情報Ⅰ教材Ｂ
作成者：筑波大学４年　高田悠矢

※本教材は情報Ⅰのコンピュータとプログラミング分野を対象にした教材で、この範囲の問題を解けるようになるように作っていますが、その分野全てを網羅しているわけではありません

○アルゴリズム、プログラミングとは

・アルゴリズム：問題を解決するための方法や手順
・プログラム　：アルゴリズムをコンピュータが実行できる形式であらわしたもの
・プログラミング言語を用いてアルゴリズムをプログラムにする(プログラミングする)ことで、コンピュータは処理を実行することができる

★アルゴリズムの例
・出席番号順を作るアルゴリズム
- 誰かを基準にしてまず列(1 人)を作る。その後、1 人ずつ先頭から列に入って、相手より番号が大きいなら後ろへ進むのを、番号が大きい相手に会うか最後尾になるまで繰り返して、そこに入る。これを全員が行う。

・料理に対しての調理レシピ

○今回扱うプログラミング言語

・共通テストで使われる言語を用いる
- 数字、文字列、変数、演算子、日本語の構文　などの組み合わせ

・数字：0~9(半角)で作られる数。小数か整数かを明確に区別して扱う
- 例）0、10000、0.5

・真理値：真(正しい)か偽(間違い)の 2 つのみ。
- 例）3 は奇数であるは真、1 は 0 より小さいは偽

・文字列：「」または""で括られた文字全般
- 例)「ありがとう」、”it”、”0”、「a 年 b 組 1 番」

・変数：英数字(先頭は英字のみ)で作られる文字。使い方は数学の変数と似ていて、
　　変数に数字や文字列などを代入できる。
- 例）x、it、time、syojikinn、tennsuu、y0、y1

○演算子

・代入演算子：←
- 例）x←3 で、x に 3 を代入。a←a-1 は、a を 1 減らすという意味。
　　y が 5 のとき、y←y+y で、y に 10 を代入する。

・算術演算子：基本数学と同じ。割り算だけ注意。

➢ 足し算のみ文字列も扱える。しかし、文字列+数字はエラー

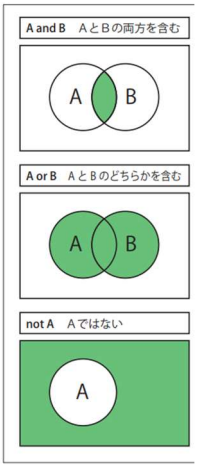
・足し算：+　　例：7+2 は 9、”a”+”b”は”ab”
・引き算：-　　例：7-2 は 5
・掛け算：X　　例：7X2 は 14
・割り算：/　　例：7/2 は 3.5
・整数の割り算　　例：7÷2 は 3
➢ 商：÷、余り：%　　7%2 は 1

・比較演算子：=, ≠, >, ≦ など。数学の使い方と同じ

➢ 結果は真理値(真か偽)で返される
➢ =と≠は文字列にも使える。英字の大文字、小文字も区別する

例：3≧3 は真、"A"="a"は偽、"Ani"="Ani"は真。
x%a=0 で、a の倍数かを確かめる(a の倍数なら真)。

・論理演算子：かつ(and)、または(or)、でない(not)
➢ 真理値を扱い、結果も真理値を返す
➢ A かつ B、A または B、A でない、のように使う(A,B は真理値)
・イメージ
A,B がそれぞれ真理値で、A,B の真の範囲を円の内側としたとき、
右の図の色が付いている部分がそれぞれの真となる範囲。
上から順にかつ(and)、または(or)、でない(not)



高等学校情報科「情報 I」教員研修用教材（本編） p.99 より抜粋

例：x が 5 のとき、x は 6 でないは真。また、
x%5≠0 かつ x≧0 (5 の倍数でなく 0 以上)は偽、
x<0 または x%2=0 (0 より小さい か 偶数)は偽。
x が 4 ならば、2 つとも真になる。

○表示文(print 文)

プログラムを実行しても、コンピュータは内部でその処理を実行するだけで、その結果などは表示しない。そのため、実行結果として表示してほしい場合はそれを宣言する必要がある。それが表示文で、「○を表示する」と書く。例えば、

・”x”を表示する　…　「x」と表示される
・x を表示する　…　変数 x の中身を表示する。x が 5 なら「5」、x が”Yes”なら「Yes」と表示される
・ "(" と x と ", " と y と")"を表示する　…　x が 5、y が－1 のとき、「(5, -1)」と表示される。

情報 I 教材 B
作成者：筑波大学 4 年　高田悠矢

○関数

プログラムの一部を機能単位で切り離したもの。数学で言う$f(x) = x^2 + 2x + 1$の場合、f が関数名で、x を引数、f(x)の値を戻り値という。引数は複数持てる。
どのような処理を行うかと同時に、引数、戻り値の把握が大切。

情報 I の範囲では、右のようにちゃんと定義することもあれば、次のように日本語で内容が定義されることもある

```
関数　<関数名>(<引数列>)を
｜　{処理}
と定義する
```

- 指定された値の二乗の値を返す関数「二乗」を用意する
- 値 m の n 乗の値を返す関数「べき乗(m，n)」を用意する
- 値 m 以上値 n 以下の整数をランダムに一つ返す関数「乱数(m，n)」を用意する
- 値 n が奇数のとき真を返し，そうでないとき偽を返す関数「奇数(n)」を用意する

例：

・ x←奇数(二乗(3))　…　中から処理を行うので、先に二乗(3)が行われて 9 が返され、次に奇数(9)が行われて真が返され、x に真が代入される

・ z←べき乗(x,y)＋乱数(1，6) …　z に「(x の y 乗)と(1~6 の整数のいずれか)の和」が代入される

・自分で定義する場合
右のように値を返さない関数も存在する。

```
関数　和を表示する(n,m)　を
｜　wa ← n+m
｜　wa を表示する
と定義する
```

○リスト(配列)

複数の値(変数)を、まとめて保持できるようにしたもの。1 つの名前と添字(index，索引)によって管理される。{(中身)}で表現され、リスト名[添字]などのように扱う。
添字がリストの大きさ以上だとエラーになる。添字は「0 から始まる」ことに注意。

例：生徒のテスト点数をまとめて保持したリスト Tokuten：{60,65,40,90,70,55,80}

添字	0	1	2	3	4	5	6
中身	60	65	40	90	70	55	80

・Tokuten[0]は 60、Tokuten[6]は 80 を表す
・リストの大きさは 7 という扱い。Tokuten[7]はエラーになる
※今回の言語では基本的に、リスト名は先頭が大文字(数字不可)の英数字で書かれる

情報 I 教材 B

作成者：筑波大学 4 年　高田悠矢

○条件分岐文(if 文)

(条件)が成り立つかどうかによって、{処理}を切り替える構文

・基本形1：もし (条件) ならば {処理1} を実行する

例… もし a≠0 ならば 100/aを表示する を実行する

➢ aが0のときは何もせず次に進み、aが0でないなら100/a(数字。aが2なら50、3なら33.3…)が表示される。

・基本形2：もし (条件) ならば {処理1} を実行し、そうでなければ {処理2} を実行する → C言語では、if (条件) {処理1} else {処理2} のように書く

例… もし a>0 ならば「100/a」を表示する を実行し、そうでなければ「NG」を表示する を実行する

➢aが 0 より大きいなら100/a(文字列。常に「100/a」という5文字)、aが0以下ならNGと表示される。

発展形：elseの所の処理に再びif文が入った状態。これもよく使われる。

```
もし (条件) ならば
｜ {処理1}
を実行し、そうでなくもし (条件) ならば
｜ {処理2}
…
を実行し、そうでなければ
｜ {処理n}
を実行する
```

例…

```
もし a<0 ならば 「負け」を表示する を実行し、
そうでなくもし a>0 ならば「勝ち」を表示する を実行し、
そうでなければ「引き分け」を表示する を実行する
```

➢ aが -1 のとき：実行結果として、 負け という文字が出力される
➢ aが 1 のとき：実行結果として、 勝ち という文字が出力される
➢ aが 0 のとき：実行結果として、 引き分け という文字が出力される

○条件繰返し文(while 文)

(条件)が成り立つ(真の)間、{処理}を繰り返す構文。
先に条件を確認するので、処理を一度も行わないこともある。

条件が成り立つ(真)の間、処理を繰り返す。　　書き方

例…

```
x←10
sum←0
x>0の間
｜　sum←sum+x
｜　x←x-1
を繰り返す。
```

```
(条件) の間
｜　{処理}
を繰り返す
※条件が偽になったら終了
```

➢xが10,9,…,1と、10回中身の処理が繰り返されて、sumは55になる。
➢ 最初が「x←-1, sum←0」だった場合、一度も中身の処理を実行せずsumは0のまま次へ進む

○順序繰返し文(for 文)

最初に宣言する構文の変数を変えながら、(条件)が成り立つ間、{処理}を繰り返す構文。

変数を条件を満たす間、
処理が1回終わる度に変数を
増やし(減らし)ながら処理を繰り返す。

書き方

```
i(変数)をOOからXXまで**ずつ増やしながら,
｜　(処理)
を繰り返す
```

例…

```
S←1
iを1から5まで1ずつ増やしながら,
｜　S←S X i
を繰り返す
```

i	1	2	3	4	5	終了(6)
S	1	2	6	24	120	120

```
n←1000
aを10から1まで2ずつ減らしながら,
｜　n←n / 2
を繰り返す
```

a	10	8	6	4	2	終了(0)
n	500	250	125	62.5	31.25	31.25

※ 下の段はi回目の処理を終えたときの値

情報Ⅰ教材Ｂ
作成者：筑波大学４年　高田悠矢

○代表的なアルゴリズム

探索アルゴリズムというものがある。探索とはリスト(配列)から求めるデータ(探索値)を探すことである。人間であればリスト全体を眺めてあるかどうかを判断できるが、コンピュータでは探す手順(アルゴリズム)を指定する必要がある。基本的に手順に沿って探索値とリストの n 番目を比較していって、一致するか、ないことが確定できたら終了する。2 種類紹介する。

・線形探索：リストの先頭から順に探索していく方法。見つけるか最後までいったら終了

引数に　探索対象のリスト：List　と探索値p を持ち、Listにpがあるなら「ある」と表示し、ないなら「ない」と表示する関数は右のように書ける

```
関数　線形探索(List, p)　を
|　n ← Listの大きさ
|　iを0からn-1まで1ずつ増やしながら
|　|　もしList[i]=pならば
|　|　|　「ある」と表示する
|　|　|　関数を終了する
|　|　を実行する
|　を繰り返す
|　「ない」と表示する
と定義する
```

A←{61,15,82,77,21,32,53}

リスト：

リストの要素	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
データ	61	15	82	77	21	32	53

例：線形探索(A,82)

・nは7となり、iを0から6繰り返すことになる。
・iが0のとき、List[0]=p (A[0](※61)=82)は
　偽だから次に進む(ループの先頭へ)。
・iが1のとき、15=82で偽。ループの先頭へ
・iが2のとき、82=82で真だから、
　もしの中身を実行する。
・「ある」と表示して、関数を終了する
(関数終了)

```
関数　線形探索(List, p)　を
|　n ← Listの大きさ
|　iを0からn-1まで1ずつ増やしながら
|　|　もしList[i]=pならば
|　|　|　「ある」と表示する
|　|　|　関数を終了する
|　|　を実行する
|　を繰り返す
|　「ない」と表示する
と定義する
```

A←{61,15,82,77,21,32,53}

リスト：

リストの要素	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
データ	61	15	82	77	21	32	53

例：線形探索(A,50)

・nは7となり、iを0から6繰り返すことになる。
・iが0のとき、List[0]=p (A[0](※61)= 50)は
　偽だから次に進む(ループの先頭へ)。
・iが1のとき、15=82で偽。ループの先頭へ
…
・iが6のとき、53=50で偽だから、次に進む。
(ループ終了)
・「ない」と表示する
(関数終了)

```
関数　線形探索(List, p)　を
|　n ← Listの大きさ
|　iを0からn-1まで1ずつ増やしながら
|　|　もしList[i]=pならば
|　|　|　「ある」と表示する
|　|　|　関数を終了する
|　|　を実行する
|　を繰り返す
|　「ない」と表示する
と定義する
```

・二分探索：リストを昇順(小さい順)にソートした(並び変えた)後、「探索範囲の中央値を基準に探索範囲を半分に減らす」のを繰返して探索していく方法。探索値を見つけるか、探索範囲が小さくならなくなったら終了する。線形探索の二分探索版の関数は下のように書ける。

```
関数　二分探索(List, p)　を
|　start ← 0　　//リストの左端
|　end ← (Listの大きさ) − 1　　//リストの右端
|　end≧start　の間
|　|　mid ← (start+end) ÷ 2　　//リストの中央
|　|　もしList[mid]=pならば
|　|　|　「ある」と表示する
|　|　|　関数を終了する
|　|　を実行し、そうでなければ
|　|　|　もしList[mid]>pならば end←mid-1　を実行し、
|　|　|　そうでないならば　start←mid+1　を実行する
|　|　を実行する
|　を繰り返す
|　「ない」と表示する
と定義する
```

○イメージ　…　0,1,2,…,8,9と整数が並んでいて、探索値が6の場合

➢探索範囲：

0	1	2	3	4	5	6	7	8	9

・中央値は4.5。中央値(4.5)<探索値(6)だから、中央より大きい方だけを見れば良い。

➢探索範囲：

					5	6	7	8	9

・今度は中央値が7。中央値(7)>探索値(6)だから、中央より小さい方だけを見れば良い。

➢探索範囲：

					5	6			

・今度は中央値が5.5。(省略)。探索範囲を5に減らす。

➢探索範囲：

					5				

・今度は中央値が5。中央値=探索値だから、探索終了！

二分探索

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
15	21	32	53	61	77	82

A←{15,21,32,53,61,77,82}

例：二分探索(A,82)

・startが0、endが6で、while文に入る
・1回目はmidが3(=(0+6)÷2)
・List[mid]=p (A[3](※53)=82)は偽 → ★の処理へ
・ List[mid]>p は偽だから、startが4(=mid+1)に
(ループ1回目終了)
・2回目はstartが4、endが6、midが5(=(4+6)÷2)
(省略)
・3回目はstartが6、endが6、midが6
・List[mid]=p (82=82)は真だから、「ある」と
　表示して、関数を終了する
(関数終了)

```
関数　二分探索(List, p)　を
|　start ← 0　　//リストの左端
|　end ← (Listの大きさ) − 1　　//リストの右端
|　end≧start　の間
|　|　mid ← (start+end) ÷ 2　　//リストの中央
|　|　もしList[mid]=pならば
|　|　|　「ある」と表示する
|　|　|　関数を終了する
|　|　を実行し、そうでなければ　　★
|　|　|　もしList[mid]>pならば end←mid-1　を実行し、
|　|　|　そうでないならば　start←mid+1　を実行する
|　|　を実行する
|　を繰り返す
|　「ない」と表示する
と定義する
```

1回目	15	21	32	53	61	77	82
2回目					61	77	82
3回目							82

二分探索

A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
15	21	32	53	61	77	82

A←{15,21,32,53,61,77,82}

例：二分探索(A,50)

・startが0、endが6で、while文に入る
・1回目はmidが3(=(0+6)÷2)
・List[mid]=p (A[3](※53)=82)は偽 → ★の処理へ
・List[mid]>p は真だから、endが2(=mid-1)に
(ループ1回目終了)
・2回目はstartが0、endが2、midが1(=(4+6)÷2)
(省略)
・3回目はstartが2、endが2、midが2
・end≧start (2≧3)が偽だからループ終了
・「ない」と表示する
・(関数終了)

```
関数　二分探索(List, p)　を
|　start ← 0　　//リストの左端
|　end ← (Listの大きさ) − 1　　//リストの右端
|　end≧start　の間
|　|　mid ← (start+end) ÷ 2　　//リストの中央
|　|　もしList[mid]=pならば
|　|　|　「ある」と表示する
|　|　|　関数を終了する
|　|　を実行し、そうでなければ　　★
|　|　|　もしList[mid]>pならば end←mid-1　を実行し、
|　|　|　そうでないならば　start←mid+1　を実行する
|　|　を実行する
|　を繰り返す
|　「ない」と表示する
と定義する
```

1回目	15	21	32	53	61	77	82
2回目	15	21	32				
3回目			32				
3回目終了							

○探索アルゴリズムの比較

どちらも正しく探索できそうな(証明は問われない)ので、この場合、時間(→計算量≒探索回数)が少ない方が良い。今回は、例えば、リストの大きさが 1000 の場合

・線形探索：先頭から 1 つずる見るから、最悪 1000 回の探索

・二分探索：探索範囲を半分にしていって、最終的に探索範囲が 1 になったら終了だから、

1000→500→250→125→63→32→16→8→4→2→1(→終了)で最悪 10 回の探索

となる。

一般に、リストが大きいときは明らかに二分探索の方が少なくなる。

➢　二分探索の方が良いことが「多い」

※しかし、リストをソートする時間が必要だったり、1 回の探索にかかる時間が比較的大きかったりなどのデメリットもある。場合によっては線形探索の方が良いこともあるので、絶対的に二分探索の方が良いという主張は偽になる。

○確認問題

問 1　右のプログラムについて、num が以下の場合、何が表示されるか答えよ。

a. num が 10 のとき
b. num が 3 のとき
c. num が-1 のとき
d. num が 4 のとき

```
もし（num % 10 = 0　または　num < 0 ）ならば
  |  {“a”を表示する}
を実行し、そうでなくもし（num % 2 ≠ 0）ならば
  |  {“b”を表示する}
を実行し、そうでなければ
  |  {num を表示する}
を実行する
```

問2　右のプログラムについて以下の問に答えよ。

a. 関数 A が呼ばれる回数
b. 関数 B が呼ばれる回数
c. 関数 A と関数 C の呼ばれる回数の関係について、正しい主張をしているのは誰か。
   ・一郎：A と C は同じ回数
   ・二郎：A は C より 1 回多い
   ・三郎：C は A より 1 回多い
   ・四郎：関数の中身を見ないと判断できない

```
i を 1 から 5 まで 1 ずつ増やしながら,
  |  関数 A(i)
  |
  |  j を 4 から 0 まで 2 ずつ減らしながら,
  |__  |  関数 B(i, j)
  |  を繰り返す
  |
  |  関数 C(i+1)
を繰り返す

*関数 A,B,C は正しく書かれていて、必ずこの繰り返し文の中に戻ってくる
```

情報 I 教材 B
作成者：筑波大学 4 年　高田悠矢

○並び替え問題

○解答(練習問題)

問 1　a: a、b: b、c: a、d: 4
問 2　a:5 回、b:15 回、c：一郎

○解答(練習問題)

問 1　a: a、b: b、c: a、d: 4
問 2　a:5 回、b:15 回、c：一郎