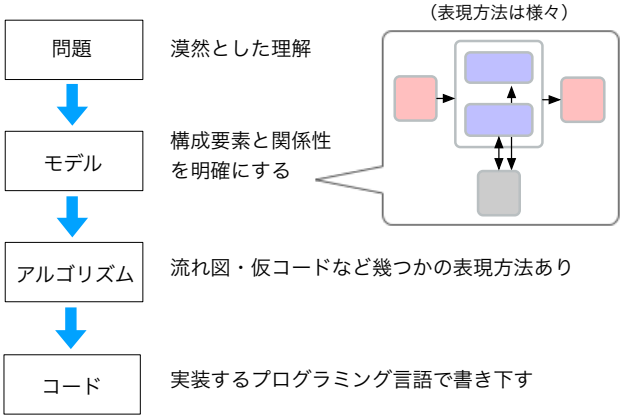
基礎プログラミング演習 II 教材（フローチャート）

■ プログラミングの考え方



プログラミングとはただコードを書くことではありません。
1. まず処理しようとする問題を決め、
2. これを明確なモデルとして理解し、
3. その処理を実現するアルゴリズムを得て、
4. それを特定のプログラミング言語で書き下す

この一連の作業を「プログラミング」と呼びます。

□ アルゴリズムの表現方法

特に重要なのはアルゴリズム（問題の解を確実に得られる手順）を明確にする段階です。明確にするには書き出す（頭の中を外在化する）ことが重要ですが、そのための方法は幾つかあります。
ここでは以下の三つの方法を紹介します。

・自然言語：「あれをこうしてこうすれば良い」などと記述する。複雑なアルゴリズムの表現には不適。
・流れ図（フローチャート）：処理の流れを小さなブロックを線で接続して示す。
・仮コード（擬似コード）：プログラミング言語による書き方に似せて、ただし文法上の制約に縛られず表現する。

以下に、「1 から 10 までの数を足した合計を求める」ためのアルゴリズムを、それぞれの方法で表現します。

■ 自然言語によるアルゴリズムの表現

例えば以下のようになるでしょう。
1. 合計をまず 0 にしておく
2. カウンターを 1 にする
3. カウンターの値を合計に加える
4. カウンターに 1 を加える
5. 処理 4. の結果、カウンターが 10 以下なら処理 3. から繰り返す
6. （そうでなければ）合計の値を出力する

合計	カウンター

□ 課題 1

この「作業手順の指示」を読んで、その通りあなたが自分の手で（紙と鉛筆で）作業すれば、正しく結果が得られる事を確かめて下さい。

処理のトレースは、手順を追いながら変化する変数の値を記録することで行います。
変数となるはずの合計とカウンターの値を書き込むための枠を左図に用意したので、これを使うと良いでしょう。

・ 手順を 1. から順にたどり、変数に書き込み（変化）があれば、それを反映させます。
（右は手順 1. 2. 3. 4. まで実行した段階の状態）

合計	カウンター
0	1
1	

・ 最終的に処理は 6. に到達することが確認できるでしょう。
（右は手順 6. つまり最終段階に到達した状態）

36	
45	10
55	11

恐らく多くの人は、処理が終了するまでのすべての段階を書き出さなくても、何回か繰り返し処理をトレースした時点で、カウンターが 10 になるところまでは正しく動きを推定できる（確信が持てる）と思います。

■ 流れ図（フローチャート）

流れ図では処理の流れを小さなブロックを線で接続してアルゴリズムを表現します。以下に、簡単に表記の例を示します。まず処理は一つの四角形で表現し、それらを線で結ぶことで処理の流れを示します。

基本的に処理は上から下に線に沿って進みますが、分かりやすさのために矢印を付けることもあります。(Fig. 1)

条件判定と、それに従った分岐は菱形と分岐先へつながる線で表現します。(Fig. 2)

ループ（繰り返し処理）は、元に戻る矢印で表現します。(Fig. 3)

条件分岐などはいろいろな場合があります。あまり定型を気にせず柔軟に表現すれば良いでしょう。(Fig. 4)

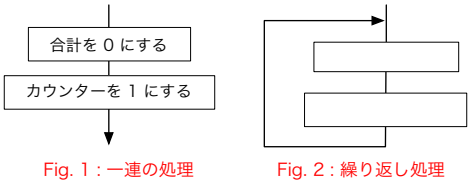


Fig. 1 : 一連の処理

Fig. 2 : 繰り返し処理

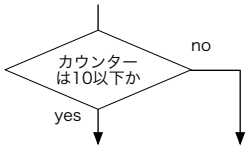
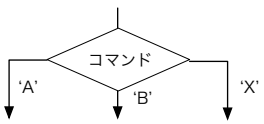
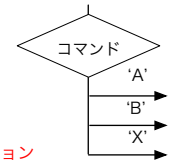


Fig. 3 : 条件分岐





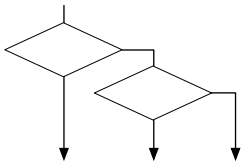


Fig. 4 : 条件分岐のバリエーション

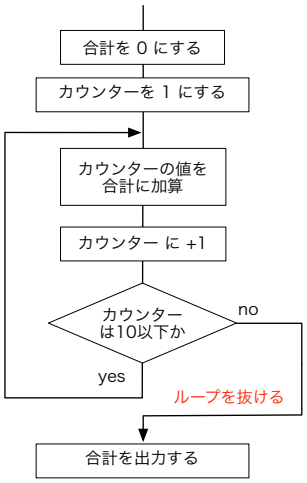
流れ図は標準化（JIS X0121）された記法がありますが、それに厳密に従う要求がされることはまずありませんので、ここでは簡単にやります。

□ 課題 2

1 から 10 までの合計を求める処理を、先に示した自然言語によるアルゴリズム表現の 1. から 6. に沿ってフローチャート化したものを右の図に示します。

このフローチャートの指示どおり、あなたが自分の手で（紙と鉛筆で）作業すれば、正しく結果が得られる事を確かめて下さい。
（どこかノートに、前のページ同様、合計とカウンターの値を書き込んで実際に作業するように）

また、自然言語によるアルゴリズム記述と対照して、両者が等価である事を確認してください。



1. 合計をまず 0 にしておく
2. カウンターを 1 にする
3. カウンターの値を合計に加える
4. カウンターに 1 を加える
5. 処理 4. の結果、カウンターが 10 以下なら処理 3. から繰り返す
6. （そうでなければ）合計の値を出力

□ 課題 3

入力された二つの数の差の絶対値を出力するための処理の流れ(アルゴリズム)をフローチャートで描いて下さい。パッと描けない場合は、まず自然言語で手順を書き下し、それを図化するつもりでやると良いです。

□ 課題 4

二つの自然数 a, b (a > b) の最大公約数を求めるアルゴリズム「ユークリッドの互除法」を右に示します。
これをフローチャートで描いて、正しく結果が得られる事を確かめて下さい。

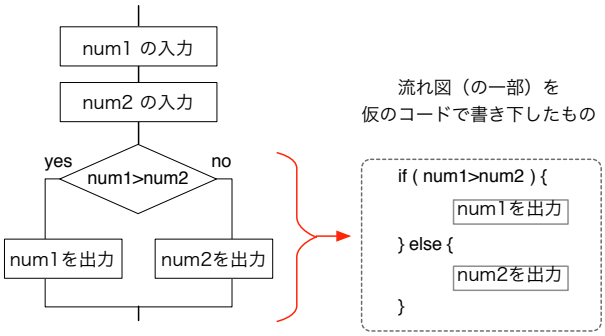
1. a, b を入力
2. b がゼロなら a を出力する
3. （そうでなければ）a を b で割った余りを調べる
4. b を新たな a とする
5. 余りを新たな b とする
6. 処理 2. から繰り返し

例：
a=84, b=60
・84 / 60 の余りは 24
・60 / 24 の余りは 12
・24 / 12 の余りは 0
よって解は 12

■ 仮コード（擬似コード）

C 言語に限らず、プログラミング言語は一言一句間違いのない記述が求められます。しかしアルゴリズムを検討する段階では、そうした各プログラミング言語の細かな文法制約などなしに、ラフな記述でアルゴリズムや処理の流れを記述する方が有効です。
そのようにして書かれた、コード（のようなもの）を、仮のコード（擬似コード, pseudo code）などと呼びます。

右に入力された二つの数のうち、大きい方を出力するための流れ図と、それに対応する仮コードを示します。
同じアルゴリズムが、異なる表現方法で書かれている事が分かるでしょう。

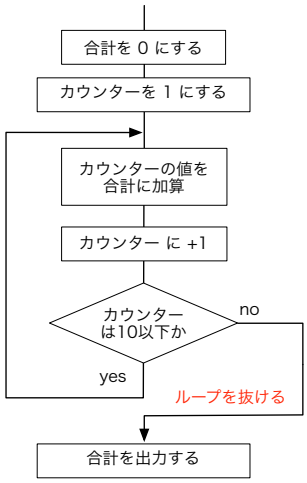


□ 課題 5

1 から 10 までの合計を求める処理のフローチャートを見ながら、それを仮コードで書き直して下さい。

ループ処理は C 言語の while 文に似せて書いても良いですし、loop: といった「オレオレ」文法を作って貰っても構いません。

また、その仮コードに書かれている通りにあなたが自分の手で（紙と鉛筆で）作業すれば、正しく結果が得られる事を確かめて下さい。



1. 合計をまず 0 にしておく
2. カウンターを 1 にする
3. カウンターの値を合計に加える
4. カウンターに 1 を加える
5. 処理 4. の結果、カウンターが 10 以下なら処理 3. から繰り返す
6. （そうでなければ）合計の値を出力