プログラミング試験の作問と評価に関する一考察

太田信宏

Ⅰ．はじめに

情報化社会の中で必要とされる人材や要求される技術というのは時代の流れとともに変化している。最近よく耳にする「情報リテラシー」,「エンドユーザコンピューティング」といった言葉からもわかるように，現代社会における情報化の波は急速な拡がりを見せている。

また一方では高度に専門化した，あるいは特化した技術を必要とする人材の需要も高まっており，これまでのようなシステムエンジニアとプログラマという単純な区分けでは対応できなくなっている。

従来のように「情報処理教育」といえばそのほとんどが「プログラミング教育」を指すような時代ではなくなっているものの，特定の分野においては「プログラミング教育」あるいは「プログラミング言語教育」が情報処理教育の中の大きな柱の一つであることもまた事実である。

プログラミングの能力は本人の適性や持って生まれたセンスに依るところが多いとも言われているが，学生を指導する立場から見た場合に，プログラミングに対する学生の理解度あるいは習得すべき水準への達成度などは最も気にかかる部分の１つと言える。

これらを一定の基準で公正に判断するにはいくつかの方法が考えられるが，その１つに「学生に対しプログラミングの能力を問うための試験を実施しそれを評価する」ということが考えられる。当然のことながら正しく評価するためには適正な試験問題が要求される。本論ではこのような点を踏まえ，

・プログラミング能力を問う試験とはどのようにあるべきか
・質の高い試験問題にするためにはどのような要件が必要か
・試験問題の中で教育的な配慮ということをどう考えるか

など「適正で適切なプログラミングの試験問題を作成するためには」という点からの考察を行うこととする。

Ⅱ．試験問題の題材

プログラミングの指導を行う場合，一般的には次の２つアプローチの方法が考えられる。

①プログラム言語にはあまり触れずに（束縛されずに），アルゴリズムの組み立てやモジュールの概念の理解を中心に学習させる方法。

②習得させるべきプログラム言語を意識しながら，言語の特徴を生かした処理技法を中心に，文法要素などを絡めて学習させる方法。

本論ではプログラミング言語としてCOBOL言語を取り上げることとし，上記の②に近い形でプログラミング教育を行った結果として，どのような試験問題が適切であるかという視点から論述することとする。

COBOLは事務処理向けの言語であり，その内容はファイル処理が中心となる。そのた

めC言語などと比べると，「できること」が限られしまい，試験問題で利用できる題材もそう多くはない。一般にCOBOL言語を学ぶ場合に，目標として習得すべきテーマは次のようなものであり，試験問題もこの中から出題されると考えられる。

表Ⅰ．試験問題の題材

テーマ	主 な 内 容
明細表印字処理	明細行への編集，出力行数の制御，改ページと見出しの処理，グループインジケーション　など
グループコントロール	グループごとに合計や平均を求める処理 ・1段階のグループコントロール ・多段階のグループコントロール
表　操　作	1次元，2次元，・・・n次元テーブルの定義とテーブルの基本操作，内部分類，表の探索（線形探索，二分探索），テーブル集計　など
マッチング	1対1のマッチング，1対nのマッチング，ファイルの更新(保守)，ファイルの照合，ファイルの併合　など
データチェック	数字チェック，範囲チェック，順序チェックなどの各種データチェックを含んだ処理
整　列	COBOLの整列機能（SORT文）を含む処理
索引ファイル 相対ファイル	レコードキーによる乱アクセス処理
そ　の　他	プログラム関連絡機能を用いた処理 条件名条件　など

III. 必要なプログラミング能力

プログラミングを学ぶ学生に身につけてほしい能力とは何か，「プログラムが作成できる」というのは何ができることなのか，試験問題の中で問うべき能力についてまとめてみた。

(1)各種処理技法に対する基本的な知識

表1にあげた各テーマについて，その処理技法に対する基本的な知識を身につけている必要がある。具体的には次のようになる。

①明細表印字処理
- ・文字項目／数値項目の編集方法
- ・ラインカウンタやページカウンタの目的と使い方
- ・一定条件のもとでの改行や改ページの制御方法
- ・グループインジケーションの意味と判定方法

②グループコントロール
- ・グループキーの意味とキーブレークの判定およびその時必要な処理方法
- ・グループキーが複数ある場合の処理方法（多重ループの制御）

③表操作
- ・1次元および多次元の表の定義
- ・添字と指標の意味，相違点，使用法
- ・表を使った集計処理
- ・内部分類のアルゴリズム
- ・線形探索，二分探索，SEARCH文

④マッチング
- ・マッチングキーの大小比較の意味とその時必要な処理の方法

・トランザクションキーがユニークな場合と複数件ある場合の処理の方法
・順ファイルに対するレコードの追加，更新，削除の処理

⑤データチェック
・データチェックの目的と各種処理技法（ニューメリックチェック，範囲チェック，順序チェック，チェックデジットなど）

⑥整列
・SORT文の目的と使い方
・USING／GIVINGを使った整列
・RELEASE／RETURNを使った整列

⑦各種ファイル編成に対する処理技法
・順アクセス／乱アクセスの意味と使用法
・レコードキーの取り扱い方法
・索引ファイルの処理技法
・相対ファイルの処理技法

⑧その他
・プログラム間連絡機能（CALL文の使い方，リンケージセクションの意味と目的など）
・条件名条件（レベル番号88の使い方）

(2)正しい言語仕様で表現する能力
・データ構造（集団項目，基本項目，編集項目，再定義，表など）を正しく表現する能力
・ファイルおよびレコードを正しく表現する能力
・手続き部における制御文を正しく表現する能力

(3)処理の手続きを正しく組み立てる能力
・処理条件や与えられた仕様の中から，論理を読みとり，適切なアルゴリズムを組み立てる能力
・繰り返し処理のアルゴリズムを正しく完成させる能力（繰り返し処理の規則性の発見，前判定型および後判定型ループの正しい終了条件の設定など）

(4)プログラムを検証する能力
・できあがったプログラムを読み取り，手続きの処理内容を推測する能力
・テストデータを使ってプログラムをトレースする能力
・プログラムの誤りを発見して，デバッグをする能力

(5)プログラムを設計する能力
・ファイルおよびレコードのデータ構造を設計する能力
・テストデータを処理条件や仕様に照らし合わせて過不足なく設計する能力
・プログラム全体を各機能単位（モジュール）へブレークダウンしていく能力
・モジュール間のインターフェース設計に関する能力
・個々のモジュールに対し機能性，汎用性，見やすさなどを考慮して適切なモジュール設計をする能力

IV. 設問の形式

設問の内容をどのような形式にするかは，問題を作成する上での非常に重要なテーマとなる。それは適切な設問こそが，学生の知識，能力，理解度を正確に評価することにつながるからである。

設問の形式は大きく分けると，「多肢選択式」と「記述式」の2つに分類できる。両者の特徴を比較してみると次のようになる。

(1)多肢選択式

「解答群の中から正しいものを選べ」となる設問形式である。一般には，解答候補のリストの中から正解を１つだけ見つけだすという形式が多いが，設問の仕方によっては以下に示すいくつかのバリエーションが考えられる。

・正解が１つとは限らないもの
「正しいものを解答群からすべて選べ。」
・正しくない記述を選ぶもの
・用語や語句の群とその定義文を結びつけるもの
・用語や語句に関する説明文の空欄を埋めることで，定義を完成させるもの
・２箇所の空欄が対になっていて，対応関係が正しくなるように埋めれば正解とするもの
(この場合，解答群に正解が２通り合ってもよい。)

【正解が２通りある場合の設問例】

次のプログラムにおいての手続きＡの部分が10回の繰り返し処理になるように(1),(2)の空欄を埋めよ。

```
PERFORM VARYING K FROM (1)
        BY 1 UNTIL (2)
        手 続 き A
END-PERFORM
```

(1)の解答群

ア．０　　イ．１　　ウ．２　　エ．３

(2)の解答群

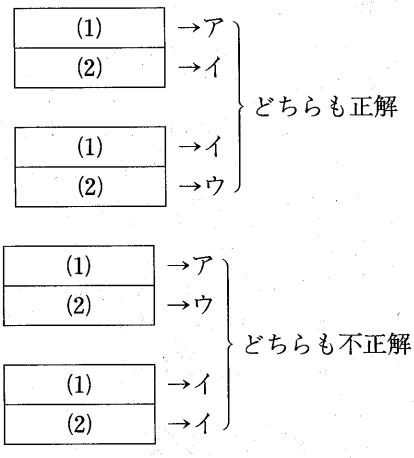
ア．Ｋ＜10　　イ．Ｋ＝10　　ウ．Ｋ＞10

このような問題の場合(1),(2)を満たす正解の組み合わせは無数に存在する。そのため記述式の設問にする場合は２箇所両方を穴にすることはできない。(どちらか一方を見せておくのが普通である。)

多肢選択式の場合は前述したような解答群を設定することで，２箇所両方を同時に問うことが可能になる。さらにこの２箇所を対にし，両方正しいときのみ正解にするような採点方法を取ることで，

「偶然当たってしまった」

となる確率を低く押さえることができる。



以上のような多肢選択式の設問において，考えられる長所と短所を列挙すると次のようになる。

「長所」

・採点基準が一律（一定）に決められる。
・設問の仕方や解答群を工夫することで，難易度を設定することができる。
・解答群があることで，受験者に対して問題への取り組み安さや安心感を与えることができる。

「短所」

・「偶然当たる」ことが起きてしまうため，能力の正確な把握に誤差が生じる可能性がある。(当然解答群の数が少ないほど誤差は大きくなる。)
・解答群の作り方によっては，受験者に不要な（無駄な）時間をとらせてしまう場

合がある。
（解答群の数がいたずらに多すぎる場合や不適切な錯乱肢が含まれている場合など。）

(2)記述式

プログラミングの試験で記述式の設問を作る場合，一般には次の２つの方法がある。

①プログラムそのものを記述させる場合
- ・次のプログラムの空欄を埋めよ。
- ・～を実行するための命令（文）を書け。
- ・～の処理を行うようなプログラム（手続き）を書け。

②文章（日本語）で記述させる場合
- ・～の部分で行っている処理を簡単に説明せよ。
- ・～の時点で，この変数はどのような意味を持っているかを述べよ。
- ・～の処理でエラーとなるのはどのような場合かを説明せよ。など

多肢選択式と同様に記述式についての長所と短所を列挙すると次のようになる。

「長所」
- ・受験生の持っている知識や能力をかなり的確に評価できる。
- ・選択式のような○か×か，だけではない中間的な評価ができる。

「短所」
- ・理解はしているが覚えていないために書けないケースと，全くわからずに白紙になっているケースとの区別がつかない。（あるいは，つけにくい。）
- ・中間点などの採点基準を厳密に決めることが困難なため，採点の手間の割りには点数を一律につけにくい。

特に長いプログラムを，まとめて記述させるような問題を作成した場合，上記の採点基準は大変難しくなる。記述式における採点方法は基本的に
- ・完成された状態からみて，間違いの箇所を減点していく方法（減点法）
- ・プログラムを小ブロックに分け，部分点を積み重ねていく方法（加算法）

の２通りが考えられる。採点はこれらのどちらか，または両方を組み合わせて行うことになる。

正解に近い答案（プログラムがほぼ完成されている答案）に対してはこれらの基準にあてはめて採点することができるが，正解から遠くなるにしたがって，例外的なケースが多くなる。悪い意味ではなく，「点数のつけようがない答案」が出てくるのである。

したがって記述式の問題にする場合（特にプログラムの記述の場合）は，記述をさせる範囲とそれに対する採点基準が明確にできる場合に限る必要がある。前述の「点数のつけられない答案」のようなケースは，問題自体に無理があると見るべきかもしれない。

以上述べてきたように，「多肢選択式」と「記述式」の設問形式にはそれぞれ特徴があり，どちらか一方が万能というわけでもなく，また一方が他方をすべて補完できるというわけでもない。現実には，両方をバランスよく混在させて作問する必要があると考える。

Ⅴ．適切な試験問題

以上述べてきたことに基づいて質の高い適切な試験問題を作成するには，どのような点に配慮をすべきかを考えてみたい。

(1)問題のボリューム

１つの題材をテーマにしたプログラムを試験問題にする場合，その特徴として問題全体のボリュームが大きくなることがあげられる。

受験者にとっては設問の難易度の前に，ボリュームの大きさ自体が負担になってくるため，ここに何らかの配慮が必要になる。問題の構成は，「仕様に関する説明部分」，「プログラム本体」，「設問」の３つの部分から成りたっているのが一般的である。

現在，プログラミングに関する問題で最も多いのは，プログラムそのものを記述させてアルゴリズムを完成させる形式の，いわゆる穴埋め問題である。この種の問題が多い理由は次の点である。

①問題が作りやすい。

仕様とプログラムが決まれば，設問（穴の位置）は配点や難易度に応じて自由に決められ，またその変更も容易である。

②設問が不要である。

「次のプログラムの空欄を埋めよ」の１行があればよい。その結果，問題全体のボリュームを小さくすることができる。

前述の「Ⅲ．必要なプログラミング能力」にあげた項目を満たすための試験として，穴埋め形式の問題を取り入れることは必要である。ただし，それだけでは「問うべき能力」のすべてを問うのには十分でない。したがって穴埋め以外の設問も問題の中に含める必要性が出てくる。

穴埋めの場合，問題のボリュームはプログラムそのものの大きさに比例することになるが，穴埋め以外の形式にした場合は，新たに別の設問が加わることになりボリュームはさらに大きくなる。（このとき設問の形式が，記述式か多肢選択式であるかはあまり本質的な問題ではない。）

問題のボリュームが一定量を超えるようなケースでは，受験者はそのすべてを一度に把握し内容を理解することが大変困難になる。そこで複数の設問を作成する場合は，順を追って思考を展開していけるような流れ（ストーリー）を設問の中に設定する必要があると考える。

一般的な受験者の思考の過程は次のようになるであろう。

> 先ず問題全体に目を通し，次に設問が示している箇所に思考を集中させ，その問題を解くために必要な仕様や条件を確認する。その後さまざまな角度から思考や判断を行い，自分の知識を駆使して解答する。

このような思考の流れを適切に導くように設問の順序，内容，問題数などを設定する必要がある。

設問の流れが唐突であったり，問題数や分量が設問ごとにアンバランスであったり，１つの設問が極端に多くなり複数ページにまたがっていたりするような設定では，思考が乱されたり中断されたりしてしまい良い問題とはいえない。

また問題のボリュームが大きくなるほど，用紙の使い方が重要なポイントとなる。「問題文」，「説明文」，「プログラム本体」，「設問」，「解答群」といった各要素を，どのページのどの位置にレイアウトするかは作問の過程で十分に検討しなければならない。これらは問題の内容やレベルには直接関係のないことでもあり軽視されがちである。しかし適切な問題を作成するという一連の作問作業の中では，問題の質や中身と同程度に重要な事柄であると言える。

(2)教育的な立場からの配慮

一般に試験とは，授業で学んだことをどのくらい理解し，習得しているかをみるために行うもので，教員はこれによって学生の理解度や目標に対する達成度を評価することができる。一方，学生は授業で身につけた自身の

実力を文字どおり「試す」ことができる。

したがって学生は（ある意味では教員も含めて）「試験」と「授業」は全く別のものという位置づけで考えている。繰り返すが試験とは，その時点で学生が持っている知識や能力をどこまで引き出せるかが，最も重要な目的となるものである。

したがって試験の時点で知識のない問題は，本来ならば絶対に解答できないのである。

ここで一歩進めて，「試験」も「授業」の一環であるという視点に立って考えてみたい。この場合，試験も授業の一部であると考える。言い方を変えれば試験の中で学生に授業を行うのである。

問題の中で，学生の知らない知識についての簡単な解説を行い，その上で問題を解かせる。すなわち

> 学生は試験の前には知識のなかったテーマを，試験中に説明を読み，設問によって思考させられ，試験が終わったときに学習できた

ことになる。

もちろん，限られた試験時間内だけですべてを学習させることは無理であろうし，またすべての問題をこのような「授業形式」にする必要もないと思う。

何よりもこの形式にした場合，導入としての解説部が必ず必要になるため，その部分があまり大きくなると試験問題としての体をなさなくなる。したがって設問の形式は，問題の提示からプログラムの最終形までの過程を，導入問題を含むストーリー的な展開にしておく必要がある。

前述したような紙面への配置やボリュームの問題，また試験時間や問題数などの物理的制約もあり，現実にこのような試験問題を作成するのはなかなか容易ではない。ただこのような問題が在ってもよいはずであるし，またそのような工夫もすべきであろう。

VI. おわりに

プログラミングの能力を机上だけの試験で正確に評価することは大変難しいと思われる。それは与えられた１つの仕様や条件から組み立てることができるアルゴリズムというものが幾通りもあるからであり，言い換えれば常に正解がただ１つとは限らないからである。

前項まで，設問のあり方や問うべき能力についてを述べてきたが，プログラミングの試験を大きくとらえてしまえば次の３つの要素に要約できる。

(1)アルゴリズムを読みとること
(2)アルゴリズムを組み立てること
(3)アルゴリズムを正しく表現すること

このうち(3)については

「文法を正しく理解しているか」
「コーディング規約に従い正しく記述ができるか」
「データ名や予約語などを正しい綴りで書くことができるか」

といったような内容が問われることになり，知識の理解と同時にどれだけ正確に覚えているかが設問の中心テーマとなる。

たとえばSORT文を記述させる次のような穴埋めの設問があったとする。

```
SORT  BUNRUI-FILE
    ON     (A)     KEY  S-KEY
    INPUT  (B)     IS   I-PRO
    OUTPUT (B)     IS   O-PRO.
```

当然(A)には「ASCENDING」や「DESCENDING」という語句が，(B)には「PROCEDURE」という語句が入る。試験である以上，これらの語

句に１文字でも綴りのミスがあれば正解とはならない。

ところがプログラミングの実際の現場ではマニュアルを片手にコーディングするのが当たり前であり，暗記していなくともマニュアルを見て正しく書くことができればそれで充分なのである。

したがって上記のような設問がプログラミングの試験としてどれだけ有効で，意味を持つかは議論の分かれるところである。正しい綴りを正確に書ける学生が，必ずしもプログラミング能力が高いとは言えないからである。(ただし世の中で行われている各種のプログラミング検定試験などで，この手の問題が含まれているのもまた事実である。)

これに対し，前述の３つの要素のうち(1)や(2)については理解力や暗記力を問うという面は少なく，むしろ限られた時間の中において，正確で効率的なアルゴリズムの作成や検証がどれだけできるかが求められる。実はこの部分がプログラミング試験の最も重要で特徴的な点である。また先に述べた授業形式の試験を実践する場合においても，題材として取り上げやすい要素がこの中に多く含まれている。

すなわち学生は自分の知らないテーマや題材について，問題の中で説明を受け，判断と思考を繰り返しながら論理の組み立てや検証をさせられる。これによって自分にとっては未知のアルゴリズムやプログラミングテクニックを試験の中で学習することができるのである。

頭の中にしっかりと覚え込んだ知識を，厳密に問うための試験を一方で行い，もう一方で問題を解きながら学習を進めていくスタイルの試験を行う。これら二種類を両立させていくことで，より教育的な試験が行えるのではないだろうか。

このような視点に立ち，両者のバランスを考えた試験問題の作成が重要であるということを最後に指摘しておきたい。