ゲームアルゴリズム資料

佐藤　陽悦[*1]

令和 3 年 6 月 7 日 Ver.0.6a

[*1] ゲームエンジニア科

1　本日のお題：アルゴリズムとは

1.1　アルゴリズムとプログラミング言語

この授業では、プログラム（最終的にはゲームプログラム）を作成するために有用となる、著名なアルゴリズムやアルゴリズムのためのデータ構造などについて学ぶ。最近では、テレビや雑誌、Web 記事などで、それほどコンピュータに精通した人でなくても、アルゴリズムという単語を使っていることも少なくない。（ただし正しい使い方がされているかどうかは謎）本来の意味でのアルゴリズムとは、以下のような意味である。

アルゴリズム (Algorithm)

問題を解決するための手順のこと。算法。プログラミング言語を使って、アルゴリズムを記述したものをコンピュータのプログラムと呼ぶ。

したがって、そもそも**プログラムを書く**と云う事は、「コンピュータに行わせたい処理」に対するアルゴリズムを考え、それをプログラミング言語を使って記述することである。

ここで用語を整理しておく。

（コンピュータ）プログラム　コンピュータがメモリに読込んで、直接実行できる形式の命令群
プログラミング言語　人間がコンピュータプログラムを楽に作成するために考えられた人工言語
ソースコード　プログラミング言語によって記述された、処理内容（アルゴリズム）のファイル
コンパイル　ソースコードを一括して実行形式（プログラム）に変換する処理のこと（≒ビルド）
アルゴリズム　コンピュータを使って問題を解くための、処理手順のこと

ここで、プログラミング言語には大きく分類して、**高級言語（高水準言語）**と**低級言語（低水準言語）**に分類される。低級言語は、コンピュータが直接理解できる**機械語（マシン語）**や、機械語を少しだけわかりやすくした**アセンブリ言語**などの、ハードウェアの仕様と密接な関係のある言語を指す。それに対し、高級言語は、それ以外の言語を指し、人間の使う言語に近い言語設計がなされているプログラミング言語である。かなり多くの種類があるが、よく聞くものでは `c/c++,java, c#, python` などを指す。

低級（低水準）言語　機械語（マシン語）などのハードウェアと密接な関係のある言語
- ハードウェアに近いため、ハードウェアの機能を有効に使うことができる
- ハードウェアに特有の処理を書くことができる
- ハードウェアに近いため、処理させたいハードウェア（CPU) の仕様に言語自体が依存する。
- 言語がほぼ機械語なので、人間には理解しづらい

高級（高水準）言語　C 言語のような人間の言語仕様に近い設計の言語
- 人間の言語に近いため、書いてあることを理解しやすい
- ハードウェア依存の処理を書く必要がない（異ハード間で同一言語が使える）
- ハードウェア依存の処理を書くには、外部のライブラリなどが必要な場合が多い
- 言語によっては、実行速度の面で不利な場合がある

1.2　アルゴリズムの記述法

これから先ゲームエンジニアとなる諸君は、与えられた問題を解いていくために、アルゴリズムを考え、ソースコードに記述していかなければならないが、アルゴリズムを記述する手法としては以下のようなものが考えられる。

1. 文章による記述
2. 図表による記述
3. 疑似言語、プログラミング言語による記述

以下、それぞれの記述法の例を見てみよう。

1.3　アルゴリズムの記述法 –文章による記述–

カップ焼きそばを食べるまでの処理

例題として、カップ焼きそばを食べるまでの処理を考えてみる。カップ焼きそばは一般的に、適切な位置までふたを開け、かやくと調味料を取り出し、かやくを袋から面の上にあけ、カップにお湯を注ぎ、ふたをして指定された時間面をゆで、お湯を湯切りから排出し、調味料をかけて、かき混ぜるところまで作業すると、おいしく食すことができる。
ここで、適切な温度のお湯と、未開封のカップやきそばはすでに用意してあるものとして考える。

文章による表記：【カップ麺を食べるまでの処理】

1. カップめんのふたを適切な位置まで開く
2. 中からかやく、調味料などを取り出す
3. かやくを袋から面の上にあける
4. 適量のお湯をカップにそそぐ
5. カップにふたをし、指定された時間待つ（面をゆでる）
6. カップのお湯を湯切りから排出する
7. 調味料（だいたいソース）をかけよく混ぜる
8. おいしく食す

入力された３値の最大値

キーボードから３つの整数値を入力し、その３つの値の最大値を出力するアルゴリズムを考える。
考え方として、変数 iMax、及び、３つの整数値を入力するための変数 $iVar_1, iVar_2, iVar_3$ を用意し、初期値として iMax を 0 で初期化しておく。$iVar_n$(n は１～３）が入力されるたびに、$iVar_n$ と iMax を比較し、iMax より iVar が大きかったら、iMax の値を $iVar_n$ で更新することを繰り返す。

文章による表記：【入力された３値の最大値】

1. iMax ← 0（初期化処理）
2. キーボードから整数値 $iVar_1$ を入力
3. 判定　$iVar_1 >$ iMax は真か？
   - 3-1： 真の時 iMax　←　$iVar_1$
   - 3-2： 偽の時 何もしない
4. キーボードから整数値 $iVar_2$ を入力
5. 判定　$iVar_2 >$ iMax は真か？
   - 5-1：真の時： iMax　←　$iVar_2$
   - 5-1：偽の時： 何もしない
6. キーボードから整数値 $iVar_3$ を入力
7. 判定　$iVar_3 >$ iMax は真か？
   - 7-1：真の時： iMax　←　$iVar_3$
   - 7-2：偽の時： 何もしない
8. iMax を出力して終了

入力された整数の和

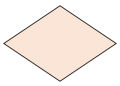
キーボードから正の整数値を入力し、値の和を出力するアルゴリズムを考える。入力は、負の整数値が入力されるまで繰り返される。
考え方として、変数 sum、整数値を入力するための変数 $iVar$ を用意し、初期値として sum を 0 で初期化しておく。$iVar$ が入力されるたびに、$iVar$ の正負判定をし、iVar が正の整数の時は sum を $sum + iVar$ で更新する。$iVar$ が負だった場合は、sum を表示して処理を終了する。

文章による表記：【入力された値の和】

1. sum $\leftarrow$ 0（初期化処理）
2. キーボードから整数値 $iVar$ を入力
3. 判定　$iVar > 0$
   - 3-1：真の時： sum　$\leftarrow$　sum + iVaswr
   - 3-2：2 へ戻る
   - 3-1：偽の時：何もしない (4 へ)
4. sum を出力して終了

1.4　アルゴリズムの記述法 –図表による記述–

次に、図表による記述をみてみよう。一番著名な図表によるアルゴリズムの記述法としては、流れ図（フローチャート）がある。基本的な機能を示す記号の中に、式や処理を記入して、線でつないで処理の流れを表す。基本的に、フローチャートでは、処理は上から下へ流れてゆくことが決まっているため、直感的に処理の流れをとらえることができる。以下に、基本的なフローチャート記号の一覧を示す。

1)		処理：処理内容を書く、基本的に 1 部品に 1 処理を書く。 $a \leftarrow b+c$ や a を b に代入 など処理の内容を簡潔に記述する。式を書くことが多い気がする。
2)		端子：処理の入口と出口を表す部品。開始、終了、サブルーチン名などを記述する。この部品で囲まれた部分が一つのまとまった処理を表すため、必ずフローチャートの上下に対で配置される（出口が複数になることもある）
3)		定義済み処理（サブルーチン）：定義済み処理名を記述し、その位置にサブルーチンを呼び出すことを明示する。呼び出すサブルーチンは、開始記号にサブルーチン名が記述されていなければならない（＝定義済みであること）
4)		判断記号：条件判断を表す記号。条件式を書きその真偽で処理を分岐させるときに使う。また、真（偽）の場合のみ処理を行うなどの条件分岐もきじゅつできる。条件式は、$a > 10$? や $a = b$? など真か偽かを表す式を書く（C++ で言う bool 型で表せるもの
5)		繰り返し記号（入口）：繰り返しの入り口を書く、前判断型の場合、繰り返しの名前（ループ名）と、「終了条件」を記述する。また、回数ループの時は、カウンタの初期値増分終了値を記述する。次の出口と必ず対になって現れる。
6)		繰り返し記号（出口）：繰り返しの出口を書く、入口と同じループ名を書き、どこからどこまでがループなのかを明確に表す。また、あと判断型ループの場合は、こちらに終了条件を書く
7)		線（コネクタ）：部品と部品をつなぐ線。基本的に処理の流れ通りに線をつないでいく。交差や合流はルールがあるのでそれに従う（後述）。また部品により、線をつなぐ位置は決まっているので、変なところから線が出ていることはない。

1.4.1　アルゴリズムの記述法 –フローチャートのサンプル–

以下に、フローチャートの例を示す。開始から終了までの線と記号をたどっていくことで、すべての記号の意味やルールがわからずとも直感的に処理の流れがわかると思う。

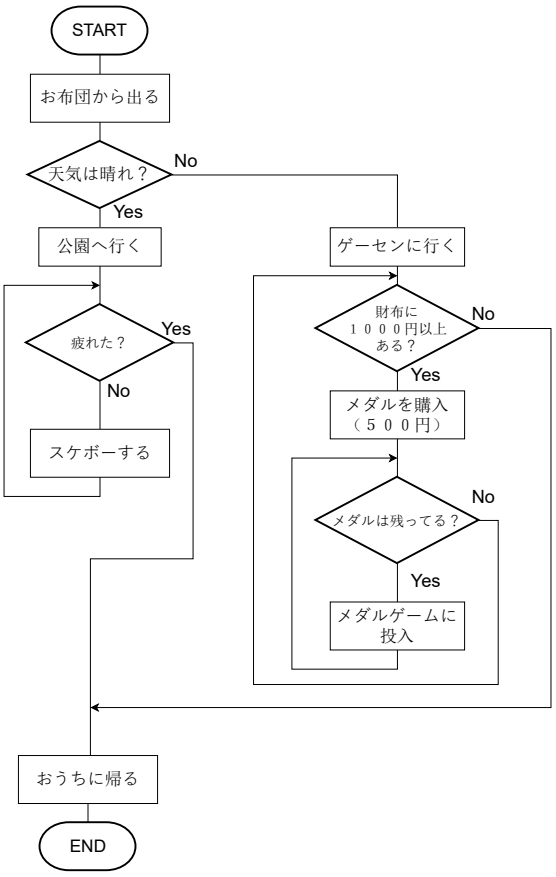


図 1: フローチャートの例

緊急 Mission！

フローチャート例の処理の流れを指でたどってすべてのパターンを確認してみよう！

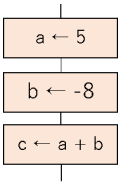
1.4.2　アルゴリズムの記述法 –フローチャート–

コンピュータで行われる処理は、プログラミング古来から大きく３つの処理構造の組み合わせでできているといわれている。（近年のプログラム、プログラミング言語の多様化による複雑な処理はこの限りではないこともある）

３つの基本構造

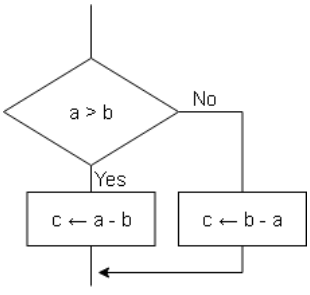
1. **順次構造**
2. **選択構造 (判断)**
3. **反復構造 (繰り返し)**

１．順次構造

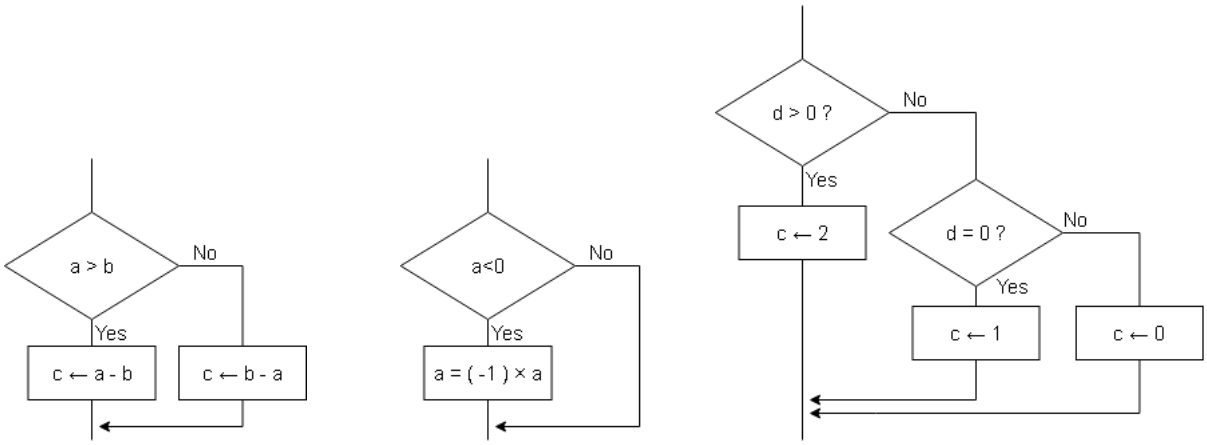


順次構造は、一番単純な処理の形で、ひたすらただ単に上から下に単純な処理を連ねたものである。順次構造では、例のように、コンピュータにさせたい仕事を、やらせたい順番に単純に上から下に記述していく。基本的に一つの記号には、一つの処理を書くのが望ましいが、紙面？節約のために関連性の高い連続したいくつかの処理を一つの記号にまとめて書くこともある。一般的なプログラミング言語で記述できるコンピュータの処理の基本になる構造である。３つの基本構造からなるほぼすべての処理は、この順次構造に帰着させることができる（はずである）。

２．選択構造 (判断)



選択構造は、条件分岐を実現する制御構造である。もしＡが成立するならばＢを実行、Ａが成立しない場合はＣを実行、のような、条件により２つの処理に分岐する処理を書くことができる。条件が成立するときだけ処理を実行するパターンと、条件の成立不成立により、２つの処理に分岐するパターンの２パターンがある。また、条件をいくつも重ねることにより（入れ子）複数の条件下での分岐処理を書くことができる。



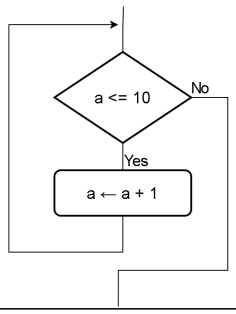
(a) 2 分岐の選択構造　**(b)** 真の場合のみ処理する選択構造　**(c)** 選択構造の入れ子構造

図 2: 選択構造のバリエーション

練習問題

1. 試験の点数を表す整数型の変数 x があり、０～１００の得点の値が格納されているとする。その時に x が６０点以上なら”合格”、６０点未満なら”不合格”と表示する処理の流れ図を書きなさい
2. 変数 x には整数の値が格納されている（初期化または代入されている、どんな値化はわからない)。x が奇数化偶数かを判断し、奇数化偶数かを表示する処理のフローチャートを書きなさい。
3. うるう年は、「西暦年数が 100 の倍数のときは 400 で割れる年、100 の倍数でないときは 4 で割り切れる年を閏年とし、その他を平年とする。」と決められてる。ある整数変数 x が西暦を表すとき、西暦 x 年はうるう年か平年か判定し、表示する処理のフローチャートを書きなさい。

３．反復構造 (繰り返し)

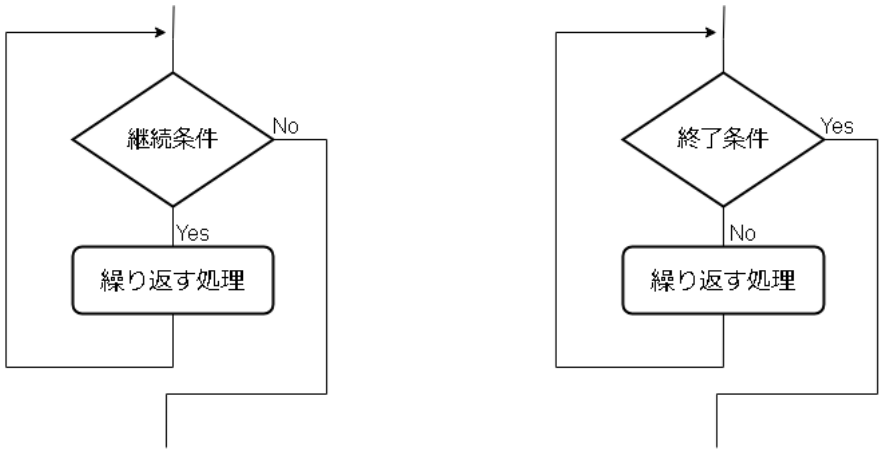


反復構造（繰り返し）では、条件が成立している間（または、条件が成立するまで）処理を繰り返す制御構造である。３つの制御構造の中では、一番複雑な構造となっている。条件の付け方により、**継続条件**と**終了条件**、**前判定**と**後判定**がある （後述)。図では、一番単純な繰り返しの例を示す。

また、さらに繰り返し条件の判定位置により、前判定型と後判定型に分かれる。前判定型では、繰り返し条件の判断がされる前に、判定に使われる値が確定されている必要がある。前判定型では、繰り返す内容が実行される前に判定が行われるため繰り返す内容が一度も実行されずにループを抜ける場合がある。一方で、後判定型では、必ず１度は処理内容の部分を通過し、その後判定が行われる。

継続条件と終了条件

以下に、継続条件と終了条件のフローチャート記述例を示す

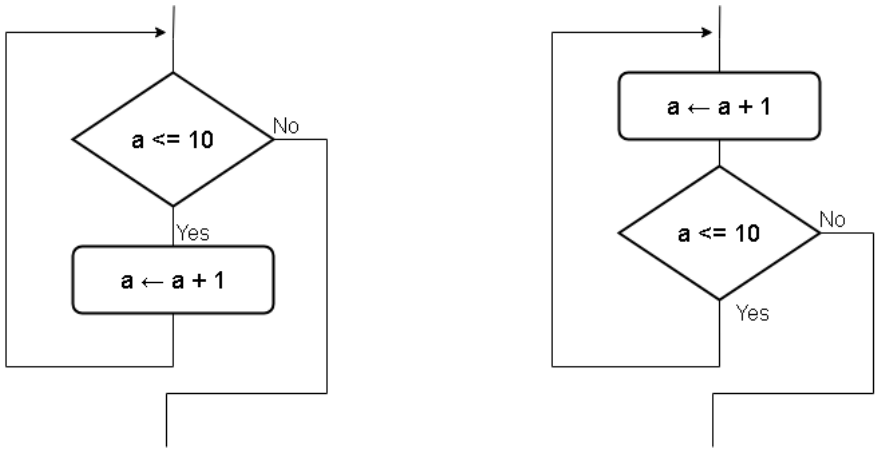


(a) 継続条件の繰り返し　　**(b)** 終了条件の繰り返し

図 3: 繰り返し-継続条件と終了条件-

前判定と後判定

さらに、前判定と後判定の例を示す。



(a) 前判定の繰り返し　　**(b)** 後判定の繰り返し

図 4: 繰り返し-前判定と後判定-

練習問題

以下の処理のフローチャートを書け。

1. 繰り返しを使って”大事なことだから何回も言います。”と 3 回表示する処理
2. 整数変数 a と変数 sum がある。a に 0 が入力されるまで繰り返しキーボードから値を読み込み、sum を使って入力された整数の合計を求める。キーボードから 0 が入力されたら、繰り返し処理を抜け合計値 sum を表示する。
3. 前問２．の処理に、入力された整数が負の数だったら、加算せず正の整数だけ加算する処理を加えなさい。

カウンタ変数

繰り返しでよく使われるテクニックに、**カウンタ変数** (又は単にカウンタ) がある。繰り返し構造を使うときには、しばしば繰り返す回数が決まっていて、その回数分処理を繰り返したいということがある。そのような、回数が決まった繰り返しには、カウンタ変数を使うと便利である。カウンタとは、繰り返し回数を数える専用の変数のことであり、ひたすら繰り返しの回数を数えることに使われる変数のことである。以下の図のフローチャートは、カウンタ変数を使って０～９まで数える処理である。図のフローチャートの変数 i がカウンタ変数である。余談だが、カウンタ変数にはよく i が使われる。*iteration*（**反復**）の頭文字から来ているという説もあるが、本当かどうかは怪しい。カウンタが複数必要な場合は、$i, j, k....$ と次のアルファベットが登場することが多い。

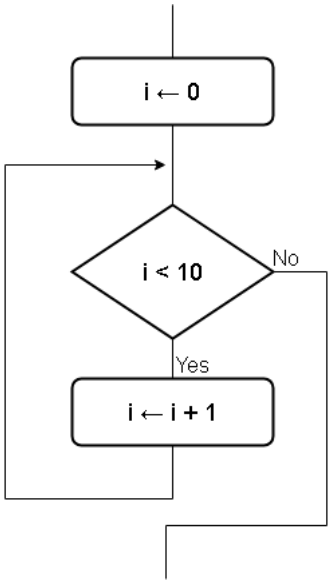


図 5: カウンタ変数の使い方

練習問題

以下に示す処理のフローチャートをカウンタ変数を使って書け。型、変数名などは、適宜適切なものを自由に考えて使って良い。

1. １～９の整数を入力して、その段の九九を表示する処理。
   例：３を入力→ 3, 6, 9,12,15,18,21,24,27
2. １～１０までの整数が奇数か偶数かを判別し表示する。
   例：1 →奇数、2 →偶数、3 →奇数....
3. ０～５０までの奇数の合計を求め表示する。

繰り返しの入れ子構造

繰り返し構造は、選択構造と同様に、繰り返し構造を繰り返すと言った**入れ子構造**にすることが可能である。簡単に言うと**ループのループ**である。このような繰り返しを繰り返すような構造を、**多重ループ**と呼ぶ。（２重になったループを２重ループ、３重なら３重ループと呼ぶ）

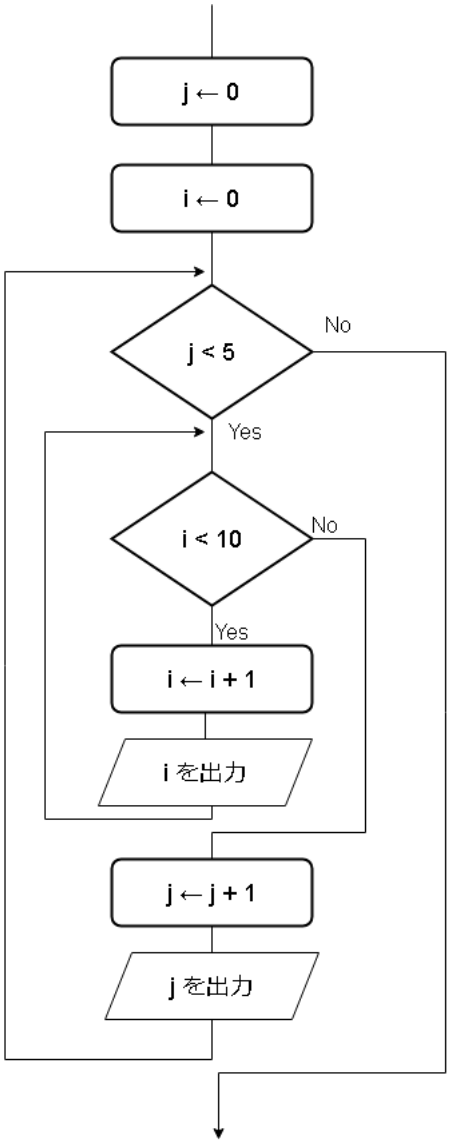


図 6: 多重ループの例：２重ループ

練習問題

以下の処理のフローチャートを考え書いてみんさい。

1. ２重ループとカウンタ変数を使って以下のようなチェック模様を表示する処理を考えなさい。

■□■□■□■□
□■□■□■□■
■□■□■□■□
□■□■□■□■
■□■□■□■□
□■□■□■□■

2. 多重ループの繰り返しとカウンタ変数を使って、以下のような九九表を表示する処理を考えなさい。

```
 1  2  3  4  5  6  7  8  9
 2  4  6  8 10 12 14 16 18
 3  6  9 12 15 18 21 24 27
 4  8 12 16 20 24 28 32 36
 5 10 15 20 25 30 35 40 45
 6 12 18 24 30 36 42 48 54
 7 14 21 28 35 42 49 56 63
 8 16 24 32 40 48 56 64 72
 9 18 27 36 45 54 63 72 81
```

3. 繰り返しとカウンタ変数を使って以下のような模様を表示する処理を考えなさい。

□□□□□□■
□□□□□■■
□□□□■■■
□□□■■■■
□□■■■■■
□■■■■■■
■■■■■■■