U-16
旭川プログラミングコンテスト
資料

2017 年度版

目次

はじめに

今日は「U-16 旭川プログラミングコンテスト」において
あなた達が作成する競技用のプログラムについて、

- 思い通りに動くプログラムを作るには
- 賢いプログラムを作るには

以上の事について説明します。自分だけの誰にも負けないプログラムを作っていきましょう。

簡単なプログラムを作成するための「基本」と、より複雑で高度なプログラムを作成するための「応用」の 2 種類を用意しました。基本が理解できた方はぜひ、応用にチャレンジしてみてください。

基本 1:変数

```
a = 0                    ;変数の宣言
b = 5
c = a + b                ;変数の足し算(c=0+5=5)
```

プログラム 1 変数

変数名 = 値

値に名前を付け、自由に値を変更することができます。
変数は普通の数字のように四則演算を行うことができます。
変数には、文字を「"」で囲むことで文字も変数として扱えます。
新しい名前を作成するたびに新しい変数が作成されます。

表記	意味
A = B	A に B を**代入する**
A = A + B または **A += B**	A に B を加える（**足し算**）
A = A – B または **A –= B**	A から B を引く（**引き算**）
A = A * B または **A *= B**	A に B をかける（**掛け算**）
A = A / B または **A /= B**	A を B で割る（**割り算**）
A ++	A に**1を加える**
A --	A から**1を引く**

表 1 変数を扱った計算の書き方と意味

基本 2:コメントアウト

```
;←この記号から右側はすべてコメントとしてみなされます
;  コンピュータは「;」から右側に書かれた内容を無視します。
;          インデント(tab キー)を使うと
;                    段落を作ることができます
```

プログラム 2 コメントアウト

; コメント文

「;」から右側の文章はコンピュータがプログラムとして認識しません。必須ではありませんが、後から確認しやすいようにコメントアウトでメモを残しておくと便利です。

また、プログラムはコメントアウト外でも、**半角スペース**を全て無視して処理します。全角スペースは無視しないので注意してください。

基本 3:インデント

tab キーを押すとプログラムに段落を付けることができます。半角スペース同様、プログラムからは無視されます。
コメントアウトと半角スペース、インデントを利用すると後から見直してもプログラムの詳細が分かりやすくなります。

基本 4:配列

```
dim value,4              ;長さ 9 の 1 次元配列の宣言
value@.0 = 2             ;配列の 0 番に「2」を代入
value@.1 = 5             ;配列の 1 番に「5」を代入

dim value,2,3            ;縦 2×横 3 の 2 次元配列の宣言
multi(0,0) = 2           ;配列の(0,0)番に「2」を代入
multi(1,2) = 5           ;配列の(1,2)番に「5」を代入
value2 = 1,2,3,4,5,6,7   ;このような定義もできる
```

プログラム 3 配列

dim 変数名 , 配列の大きさ

変数をまとめたもの(1 次元配列)を作ります。配列を簡単に表すと、「**変数が入った部屋が複数連なっているもの**」です。
配列を使うことで、後述する for などで関連性を持った変数をひとまとめに操作することができ、便利です。
競技用プログラムにおいて受け取った情報は「長さ 10 の配列」として扱われています。

プログラム 3 のようにカンマで配列の大きさの項を増やすと、2 次元配列、3 次元配列・・・となります。これらは「配列が複数連なっているもの」と考えることができます。

変数名@.(要素の番号) (1 次元配列)

変数名(番号 1,番号 2,…) (2 次元配列以上)

上記のとおりに表記することで配列の(要素の番号)番の中身の変数を呼び出すことができます。ただし注意として、配列の大きさが **n 個**だった場合、(要素の番号)は **0 から n-1 番**までとなります。

value@.0	value@.1	value@.2	value@.3

配列 value 大きさ 9

multi(0 , 0)	(0 , 1)	(0 , 2)
(1 , 0)	(1 , 1)	(1 , 2)

配列 multi 大きさ 2 , 3

図 1 配列の図解

プログラム 3 を図に表すと、図 1 の通りになります。

基本 5:条件分岐

```
a = rnd(10)

if( a == 5 ){
        ;a は 5 です
}else : if ( a > 5 ){
        ;a は 5 より大きいです
}else{
        ;それ以外です
        ;5 でも、5 よりも大きくない
        ;        →5 以下
}
```

プログラム 4 条件分岐

変数の値などを条件にして、**処理の場合分け**をすることができます。プログラム 5 は「0~9 までのランダムな値 a が 5 である、5 ではないが 5 より大きい、それ以外の 3 パターンに分ける」というプログラムです。条件は一度に else:if で何個も合わせることが出来ます。

if(*条件式*) { 処理 }

*条件式*が満たされている場合、中括弧内の処理を実行します。満たされていない場合、中括弧内の処理を無視します。

if (*条件式 1*) { 処理 1 } else : if (*条件式 2*){ 処理 2 }

条件式 1 が満たされた場合は if 同様に処理 1 を実行し、*条件式 1* が満たされず、その代わりに*条件式 2* が満たされた場合は処理 2 を実行する。

if(*条件式*){処理 1}else{処理 2}

条件式 が満たされなかった場合、処理 2 を実行する。

書き方	条件式の意味
A == B　または　A = B	A と B が**等しい**
A > B	A が B **より大きい**
A >= B	A が B **以上**
A < B	A が B **より小さい**
A <= B	A が B **以下**
A != B	A が B **以外**
条件式 A　||　条件式 B	条件式 A か条件式 B どれか**1つでも満たしているとき**
条件式 A　&&　条件式 B	条件式 A と 条件式 B を**どちらも満たしているとき**

表 2 条件式の書き方と意味

基本 6:繰り返し処理

```
a = 0
while a < 10        ;a が 10 を超えるまで繰り返す
        a++         ;a に 1 を足す
wend                ;while まで戻る

b = 0
for i,1,5           ;i=1 から i=4 まで繰り返す（4 = 5 − 1）
        b++         ;b に 1 を足す
next
```

プログラム 5 繰り返し

while ***条件式***

処理

wend

・*条件式* が満たされている間、**while から wend の間**に囲まれた処理を**条件式が満たされなくなるまで**繰り返して実行します。

for 変数,初期値,最終値,増分

処理

next

for から next までの間を繰り返します。

一番初めに変数に初期値が代入されます。next に到達すると変数に増分(書かなかった場合は 1)を足して繰り返します。変数が最終値に達すると繰り返しを抜け出します。

```
a = 0

while a < 10        ;a が 10 未満なら繰り返す
        a--         ;a を 1 引く
                    ;a は 10 以上にならないので無限に繰り返す
wend                ;while まで戻る
```

プログラム 6 無限ループ

※無限ループについて

while から抜け出せる条件と処理を記述しないと while の中身の処理を無限に繰り返し、**無限ループ**を発生させます。無限ループはコンピュータのフリーズを引き起こすためとても危険です。

無限ループを起こさないように気を付けてプログラムを作りましょう。

プログラム 7 は無限ループの例です。

基本 7:簡単なプログラム

次ページに記述されているプログラム 8 は、図 2 のメモをもとに作成された「**進行方向先に壁があったら回避する**」という、プログラミングコンテストの中でも基本的なプログラムの一例です。

プログラムと共にコメントアウトで処理の説明を記述していますので、どのような動きをしているか確認してみましょう。

内容がある程度分かった方は「U-16AsahikawaProconClient2017」フォルダに含まれている「main.hsp」をコピー&ペーストし、プログラムを書いてみて実際に動きを見てみましょう。また、プログラムの一部を書き換えて、書き換えた箇所がどのような処理をしていたのかを確かめてみましょう。

なおこのプログラムは完全ではなく、**右と上に壁があった場合、上の壁に向かって移動してしまいます**。その理由と対策も考えてみましょう。

```
mode = “up”                    ;プログラムを始めた時には上
while ( checkgame == 1 )       ;ゲームが続いている限り回り続ける
        ;===1 ターンに必ず 1 回だけ実行します===
        ;余分に getReady を選ばないようにしています。
        getReady
        ;===壁があるかどうか、判定します===
        ;===壁があれば新しい方向を記憶させます===
        if(mode == “up” && Value@.2 == 2){      ;上に壁がある
                mode = ″left″          ;左を向かせます
        }
        if(mode == ″left″ && Value@.4 == 2){    ;左に壁がある
                mode = ″down″          ;下を向かせます
        }
        if(mode == ″down″ && Value@.8 == 2){    ;下に壁がある
                mode = ″right″         ;右を向かせます
        }
        if(mode == ″right″&& Value@.6 == 2){    ;右に壁がある
                mode = ″up″            ;上を向かせます
        }
        ;===記憶している方向に移動します===
        ;else if を用いて一度に 2 回行動を選ばないようにしています。
        if(mode == ″up″){              ;上
                walkUp                 ;上に移動します
        }else : if(mode == ″left″){    ;左
                walkLeft               ;左に移動します
        }else : if(mode == ″down″){    ;下
                walkDown               ;下に移動します
        }else : if(mode == ″right″){   ;右
                walkRight              ;右に移動します
        }
        ;===以下省略===
```

プログラム 7 壁を回避するプログラム

応用 1:ユーザー定義関数

HSP における**関数**とは、メインプログラムとは別の場所にてあらかじめ作成した処理を呼び出す機能です。

プログラムを書く上で同じ処理を何度も書くと行数が増え後のチェックが大変になります。そこで、一度に複数回扱う処理を関数として新たに作成し、メインプログラムから呼び出します。結果、プログラムの行数が少なくなり、複数回扱う処理に問題があった場合は関数を修正することにより一括で修正することが出来ます。

ユーザーが新しく作った関数は**ユーザー定義関数**といいます。

関数には**引数**という関数の外から与えることのできる値を元に結果を変化させることができ、**戻り値**という呼び出し元に対して与える値を設定することができます。戻り値の必要がない関数を HSP では**命令**と呼びます。

応用 2:関数の作成

#module　　#global

上記 2 つの文字列に囲まれた空間を**モジュール空間**と呼び、#global 以下の空間を**グローバル空間**と呼びます。モジュール空間は複数定義でき、#module ~ #global 間で囲った数だけ作成されます。

モジュール空間の中で作成した変数は作成したモジュール空間の中でのみ使用可能で、他のモジュール空間やグローバル空間に存在する同じ名前の変数とは別物として扱われます。

関数を作るときはモジュール空間に書くのが基本です。

#defcfunc 関数名 型 1 変数名 1, 型 2 変数名 2…

以上の記述で関数を定義することができます。命令の場合は

#defcfunc 命令名 型 1 変数名 1, 型 2 変数名 2…

となります。

関数の引数に用いる変数の種類を明確にするため、型という形で設定する必要があります。型の種類は表 3 の通りです。

型の名前	用いることのできる変数の種類(**パラメータ**)
int	**整数値**
double	**実数値**
array	**配列**
str	**文字列**
表 3 型の種類	

詳しい記述方法は次ページのプログラム 9 を参考にしてください。

return (値)

関数を終了させ、(値)に設定した戻り値を返します。関数の終わりに必ず return を行わなければなりません。行わなかった場合はエラーとなります。

命令の場合は戻り値が存在しないため、(値)が必要ありません。こちらも必ず命令の中で行う必要があり、行わなかった場合は命令の中身で無限ループが発生します。

```
#module                          ;ここからモジュール空間
#defcfunc nijo    int a          ;a を 2 乗(a×a)する関数
        return (a * a)           ;戻り値

#defcfunc nibai    int b         ;b が 5 以上なら 2 倍する関数
        if( b >= 5 ){            ;条件分岐
                return (b * 2)   ;5 以上なら b*2
        }
        else{                    ;5 以上ではない場合
                return b         ;そのまま返す
        }
        return -1                ;念のために置くと
                                 ;エラーを回避しやすい

#global                          ;ここからグローバル空間
a = nijo(5)                ;5 の 2 乗(5×5=25)を a に代入
                           ;nijo にて使っていた a とは別物
b = 0                      ;新しい変数 b
for i,1,11                 ;i=1 から i=10 まで繰り返す
        b += nibai(i)      ;i が 5 以上なら 2 倍して b に足す
next                       ;この繰り返しは以下と同じ意味です
                           ;b=1+2+3+4+5*2+6*2+7*2+8*2+9*2+10*2
```

プログラム 8 関数とモジュール空間

応用 3:配列の大きさ分繰り返す

```
arr = 1,2,3,4,5                  ;大きさ 5 の配列
foreach arr                      ;arr の大きさ(5 回)繰り返す
        arr.cnt += 10            ;arr の各要素に 10 足す
loop
```

プログラム 9 配列の大きさ分繰り返す

foreach (配列名) ~ loop

(配列名)の配列の大きさだけ、Foreach から loop までの間を繰り返します。for 文と異なり、

(配列名).cnt

と書くことで、foreach を繰り返すたびに配列の次の要素を選択し、参照してくれます。つまり、以下のプログラム 11 と同じ働きをしてくれます。

```
arr = 1,2,3,4,5                  ;大きさ 5 の配列
for i,0,length(arr)              ;arr の大きさ(5 回)繰り返す
        arr@.i += 10             ;arr の各要素に 10 足す
next
```

プログラム 10 プログラム 9 と同じ動きをするプログラム

プログラム 11 における **length(配列名)**は、配列の大きさを返してくれるものです。

応用 4:乱数

```
randomize                ;並びをランダムにする

a = rnd(20)              ;0~19 までの乱数を作成
```

プログラム 11 ランダムな値を生成する

rnd(値)

0 から(値)-1 までの間から**ランダムな値(乱数)**を出します。rnd を実行するごとに出力される値はランダムとなります。randomize を実行しないと、プログラムを起動するたびに同じ並びの数字が出てきます。

randomize

乱数の並びを新しいものとします。これにより、プログラムを実行するごとに異なった並びの乱数を扱えます。

一度に何度もこの処理を行うと不具合が発生することがあるので、なるべく繰り返さない箇所にこの処理を置く必要があります。

応用 5:ファイル分割

```
#module
#defcfunc hogege int x, int y       ;x×y を行う関数
        return (x * y)
#global
```

プログラム 12(1) function.hsp

```
#include "function.hsp"             ;「function.hsp」をインクルード

a = hogege(10, 30)                  ;「function.hsp」の関数を実行
```

プログラム 12(2) main.hsp

#include "ファイル名.hsp"

別のファイルを読み込み（**インクルード**）、一つのプログラムとして扱います。プログラム 12 では、function.hsp というファイルに作成した hogege という関数を main.hsp というファイルで function.hsp をインクルードする形で main.hsp の中身で関数 hogege を扱っています。

プログラムを記述したファイルを機能ごとなどに分けてファイル分割することでバグやエラーを取り除くときに探しやすくなります。

注意として#include でファイルをインクルードした場合、追加されたプログラムは#include より下に適用されるため、ファイルのできるだけ上にまとめて書かなければエラーが起きる可能性があります。

応用 6:数値と文字の変換

```
a = str(12345)              ;数値を文字に変換
b = int("12345")            ;文字を数値に変換

c = a + "です"              ;12345 です
d = b + 100                 ;12445
```

プログラム 13 数字と文字の変換

str(値)

数値を文字列に変換します。

int(文字列)

文字列を数値に変換します。数字以外の文字列は無視されます。

応用 7:マクロ

```
#define LOOPMAX 10          ;10 を LOOPMAX と置き換える

a = 10
for i,0,LOOPMAX             ;LOOPMAX(=10)回繰り返す
        a += i              ;a に i を足す
next

for i,0,LOOPMAX*2           ;LOOPMAX*2(=20)回繰り返す
        a += i              ;a に i を足す
next
```

プログラム 14 マクロ

#define (マクロ名) (マクロ定義)

(マクロ定義)に書いた内容を(マクロ名)として置き換えて表記できるようにします。コンピュータの中身では置き換えられたものは(マクロ定義)のままであるように振る舞います。

ひとつのプログラムに何回も使う値を一気に変えたい場合、#define で定義しておくことで実現することができます。

こちらも#include 同様、定義した箇所の下から適用されていくので、プログラムの上方に配置しておくべきです。