スパイラル・アプローチによるC言語プログラミング教育の実践と評価――受講生中心の学習を目指した試み

Trials and Evaluation of Spiral-Approach-based Curriculum for Programming Course in C Language――Challenge to Student-oriented Education

和田　浩

要　約　プロのプログラマにはエンドユーザには実現できない高度な機能を持つプログラムの開発能力が要求される．またプログラムには「使いやすさ」や「高い保守性」や「効率の良さ」が期待される．すなわちプログラミング教育も，教育の初期段階からプログラムの品質を意識させることが望まれている．基本的なアルゴリズムや文法だけを教えるのではなく，利用者の要求を真に満足し仕様の拡張への対応も意識したプログラミングを教える必要がある．

本稿では，プロフェッショナル・プログラマを育成するための，スパイラル・アプローチに基づくカリキュラムを提案する．

このカリキュラムでは演習課題を漸進的に展開（スパイラル方式）し，徐々にプログラムを進化させる．この学習アプローチでは，開発するプログラムの規模が徐々に大きくなるため，その過程で，プログラムの品質を意識させることが可能になる．最終課題ではユーザ・インタフェースを利用者の立場から考え，上流工程の一部を体験させる．

Abstract　Professional Programmers will be expected to develop programs which offer more powerful and advanced functions than average endusers do. Also, programs will be required for the usability (easy to use), high-level maintainability and efficiency. Therefore, programming education should give a chance for students to consider the quality for programs from an early phase of educational course.

In addition to the basic algorithm and syntactical matters, programmer should learn how to write programs which satisfy user's requirements and endure changes to program spcifications.

This paper discusses the spiral-approach-based curriculum for beginners who aim to become professional programmers.

This curriculum grows an exercise gradually (spiral approach), so a program will be evolved bit by bit. In this approach, the program size will become large, so there are many chances for students to consider the quality of program. It the final exercise, students try to design the user interface from the side of enduser, so they can have a chance to develop the program specification.

1. はじめに

パソコンの低価格化・高性能化に伴ってEUC（End User Computing)/EUD（End User Design）が進展する現在，プログラマのあり方が問い直されている．職業（プロ）としてのプログラマには，エンドユーザには実現できない高度な機能を持つプログラムの開発能力が要求される．また彼らが作るプログラムは，今まで以上に使いやすさの面からの品質も要求される．作成されるプログラムコードは，変更への対応が容易であり，効率が良いことも求められる．

プロのプログラマを育成することを目的としたプログラミング教育は，プログラムの品質を意識させるような教育コースが必要である．単に基本的なアルゴリズムや文

法だけを教えるのではなく，利用者の要求を満足し，仕様の拡張への対応も意識させるプログラム教育が必要である．

アルゴリズムや文法だけを教えるのであれば，個々のテーマごとに最適の例題や演習課題を用意するだけで，十分に対応することができる．市販されている教材の多くは，このような形式に基づいて記述されている．このような細切れの課題対応型に期待できるのは「コードの書き方」の修得までであろう．

学習テーマごとに演習課題を別々に用意するのではなく，プログラム開発の成果物を一つ想定し，その開発過程のそれぞれの段階に適した課題を提供する方法を提案する．この方法を用いると，開発するプログラムの規模が徐々に大きくなるため，それぞれの過程で，プログラムの品質を意識させることが可能となる．また初級・上級のようにコースが複数に分割されず，短期間で効果的に学習できる．

文法中心の学習アプローチでC言語の個々の機能を学習しても，それだけではプログラムを開発することはできない．これに対し，提案する方式でプログラミングを学習した受講生は，C言語のマニュアルがあれば，C言語でプログラムを開発することが可能になる．

本稿の2章で，現在のプログラミング教育の問題点とその解決方法，3章で，選定した演習課題と選定理由，4章で，カリキュラムの狙い，5章で，コースを実施したときの受講生の反応・評価，6章で，このコースのコンセプトである「受講生中心の教育」について述べる．

2. 新しいカリキュラムの提案

本章では，プログラミング教育の現状とその問題点を述べ，次にその問題点を解決するためには何が実現されれば良いか，そして，その実現方法に関する方針を述べる．

2.1 文法・機能中心のプログラミング教育の限界

現行のプログラミング教育のほとんどは，プログラミング言語の文法や機能を短期間で習得することが中心になっている．したがって実際にプログラム開発を行なうときに守らなければならないプログラム書法等の習得に関しては，OJTで行なうことが多いと想定される．

OJTでの指導内容は，指導者の品質に関する意図や意識，さらに開発内容に依存するため，教育レベルを一定に保つことが困難である．一般に品質の検証も，意図して実施しない限りできていないのが現状である．受講生が機能の動作確認程度のプログラミングしか体験していない場合には，どのようなプログラムが「よいプログラム」であるかの判断や課題の意味・重要性の認識をさせることも困難である．

担当分野が限定されており，品質を意識することなくコーディングできるような，小さくて簡単なプログラミングの仕事だけを期待する場合は，上記の教育で十分であろう．しかし，プログラムの設計方針から自分で考えられるプログラマに育てたい場合には，OJTの指導者に高度な教育技術が要求される．なぜなら，受講生にプログラムの品質について悟らせるために，様々な工夫が必要だからである．

2.2 プログラミング教育に期待されること

上級プログラマやSEへと育成していく場合には，教育の初期段階からプログラム

の品質を意識させ，課題の意味・重要性も認識させることが大切である．

初心者にプログラムの品質を意識させることは簡単ではない．「他人が無理なく理解できるプログラムを作りなさい」とか「仕様の変更に対処しやすいプログラムを作りなさい」と説明するだけでは，どのように工夫すべきかが分からないであろう．同様に，数十ステップの小さなプログラムを何本作成しても，プログラムの品質の重要性はわからないであろう．品質を意識させるためには，最低でも数百ステップのプログラムを開発させる必要があると想定される．また，受講生同士でプログラムを交換し，メンテナンスを体験させる等の工夫も必要である．

課題の意味・重要性を認識させ，課題に集中するための最も効果的な方法は，受講生自身にとって，本当に便利で役立つプログラムを作成させることである．受講生自身がエンドユーザになれば，プログラムを利用者の立場から評価することができる．また仕様の一部を受講生に考えさせるようにすれば上流工程の体験にもなる．

2.3 スパイラル方式を用いたカリキュラム編成の提案

文法・機能を教えるための課題を個々に準備する方法では，プログラムの品質を意識させる教育を実現することはできない．OJT のように本格的な課題（仕事）を与え，実際にプログラムの開発を体験させれば，プログラムの品質を意識させることが可能になる．ただし受講生にいきなり本格的なプログラムの仕様を与えても，どのように開発すべきかが，ただちには分からないであろう．

これらを解決する新しい教育手法を提案する．ある程度大きいプログラムサイズの最終課題を想定し，初めはその最終課題を極端に簡単にしたもので文法等を学習し，徐々に課題を最終課題へと近付けていく．この方式であれば初めは課題が簡単なため初歩的な文法や基本アルゴリズムの学習に専念することができる．学習が進むにつれ徐々に課題は難しくなるが，前回作成したプログラムを修正することで対応できるため負担は少ない．またプログラムの改良，メンテナンスも同時に体験できる．最終課題までにプログラムが数百ステップ以上になれば，そのプログラムの開発過程で，プログラムの品質についても十分に学習する機会ができる．最終課題を受講生自身にとって本当に便利で役立つものにしておけば，最終仕様の詳細は受講生自身に考えさせることも可能になる．このように演習課題を漸進的に展開し，徐々にプログラムを進化させることを「スパイラル方式」と呼ぶことにする．

3. 演習課題の選定

演習課題をスパイラルに展開し，そのプログラム開発体験を通してプログラミング教育を行なうためには, コース開発では適切な演習課題を設定することが重要である．

3.1 課題が具備すべき特徴

1） C 言語の基本機能をすべて説明できる課題

到達すべき最終課題は一つにする．ただし初めは課題を極端に簡単なものにして，その課題を徐々に現実的な複雑なものへと変更していく．すなわち課題は漸進的展開のできるものにする．この課題を通して，次の学習項目を修得できなければならない．

・アルゴリズムとして，単純列構造・選択構造・繰り返し構造

・C言語文法として，入力と出力・配列・ポインタ・関数・構造体・ファイルの概念

2) C言語に向いた課題

C言語はCOBOLと比べると可変的なデータの扱いが容易であり，小回りのきくプログラミングが可能である．このようなC言語の特徴を生かせる課題としたい．

3) 親しみやすく魅力のある課題

学校を卒業したばかりの新人にも理解しやすい課題にする．課題は現実的なものにしたいが特定の業務知識を前提にすることはできない．1995年度までは当社顧客教育のC言語コースの期間は15日であったが，新規に開発した1996年度版のコースは18日以内にする必要があった．このような短期間では，業務知識の学習に多くの時間を当てることはできない．また受講者が本当に開発したいと思うような魅力のある課題にする．できれば最終作品を受講者が持ち帰り自分で使いたくなるようなものにしたい．

3.2 キータッチ・トレーニング・ツール

このような条件を満たす課題として，タッチ・タイピング（キーを見ないですらすら入力すること)を習得するときに役立つ,「キータッチ・トレーニング・ツール」(以下KTTと記述する）を考えた．KTTをスパイラルに開発するのである（実際には，全部で25題，最後の1題は約500ステップのコースである)．

4. カリキュラムのねらい

カリキュラムはKTTの開発体験を中心に構成した．期間は18日間とし，補助資料の文法編として「C言語プログラミング―基礎編（当社顧客教育のテキスト)」とアルゴリズム編として「C言語によるプログラミング入門(当社顧客教育のテキスト)」を使用する．

主教材は，課題の説明と設計上のヒントから成り立っており，各課題で必要となるC言語の機能や設計方法については，上記の文法編とアルゴリズム編を使用する．最終課題までに文法編とアルゴリズム編のほとんど全てを説明するし，受講生が自力で学習できるレベルに達しているため，特に最終課題完成後に補助資料の未学習部分を説明する必要はない．

表1は，KTTのカリキュラム例である．

以下の4.1節～4.4節で四つのフェーズに分けてカリキュラムの狙いを説明する（表1の各課題の内容は付録の「KTT演習課題」参照)．フェーズ1では，簡単なキータッチ入力処理を目的として，課題1～9を実施する．フェーズ2では，簡単な入力データの分析を目的として，課題10～17を実施する．フェーズ3では，入力単語の分析を目的として，課題18～21を実施する．フェーズ4では，定数データファイルの使用を目的として，課題22～25を実施する．

4.1 簡単なキータッチ入力処理

既存コースのコンピュータ入門，UNIX入門，VI入門（合計2.5日）の中で，課題1で必要となる機能を0.5日で教える．UNIXコマンドやエディタ機能に関しては必

表 1　カリキュラム概要

	9：30	12：00	13：00	16：30
1日目	オリエンテーション コンピュータ入門		ＵＮＩＸ入門 ＶＩ入門 **簡単なキータッチ入力処理** 課題１～３	
2日目	課題４　（説明）		（実習）　課題５　課題６	
3日目	課題７～９ **簡単な入力データの分析** 課題１０		課題１１　ドキュメント作成	
4日目	受講生同士によるドキュメントのチェック 課題１２		課題１３　（設計開始）	
5日目	（実習）		（実習）	
6日目	ドキュメント作成 受講生同士によるドキュメントのチェック		課題１４	
7日目	課題１５		課題１６	
8日目	課題１７		**入力単語の分析** 課題１８ プログラム・リストの分析作業開始	
9日目	（設計開始）		（設計） 受講生同士によるドキュメントのチェック	
１０日目	（設計）		（実習）	
１１日目	課題１９		ドキュメント作成	
１２日目	メンテナンス作業　課題２０		課題２１　メンテナンス作業	
１３日目	**定数データファイルの使用** 課題２２		課題２３	
１４日目	課題２４			課題２５
１５日目				
１６日目			受講生同士によるドキュメントのチェック	
１７日目				
１８日目	ドキュメント等のまとめ		課題２５の完成 受講生同士による批評	

要に応じて随時紹介する．上記入門部分も含め本節の標準学習期間は 2.5 日間である．

1)　メッセージの表示

課題 1〜3 で開始メッセージや手本文字列等の表示を行う．このとき改行文字を一つ追加するだけでメッセージの表示がダイナミックに変わることを体験する．

2)　文字の入力

課題 4 では手本文字列の表示だけでなく，入力もできるようにプログラムを修正する．ここで繰り返し構造のアルゴリズムについて学習する．

3)　プログラムの洗練

課題 5〜9 で C 言語特有の機能を使用してプログラムを洗練（読みやすく）する．ここで同じ課題でも様々なプログラムの記述の仕方があることを理解する．

4.2　簡単な入力データの分析

ここでは機能を追加しながら，配列やポインタについて学習する．

1)　入力スピードの測定

課題 10 で入力スピードの測定を行い，課題 11 で入力スピードの値に応じたメッセージの表示機能を実現する．ここで選択構造を学習する．

課題１０

課題９に以下の入力スピードの測定処理を追加する。
KTT の入力文字列が表示されてから、実際にデータが入力されるまでの時間を秒単位で表示する。

＜KTT の入力文字列＞
入力データ　　} 表示例
ｔｉｍｅ＝３２

課題１１

課題１０を修正し、入力スピードの値によって、以下のメッセージも表示する。
２０秒以下　　…　“すばらしい！”
４０秒以上　　…　“がんばって！”
２０秒以上、かつ４０秒以下　…　“その調子！”

課題 12 では for 文と if 文を応用して，課題 11 のプログラムを洗練（冗長性の排除）する．

2)　入力データと文字列定数の比較

課題 13 で手本文字と入力文字の比較処理を行う．ここで一次元配列を学習する．またこの課題から，適性の高い受講生には，課題の追加機能を自分で考え挑戦させる．

課題１３

課題１２に以下の入力データの分析処理を追加する。
表示した「KTT の入力文字列」と実際の入力データに１文字でも違いがあれば、
入力した文字の内、エラーした文字の数を表示する。
前回までの処理を＜スピードの練習＞とし、
今回追加した処理を＜正確さの練習＞と呼ぶことにする。
＜正確さの練習＞は３回繰り返すこと。

3)　プログラムの洗練

課題 14～16 では，二次元配列やポインタを使用してプログラムを洗練（冗長性の排除）する．

課題 17 では EOF データを入力してプログラムを終了する機能を実現する．この課題で論理演算子を学習する．

4.3　入力単語の分析

ここでは既存プログラムの分析とメンテナンスを体験しながら関数と構造体を学習する．

1） 空白文字の処理

課題 18 では，単語単位で入力データを手本単語と比較する．この課題では初めに解答例のプログラムリストを受講生に与え，以下の手順でプログラムを開発させる．

① プログラムリストの分析

② 内部仕様書の作成

分析したプログラムリストから内部仕様書を作成する．

③ 内部仕様書の洗練

②で作成した内部仕様書でアルゴリズムの不統一な箇所が見つかれば修正する．

④ プログラムの作成

③を参考に，課題 17 のプログラムに修正・追加を行なって，課題 18 のプログラムを作成する．

2） プログラムの洗練

課題 19 では，処理の共通部分を関数で実現する．この課題では他の受講生のプログラムを修正する．したがって講師は各受講生の相手を決め，以下の手順でプログラムを開発させる．

① 変更指示書の作成

・関数の外部仕様書

・プログラム内部仕様書

・関数の内部仕様書

② プログラムのメンテナンス

他の受講生が作成した①の変更指示書に従って，その受講生のプログラムを修正させる．

課題 20 でプログラム構造に対して関数を使用し，課題 21 で構造体を使用してプログラムを洗練する．課題 21 でもメンテナンスを体験させる．

4.4 定数データファイルの使用

ファイルも使用して，より現実的なプログラムにする．また最終課題では，仕様も受講生に決定させ，スケジュールにしたがってプログラム開発を体験させる．

1） ファイルの入力処理

課題 22 では，プログラム内部で持っていた定数データを，ファイルからプログラムの配列に読み込むように修正する．課題 23〜24 では，手本文字データを増やし複数レッスンの練習ができるようにプログラムを修正する．

2） キータッチ・トレーニング・ツールの完成

課題 25 では，これまで開発してきたプログラムをベースに，さらに使いやすくする為の仕様を自分で考え，キータッチ・トレーニング・ツールを完成させる．以下の手順でプログラムを開発させる．

① プログラム開発スケジュールの作成

② プログラム外部仕様書の作成

③ プログラム内部仕様書の作成

④　コーディング

5.　プログラマ養成コースの実施と受講生の反応と評価

1996年度より，当社顧客教育にてKTTのカリキュラムで筆者が担当した「C言語プログラマ養成コース」を4回担当した．そのとき講師が感じたこと，注意・工夫した点，受講生の反応や評価を述べる．

5.1　KTT課題の評価

選定したKTT課題で，C言語の基本機能をすべて無理なく説明できたと考えている．またタッチ・タイピングのエラー分析等では，C言語の特徴を十分に生かすことができたと思われる．受講生にも好評であり，KTTが最適の課題の一つであることが確認された．

5.2　課題のスパイラルな展開による学習アプローチ

KTTのカリキュラムでは課題をスパイラルに展開しながら，プログラム開発を体験させ，新知識・技術が必要になったところで学習させているが，実際にこの方式でコースを実施してみると以下のような利点が認められた．

1)　適切な動機付け

初心者にとっては，身近に感じられる課題から話が始まり，その課題達成のために必要な最低限の知識の説明が後に続くため，新しい知識を無理なく吸収できる．

はじめにアルゴリズムや文法を教える方法の場合，初心者から見ると，いきなりたくさんの機能や技術の説明を受けることになり，負担が大きかった．それに比較すると，このスパイラル学習アプローチでは，動機付けが十分に与えられ，さらに前日までのプログラム開発体験を基にして新知識を学習できるため，初心者への負担が少ない．

2)　現実的なプログラムの規模

多少のプログラミングの経験や知識のある人にとっても，最終的には700〜1500ステップ（COBOLで約2000〜4000ステップに相当する）規模のプログラム開発を体験でき，自己流の開発スタイルの悪い所がわかるなど，好評であった．

最終課題（課題25）のプログラムの規模は，平均で約500ステップになる．この最終課題に到達する前に，初心者の場合で150ステップ規模，プログラミング経験者でも300ステップ規模になると，自然にプログラムの設計の重要性を認識するようになった．

文法・機能中心のプログラミング教育の場合，学習した機能の動作確認を目的とする数十ステップ規模のプログラムを開発する．このとき，コース前半で学習したアルゴリズム設計法で開発させることになるが，これは講師と受講者の両方にとって効率の悪い方法である．すなわち講師は設計の重要性を何度も強調しなければならず，受講生は人工的に作られた極端に簡単な課題でプログラムの設計作業を強要される．

プログラムの品質を意識させる為には，プログラムの規模を大きくするだけでは不十分であり，受講生同士によるチェックや，プログラムリストの分析，メン

テナンス作業のような工夫が必要である．

5.3 受講生同士による検証作業の有効性

受講生同士で数回，内部仕様書（後半では外部仕様書も）やプログラムリスト（最終日には実行も）をチェックさせた．他人の作品を見ることによって，その悪い所も良い所も参考になり，さらに自分の作品が複数の他人から見られることによって他人の目というものを自然に意識するようになった．

また講師にとっては受講生のコメントから，個々の受講生の理解力や考え方等を知ることができ，個別アドバイスをするときの参考になった．

5.4 既存プログラムの分析作業の有効性

課題 18 では既存プログラムのリストを渡し分析作業を行なわせた．具体的には既存プログラムリストから内部仕様書を作成し，修正コーディングを行なわせた．用意した既存プログラムは一部分かりにくい所や無駄な所があり，さらに特殊データに関してバグが発生するような不完全なものである．

課題もそれまでとは比較にならないほど難度が高く，受講生にはかなりの負担になる．全員が課題をクリアするのに 2 日半は必要である．特に既存プログラムのリストの分析作業が初心者には大変な負担になるので，毎年この分析作業用に別室を用意している．

初心者には孤独で辛い作業なので，講師は個別に適切なアドバイスを行ない，受講生が「自力で分析作業を行なうこと」を助けなければならない．

しかしこの課題（課題 18）を境にして，受講生の分析力は飛躍的に向上したと感じられる．たとえば初心者の場合，課題 17 までは自力でのデバッグが難しく，講師がサポートする必要があるが，課題 18 以降ではほとんど自力で自分のプログラムを分析してバグを発見できるようになる．

5.5 プログラムのメンテナンスの有効性

課題 19 と課題 21 で，プログラムのメンテナンスを体験させる．課題 19 では変更指示書にしたがってメンテナンスを行ない，課題 21 では変更指示書無しで，それまでの内部仕様書とプログラムリストだけでメンテナンスを行なう．

講師は，ある受講生が，2 回のメンテナンスでそれぞれ他社の 2 人の受講生のプログラムを保守してもらうように組み合わせを工夫する．保守作業を通して他会社の 4 人の方と接することになる．この工夫によって本番同様の緊張感が生まれる．

変更指示書の書き方が未熟で，変更内容について質問を受ける受講生が毎年いる．逆に内部仕様書が分かりやすく正確に記述されているために，作業がはかどった受講生もいる．

この体験により，ドキュメントの重要性をしっかりと認識することができる．またプログラムは，メンテナンスを考慮して開発しなければならないことを認識し，以後の作業では，自分のプログラムを保守性の面から厳しくチェックするようになる．

ほとんどの新人が，自分が他人に伝えたいことを，日本語や図を使用して正確でわかりやすく表現することの難しさを悟っていない．またプログラミング経験者も，作成したプログラムはすべて開発者本人の個人的なプログラムであり，他人が使用したり保守したりすることを考慮していないことが多い．このために，ドキュメントの不

備や洗練されていないプログラム構造によって，他人が苦労しているところを目撃すると，自分の未熟さが恥ずかしくて，一時的に自信を失うほどのショックを受ける人もいる．

従来のプログラミング教育では，設計やドキュメントの重要性を認識させることが困難であった．KTTのカリキュラムではこの点に関しては飛躍的に改善された．

5.6 プログラム外部仕様書の作成

最終課題では，ユーザインタフェースを中心とするプログラムの仕様を，受講生自身で考えさせた．このときプログラム開発スケジュール(期間は4日間)も作成させ，納期厳守を目標とした．この作業を通して，自分の実力を見極めることと，納期を守ることが予想以上に難しいことを悟ることができる．

最終日の午後，受講生同士でプログラムの実行チェックを行なうが，ひとつとして同じものが無く，人間の多様性を感じさせる．それだけにドキュメントも含めて，プログラムを個性も価値観も異なる様々な人々すべてに理解され受け入れてもらうためには，自分の作品に対する，多面的角度からの客観的で厳しいチェックが必要であることを，多くの受講生が認識した．

6. 受講生中心の学習を目指して

弊社顧客教育における従来の新人教育では，講師中心に進められることが多く，受講生は受動的になりがちだった．今後はこの点を改善し，「受講生自らが学習していくこと」を支援する教育へと脱皮する必要がある．

この目的のために，過度に具体的で詳細なアドバイスは控えるように努め，模範解答例も準備しないようにした．結果として，受講生同士のドキュメントやプログラムのチェックが重要になる．受講生にはドキュメントの先頭に，コメント用の用紙を数枚用意させ，作品全体に対する意見を書き込めるようにする．またポストイットも使用して自由にコメントを付けさせる．

受講生の作品は，講師もチェックしコメントも付けるが，なるべく受講生自身によって問題点を悟らせ，解決策を考えさせるように心掛ける．これは，コース期間が18日間と長く，プログラムの規模も大きくなるため，その開発過程で自然に受講生が気づくことが多いことによる．また受講生のコメントに対する講師のコメントは行なわない．はじめのうちは，的外れなコメントもあるが，自分でプログラム開発をしながら，仲間の作品を多く見ているうちに，プログラムの質だけでなく，コメントの質も良くなる．講師は受講生を高所から指導するのではなく，受講生が“今，ここで体験していること”を尊重し，受講生自身が自分の頭で考えてプログラムの品質を向上していくことに協力しなければならないと感じる．

受講生に，知識・技術を強制的に押しつけるのではなく，自然に理解してもらえるのが理想と考え，忍耐強く，受講生が本来持っている能力を信じてコースを進める必要がある．実際に行なってみると，受講生の潜在能力には素晴らしいものがあり，講師はほとんど進行役に徹すれば良い日もあった．

文法・機能中心のプログラミング教育の場合は，学習テーマごとに，別々の課題が用意される．プログラムの規模もほとんど変わらないため，受講生が自然に自分のド

キュメントの書き方やプログラミング・スタイルの欠点に気付く可能性が低い．そのために，通常は各課題ごとに解答例を用意する．この方法だと受講生は解答例を頼るようになり，自分で悩みプログラミングに関して熟考する機会を失う．講師も個々の課題ごとに結論を出さなければならないため，解答例と大幅に異なる書き方やスタイルに対しては否定的になりやすい．本当は受講生のユニークなアイデアに対して，たとえ未熟なものでも否定せず，むしろ自力でゼロから考え出したことを評価したいのであるが，そのゆとりが無くなるのである．

「新人はプログラミングに関して未熟だから，はじめに手本として洗練されたドキュメントと，プログラムリストを渡すべきだ」と主張する人達がいる．筆者は反対である．新人の中にはドキュメントに関して才能のある人がいる．ユニークな発想でアルゴリズムを設計できる人もいる．多様で異なる知識・経験を持ち，情報処理技術者という共通の目標を持つ受講生が互いに協力して，自分達の弱いところを分析し，プログラムに改良を加え，仲間のコメントや講師のアドバイスを参考に，自分の作品を様々な観点から見直して，納期に間に合うよう努力する．この体験を通してプログラミングが理解できた人であれば，自分の会社の先輩達のプログラムを鑑賞できるであろう．はじめに手本として洗練されたドキュメントとプログラムリストを渡しても，その価値を正しく認めることができず，受講生を受身にさせるだけである．

7. おわりに

1996 年度より KTT のカリキュラムを用いて，弊社顧客教育コースとして 16 クラスが実施され，筆者はその内の 4 クラスを担当した．顧客教育では，受講生のアンケートの総合評価の目標が 3.4（4 点満点）であるが，筆者の担当した 4 クラスの平均は 3.6 だった．この評価は，自分の平均と比べてもかなり高いものである．ただしコースの客観的評価は難しく，この数値もあくまでも目安にしかならない．

主観的には，筆者が担当したクラスの内の二つは，通常のクラスとは良い意味で異質なものだった．1997 年度の 5 月に担当したクラスでは，2 回のメンテナンス体験後，ドキュメントやプログラムの設計に関して，受講生の意識が変化した．「プログラムは動けば良いのだ」と考えて，いきなり画面に向ってしまう受講生が，真面目に設計から行ない，プログラムが正しく動くことを確認しても，再度ドキュメント上の不備がないか慎重にチェックするようになったのである．「そろそろ次の課題に入ったら？」と催促すると，「和田さん，ドキュメントが重要なんですよ」と言われてしまうような体験は初めてであった．また 1997 年度の 5 月に担当したクラスでは，学生時代に C 言語を勉強していた受講生がベストを尽くしてくれ，1550 ステップのプログラムを完成させた．彼は自分の作品の設計方針などを，わかりやすく表現することにも挑戦したため，後半ではクラスで一番遅くまで開発作業に取り組むことも多かった．通常は初心者が最後まで残ってしまうのだが，そうならなかったことで，初心者にとっても良い雰囲気のクラスだったと思う．どちらのクラスでも印象的だったのは，後半で受講生自身が「C 言語の機能」よりも「プログラムの品質」に興味を持つようになったことである．

筆者も初めからスパイラル方式で，これらのクラスのような「受講生中心の学習」

を実現できたわけではない．一回目にコースを担当したときは，受講生の作品の悪いところが見つかると，受講生の成長過程や，自然に気付く可能性を考慮しないで，単純に修正指示を与えてしまう傾向があった．数回担当すると，アドバイスの程度やタイミングについて要領がわかるのだが，このことをインストラクタ・ガイド（インストラクタのための教授指示書）で明確に記述することは難しい．「受講生中心の学習」をスパイラル方式で実現するためには，担当講師の高水準の教育技術が前提になる．特に担当講師が「文法中心」の教育を受け，職場で先輩から設計やコーディングのスタイルを強制的に守らされてきた場合には，「課題中心だが講師中心」の教育に退化しがちだ．今後スパイラル方式でプログラミングについて学んだ人達の中から，真の「受講生中心の学習」を実現できる優秀な講師が多く生まれることに期待したい．現在，本稿で用いた課題と方式はVirtualCampusの教材として活用されている．

[添付資料]

KTT演習課題

課題1
　“キータッチ・トレーニングを始めましょう。”を表示する。
［学習内容］
　・Cプログラムの構造
　　main関数、注釈、printf関数
　・Cプログラムのコンパイル方法
課題2
　課題1の文字を表示し、改行する。
［学習内容］
　・定数
　・エスケープ・シーケンス
課題3
　課題2にキータッチ・トレーニングで使用する一つ目の入力文字列の表示を追加する。
（以後「キータッチ・トレーニングで使用する一つ目の入力文字列」のことを「ＫＴＴの入力文字列」と記述する。）
課題4
　課題3にＫＴＴの二つ目の入力文字列の表示を追加し、入力もできるようにする。
［学習内容］
　・getchar関数
　・変数
　　変数の種類、命名規則、変数の宣言、値の代入
　・アルゴリズム
　　繰り返し構造、単純列構造、アクション・ダイアグラム
　・while文
　・関係演算子の種類と優先順位
　・文字定数の使用法

課題５

while 文の条件式で代入文を使用し、課題４を書き直す。

［学習内容］

・式と文

・演算子の優先順位

課題６

課題５を修正し、５回繰り返し練習できるようにする。

［学習内容］

・算術演算子

・代入文

課題７

インクリメント演算子を使用し、課題６を書き直す。

［学習内容］

・インクリメント演算子とデクリメント演算子

課題８

for 文を使用し、課題７を書き直す。

［学習内容］

・for 文

課題９

マクロ定義を使用し、課題８を書き直す。

［学習内容］

・マクロ定義

課題１０

課題９に以下の入力スピードの測定処理を追加する。

KTT の入力文字列が表示されてから、実際にデータが入力されるまでの時間を秒単位で表示する。

表示例

＜KTT の一つ目の入力文字列＞

入力データ

ｔｉｍｅ＝３２

＜KTT の二つ目の入力文字列＞

入力データ

ｔｉｍｅ＝２６

［学習内容］

・printf 関数による変数の出力

・ライブラリ関数とファイルの取り込み

time 関数

課題１１

課題１０を修正し、入力スピードの値によって、以下のメッセージも表示する。

２０秒以下　　…　“すばらしい！”

４０秒以上　　…　“がんばって！”

２０秒以上、かつ４０秒以下　…　“その調子！”

［学習内容］

・選択構造
・if文

課題１２
課題１１で、全く同一のメッセージ表示処理等を複数の場所で行っている場合は、
for文やif文を使用して一ヶ所で行うようにプログラムを修正する。

課題１３
課題１２に以下の入力データの分析処理を追加する。
表示した「KTTの入力文字列」と実際の入力データに１文字でも違いがあれば、入力した文字の内、エラーした文字の数を表示する。
前回までの処理を＜スピードの練習＞とし、今回追加した処理を＜正確さの練習＞と呼ぶことにする。＜正確さの練習＞は３回繰り返すこと。

［学習内容］
・配列
・ｓ変換
・段階的詳細化

課題１４
二次元配列を使用して、課題１３のプログラムを洗練する。

［学習内容］
・二次元配列

課題１５
ポインタを使用して、課題１３のプログラムを書き直す。

［学習内容］
・ポインタとアドレス

課題１６
ポインタ配列を使用して、課題１４のプログラムを書き直す。

［学習内容］
・ポインタ配列

課題１７
control D（EOFの入力）で終了できるように、課題１６で作成したプログラムを修正する。

［学習内容］
・入力文字と定数との比較
・論理演算子

課題１８
単語と単語の間の空白文字を調べ単語単位で入力データを分析し、
＜正確さの練習＞で
“Mワード中 Nワード　正確に打てました。”
と表示する。

［学習内容］
・プログラムリストの分析

課題１９
課題１８の処理の共通部分に対して関数を使用する。

［学習内容］

・関数の作成
・自動変数と外部変数
・ポインタと関数引数
・アクション・ダイアグラム上での関数の記述法
・プログラムのメンテナンス

課題２０
課題１９のプログラム構造に対して関数を使用する。

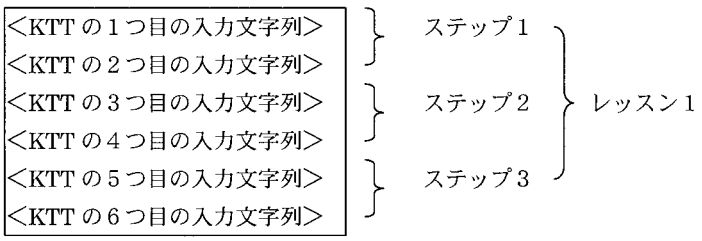
課題２１
構造体を使用して、課題２０のプログラムを洗練する。

［学習内容］
・構造体
・プログラムのメンテナンス

課題２２
課題２１のプログラム内部で持っていた定数データ（KTT の入力文字列）を、
ファイルからプログラムの配列に読み込むように修正する。

［学習内容］
・ファイルの定義
・ファイルのオープン
・ファイルの読み込み
・ファイルのクローズ

課題２３
以下のような定数データファイルがある。



この定数データファイルを読み込み、単一レッスンのキータッチ・トレーニングができるように課題２２のプログラムを修正する。
ただしキータッチ・トレーニングの順番は

ステップ１ ⇒ ステップ２ ⇒ ステップ３

とし、各ステップでは前回までと同様に

スピードの練習（５回） ⇒ 正確さの練習（３回）

で行うこと。

課題２４

以下のような定数データファイルがある。

```
＜レッスン１＞
空白行
＜レッスン２＞
空白行

   …

＜レッスン８＞
```

この定数データファイルを読み込み、複数レッスン（８レッスン）のキータッチ・トレーニングができるように課題２３のプログラムを修正する。

ただしキータッチ・トレーニングの順番は

レッスン１ ⇒ レッスン２ ⇒ … ⇒ レッスン８

で行うこと。

課題２５

今回の課題は、プログラム改良の「開発スケジュール」を作成するところから始める。プログラムの仕様については、特に指定は無い。これまでに作成してきたプログラムをもとに、タッチ・タイピング練習ツールとしてより使用しやすいプログラムを、自分なりに工夫して作成しなさい。

［学習内容］

・switch 文
・do while 文
・プログラム開発スケジュールの作成
・プログラム外部仕様書の作成

参考文献　［1］ 竹田尚彦・大岩　元, プログラム開発体験に基づくソフトウェア技術者育成カリキュラム, 情報処理学会論文誌, Vol. 33, No. 7, pp. 944〜954, 1992.

執筆者紹介　和　田　　浩（Hiroshi Wada）
1979年早稲田大学理工学部数学科卒業．同年日本ユニシス(株)入社．Aシリーズの顧客システム開発に従事．1989年よりAシリーズの顧客教育を担当する．1992年よりプログラミングや設計に関する顧客教育を担当し，現在，総合教育部教育サービス推進室に所属．