4 プログラミング序論

4–1 アルゴリズムとは何か?

コンピュータは与えられた「アルゴリズム」に従って動く。

アルゴリズムとは?

(説明1) **a process or rules for calculating something, especially by machine** ··· Oxford Paperback Dictionary

(説明2) 問題を解くための計算法(一般的な意味) ··· 岩波情報科学辞典

⇨ 「...を計算する方法」といった類のものがアルゴリズムに相当

⇨ 例えば小中学校で習った
- 三角形の面積を計算する方法,
- 二次方程式の解を求める方法

例4. 1 (二次方程式の解を求めるアルゴリズム)

二次方程式 $ax^2+bx+c=0$ $(a\neq 0)$ の解は... $x = \dfrac{-b\pm\sqrt{b^2-4ac}}{2a}$

4–2 アルゴリズムの記述

どんな風にアルゴリズムを記述すべきか:

- コンピュータは、我々人間が与えた
  最初に何をして,次に何をして,... という動作指示の列
  を逐次的に実行しているだけである。

⇨ コンピュータに思い通りの計算を行わせたければ、
その計算 (処理) の手順を明示的に記述する必要がある。

補足：
処理の手順を明示的に記述するというのは、プログラミングに限ったことではない。例えば、「ハウス バーモントカレー」の箱の裏にはカレーの作り方が次の様に示されている。

① 厚手の鍋にサラダ油を熱し、一口大に切った肉、野菜をよくいためます。
② 水を加え、沸騰したらあくを取り、材料が柔らかくなるまで弱火~ 中火で煮込みます。
③ いったん火を止め、ルウを割り入れて溶かし、再び弱火でとろみがつくまで煮込みます。

どんな風にアルゴリズムを記述すべきか:

- コンピュータは、我々人間が与えた
  最初に何をして,次に何をして,... という動作指示の列
  を逐次的に実行しているだけである。

  ⇨ コンピュータに思い通りの計算を行わせたければ、
  その計算 (処理) の手順を明示的に記述する必要がある。

- コンピュータはその内部に格納された所在のはっきりしたデータだけを計算 (処理) に使うことが出来る。

  ⇨ コンピュータの基本動作を指示する際は、
  その動作で扱うデータの所在を明らかにする必要がある。

例4. 2 (二次方程式を解くアルゴリズム, コンピュータ向きの記述)

二次方程式を解くアルゴリズム (a)

(step1) 二次方程式の3つの係数の値を (キーボードから) 受け取り、それらのデータを各々 a, b, c という名前の付いた場所に格納する。
(以後、aに格納された値は 0 でないと仮定して計算を進める。)

(step2) $\dfrac{-(\boxed{b}\text{内の値})+\sqrt{(\boxed{b}\text{内の値})^2-4(\boxed{a}\text{内の値})(\boxed{c}\text{内の値})}}{2(\boxed{a}\text{内の値})}$ の計算をしてその結果を x1 という名前の付いた場所に格納する。

(step3) $\dfrac{-(\boxed{b}\text{内の値})-\sqrt{(\boxed{b}\text{内の値})^2-4(\boxed{a}\text{内の値})(\boxed{c}\text{内の値})}}{2(\boxed{a}\text{内の値})}$ の計算をしてその結果を x2 という名前の付いた場所に格納する。

(step4) (x1内の値), (x2内の値) をコンピュータの画面に表示する。

あるいは、略記的に

二次方程式を解くアルゴリズム(a')

(step1) 二次方程式の3つの係数の値を入力して、各々 **a, b, c** という場所に格納する。　　(以後、**a≠0** と仮定して計算を進める。)

(step2) $\mathbf{x1} \longleftarrow \dfrac{-\mathbf{b}+\sqrt{\mathbf{b^2}-\mathbf{4ac}}}{\mathbf{2a}}$

(step3) $\mathbf{x2} \longleftarrow \dfrac{-\mathbf{b}-\sqrt{\mathbf{b^2}-\mathbf{4ac}}}{\mathbf{2a}}$

(step4) **x1, x2** の値を出力

コンピュータが虚数を基本データとして取り扱うことが出来ない場合

二次方程式を解くアルゴリズム(b)

(step1) 二次方程式の3つの係数の値を入力して、各々 **a, b, c** という場所に格納する。　　(以後、**a≠0** と仮定して計算を進める。)

(step2) **D⟵ b^2-4ac**

(step3) **D≥0** かどうかによって、次の **(3.1), (3.2)** のいずれかを実行

(3.1) **D≥0** なら、

① **x1** $\longleftarrow \dfrac{-b+\sqrt{D}}{2a}$　　② **x2** $\longleftarrow \dfrac{-b-\sqrt{D}}{2a}$

③ “実根を持つ” という表示を添えて、**x1, x2** の値を出力

(3.2) **D<0** なら、

① **re** $\longleftarrow \dfrac{-b}{2a}$　　② **im** $\longleftarrow \dfrac{\sqrt{-D}}{2a}$

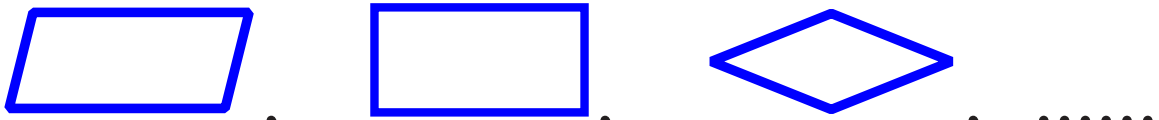
③ “虚根を持つ” という表示を添えて、
実部 **re** と虚部 **im** の値を出力

流れ図:

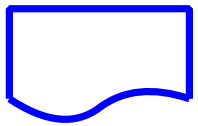
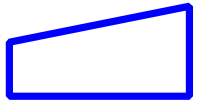
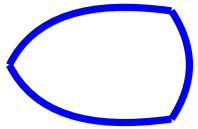
アルゴリズムを視覚的に分かり易く表示するために、
流れ図 (**flowchart**) が用いられている。

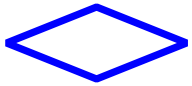
(例) $\Longrightarrow$ **p.9** の中程

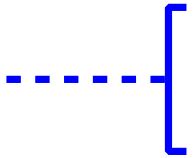
(構成)　処理内容の書き込まれた



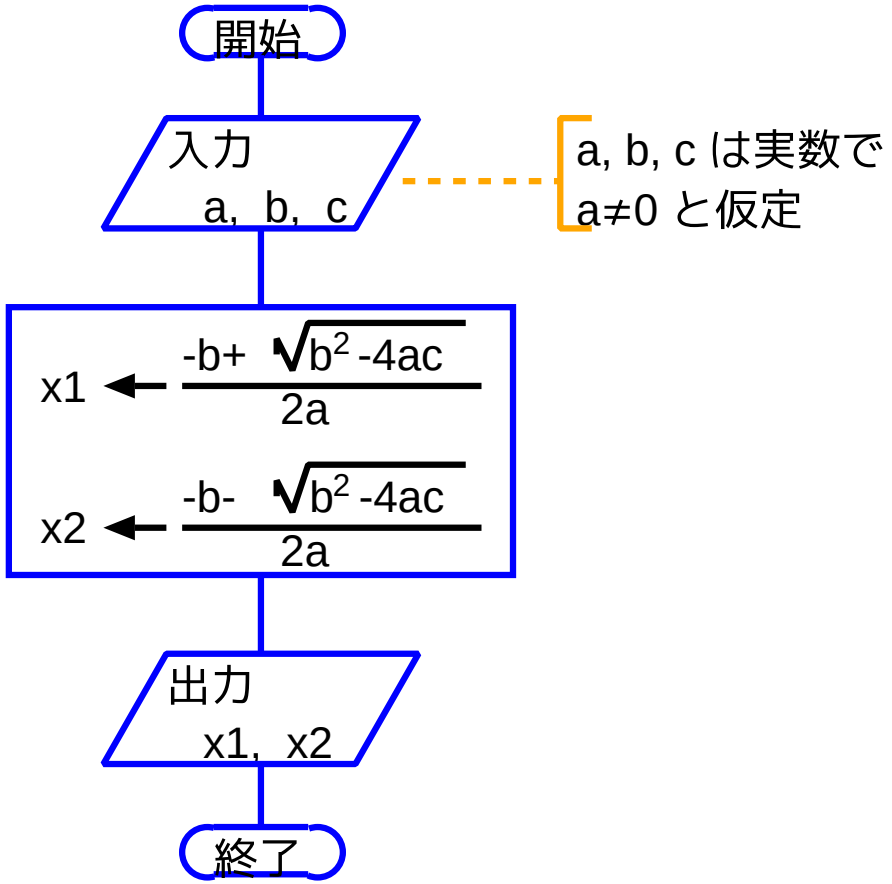
といった形の記号を繋げて構成される。
各々の記号の意味は次の通りである。

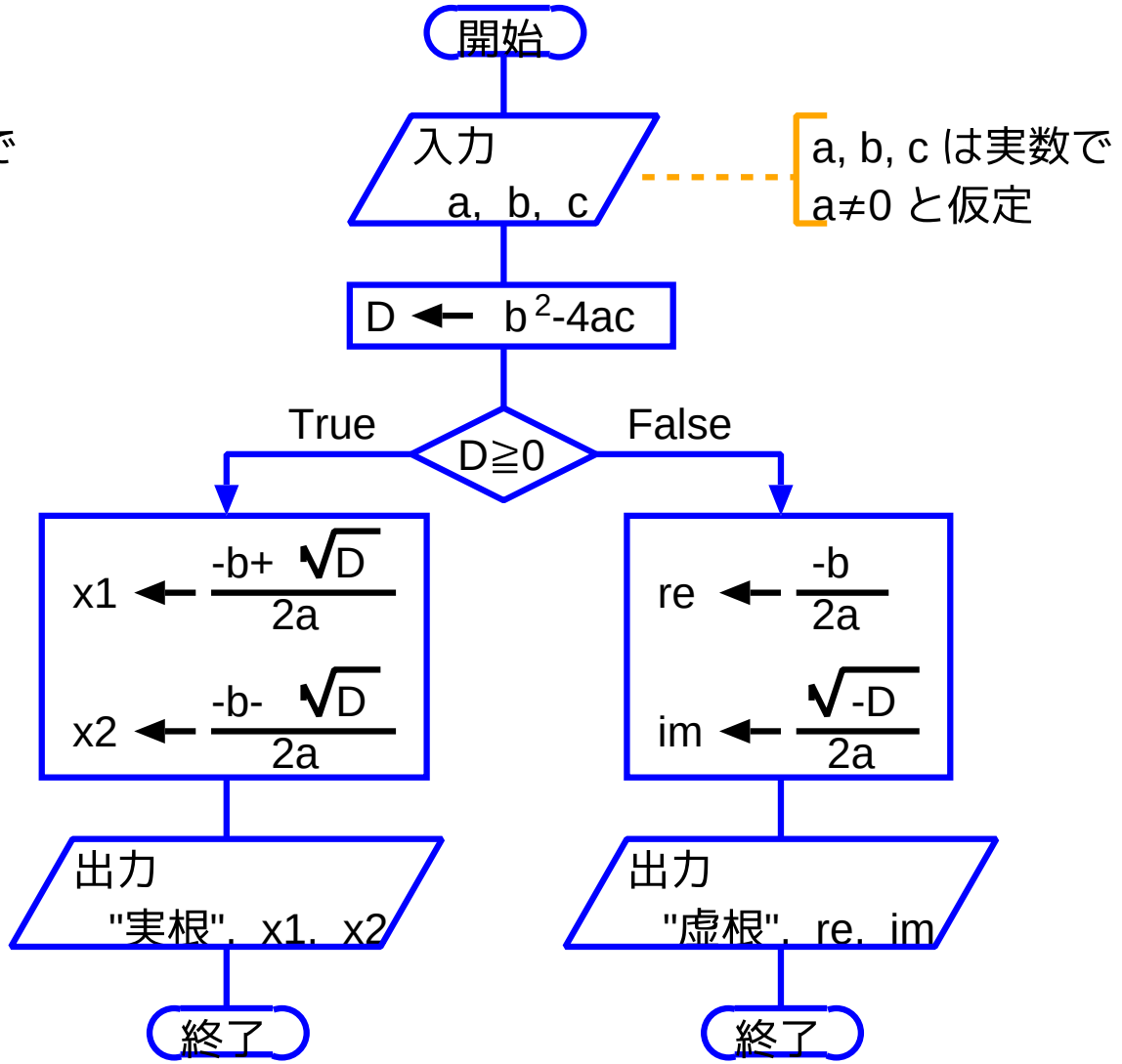
記号の分類		記号	名称	説明
データ記号	基本 データ記号		データ	媒体を指定しないデータ (の入出力) を表す。
	個別 データ記号		書類	人間の読める媒体上のデータ (の入出力) を表す。媒体の例としてはプリンタの出力。
			手操作入力	キーボードなどによる手操作入力、またはこれらの入力によるデータを表す。
			表示	人が利用する情報を表示する媒体 (e.g. ディスプレイ画面) 上のデータ、またはそれらのデータの出力を表す。

記号の分類		記号	名称	説明
処理記号	基本 処理記号		処理	任意の種類の処理機能を表す。
	個別 処理記号		定義済み処理	サブルーチンやモジュールなど、別の場所で定義された１つ以上の演算または命令群からなる処理を表す。
			判断	１つの入口と幾つかの択一的な出口を持ち、記号中に定義された条件の評価に従って唯一の出口を選ぶ、判断機能またはスイッチ形の機能を表す。
線記号	基本 線記号		線	データまたは制御の流れを表す。標準的な流れの方向は左から右，上から下であって、それ以外の時、または見易さを強調する時は矢先を付ける。

記号の分類	記号	名称	説明
		結合子	同じ流れ図中の他の部分への出口または他の部分からの入口を表したり、線を中断し他の場所に続けたりするのに用いる。 対応する結合子は同一で一意の名前を含まなければならない。
特殊記号		端子	外部環境への出口、または外部環境からの入口を表す。
		注釈	明確にするために説明または注を付加するのに用いる。注釈記号の破線は関連する記号に付けるか、または記号群を囲んでも良い。 説明または注は範囲を示す記号の近くに書く。

例4. 3 (二次方程式を解くアルゴリズムを流れ図で)

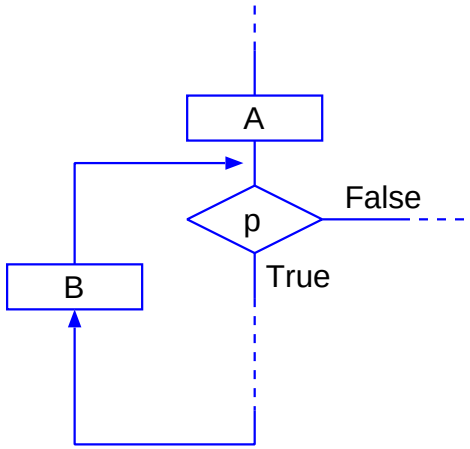




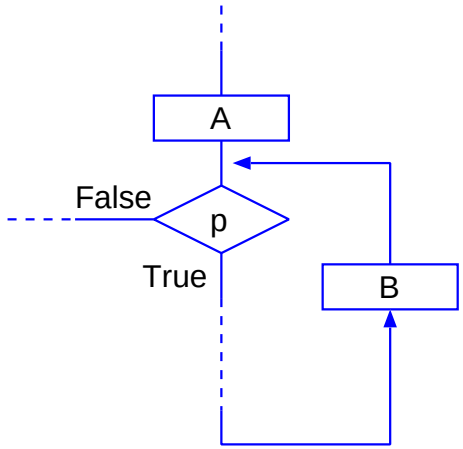
流れ図の拡張:

流れ図では、処理の繰り返しを表す手段が用意されていない。

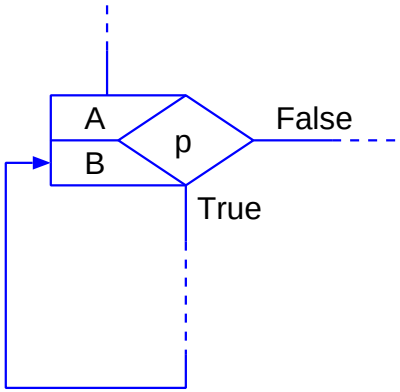
⇨ この欠点を補うために拡張表記を導入することもある。例えば、



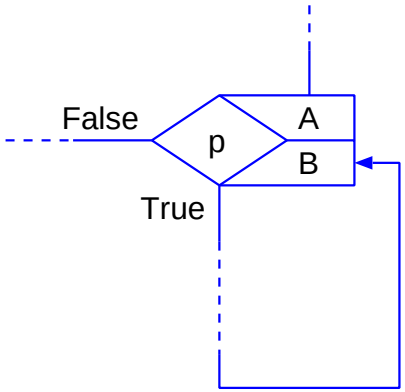
,



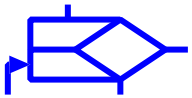
の処理を表すのにそれぞれ次の様な略記法を導入したりする。



,



,

この  という形の箱を繰り返しの箱と呼ぶ。

流れ図に代わる表現形式:

流れ図は次の様な欠点を持つ。

- コンピュータへの動作指示(プログラム)は一次元的な構造をしている。
  これに対して
  流れ図は二次元的な処理の流れを記述することができ、
  処理の流れが複雑になる可能性がある。

  $\Longrightarrow$ 流れ図で記述されたアルゴリズムをプログラムの形に直しても、
  見易いものができるとは限らない。

- 流れ図では処理の繰り返しを表す手段が特別に用意されていない。

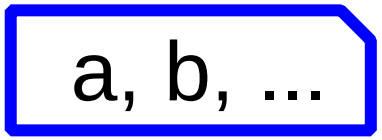
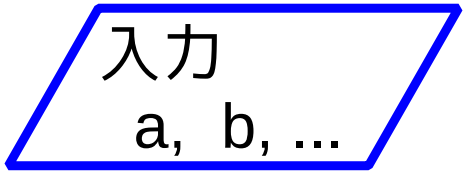
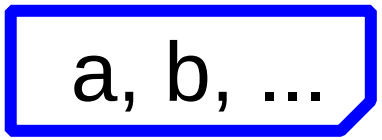
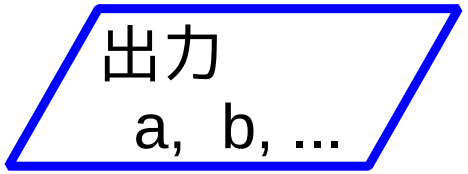
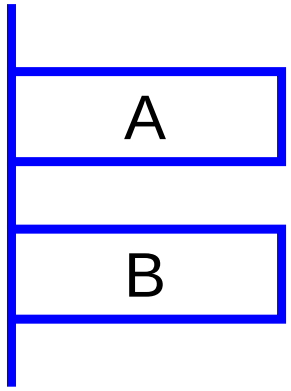
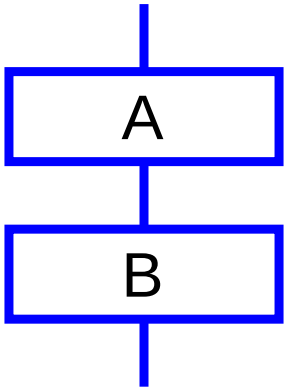
⇨ 流れ図に代わる表現形式が色々と提案された。例えば、

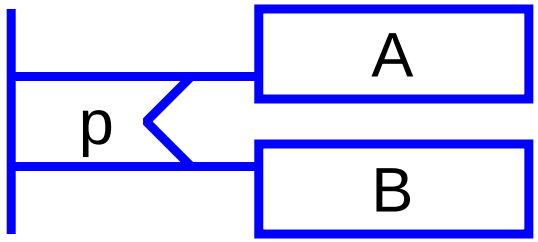
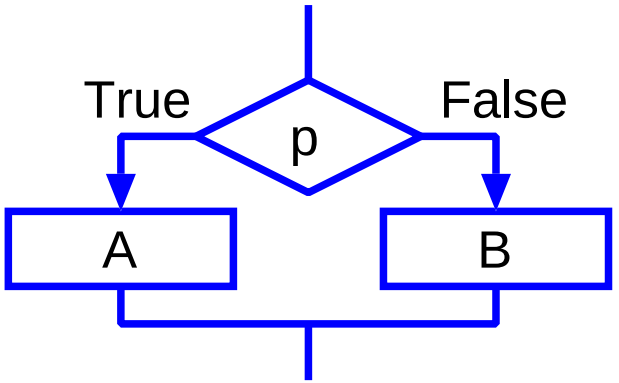
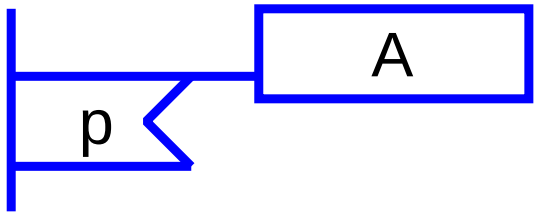
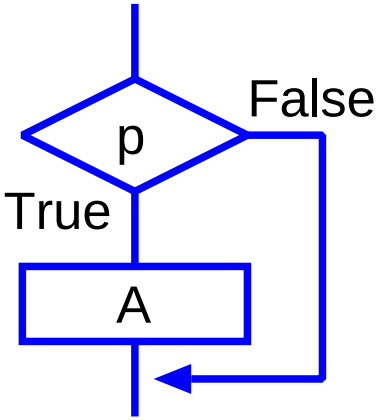
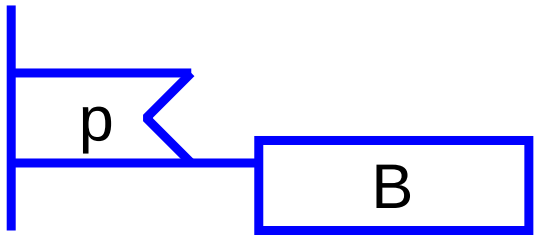
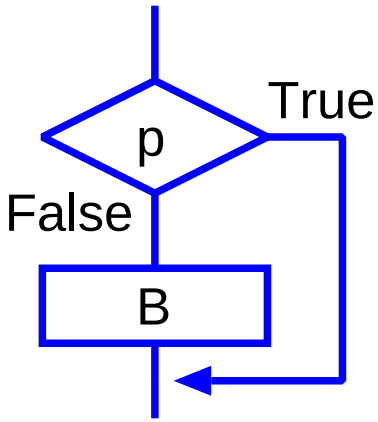
PAD(problem analysis diagram, 日立),
NSチャート(Nassi-Schneiderman chart),
HCP(hierarchical compact chart) など。

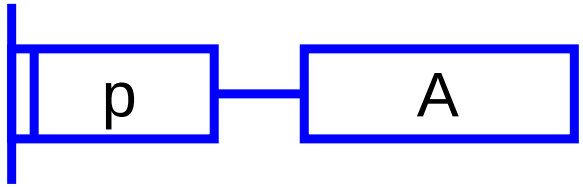
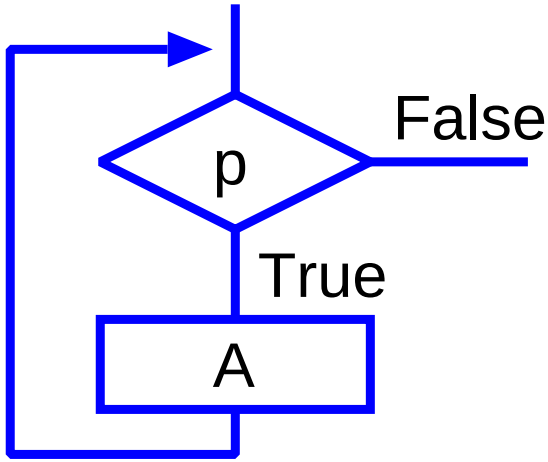
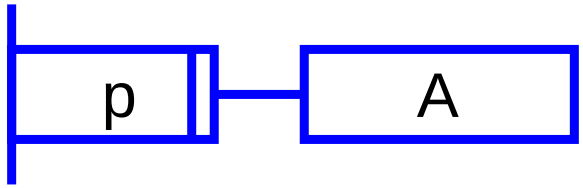
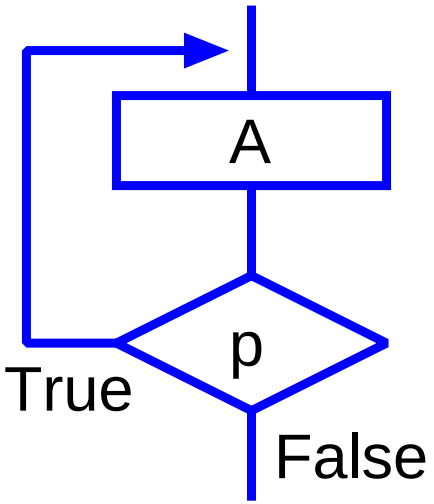
PAD(problem analysis diagram):

PADにおいては、処理の合成の仕方として 連接, 分岐, 繰り返し の3種類だけを用いて全ての処理の流れを表す。

連接, 分岐, 繰り返し、およびデータの入出力を次のように表す。

	PADにおける表し方	同等の流れ図表現
入力		
出力		
連接		

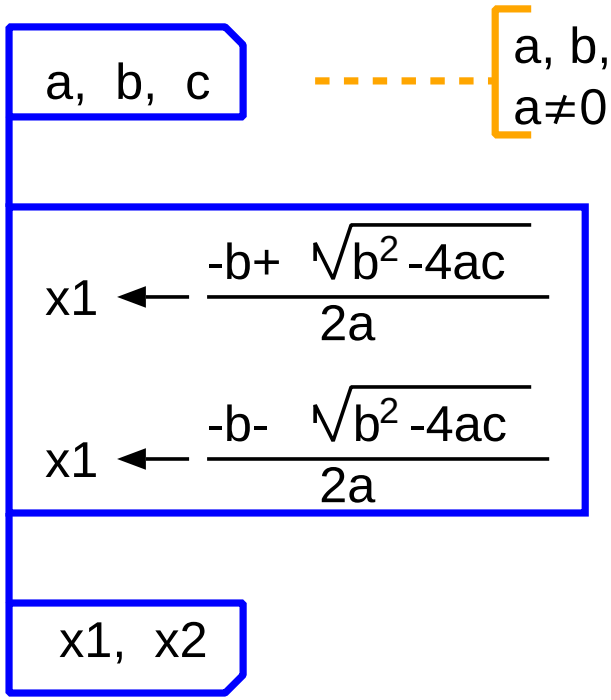
	PADにおける表し方	同等の流れ図表現
分岐		
		
		

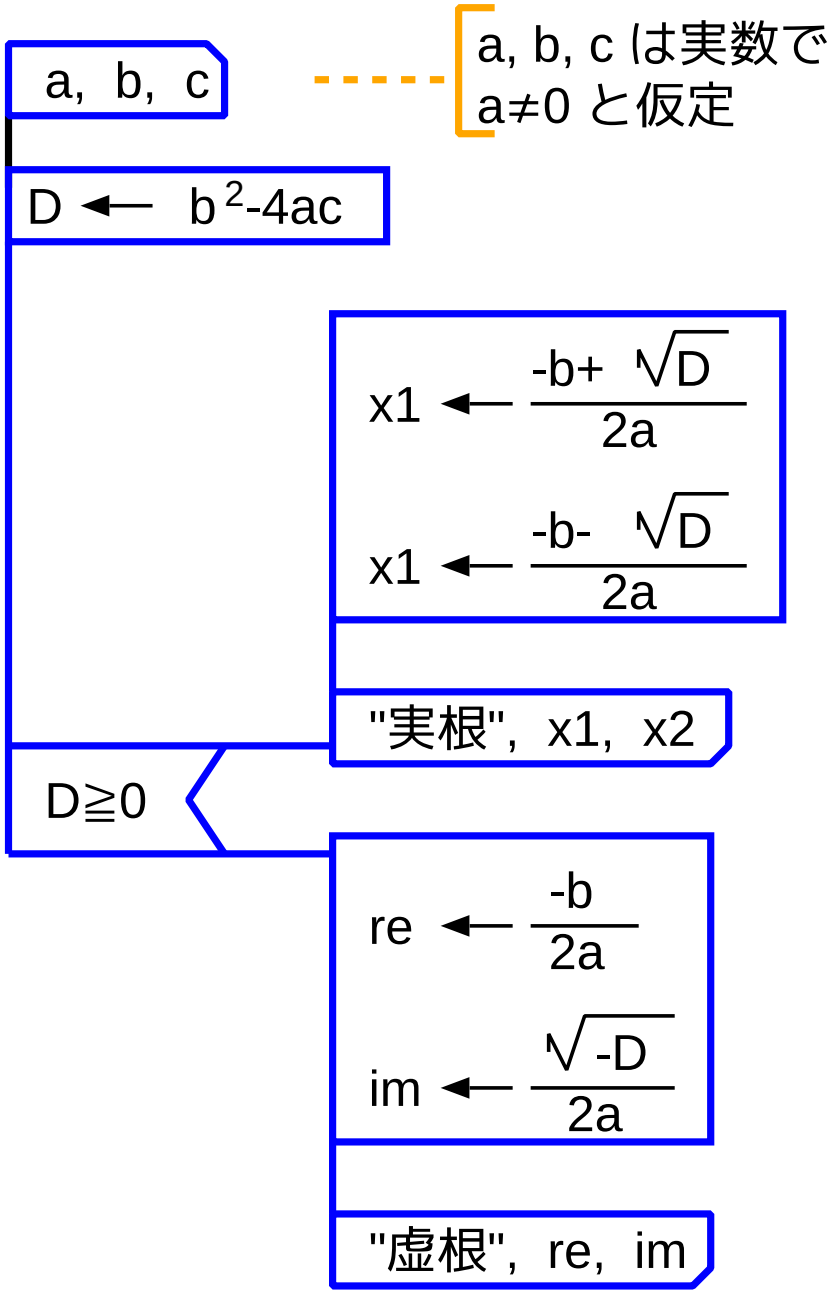
	PADにおける表し方	同等の流れ図表現
繰り返し		
		

3種類の処理合成しか用いない理由：

- 入り組んだ処理の流れは分かりにくい。
- 大抵のアルゴリズムは、これら3つの処理合成を組み合わせることによって表せる。
- 作り上げたアルゴリズムを実際にコンピュータに実行させる際には、アルゴリズムをコンピュータが理解できる「プログラム」の形に書き表さなければならない。その際 **C言語, Pascal, Fortaran** 等の (高級) 手続き型プログラミング言語を使うことがほとんどであるが、これらの言語では 連接, 分岐, 繰り返し という3種の処理合成を組み合わせてアルゴリズムを記述するのが基本となっている。

例4. 4 (二次方程式を解くアルゴリズムをPADで)





4–3 アルゴリズム設計の重要性

アルゴリズムの設計:

コンピュータが理解できる言語でアルゴリズムを細かく記述したものが
プログラム (program) である。当然、

悪い (e.g. 分かりにくい, 手間のかかる) アルゴリズムからは
悪いプログラムしかできない。

⇨ プログラムを作成する上で最も大切なことは、

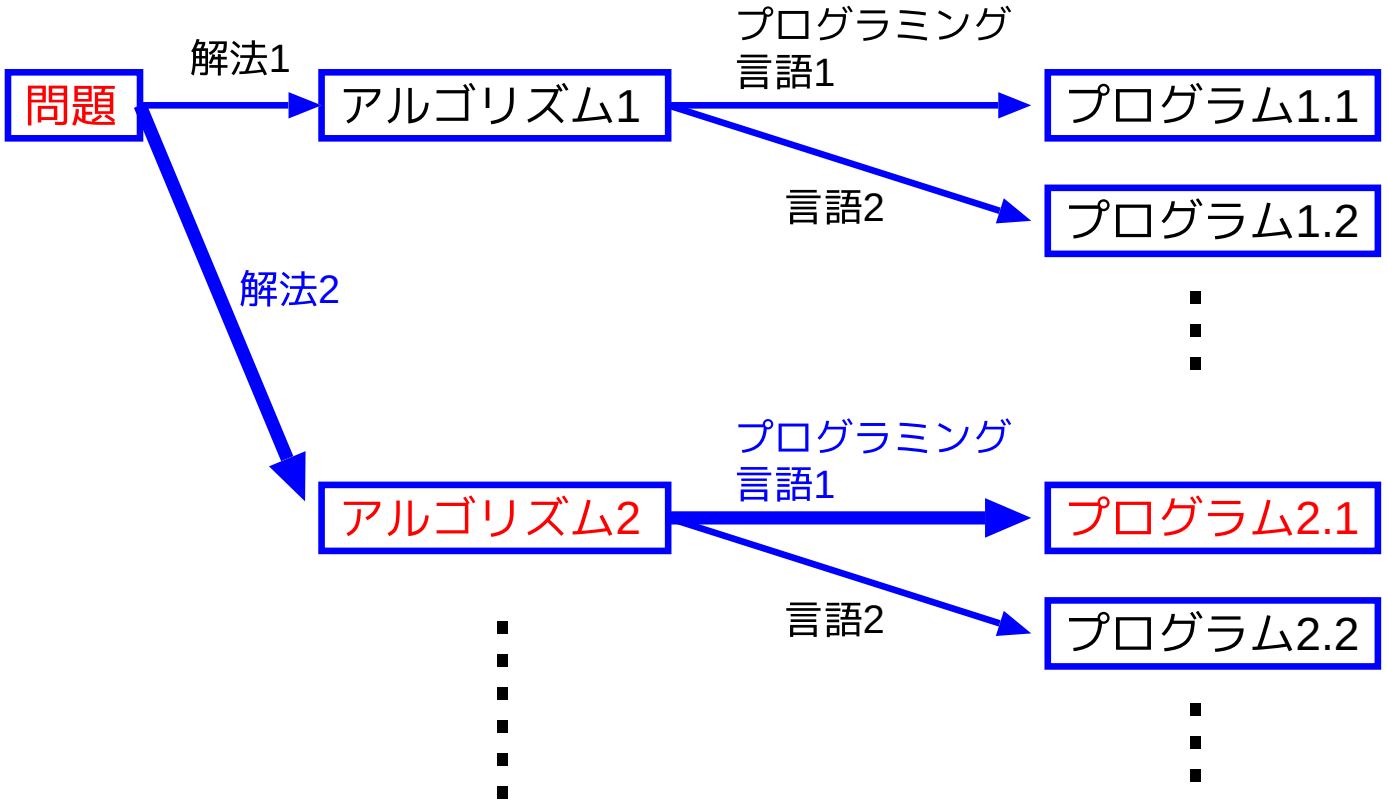
問題を分析して、
良い (e.g. 分かり易い, 速い) アルゴリズムを設計する

ことである。

プログラミング言語の選択:

アルゴリズムをプログラムとして記述するための言語(プログラミング言語)としては、これまで色々な利用目的に対して色々なものが設計され、そのうちの幾つかは現在も利用されている。

⇨ これらの中から問題やアルゴリズムに合ったものを選ぶことも大切である。



補足：

- どのプログラミング言語を選ぶかで、プログラムが単純になったり複雑になったりする。このことから、与えられた問題に対してまず適切なプログラミング言語を選び、次にこの言語に依存した形でアルゴリズムを記述することがある。しかし、
  一般には、
  アルゴリズムはどのプログラミング言語にも依存しない形で記述することが望ましい。

- 実際には、使用するコンピュータ環境で全てのプログラミング言語が使える訳ではないので、言語の選択肢はそれほど多くはない。