学習指導要領（2）- 知・技 - ア
学習指導要領（3）- 知・技 - イ
学習指導要領（3）- 思・判・表 - イ
学習内容（2）- ア メディアとコミュニケーション
学習内容（3）- イ アルゴリズムとプログラム

〔V〕以下の文章を読んで、設問（A）～ 設問（C）に答えなさい。ただし、この問〔V〕では、問題冊子の最後に示す記法を使ってプログラムを記述する。

あるコンピュータにセンサが接続されている。センサからは 0 ～ 100 （0 以上 100 以下）の整数値を読み取ることができ、コンピュータではこれを 1 バイト（8 ビット）のデータとして扱う。このセンサから 1 秒ごとにデータを 1 回読み取り、1 時間かけて得たすべての値をひとつのファイルに保存した。

このファイルからデータを読み込んで表示や計算を行うプログラムを作成する。

プログラム 1 は、ファイルに保存されたデータを 1 バイトずつ読み込んで、コンピュータの画面に整数値として表示するプログラムである。

ここで関数 eofData()は、ファイルから読み込めるデータがあるかどうかをチェックするものである。もう読めるデータがない（ファイルの終わりに到達した）場合に値 1 を、まだ読み込むことができるデータがある場合は 0 を返す。

関数 getData()はファイルからデータを読み込む。呼び出されるたびにファイルから 1 バイトずつデータを読み込んで 0 ～ 255 の整数値として返す。

```
var data # 変数の宣言
while eofData() == 0    # 読み込めるデータがある限り繰り返す
    data = getData()    # ファイルから 1 バイト読んで変数に代入
    print(data)         # 整数値を表示
end
```

プログラム 1

設問（A）

プログラム 2 は、上で述べたファイルから 10 個のデータを読み込むたびに、それらの平均値を計算して出力するプログラムである。

意図した通りに正しく動作するように、空欄を埋めなさい。

```
var sum, count, data
sum = 0
count = 0
while eofData() == 0
    data  = getData()
    [          ]
    [          ]
    if count == 10
        print(sum / 10)
        sum = 0
        count = 0
    end
end
```

プログラム 2

より長い時間、センサからデータを取得することにしたが、データを保存するファイルのサイズが巨大になってしまう。センサからは繰り返し 0 が返される傾向が高いことが分かっているため、これを利用してデータを 1 日の終わりに圧縮して保存することにした。

センサから得られるのは 1 バイトの整数データだが、100 よりも大きな値は出現しない。そこで、101 から 255 の範囲の整数値をデータの圧縮表現のために利用する。

圧縮方法を以下に示す。

データの圧縮方法

センサから読み取ったデータに値 0 が N 個連続して出現した場合、(257 − N) を計算し、その値を N 個の 0 の代わりに 1 バイトのデータとしてファイルに保存する。ただし、N は 2 以上、156 以下でなければならない。それ以外のデータと、連続しない 1 個だけの 0 はそのままファイルに保存する。N が 156 より大きい場合は、101 をファイルに保存して N の値を 156 減らすという手順を繰り返す。

たとえば、センサから取得した 0 5 0 0 0 0 0 1 というデータ列に上記の圧縮方法を適用することを考える。先頭の 0 はそのまま表すことになるが、途中に 5 個の 0 が連続しているため、 0 5 252 1 という 4 バイトで表現できる。

逆に、圧縮されたデータが 66 0 7 7 7 254 29 というバイトの列としてファイルに保存されていたとする。復元すると 66 0 7 7 7 0 0 0 29 というデータ列がセンサから得られていたことが分かる。

設問（B）

まず上のアルゴリズムで圧縮されたデータを復元する方法について考える。

プログラム 3 は圧縮されたデータをファイルから読み込み、復元してセンサから得られた元のデータ列を表示するプログラムである。

意図した通りに正しく動作するように、空欄をすべて埋めなさい。

```
var data, n, i
while eofData() == 0
    # 圧縮されたファイルからデータを読む
    data = getData()
    if data <= 100
        print(data)
    else
        n = [          ]
        for (i = 0; i < n; [          ])
            [          ]
        end
    end
end
```

プログラム 3

設問（C）

センサからのデータは圧縮せずにファイルに保存し、1 日の終わりに得られたデータを圧縮形式に直して、別のファイルに保存する。

プログラム 4 の関数 saveCompressedData() は、センサのデータを保存しているファイルからデータを読み取り、圧縮形式に直して別のファイルに書き込む。この関数では、処理の見通しを良くするために関数 putZeros() を用いている。これらの関数は戻り値を返さない。なお、関数 putData() は新しく作成する保存用ファイルに 1 バイトのデータを書き出す。

関数 saveCompressedData() が意図通りに動作するように、空欄をすべて埋めなさい。

データの圧縮方法（再掲）

センサから読み取ったデータに値 0 が N 個連続して出現した場合、（257 − N）を計算し、その値を N 個の 0 の代わりに 1 バイトのデータとしてファイルに保存する。ただし、N は 2 以上、156 以下でなければならない。それ以外のデータと 1 個だけの 0 はそのままファイルに保存する。N が 156 より大きい場合は、101 をファイルに保存して N の値を 156 減らすという手順を繰り返す。

```
func putZeros(n)
    # n個の連続する 0 を処理する
    var count, data
    count = n
    while count > 156
        putData(101)
        count = count - 156
    end
    if count > 0
        if count == 1
            data = [          ]
        else
            data = [          ]
        end
        putData(data)
    end
end
```

```
func saveCompressedData()
    var length, data
    length = 0
    while eofData() == 0
        # 圧縮されていないデータを読む
        data = getData()
        if data == 0
            [                  ]
        else
            if length > 0
                putZeros(length)
                [              ]
            end
            putData(data)
        end
    end
    if length > 0
        putZeros(length)
    end
end
```

プログラム 4

プログラムの記法の説明

本試験の問、および解答においてプログラムを記述するために用いる記法について説明する。

文

変数 = 式	変数に式の値を代入する。以下の説明ではこの文を「代入文」と呼ぶ。
for (代入文１; 条件式; 代入文２) ... end	まず、代入文１を実行し、条件式を評価する。条件式が偽であれば何もしない。条件式が真のとき、end までの命令を実行し、次に代入文２を実行する。その後、条件式が真である間、end までの命令と代入文２を実行する。
while 条件式 ... end	条件式が真である（条件が成立する）間、end までの命令（文の並び）を実行する。
if 条件式 ... end	条件式が真（条件が成立）のとき、end までの命令（文の並び）を実行し、偽（条件が不成立）のときは何もしない。

if 条件式
  ...(命令 1)...
else
  ...(命令 2)...
end
	条件式が真のとき、else までの部分（命令 1）を実行し、偽のときは else から end までの部分（命令 2）を実行する。

if 条件式１
  ...(命令 1)...
else if 条件式２
  ...(命令 2)...
else if 条件式３
  ...(命令３)
else
  ...(命令 n)...
end
	複数の条件式を順番に調べ、最初に真になった条件式に対応する命令だけを実行したい場合、else と if を組み合わせて左のように記述する（左は条件式が３つある場合）。どの条件式も偽の場合、最後に else があればその部分の命令（命令 n）を実行するが、なければ何もしない。
	曖昧さを避けるために「else if 条件式」の部分は１行に記述しなければならない。また、else は最後の部分として記述しなければならない。

break	for 文、または while 文（これらをループ文と呼ぶ）の内部でのみ利用できる。実行すると繰り返しの処理を打ち切り、ループ文の次の文の実行に移る。ループ文の中でループ文が使われている（ネストしている）時は、一番内側のループ文の処理だけが打ち切られる。
return 式 return	関数の内部でのみ利用できる。実行すると関数の実行を打ち切り、式の値を関数の評価値として呼び出し側に渡す。 戻り値の必要ない関数では、return の後の式を記述しない。
print(式)	式の値を表示する。整数値の場合、10 進数で表示する。

関数

func *F*(*x*)
  var *a*, *b*
  ...
  return *c*
end
	名前 *F* を持つ関数（サブルーチン）を定義する。*x* は引数で、実行の際に呼び出し側から値が渡される。引数を指定しない場合は ( )だけを記述し、複数の引数を指定する場合は「,」で区切る。
	var の後に変数名を記述して、関数の内部でのみ利用可能な変数（ローカル変数）を宣言できる。
	戻り値のない関数を定義することもでき、その場合、return 文の式を省略できる。さらに、関数の末尾の end の直前が戻り値のない return 文の場合、return 文自体を省略することができる。

F(式)	名前 *F* を持つ関数（サブルーチン）を実行する。式の値は引数として関数に渡される。式は、関数の定義で示された引数の数だけ記述する。引数が必要ない場合は ( ) だけを記述し、複数の引数がある場合は、foo(3, a+1) のように「,」で区切る。 戻り値のある関数は a = foo(3, a+1) − 1 のように式の中で呼び出すことができる。関数の値は、return で返される式の値である。

定数

整数	プログラムでは 10 進数の整数値を記述できる（例: 120, −1, +5）。
文字列	対になった " " で囲って文字列を表現できる（例: "ABC", "Level 5"）。 文字を 1 つも含まない文字列（空文字列）は "" のように表す。

変数と配列

var *a*, *b*[整数] （*a*, *b* は名前の例）	関数の内部で宣言した場合、その関数でのみ利用可能な変数（ローカル変数）となり、関数の外部で宣言するとどこからでも利用可能な変数（グローバル変数）となる。 名前の直後に [整数] を続けて記述すると、整数値が示す要素数を格納できる配列を宣言できる。ただし、値は正整数でなければならない。 変数名、配列名を「,」で区切って複数個が宣言できるが、その場合の変数、配列の初期値は不明な値になる。
var *a* = 定数	変数の宣言と同時に初期値を指定できる。グローバル変数の場合、プログラムの実行開始に初期値が与えられる。ローカル変数には、関数が実行される際に値が代入される。
var *b*[ ] = { 定数, ... }	変数の宣言と同時に初期値を指定できる。グローバル変数の場合、プログラムの実行開始に初期値が与えられる。ローカル変数には、関数が実行される際に値が代入される。 なお、この形式では配列の要素数を省略できる。
配列名[式]	宣言された配列の要素を代入先として指定したり、式の中で参照したりできる。配列 a の要素数が N (N は正の整数) のとき、配列 a は 0 番目から (N−1) 番目までの要素を持ち、i 番目の要素は a[i] と表現する。

算術演算子

整数値に対して算術演算を行うことができる。

+	加算（足し算）を行う。
−	減算（引き算）を行う。
*	乗算（掛け算）を行う。
/	除算（割り算）を行う。ただし、結果は実数で表される。
%	剰余算を行う。剰余算は割り算の余りを求める。例えば 7%3 は 1 となる。

また、- は式の先頭(左)に付けて -1 を乗じるのと同じ演算を表すことができる(例: -x)。

複数の算術演算子が混在した式では * と / の計算が + や − よりも先に行われる。
算術演算子と比較演算子が混在した式では、算術演算子が先に計算される。
式の中で( ) を使い、計算の順序を示すことができる。

比較演算子

整数値同士、文字列同士を比較できる。結果は真か偽のいずれかである。

A == B	AとBの値が等しい。
A != B	AとBの値が等しくない。

２つの整数値の大小を比較できる。結果は真か偽のいずれかである。

A <= B	Aの値がBの値以下である。
A < B	Aの値がBの値より小さい。
A >= B	Aの値がBの値以上である。
A > B	Aの値がBの値より大きい。

条件式

if 文やループ文の実行は条件式の評価結果（真または偽）で制御される。比較演算子を用いた値の比較は条件式である。また、条件式同士を AND、OR、または ! で組み合わせた式も条件式となる。

A AND B	２つの条件式AとBが両方とも真の場合のみ、結果が真になる。
A OR B	AとBのどちらか一方、または両方が真の場合、結果が真になる。
!A	条件式Aの否定を表す。つまり、Aが真の場合、!A は偽であり、Aが偽の場合、!A は真になる。

AND と OR が混在した条件式では AND の評価が OR よりも先に行われる。
AND、OR、または ! よりも算術演算子、比較演算子が先に計算される。
式の中で( ) を使い、計算の順序を示すことができる。

コメント

プログラムの記述中に # が現れた場合、そこから行末までの文字列はコメントとみなし、実行されない。ただし、文字列の中に現れた # はコメントとしては扱わない。

プログラムの例

(1) 右のプログラムで関数 main を実行すると、1＋2＋3＋… と加算を繰り返し、その値を表示する。合計が 100 を超えたら関数の実行は終わる。

関数 add()は値を返す関数、関数 accumulate()は値を返さない関数の例になっている。関数 accumulate()の最後に return 文はあってもなくてもよいことに注意。

```
var sum # グローバル変数の宣言

func add(n)
    sum = sum + n
    return sum
end

func accumulate()
    var i, s # ローカル変数
    sum = 0  # グローバル変数を初期化
    for (i = 1; i < 100; i = i+1)
        s = add(i)
        print(s)
        if s > 100
            return
        end
    end
end
```

(2) 1 から 9 までの整数に対応するローマ数字を表示するプログラム。ローマ数字の表記で用いられる 3 種類の文字を、関数の第 2 引数として配列で渡している。

複雑な if 文の例となっている。

```
func printRoman(arabic, figures)
    var d
    if arabic == 9          # 9の場合, "IX"
        print(figures[0])
        print(figures[2])
    else if arabic == 4     # 4の場合, "IV"
        print(figures[0])
        print(figures[1])
    else
        if arabic >= 5      # 5, 6, 7, 8 の場合, "V"
            print(figures[1])
            d = arabic - 5
        else
            d = arabic
        end
        while d > 0         # 必要なだけ "I" を表示
            print(figures[0])
            d = d - 1
        end
    end
end

var figs[] = [ "I", "V", "X" ]
printRoman(8, figs)         # "VIII" が表示される
```