第２章．配列（構造）を使った処理

【学習のねらい】

① 配列を使った（典型的な）処理を学習する。

② 処理内容に応じて配列を利用できるようになる。

２－１　配列の利点

配列とは？→ A[1], A[2], …のように、括弧内の数字（添え字）で変数の値を指定できるデータ構造。

例　１，５，３，９，７という 5 個の整数を変数に保存する

★ 一般の変数の場合：**5 個**の変数（A1，A2，…，A5）を用意する。

1	5	3	9	7
A1	A2	A3	A4	A5

例えば、「9」は変数 A4 に入っている。

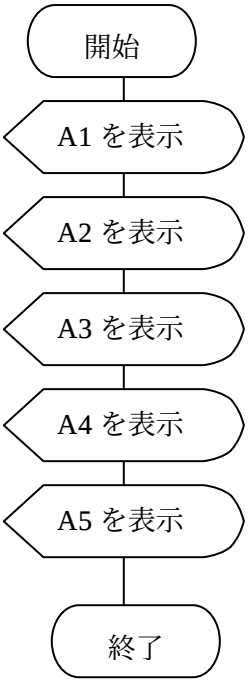
★ 配列の場合：大きさが「５」の配列変数Ａを**一個**用意する。

A	1	5	3	9	7
	[1]	[2]	[3]	[4]	[5]

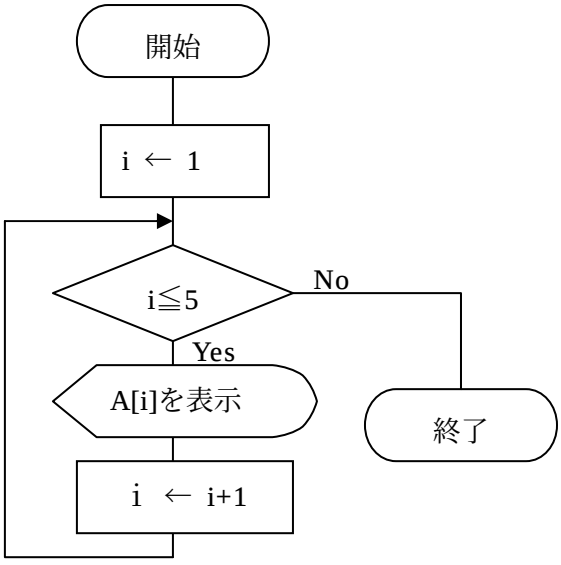
例えば、「9」は配列 A[4]に入っている。

処理の例　5 個の数値を順に表示させる。

変数 A1、A2、…、A5 に値が入っている場合。



配列 A(1)～A(5)に値が入っている場合



※ 一般に、大量のデータに同種の処理を繰り返す場合、配列（構造）を利用します。

＜　配列（利用の）利点　＞

- 拡張が容易。例えば上の例の場合、データ数が 100 になっても上の判断記号内の「５」を「100」に変えるだけで良い。
- 処理内容をコンパクトに表現できるので、見通しがよくなる。→アルゴリズムの構造がはっきりとする。

２－２　Java 言語による配列の宣言

Java 言語では、例えば 10 個の要素を持った整数型の配列 A を宣言する場合、次のように記述します。

```
int A[];
A= new int[10];
```

あるいは上の２つをまとめて以下のように一度に書くこともできます。

```
int A[] = new int[10];
```

もし、実数型で要素数を 20 個持っている配列 B を宣言する場合は、同様に

```
double B[] = new double[20];
```

のように宣言します。

何となく分かるとは思いますが、配列の宣言が単純に「int A[10]」等とはならずに、「new」演算子を用いている点が少し気になりますね。その点については、第４章のクラスに関する説明の部分で解説することにします。ここでは、上の記述方法を ” 約束事 ” だと了解しておいて下さい。ただ、「Java 言語では配列の扱いは通常の（整数型や実数型などの）変数と異なるらしい」という点は心に留めておきましょう。

補足

Java 言語の配列宣言については、次のような記述の仕方も認められています（[]が型名の方に移動しています）。

```
int[] A = new int[10];
```

というよりも、Java 言語を開発したサン・マイクロシステムズ社の開発チームは、こちらの記述の方を推奨しているようです。どちらの記述も意味は同じですが、（最初に）上で説明した記述では、例えば配列 A と普通の変数Ｂとを

```
int A[] = new int[10],B;
```

のように、一度に宣言できるというメリットがあります。そのため、こちらの記述を用い

る技術者は多いようです。とりあえず、以下では、最初に説明した

```
int A[] = new int[10];
```

の記述を用いることにします。

配列の扱いについてごく基本的な事項のみを以下にまとめておきますので確認しておいて下さい。

＜配列要素の扱いに関する基本事項＞

① 配列要素は 0 番目から始まります。

```
int A[] = new int[10];
```

と宣言した場合、配列要素として A[0]～A[9]（の 10 個の要素）が確保されます。最初の要素番号（**添え字**）が 0 から始まることに注意して下さい。

② 各要素の参照や値の代入

配列の各要素、例えば A[1]などは、通常の変数と全く同様に扱えます。例えば、次の処理が実行されると・・・

```
int A[] = new int[3];
int B;
A[0]=10;  A[1]=3;  A[2]=8;   //各配列要素へ値を代入
B=A[2];    //３番目の配列要素の値を代入
```

A[0]～A[2]の値はそれぞれ 10,3,8 になり、B の値は 8 になります。

③ 配列要素の初期化

次のような記述により、配列の宣言と初期化を同時に行うことができます。

```
int c[]={1,2,3,4,5};
```

この例では、c[0]～c[4]の値がそれぞれ、1,2,…,5 になります。配列の大きさ（要素数）は初期値の数で自動的に設定されます（上の場合は５）。

④ 範囲外の要素へのアクセスの禁止

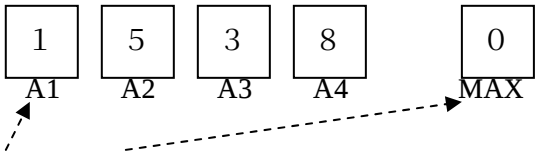
宣言された範囲外の添え字を指定した場合は、プログラム実行時にエラーとなります。例えば上の③の配列 c の場合、c[5]にアクセスしようとした場合、これは最大の添え字を越えていますからエラーになります。同様に、0 より小さい添え字を指定した場合もエラーとなります。

２－３　配列の応用－最大・最小を求める

今、４つの正の整数が変数 A1～A4 に値が入っているものとします。この中の最大値を求めるアルゴリズムを考えましょう。考え方は次の通りです。

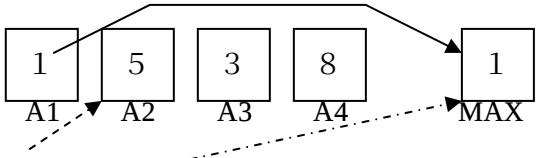
今仮に変数 A1～A4 に１，５，３，8 が入っているものとします。そして最大値を入れる変数として MAX を用意し、最初に「０」を入れておきます。以下の手順を追ってください。

０．処理開始時点



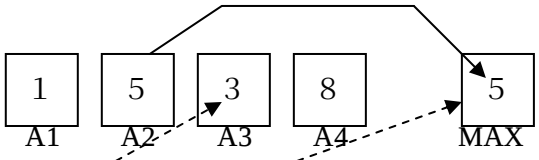
１．A1 と MAX の比較

A1 の方が大きい（A1>MAX）ので A1 の値を MAX に代入する。



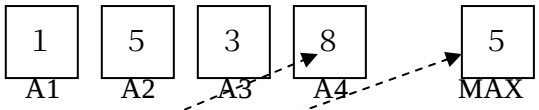
２．A2 と MAX との比較

A2 の方が大きい（A2>MAX）ので、A2 の値を MAX に代入する。



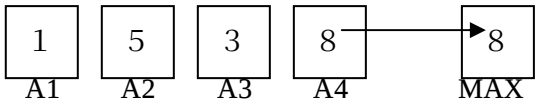
３．A3 と MAX との比較

MAX の方が大きい（A3<MAX）のでそのまま。



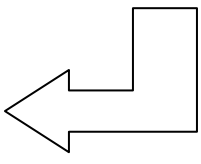
４．A4 と MAX との比較

A4 の方が大きい（A4>MAX）ので、A4 の値を MAX に代入する。



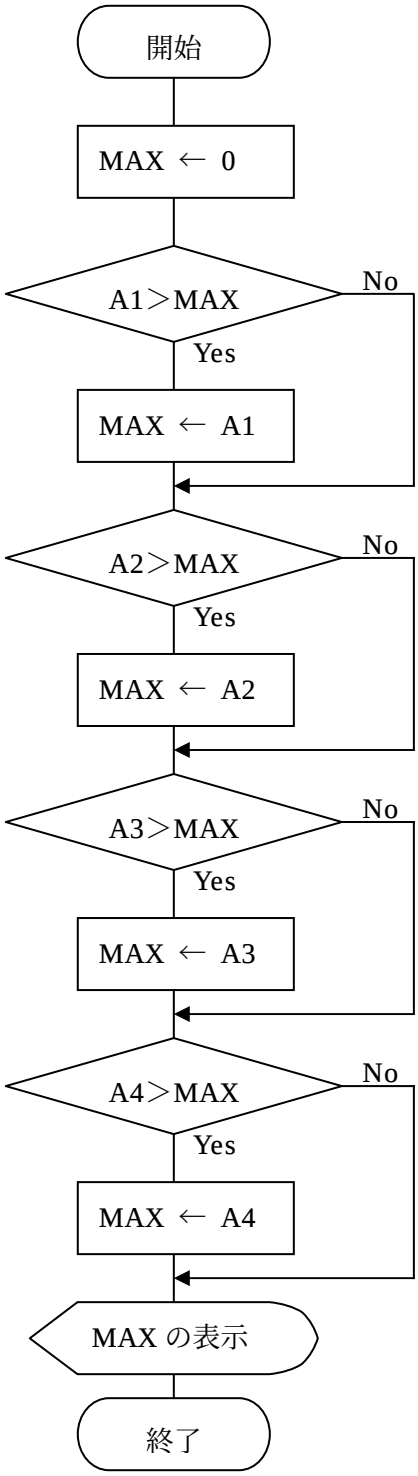
処理終了！変数 MAX に最大値「８」が入っている。

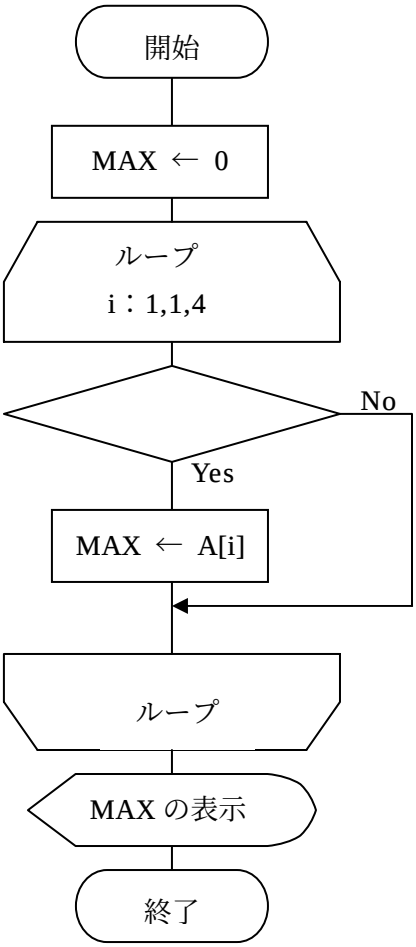
このアルゴリズムは、最大値候補（MAX）と全変数を比較し、大きい方を MAX に入れるので、「勝ち抜き戦方式」と言えます。



【基礎課題 2-1】

上の勝ち抜き戦方式を流れ図で書くと、次の左図の様になります。これを配列 A[1]～A[4] を用いて表したものが、右図です。空欄を埋めてください。





＜メリットの確認＞

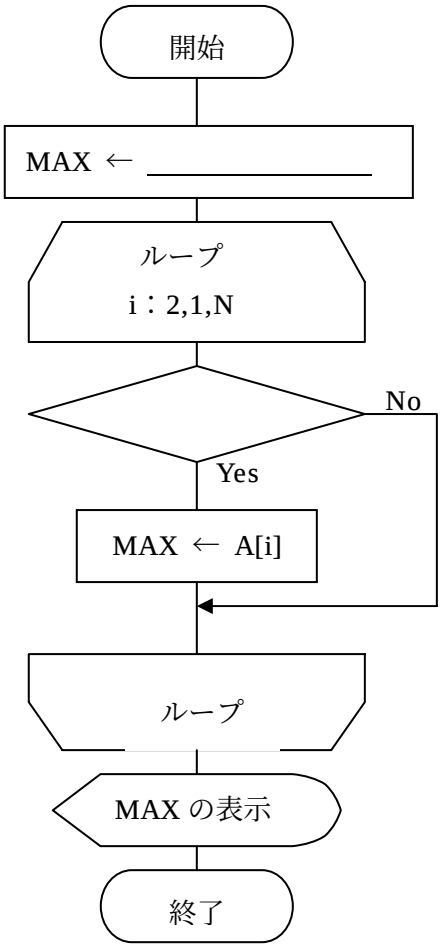
- 整数の数が 100 個の場合は、ループの中の「i：1,1,**4**」を「i：1,1,**100**」に変更するだけでよい。→ 拡張が容易。
- 配列要素を用いると、繰り返し処理を、ループの中に（一つだけ）記述するだけで済む。→ 記述がコンパクトになる。

【基礎課題 2-2】

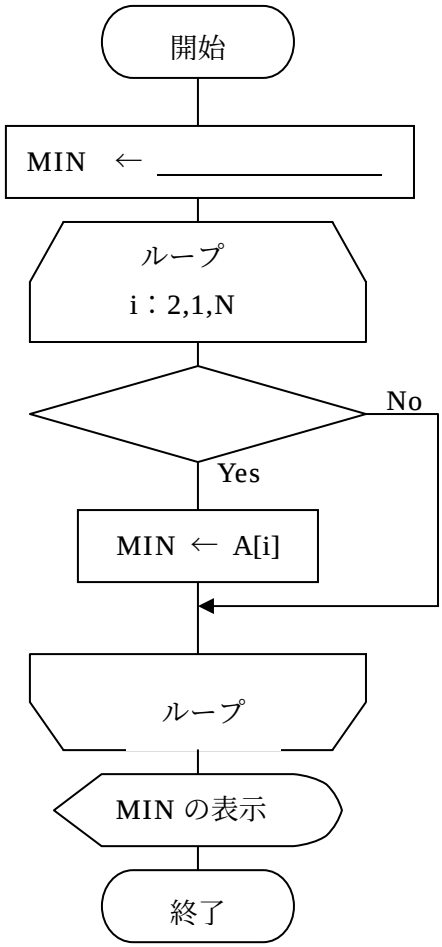
上の例題では、正の整数が入力される事を前提としているため、最大値候補として０からスタートさせることができました。しかし、一般の整数（負の場合も含む）が入力される場合は０からスタートすることはできません。例えば、入力された整数が全て負の場合、最初に入れた「0」が最大値になってしまい、データ中の最大値を求めることが出来ないからです。この場合、どのようにアルゴリズムを改良すれば良いでしょうか？

今、A[1]～A[N]の N 個の配列要素に整数（負の値も含む）が入力されている場合、最大値を求めるアルゴリズムを考えましょう。以下の左図の空欄および下線部を埋めてくアルゴリズムを完成させてください。同時に最小値 MIN を求め表示するアルゴリズムを、右図の空欄および下線部を埋めて完成させてください。

最大値を求めるアルゴリズム



最小値を求めるアルゴリズム

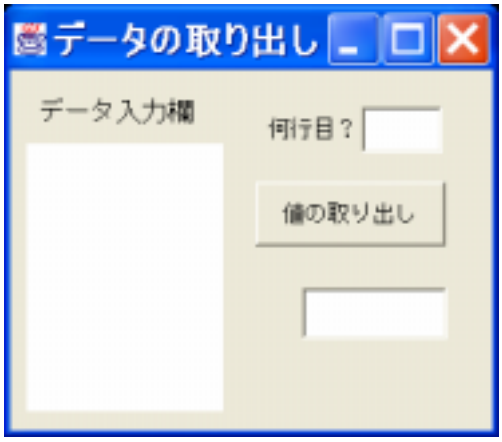


ヒント　A[1]～A[N]のいずれかを最初の最大値（最小値）の候補とすれば良いのです。そして、カウンタ i が２から始まっていることに注意して下さい。

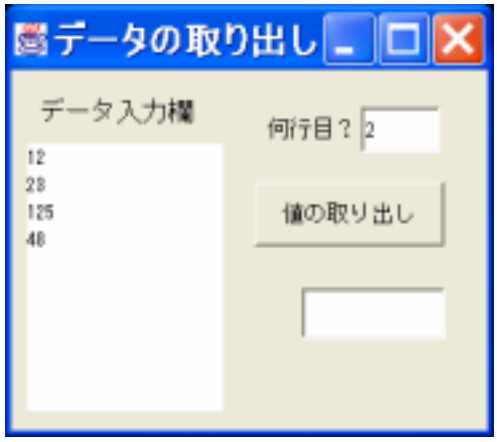
【基礎課題 2-3】

ここで、配列要素の扱いになれるために、次のようなプログラムを作ってみましょう。①プログラムを起動し、②データ入力欄に適当に整数を入力しさらに、取り出したい行番号（上から何番目か）を指定する、③［値の取り出し］ボタンをクリックすると、指定した行番号の値が表示される。

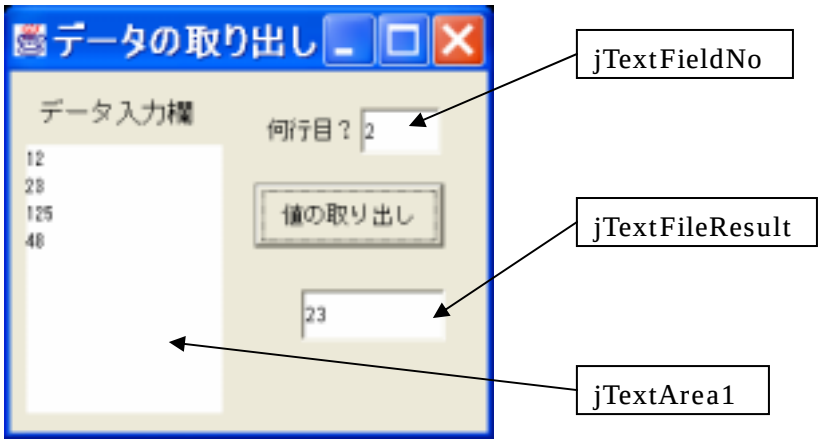
①



②



③



ここに、主要コンポーネントの name プロパティは上に示した通りです。プログラム（ボタンクリック時のイベントハンドラ）は、次ページ以降のプログラムの通り入力すれば完成するようになっています。

このプログラムでは、データ入力欄に入力したデータをそれぞれ配列 Data[0],Data[1],･･･に保管するようにしています。実はこの処理の場合、必ずしも配列を使用なくても良いのですが、ここでは、配列を用いたプログラムを経験することが目的です。作成したら実行して動作を確認して下さい。

配列の使用の仕方を理解したら、次の【基礎課題 2-4】で最大値を求めるプログラム、つまり配列を応用するプログラムに進みましょう。

＜作成するプログラム＞

```
  void jButton1_actionPerformed(ActionEvent e) {
//テキストエリアに入力した全テキストの取得
    String Text=jTextArea1.getText();                    ①
//取り出したい行番号の取得
    int No=Integer.parseInt(jTextFieldNo.getText());     ②
//テキストエリア内の入力行数の取得
    int NLine=jTextArea1.getLineCount();                 ③
//各行に入力した数値を保管する配列の宣言
    int Data[]=new int[NLine];                           ④
//  テキストエリア内のデータを配列 Data に入力する。
    for (int i=0;i<NLine;i++) {
      Data[i]=Integer.parseInt( getLine(Text,i) );       ⑤
    }
//  指定した行番号のデータを表示                          ⑥
      jTextFieldResult.setText(String.valueOf(Data[No-1]));
  }
```

＜プログラムの解説＞

① テキストエリア内に（複数行にわたって）入力した全文字列を変数「Text」に代入します。

② 「何行目か？」欄に指定した番号を整数型変数「No」に保管します。

③ テキストエリアコンポーネントは、getLineCount()メソッドを持っており、これにより、テキストエリア内の入力行数を取得できます。上の実行例では「４」となります。

④ 入力したデータを保管する整数型の配列変数「Data」を宣言します。

⑤ 今の場合、１行あたりに一つのデータを入力しているので、各行のデータを配列 Data に保管します。そこで、テキストエリア内の各行毎の文字列を取得したいのですが、残念ながらそのようなメソッドは定義されていません。そこで、点線枠で囲った getLine(Text,i)というメソッドを新たに定義しました。これは、テキストエリア内の入力文字列 Text の中の i 行目の値（文字列）を取得するメソッドです。具体的な定義は次ページに示しています。ですから、この **getLine メソッドの定義を（次ページ通り）必ず記述して下さい**。場所は、上のイベントハンドラの上方あるいは下方いずれでも結構です。

⑥ 指定した行番号のデータをテキストフィールドに表示させます。ここに、配列は 0 番目から始まっている事に注意して下さい。上から何番目か、を意味する No は１番目から数えているので、一つずらさなければなりません。そのため、配列 Data の添え字が「No-1」となっている訳です。

＜新たに定義したメソッド＞

```
  String getLine(String Text,int Line) {
    int Num=Text.length();  //文字列 Text の文字数の取得
    int N=0;                //テキストの（探索）行番号の初期化
    String Work;             //作業用文字列型変数の宣言
    String Result="";       //求める文字列を格納する変数の初期化
//  １文字ずつ探索し、改行記号(¥n)毎に文字列を区切る
    for (int i=1;i<=Num;i++) {
      Work=Text.substring(i-1,i); //文字列 Text の i 番目の文字を取得
      if (Work.equals("¥n")) {     //改行記号を見つけ場合の処理
        if(N==Line) {
          break;     //求める行番号の文字を取得したので for ループを抜ける
        }
        N=N+1;       //探索行番号を１つ増やす
        Result="";  //新たに次の行の文字を格納するため初期化する。
      }
      else {             //改行記号以外の文字の場合、文字を新たに連結する。
        Result=Result+Work;
      }
    }
    return Result;
  }
```

＜プログラムの解説＞

このプログラムの内容は配列の学習には直接関係しないので、ここではその詳細を気にする必要はありません。HP の該当部分に上のプログラムを掲載していますので、コピーして使って結構です。ただ、中身が気になるという人は以下のポイントを読んで下さい。大まかな流れは理解できるはずです。

① p.19 のようにデータを入力した場合、上の Text の中身は「12¥n23¥n125¥n48¥n」となります。ここに「¥n」は改行記号を意味します。

② そこで、この文字列 Text を左端から１文字ずつ読み取り、「¥n」で区切られる文字列毎に、文字列型変数「Result」に保管します。文字列型変数には文字列の i 番目を抜き取るメソッドが用意されており、それが substring(i-1,i)です。一般に、i 番目から N 文字抜き取る場合は substring(i-1,i+N-1)となります。

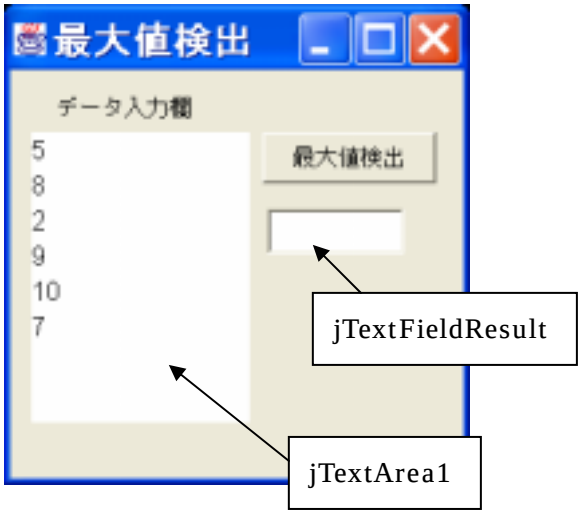
③ 文字列 Text の中で「¥n」を見つけた回数が行番号（何行目か）に対応しますから、その回数（上のプログラムでは N）が指定した行番号「Line」に等しいときの「Result」を、求める文字列として返します。

④ なお、JBuilder の編集画面では、使用フォントの関係で「¥n」が「\n」と表示されます。

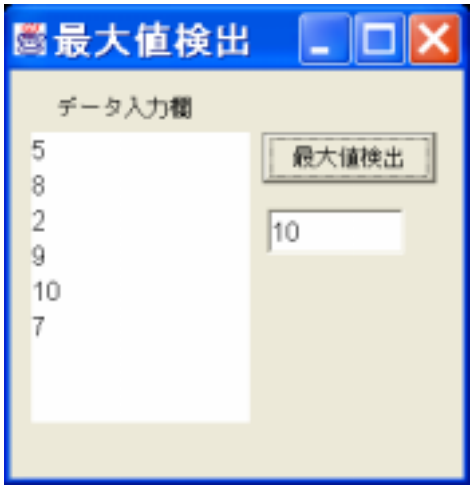
【基礎課題 2-4】

それでは、【基礎課題 2-2】で学習したアルゴリズムを用いて次のように、入力データから最大値を検出するプログラムを作成しましょう。

①プログラムを起動し、データ入力欄に任意の整数を入力する。



②　［最大値検出］ボタンをクリックすると、最大値が表示される。



ここに、主要コンポーネントの Name プロパティは上に示した通りです。プログラムの作成に当っては、次ページのプログラムの空欄（最大値検出部分）を埋めれば完成するようになっています。作成したら実行して動作を確認して下さい。

＜作成するプログラム＞

```
  void jButton1_actionPerformed(ActionEvent e) {
//テキストエリアに入力した全テキストの取得
    String Text=jTextArea1.getText();
//テキストエリア内の入力行数の取得（今の場合データの個数）
    int N=jTextArea1.getLineCount();
//各行に入力した数値を保管する配列 A の宣言
    int A[]=new int[N];
//  テキストエリア内のデータを配列 A に入力する。
    for (int i=0;i<N;i++) {
      A[i]=Integer.parseInt(getLine(Text,i));
    }
//  最大値の検出
    int Max;
```

```
    jTextFieldResult.setText(String.valueOf(Max));
  }
```

＜プログラムの解説および作成上の注意＞

① 前半部、つまり配列にデータを保管するところまでは、【基礎課題 2-3】と基本的に同じです。

② 【基礎課題 2-3】と同様、メソッド getLine(Text,i)を利用しています。この定義部分を忘れずに記述しておいて下さい。

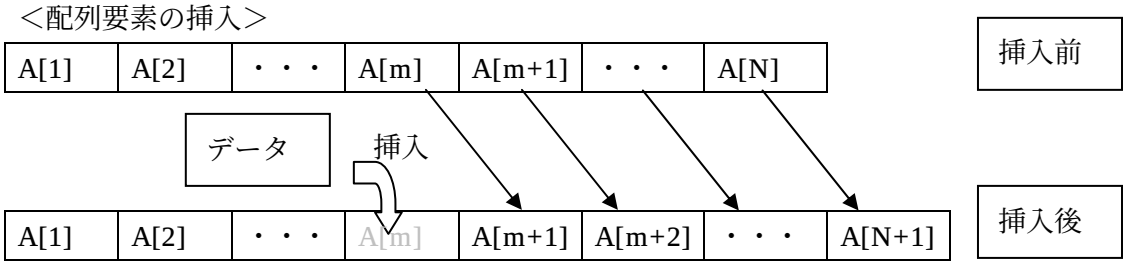
③ 空欄部に、【基礎課題 2-2】で学習したアルゴリズムを用いて、最大値 MAX を求める部分を記述します。ただし、配列は0番目から始まっていることに注意してください。【基礎課題 2-2】では、配列要素が A[1]から始まるとしていましたが、今は A[0]から始まっているので、カウンタ i の値を一つずつ、ずらす必要があります。

２－４　配列の挿入・削除

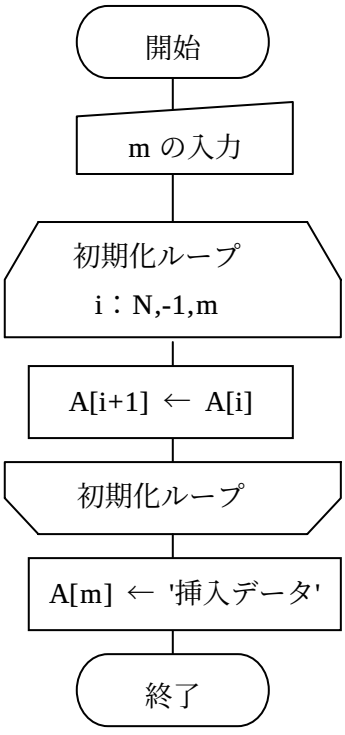
ここでは、配列にデータを挿入したり削除したりする方法を学習します。実際のプログラムではしばしば必要になる処理です。

今、配列 A[1]～A[N]に（何らかの）データが入っているものとします。この配列要素の m 番目に新たなデータを挿入するものとします。つまり、元のm番目以降は一つずつ後ろに順番がずれて、全部で N+1 個の配列要素となります（配列の大きさは少なくとも N+1 よりも大きくとっているものとします)。

＜配列要素の挿入＞



【基礎課題 2-5】



上の（配列要素の）挿入処理のアルゴリズムは左の通りとなります。これを確認した上で、下の設問に答えてください。

【設問】

左の処理は、

- A[N+1]←A[N]
- A[N]←A[N-1]
- ・・・
- A[m+1]←A[m]

という様に、配列要素の後ろから順に代入（入れ替え）を行っています。どうして A[m]から始めないのでしょうか？その理由を以下に記述してください。記述後、補助員あるいは指導員にチェックを受けてください。不十分な解答の場合は、やり直しを指示します。OK が出るまで修正してください。

※　トレースを行えばわかるはずです。

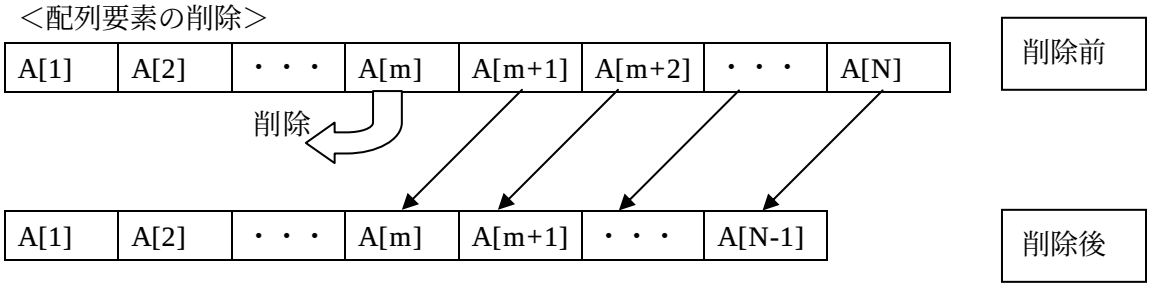
＜理由＞

A[m+1]←A[m]から始めると

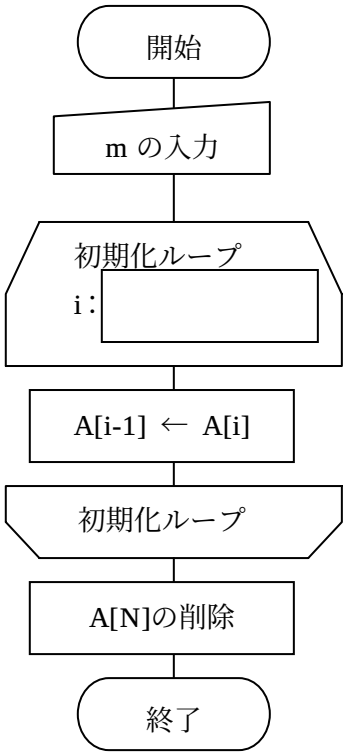
から

【基礎課題 2-6】

配列要素の削除については、次のような処理を行います。

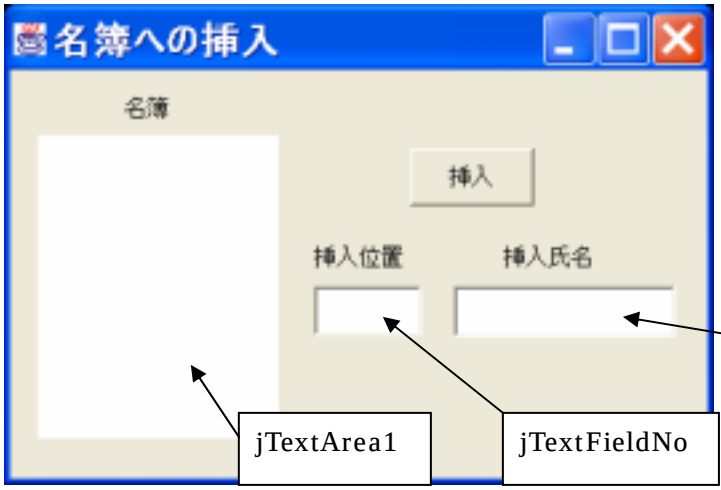


この削除処理のアルゴリズムは下の様になります。空欄を埋めてアルゴリズムを完成させてください。

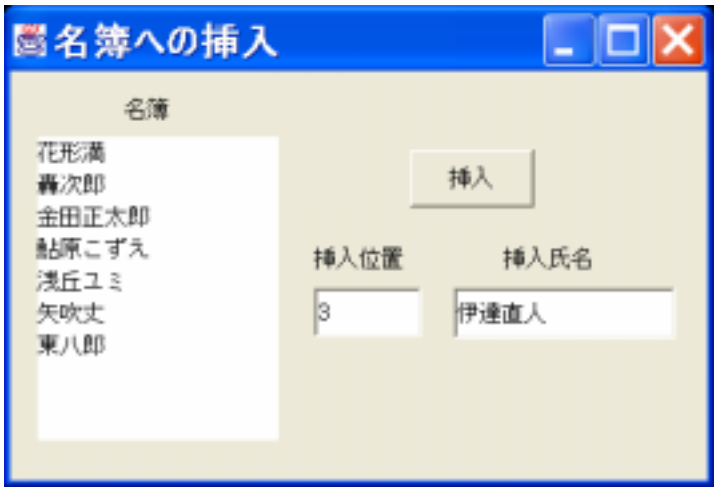


【応用課題 2-A】

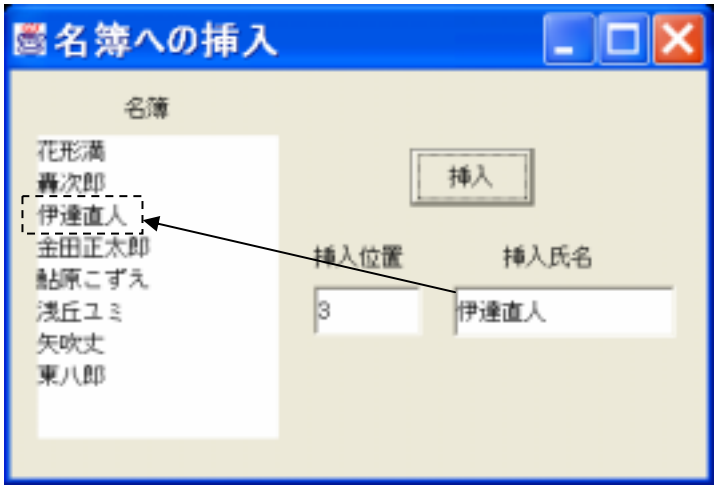
以下の様に、挿入を行うことが出来る名簿（プログラム）を作成しましょう。動作内容は次の通りです。



まず、プログラムを起動すると、左のような名簿が現れます。
枠内は、主立ったコンポーネントの name プロパティです。



ここで、名簿欄に適当な氏名を入力します。その後、「挿入位置」欄に挿入したい位置（上から何番目か）、「挿入氏名」欄に挿入したい氏名を入力します。



ここで、［挿入］ボタンをクリックすると、指定した位置に「挿入氏名」が挿入されます。

このプログラムを作成すると次のようになります。挿入処理の部分の空欄を埋めてプログラムを完成させてください。作成後は必ず動作を確認して下さい。

＜作成するプログラム＞

```
  void jButton1_actionPerformed(ActionEvent e) {
    int No=Integer.parseInt(jTextFieldNo.getText()); //挿入位置
    String Name=jTextFieldName.getText();            //挿入氏名
    String Text=jTextArea1.getText();  //テキストエリア内の全テキスト
    int N=jTextArea1.getLineCount();   //データの個数（入力行数）
//  名簿を保管するための配列 Meibo の宣言（とりあえず大きさを 50 としている）
    String Meibo[]=new String[50];
//  テキストエリア内のデータを配列 Meibo に入力する。
    for (int i=0;i<N;i++) {
      Meibo[i]=getLine(Text,i);
    }
//  No 番目に挿入氏名を挿入する。

    for (int i=N-1; [          ] ;i--){

      [                        ]

    }
    Meibo[ [          ] ]=Name;

//  更新した名簿を表示する
    Text="";
    for (int i=0;i<N+1;i++) {
      Text=Text+Meibo[i]+"¥n";
    }
    jTextArea1.setText(Text);
  }
```

＜プログラムの解説および作成上の注意＞

① 【基礎課題 2-3】と同様、メソッド getLine(Text,i)を利用しています。このメソッドの定義部分を忘れずに記述しておいて下さい。

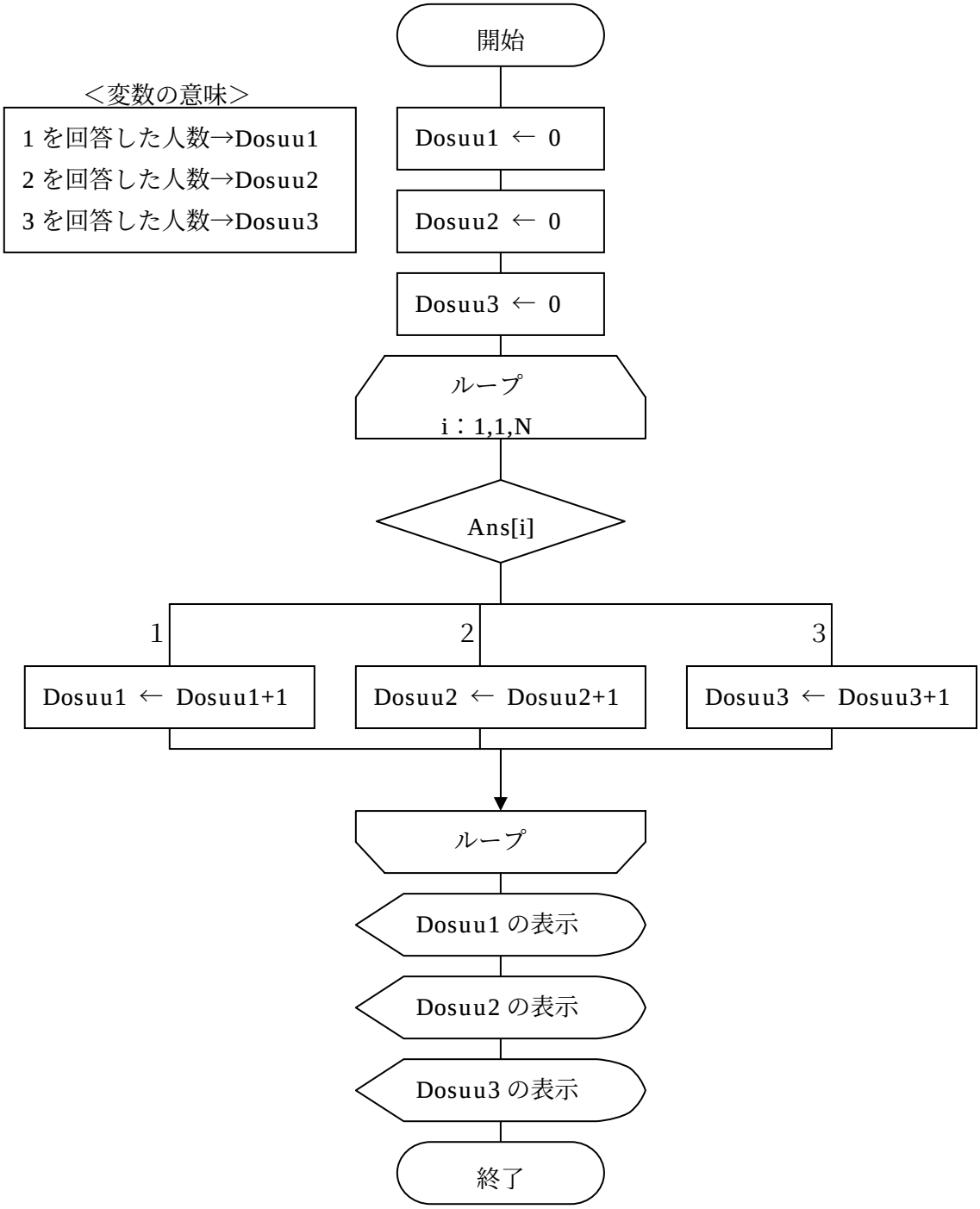
② これまで同様、配列は 0 番目から始まっていることに注意してください。つまり、プログラムでは Meibo[0]～Meibo[N-1]の N 個の配列を扱っていますが、【基礎課題 2-5】では、Meibo[1]～Meibo[N]を用いた流れ図になっています。添え字が一つずれていることに注意して下さい。

③ 更新した名簿を表示する部分では、「氏名+改行記号（¥n）」を順次連結させ、その結果をテキストエリアに表示させています。処理内容がよく分からない人は、p.24 の getLine()メソッドの解説部分を参照して下さい。

２－５　添え字の参照

配列要素の添え字を参照することで、処理が簡単になる事がよくあります。これも配列を用いるメリットの一つです。まずは、具体例でそのメリットを体験してみましょう。

あるアンケートを実施しました。その回答の選択肢は１～３までの３つ、つまり 3 択の設問でした。今、N 人分の回答が、配列 Ans[1]～Ans[N]に入っているものとします。この場合、回答の度数分布を集計するアルゴリズムは以下の通りとなります。



前頁をみると、回答 Ans[i]の値に応じて以下の処理を行っています。

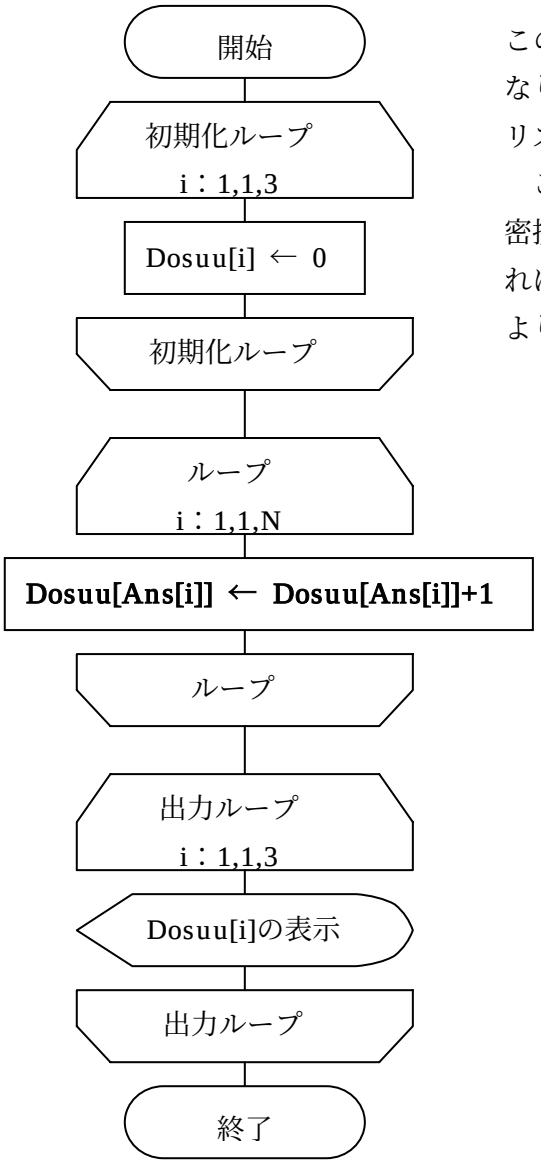
Ans[i]の値	1	2	3
処理	Dosuu1←Dosuu1+1	Dosuu2←Dosuu2+1	Dosuu3←Dosuu3+1

ここで、Dosuu を配列にすると、上の表は次の様に表せます。

Ans[i]の値	1	2	3
処理	Dosuu[1]←Dosuu[1]+1	Dosuu[2]←Dosuu[2]+1	Dosuu[3]←Dosuu[3]+1

これをよく見ると、上の 3 つの処理は一まとめにして、次のように表せることが分かります。

Dosuu[Ans[i]] ← Dosuu[Ans[i]]+1



このことを利用すればアルゴリズムは左の通りとなり、分岐処理は必要なくなります。つまりアルゴリズムは、より簡明になります。

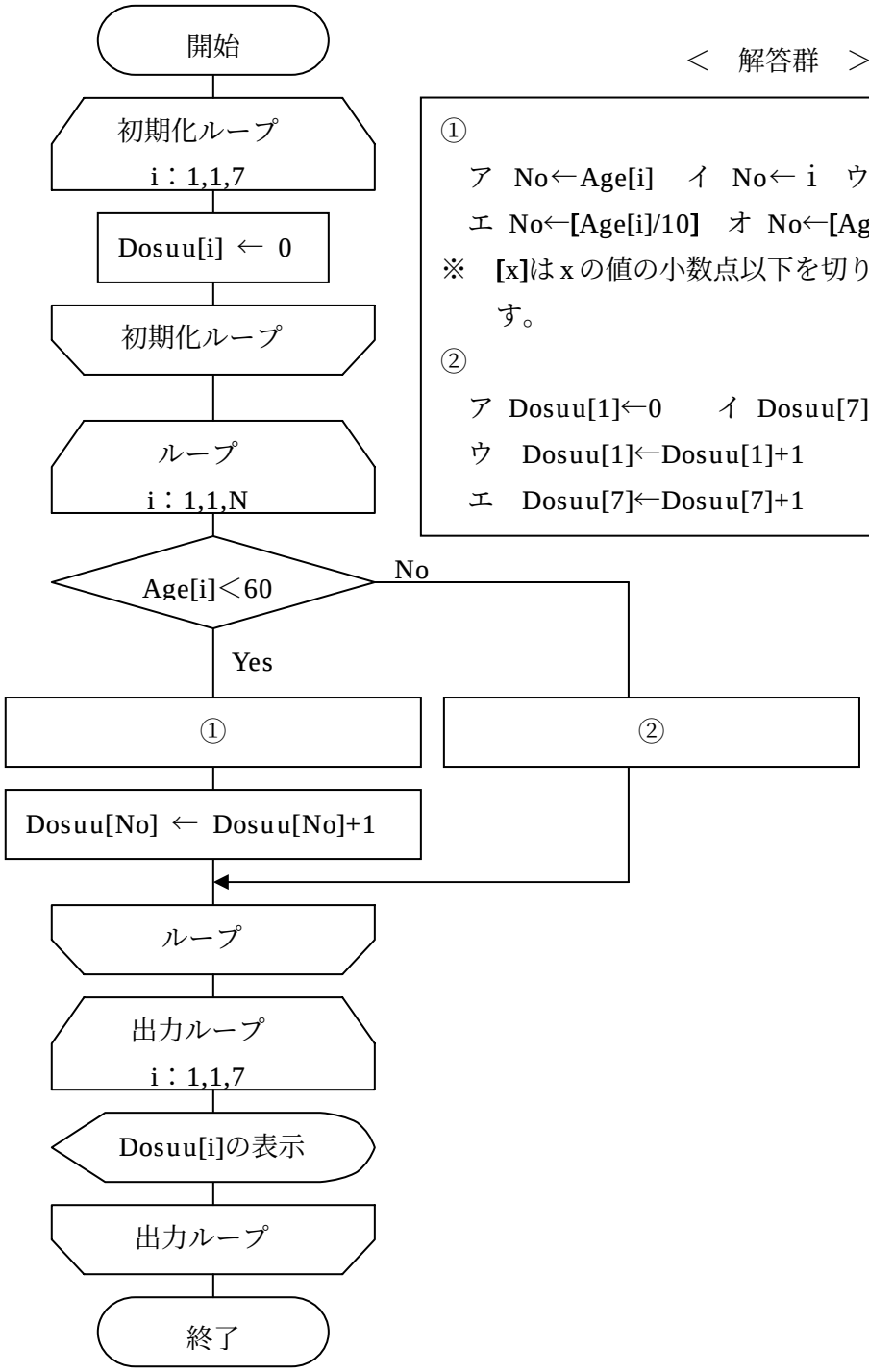
　このように、一般にデータ構造とアルゴリズムは密接に結びついており、与えられた問題に対してそれに適したデータ構造を用いれば、アルゴリズムがより簡明になる、という関係があります。

【基礎課題 2-7】

N 人の年齢調査を行い､各人の年齢が配列 Age[1]～Age[N]に入っているものとします。今、このデータを用いて、10 代、20 代・・・毎に何名ずついるか、年代の度数分布をとり、その各度数を、下表の様に配列 Dosuu[1]～Dosuu[7]に保管するものとします。

年齢	0～9	10～19	20～29	30～39	40～49	50～59	60 以上
度数	Dosuu[1]	Dosuu[2]	Dosuu[3]	Dosuu[4]	Dosuu[5]	Dosuu[6]	Dosuu[7]

下の空欄にあてはまる処理を解答群から選択し、アルゴリズムを完成させてください。



＜　解答群　＞

①

ア　No←Age[i]　イ　No← i　ウ　No←[Age[i]/10]+1

エ　No←[Age[i]/10]　オ　No←[Age[i]/10]-1

※　[x]は x の値の小数点以下を切り捨てる事を意味します。

②

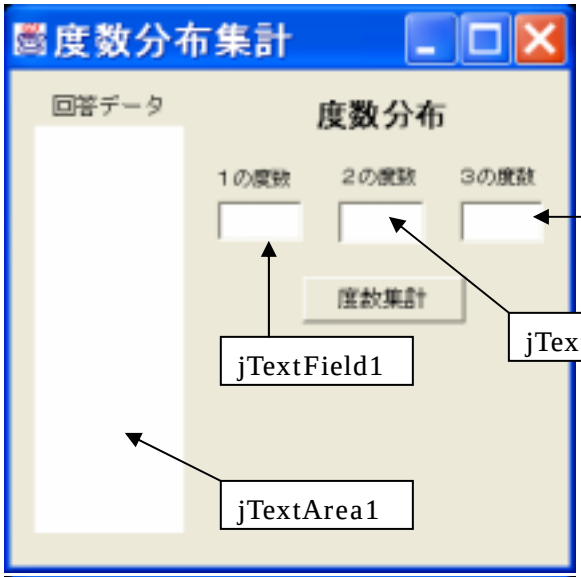
ア　Dosuu[1]←0　　イ　Dosuu[7]←0

ウ　Dosuu[1]←Dosuu[1]+1

エ　Dosuu[7]←Dosuu[7]+1

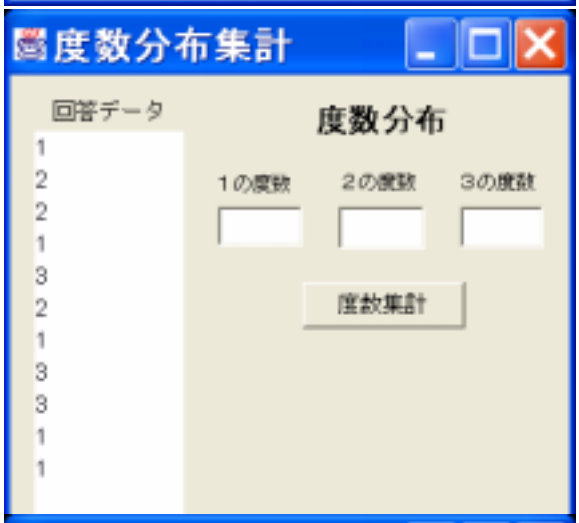
【応用課題 2-B】

p.29 のアルゴリズムを用いて、3 択のアンケートの回答（1～3）を入力すると、それぞれの回答の度数を求めるプログラムを作成しましょう。動作内容は次の通りです。

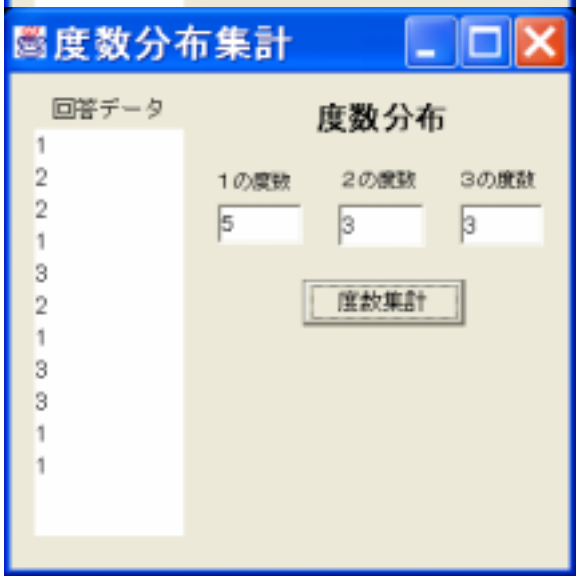


プログラムを起動すると次の画面が現れます。

※　白枠内は、コンポーネントの name プロパティです。



回答データ欄にアンケートの回答を入力します。



［度数集計］ボタンをクリックすると、各々の回答の度数が表示されます。

このプログラムは以下の通りとなります。「度数分布の集計」部分を埋めてプログラムを完成させてください。作成後は実行して必ず動作確認を行って下さい。

＜作成するプログラム＞

```
  void jButton1_actionPerformed(ActionEvent e) {
    String Text=jTextArea1.getText(); //テキストエリア内の全テキスト
    int N=jTextArea1.getLineCount(); //データの個数
    int Ans[]=new int[N];    //回答データを保管する配列
    int Dosuu[]=new int[3]; //回答の度数を保管する配列
//  テキストエリア内のデータを配列 Ans に入力する。
    for (int i=0;i<N;i++) {
      Ans[i]=Integer.parseInt(getLine(Text,i));
    }
//  度数分布の集計
```

```
//  集計結果の表示
    jTextField1.setText(String.valueOf(Dosuu[0]));
    jTextField2.setText(String.valueOf(Dosuu[1]));
    jTextField3.setText(String.valueOf(Dosuu[2]));
  }
```

＜プログラムの解説および作成上の注意＞

① 【基礎課題 2-3】と同様、メソッド getLine(Text,i)を利用しています。このメソッドの定義部分を忘れずに記述しておいて下さい。

② p.29 の流れ図では、Dosuu[1]～Dosuu[3]としていましたが、今は添え字が 0～2 になっている事に注意してください。

③ 同様に、配列 Ans は Ans[0]～Ans[N-1]のように、0 番目から始まっていることに注意してください。