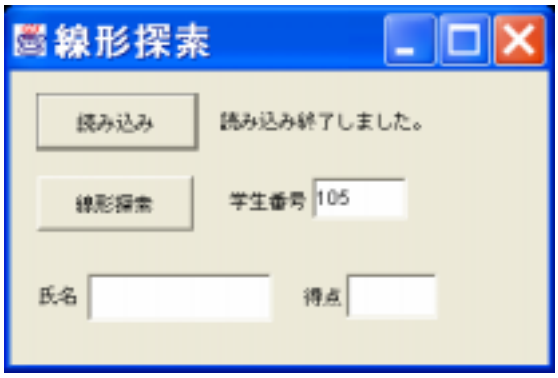
＜score4.txt＞



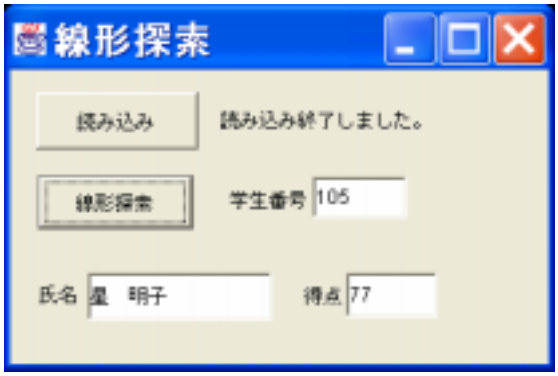
プログラムを起動すると左の画面が現れます。



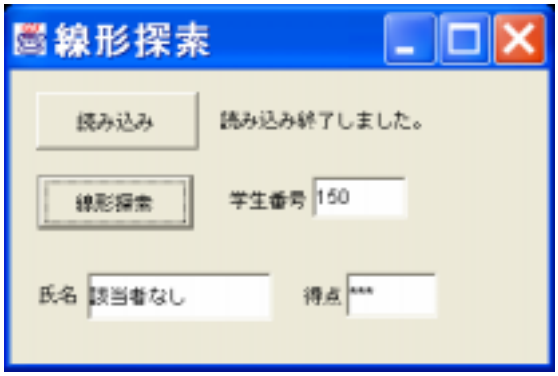
［読み込み］ボタンをクリックし、入力ファイルとして上の「score4.txt」を指定します（左は、ファイルからのデータの読み込みが終了したところ）。



成績を調べたい学生の学生番号を入力します。



［線形探索］ボタンをクリックすると、指定した学生番号の学生の氏名と得点が表示されます。



もし、指定した学生番号がなかった場合、左のように、氏名欄に「該当者なし」、得点欄に「***」と表示されます。

＜ヒント＞

① まず、ファイル中のデータ（の項目）に対応して、学生番号、氏名および得点をフィールドとして持つクラスを定義します。要領は、【基礎課題 5-1】と同様です。例えばクラスの名前を TestMeibo とすると、次のような定義になります。

```
class TestMeibo {
  private int No,Tokuten;
  private String Name;

  public TestMeibo(int Bangou,String Shimei,int Ten) {
    Name=Shimei;
    No=Bangou;
    Tokuten=Ten;
  }
  public int getNo() {
    return No;
  }
  public String getName() {
    return Name;
  }
  public int getTokuten() {
    return Tokuten;
  }
}
```

② 次に、上で定義したクラスのオブジェクトを配列として宣言します。要領は【基礎課題 5-2】（p.81 の①部分参照）と同様です。

③ ［読み込み］ボタンの処理は、【基礎課題 5-2】を参考にすると良いでしょう。ただし、そこでは、クラス TestMeibo のコンストラクタの引数は２つでしたが、今の場合、前ページのように３つあることに注意して下さい。

④ ［線形探索］ボタンの処理は、次のようになります。【基礎課題 7-4】を参考にして空欄を埋めてプログラムを完成させて下さい。ただし、以下のプログラムでは、各テキストフィールドの名前を次のようにしています。

学生番号欄→jTextFieldNo　　氏名欄→jTextFieldName

得点欄→jTextFieldTokuten

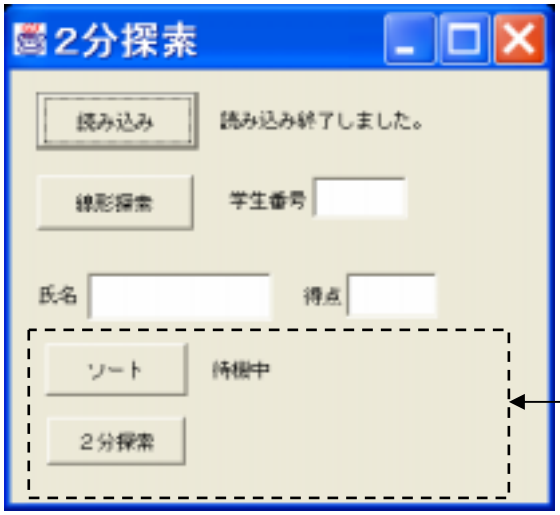
また、TestMeibo クラスのオブジェクトを Meibo1 という名前の配列として宣言しているものとしています。

```
void jButtonSearch_actionPerformed(ActionEvent e) {
  int No=Integer.parseInt(jTextFieldNo.getText());
  int i=0;
  Meibo1[Num]=new TestMeibo(No,"",0); //番兵の定義
                                      //（Num はデータの個数）
  while (Meibo1[i].getNo()!=No) {
    i++;
  }
  if(i<=      ) {
    jTextFieldName.setText(                          );
    jTextFieldTokuten.setText(
              String.valueOf(Meibo1[i].getTokuten()) );
  }
  else {
    jTextFieldName.setText("該当者なし");
    jTextFieldTokuten.setText("***");
  }
}
```

※ 番兵 Meibo1[Num]の定義を行う際、指定した学生番号 No のみを照合に用いるので、氏名と得点には何を入れても構いません（結果に関係しません）。そこで、ここでは、氏名として空白文字を、得点としては０を入れています。

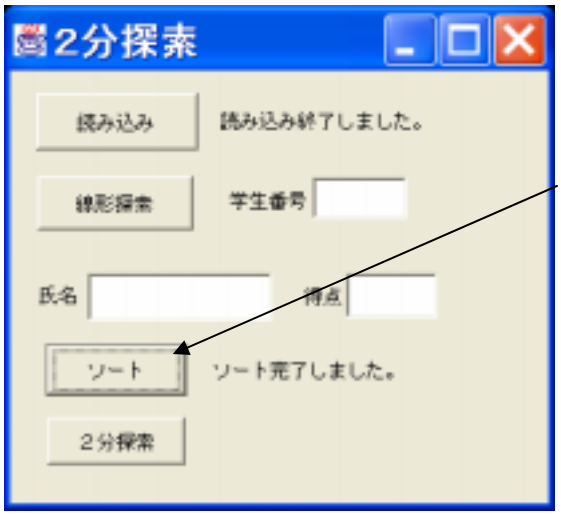
【応用課題 7-B】

【応用課題 7-A】のプログラムに、２分探索の処理を加えましょう。作成するプログラムの動作内容は次の通りです。

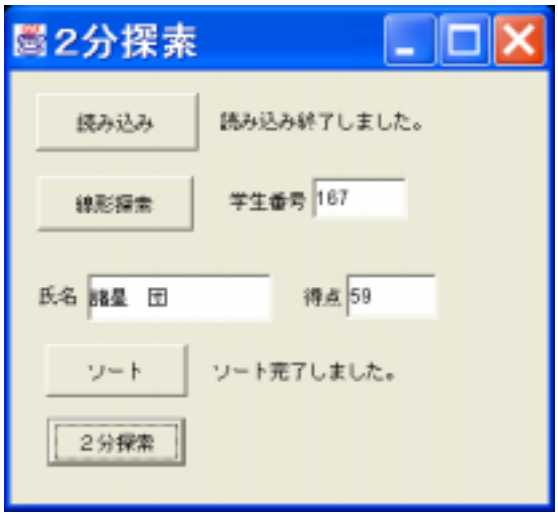


プログラム起動後、入力ファイルを指定してデータを読み込む処理部分は【応用課題 7-A】と全く同様です。

【応用課題 7-A】のプログラムに追加する部分



データ読み込み後、２分探索を行うため、［ソート］ボタンをクリックして、読み込んだデータを学生番号順にソートします。ソートとしては、挿入ソートを用いて下さい。→【基礎課題 6-6】参照。



学生番号欄に（成績を表示させたい）学生番号を入力し、［２分探索］ボタンをクリックします。すると、該当する学生の氏名と得点が（２分探索によって探索され）表示されます。
該当者がいなかった場合の表示は【応用課題 7-A】と同様とします。