U-16
旭川プログラミングコンテスト
資料

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」で扱う事になる競技用プログラムについて、条件分岐や、繰り返しの処理の作り方に加え、キャラクターを動かすために何を書けば良いのか、基本の内容をこの資料では説明します。今回説明する処理を組み合わせて、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。

変数

```
2
3 ;←この記号から右側は全てコメントとして扱われます
4 ;  コメントはコンピュータからは一切見られません
5
6 a = 0    ;変数を宣言する
7
8 b = 5
9
10 c = a + b    ;変数の足し算
11
12
```

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。

意味	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 --**）	A に**1を加える**（A から**1を引く**）

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。

後から確認しやすいようにメモを残しておく場合に使用します。

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。

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。

```
2 a = 5
3 if (a == 5){
4 mes "5です"
5 a += 2
6 };インデントを使わない場合
7
8 a = 5
9 if (a == 5){
10     mes"5です"
11     a += 2
12 }                ;インデントを使用した場合
13
```

If 文「条件分岐」

```
1 pos 0,0
2
3 a = 5
4
5 if( a == 5 ){
6     mes "5です！"
7 }else : if ( a > 5 ){
8     mes "5じゃないけど5より大きい！"
9 }else{
10    mes "5でもないし5よりも大きくない！"
11 }
12
13
```

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処理 1 を実行し、
条件式 1 が満たされなかったが、*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。
else : if と組み合わせた場合、すべての *条件式* が満たされなかった
場合、else の後の中括弧内の処理を実行する。

書き方	条件式の意味
A == B **※「A=B」ではない**	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**

While 文「繰り返し処理」

```
1 pos 0, 0
2
3 a = 0
4
5 while a < 10
6     mes ""+a
7
8     a++
9 wend
```

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する。

※無限ループについて

while は while から wend までの間で条件式が満たされなくなるまで繰り返す処理だが、繰り返してもなおずっと条件が満たされている場合、その処理は無限に繰り返され、強制停止を　行わない限り止めることができなくなってしまう。

この現象を**無限ループ**といい、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。そのため、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。

```
2 a = 0
3
4 while a < 5
5     a --
6 wend
7
```

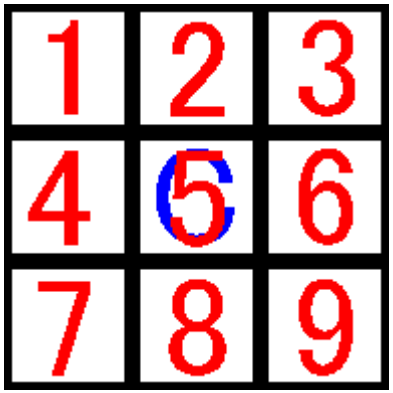
無限ループの一例

while～wend の中で
a<5 を満たし続けるため、
永遠に繰り返し続けてしまう。

競技プログラムで使用する関数

■関数：処理のまとまりのこと

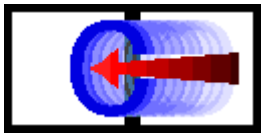
getReady



数値	状況
0	**何も存在しない**
1	**キャラクター**
2	**壁**
3	**アイテム**

キャラクターが1回行動する時、**必ず呼び出さなければならない**関数。
自身の位置と周囲8マスの状況を数値として取得することができます。
取得した状況は、「**Value@.数字**」（**数字**は上図参照）で1ターンにつき何度でも参照することができます。
この値を用いて if 文を使って条件を分けていきましょう。

walk



関数名	処理
walkUp	キャラクターを上に移動させる
walkLeft	キャラクターを左に移動させる
walkRight	キャラクターを右に移動させる
walkDown	キャラクターを下に移動させる

移動先に壁があっても進んでしまうため、壁と重なり負けとなってしまいます。よって壁の有無をキャラクターに教える必要があります。

put

関数名	処理
putUp	キャラクターの上に壁を設置する
putLeft	キャラクターの左に壁を設置する
putRight	キャラクターの右に壁を設置する
putDown	キャラクターの下に壁を設置する

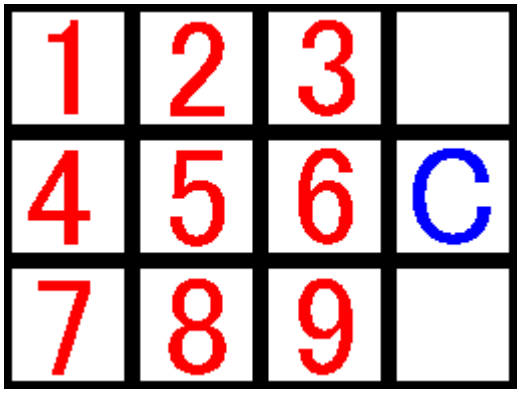
指定した方向に壁を設置する。敵の上に設置するのはもちろん、壁をいくつも設置することでステージの形を変えることができます。

search

関数名	処理
searchUp	キャラクターの上１列９マスを調べる
searchLeft	キャラクターの左１列９マスを調べる
searchRight	キャラクターの右１列９マスを調べる
searchDown	キャラクターの下１列９マスを調べる

取得した状況は getReady 同様「**Value@.数字**」に保存されます。
但し、次に getReady を使用した際に上書きされてしまうので、
値を残しておきたいときは別の変数に残す必要があります。
後述の look とは違い横幅は狭いものの遠くまで見ることができ、
移動先に何があるのかを確かめたい時に使うと便利です。

look



関数名	処理
lookUp	キャラクターの上３×３の９マスを調べる
lookLeft	キャラクターの左３×３の９マスを調べる
lookRight	キャラクターの右３×３の９マスを調べる
lookDown	キャラクターの下３×３の９マスを調べる

こちらも取得された状況は「**Value@.数字**」に保存されます。
search と違い近距離を広く調べることができるため、
近くをキャラクターが通るといった、付近の状況が素早く変わる
場面で使ってみるといいかもしれません。

さいごに

歩く、調べるなどの機能があっても、これらだけでは
壁に向かって走ってしまったり、敵が目の前を通っても
倒さずに素通りしてしまいます。これらを避けるには、
「近くに壁があるから進まないようにする」
「近くに敵がいるから壁を設置する」
「近くにアイテムがあるから取りに行く」
などの事柄をキャラクターに教える必要があります。
キャラクターに教えるには、変数や if 文を利用して、
「近くに壁がある」　→「ある」→「避ける」
→「ない」→「進む」
と処理と分岐させて作りましょう。

しかしこれらを踏まえてプログラムを書いていくと、
同じ道を何回も往復する、同じ場所から抜け出せない
現象が発生します（**ループ**）。
ループが発生しないようにするにはわざと壁を設置したり、
繰り返した回数を記憶して違った道を通らせるなど、
他にも沢山ループを回避するための対策は存在します。
あくまでも上記の対策法は一例なので、以上を踏まえて
どのような局面にでも対応することができる、
優れた自分だけのプログラムを目指してみましょう。