導入プログラミング教育におけるオンライン ジャッジシステムの活用の試み

松 永 賢 次 (専修大学ネットワーク情報学部)

An Experiment on Using Online Judge System in Introductory Programming Education

Kenji MATSUNAGA (School of Network and Information, Senshu University)

Students majored in Computing are required to acquire programming skills for fifty years. In introductory programming classes, after students solve ten of programming exercises, their ability to build correct programs is undeveloped. We introduced an online judge system which automatically judge students' program to be correct in order that students can recognize whether their programs are correct. Some students acquire the ability to build correct programs by using the judge system. On the other hand, some students used the judge system by try and error manner so that they could acquire the expected ability. We conclude that ratio between submitted program and correct ones is important factor in order to check weather a student acquire the ability.

キーワード: プログラミング, 教育支援, デバッギングスキル
Key words: programming, education support, debugging skills

1. はじめに

プログラミング教育は, 大学でコンピュータを学ぶ学科では, 半世紀以上にわたって実施されてきており, その経験は蓄積され, 数多くの教科書が出版されてきている。学生がプログラミングを行う計算機環境は, 1980年代までの大型計算機時代と比べると, 現在では一人一台のパソコンが用意され, 作成したプログラムを瞬時にコンパイル・実行でき, 統合開発環境によってエラーの場所がすぐにわかるといった, 学生にとって大変便利なものになっている。そのような環境下にありながら, 情報系学科の導入プログラミング教育によって, 学生が基礎的なプログラミング能力を確実に身に付けているのか, という点での疑問が, 上級の専門科目教育の担当教員, 研究室の指導教員, 就職先の新人教育担当者から上がっている [1]。

研究室での研究や, 実社会でのソフトウェア開発で求められるプログラムと, 導入プログラミング教育で許容されているプログラムとでは, その正しさ, 計算やメモリの効率, 可読性, 保守性の面で大きな乖離があることがその要因として考えられる。研究室や会社の現場では, OJTにより, 教える側の人数と教わる側の人数比の条件が, 大学での導入プログラミング教育の現場よりも良い。プログラミングは, 徒弟制でしか教えられないという意見も多い。大学では, 現在よりも多くの人的リソースをプログラミング教育にかけることは難しい状況にある。そこで, 正しさや効率によってプログラミング技術を争うコンテストで利用されている, ソフトウェアでプログラムの正しさを判断するオンラインジャッジ

受付: 2010年10月18日
受理: 2010年11月24日

システムに着目した。オンラインジャッジシステムはあらかじめ正解となるデータを用意しておけば，受講生数が多くなっても，また演習授業時間外以外にも，学生のプログラムが正しいかどうか判断することができる。筆者は，専修大学ネットワーク情報学部の導入プログラミング科目でオンラインジャッジシステムを導入し，それにより，学生たちが正しいプログラムを作成する能力が身につけさせる試みを行った。

2節では，情報系学科の導入プログラミング教育の問題点を，特に筆者が実際に教育を行っている専修大学ネットワーク情報学部での事例を中心に述べ，そこからどのような教育方法の改善が必要か議論する。3節では，オンラインジャッジステムの仕組みについて紹介し，それをプログラミング教育に導入することでどのような効果が期待できそうか述べる。4節では，実際に専修大学ネットワーク情報学部のクラスでオンラインジャッジシステムを導入することで，どのような効果が得られたのか，どのような問題点があったのかについて述べる。最後に5節では，問題点を改善するためにどのような試みが可能か考察する。

2. 情報系学科の導入プログラミング教育とその問題点

2.1 導入プログラミング教育

1960年代では，情報系学科におけるプログラミング教育は，カリキュラム内で極めて重要な位置を占めていた。その後，情報系学科で教えられる学習内容は，年々多様になり，プログラミング教育にかける割合は低減傾向にあるが，それでも入門段階で学ぶべき必須知識・スキルとして考えられている。

技術者教育の専門認証機関であるJABEEでは，情報および情報関連分野の分野別要件（図1）に，修得すべき知識・能力の内の一つとして「プログラミング能力」があげている［2］。また，情報処理学会が策定した情報専門学科のカリキュラム標準J07[3]においては，その中の五つの領域（CS：コンピュータサイエンス，CE：コンピュータエンジニアリング，SE：ソフトウェアエンジニアリング，IT：情報テクノロジー，IS：情報システム）の内のISを除くすべてで，プログラミングはコア科目として指定されている。また，コア科目の考えをもたないISにおいてもモデルコースの中の科目として位置づけている。

多くの情報系学科では，専門科目履修の最初の段階である，1年次あるいは2年次に導入プログラミング教育をする科目を用意している。これは，情報系の多くの専門科目で，計算アルゴリズムを扱うこ

JABEE 2009年度分野別要件　－情報および情報関連分野－
1. 修得すべき知識・能力
　教育プログラムの修了生は，次に示す知識・能力を身に付けている必要がある。
(1) つぎの学習域すべてにわたる，理論から問題分析・設計までの基礎的な知識およびその応用能力
— アルゴリズムとデータ構造
— コンピュータシステムの構成とアーキテクチャ
— 情報ネットワーク
— ソフトウェアの設計
— プログラミング言語の諸概念
(2) プログラミング能力
(3) 離散数学および確率・統計を含めた数学の知識およびその応用能力
(4) 教育プログラムが対象とする領域に固有の知識およびその応用能力

図1 JABEE 2009年度分野別要件 — 情報および情報関連分野 — の一部［2］

とが多く，それを理解するためにプログラミングの知識が必要なことはもちろんのこと，場合によっては演習課題として実際に計算アルゴリズムをプログラムとして実現し実行して確認することが求められているからである。

情報系専門科目の基礎として重要と考えられているプログラミングの導入教育がうまく行っているのかというと疑問が多い。JABEE による認定を受けている大学では，一定のプログラミング能力を卒業時に担保しているが，卒業研究担当教員数人にインタビューしたところ，研究室に来る段階では基本的なプログラミング能力が身についていない学生が多く，研究室に入ってから再度勉強をし直さなければならないという意見が多く出てくる。研究室に入れば，具体的な研究テーマに関連したプログラミングを，先輩大学院生の指導を受けながら，OJT に近い環境で学ぶことができ，それによりプログラミング能力を獲得することができる。一方，プログラミングの導入科目では，条件が比較的恵まれている国立大学においてでも，数十人のコンピュータ演習教室で，教師に加えて数名のアシスタントがサポートして授業が行われているのが現状である。研究室で行われている木目の細かい指導とはほど遠く，教員の説明に基づいて受講学生が何らかの実践をするが，その実践に対する善し悪しの反応を教員やアシスタントがするわけではない，一方向的な授業となっている。

2.2　専修大学ネットワーク情報学部での導入プログラミング教育とその問題点

筆者が実際に導入プログラミング教育を担当している専修大学ネットワーク情報学部の状況を具体的に見てみる。専修大学ネットワーク情報学部では，1 年次後期の必修科目として，プログラミング入門（講義科目）とプログラミング演習（演習科目）が用意されている[1]。講義科目は普通の教室において実施される。授業中に問題を解かせることはあるが，その際には学生たちは紙の上でプログラムを記述する。一方，演習科目は，一人一台のパソコンが用意されているコンピュータ教室で行われ，学生たちはパソコン上にあるプログラム開発環境を使用してプログラムを作成する。

この二つの科目は，別々の科目であるが，扱う内容は密接な関係を持たせている。2009 年度の場合は，火曜日に講義科目，木曜日に演習科目が実施されており，火曜日の講義科目で扱った内容を範囲として，木曜日の演習科目での演習課題が出題される。演習科目の課題の出題は，講義科目担当者が行っており，講義で扱った内容を復習して確認し，応用する題材を取り上げるようにしている。講義科目は 150 名から 200 名程度の学生で一つのクラスを編成し，演習科目は 30 名から 50 名程度で一つのクラスを編成している[2]。演習科目のクラスには，大学院生又は学部生のアシスタントを 1〜3 名配置している。教員及びアシスタントは，10 名強の学生あたり 1 名という配置としている（図 2)。

成績評価方法は次のように行っている。講義科目は，中間テストと期末テストの 2 回のペーパーテストの合計点によって判定している。一方，演習科目は，毎回の授業ごとに課される課題によって判定している。講義科目は中間及び最終の到達度が主たる評価対象であり，演習科目は毎週のプロセスが主たる評価対象ということである。この方法だと，講義科目はテストの実施前に集中して学習してキャッチアップすることで，合格できる可能性があるが，演習科目は安定して毎回学習を積み重ねていかなければ合格できない。

講義・演習科目とも担当者が複数いるが，出題問題を共通にし，成績判定，特に合否判定に関わる点についても，担当者全員ですりあわせている。講義科目のテストの点数が不合格レベルにある学生に対しては，演習科目の合否判定のために，実際に教員の目の前でプログラミング課題を解かせる実技テス

[1] 2010 年度の講義要項（シラバス）を論文の最後の付録に載せている。
[2] 再履修者の人数，入学者の人数，コンピュータ教室の定員などの要因によって，受講生数の変動が出る。

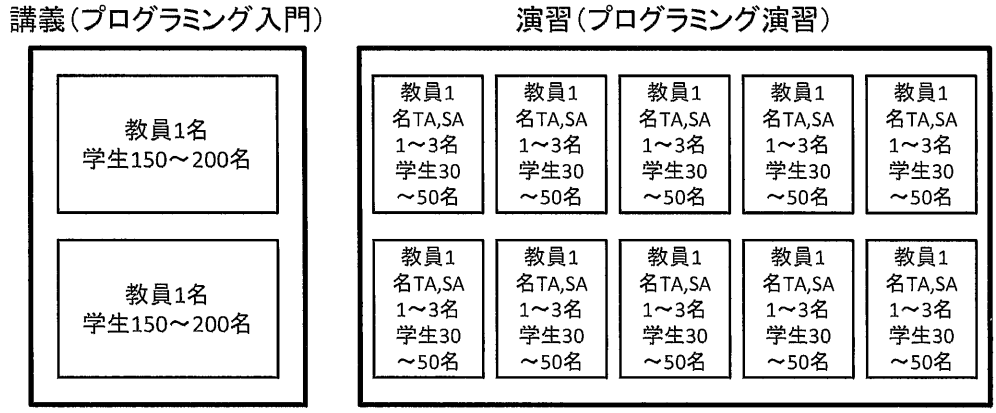

図2 専修大学ネットワーク情報学部の導入プログラミング教育のクラス編成

トを実施し，その状況を全担当教員が見て合否判断をするという方法を採用している[3]。このことにより，担当者間で導入プログラミング教育の最低到達レベルについて，すりあわせがなされていることになる。

講義科目のテスト問題も，演習科目の課題も，基本的には同じような出題方法となっている。入力値に対して，出力値がどのような関係になっているか述べられ，その計算を実行するプログラムを記述するものである (図3)。

すべてのプログラムを学生が書かなければならない場合 (スクラッチからの記述) もあるし，プログラムの多くが書かれていて一部の穴を埋める場合 (穴埋め問題) もある。講義テストのペーパーテストにおけるスクラッチからの記述を求める問題においては，基本的なアルゴリズム構造があっていない限りは点数がつかない。基本的な構造があっていれば，正解からの差異に対して減点していき，大きな誤りがない限りは 50% 未満にはならないように採点している。

例えば，条件式を記述するところで，不等号を逆にしてしまった場合，講義科目では不合格となる点

1. 二つの整数 m, n を入力として受け付ける。$\sum_{i=m}^{n} i$ すなわち $m+(m+1)+\cdots+(n-1)+n$ (m から n までの総和) を求め，その結果を出力する。ただし，$m>n$ の場合は，0 が結果となる。
2. 二つの整数 m, n を入力として受け付ける。二つの整数の内の大きい方の数を b，小さい方の数を s とする。すなわち $s+(s+1)+\cdots+(b-1)+b$ (s から b までの総和) を求め，その結果を出力する。
3. 二つの整数 m, n を入力として受け付ける。ある整数が black である条件とは「偶数かつ 0 以上　又は　奇数かつ負」である。二つの数のうち一つだけが black のときには diff と，そうでないときは same と出力する。
4. 検索対象とする整数を 1 行に一つずつ入力として受け付ける。受け付ける整数は 20 個とする。最後に検索する整数を一つ入力として受け付ける。検索対象の中に，最後に入力した整数がいくつあったか，出力しなさい。
5. 検索対象とする整数を 1 行に一つずつ入力として受け付ける。受け付ける整数は 20 個とする。最後に検索する整数を一つ入力として受け付ける。検索対象の中に，最後に入力した整数が何番目にあったか出力しなさい (何番目かは 0 から数えるものとする)。なお複数見つかった場合は，一番後ろで見つかったもの (大きな番号) を出力しなさい。もしなかった場合は -1 を出力しなさい。

図3 専修大学ネットワーク情報学部での出題問題の例

[3] 2005 年度までは，「講義科目の期末テストの点数に基づいて下位 25%の学生を，演習科目の合否判定に連動させ，演習科目を不合格とする」という方法をとっていた。2006 年度以降は，後述するように講義科目の期末テストで必要とされる能力と，演習科目で必要とされる能力の一部に差異があるため，演習の合否判断に実技テストを加えるように変更した。

数までは減点するようなことはない。一方，演習においては実際にプログラムをパソコン上で動作させるので，不等号を逆にしていれば，正しい答えが出ない，あるいはループが終了しないということがおこる。そのまま修正できないと，その1箇所のミスだけで，できなかったと判断されてしまうことになる。演習科目では，誤った実行結果を認識して，それに対して何らかの対応をしてプログラムを修正（デバッグ）することが求められる。デバッグ能力が身についているかどうかが，演習科目の達成度にとって，講義科目の場合と比べて重要ということである。

演習科目の合否判定に実技テストを実施した2006年当初は，学生たちのテストの様子を観察していると，最初に作成したプログラムを実行した結果，誤りに気がつき，適切にデバッグして正しいプログラムを完成できる学生が大半であることがわかった。これは，演習科目を通してプログラミングのスキルが身に付いている証拠と言える。一方，2008年以降は，実技テストを実施しても，正しいプログラムに修正していくことができず，不合格となる割合が，実技テスト対象者（講義科目で30点～50点程度の学生）のうち，半数を超えるような事態となっている。担当教員たちは，演習科目で身に付けていなければならない能力を，もう一度再確認し，学生たちがその能力をきちんと身に付けられるように誘導することが求められている。

2.3 導入プログラミング教育で身に付いていなければならない能力

1960年代の初期のプログラミング教育では，当時学生だった方々にインタビューしたところでは，講義内では，プログラミング言語の文法のみが教えられ，講義時間外に行う演習課題が与えられ，自分たちでプログラミング技術を学ばなければならなかった。その後，講義内容は工夫されたものの，入門テキストの構成は，文法事項に沿って展開され，その文法を活用したプログラムの例題が示されるという形式が一般的である。1980年代までの大型計算機中心のコンピュータ環境では，演習は，授業時間外に自主的に行うように，というやり方をとらざるを得なかった。1990年代以降，一人一台のパソコンを用意できる演習教室が整備されてきて，演習授業時間内にプログラミング作業をすることができるようになった。

そういった演習環境の変化に対応して，さらにプログラミング能力を高める講義・演習内容になっているのか考えると，十分ではないと言える。1960年代のように，プログラミング言語の文法さえ示せばプログラミングができると考える教師は，現在ではほとんどいないであろうが，アルゴリズムをきちんと考えつけばプログラミングができる，あるいは類題を与えてそれを修正するような形の演習を与えればプログラミングができる，と考える教師は多いであろう。実際に，入門的なプログラミングの教科書を見れば，そのような考えに基づいて構成されている。

アルゴリズムが考えつく，あるいは，元となる類題を修正する考えが思いついたとしても，それはプログラムの最初の版を作るところまではできる。多くの学生にとって，最初の版のプログラムは完璧なものではなく，修正が必要な場合が多い（図4）。これは文法エラーがあるというだけではなく，論理的にも誤りが含まれているということである。

上述したプログラミング演習の実技テストでの状況の観察でわかるように，論理エラーを見つけてそれを修正する能力も，導入的プログラミングの演習科目では求められる。具体的には，論理エラーを発見するために必要な入力テストデータを用意し，そのデータを入力したときの出力結果を想定する結果と比較し，どの入力データのときに誤りがあるか見つけ，その入力データを処理するプログラムルーチンを検証していき誤りを見つけていく，という繰り返しプロセスとなる。このプロセスは，より高度なプログラミングをするにつれ，ますます重要となってくるし，社会で利用される実システムの開発では

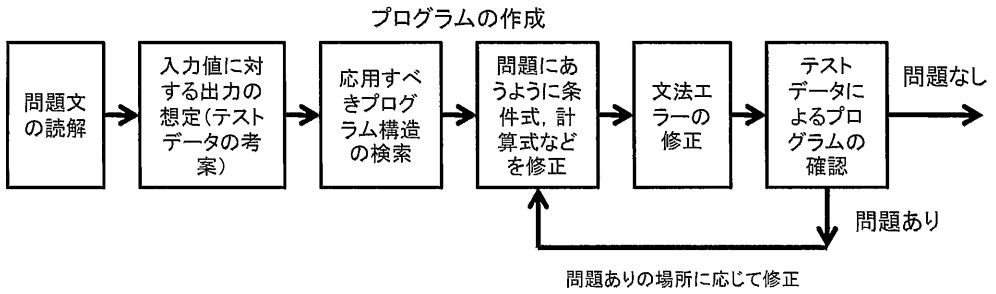

図4 プログラミング課題を遂行するために必要なプロセス

必須のものである。プログラム構造が複雑になるにつれ，論理的な誤りを見つけ修正することはより困難になるので，導入的なプログラミング科目を受講している段階でこの能力の基本を身に付けることが重要である。

2.4 導入プログラミング教育で求められる能力を身に付けさせる方法

論理的な誤りを，テストデータを用いて確認していくことは，個人のプログラミング作業においては面倒に感じるものである。プログラミング演習では，演習の提出前に，少なくとも3つの入力データを検証した結果をつけて提出するように求めている。そのように強制したとしても，自分の都合の良いように入力データを選ぶ学生が後を絶たない。

Kent Beck 氏が主張する eXtreme Programming (XP) [4] の考えは，そのような状況に対応する良い方法である。XP では，二人がペアとなり，一人がテストを作成し，もう一人がプログラミングを担当するテストファーストのペアプログラミングが重要な方法として提案されている。この方法を，導入的なプログラミング演習の授業に取り入れることは可能であろう。しかし，実際に取り入れるとなると，ペアとなる学生のパフォーマンスがあっていないと，パフォーマンスの高い学生の時間が余ってしまうことが想定される。XP は，基本的には実開発現場の手法であり，中級以上のプログラミング演習科目に持ち込むことは可能かもしれないが，導入的な演習科目で実践することは難しい。

大学での導入プログラミング教育に，1995 年以来関わっている筆者は，演習授業時間内に様々な教育方法を試してきた。これまでの様々な方法において最も良いパフォーマンスを得られる方法は，演習授業時間に，学生が作成したプログラムを見て，学生に対して直接コメントし，必要な改善を促すことである。筆者が演習授業時間内にチェックしている項目は，次の4つに整理できる。

1. 正しく動作するプログラムかどうか
2. 正しく動作しているかどうか調べるテストデータが適切かどうか
3. 正しいとしても拡張や修正が容易なようにプログラムが作られているか
4. プログラムのスタイル（字下げ，名前の付け方，など）が適切で，プログラムが読みやすいかどうか

3 の事例としては，データ数の変化に対して強いかどうかあげることができる。10 個の数の平均を求める問題に対して，変数を 10 個用意することは誤りではない。しかし，個数が多くなれば，それだけの変数を用意することは事実上できないので，繰返しや配列を使わなければならない。繰返しを使用していないプログラムが提出されてきた場合，チェックして，なぜそれが適当ではないのか学生に説明して，納得してもらい対応を促すことができる。

人間がチェックすれば，同じような誤りを繰り返す学生に対しては，減点等の処置をすることができ

るので，学生たちは改善しなければならないというプレッシャーを受けて行動するようになる。そのことにより，より良いプログラムを作成するための行動が習慣化されやすい。

ネットワーク情報学部のプログラミング演習では，3〜4 題の問題が演習授業冒頭で出題され，90 分間の演習授業内に，少なくとも 2 題に対してプログラムを作成し，そのプログラムのソースコードと，3 つ以上のテストデータを付した実行結果を提出することを学生に求めている。90 分間に，全受講生に対して 2 題のプログラムを教員がチェックするためには，次の条件が揃っていることが経験上わかっている。

- 受講生の人数。一つのプログラムは 10 行から 30 行程度なので，そのチェックは，平均すれば 1 分もかからない程度であり，延べ 90 問のチェックは可能である。学生が提出する時間帯はばらついている訳ではないので，これまでの経験では，20 人程度の受講生数を超えるようになると，教員の前に待ち行列ができてしまうことになる。20 人の受講生を超えるとなると，チェックを担当できるティーチングアシスタントが別途必要になる。
- 受講生の質。出題者担当者は，演習の 2 日前に行っている講義の内容を理解していれば，演習授業内に 2 問は解けることを想定して問題を作成している。実際には，2 問目を解けない学生が，平均すると 3 割程度はいるし，1 問も解けない学生も 1 割程度はいる[4]。学生がプログラムを作成し，教員にチェックを求めてきて初めて指導ができるので，指導を演習授業でできない学生は，自分のスキルを改善する機会がなくなり，単位取得さえ困難になってしまう。

学生の受講人数のスケーラビリティを担保し（受講人数が増えても対応可能であり），演習授業時間外に課題プログラムを作成する場合でも一定の指導をできるための人的リソースを永続的に用意することは，コスト面からだけでも困難なことである。

そこで，情報技術を活用して教員がアドバイスすることを補えないか，という発想のものに，演習授業で試したみたものが，本論文で述べる筆者の試みである。プログラムの正しさを自動判定するオンラインジャッジシステムを利用することにした。今回利用したオンラインジャッジシステムでは，直接的にはアドバイスの 1（正しく動作するかどうか）のみをチェックするものであるが，誤った場合には，学生は 2（正しく動作しているかどうか調べるテストデータが適切かどうか）について自らチェックせざるを得なくなる。またコンピュータがジャッジするため，テストデータにおけるデータ数はかなり多くのものを試せるので，3 の一部について（データ数の変化に対する対応の容易性）は学生が意識せざるを得なくなる。図 3 に示した問題の 4, 5 においては，問題記述上は，データ数が 10 個になっているが，オンラインジャッジシステムに提出するときには，データ数が 1,000 個の場合に動作するように直して提出させている。

3. プログラムの正誤を判定するオンラインジャッジシステム

3.1 オンラインジャッジシステムの仕組み

プログラムの正確性を競うプログラミングコンテストにおいて，正誤の自動判定用システム（オンラインジャッジシステム）が使用されている。このような性質のプログラミングコンテストとして歴史が

[4] 出題者の想定しない困難に学生が直面してしまうことがよくあり，多い場合には 8 割程度の学生が，2 問目が演習授業時間内に終了しない，ということもある。

長いものとして，ACM が主催する ICPC[5] (International Collegiate Programming Contest) がある。ICPC で使用されているオンラインジャッジシステムとして California State Univ. の Sacramento 校で開発されている PC^2[6] がある。

オンラインジャッジシステムは，ネットワーク上のサーバプログラムとして用意され，クライアント (コンテスト参加者) 側から送られてきたプログラムに対して正誤判定を行う。正誤判定のために，審判は，入力データと，そのデータに対する正解出力を用意し，システム上にファイルとして置いておく。コンテスト参加者はプログラムを完成させると，ソースコードをシステムに提出する。システムはそのプログラムをコンパイルし，ジャッジデータを入力として実行した出力を，正解データと比較することで正解かどうか判定する (図 5)。

出力が正しくない場合に加えて，コンパイル時のエラー，実行時のエラー，所定時間内に計算が完了しない場合 (無限ループや選んでいるアルゴリズムの効率が悪い場合), 必要以上にメモリ消費量が多い場合も，システムは誤りとして報告する (表 1)。コンテストを目的としているため，エラーの種類は報告されるものの，どこに誤りがあるのか，あるいはどのような入力データに対して間違えたのかは報告されない。

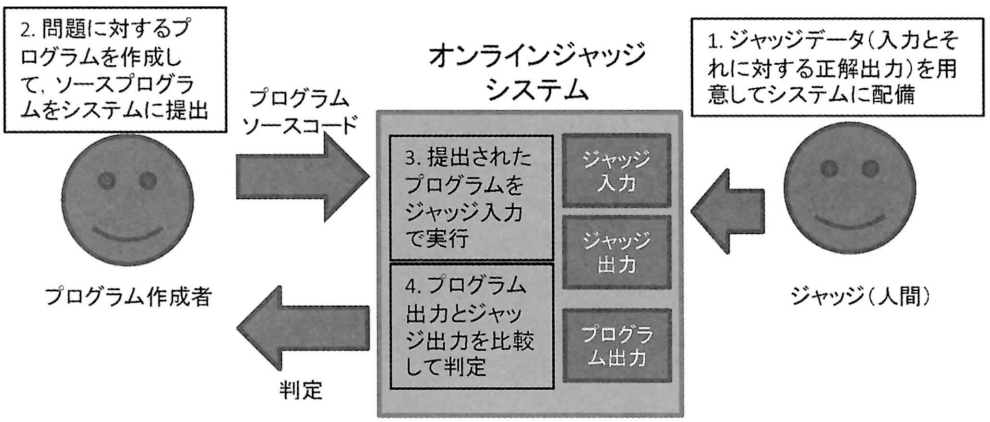

図 5 オンラインジャッジシステムの仕組み

表 1 オンラインジャッジシステムの判定

Accepted	正解
Presentation Error	答えとしてはあっているが，出力書式がジャッジ出力とあっていない
Wrong Answer	誤り。出力がジャッジ出力とあっていない
Runtime Error	実行途中でプログラムが失敗する
Time Limit Exceeded	ジャッジが指定した時間以上に実行時間がかかった
Memory Limit Exceeded	ジャッジが指定したメモリ使用量以上にメモリを使用した
Output Limit Exceeded	プログラムが出力する文字量が，ジャッジ出力の 2 倍を超えた
Compile Error	コンパイラがエラーを出してコンパイルできない

[5] http://cm.baylor.edu/welcome.icpc
[6] http://pc2.ecs.csus.edu/

5 500	nが大きく両方正
400 400	二値が等しく両方正
400 399	mが大きく両方正
−5 −500	mが大きく両方負
−2 −2	二値が等しく両方負
−500 −5	nが大きい両方負
−5,000 5,001	nが大きく負と正

図6　入力データの例（値とその意味）

プログラミングコンテスト本番用のPC^2に対して，過去のコンテストの問題をアーカイブし，練習用にオンラインジャッジをするシステムがいくつか開発されている。北京大学が開発したPOJ[7] (PKU Online Judge) はその一つであり，そのフリーバージョンは教育機関に無償で供与されている。筆者は，プログラミング演習の授業で利用したものは，このPOJフリーバージョンである。POJフリーバージョンでは，入力データと出力結果の組（ケース）を複数用意することができる。

ジャッジにとってはどのようなジャッジ入力データを用意するのかが重要で，その問題における様々なパラメータの境界値（特に上限値）を試すものを含めることが必要である。

図6は，図3に示した問題1, 2に対するジャッジ入力データである。この二つの問題では，$m>n$, $m=n$, $m<n$の三通りが用意されていれば判定可能であるが，冗長性を含めて7通り用意してある。

また図3の問題3では，一つの値に対して，

1. 正の奇数
2. 正の偶数
3. ゼロ
4. 負の奇数
5. 負の偶数

の5通りに場合分けできる。二つの値があるので，すべての場合分けに対応したジャッジデータを用意するためには，25 ($=5^2$) 通りが必要となる。

3.2　オンラインジャッジシステムの教育上の想定される効用

ICPCのようなプログラミングコンテストの学生参加者が，プログラミングの演習授業でプログラム開発をしている学生と比べて，どのような点に注意深くなっているのか確認することで，オンラインジャッジシステムをプログラミング演習に導入する際の効用をあらかじめ推測できると考えられる。プログラミングコンテストに参加するような学生たちは，導入プログラミング教育で課されるようなレベルの問題を間違えることはないが，問題レベルが上がると急に，導入レベルの学生と同じような種類の誤りをするようになる[5]。そのことから，コンテスト参加者の振る舞いからでも，導入レベルの学生の振る舞いも推測可能と考えられる。

1. 問題文を読み直して，必要な条件を読み落としていないか，あるいは条件を勘違いしていないか確認できるようになる。特に，条件の境界に対する勘違いがないか確認できるようになる（例えば，不等号に関して等号付きかどうか)。
2. 問題文の理解が正しかったとして，分岐条件が正しく書けているかどうか調べられるようにな

[7] http://poj.org/

る。ソースコードを字面でチェックすることも，テストデータを用意してチェックすることもできるようになる。

3. 分岐条件が正しく書けたとして，分岐毎の計算処理が正しいかどうか調べられるようになる。ソースコードを字面でチェックすることも，テストデータを用意してチェックすることもできるようになる。
4. 変数の上限値になったときに，用いているアルゴリズムやデータ構造が，計算量あるいは使用メモリ量の点で実行可能な範囲にあるかチェックできるようになる。

4. オンラインジャッジシステムの導入プログラミング教育での実践とその分析

オンラインジャッジシステムを利用する演習授業を，2010年度前期に開講したプログラミング演習の再履修クラスで導入した。この再履修クラスは，前年度（2009年度）後期には講義（プログラミング入門）の単位は修得したものの，プログラミング演習の単位を修得できなかった学生が受講している，特殊な状況のクラスである。このクラスの受講生数は履修名簿上43名と，2.4節で示したような人間によるチェックをするには多すぎる。アシスタントの学部学生が2名ついているが，彼らにプログラムをチェックさせるには荷が重く，コンパイルエラーが発生したときなどの基本的なトラブルのみ対応してもらうことにした。

受講生たちは講義の単位を修得していることから，一定のプログラミング知識を有していることが期待できる。ただし，講義の点数は，合格点をぎりぎり超えた程度であり，持っている知識が不正確な部分が多くあることが予想される。彼らにとって，このクラスで学ぶべきことは，より正確なプログラミングの能力を徐々に獲得していくことであり，オンラインジャッジシステムを導入する効用は高いと考えた。

図7は，ある学生が誤りを2回した後に正解を得た問題に対する提出プログラムの修正履歴である。

1. 最初の修正では，データの個数を数える変数 c を，値ゼロに初期化していなかったことを修正している。C言語のauto変数の初期値は，処理系に依存しており，この学生が手元で実行した処理系はゼロに初期化されていた可能性がある。その場合にはテストデータを試してもこの誤りには気がつかないので，ソースコードをチェックして発見したことが推測される。
2. 2回目の修正は，問題文が「正の値（すなわちゼロより大きい）」であったのに，実際のソースコードの条件式が「ゼロ以上」となっていたことに気がついて修正している。

この程度の誤りは，オンラインジャッジシステムや教員が指摘しなければ，学生本人は気がつかないままであったであろう。学生に誤りを気がつかせ，正しいプログラムとなるまで修正するように導いたということは，オンラインジャッジシステムの効用があったと判断できる。

この演習科目では，8回の演習授業を経過した後，オンラインジャッジシステムの履歴から一定数以上の問題を正解し，十分に正確なプログラムを作れる能力があると筆者が判断した学生から実技テストを行い，それにパスすると合格するという単位認定方法を使用した。実技テストは，問題文と，それに対する誤ったプログラムを渡し，プログラムを問題文に記述されている結果が得られるよう修正してもらうテスト形式とした。これまで述べてきたような能力が身についたかどうか判断するには，適当な出題形式と考えた。

図8は，最も早く合格した学生の一人の，オンラインジャッジシステムの履歴を示したものである(下が古いものになっている)。この学生の場合，受講の最初の頃は誤りが多く，左から2列目のResult

100万個の整数を読み込み，そのうち正の数の総和と総数を求める問題。

(a) 第1回目に提出されたプログラム (不正解)

```
# include <stdio.h>
int main (void){
        int n,x,c,sum=0;
        for(x=0;x<1000000;x++){
            scanf("%d",&n);
            if(n>=0){
                sum=sum+n;
                c=c+1;
            }
        }
        printf("%d",sum);
        printf(" %d¥n",c);
        return (0);
}
```

(b) 第2回目に提出されたプログラム (不正解)

```
# include<stdio.h>
int main (void){
        int n,x,c=0,sum=0; /*筆者注：cの値をゼロに初期化した*/
        for(x=0;x<1000000;x++){
            scanf("%d",&n);
            if(n>=0){
                sum=sum+n;
                c=c+1;
            }
        }
        printf("%d",sum);
        printf(" %d¥n",c);
        return (0);
}
```

(c) 第3回目に提出されたプログラム (正解)

```
# include<stdio.h>
int main (void){
        int n,x,c=0,sum=0;
        for(x=0;x<1000000;x++){
            scanf("%d",&n);
            if(n>0) {/*筆者注：比較の不等号が，等号付きではなくなった*/
                sum=sum+n;
                c=c+1;
            }
        }
        printf("%d",sum);
        printf(" %d¥n",c);
        return (0);
}
```

図7 ある学生の誤りと判定されたプログラムの修正履歴

Problem	Result	Memory	Time	Language	Code Length	Submit Time
1180	Accepted	92K	0MS	GCC	0.25K	2010-06-10 17:40:49
1179	Accepted	92K	0MS	GCC	0.19K	2010-06-10 17:03:20
1179	Wrong Answer			GCC	0.18K	2010-06-10 17:03:00
1178	Accepted	92K	0MS	GCC	0.2K	2010-06-10 16:59:10
1177	Accepted	92K	30MS	GCC	0.14K	2010-06-10 16:47:39
1176	Accepted	92K	0MS	GCC	0.19K	2010-06-10 16:44:13
1168	Accepted	108K	717MS	GCC	0.32K	2010-06-03 17:54:19
1168	Wrong Answer			GCC	0.18K	2010-06-03 17:37:52
1167	Accepted	112K	15MS	GCC	0.27K	2010-06-03 17:22:10
1166	Accepted	108K	15MS	GCC	0.19K	2010-06-03 16:56:09
1165	Accepted	108K	0MS	GCC	0.18K	2010-06-03 16:53:08
1164	Accepted	108K	0MS	GCC	0.2K	2010-06-03 16:47:17
1146	Accepted	92K	0MS	GCC	0.36K	2010-05-20 17:55:22
1146	Wrong Answer			GCC	0.33K	2010-05-20 17:46:45
1146	Wrong Answer			GCC	0.33K	2010-05-20 17:45:17
1144	Accepted	92K	0MS	GCC	0.18K	2010-05-20 17:33:48
1144	Wrong Answer			GCC	0.18K	2010-05-20 17:30:08
1144	Wrong Answer			GCC	0.18K	2010-05-20 17:28:24
1144	Wrong Answer			GCC	0.25K	2010-05-20 17:26:34
1144	Wrong Answer			GCC	0.25K	2010-05-20 17:21:31
1144	Wrong Answer			GCC	0.25K	2010-05-20 17:20:06
1142	Accepted	92K	15MS	GCC	0.22K	2010-05-20 17:12:46
1141	Accepted	92K	0MS	GCC	0.22K	2010-05-20 17:12:17
1143	Accepted	92K	0MS	GCC	0.23K	2010-05-20 17:11:33
1142	Wrong Answer			GCC	0.21K	2010-05-20 17:04:36
1142	Wrong Answer			GCC	0.21K	2010-05-20 16:59:14
1141	Wrong Answer			GCC	0.21K	2010-05-20 16:52:48
1141	Wrong Answer			GCC	0.21K	2010-05-20 16:50:58
1134	Accepted	92K	859MS	GCC	0.2K	2010-05-13 17:21:23
1133	Accepted	92K	875MS	GCC	0.2K	2010-05-13 17:13:53
1133	Wrong Answer			GCC	0.17K	2010-05-13 17:10:33
1132	Accepted	92K	843MS	GCC	0.16K	2010-05-13 17:07:48
1131	Accepted	92K	0MS	GCC	0.2K	2010-05-13 17:03:52
1130	Accepted	92K	15MS	GCC	0.13K	2010-05-13 17:03:29
1131	Wrong Answer			GCC	0.2K	2010-05-13 17:02:03
1130	Wrong Answer			GCC	0.13K	2010-05-13 16:53:34
1130	Wrong Answer			GCC	0.13K	2010-05-13 16:47:21
1128	Compile Error			GCC	0.25K	2010-05-06 18:18:12
1127	Accepted	92K	0MS	GCC	0.27K	2010-05-06 17:56:07
1124	Accepted	92K	0MS	GCC	0.21K	2010-05-06 17:55:26
1124	Wrong Answer			GCC	0.21K	2010-05-06 17:44:37
1126	Accepted	92K	15MS	GCC	0.17K	2010-05-06 17:40:55
1125	Accepted	92K	0MS	GCC	0.16K	2010-05-06 17:36:23
1124	Wrong Answer			GCC	0.17K	2010-05-06 17:21:01
1117	Wrong Answer			GCC	0.18K	2010-04-22 18:03:08
1117	Wrong Answer			GCC	0.17K	2010-04-22 18:02:04
1117	Wrong Answer			GCC	0.18K	2010-04-22 18:01:32
1123	Accepted	92K	0MS	GCC	0.19K	2010-04-22 17:52:30
1123	Wrong Answer			GCC	0.18K	2010-04-22 17:49:57
1123	Wrong Answer			GCC	0.16K	2010-04-22 17:43:57
1123	Compile Error			GCC	0.16K	2010-04-22 17:38:57
1123	Compile Error			GCC	0.17K	2010-04-22 17:35:26
1123	Compile Error			GCC	0.17K	2010-04-22 17:31:31
1123	Compile Error			GCC	0.18K	2010-04-22 17:30:56
1122	Accepted	92K	0MS	GCC	0.23K	2010-04-22 17:27:44
1122	Wrong Answer			GCC	0.23K	2010-04-22 17:26:29
1120	Accepted	92K	15MS	GCC	0.16K	2010-04-22 17:24:11
1120	Wrong Answer			GCC	0.17K	2010-04-22 17:23:34
1120	Wrong Answer			GCC	0.16K	2010-04-22 17:22:51
1120	Wrong Answer			GCC	0.16K	2010-04-22 17:22:23
1120	Wrong Answer			GCC	0.16K	2010-04-22 17:21:24
1121	Accepted	92K	0MS	GCC	0.2K	2010-04-22 17:18:51
1121	Presentation Error			GCC	0.23K	2010-04-22 17:17:52
1121	Wrong Answer			GCC	0.27K	2010-04-22 17:16:04
1121	Wrong Answer			GCC	0.28K	2010-04-22 17:04:18
1121	Wrong Answer			GCC	0.27K	2010-04-22 17:02:10
1120	Wrong Answer			GCC	0.23K	2010-04-22 16:57:30
1117	Wrong Answer			GCC	0.25K	2010-04-15 18:07:41
1118	Compile Error			GCC	0.24K	2010-04-15 17:59:27
1117	Wrong Answer			GCC	0.25K	2010-04-15 17:56:32
1118	Compile Error			GCC	0.24K	2010-04-15 17:50:09

図8　ある学生のオンラインジャッジステムの判定履歴

の欄が Accepted である「正解」を得るまでに多くの提出回数を要している。しかし，最後の方になると，1 回目の提出で正解を得るか，又は誤りだとしても 1 回の修正で済むようになっている。この学生は，正しいプログラムを書くための方法を会得したと見ることができるし，実際に実技テストも 1 回で通過している。

このようにオンラインジャッジシステムを活用することで，プログラミング能力を高めた学生がいる一方，実際には，実技テストで不合格となる学生も，実技テスト対象者 34 名中，半数弱の 16 名に及ぶこととなった[8]。この原因について検討してみる。

図 9 に示すグラフは，実技テストの合格者(縦軸 0 が最終回より前の実技テストに合格した学生，1 が最終回の実技テストに合格した学生[9])と不合格者（縦軸 2）ごとに，オンラインジャッジの正答率をプロットしたものである。合格，不合格別に見た場合，次のことがわかる。

- 早期に合格した学生（縦軸 2）は，1 名を除いては，正答率が 35% を超えている。3 回に 1 回以上の割合で正解を得ている。
- 実技テストの不合格者 16 名（縦軸 0）のうち，30% 以上の正答率の学生は 4 名で，10% 台が 4 名，20% 台が 8 名を占める。
- 最後に合格した学生 11 名（縦軸 1）は，40% 以上の正答率の 4 名と，30% 未満の正答率の 7 名の二つのグループに分かれる。

正答率が 40% 以上の学生では 8 名中 7 名が合格するのに対して，30% 未満の学生では 20 名中 8 名しか合格しておらず，合格者のうち 7 名が最終回に合格している。正答率が 30% 台だと，5 人に 2 人が合格しているので，30% 未満の正答率の学生と合格率は同じであるが，合格者は早期にパスしている。ある数以上の問題数を解いたときに実技テストを受けられるようにしている。早期合格者が正解を得た問題数の平均は 35.1 題，最終回での合格者の平均が 33.9 題，不合格者の平均が 32.8 題と，ほとんど差は見られない。解いた問題数が合否に大きな影響を及ぼしているのではなく，正答率が合否と強い関係をもっ

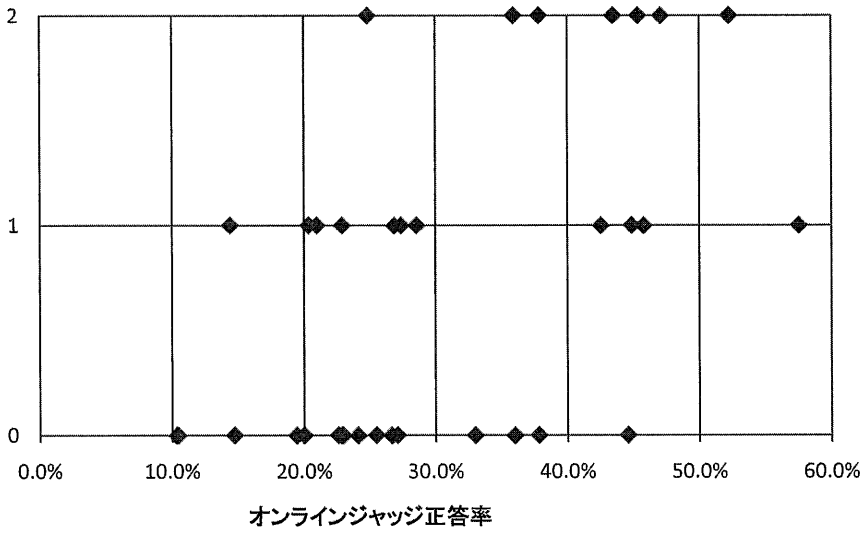

図 9　最終実技テストの合否ごとの，オンラインジャッジ正答率

[8] 実技テスト不合格者の中で，提出されたプログラムの系列を見ていき，正解に近づいていた場合には，科目としては合格にしている。

[9] 実技テストの最終回は，2 回目の受験者が含まれ，誤りの箇所も 3 箇所とそれ以前の実技テストよりも少ないため区別した。

ていることがわかる。

5. 考察と今後の改善

5.1 考察

オンラインジャッジシステムは，正しいプログラムを書くことを学生たちに強いるシステムであるが，実技テストの結果から見ると，正しいプログラムを論理的に導くことができる能力を確実に身に付けさせる道具ではないことがわかった。システムでの正答率が実技テストの合否に大きく関連していることから，次のことが考えられる。

実技テストの内容は，正しいプログラムに対して，条件式，初期化，変数の入れ替え，計算式の誤り，計算式の順序の入れ替え，などの誤りを 3〜5 個程度入れたものを渡し，それを正しいプログラムになるように，コンピュータを目の間にしながら修正していくものである。

実際の演習での学生の様子を観察していると，一部の学生は，トライアンドエラーの手法によりプログラムを修正している。彼らは，これまでの経験上，修正方法のパターンをいくつか持っており，その修正方法を闇雲に適用することはできる。今回の実技テストの問題では，誤りの個数から考えて，根拠が明白でないアドホックな修正を加えていっても，正解までたどり着けない可能性が高い。

オンラインジャッジシステムは，誤りのプログラムを指摘する道具である一方，学生にとって確信が持てないプログラムに対して，正解であればお墨付きを与えてくれる道具でもある。トライアンドエラーにより，作成したプログラムのお墨付きをもらうという姿勢でオンラインジャッジシステムを使った場合，今回の演習授業で与えた問題セットでは正答率が 30%〜40% 以下になるということが考えられる。

5.2 今後の改善に向けて

学生たちにひたすら問題を解かせてもプログラミング能力が身についていない，という導入プログラミング演習への問題点から，オンラインジャッジシステムを使って改善しようという試みをしたが，結果としては，ただオンラインジャッジシステムを使用させるだけでは，それを活用してプログラミング能力を自ら向上させられる学生は一部にとどまることがわかった。そこで 2010 年度後期にスタートしたプログラミング演習のクラスでは，いくつかの指導を加える試みをしている。紙と鉛筆を使うことは大型計算機時代には当然だった行為が，トライアンドエラーの行動を予防するものとして取り入れている。コンピュータ教室で目の前にパソコンがある環境下でプログラミングをさせると，ほとんどの学生は，紙と鉛筆をまったく使わずパソコンだけを使って作業をする。具体的には，紙を使ってテストデータ，期待される結果，テストデータに対する実行結果を記録することを促している。このことによって，プログラムの作成・修正の前に，論理的に正しいのかどうか一歩詰めて考えるようになることを期待している。

今回は，既存のオンラインジャッジシステムをそのまま利用したが，これはコンテスト用に作成したものであり，どのジャッジデータに対して誤りがあったかわからない。このことが，トライアンドエラーの修正方法を誘発している可能性がある。誤りと判定する根拠になったデータ値，あるいはそのデータ値の意味（例えば，二つの数が同じ値）を教示するモードを持ったジャッジシステムであれば，より自分の誤りに気がつきやすくなり，誤ったプログラムに対して，論理的に詰めて考察した上で修正を施すことを促すことが期待できる。

参考文献

[1] 独立行政法人情報処理推進機構 IT 人材育成本部編：IT 人材白書 2010.

[2] 日本技術者教育認定機構：日本技術者教育認定基準 2009 年度適用.

[3] 情報処理学会情報処理教育委員会 J07 プロジェクト連絡委員会：情報専門学科におけるカリキュラム標準 J07, 2009.

[4] Beck, K., Extreme Programming Explained: Embrace Change, Addison-Wesley, 1999.

[5] 松永賢次：アルゴリズム構造の理解を促すプログラミング教育の提案，情報科学研究，No. 30（2010），pp. 59-78.

付録

プログラミング入門の講義要項

講義内容	＜科目の概要＞ 本講義の目的は，(1)手順的な自動処理（課題を分析し，系統的に解決策を考え，コンピュータに実行可能な形で明示的に表現し，実行結果を検討し必要なら反復改良するプロセス）を理論的かつ体験的に理解してもらうこと，(2)ネットワーク情報学部の他の講義においてプログラミングをするときの基礎を与えることにある。 具体的には，(1)プログラミング言語の文法と動作モデルに関する知識，(2)プログラミング言語を使って問題を定式化するための技術，(3)意図した通りの結果となっているかテストし誤っている場合に修正する（デバッグ）ための技術，を扱う。 本講義では，プログラミング言語として ANSI の規格に準拠した C 言語を使う。授業で扱う範囲は，他のプログラミング言語でも共通する内容を中心とする（ポインターは扱わない）。本講義は，「プログラミング演習」で実際に計算機上の処理系を使うことでより深い理解を目指していく。 ＜学習・教育目標＞ 1. プログラミング言語の文法を理解し，作成したプログラムがどのような構造をもち，どのように実行されるのか正確にわかること。 2. プログラミングのテスト方法及びテスト結果からプログラムを修正する方法を理解し実践できること。 3. プログラムで典型的に利用されるアルゴリズムを理解し，そのアルゴリズムを利用すべき問題に対してそれを具体的に適用できること。 4. 関数を利用したり自分で定義するプログラミング方法を理解し実践し，将来，構造化プログラミング及びオブジェクト指向プログラミングを理解する土台を作ること。 ＜授業計画＞ 1　導入（プログラムとは，プログラミングとは） 2　プログラミング言語の基本（型，変数，式，文，評価など） 3　入出力，条件分岐 4　配列，繰り返し 5　これまでの復習，基本アルゴリズム 6　中間テスト 7　実数型，ライブラリ関数を利用するプログラミング 8　2 重ループ 9　二次元配列，ユーザ定義関数 10　配列を使用するユーザ定義関数，ファイル入出力 11　文字列（文字型と文字配列） 12　期末テスト 13　レビュー，2 年時以降の学習に向けて

	＜テキスト・参考文献＞ 特定のテキストに基づいて講義を行うことはないが，文法や関数のレファレンスとして，あるいは自習の目的のために，何か1冊，書籍を購入しておくことが必要である。 例えば， 柴田望洋『新版 明解C言語 入門編』ソフトバンククリエイティブ
成績評価の方法	中間テスト(50%)，期末テスト(50%) ただしいずれかのテストの点数が極端に悪い場合は，合計点が50点に達していても不合格とすることがある。
履修上の留意点	アルゴリズムに関する予備知識がない学生は，前期に「アルゴリズム的思考法」を履修することを推奨する。「コンピュータとネットワーク」，「数理リテラシー」で学習する内容の一部が，プログラミングの理解には必要となる。 本講義は「プログラミング演習」と一体となって進められる。
その他	プログラミングに対して，多くの最初は困難を感じるが，実際に自分の意図通りのプログラムができたときの喜びは格別なものがあり，それにより高い学習意欲を得ることができる。 時間をかけて自ら考え実践すれば，本学部の学生誰もが修得できる範囲を扱うようにしている。本授業内で用意する演習時間及び「プログラミング演習」の授業に集中して取り組むとともに，授業時間外にも自習する時間を用意すること。不明な点があれば，積極的に質問し，速やかに解決すること。 処理系は，大学で貸し出すことができるので，自宅でも学習できる。

プログラミング演習の講義要項

講義内容	＜科目の概要＞ 本講義は，プログラミング入門と対になった演習科目である。プログラミング入門で学習した知識に関連する具体的な課題が示され，それを計算機上でプログラミングし実行し，正しいプログラムが完成したら提出する。 プログラミング入門における理論的な説明と，プログラミング演習での実習を組み合わせることによって，プログラミングをより深く理解してもらうことが，2つの講義の目的である。 プログラミングに関しても，「習うより慣れろ」という外国語習得の一般原則が当てはまる。自力でプログラム構造を考え，プログラムを入力し，実行し，誤りを修正する作業を体験することなしに，プログラミング能力を会得することはできない。プログラミング入門に対するプログラミング演習の意義は，まさにこの点にあるといえる。 ＜学習・教育目標＞ プログラミング入門の学習・教育目標と同じ。特に，以下の点が重要である。 (1) 自分にも他人にも理解しやすいプログラムを記述できる。 (2) 正しくプログラムが作成できているか確認する手法を身につける。 (3) プログラムが誤っている場合に，誤りを発見し，修正する方法を身につける。 ＜授業計画＞ プログラミング入門と連動して進めて行くが，次のような内容が含まれる。(順番が入れ替わる可能性がある) 1 処理系の利用方法 2 プログラミング言語の基本 (型，変数，式，文，評価など) 3 入出力，条件分岐 4 繰り返し 5 配列 6 これまでの学習した内容の総合課題，デバッグ技術 7 実数型，ライブラリ関数を利用するプログラミング 8 2重ループ，2次元配列

	9　ユーザ定義関数を利用したプログラミング(1) 10　ユーザ定義関数を利用したプログラミング(2) 11　文字列，文字型 12　これまでの学習した内容の総合課題，デバッグ技術 13　実技テスト <テキスト・参考文献> 「プログラミング入門」の講義資料 さらに，文法や関数のレファレンスとして，あるいは自習の目的のために，何か1冊，書籍を購入しておくことが必要である。 例えば， 柴田望洋『新版 明解C言語 入門編』ソフトバンククリエイティブ
成績評価の方法	実習科目なので期末試験ではなく，毎授業で出題する課題に対する提出物の内容に対して評価する。正しく適切なアルゴリズムを考案しているか，考案したアルゴリズムを適切なコーディング技法を用いてプログラムにしているか，適切な入力データを使ってテストしているか，以上のことを適切に説明できているか，などの観点で評価する。毎回の講義を10点満点で評価し，それを合計したものを100点満点に換算する。 　また「プログラミング入門」のテストの点数が著しく悪い場合，理解しているかどうかチェックするため，実技テストを課して合否を判断する。 　3回を超えて欠席した場合には，いかなる欠席理由であろうと単位をつけない。他の学生の作成したものを単に写して提出したことが判明した場合には単位をつけない。
履修上の留意点	アルゴリズムに関する予備知識がない学生は，前期に「アルゴリズム的思考法」を履修することを推奨する。 本講義は「プログラミング入門」と一体となって進められる。 具体的な諸注意は，1回目の授業のときに説明する。 1)　すべての授業に出席することが原則である。 2)　提出が求められているすべての課題を提出することが必要である。
その他	プログラミング演習の授業時間は，決して長くはないので，あらかじめプログラミング入門の授業内容を復習してから授業に臨むこと。積み重ね式の授業構成になっているので，それまでの課題を自ら取り組み理解していかないと，次の課題に取り組めなくなるので注意すること。 　習熟度別クラスを編成する予定である。クラス編成の方法は，1回目の授業のときに説明する。