2013 年度 前期 電子計算機 1

更新日時 2013-07-25 00:29:32　担当 和地 輝仁

目 次

1　シラバス抜粋

授業の目標　コンピュータを用いて文書を作成したり、資料の整理を行うために、いくつかのアプリケーションソフトを利用できるようになる。また、効率を上げるためにプログラムを利用したり、論理構造を重視した文書の作成ができるようになる。

到達目標

1. 基本的なプログラムの作成方法を学び、作業の効率化を図れる。
2. 文書組版システムを用いて、論理構造をもつ文書の作成ができる。
3. コンピュータを用いた学習教材の作成ができる。

授業計画　順序を交換する場合もあるので注意すること。

1. 表計算ソフトウェアによる統計処理
2. プログラム
3. 変数・演算
4. 関数とプロシージャ
5. 制御構造 1
6. 制御構造 2
7. 制御構造 3
8. 文書組版システム TeX の導入と使用方法
9. 文書の構造
10. 表・箇条書き
11. 数式
12. 相互参照・目次・索引
13. 画像
14. マクロ
15. 期末試験

成績評価　期末試験では、プログラムの作成と、文書組版システムを用いた文書の作成ができるかどうかを評価する (50%)。各回の授業では演習問題を

出題し、その提出状況と内容も評価する (50%)。原則として全ての時間の出席を求めるが、やむを得ない理由で欠席をする (した) 場合はできるだけ速やかに申し出て、指示を受けること。

2 授業のノート (Small Basic 編)

§1 表計算ソフトウェアによる統計処理

(1.1) **課題 01 – 課題提出の練習** 以下のことを実行して下さい。

(1) 小手調べに「メモ帳」を起動して、適当な俳句を詠んで入力する。
(2) 「0000 和地-01」のような「学生番号名前-課題番号」というファイル名[1]で保存する。数字は半角。
(3) 「提出用ページ」[2] をウェブブラウザで開き、「課題提出」の所に配布したパスワードを入力し、提出するファイルを選択する。そして「提出」ボタンを押すと提出が完了する。
(4) 「提出用ページ」の「提出状況閲覧」の所に、パスワードと学生番号を入力し、「閲覧」ボタンを押すとこれまでのすべての提出状況が見られる。

また、提出した課題に修正を求められたときなどは、修正後のファイルを 1 回目と同様に提出して下さい。同一ファイル名でも区別して管理されるため、「提出状況閲覧」から提出状況を見ると、2 回提出したこともわかります。

(1.2) **課題の合否と再提出** 「提出状況閲覧」では、提出したファイルが課題番号ごとに一覧できます。複数回提出した課題では、新しいファイルほど下に表示されます。ファイル名の右に「合格」とあれば、その提出課題は合格です。「再提出」とあれば、修正して再提出して下さい。

(1.3) **Excel の式** Excel のセルには、= で始まる**式**を入力することができます。

Excel の式の文法

= 式

次の課題では「IF」という関数を用います。

[1] テキストファイルの拡張子「.txt」も含めれば、「0000 和地-01.txt」となる
[2] URL は板書します

IF の文法

```
IF(論理式, 真の場合, 偽の場合)
```

使用方法は、Excel のヘルプでも見られます。

式にはセルの値も利用できるので、例えば、「`= IF(A1 > 3, "3 より大きい", "小さい")`」のような式が書けます。

(1.4) **課題 02 − Excel でプログラムの初歩** 次の内容を記載した Excel のファイルを提出して下さい。ファイル名は、「0000 和地-02」のような「学生番号名前-課題番号」というファイル名にします。

(1) 「提出用ページ」から、テキストファイル「grades.txt」をダウンロードし、その内容を、エクセルのシートの A 列の A1 から A100 までに埋める (コピーした後のコピー領域右下のアイコンを使うのが楽)。
(2) A 列が 60 点以上ならば B 列に「合格」、そうでなければ「不合格」と表示する **IF を用いた式**を B 列に入力する。例えば、A1 のセルが 70 ならば B1 のセルに「合格」、A2 のセルが 50 ならば B2 のセルに「不合格」と表示されるような **IF を用いた式**を B1 から B100 までに入力する。
(3) A 列の数値が、90 を越えていたら「!」(半角感嘆符) と表示されるような **IF を用いた式**を C1 から C100 までのセルに入力する。90 以下のときは何も表示しなくてよい。
(4) A 列の数値が、10 以下ならば「SS10」 (すべて半角) と表示されるような **IF を用いた式**を D1 から D100 までのセルに入力する。
(5) A 列の数値が、7 の倍数ならば「なな」と表示されるような **IF を用いた式**を E1 から E100 までのセルに入力する。

完成したシートは以下のようになる。ただし、B 列から E 列までは、このように手で入力するのではなく、このように表示される **IF を用いた式**を入力すること。

.	A	B	C	D	E
1	3	不合格		SS10	
2	49	不合格			なな
3	63	合格			なな
4	44	不合格			
5	84	合格			なな
6	65	合格			
7	94	合格	!		
8	22	不合格			
9	21	不合格			なな
:	:	:	:	:	:

§2 最初の Small Basic プログラム

(2.1) **インストールと起動** Microsoft のサイト `http://smallbasic.com` に、Small Basic のインストーラ `SmallBasic.msi` があります。これをダウンロードして実行すると Small Basic をインストールできます[3]。

Small Basic を起動するには、「スタートメニュー - すべてのプログラム - Small Basic - Microsoft Small Basic」と辿ります。

(2.2) **最初のプログラム** 起動すると図のようなウィンドウが現れます。ウィンドウ上部にはボタンが配置された「**ツールバー**」があり、その下にはプログラムを入力する「**エディタ**」があります。

実際に次の 1 行からなるプログラムをエディタに入力してみます。これは、テキストウィンドウに文字列を表示して改行する命令です。

最初のプログラム

```
1 TextWindow.WriteLine("スイトピー赤と答えて年がばれる")
```

[3] Windows のみで実行可能です。また、Windows XP など古い OS だと、「.NET Framework 3.5 SP1」もインストールする必要があります。

すると入力中に、候補を表示したメニューが現れます。**インテリセンス**と呼ばれる入力補助で、クリックで候補を選択し、ダブルクリックで確定できます。また、TAB キーで候補を補完することもできます。

プログラムを**実行**するには、ツールバーの実行ボタンを押します。実行すると、テキストウィンドウを呼ばれるウィンドウが開いて[4]そこに俳句が表示されます[5]。

出力

```
スイトピー赤と答えて年がばれる
Press any key to continue...
```

プログラムを**保存**するには、ツールバーの保存ボタンを押します。保存すると、4 つのファイルが作成されます。

- `.sb` ファイル: プログラム本体です。メモ帳でも開けます。
- `.exe` ファイル: 実行ファイルです。ダブルクリックで起動します。
- `.pdb` ファイル: 各種データを保存しています。
- `.dll` ファイル: `.exe` ファイルの実行に必要なものです。

このうち、`.dll` ファイルは、複数のプログラムで共用できますので、同じフォルダに複数のプログラムを保存しても、`.dll` ファイルは 1 つだけ作成されます。

(2.3) **保存したプログラムの開き方** プログラムの書かれた`.sb` ファイルは、特定のアプリケーションに関連づけられていなければ、ダブルクリックしても開くことができません。Small Basic から開きたければ、Small Basic のメニューの「ファイル - 開く」から開けます。また、メモ帳から開きたければ、メモ帳のメニューの「ファイル - 開く」のダイアログで、ファイルの種類を「すべてのファイル」に変更すれば開けます。

.sb ファイルをダブルクリックして Small Basic で開くようにしたければ、.sb ファイルを Small Basic に関連づけるとよいです[6]。

[4] Small Basic のウィンドウの背面に隠れているかも知れません。
[5] 終了するには、何かのキーを押します。
[6] 方法は授業で言うかも知れません

(2.4) **課題 03 – Small Basic で俳句** 何か俳句を考え、次の出力例のように俳句を表示するプログラムを書き、`.sb` ファイルを提出して下さい。つまり、「`0000 和地-03.sb`」のようなファイルを提出して下さい。

出力例

```
スイトピー
赤と答えて
年がばれる
```

(2.5) **構造「順次」** 上の課題のように、複数の文を連続して実行するプログラムの構造を「**順次**」と呼びます[7]。

順次

```
文1
文2
:
```

(2.6) **コメント** プログラムの途中に 1 重引用符「'」を置くと、そこから行末までが無視されます。これを利用して、プログラムの中にコメントを書くことができます。例えば、次の項では、コメントを利用して説明を記しています。

(2.7) **文、オブジェクト、メソッド、プロパティ** 下のプログラムの各部分は、**オブジェクト**、**メソッド**、**入力** (**引数**とも言う) と呼ばれます。

用語の説明

```
1 TextWindow.WriteLine("スイトピー赤と答えて年がばれる")
2 '      ↑            ↑            ↑
3 'オブジェクト   メソッド      入力 (引数とも言う)
```

[7] これだけでは何が大事かわからないですが、順次はプログラムにおいて大事な 3 つの構造のうちの (最も簡単な) 1 つです。残り 2 つもそのうち紹介します。

また、ドットの後には、**プロパティ**と呼ばれるものも来ます。例えば、TextWindow オブジェクトは、文字の色を表す ForegroundColor プロパティを持ち、下のようにすると文字の色[8] を変更してから文字列を表示します。

プロパティ

```
1 'オブジェクト　プロパティ
2 '　　↓　　↓
3 TextWindow.ForegroundColor = "Green"
4 TextWindow.WriteLine("スイトピー赤と答えて年がばれる")
```

まとめると次のようになります。

- 文: 命令の最小単位。上の例では 1 行が 1 文だが、複数行に渡ることもある。
- オブジェクト: 操作の対象となるもの。ドットの前にある。
- メソッド: 操作の名前。ドットの後にある。
- 入力 (引数): メソッドに与えるデータ。丸かっこでくるむ。
- プロパティ: オブジェクトの持つ状態の名前。ドットの後にある。

(2.8) **演算子**　Small Basic では、次の演算子が使えます。商は、9 / 4 の結果が 2.25 になるなど、小数点以下も割り算を続ける商です。また、整数の商の演算子は用意されていませんが、Math.Remainder(割られる数，割る数) というメソッドで計算できます[9]。

数値演算子	説明
+	和
-	差
*	積
/	商

比較演算子	説明
=	等しい
<>	等しくない
>	(左辺が) 大きい
>=	大きいか等しい
<	小さい
<=	小さいか等しい

論理演算子	説明
And	かつ
Or	または

文字列演算子	説明
+	連結

(2.9) **演算の例**　以下に演算の例を示します。TextWindow.Write メソッドは、TextWindow.WriteLine メソッドと同じくテキストウィンドウに文字列を表示しますが、表示した後に改行しない点が異なります。また、今までも既に用いてきましたが、文字列は 2 重引用符 (") で囲む必要があります。

演算の例

```
1 TextWindow.Write("1 + 2 = ")
2 TextWindow.WriteLine(1 + 2)
3 TextWindow.Write("文字列のa+文字列のb = ")
4 TextWindow.WriteLine("a" + "b")
5 TextWindow.Write("文字列の1+文字列の2 = ")
6 TextWindow.WriteLine("1" + "2")
```

出力

```
1 + 2 = 3
文字列のa+文字列のb = ab
文字列の1+文字列の2 = 3
```

3 行目の演算結果が、「12」にならない点は注意が必要です[10]。

[8] Green の他に使えるものは、Black, Blue, Cyan, Gray, Magenta, Red, White, Yellow, DarkBlue, DarkCyan, DarkGray, DarkGreen, DarkMagenta, DarkRed, DarkYellow です。

[9] これらの表以外でよく使われる整数の商を求める演算子はなく、メソッドも用意されていません。また、論理否定 (not) もありません。

[10] Perl などもこのような結果になりますが、利点より害が大きい気がします。

(2.10) **課題 04 – 演算** 上の例にならって、下の出力例のような出力を行うプログラムを作成し、.sb ファイルを提出して下さい。つまり、「0000 和地-04.sb」のようなファイルを提出して下さい。ただし、左辺は単に文字列を出力し、右辺は Small Basic の演算子を用いて計算した結果を出力すること。また、文字の色は、1 行目は White、2 行目は Green、3 行目は Blue、4 行目は Cyan で表示すること。

出力例

```
電子 + 計算機 = 電子計算機
37 * 3 = 111
100 / 8 = 12.5
30を7で割った余り = 2
```

§3 変数

(3.1) **変数** **変数**とは、数値や文字列の値を保存しておく入れ物です。変数名は、If などの Small Basic のキーワードでない限り、先頭が英字、その後は英数字であれば自由に付けられます。

変数

```
1 a = 3
2 b = "c"
3 TextWindow.WriteLine(a)
4 TextWindow.WriteLine("a")
5 TextWindow.WriteLine(b)
```

出力

```
3
a
c
```

このように、2 重引用符で囲むと文字列そのものを意味し、囲まなければ、その名前の変数を意味します。

(3.2) **キーボードからの入力** TextWindow.Read メソッドを実行すると、キーボードからの入力待ちになり、何か入力して Enter を押すと、メソッドは入力した文字列を値として返します。これをメソッドの**戻り値**と呼びます。

キーボードからの入力

```
1 TextWindow.WriteLine("俳句を入力して下さい")
2 a = TextWindow.Read()
3 TextWindow.WriteLine("入力は、" + a)
```

入力を促されたときに、「ぶらんこに乗りて笑いて漕ぎもせず」と入力したならば、次のような画面になります。

出力例

```
俳句を入力して下さい
ぶらんこに乗りて笑いて漕ぎもせず
入力は、ぶらんこに乗りて笑いて漕ぎもせず
```

ただし、上の出力例では、2 行目は実行時に人間がタイプしたものです。

(3.3) **課題 05 – 変数とキーボードからの入力** 次のような動作をするプログラムを作成し、.sb ファイルを提出して下さい。

(1) 「大事なことを入力」と表示する。
(2) キーボードからの入力をする
(3) 「大事だから 1 度しか言いません。○○○○○」と表示する (○○○○○は入力した文字列)
(4) 「さらに大事なことを入力」と表示する。
(5) キーボードからの入力をする
(6) 「×××××、×××××。大事だから 2 度言いました。」と表示する (×××××は入力した文字列)

出力例

```
大事なことを入力
```

```
燕は春の季語です
大事だから1度しか言いません。燕は春の季語です
さらに大事なことを入力
蜥蜴は夏
蜥蜴は夏、蜥蜴は夏。大事だから2度言いました。
```

ただし、上の出力例では、2 行目と 5 行目が人間が実行時にタイプしたものです。

(3.4) **キーボードからの数値の入力** キーボードから、文字列ではなく数値を入力したいときは、TextWindow.ReadNumber メソッドを用います。このとき、数字以外は入力できません。このメソッドは入力した数値を戻り値として返します。

キーボードからの数値の入力

```
1 TextWindow.WriteLine("何か数値を入力して下さい")
2 a = TextWindow.ReadNumber()
3 TextWindow.WriteLine("入力の2倍は、" + a*2)
```

入力を促されたときに、「12」と入力したならば、次のような画面になります。ただし、2 行目の「12」は人間がタイプしたものです[11]。

出力例

```
何か数値を入力して下さい
12
入力の2倍は、24
```

(3.5) **課題 06 – 変数とキーボードからの数値の入力** 次のような動作をするプログラムを作成し、.sb ファイルを提出して下さい。

(1) 「A さんは分速 72m、それより遅れて、B さんは分速 80m で歩き始めました。」と表示する。

(2) 「何分遅れたか入力」と表示する。

(3) キーボードから数値の入力をする

(4) 「B さんが出発してから○分後に、× m 先で A さんに追い付きます。」と表示する (○と×は計算してその値を表示させる)。

10 と入力したときの出力

```
Aさんは分速72m、それより遅れて、Bさんは分速80m
で歩き始めました。
何分遅れたか入力
10
Bさんが出発してから90分後に、7200m先でAさんに追
い付きます。
```

23 と入力したときの出力

```
Aさんは分速72m、それより遅れて、Bさんは分速80m
で歩き始めました。
何分遅れたか入力
23
Bさんが出発してから207分後に、16560m先でAさんに
追い付きます。
```

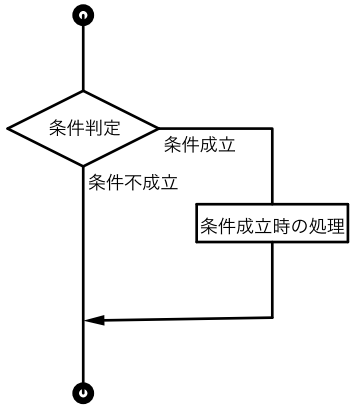
ただし、上の出力例では、3 行目は人間がタイプしたものです。また、紙面の都合で文の途中で折り返してありますが、プログラムではそうする必要はありません。

§4 条件分岐

(4.1) **If 文** ある条件が成立するときのみ、何かの処理をしたい場合 **If 文** を用います。

If 文の流れ図

[11] このプログラムでは、文字列と数値の和を計算していますが、文字列どうしの連結のように計算されています。これは、あまり良い挙動ではない気がします。



分岐（If..Then..EndIf）

```
If 条件 Then
  条件成立時の処理（複数行でもよい）
EndIf
```

次の例では、Math.GetRandomNumber(100) というメソッドで、1 から 100 までの乱数を取得して、変数 point に代入して表示し、その値が 60 以上ならば「合格」などと表示します。

If 文の例

```
1 point = Math.GetRandomNumber(100)
2 TextWindow.WriteLine(point)
3 If point >= 60 Then
4   TextWindow.WriteLine("合格")
5   TextWindow.WriteLine("おめでとう")
6 EndIf
```

出力例その 1

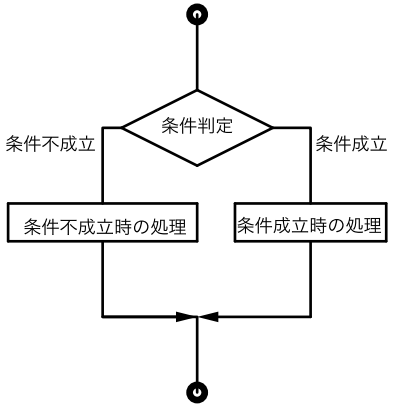
```
95
合格
おめでとう
```

出力例その 2

```
59
```

(4.2) **Else 節** If 文では、条件が成立したときだけでなく、Else を用いて不成立の場合も処理をさせることができます。

If･･･Else の流れ図



分岐（If..Then..Else..EndIf）

```
If 条件 Then
  条件成立時の処理
Else
  条件不成立の時の処理
EndIf
```

構造「順次」は既に見ましたが、この If 文の構造は「分岐」と呼ばれます。

Else 節の例

```
1 t = TextWindow.ReadNumber()
2 If t >= 25 Then
3   TextWindow.WriteLine("夏")
4 Else
5   TextWindow.WriteLine("冬")
6 EndIf
```

出力例

```
24
冬
```

ただし、上の出力例では、1 行目は人間がタイプしたものです。

(4.3) **プログラムのフォーマット** `If t >= 25 Then` まで入力して、Enter を入力すると、カーソル位置は行頭に戻るのではなく、スペース 2 つ分インデントした位置に来ます。

`Then` の後に `Enter` を入力したときのカーソル位置

```
If t >= 25 Then
  _ ←ここ
```

「条件成立時の処理」がひと固まりのブロックとして判別し易いように、ブロックごとインデントするのが作法とされているからです。編集作業の結果、インデントが狂ってしまっても、エディタのウィンドウを右クリックして「プログラムのフォーマット」を選択すると、プログラム全体のインデントを適切に修正してくれます。

(4.4) **課題 07 − 条件分岐その 1** 次のような動作をするプログラムを作成し、`.sb` ファイルを提出して下さい。

(1) 「馬の年齢を入力」と表示する。
(2) キーボードから数値の入力をする
(3) 人間の年齢に換算すると何歳かを「人間だと○歳」と表示する。

ただし、馬の年齢 x を人間の年齢に換算するには次の式を用います。

$$(\text{人間に換算した年齢}) = \begin{cases} 6x & (x < 3) \\ 3x + 8 & (x \geq 3) \end{cases}$$

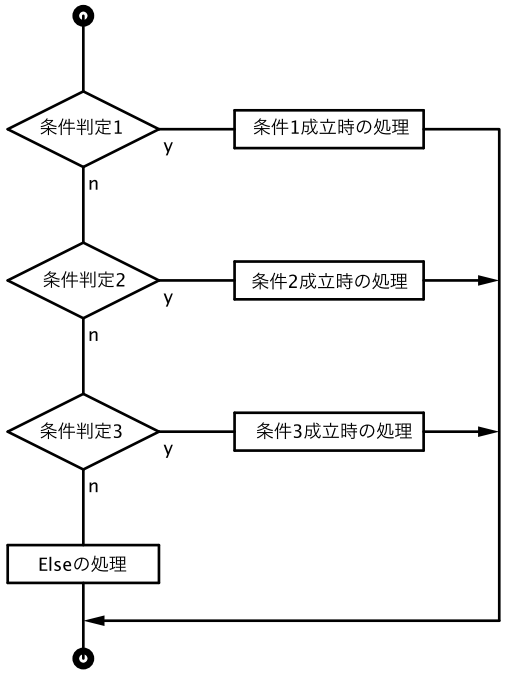
出力例

```
馬の年齢を入力
5
人間だと23歳
```

ただし、上の出力例では、2 行目は人間がタイプしたものです。

(4.5) `ElseIf` **節** `If` 文で `ElseIf` を用いると、いくつかの場合分けをして処理をさせることができます[12]。

`If`･･･`ElseIf`･･･`Else` の流れ図



分岐 (`If..Then..ElseIf..Else..EndIf`)

```
If 条件A Then
```

[12] ここに示したものが、`If` 文の完全な文法です。`ElseIf` は何回出現しても (しなくても) 構いませんし、`Else` は最後に 1 回あるか、ないかのいずれかです。

```
  条件A成立時の処理
ElseIf 条件B Then
  条件B成立時の処理
:
Else
  どの条件も不成立の時の処理
EndIf
```

複数の条件が成立するような場合でも、最初に成立した条件の処理だけを実行します。

下の例で、Clock.Hour メソッドは、現在時刻の時間の数値を 24 時制で返します。その値に応じて表示する文字列を変えています。

ElseIf 節の例

```
1 h = Clock.Hour
2 If h <= 7 Then
3   TextWindow.WriteLine("睡眠")
4 ElseIf h >= 9 And h < 18 Then
5   TextWindow.WriteLine("大学")
6 Else
7   TextWindow.WriteLine("暇")
8 EndIf
```

16:30 に実行した場合の出力例

```
大学
```

(4.6) **課題 08 − 条件分岐その 2** 次のような動作をするプログラムを作成し、.sb ファイルを提出して下さい。ElseIf を最低 1 回は用いること。

(1) 「点数を入力」と表示する。
(2) キーボードから数値の入力をする
(3) 点数が 60 以上ならば「合格」と表示する。
(4) 点数が 60 未満ならば「不合格」と表示する。
(5) 点数が 90 以上ならば「A」と表示する。
(6) 点数が 50 以上 60 未満ならば「F*」と表示する。

出力例その 1

```
点数を入力
70
合格
```

出力例その 2

```
点数を入力
95
合格
A
```

出力例その 3

```
点数を入力
45
不合格
```

ただし、上の各出力例では、2 行目は人間がタイプしたものです。

(4.7) **課題 09 − 最大値** 次のような動作をするプログラムを作成し、.sb ファイルを提出して下さい。**ただし、Math.Max メソッドは用いないこと**[13]。

(1) 「数値を 3 つ入力」と表示する。
(2) キーボードから数値の入力を 3 回行う。
(3) 3 つの数値の最大値を求め、「最大値 = ○○」と表示する。

出力例

```
数値を3つ入力
1
6
-3
最大値 = 6
```

ただし、上の出力例では、2, 3, 4 行目は人間がタイプしたものです。

[13] Math.Min メソッドも用いないこと。

(4.8) **課題 10$^{\#}$ – 最大値 (義務ではない)** 課題 09 と同じ動作をするプログラムを作成し、.sb ファイルを提出して下さい。今度は、If 文は用いず、Math.Max メソッドを用いて作成すること。

(4.9) **課題 11$^{\#}$ – 最大値 (義務ではない)** 課題 09 と同じ動作をするプログラムを作成し、.sb ファイルを提出して下さい。ただし、Math.Max メソッドは用いてはならず、If 文は 2 つしか用いてはいけない。Else や ElseIf も用いないこと[14]。

§5 For 文による繰り返し

(5.1) **For 文** 同じような処理を繰り返す場合は **For 文**を用います。同じ俳句を 18 回表示したいときは、同じ命令を 18 回続けて書いてもよいですが、For 文を用いると次のように書けます。

For の例

```
1 For i = 1 To 18
2   TextWindow.WriteLine("茨豌豆むく背を見れば思い出す")
3 EndFor
```

i を**カウンタ変数**と呼び、この変数が**開始値** 1 から**終了値** 18 まで変わりながら、For と EndFor の間が繰り返されます。

繰り返し (For)

```
For 変数 = 開始値 To 終了値
  繰り返す処理
EndFor
```

次のように、繰り返す処理中でカウンタ変数を使用できます。

For 文でのカウンタ変数の利用

```
1 For j = 3 To 5
2   TextWindow.WriteLine(j + "番射出")
3 EndFor
```

出力

```
3番射出
4番射出
5番射出
```

(5.2) **課題 12 – For 文** 次のような動作をするプログラムを For 文を用いて作成し、.sb ファイルを提出して下さい。

(1) 「1 の 3 乗は 1 です」と表示する。
(2) 「2 の 3 乗は 8 です」と表示する。
(3) 「3 の 3 乗は 27 です」と表示する。
(4) 以下順に表示し続け、最後に「55 の 3 乗は 166375 です」と表示する。

出力例

```
1の3乗は1です
2の3乗は8です
3の3乗は27です
(中略)
54の3乗は157464です
55の3乗は166375です
```

(5.3) **Step$^{\#}$** For 文で Step を用いると、カウンタ変数の増分を指定できます。

Step

```
For 変数 = 開始値 To 終了値 Step 増分
  繰り返す処理
EndFor
```

[14] 実際には、If 文、Math.Max メソッド、Math.Min メソッドのいずれも使わずに最大値を求められます。

Step の例

```
1 For j = 3 To 8 Step 2
2   TextWindow.WriteLine(j + "番射出")
3 EndFor
```

出力

```
3番射出
5番射出
7番射出
```

(5.4) **課題 13# − Step (義務ではない)** 次のような出力をするプログラムを作成し、.sb ファイルを提出して下さい。

出力例

```
1の平方根は1です
4の平方根は2です
7の平方根は2.64575131106459です
10の平方根は3.16227766016838です
(中略)
55の平方根は7.41619848709566です
```

平方根の計算には、Math.SquareRoot メソッドを用います。

(5.5) **For 文の利用** 繰り返しの処理で変数の状態を変更することで、さまざまな処理が可能になります。

For 文の利用

```
1 sum = 0
2 For k = 1 To 100
3   sum = sum + k*k
4 EndFor
5 TextWindow.WriteLine("合計 = " + sum)
```

上のプログラムの sum = sum + k*k は、左辺の変数 sum の中身を、右辺の式 sum + k*k の値で置き換えるという意味です。つまり、For 文の処理を 1 回するごとに、変数 sum の値が、k*k 加算されたものに更新されていくため、合計として表示されるのは、$1^2 + 2^2 + \cdots + 100^2 = 338350$ です。

出力

```
合計 = 338350
```

(5.6) **課題 14 − 数列の和** 次のような動作をするプログラムを作成し、.sb ファイルを提出して下さい。

(1) 「数値を入力」と表示する。
(2) 数値を入力する (入力した数値をここでは λ とおく)。
(3) 1 から λ までの逆数の和を計算する。
(4) 求めた和を「合計 = ○○」と表示する。

出力例

```
数値を入力
82
合計 = 4.990020079909081716378927 1602
```

ただし、上の出力例では、2 行目は人間がタイプしたものです。また、変数の名前に全角文字である「λ」を使うのは望ましくないので、適当な名前を自分で付けて下さい。適切な変数名を付けることも、読み易いプログラム作成のためには大事です。

(5.7) **課題 15# − 階乗 (義務ではない)** 次のような動作をするプログラムを作成し、.sb ファイルを提出して下さい。

(1) 「数値を入力」と表示する。
(2) 数値を入力する。
(3) 入力した数値の階乗を計算して、例えば「6 の階乗は 720 です」と表示する。

出力例

```
数値を入力
10
10の階乗は3628800です
```

ただし、上の出力例では、2 行目は人間がタイプしたものです。

(5.8) **課題 16$^{\#}$ − フィボナッチ数列 (義務ではない)** 次のような動作をするプログラムを作成し、.sb ファイルを提出して下さい。

(1) 「数値を入力」と表示する。
(2) 数値を入力する (入力した数値をここでは λ とする)。
(3) フィボナッチ数列の第 1 項から第 λ 項までを表示する。

ここで、フィボナッチ数列は、漸化式

$$F_1 = 1,$$
$$F_2 = 1,$$
$$F_{n+2} = F_{n+1} + F_n$$

で定められる。

出力例

```
数値を入力
10
1 1 2 3 5 8 13 21 34 55
```

ただし、上の出力例では、2 行目は人間がタイプしたものです。

§6 While 文による繰り返し

(6.1) **While 文** 条件成立の間処理を繰り返すには **While 文**を用います。For 文との違いは、

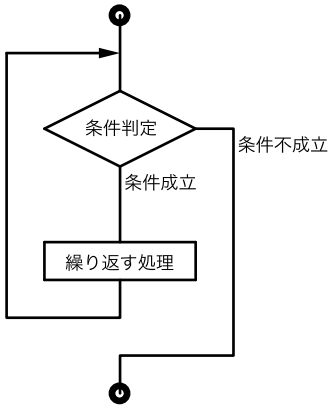
- For 文はあらかじめ反復回数が決まっているが、
- While 文は、反復回数が事前にはわからない

という点です。次のような構造を「反復」と呼びます[15][16]。

繰り返し (While)

```
While 条件
  条件成立の間繰り返す処理
EndWhile
```

While 文の流れ図



次の例では、貯金の合計が 100 円を超えると、繰り返しを終了します。

While の例

```
1 sum = 0
2 While sum <= 100
3   TextWindow.WriteLine("貯金せよ")
4   money = TextWindow.ReadNumber()
5   sum = sum + money
6 EndWhile
7 TextWindow.WriteLine(sum + "円貯まりました")
```

下の実行例では、「人間の入力」とある行は、人間がタイプしたものです。

[15] For 文も広い意味での「反復」の構造と見なせます。
[16] 最近は避けられる傾向にもありますが、処理の流れを**流れ図** (**フローチャート**) で表すと便利なことがあります。If 文などの条件判定はひし形で、通常の処理は長方形で図示し、処理の流れを線で結びます。

出力例

```
貯金せよ
30                    ’人間の入力
貯金せよ
45                    ’人間の入力
貯金せよ
30                    ’人間の入力
105円貯まりました
```

(6.2) **課題 17 – 割れるだけ割る** 次のような動作をするプログラムを作成し、.sb ファイルを提出して下さい。

(1) 「数値を入力」と表示する。
(2) 数値を入力する (入力した数値をここではλとする)。
(3) $\lambda = 2^n \cdot a$ (a は奇数) と書いたときの a を、「λを 2 で割れるだけ割ると a です」と表示する。

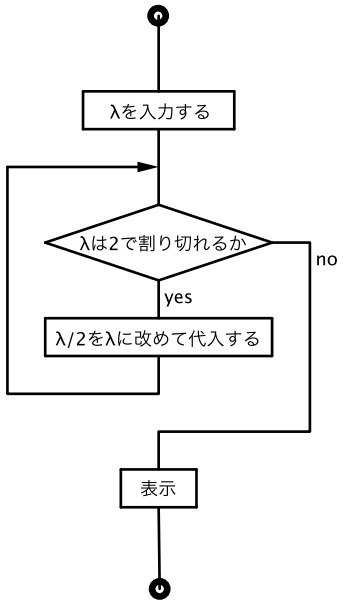
a は、「λが偶数である間、λを λ/2 に置き換える」という繰り返しを行うと求められます。

出力例

```
数値を入力
240
240を2で割れるだけ割ると15です
```

ただし、上の出力例では、2 行目は人間がタイプしたものです。

課題 17 の流れ図



(6.3) **無限ループ** While 文の条件に、常に成立する条件 "true" (真)[17]を書くと[18]、いつまでもループし続ける、**無限ループ**になります。

無限ループ

```
1 TextWindow.WriteLine("2乗計算機はじまり")
2 While "true"
3   a = TextWindow.ReadNumber()
4   TextWindow.WriteLine("2乗は" + a*a)
5 EndWhile
```

出力例

```
2乗計算機はじまり
3
```

[17]逆に、常に成立しない条件 (偽) は "false" です。
[18]"true"でなくても、1<2 などの常に成立する条件を書くと無限ループになります。

```
2乗は9
5
2乗は25
-4
2乗は17
:
```

ただし、上の出力例では、2, 4, 6 行目は人間がタイプしたもので、ウィンドウを閉じない限りプログラムは停まらないので、途中までを例示してあります。

(6.4) **諸悪の根源 Goto** これから学ぶ Goto 文は、**みだりに使わないで下さい**[19]。読み易いプログラムのためには Goto 文は害悪であり、代わりに、「順次」、「分岐」、「反復」という 3 つの基本構造の組合せでプログラムを記述するというのが、**構造化プログラミング**の考えです[20]。

Goto 文は、プログラム中に定義された**ラベル**の位置にプログラムの制御を移す命令です。

Goto

```
  (A)
Goto hogehoge
  (B)
hogehoge:
  (C)
```

上の hogehoge のように、後ろにコロンを付けた文字列がラベルとして認識されます。上のプログラムだと (A) を実行した後、Goto 文で hogehoge: の位置に飛び、(C) を実行します。つまり、(B) は実行されません。

(6.5) **ループからの脱出** While 文は、ループの開始時点で条件を判断しますが、それでは不便なこともあるので、Goto 文を用いて強制的にループを脱出することができます[21]。

次の例は、無限ループだった (6.3) とほぼ同じですが、0 を入力すると停止する点が異なります。

ループ脱出の例

```
1  TextWindow.WriteLine("停まる2乗計算機はじまり")
2  While "true"
3    a = TextWindow.ReadNumber()
4    If a = 0 Then
5      Goto finish
6    EndIf
7    TextWindow.WriteLine("2乗は" + a*a)
8  EndWhile
9  finish:
10 TextWindow.WriteLine("おわり")
```

出力例

```
停まる2乗計算機はじまり
3
2乗は9
5
2乗は25
0
おわり
```

ただし、上の出力例では、2, 4, 6 行目は人間がタイプしたものです。

(6.6) **課題 18 – 入力した数の合計** 次のような動作をするプログラムを作成し、.sb ファイルを提出して下さい。

(1) 「貯金せよ」と表示する。
(2) 数値を入力する。
(3) 入力した数値が 0 でなければ、(1) に戻る。

[19] 過去には、使う、使わないで大論争を巻き起こしてもいますし、現在でもすぐに論争になります。

[20] If 文、For 文、While 文において、インデントを用いてプログラムを書くことも、構造化プログラミングの考えに沿ったものです。

[21] Small Basic ではループ脱出のための専用の命令がないので、仕方なく Goto 文を用いますが、ループ脱出のための命令 (たいてい break という命令) を持つプログラミング言語も多いです。

(4) 入力した数値が 0 ならば、今までの入力の合計を、「合計 = ○○」と表示する。

出力例

```
貯金せよ
30
貯金せよ
45
貯金せよ
30
貯金せよ
0
合計 = 105
```

ただし、上の出力例では、2, 4, 6, 8 行目は人間がタイプしたものです。

(6.7) **課題 19# − ユークリッドの互除法 (義務ではない)** 次のような動作をするプログラムを作成し、.sb ファイルを提出して下さい。

(1) 「数値を 2 つ入力」と表示する。
(2) 数値を 2 回入力する (入力した数値をここでは λ, μ とする)。
(3) λ, μ の最大公約数を求め、「最大公約数 = ○○」と表示する。

ここで、最大公約数を求めるには、

(i) λ を μ で割った余りを r と置く。
(ii) $r = 0$ ならば、μ は λ の約数になるので、最大公約数は μ.
(iii) $r \neq 0$ ならば、$\lambda = \mu$, $\mu = r$ と置き直して、最初に戻る。

という**ユークリッドの互除法**を用いるとよい。

出力例

```
数値を2つ入力
72
120
最大公約数 = 24
```

§7 サブルーチン

(7.1) **サブルーチン　サブルーチン**とは、プログラムを実行したときの処理の流れ (**メインルーチン**) から独立させた、プログラム断片です。サブルーチンには、一般に次のような利点があります[22]。

コード量の減少 複数回実行する処理を何度も記述するのではなく、サブルーチンにして呼び出すことで、コード量が減少する。

保守性 複数回実行する処理を何度も記述すると、修正を施すときに、間違いなくすべての該当箇所を修正するのが難しいが、サブルーチンに独立しておけば、修正箇所はサブルーチン中の 1 箇所で済む。

可読性 メインルーチンに複雑な処理があると、全体として何をしているか把握しづらいが、意味のあるまとまりをサブルーチンにしてメインルーチンから呼び出すようにすると、処理内容を把握し易くなる。

再利用性 意味のあるまとまりをサブルーチンにしておけば、別のプログラムで同じ処理が必要になった場合、再利用できる。

実装の隠蔽 何らかの状態を表す変数を直接変更したり参照したりするのではなく、サブルーチンを介してのみ変更・参照を許すなど、触れて欲しくない部分に触れないようにできる。将来サブルーチンのコードを改良して、状態を表す変数の役割が変わってしまったりしても、サブルーチンを変更するだけで呼び出し側のコードは変更せずに済む。

(7.2) **サブルーチンの定義と呼出し** Small Basic では次のようにサブルーチンを定義します。定義の場所は、いわゆるトップレベルならばどこでも構いませんが、メインルーチンの前か後でまとめて定義するのが良いでしょう。

サブルーチンの定義

```
Sub サブルーチン名
```

[22] Small Basic には変数の**スコープ**がないため、ここにあげた利点のうち、再利用性と実装の隠蔽については、十分に恩恵を受けられません。スコープとは変数の有効範囲のことで、Small Basic では、一度定義された変数はプログラム全体で有効です。つまりサブルーチンの中で変数を変更すると、メインルーチンの同名の変数は同一の変数なので変更されてしまいます。

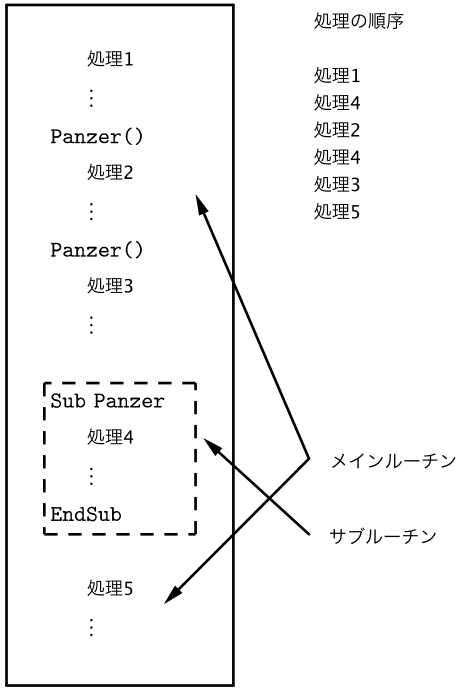
```
    処理（複数行でもよい）
EndSub
```

定義されたサブルーチンの呼び出しは次のように行います。

サブルーチンの呼び出し

```
サブルーチン名()
```

サブルーチンの処理の順序



下の例では、サブルーチン「Hantei」を定義して 2 回呼び出しています。

サブルーチンの例

```
1  TextWindow.WriteLine("あなたの点数を入力")
2  point = TextWindow.ReadNumber()
3  Hantei()
4  TextWindow.WriteLine("隣の人の点数を入力")
5  point = TextWindow.ReadNumber()
6  Hantei()
7
8  Sub Hantei
9    If point >= 60 Then
10     TextWindow.WriteLine("合格")
11   Else
12     TextWindow.WriteLine("不合格")
13   EndIf
14 EndSub
```

出力

```
あなたの点数を入力
80
合格
隣の人の点数を入力
55
不合格
```

ただし、上の出力例では、2, 5 行目は人間がタイプしたものです。

(7.3) **課題 20 – 誕生月占い** 次のような動作をするプログラムを作成し、.sb ファイルを提出して下さい。ただし、2 回ある表示する部分は、サブルーチンにすること。

(1) 「あなたの誕生月を入力」と表示する。
(2) 誕生月の数値を入力する。
(3) 2 月生まれならば「素晴しい」、それ以外は「ごく普通」と表示する。
(4) 「隣の人の誕生月を入力」と表示する。
(5) 誕生月の数値を入力する。
(6) 2 月生まれならば「素晴しい」、それ以外は「ごく普通」と表示する。

出力例

```
あなたの誕生月を入力
```

```
2
素晴しい
隣の人の誕生月を入力
3
ごく普通
```

ただし、上の出力例では、2, 5 行目は人間がタイプしたものです。

(7.4) **サブルーチンの計算結果の利用** 下のプログラムでは、「Fibonacci」という名前のサブルーチン[23]を 2 回呼び出しています。

サブルーチンの例

```
 1 TextWindow.WriteLine("数を2つ入力")
 2 n = TextWindow.ReadNumber()
 3 Fibonacci()
 4 a1 = a ' 一旦退避
 5 n = TextWindow.ReadNumber()
 6 Fibonacci()
 7 TextWindow.WriteLine(a1 + ", " + a)
 8
 9 Sub Fibonacci
10   x = Math.Power(1.618034, n)
11   y = Math.SquareRoot(5)
12   a = Math.Round(x / y)
13 EndSub
```

Math.Round(x) は、x を四捨五入した整数を返すメソッド、Math.Power(x, n) は、x の n 乗を返すメソッドです。

4 行目の「a1 = a」は、6 行目の「Fibonacci()」を実行したときに変数 a が変更されるので、その前に計算結果を保存するためのコードです。

出力例

```
数を2つ入力
4
5
3, 5
```

ただし、上の出力例では、2, 3 行目は人間がタイプしたものです。

(7.5) **課題 21 — フィボナッチ数列** 上のプログラム例で定義されているサブルーチン「Fibonacci」を**変更なしに再利用して**、次のような動作をするプログラムを作成し、.sb ファイルを提出して下さい。

(1) 「数値を入力」と表示する。
(2) 数値を入力する (入力した数値をここでは λ とする)。
(3) フィボナッチ数列の第 1 項から第 λ 項までを表示する。

出力例

```
数値を入力
10
1 1 2 3 5 8 13 21 34 55
```

ただし、上の出力例では、2 行目は人間がタイプしたものです。また、「Press any key to continue ...」が数列の直後に表示されるのは格好が悪いので、数列の次の行に表示されるようにすること。

§8 いろいろな関数

(8.1) **数学関数の一覧** Math オブジェクトに属する数学関数の一覧を、以下に示します。Small Basic のエディタで「Math.」とタイプすると、インテリセンスで確認することもできます。Math.Pi のみプロパティです (つまり後続の括弧が不要)。

[23] フィボナッチ数列の第 n 項を計算して、変数 a に設定するサブルーチンです。この計算方法を課題 16 に用いてはいけません。ちなみに、この計算方法を即座に理解できる人は少ないと思いますが、入力に対して所望の出力 (今の場合はフィボナッチ数列の項) を返すことさえわかっていれば、ブラックボックスとして利用できます。これもサブルーチンの特徴のひとつです。

式	戻り値
`Math.Pi`	円周率
`Math.Cos(x)`	$\cos$ x
`Math.Sin(x)`	$\sin$ x
`Math.Tan(x)`	$\tan$ x
`Math.ArcCos(x)`	$\cos^{-1}$ x
`Math.ArcSin(x)`	$\sin^{-1}$ x
`Math.ArcTan(x)`	$\tan^{-1}$ x
`Math.Log(x)`	$\log_{10}$ x
`Math.NaturalLog(x)`	$\log_e$ x
`Math.Max(x, y)`	x と y の最大値
`Math.Min(x, y)`	x と y の最小値
`Math.Ceiling(x)`	x 以上の整数のうち最小のもの
`Math.Floor(x)`	x 以下の整数のうち最大のもの
`Math.Round(x)`	x を四捨五入した整数
`Math.Abs(x)`	x の絶対値
`Math.Power(x, y)`	x^y
`Math.SquareRoot(x)`	$\sqrt{x}$
`Math.Remainder(x, y)`	x を y で割った余り
`Math.GetRandomNumber(max)`	1 から max の乱数
`Math.GetDegrees(x)`	x ラジアンを度に変換した値
`Math.GetRadians(x)`	x 度をラジアン に変換した値

三角関数、逆三角関数の引数や戻り値の単位はラジアンですので、度数法との間で変換したいときは、`Math.GetDegrees` や `Math.GetRadians` を使います。

(8.2) **文字列関数の一覧** `Text` オブジェクトに属する文字列関数の一覧を、以下に示します。Small Basic のエディタで「`Text.`」とタイプすると、インテリセンスで確認することもできます。

式	戻り値
`Text.Append(text1, text2)`	2 つの文字列の連結
`Text.GetLength(text)`	文字列の長さ
`Text.ConvertToLowerCase(text)`	小文字に変換した文字列
`Text.ConvertToUpperCase(text)`	大文字に変換した文字列
`Text.GetCharacter(code)`	文字コードが `code` の文字
`Text.GetCharacterCode(ch)`	文字 `ch` の文字コード
`Text.GetIndexOf(text, subText)`	文字列 `text` 中の `subText` の出現位置。現れなければ 0。
`Text.GetSubText(text, start, len)`	文字列 `text` の位置 `start` から `len` 文字分の文字列
`Text.GetSubTextToEnd(text, start)`	文字列 `text` の位置 `start` から最後までの文字列
`Text.IsSubText(text, subText)`	文字列 `text` 中に `subText` が現れれば `True`。
`Text.StartsWith(text, subText)`	文字列 `text` が `subText` から始まっていれば `True`
`Text.EndsWith(text, subText)`	文字列 `text` が `subText` で終わっていれば `True`

上の表で、文字コードは Unicode の文字コードです。`Text.Append` は演算子「`+`」と同じ動作ですが、文字列が数値を表しているときにも、数値として解釈して和を計算するのではなく、文字列として連結をします。

(8.3) **課題 22# — 大きい数の桁数 (義務ではない)** 次のような動作をするプログラムを作成し、`.sb` ファイルを提出して下さい。

(1) 「a の b 乗の a と b を入力」と表示する。
(2) 2 数を入力する。(入力した 2 数を、ここでは α, β とする)
(3) α^β の桁数を求め、「α の β 乗は○○桁です」と表示する。

出力例

```
aのb乗のaとbを入力
7
1234
7の1234乗は1043桁です
```

ただし、上の出力例では、2, 3 行目は人間がタイプしたものです。また、上のように、1000 桁程度の結果も正しく表示されるようにすること[24]。

(8.4) **課題 23# — 部分文字列 (義務ではない)** 次のような動作をするプログラムを作成し、.sb ファイルを提出して下さい。

(1) 「3 文字以上の文字列を入力」と表示する。
(2) 文字列を入力する (入力した文字列をここではλとする)。
(3) λが 3 文字に満たなければ、何もしない。
(4) λが 3 文字以上ならば、「先頭から 3 文字は○○○です」、「末尾から 3 文字は○○○です」と表示する。

出力例その 1

```
3文字以上の文字列を入力
hokkyodai
先頭から3文字はhokです
末尾から3文字はdaiです
```

出力例その 2

```
3文字以上の文字列を入力
tt
```

ただし、上の各出力例では、2 行目は人間がタイプしたものです。

§9 配列

今学期はやりません。

[24] ヒント: 対数を使えば 1000 桁程度の結果にも対応できます。

§10 グラフィックス

今学期はやりません。

§11 練習問題

(11.1) **課題 24# — カウントダウン (義務ではない)** 次のような動作をするプログラムを作成し、.sb ファイルを提出して下さい。

(1) 「秒数を入力」と表示する (入力した数値を、ここではλとする)。
(2) λから始めて、1 ずつ減少しながら 0 まで、数値を 1 秒おきに表示する。

出力例

```
秒数を入力
5
5
4
3
2
1
0
```

ただし、上の出力例では、2 行目は人間がタイプしたものです。3 行目以降の出力は、一斉に表示されるのではなく、1 秒おきに表示されるようにすること。

Program.Delay(msec) を用いると、msec ミリ秒何もせずに停止するので、これを用いると 1 秒間隔の表示が実現できます。また、変数が減少しながらの繰り返しは、For 文の Step に負の値 (今の場合ならば −1) を指定しても実現できますし、i が 0 からλまで増加するとき、λ − i はλから 0 まで減少することを用いても実現できます。

(11.2) **課題 25# — 約数列挙 (義務ではない)** 次のような動作をするプログラムを作成し、.sb ファイルを提出して下さい。

(1) 「整数を入力」と表示する (入力した数値を、ここではλとする)。

(2) λ の正の約数をすべて表示する。

出力例

```
整 数 を 入 力
12
約 数 は 1 2 3 4 6 12
```

ただし、上の出力例では、2 行目は人間がタイプしたものです。

(11.3) **課題 26$^{\#}$ — 最大の約数 (義務ではない)** 次のような動作をするプログラムを作成し、.sb ファイルを提出して下さい。

(1) 「整数を入力」と表示する (入力した数値を、ここでは λ とする)。
(2) λ の約数のうち、λ 未満で最大のものを表示する。

出力例

```
整 数 を 入 力
12
最 大 の 真 の 約 数 は 6
```

ただし、上の出力例では、2 行目は人間がタイプしたものです。

(11.4) **課題 27$^{\#}$ — 公約数 (義務ではない)** 次のような動作をするプログラムを作成し、.sb ファイルを提出して下さい。

(1) 「整数を 2 つ入力」と表示する。
(2) 2 つの整数の公約数をすべて表示する。

出力例

```
整 数 を 2 つ 入 力
12
18
公 約 数 は 1 2 3 6
```

ただし、上の出力例では、2, 3 行目は人間がタイプしたものです。

(11.5) **課題 28$^{\#}$ — 素因数分解 (義務ではない)** 次のような動作をするプログラムを作成し、.sb ファイルを提出して下さい。

(1) 「整数を入力」と表示する。
(2) 入力された整数の素因数分解を表示する。

出力例

```
整 数 を 入 力
1800
素 因 数 分 解 は 2 2 2 3 3 5 5
```

ただし、上の出力例では、2, 3 行目は人間がタイプしたものです。

(11.6) **課題 29$^{\#}$ — 素数 (義務ではない)** 次のような動作をするプログラムを作成し、.sb ファイルを提出して下さい。

(1) 「正の整数を入力」と表示して、正の整数を入力
(2) 入力された整数以上で最小の素数を p として、「次の素数は p」と表示する。

出力例

```
正 の 整 数 を 入 力
6
次 の 素 数 は7
```

出力例

```
正 の 整 数 を 入 力
24
次 の 素 数 は29
```

ただし、上の各出力例では、2 行目は人間がタイプしたものです。

(11.7)　**課題 30$^{\#}$ — 剰余環における逆元 (義務ではない)**　次のような動作をするプログラムを作成し、`.sb` ファイルを提出して下さい。

(1) 「法を入力」と表示し、正の整数を入力 (ここでは m とする)。
(2) 「整数を入力」と表示し、整数を入力 (ここでは x とする)。
(3) $xy-1$ が m の倍数となるような整数 y が存在するならば、「m を法とする x の逆元は y」と表示し、存在しないならば、「逆元はない」と表示する。

出力例

```
法を入力
5
整数を入力
2
5を法とする2の逆元は3
```

出力例

```
法を入力
9
整数を入力
7
9を法とする7の逆元は4
```

出力例

```
法を入力
8
整数を入力
2
逆元はない
```

ただし、上の各出力例では、2, 4 行目は人間がタイプしたものです。

3 授業のノート (TEX 編)

§1 文書組版システム TEX の導入

TEX(テフ, テック) とは「組版(くみはん)ソフトウェア」です。組版とは活字を組んで版を作ることなので、これは文書を作成するソフトウェアを意味しています。以下に TEX の大事な特徴をあげます。

- フリーソフトウェアであり、かつ、あらゆる OS で動作する。
- 文書の論理構造を明確にでき、相互参照も強力。
- 数式が美しい。

これらほど大事ではない特徴も脚注[25] や傍注 にあげます。

- 高精度
- 目次が作れる
- 索引が作れる
- 1980 年頃に誕生
- 自動で hyphenation
- office ではなく office になる

大学のコンピュータ室や、数学研究室のコンピュータには、いくつかある TEX の亜種のうち、最も広く使われている LATEX2$_\varepsilon$ (ラテフツーイー、ラテックツーイー) が導入されています。自宅のコンピュータでも TEX を利用したい場合、以下のような方法があります。上ほどお薦めです。

(1) 書籍**「[改訂第 5 版] LaTeX2e 美文書作成入門」**(ISBN 978-4-7741-4319-4) の付録 DVD-ROM からインストールする。Windows にも Macintosh にも対応。

(2) 阿部紀行氏の Windows 用 TeX インストーラを利用する[26]。
`http://www.math.sci.hokudai.ac.jp/~abenori/`

(3) 角藤亮氏のサイト
`http://www.fsci.fuk.kindai.ac.jp/kakuto/win32-ptex/` の Windows 用簡易インストーラを利用する。

(4) Macintosh の場合、桐木紳氏のサイト
`http://math.kyokyo-u.ac.jp/~kiriki/ptex/` や、小川弘和氏のサイト `http://www2.kumagaku.ac.jp/teacher/herogw/` から入手する。

(5) Linux の場合 apt-get (synaptic) などを利用した簡単なインストール方法が用意されている。始めからインストールされていることも多い。

[25]
- どの OS や、どのコンピュータでも出力が同一。
- TEX 方式の数式入力方法を採用するソフトウェアが多い。
- TEX で書かれた本も多い。
- WYSIWYG (what you see is what you get. 画面のままの印刷が得られる) ではなく「バッチ式」であり、この点は HTML に近い。

[26]ただし、惜しいことに texworks の日本語環境が整わない。いつの間にか対応してくれるかも知れませんが。

§2 TEX の使用方法

TEX は Microsoft Word などとは異なり、入力から印刷までを単一のソフトウェアで行うわけではありません。Pdf ファイルで閲覧・印刷を行う場合、その過程は以下のようになります。

TEX での文書作成の流れ

`.tex` ファイル	人間がエディタで作成する
↓	
`.dvi` ファイル	`.tex` ファイルを「コンパイル」すると得られる
↓	
`.pdf` ファイル	`.dvi` ファイルから得られる

実際には、これらを統合する **TeXworks** というソフトウェアがインストールされており、複数のプログラムが動いていることは意識せずとも TEX を利用できるようになっています。TeXworks を起動するには (他にも方法はありますが)、Windows XP ならば、スタートメニューから「ファイル名を指定して実行」に、「`texworks`」と入力し Enter を押します。Windows Vista 以降ならば、スタートメニューから「検索を開始」の所に「`texworks`」と入力して検索してプログラムの TeXworks を実行します。

(2.1) **課題 51 – 最初の TEX 文書** LATEX2$_\varepsilon$ の書式に従った最小の tex ファイルは次のようになります。バックスラッシュ (\) はキーボードの円記号 (¥) のキーを押すと入力でき、コンピュータの環境によっては画面上でも円記号です。

0000 **和地**-51.tex

```
1 \documentclass{jarticle}
```

```
\begin{document}
夏空と駆ける先には蝸牛
\end{document}
```

\documentclass{jarticle} は、「jarticle」という文書クラスで文書を作成するという命令です。用途に応じて次のような文書クラスがあります。

用途	欧文	和文横書き	和文縦書き	新和文横書き
論文・レポート	article	jarticle	tarticle	jsarticle
長い報告書	report	jreport	treport	(ない)
本	book	jbook	tbook	jsbook

また、\documentclass[a4j,12pt]{jarticle} のように、**オプション**を 1 つ、またはカンマ区切りで複数個与えることができます[27]。

\begin{document} ～ \end{document} は **document 環境**で、この中に書かれたものが出力されます。

次の作業の結果得られた pdf ファイルを提出して下さい。

(1) 上の例の俳句を他のものに替えて 0000 和地-51.tex のようなファイル名で保存する。
(2) コンパイルして pdf ファイルを作成し、0000 和地-51.pdf のようなファイル名の pdf ファイルを提出する。

(2.2) **コメント** tex ファイル中で、パーセント記号 (%) から行末までは無視されます (改行も無視)。これは**コメント**と呼ばれ、出力はしたくないけれど、tex ファイル中に残しておきたいことなどを書けます。

入力

```
\documentclass{jarticle}
\begin{document}
夏空と駆ける先には蝸牛 % 字余り
\end{document}
```

[27] 例えば次のようなオプションが利用可能です。a4j, b5j: 用紙サイズの指定。無指定の時はやや余白の広い a4paper を指定したことになる。landscape: 用紙を横置きで使う指定。10pt, 11pt, 12pt: フォントのサイズの指定。twocolumn: 2 段組にする指定。

出力

夏空と駆ける先には蝸牛

(2.3) **改行や空白文字の扱い** Microsoft Word では、改行したりスペースを入力すれば、その通りの画面表示と印刷結果になります。しかし、TEX での扱いはこれとは異なり次のようになります。

- 行末が和文文字 (全角) の時、改行は無視される。
- 行末が欧文文字 (半角) の時、改行は空白 (半角スペース) と解釈される。
- 連続する空白 (半角スペース) は 1 つの空白と同等。
- 行頭や行末の空白 (半角スペース) は何個あっても無視される。
- 空白だけの行 (空行) は、段落の切れ目になる。

このように、半角スペースは良い組版をするため特別の扱いを受けますので、全角スペースで空白を空けることは必要でなければしない方がよいです。

入力

```
\documentclass{jarticle}
\begin{document}
tex ファイルを書くときは、
適宜改行を入れて見易く整えるのが
習慣になっています。

She wore              a yellow ribbon.
\end{document}
```

出力

tex ファイルを書くときは、適宜改行を入れて見易く整えるのが習慣になっています。

She wore a yellow ribbon.

(2.4) **初めての命令** \documentclass{jarticle}のように\ (バックスラッシュ。環境によっては ¥) の付いたものが TEX の命令です。例えば、\TeX という命令は、TEX を出力します。末尾が英字や和字である命令の後の空白については、次の例のように注意が必要です。

入力

```
\documentclass{jarticle}
\begin{document}
\TeX is joyful. % 命令の後の空白は無視される
\TeX{} is joyful. % こうすればよい
{\TeX} is joyful. % あるいはこう
\end{document}
```

出力

TEXis joyful. TEX is joyful. TEX is joyful.

反対に、TEX の直後に空白なしに続けたいときに、

\TeX の楽しみ % エラー

と入力すると「\TeX の楽しみ」までで 1 つの命令とみなされてエラーになります。

(2.5) **特殊文字** パーセント記号を出力したいとき、単に tex ファイルに書いてもコメントの始まりとみなされて出力されません。バックスラッシュ (円記号) なども同様で、このような**特殊文字**には以下ものがあります。

```
# $ % & _ { } \ ^ ~ < > |
```

これらのうちいくつかは、バックスラッシュを付加すると次のように出力できます。

入力

```
\documentclass{jarticle}
\begin{document}
\# \$ \% \& \_ \{ \}
\end{document}
```

出力

$ % & _ { }

また別の方法として、入力をそのまま出力する\verb 命令を用いることもできます。\verb|foo|のように、任意に選んだ同じ文字 (ここでは |) で挟まれた部分をそのまま出力します。

入力

```
\documentclass{jarticle}
\begin{document}
\verb@\@ \verb|$29 > $28|
\end{document}
```

出力

\ $29 > $28

また、< > | は後述の数式モードでは出力できます。

(2.6) **課題 52 – 特殊文字の出力** 次のような出力を与える tex ファイル (ファイル名は 0000 和地-52.tex のように) を作り提出して下さい。ただし、わずかな (無害の) スペースの有無などにより、作成したファイルの出力結果と下の出力例を比べたときに、TEX が自動的に決定する行の折り返し位置が異なる場合がありますが、それは構いません。逆に、改行位置を下の出力例に合わせるために、無理な操作をするのは良くありません。全角スペースの利用は特に有害です。

出力

TEX で%を出力するためには\%と入力します。2 つ目のパーセントは\verb 命令を使いました。

上の出力では、1 つ目の%と 2 つ目の%は、tex ファイルでの記述が異なるため、書体が違うことにも注意して下さい。最初の TEX は、単に TeX と入力

するわけではありません (少し前を見て下さい)。また、上の出力では掲載を省略していますが、用紙の下部にページ番号も出力されています。

§3 文書の構造

文書には、章があり、その中にいくつかの節 (セクション) があり、さらに、小節 (サブセクション) や段落を含むといった**論理構造**があります。他方、章の見出しは文字サイズを大きくし、節の見出しはそれより少し小さくし、また、これらの見出しは太字にするといった**物理構造 (レイアウト)** もあります。TEX では、論理構造の指定のみ行い、物理構造の指定は陽には行わないことが推奨されます。つまり、「文字サイズを大きくする」と指定するのではなく、「ここから節を始める」と指定します。どのような物理構造になるかは jarticle などの文書クラスによって決まります。

(3.1) **論理構造を指定する命令** jarticle クラスで論理構造を指定するには、主に次のような命令があります。ただし、段落については、**空行**を作れば段落の区切りになります。段落に見出しの必要がないときは、普通は\paragraph 命令を用いずに、空行で段落を作成します。

命令	意味	使用方法
\section	節	\section{節の名前}
\subsection	小節	\subsection{小節の名前}
\subsubsection	小小節	\subsubsection{小小節の名前}
\paragraph	段落	\paragraph{段落の名前}

入力

```
\documentclass{jarticle}
\begin{document}
\section{論理構造を指定する命令}
\subsection{節と小節}
\verb@\section@命令で節、
\verb@\subsection@命令で小節の開始を指定します。
\subsection{段落}
単に空行を作るだけでも段落を開始できますが、
\verb@\paragraph@命令ならば
見出し付きの段落を開始します。

段落先頭のインデントは自動で付くため、
入力の必要はありません。
見出しの節番号も自動で付きます。
\end{document}
```

出力

1 論理構造を指定する命令

1.1 節と小節

\section 命令で節、\subsection 命令で小節の開始を指定します。

1.2 段落

単に空行を作るだけでも段落を開始できますが、\paragraph 命令ならば見出し付きの段落を開始します。

段落先頭のインデントは自動で付くため、入力の必要はありません。見出しの節番号も自動で付きます。

また、上の出力では掲載を省略していますが、用紙の下部にページ番号も出力されています。

(3.2) **物理構造を指定する命令 (下線、改ページ、改行)** TEX では物理構造を陽に指定しないことが推奨されるものの、実際には文字サイズなどの物理構造を指定したいこともあります。

文章の一部に**下線**を引くこと、ページを改めること (**改ページ**)、行を改めること (**改行**) はどれも物理構造の指定ですが、次の命令で実現できます。

入力	出力
\underline{下線}	下線
\newpage	改ページする
\\	改行する

TEX が自動的に決定する改行位置とは無関係に、強制的に改行したいとき \\命令を使います。しかし、(数式などではない地の文で) 強制的に改行したいのは、人間の方が間違っていることが多いので、\\命令は極力使わないようにします。

また、\\[1cm] のように後ろにブラケットがあると、囲まれた寸法だけ行を送ります。この命令も、微調整が必要な時にはやむを得ないこともありますが、極力使わないようにします。\\の後にブラケットを書きたい場合は、こう解釈されないように、何もしない命令\relax を用いて、\\ \relax [] とするなどの注意が必要です。

(3.3) **物理構造を指定する命令 (文字サイズ、書体)** 文字サイズも物理構造です。次の命令で文字サイズが変更できます。

命令	見本
\tiny	サイズ Sample
\scriptsize	サイズ Sample
\footnotesize	サイズ Sample
\small	サイズ Sample
\normalsize	サイズ Sample
\large	サイズ Sample
\Large	サイズ Sample
\LARGE	サイズ Sample
\huge	サイズ Sample
\Huge	サイズ Sample

サイズ変更の影響範囲を限定するためには、

```
{\small 小さい} 普通
```

のようにブレース ({ }) で囲みます。

入力

```
1 \documentclass{jarticle}
2 \begin{document}
3 {\small ここだけ小さい、}
4 普通の大きさ、
5 \large ここだけ大きい、
6
7 つもりだったが以降ずっと大きい。
8 \end{document}
```

出力

ここだけ小さい、普通の大きさ、ここだけ大きい、
つもりだったが以降ずっと大きい。

また、次の命令で文字の書体が変更できます。上の\small などの命令とは、ブレースの付き方が違うことに注意が必要です。

入力	意味	出力
\textrm{Roman}	ローマン	Roman
\textbf{Bold}	ボールド	**Bold**
\textit{Italic}	イタリック	*Italic*
\textsl{Slant}	斜体	*Slant*
\textsf{Sans Serif}	サンセリフ	Sans Serif
\texttt{Typewriter}	タイプライター	Typewriter
\textsc{Small Caps}	スモールキャップス	Small Caps
\textgt{ゴシック体}	ゴシック	**ゴシック体**

(3.4) **課題 53 – 文書の論理構造と物理構造** 次のような出力を与える tex ファイル (ファイル名は 0000 和地-53.tex のように) を作り提出して下さい。

出力

1　節など

1.1 節・小節・段落

節を始めるには\section 命令を使い、小節を始めるには\subsection 命令を使います。

このように新しい段落を始めるためには、単に空行を作ればよいです。また、行の途中で
改行もできます。

1.2 見出し付き段落

段落に見出しを付けたい ときは、\paragraph 命令を使います。

見出し付き段落の例　この段落の上方直前の空きは、\paragraph 命令を使うと自動的に付くものです。

2　文字サイズや書体

2.1 文字サイズ

例えば\footnotesize 命令を使うと、小さい文字になります。

2.2 書体

例えば\textsf 命令を使うと、read me!のような書体になります。太字で大きい文字サイズにしたいときは、\textbf と\Large を組み合わせると、**like this** のように出力されます。

上の出力では掲載を省略していますが、用紙の下部にページ番号も出力されています。ただし、TEX が自動的に決定する行の折り返し位置が、無害な半角スペースなどで多少異なっていても構いません。

§4　箇条書き

(4.1)　**環境**　\begin{document} ··· \end{document} のように\begin と\end で囲まれた構造を**環境**と呼びます。document 環境の他には、

環境名	意味
quote 環境	引用を行う。段落の先頭はインデントしない。
quotation 環境	引用を行う。段落の先頭をインデントする。
flushright 環境	右寄せする。
flushleft 環境	左寄せする。
center 環境	中央揃えする。

などもあります

入力

```
1  \documentclass{jarticle}
2  \begin{document}
3  \begin{quote}
4    引用します
5  \end{quote}
6  \begin{quotation}
7    引用します。インデントもします。
8  \end{quotation}
9  \begin{flushleft}
10   \Large 大きい字で左寄せ
11 \end{flushleft}
12 \begin{flushright}
13   右寄せ
14 \end{flushright}
15 \begin{center}
16 中央揃え
17 \end{center}
18 \end{document}
```

出力

引用します

引用します。インデントもします。

大きい字で左寄せ

右寄せ

中央揃え

上の例でわかるように、環境は{ ··· }と同じように、\Large 命令の影響範囲を限定しています。

(4.2) **箇条書き環境** 次のような、意味の異なる箇条書き環境があります。

環境名	意味
itemize	番号の付かない箇条書き
enumerate	番号の付く箇条書き
description	見出しの付く箇条書き

これらの環境の本体では、\item 命令に続いて箇条書きする項目を書きます。また、description 環境では、\item[見出し] として見出しを指定します。itemize と enumerate 環境でもこうすると見出しを変更できます。

入力

```
1  \documentclass{jarticle}
2  \begin{document}
3  \begin{itemize}
4    \item かき氷ともに頬ばる日を思う
5    \item 向日葵を見上げる君の大き顔
6  \end{itemize}
7  \begin{enumerate}
8    \item 母の持つ砂場の蚯蚓に後ずさる
9    \item 炎天下滑りて昇りまた滑り
10 \end{enumerate}
11 \begin{description}
12   \item [仲夏] 逃げ惑い泣き疲れ蝿を夢に見る
13   \item [晩夏] 水風船親の心を子は知らず
14 \end{description}
15 \end{document}
```

出力

- かき氷ともに頬ばる日を思う
- 向日葵を見上げる君の大き顔

1. 母の持つ砂場の蚯蚓に後ずさる
2. 炎天下滑りて昇りまた滑り

仲夏 逃げ惑い泣き疲れ蝿を夢に見る

晩夏 水風船親の心を子は知らず

また、例えば、document 環境の中に itemize 環境を入れるなど、多くの場合環境は入れ子にできます。

(4.3) **課題 54 – 箇条書きなどの環境** 次のような出力を与える tex ファイル (ファイル名は 0000 和地-54.tex のように) を作り提出して下さい。ただし、TEX が自動的に決定する行の折り返し位置が、無害な半角スペースなどで多少異なっていても構いません。また、下の出力では掲載を省略していますが、用紙の下部にページ番号も出力されています。

出力

本日のまとめ

0000 和地輝仁

上のタイトルは\Large の大きさです。今日学んだ環境をまとめると以下のようになります。

引用 quote, quotation

寄せ・揃え flushleft, flushright, center

箇条書き itemize, enumerate, description

このうち、番号付きの箇条書きの例は下のようになります。

1. itemize 環境は番号なし
2. enumerate 環境は番号付き
3. description 環境は見出し付き

§5 表組み

(5.1) **tabular 環境** 表組みは tabular 環境で行います。

tabular 環境の文法

```
\begin{tabular}{列 指 定}
  表 本 体
\end{tabular}
```

列指定には、1 列につき、l (左寄せ)、c (中央揃え)、r (右寄せ) のどれかを 1 文字指定します。これらの各文字の前後に| (縦罫線) が指定できます。

表本体において、列の区切りは&、行の終わりは\\で表します。行の先頭に\hline を書くと横罫線が引かれ、\cline{2-3}のように書くと部分的な横罫線 (この場合 2 から 3 列目まで) が引かれます。

横方向にコマを連結したいときは、\multicolumn{連結するコマ数}{列指定}{コマの内容} を用います。

入力

```
1  \documentclass{jarticle}
2  \begin{document}
3  \begin{tabular}{|l|c|rr|}
4  \hline \multicolumn{4}{|c|}{りんごの表} \\
5  \hline
6  \hline 品名 & サイズ & 単価 & 数量 \\
7  \hline       りんご & 大きい & 180 & 15 \\
8  \cline{2-4}         & ミニ   &  90 & 8 \\
9  \hline
10 \end{tabular}
11 \end{document}
```

出力

りんごの表			
品名	サイズ	単価	数量
りんご	大きい	180	15
	ミニ	90	8

(5.2) **課題 55 – 表組み** 次のような出力を与える tex ファイル (ファイル名は 0000 和地-55.tex のように) を作り提出して下さい。下の出力では掲載を省略していますが、用紙の下部にページ番号も出力されています。

出力

tabular 環境のまとめ		
列指定	揃え	l, c, r
	縦罫線	|
表本体	列区切り	&
	改行	\\
	横罫線	\hline, \cline
	コマの連結	\multicolumn

§6 数式

(6.1) **インライン数式と別行立て数式　数式モード**[28]には次の 2 つがあり、1 つの数式には**一方のみ**を使います。

- インライン数式:　　$ ··· $
- 別行立て数式:　　\[··· \]

つまり、

```
$ \[ y = -2x - 1 \] $
```
（これは間違い）

のような使い方はしません。

入力

```
1  \documentclass{jarticle}
2  \begin{document}
3  テキストモードだと、
4  a = -2x - 1 という出力になってしまいますが、
5  ドルで囲んでインライン数式モードにすると、
6  $a = -2x - 1$ となります。
7  また、
8  \[ a = -2x - 1 \]
9  とすると別行立て数式になります。
10 \end{document}
```

出力

テキストモードだと、a = -2x - 1 という出力になってしまいますが、ドルで囲んでインライン数式モードにすると、$a = -2x - 1$ となります。また、

$$a = -2x - 1$$

とすると別行立て数式になります。

後述の amsmath パッケージには、他にも便利な別行立て数式の環境があります[29]。

(6.2) **累乗、添字**　上付きの**累乗**は「^」、下付きの**添字**には「_」を用います。累乗や添字が 1 文字でない場合は、{ ··· } で**グループ化**します。 累乗や添字も入れ子にできます。

入力	出力
e^x + e^{-x}	$e^x + e^{-x}$
a_1^{b_1} + a_2^{b_2}	$a_1^{b_1} + a_2^{b_2}$

(6.3) **点**　1 つの点や複数の点を出力する以下のような命令があります。すべて数式モードでのみ使えます。これらを用いると前後の空白を自動調節してくれるので、全角の「・」や「…」で代用すべきではありません。

\ldots と\cdots の使い分けは、下の表にあるものが正当とする考えが普通ですが、日本の高校までの教科書に見られるように、すべて\cdots で済ませる場合もあります。

[28] 今まで使っていた、数式ではないモードは、**テキストモード**と呼びます。

[29] amsmath パッケージを使う方がおすすめです。複数行に渡る数式や、それらをイコールの位置で揃えたいだとか、数式に番号を付けたいときは、多くの改善が施された amsmath パッケージを必ず使って下さい。

入力	意味	出力	使用例
\cdot	掛け算記号の点など	$\cdot$	$a \cdot b$
\cdots	長い演算の途中の省略	$\cdots$	$a + b + \cdots + z$
\ldots	長い列挙の途中の省略	$\ldots$	$a, b, \ldots, z$
\vdots	縦方向の省略	$\vdots$	
\ddots	斜め方向の省略	$\ddots$	

(6.4) **根号** 「\sqrt{根号の中身}」でルートの記号を出力できます。3 乗根などを出力する場合は、「\sqrt[3]{根号の中身}」のようにします。

入力	出力
\sqrt{x^2+1}	$\sqrt{x^2+1}$
\sqrt[3]{a_1 + a_2}	$\sqrt[3]{a_1 + a_2}$

(6.5) **分数** 「\frac{分子}{分母}」で分数を出力できます。入れ子にすれば繁分数も可能です。この命令は、インライン数式と別行立て数式で文字の大きさなどが異なります。

入力	出力 (インライン)	出力 (別行立て)
\frac{1}{2}	$\frac{1}{2}$	$\frac{1}{2}$
\frac{1+x}{\sqrt{1+x^2}}	$\frac{1+x}{\sqrt{1+x^2}}$	$\frac{1+x}{\sqrt{1+x^2}}$
1+\frac{2}{3+\frac{4}{5}}	$1+\frac{2}{3+\frac{4}{5}}$	$1+\frac{2}{3+\frac{4}{5}}$

(6.6) **シグマ記号** 和のシグマ記号は「\sum」で出力します。「_」と「^」で、パラメータの動く範囲を書くことができます。この命令は、インライン数式と別行立て数式で記号の大きさや、範囲の付く場所が異なります。

入力	出力 (インライン)	出力 (別行立て)
\sum_{k=1}^n k^2	$\sum_{k=1}^{n} k^2$	$\sum_{k=1}^{n} k^2$

またシグマ記号と同様の振舞いをする大型演算子には以下のようなものもあります。

入力	出力	入力	出力	入力	出力
\prod	$\prod$	\bigcap	$\bigcap$	\bigcup	$\bigcup$

(6.7) **積分記号** 積分記号は「\int」です。積分の上端・下端は「^」と「_」で指定します。この命令は、インライン数式と別行立て数式で記号の大きさや、範囲の付く場所が異なります。

また、そのままだと被積分関数と「dx」などの間が詰まりすぎるので、少しの空白を空ける命令「\,」を挿入するのが適当です。

入力	出力 (インライン)	出力 (別行立て)
\int x dx % 詰まりすぎ	$\int xdx$	$\int xdx$
\int x \, dx	$\int x\,dx$	$\int x\,dx$
\int_0^{10} \sqrt{x} \, dx	$\int_0^{10} \sqrt{x}\,dx$	$\int_0^{10} \sqrt{x}\,dx$

(6.8) **課題 56 – 数式その 1** 次のような出力を与える tex ファイル (ファイル名は 0000 和地-56.tex のように) を作り提出して下さい。

出力

r が 1 ではないとき、等比数列の和の公式は、

$$a + ar + ar^2 + \cdots + ar^{n-1} = \sum_{k=0}^{n-1} ar^k = \frac{a(1-r^n)}{1-r}$$

である。また、$\frac{1}{\sqrt{x}}$ の不定積分は次のようになる。

$$\int \frac{1}{\sqrt{x}}\,dx = \int x^{-\frac{1}{2}}\,dx = -2x^{\frac{1}{2}} + C = -2\sqrt{x} + C$$

(6.9) log **型関数** 対数関数を「`$log x$`」と入力してしまうと、「$logx$」となります。しかし、数学の習慣では log 等のできあいの関数名は斜体ではなくローマン体で書きます。また、x の前に適切な空きもありませんから、決してこうしないで下さい。

正しくは「`\log`」命令を使い、「`$\log x$`」とします。主な log 型の関数を以下に記します。

入力	出力	入力	出力	入力	出力
`\log x`	$\log x$	`\arg x`	$\arg x$	`\exp x`	$\exp x$
`\sin x`	$\sin x$	`\deg x`	$\deg x$	`\lim x`	$\lim x$
`\cos x`	$\cos x$	`\det x`	$\det x$	`\max x`	$\max x$
`\tan x`	$\tan x$	`\dim x`	$\dim x$	`\min x`	$\min x$

シグマ記号と同様に「`^`」や「`_`」を用いると、上限、下限の数式を添えられますが、関数によって、あるいは、インラインか別行立てかによって上限、下限の付く場所が異なります。

入力	出力 (インライン)	出力 (別行立て)
`\log_a x`	$\log_a x$	$\log_a x$
`\lim_{n\to\infty} a_n`	$\lim_{n\to\infty} a_n$	$\lim_{n\to\infty} a_n$

`\to` と`\infty` は、それぞれ矢印と無限大の記号を出力する命令です。

(6.10) **mod** mod には、使い方に合わせて 2 通りの命令があります。

入力	出力
`x \bmod 3`	$x \bmod 3$
`x \equiv 4 \pmod{3}`	$x \equiv 4 \pmod 3$

(6.11) **賢いかっこ** 分数など縦に大きな数式に合わせて、かっこを自動的に伸縮できます。開きかっこは「`\left(`」、閉じかっこは「`\right)`」とすると、その間の数式の高さに合わせてかっこが伸縮します。「`{ }`」や「`[ ]`」など他のかっこも「`\left, \right`」をつけると伸縮します。ただし、ブレースは特殊文字なので、`\left{` ではなく `\left\{` と、バックスラッシュを前置します。 また、左右の片方だけにかっこを出力したいときは、「`\left.`」のように、出力しないかっこの代わりにピリオドを用います。以下では別行立て数式の出力のみ掲げます。

入力	出力 (別行立て)
`\{ \frac{1}{2}, \frac{3}{4} \} % 不釣合い`	$\{\frac{1}{2}, \frac{3}{4}\}$
`\left\{ \frac{1}{2}, \frac{3}{4} \right\}`	$\left\{\frac{1}{2}, \frac{3}{4}\right\}$
`\left( \frac{1}{2} \right)`	$\left(\frac{1}{2}\right)$
`\left[ \frac{1}{2} \right.`	$\left[\frac{1}{2}\right.$

(6.12) **数式の表組み** 地の文では tabular 環境を用いて表組みしましたが、数式モードでは array 環境を用います。列指定などは tabular 環境のときと同じです。

入力

```
\[
  \left(
    \begin{array}{cc}
      1 & 2 \\
      3 & 4
    \end{array}
  \right),
  \left\{
    \begin{array}{l}
      2x + 3y = 5 \\
      x - 6y = -5
    \end{array}
  \right.
\]
```

出力

$$\left(\begin{array}{cc} 1 & 2 \\ 3 & 4 \end{array} \right), \left\{ \begin{array}{l} 2x + 3y = 5 \\ x - 6y = -5 \end{array} \right.$$

ただし、行列や連立方程式は、後述の amsmath パッケージを使う方がすっきりと書けます[30]。

(6.13) **上下に付けるもの** 数式モードだけで使えるアクセント記号には、おもに以下のものがあります。

入力	出力	入力	出力
\hat{a}	$\hat{a}$	\tilde{a}	$\tilde{a}$
\bar{a}	$\bar{a}$	\vec{a}	$\vec{a}$

以下のものは、数式の長さに応じて伸縮します。

入力	出力	入力	出力
\overline{a+b}	$\overline{a+b}$	\underline{a+b}	$\underline{a+b}$
\widehat{a+b}	$\widehat{a+b}$	\widetilde{a+b}	$\widetilde{a+b}$
\overbrace{a+b}	$\overbrace{a+b}$	\underbrace{a+b}	$\underbrace{a+b}$
\overleftarrow{a+b}	$\overleftarrow{a+b}$	\overrightarrow{a+b}	$\overrightarrow{a+b}$

\overbrace と underbrace は、上限・下限が真上・真下に付きます。

入力	出力
\overbrace{a+\cdots+z}^{26}	$\overbrace{a+\cdots+z}^{26}$
\underbrace{a+\cdots+z}_{26}	$\underbrace{a+\cdots+z}_{26}$

(6.14) **その他の記号** これまでに紹介したもの以外にも、記号を出力する命令が大量に定義されています。おもなものを以下に記します。

矢印

入力	出力	入力	出力
\leftarrow	$\leftarrow$	\rightarrow	$\rightarrow$
\uparrow	$\uparrow$	\downarrow	$\downarrow$
\updownarrow	$\updownarrow$	\leftrightarrow	$\leftrightarrow$
\Leftarrow	$\Leftarrow$	\Rightarrow	$\Rightarrow$
\Uparrow	$\Uparrow$	\Downarrow	$\Downarrow$
\Updownarrow	$\Updownarrow$	\Leftrightarrow	$\Leftrightarrow$
\nearrow	$\nearrow$	\swarrow	$\swarrow$
\searrow	$\searrow$	\nwarrow	$\nwarrow$

[30] 必ず amsmath パッケージを使って下さい。

ギリシア文字

入力	出力	入力	出力	入力	出力
\alpha	α	\beta	β	\gamma	γ
\delta	δ	\epsilon	ϵ	\zeta	ζ
\eta	η	\theta	θ	\iota	ι
\kappa	κ	\lambda	λ	\mu	μ
\nu	ν	\xi	ξ	o	o
\pi	π	\rho	ρ	\sigma	σ
\tau	τ	\upsilon	υ	\phi	ϕ
\chi	χ	\psi	ψ	\omega	ω
\varepsilon	ε	\varphi	φ		
\Gamma	Γ	\Delta	Δ	\Theta	Θ
\Lambda	Λ	\Xi	Ξ	\Pi	Π
\Sigma	Σ	\Upsilon	Υ	\Phi	Φ
\Psi	Ψ	\Omega	Ω		

二項演算子

入力	出力	入力	出力	入力	出力	入力	出力
\pm	$\pm$	\mp	$\mp$	\times	$\times$	\div	$\div$
\circ	$\circ$	\cap	$\cap$	\cup	$\cup$		

関係演算子 [31]

入力	出力	入力	出力	入力	出力	入力	出力
<	$<$	\subset	$\subset$	\in	$\in$	\equiv	$\equiv$
>	$>$	\supset	$\supset$	\ni	$\ni$	\cong	$\cong$
\le	$\le$	\subseteq	$\subseteq$	\notin	$\notin$	\sim	$\sim$
\ge	$\ge$	\supseteq	$\supseteq$	\neq	$\neq$	\perp	$\perp$

[31] \le と \ge は、less than or equal to や greater than or equal to が由来。

その他

入力	出力	入力	出力	入力	出力
\infty	∞	\partial	∂	\ell	ℓ

(6.15) **課題 57 – 数式その 2** 次のような出力を与える tex ファイル (ファイル名は 0000 和地-57.tex のように) を作り提出して下さい。

出力

$0^\circ \le \theta < 180^\circ$ とすると $\sin\theta \ge 0$ だから、$\sin\theta = \sqrt{1-\cos^2\theta}$ である。また、$a, b \in \{3n+1 \,|\, n = 1, 2, \ldots\}$ のとき、$a + b \equiv 2 \pmod 3$ である。

$$\lim_{n\to\infty} \left(1 + \frac{1}{n}\right)^n = e.$$

(6.16) **行列、行列式** 行列を記述する、より簡単な方法があります。\left(, \right) と array 環境の組合せによる記述と比較すると、はるかに簡潔であることがわかります。

入力

```
1  \documentclass{jarticle}
2  \usepackage{amsmath}
3  \begin{document}
4  \[
5    \begin{pmatrix}
6      1 & 2 & 3 \\
7      4 & 5 & 6
8    \end{pmatrix},
9    \begin{vmatrix}
10     a & b \\
11     c & d
12   \end{vmatrix}
13 \]
14 \end{document}
```

出力

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}, \begin{vmatrix} a & b \\ c & d \end{vmatrix}$$

pmatrix 環境は上のように使用しますが、\begin{document}の前に\usepackage{amsmath}と記述して、amsmath パッケージを読み込む必要があります。後述のように amsmath パッケージには、他にも便利な数式の命令が多数定義されています。

また、vmatrix 環境は行列式のための環境です。

(6.17) **場合分けのある数式** 連立方程式や場合分けを記述するための cases 環境が、amsmath パッケージに定義されています。\left\{, \right. と array 環境の組み合せによる記述と比較して下さい。

入力

```
1  \documentclass{jarticle}
2  \usepackage{amsmath}
3  \begin{document}
4  \[
5  \begin{cases}
6    2x + 3y = 5 \\
7    x - 6y = -5
8  \end{cases}
9  \]
10 \end{document}
```

出力

$$\begin{cases} 2x + 3y = 5 \\ x - 6y = -5 \end{cases}$$

(6.18) **複数行の別行立て数式** 長い式変形や複数行の別行立て数式のための環境が、いくつか amsmath パッケージに定義されています。これまでに学んだものと合わせてまとめると、次のようになります。再び注意ですが、1 つの数式には**これらの 1 つのみ**を使います。

- インライン数式: $ ··· $
- 別行立て数式: \[··· \]
- 複数行で揃えのある別行立て数式: \begin{align} ··· \end{align}
- 複数行で中央揃えの別行立て数式: \begin{gather} ··· \end{gather}

つまり、

```
\[ \begin{align} y = -2x - 1 \end{align} \]   (間違い)
```

のような使い方はしません (最初の \[と最後の \] が余分です)。

入力

```
1  \documentclass{jarticle}
2  \usepackage{amsmath}
3  \begin{document}
4  \begin{gather}
5    x^2+x+1 = 1 \\
6    y = 2
7  \end{gather}
8  \begin{align*}
9    f(x) &= x^2+x+1 \\
10   y &= 2
11 \end{align*}
12 \begin{align}
13   (\sqrt{x})'
14   &= (x^{\frac{1}{2}})' \\
15   &= \frac{1}{2} x^{-\frac{1}{2}} \\
16   &= \frac{1}{2\sqrt{x}}
17 \end{align}
18 \end{document}
```

出力

$$x^2 + x + 1 = 1 \tag{1}$$
$$y = 2 \tag{2}$$

$$f(x) = x^2 + x + 1$$
$$y = 2$$

$$(\sqrt{x})' = (x^{\frac{1}{2}})' \tag{3}$$
$$= \frac{1}{2} x^{-\frac{1}{2}} \tag{4}$$
$$= \frac{1}{2\sqrt{x}} \tag{5}$$

gather 環境は中央揃え、align 環境は&の位置で揃えられます。こういった数式の揃えは amsmath パッケージを使わなくても可能ではありますが、amsmath パッケージを利用するのが一番良い方法です。gather, align 環境とも数式番号が表示されます。これは*が付加されると抑制されます。

(6.19) **課題 58 – 数式その 3** 次のような出力を与える tex ファイル (ファイル名は 0000 和地-58.tex のように) を作り提出して下さい。

出力

$$\begin{vmatrix} 1 & 3 & 2 \\ 3 & 4 & 5 \\ 1 & -1 & 3 \end{vmatrix} = \begin{vmatrix} 1 & 3 & 2 \\ 0 & -5 & -1 \\ 0 & -4 & 1 \end{vmatrix}$$
$$= \begin{vmatrix} -5 & -1 \\ -4 & 1 \end{vmatrix}$$
$$= -5 \cdot 1 - (-1) \cdot (-4)$$
$$= -9$$

§7 相互参照・目次 (・索引)

(7.1) **相互参照** 「第 7 節を参照のこと」とか、「33 ページにある式 (2) により」のように節などの番号を文中に記述したいとき、じかに「32 ページ」のように書くと、原稿の修正でページがずれたときや、節を新たに挿入したときに手作業ですべて直す必要があります。\label, \ref, \pageref の命令を使うと、TEX が自動的に番号を適切に決定してくれます。

入力

```
1  \documentclass{jarticle}
2  \usepackage{amsmath}
3  \begin{document}
4
5  \section{目次と索引}
6  \begin{align}
7    \label{eq:parab}
8    y = x^2
9  \end{align}
10
11 \section{相互参照}
12 \label{sec:sogo-sansho}
13
14 \subsection{節番号やページ番号}
15 \label{subsec:setsu-page}
16 ここは、第\ref{sec:sogo-sansho}節の
17 \ref{subsec:setsu-page}です。
18 式(\ref{eq:parab})は
19 \pageref{subsec:setsu-page}ページにあります。
20
21 \end{document}
```

出力

1　目次と索引

$$y = x^2 \tag{1}$$

2　相互参照

2.1　節番号やページ番号

ここは、第 2 節の 2.1 です。式 (1) は 1 ページにあります。

align 環境の他にも、gather 環境のように式番号の付く環境では、相互参照が可能です。

(7.2)　**課題 59 – 相互参照**　TEX の相互参照の機能を用いて、次のような出力を与える tex ファイル (ファイル名は `0000 和地-59.tex` のように) を作り提出して下さい。

出力

1　解と係数の関係

$x^2 - x + 1 = 0$ の 2 つの解を α, β とすると、

$$\alpha\beta = 1, \quad \alpha + \beta = 1 \tag{1}$$

であるから、

$$\alpha^2 + \beta^2 = (\alpha + \beta)^2 - 2\alpha\beta \tag{2}$$
$$= 1 - 2 \cdot 1 \tag{3}$$
$$= -1 \tag{4}$$

である。

2　補足

第 1 節の (2) 式から (3) 式への変形では (1) 式を用いた。

2 つ目の別行立て数式では、align 環境を用います。また 1 つ目の別行立て数式の、2 つの式の間の空白は、`\quad` 命令で空けます。これは以前用いた`\,` 命令よりも広い空きを作ります。

(7.3)　**目次**　`\tableofcontents` 命令を書くと、そこに目次が作成されます。

入力

```
1 \documentclass{jarticle}
2 \begin{document}
3 \tableofcontents
4 \section{目次と索引}
5 最初の節
6 \section{相互参照}
7 次の節
8 \subsection{節番号やページ番号}
```

```
9  いろいろ書く。
10 \end{document}
```

出力

目 次

1 目次と索引

最初の節

2 相互参照

次の節

2.1 節番号やページ番号

いろいろ書く。

§8 画像

(8.1) **画像の挿入** ここでは、Windows の texworks を用いている場合に、画像ファイルを表示させることを説明します。以下では、JPEG 画像ファイル (拡張子は.jpg, .jpeg) の場合を説明しますが、他に PNG, BMP, PDF, EPS 形式などの画像ファイルも同じ方法で表示できます[32]。

[32] textworks 以外の処理系を用いて JPEG, PNG, BMP, PDF 形式の画像を表示させる場合、拡張子が.xbb であるファイルを準備する必要があります。このファイルには画像のサイズが記録してあります。texworks では、コンパイルすると、必要な場合は.xbb ファイルを自動的に生成しており、ソースファイルを保存してあるフォルダを見ると.xbb ファイルがあるのが確認できます。

TEX の文書に画像を挿入するには、graphicx パッケージが必要です (スペル注意)。pdf ファイルを作成するときには、dvipdfmx オプションが必要です。また、画像サイズを高精度で認識させるため hiresbb オプションを指定することができます。

graphicx パッケージを読み込む文法

```
\usepackage[dvipdfmx,hiresbb]{graphicx}
```

上のようにパッケージにオプションを指定して読み込むときは、複数のパッケージを同時に\usepackage[dvipdfmx,hiresbb]{graphicx,amsmath} と読み込むことはできません。\usepackage 命令を 2 度書いて別々にパッケージを読み込みます。

画像を表示させるには、表示させる場所で\includegraphics 命令を用います。画像の拡大縮小をする scale オプションが指定できます。下の 2 つ目は画像を 1.2 倍に拡大するオプションを指定した例です。

includegraphics 命令の文法

```
\includegraphics{画像ファイル名}
\includegraphics[scale=1.2]{画像ファイル名}
```

入力

```
1  \documentclass{jarticle}
2  \usepackage[dvipdfmx,hiresbb]{graphicx}
3  \begin{document}
4  原寸。
5  \includegraphics{streetcar.jpg}
6  \vspace{1cm}
7
8  0.2倍に縮小。
9  \includegraphics[scale=0.2]{streetcar.jpg}
10 \end{document}
```

出力

原寸。 

0.2 倍に縮小。 

(8.2) **課題 60 – 画像の挿入** 次のような出力を与える tex ファイル (ファイル名は 0000 和地-60.tex のように) を作り提出して下さい。画像ファイルは、fig60a.pdf と fig60b.pdf を用いて下さい。また、画像をそのままのサイズで表示すると出力が 1 ページに収まらないので、適切に縮小して出力を 1 ページに収めて下さい。

出力

1 調和級数

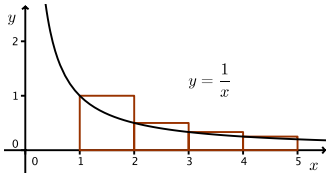
1.1 で調和級数が発散することを証明し、1.2 では、調和級数が $\log n$ のオーダーであることを証明する。

1.1 調和級数の発散

ここでは $y = \frac{1}{x}$ のグラフの下側の面積との比較をして証明する。下図より、

$$1 + \frac{1}{2} + \cdots + \frac{1}{n} \geq \int_1^{n+1} \frac{dx}{x} = \log(n+1) \tag{1}$$

である。$n \to \infty$ とすると、調和級数が無限大に発散することがわかる。



1.2 調和級数が $\log n$ オーダーであること

正整数 n に対して、

$$A_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n} - \log n,$$

と置き、数列 $\{A_n\}$ が収束することを示す。(1) 式より、

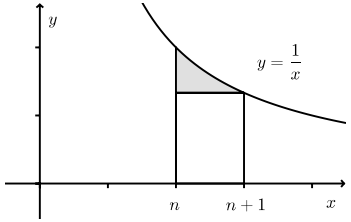
$$1 + \frac{1}{2} + \cdots + \frac{1}{n} - \log n \geq \log(n+1) - \log n,$$

$$A_n \geq \log \frac{n+1}{n}$$

となる。$(n+1)/n > 1$ だから、$A_n > 0$、つまり、数列 $\{A_n\}$ は下に有界である。また、

$$A_n - A_{n+1} = \log(n+1) - \log n - \frac{1}{n+1} = \int_n^{n+1} \frac{dx}{x} - \frac{1}{n+1}.$$

これは、下図の網掛け部分の面積であるから、$A_n - A_{n+1} > 0$、 つまり A_n は単調減少である。以上より、A_n は収束する。



§9 マクロ

(9.1) **マクロ定義** 一連の命令を組合せて自前の命令を作ることができます。TEX では自前の命令を**マクロ**と呼びます。

マクロ定義の文法

```
\newcommand{\命令の名前}{命令の内容}
```

\newcommand 命令は、プリアンブル (\documentclass のよりも後で、\begin{document}の前の部分) でも、本文でも用いることができますが、マクロ定義は、例えばプリアンブルにまとめると見易いでしょう。

下の例では、中央揃え、\Large の大きさ、ゴシック体で俳句を表示するマクロ \haiku を定義して、2 度使っています。

入力

```
\documentclass{jarticle}

\newcommand{\haiku}{
  \begin{center}
    \textgt{\Large かき氷削り機を逃し夏終わる}
  \end{center}
}

\begin{document}
ここで1句、\haiku
そして1句、\haiku
\end{document}
```

出力

ここで 1 句、

かき氷削り機を逃し夏終わる

そして 1 句、

かき氷削り機を逃し夏終わる

マクロの名前は、\Meirei のように半角アルファベット (大文字、小文字は区別されます) や、\命令なり のように和字が使えます。数字や記号は使える場合もありますが、必要がなければ用いない方がよいでしょう。

(9.2) **引数付きのマクロ定義** 上の例では、毎回同じ出力になりますが、マクロに引数を与えてマクロの実行ごとに異なる出力にすることも可能です。

引数付きマクロ定義の文法

```
\newcommand{\命令の名前}[引数の個数]{命令の内容}
```

引数の個数は 9 個が最大です。「命令の内容」の所で、#1, #2, ... とすると、1, 2, ... 番目の引数を参照します。

入力

```
\documentclass{jarticle}

\newcommand{\Haiku}[2]{
  \begin{center}
    \textgt{\Large #1や、ああ#1や、#1や}
  \end{center}
  \begin{flushright} #2 \end{flushright}
}

\begin{document}
ここで1句、\Haiku{松島}{松尾芭蕉ではないらしい}
そして1句、\Haiku{もろへい}{野菜の王様}
\end{document}
```

出力

ここで 1 句、

松島や、ああ松島や、松島や

松尾芭蕉ではないらしい

そして 1 句、

もろへいや、ああもろへいや、もろへいや

野菜の王様

(9.3)　**課題 61$^{\#}$ – マクロ (義務ではない)**　次のような出力を与える tex ファイル (ファイル名は 0000 和地-61.tex のように) を作り提出して下さい。

出力

まず 1 人目の俳句です。

ふきのとうひまわりよりも背高く

みずすまし

なかなか良いですね。では 2 人目の俳句はこちら。

きりぎりすひまわりよりも背高く

たかあしがに

ではまた来週。投稿の宛先はこちらです。

ただし、俳句の部分は、画像、俳句 (\Large の大きさ)、作者を出力するマクロを定義し、それを用いて出力して下さい。そのマクロは、初句と作者を引数で与えるように定義して下さい。画像ファイルは himawari.jpg ですが、適当なサイズに縮小して使って下さい[33]。

§10　演習問題

(10.1)　**課題 62$^{\#}$ – 数式 (義務ではない)**　箇条書き環境では、\item [(1)] とすると見出しを変更できます。

出力

定理　G を有限群とする。

(1) G の既約表現の同値類の個数は、有限であり、G の共役類の個数に等しい。

(2) 既約指標は次の直交関係を満たす:

$$\sum_{g\in G}\chi(g)\eta(g^{-1}) = \begin{cases} \#G & (\chi=\eta \text{ のとき}),\\ 0 & (\chi\neq\eta \text{ のとき}).\end{cases}$$

$$\sum_{\lambda\in\widehat{G}}\chi_\lambda(g)\chi_\lambda(h^{-1}) = \begin{cases} \#G/\#C_g & (g \text{ と } h \text{ が共役であるとき}),\\ 0 & (g \text{ と } h \text{ が共役でないとき}).\end{cases}$$

ここで、C_g は g を含む G の共役類である。特に、$\lambda\in\widehat{G}$ に属する既約表現の次元を d_λ とすると、

$$\sum_{\lambda\in\widehat{G}} d_\lambda^2 = \#G.$$

となる。

[33] himawari.jpg は郵便局ホームページのフリーイラスト集より

(10.2)　**課題 63$^{\#}$ − 数式 (義務ではない)**　$\lfloor n/2 \rfloor$ の括弧の記号は、`\lfloor`, `\rfloor` で出力できます。

出力

2 次交代行列 $J_2^- = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$ を m 個対角線上に並べてできる $2m$ 次交代行列を I_{2m}^- とする:

$$I_{2m}^- = \begin{pmatrix} 0 & 1 & \cdots & 0 & 0 \\ -1 & 0 & & 0 & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & 0 & & 0 & 1 \\ 0 & 0 & \cdots & -1 & 0 \end{pmatrix}.$$

また、

$$I_{2m+1}^- = \begin{pmatrix} I_{2m}^- & 0 \\ 0 & 0 \end{pmatrix}$$

とおく。

命題　n 次交代行列 A に対して、その偶数次主座小行列のパフィアン $P_k(A) = Pf(A_{2k,2k})$ $(1 \leq k \leq \lfloor n/2 \rfloor)$ がどれも 0 にならなければ、

$$A = {}^tBI_n^-B \quad (B \in B_n)$$

と表される。

(10.3)　**課題 64$^{\#}$ − 数式、相互参照 (義務ではない)**

出力

1　代数学の基本定理

n を正整数とし、

$$f(z) = a_0z^n + a_1z^{n-1} + \cdots + a_n, \quad a_0 \neq 0, \quad n \geq 1 \tag{1}$$

とおく。このとき、n 次代数方程式

$$f(z) = 0$$

は、複素数の範囲で少なくともひとつの根を持つ。これを代数学の基本定理と呼ぶ。以下でこれを証明する。証明は、1.2 で完結する。

1.1　補題

R を正の数とする。$|z| = R$ をみたす複素数 z に対して、つねに $|f(z)| > |f(0)|$ となるならば、

$$f(z_0) = 0, \quad |z_0| < R$$

をみたす z_0 が存在する。

1.2　代数学の基本定理の証明

式 (1) より、

$$\begin{aligned} |f(z)| &= |z|^n \left| a_0 + \frac{a_1}{z} + \frac{a_2}{z^2} + \cdots + \frac{a_n}{z^n} \right| \\ &\geq |z|^n \left(|a_0| - \left|\frac{a_1}{z}\right| - \left|\frac{a_2}{z^2}\right| - \cdots - \left|\frac{a_n}{z^n}\right| \right) \end{aligned}$$

である。ところが、$a_0 \neq 0$ であるから、

$$\lim_{|z| \to \infty} |z|^n \left(|a_0| - \left|\frac{a_1}{z}\right| - \left|\frac{a_2}{z^2}\right| - \cdots - \left|\frac{a_n}{z^n}\right| \right) = +\infty$$

となるので、R を十分大きい正の数とすると、$|z| = R$ をみたす z に対してつねに

$$|f(z)| > |f(0)|$$

が成り立つことになる。ゆえに、1.1 の補題により、

$$f(z_0) = 0, \quad |z_0| < R$$

をみたす $|z_0|$ が存在する。(証明終わり)

(10.4) **課題 65$^{\#}$ – 表組み (義務ではない)**

出力

1　2 円の位置関係

2 円の半径を r, R $(r < R)$、中心間距離を d とすると、5 種類の位置関係がある。共通接線の数とともに表にまとめると次のようになる。

	位置関係				
	離れている	外接	交わる	内接	包含
	$d > R + r$	$d = R + r$	$R - r < d < R + r$	$d = R - r$	$d < R - r$
共通外接線	2	2	2	1	0
共通内接線	2	1	0	0	0

2　指数関数・対数関数の性質

底 a $(a > 0, a \neq 0)$ に対し、指数関数 $y = a^x$ が定義され、その逆関数である対数関数は、$x = \log_a y$ と定義される。関数の特徴を記す。

名称	記号	定義域	値域	増減
指数関数	a^x	実数全体	正の実数全体	単調増加 ($a > 1$ のとき) 単調減少 ($0 < a < 1$ のとき)
対数関数	$\log_a x$	正の実数全体	実数全体	単調増加 ($a > 1$ のとき) 単調減少 ($0 < a < 1$ のとき)

§11　付録: 主なエラーメッセージ

(11.1) **エラーの発生箇所**　エラーが発生した場合、次のような出力が表示されます。

エラー例

```
! Undefined control sequence.
l.3 $y = \loq
             x$
```

- 感嘆符「!」 で始まるのがエラーの名前です。
- `l.3` は、エラーが 3 行目で発生したことを示します[34]。
- 元のソースで同一の行に連続していた`\loq` と `x$` が、行を分けて表示されているのは、分けられた**直前**がエラーであるという意味です。

つまり、ソースファイルの 3 行目の`\loq` が、未定義命令 (undefined control sequence) であるというエラーです。この場合はスペルミスが原因です。

(11.2) **主なエラーメッセージ**　以下では、エラーメッセージが 1 行に収まらない場合は、適宜折り返して掲げます。

`begin` と `end` が対応していない

```
! LaTeX Error: \begin{itemize} on input line 3
  ended by \end{document}.
```

`\begin` で始めた環境を閉じ忘れているときに発生するエラーです。

プリアンブル以外でプリアンブル限定の命令を使用した

```
! LaTeX Error: Can be used only in preamble.
:
中 略
:
l.3 \usepackage
               {amsmath}
```

プリアンブル (`\documentclass` と`\begin{document}`の間) でしか使えない命令を、`\begin{document}`以降で用いるなどした場合に発生するエラー。

数式モードで使えない命令を使用した

```
! LaTeX Error: Command \item invalid in math mode.
```

[34] しかし、その行のずっと以前に誤りがあるものの、その行でやっと矛盾が生じたという場合には、修正すべき箇所はエラーの発生した行のずっと前という可能性もあります。

```
:
中略
:
l.4 $\item
           $
```

数式モードで使えない命令を使用した場合に発生するエラー。

未定義の環境を使用した

```
! LaTeX Error: Environment docment undefined.
:
中略
:
l.2 \begin{docment}
```

環境の名前をスペルミスした場合など、未定義の環境を使用した場合に発生するエラー。

改行できない所で改行した

```
! LaTeX Error: There's no line here to end.
```

段落先頭や、改行した直後で\\で改行しようとすると発生するエラー。

余分な閉じブレース（}）

```
! Argument of \texttt  has an extra }.
<inserted text>
                \par
l.3 {\texttt}
```

命令に引数が不足しているときなどに発生するエラー。

別行立て数式の終了の誤り

```
! Display math should end with $$.
<to be read again>

l.3 \[ abc $
```

\[で始めた別行立て数式を\] で終えていないというエラー。諸々の事情により「$$」で終えていないというメッセージになっていることに注意。

二重の下付き

```
! Double subscript.
l.3 $ a_b_
         c $
```

上の例だと a_{b_c}などと書くべき。

二重の上付き

```
! Double superscript.
l.3 $ a^b^
         c $
```

上の例だと a^{b^c}などと書くべき。

$が不足

```
! Missing $ inserted.
<inserted text>
               $
l.3 a_
      b
```

下付きや\frac など、数式モードでしか使用できない命令を用いたときに発生するエラー。数式モードを終了するための$を忘れたときにも発生する。

閉じブレース（}）が不足

```
! Missing } inserted.
<inserted text>
                }
l.3 $ \frac{1} $
```

開きブレースに対応する閉じブレースが不足しているときに発生するエラー。

閉じブレース（}）が余分

```
! Too many }'s.
```

```
<recently read> }

l.3 \texttt{abc}}
```

閉じブレースが多すぎるとき発生するエラー。

未定義命令

```
! Undefined control sequence.
l.3 $y = \loq
              x$
```

スペルミスなどが原因で、未定義の命令を使用したときに発生するエラー。

引数の終わりが不明

```
Runaway argument?
{2 $
! Paragraph ended before \frac was complete.
```

閉じブレースがないなどが原因で発生するエラー。

索 引