令和 4 年度

修士学位論文

フローチャートおよび構造化チャートを用いたプログラミング授業支援システムの構築

Development of a Programming Education Support System Using Flowchart and Structured Chart

1255107　　獅々堀 達哉

指導教員　　妻鳥 貴彦

2023 年 2 月 28 日

高知工科大学大学院 工学研究科 基盤工学専攻

情報学コース

要 旨

フローチャートおよび構造化チャートを用いたプログラミング授業支援システムの構築

獅々堀 達哉

学習指導要領の改訂により, 2020 年からプログラミング教育が小学校で必修化され, 2021 年に中学校, 2022 年には高校と本格的にプログラミング教育が導入された. これによると, プログラミング的思考力の育成が求められており, 情報のみならず, 如何なる教科を学習する際や社会で生きる上でも必要な力とされている. そして, 大学入試センターは, 2025 年の大学入学共通テストから「情報」を加える方針を公表している. 大学入学共通テストのサンプル問題例では, ソースコードの書き方に近い書き方で処理の流れを説明した「擬似コード」が出題されると考えられる. このような状況にあるにも関わらず, プログラミング教育の具体的な指導方法は明確に示されておらず, 情報を指導可能な教師の不足問題に陥っている.

本研究では, フローチャートから構造化チャートの一種である PAD(Problem Analysis Diagram) を介して擬似コードを理解できる事を想定し, プログラミング経験のない教師が授業で使用できる授業支援システムを構築する. 構築したシステムでは, フローチャートから構造化チャートを表しつつ, 手順を生成できるようにすることで, データとプログラミングの流れを明確に意識させる. システムに関する評価アンケートを実施した結果, フローチャートがどんな仕組みか知らなかった被験者が, 本システムを通じてフローチャートの仕組みを理解することができたとの回答が多かった. ポジティブな意見が多く見受けられたので, プログラミング経験のない教師でも, フローチャートから PAD を介して手順化支援できることが示唆された.

キーワード　プログラミング教育, プログラミング的思考力, フローチャート, PAD

Abstract

Development of a Programming Education Support System Using Flowchart and Structured Chart

SHISHIBORI Tatsuya

According to the revision of the Courses of Study in Japan, programming education has been made compulsory in elementary schools from 2020, junior high schools in 2021, and high schools in 2022. In this report, the development of programming thinking skills is required, not only for information but also for learning any other subjects and for living in society. In addition, the National Center for University Entrance Examinations has announced that it plans to add "information" to the 2025 Common University Entrance Test. In the sample questions of Common University Entrance Test, "pseudo-code" is expected to be presented, in which the flow of processing is explained in a way similar to the way source codes. In spite of this situation, specific teaching methods for programming education have not been clearly indicated, and the problem of a shortage of teachers who can teach information has arisen.

In this study, we develop a teaching support system that can be used in classes by teachers who have no programming experience, assuming that pseudo-code can be understood through PAD (Problem Analysis Diagram), a kind of structured chart, from flowcharts. In this system, teachers are made to be clearly aware of the flow of data and programming by being able to generate procedures from flowcharts while representing them in a structured chart. As a result of the evaluation questionnaire on this system, many of the subjects who did not know how flowcharts work answered that they were

able to understand how flowcharts work through this system. Many positive comments were found, suggesting that even teachers with no programming experience can support proceduralization from flowcharts through PADs.

key words Programming education, Programming thinking , Flowchart, PAD

目次

第 1 章

はじめに

近年, 情報技術の普及により私たちの日常生活の中において情報技術は切り離せないものとなっている．情報技術が発展していく社会で, これからの時代を生きていく子どもたちは情報技術の恩恵無しで生活することが困難であると推測される．現に学習指導要領の改訂により, 2020 年からプログラミング教育が小学校で必修化され, 2021 年に中学校, 2022 年には高校と本格的にプログラミング教育が導入された．更に, 2025 年から大学入学共通テストで「情報」が導入される．このような状況にあるにも関わらず, プログラミング教育の具体的な指導方法が不明確であったり, 情報科目を指導可能な教師の不足問題に陥っている．また, プログラミング教育はプログラミングのスキルを定着させることが目的ではなく, プログラミング的思考力の育成が求められる．プログラミング的思考力は特定の教科に依存せず, 如何なる教科を学習する際や社会で生きる上でも必要な力とされている．

プログラミング的思考力の育成の為に用いられる問題解決の方法を視覚的に表現する手段の一つとして, 中学校や高等学校の教科書ではフローチャートが扱われている．しかし, フローチャートのみでは全て同じループを表したり, 矢印を多用することで, データの処理や流れを把握しきれない．これらの問題点を解決するために, プログラムの構造を細分化し, 構造化されている部分のまとまりを明確に表現できる構造化チャートがある．

本研究では, フローチャートから手順を生成するのでは無く, フローチャートから構造化チャートの一種である PAD(Problem Analysis Diagram) を表しつつ, 手順を生成できるようにすることで, プログラミング経験のない教師でも, データとプログラミングの流れを明確に意識して手順化を理解でき, 授業を行うことができる授業支援システムを構築する．

第 2 章

プログラミング教育

2.1　プログラミング教育

プログラミング教育とは,「子供たちに, コンピュータに意図した処理を行うように指示することができるということを体験させながら, 将来どのような職業に就くとしても, 時代を超えて普遍的に求められる力としての「プログラミング的思考」などを育成するもの. コーディングを覚えることが目的ではない」と文部科学省は定義している [1]. また, 2020 度から小学校, 2021 年度から中学校, 2022 年度から高等学校でプログラミング教育が必修化されている.

プログラミング教育の学習活動は下記のようになっている.

- 児童・生徒がコンピュータで文字を入力するなどの学習の基盤として必要となる情報手段の基本的な操作を習得するための学習活動
- 児童・生徒がプログラミングを体験しながら, コンピュータに意図した処理を行わせるために必要な論理的思考力を身に付けるための学習活動

プログラミング教育を行う目的は以下のようなものが挙げられる.

- プログラミング的思考を育む

  コンピューターを自分の思い通りに動作させるため, 命令の順序立て, 条件分岐, 繰り返しなどの組み合わせを考え, 論理的に考える力を養うことができる. プログラミング的思考を育むと同時に, 複数の情報を結び付けて新たな意味を見出す力や, 問題の発見・解決等に向けて情報技術を効果的に活用する力を育む.

- プログラムの働きやよさに気付く

  身近なものにコンピュータが活用されていることに気付き, コンピュータの働きを社会づくりに役立てる意欲を養うことを目指す.
- 各教科での学びをより確実なものにする

  各教科での学習内容を, プログラミング教育を用いることでより確実にする.

また, プログラミング教育の現状については以下のようなものが挙げられる.

- 教育現場の ICT 環境の設備が不十分
- 学習指導要領等の指導書に指導内容が具体的に明記されていないため, 何を指導すべきなのかが不明確
- 教員側にもプログラミング教育に関する知識不足が目立つため, プログラミング教育に関する研修などが必要

2.2 Computational Thinking

Computational Thinking （計算論的思考） は, 2006 年にコンピュータ・サイエンティストの J.Wing(ジャネット・ウィング) 氏が雑誌に寄稿した視点である. ウィング氏は「読み, 書き, 計算」に加え、全ての子どもが Computational Thinking を学ぶべきだと主張している. Computational Thinking は問題解決のプロセスを分解したものであり, コンピュータ科学者のみならず, 全ての人にとって, あらゆる問題解決に使える手法である. 初等教育におけるプログラミング教育の目的は, プログラムを書けるようになるのがゴールではなく, Computational Thinking を身につけることに他ならない. Computational Thinking では, 問題発見能力, 課題の分析力, アイデア実現するための思考力・表現力, 協働力が身に付ける事ができる.

以下が Computational Thinking の 6 要素である.

- アルゴリズム (問題を解決する為に, 必要な一連のステップ)

- 分解 (問題を理解する為に, 取り組みやすい単位まで幾つかに分解する)
- 抽象化 (問題を単純化する為に, 重要な部分は残し, 不要な部分は除外する)
- パターン化 (パターンに気づくことで、次に何が起こるかを予測)
- 評価 (目的の為に、さまざまな要素・要因に基づいて判断する)
- 論理 (事実を確認し、予測を行うのに役立つ)

プログラミング的思考と Computational Thinking の相互関係について表 2.1 に示す.

表 2.1　プログラミング的思考と Computational Thinking の相互関係

プログラミング的思考	Computational Thinking
必要な動作を分割させて考える	分解
動作に適した命令を行う	抽象化
組み合わせる	アルゴリズム, パターン化, 論理
試行錯誤して改善する	評価

2.3　プログラミング的思考モデル

前述したように, プログラミング教育が必修化された大きな目的の一つとしてプログラミング的思考の育成がある. プログラミング的思考とは, 「自分が意図する一連の活動を実現するために, どのような動きの組合せが必要であり, 一つ一つの動きに対応した記号を, どのように組み合わせたらいいのか, 記号の組合せをどのように改善していけば, より意図した活動に近づくのか, といったことを論理的に考えていく力」と文部科学省は定義している [1]. プログラミング的思考は, プログラミングと論理的思考との関係を整理しながら提言された定義であり, 「Computational Thinking」を元に作られた日本独自のものである. 図 2.1 に示すように, プログラミング的思考は, Computational Thinking のごく一部の範囲でしかない概念であると言える. 文部科学省は Computationla Thinking を教えることは現実的ではないだろうと判断し, プログラミング的思考というかなり範囲を限定した言葉を定

義したと考えられる.

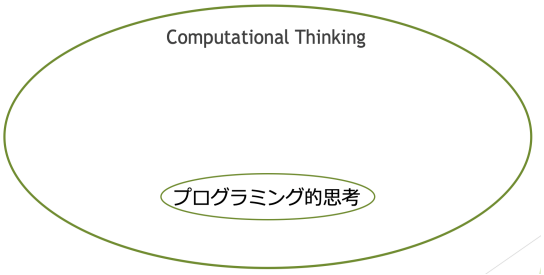


図 2.1　プログラミング的思考と Computational Thinking の位置付け

プログラミング的思考モデルとは，学習指導要領等を基にプログラミング的思考の育成に必要な要素や育成方法を整理することを目的に作成された [2].

図 2.2 の左側は一般的なプログラミング教育であり, 図 2.2 の右側はプログラミング的思考モデルを図式化したものである. 図中の Target は目標や解決する課題を， Process は手順や思考を， Program はソースコード等をそれぞれ意味している. 図 2.2 の左側の一般的なプログラミング教育では, 学習者は Program を用いて Target にアプローチすることで問題の解決を目指す. この際, Process は Target にアプローチする中で身に付く. 一方, 図 2.2 の右側のプログラミング的思考モデルでは, 学習者は頭の中で考えた Process をそのまま外化し, その Process を用いて Target にアプローチすることで問題の解決を目指す. Program は学習の状況や理解度に応じて必要が生まれれば, 発展的に学んでいくものと考えられている. プログラミング教育の目的の一つはプログラミング的思考を育成することであり， コーディングを覚えることが目的ではないと文部科学省は定義している. そのため, ここでは Process(手順や思考) と Program(ソースコード等) を区別している. プログラミング的思考モデルでは Process を外化し着目させることで, プログラミング的思考の育成につなげる.

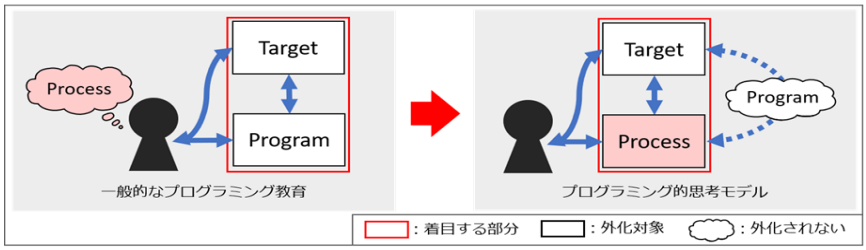


図 2.2　プログラミング的思考モデル

2.4　学習指導要領の現状

平成 15 年度（2003 年度）から始まった教科「情報」は, 情報活用の実践力を中心に学ぶ「情報Ａ」, 情報の科学的な理解を中心に学ぶ「情報Ｂ」, 情報社会に参画する態度を中心に学ぶ「情報Ｃ」の３科目から１科目を選択し, 履修していた. しかし, 生徒の自由意思ではなく, 学校側が科目指定を行うという状態が一般的だった. 全国の高等学校における各科目の開設状況は, 「情報 A」が約 80 %であるのに対し, 「情報 B」はわずかに 5 %, 「情報 C」も 15 %程度にとどまるなど, 「情報 A」に偏重していた [3].

平成 25 年度（2013 年度）から施行された学習指導要領では, 情報モラルを学ぶ「社会と情報」(以前の「情報 C」) とプログラミングを学ぶ「情報の科学」(以前の「情報 B」) の２科目から１科目を選択必履修に変わった. しかし, 学校側が科目指定を行うという状況に変わりはなく, 「社会と情報」が 80 %, 「情報の科学」が 20 %程度という開設状況であった [3]. よって, 生徒の 8 割程度はプログラミングを学んでいない.

さらに, 令和４年度（2022 年度）から施行されている新学習指導要領では, プログラミングやデータ活用を学ぶ新科目「情報 I」が全員必修となり, 発展科目として「情報 II」が設定された.

「情報 I」と「情報 II」の内容ついては以下の通りである.

- 情報 I
    1. 情報社会の問題解決
    2. コミュニケーションと情報デザイン

3. コンピュータとプログラミング

4. 情報通信ネットワークとデータの活用

- 情報 II

1. 情報社会の進展と情報技術

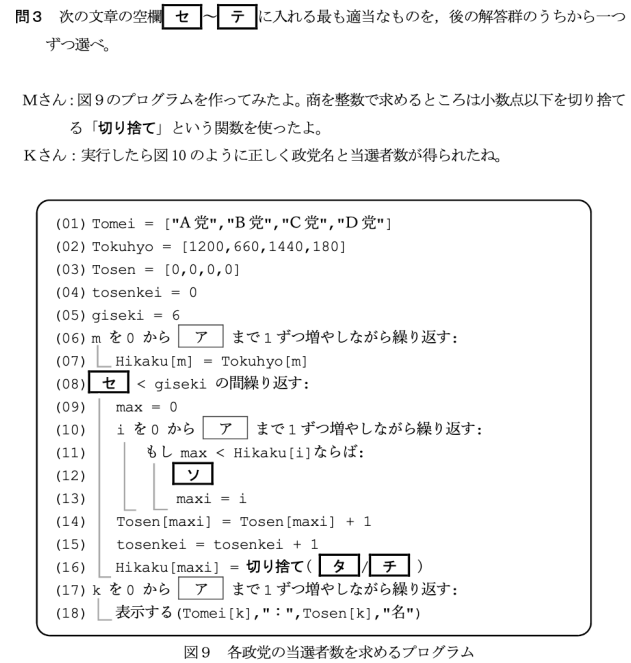
2. コミュニケーションとコンテンツ

3. 情報とデータサイエンス

4. 情報システムとプログラミング

5. 情報と情報技術を活用した問題発見・解決の探究

「情報 I」の第 3 章の「コンピュータとプログラミング」でアルゴリズムを考える手段の 1 つとしてフローチャートが挙げられている．また, 大学入試センターは, 2025 年の大学入学共通テストから「情報」を加える方針を公表している．図 2.3 は大学入学共通テストのサンプル問題例である．実際のソースコードの書き方とは異なるが, ソースコードの書き方に近い書き方で処理の流れを説明した「擬似コード」が出題されると考えられる．



問3　次の文章の空欄 セ ～ テ に入れる最も適当なものを，後の解答群のうちから一つずつ選べ。

Mさん：図9のプログラムを作ってみたよ。商を整数で求めるところは小数点以下を切り捨てる「**切り捨て**」という関数を使ったよ。

Kさん：実行したら図10のように正しく政党名と当選者数が得られたね。

```
(01) Tomei = ["A党","B党","C党","D党"]
(02) Tokuhyo = [1200,660,1440,180]
(03) Tosen = [0,0,0,0]
(04) tosenkei = 0
(05) giseki = 6
(06) m を 0 から [ ア ] まで 1 ずつ増やしながら繰り返す:
(07)     Hikaku[m] = Tokuhyo[m]
(08) [ セ ] < giseki の間繰り返す:
(09)     max = 0
(10)     i を 0 から [ ア ] まで 1 ずつ増やしながら繰り返す:
(11)         もし max < Hikaku[i]ならば:
(12)             [ ソ ]
(13)             maxi = i
(14)     Tosen[maxi] = Tosen[maxi] + 1
(15)     tosenkei = tosenkei + 1
(16)     Hikaku[maxi] = 切り捨て( [ タ ] / [ チ ] )
(17) k を 0 から [ ア ] まで 1 ずつ増やしながら繰り返す:
(18)     表示する(Tomei[k],"：",Tosen[k],"名")
```

図9　各政党の当選者数を求めるプログラム

図 2.3　共通テスト「情報」サンプル問題例 (独立行政法人大学入試センター)[4]

2.5　教える側の問題

情報教育の重要性が高まりつつあるこのような状況下において, 情報科目について専門性のある教員の確保が不可欠である. 文部科学省の調査では, 「情報」を教えている公立高校の教員 5100 人のうち, 1200 人は「情報」の免許を持っていない. 数学や理科などの教員が掛け持ちで教えているのが実情であり, 準備に十分な時間を割けないケースも多々ある. また, 情報科指導教員は, 国語, 数学, 英語といった主要教科ではないため, 各教育委員会は免許を保有している教員を積極的に採用してこなかったとされる [5]. これまで情報科目を指導にあたってきた教員は, 今後, 不慣れなプログラミングまで教えることに不安を感じている人が少なくない [5]. 情報科目を指導する教員のみに限らず, それ以外の各教員に対しても情報に関する知識は今後ますます必要となる.

2.6　関連研究

フローチャートを使うより PAD を使った方が正答率が上昇する研究は但馬等が行なっている [6]. 図 2.4 は累積正答者数と経過時間との関係, 表 2.2 は学生のプログラムの評価結果を示している.

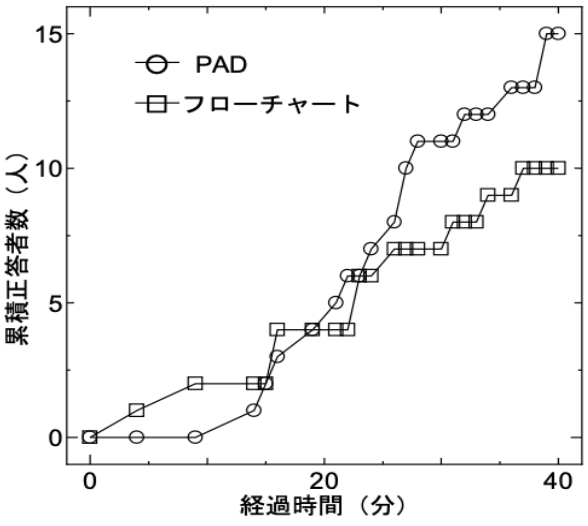


図 2.4　累積正答者数と経過時間との関係 [6]

表 2.2　学生のプログラムの評価結果 [6]

	フローチャート	PAD
プログラム未完成	1	0
文法エラー	4	2
エラーではないが, 正答ではない	4	2
正答である	10	15
計	19	19
正答率 (%)	53	79

但馬等の研究により, 以下の事柄が明確になっている.

- フローチャートを書ける人はデータの流れが把握できていないので, コードを記述するのが難しい
- PAD を書ける人は構造化を理解しているので, コードを記述できる

また, 河村等の研究では, JPADet(Japanese PAD editor and interpreter) という PAD を用いた学習支援システムを設計・開発している [7]. 表 2.3 により, 平均正解数が C 言語クラスよりも JPADet クラスの方が明らかに高いことが分かる. アルゴリズムを作成する能力に関しては, JPADet クラスの方が C 言語クラスよりも身についていることが明確になっている. この結果より, 河村等は 1 次元的にアルゴリズムを表現する必要がある C 言語に比べて, 2 次元的に図を用いてアルゴリズムを表現できる JPADet の方が優位であると結論づけられている.

表 2.3　平均正解数 [7]

	C 言語クラス	JPADet クラス
問題番号 1 (8 題)	2.36	2.31
問題番号 2 (6 題)	0.73	1.46
合計 (14 題)	3.09	3.77

2.7　本研究の目的

プログラミング教育では, データと処理の流れが一体になっているフローチャートが書ける必要がある. プログラミング教育では, プログラムを書くことが目的では無いとしても, データとプログラムの流れを構造化して取り扱うことができるようにするという考え方は重要である. ほとんどの高校の教科書ではフローチャートのみだが, そのフローチャートを構造化チャートとして表現することは, 実際に手順を記述する上で重要である. 従って, フローチャートから手順を生成するのでは無く, フローチャートから構造化チャートを表しつつ, 手順を生成できるようにすることで, データとプログラミングの流れを明確に意識して授業を行うことができる. フローチャートと PAD の関係, あるいはどのようにしたら変換できるのか. フローチャートでいうと, ループは回数によるループ, 条件を満たしている間の前置ループと後置ループの 3 通りある. それををどのように使い分けるか, あるいは構造化していくかという部分が難しい部分である. フローチャートで書くとただのループだが, ループの仕方の 3 通りを最初に選択できるようにすることによって, どのようなタイプのフローチャートであるのかの使い分けを実現する. フローチャートと PAD の関係を踏まえた上で, 疑似コードを生成する授業支援システムを構築する. 図 2.5 は本研究想定図, 図 2.6 は研究概要図を示している.

2.7　本研究の目的

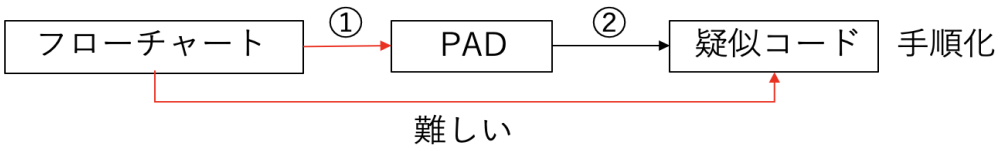


図 2.5　本研究想定図

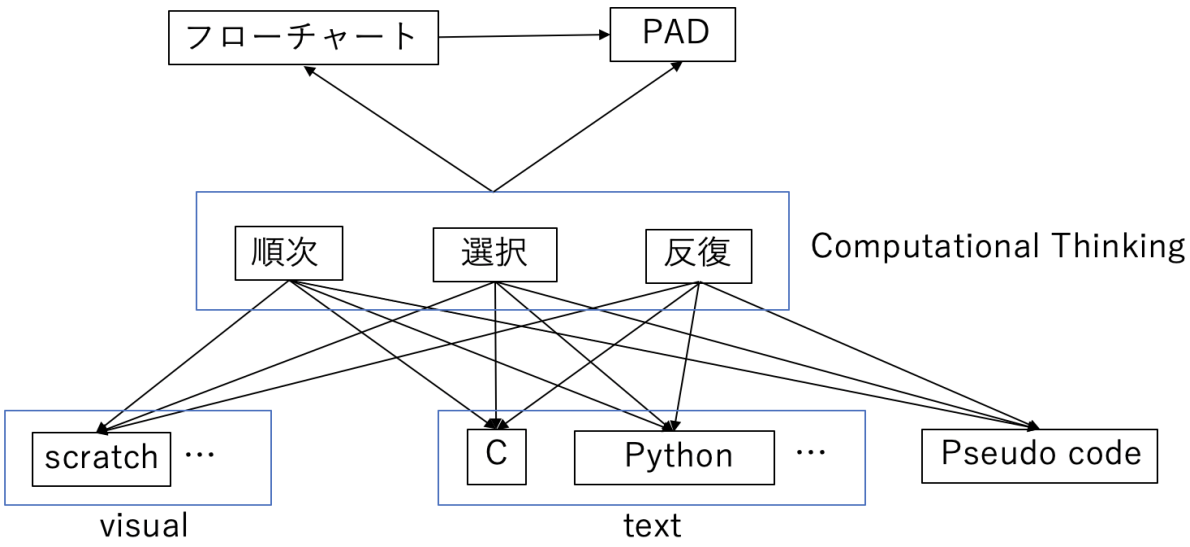


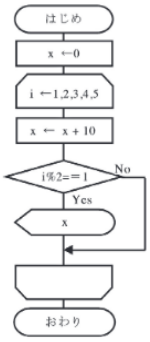
図 2.6　研究概要図

第 3 章

プログラミング授業支援システムの提案と設計

3.1　フローチャート

フローチャートは中学校の技術・家庭分野や高等学校の情報の教科書でも扱われているアルゴリズムを理解する手段の一つである．図 3.1 は「情報 I」の教員研修用教材の一部である．



図表 14　流れ図

1：x = 0 とする。
2：i = 1,2,3,4,5 とカウントを進めながら
以下の処理を繰り返し行う。
2 − 1：x に 10 を加える。
2 − 2：もし i%2==1 ならば x の値を表示する。

図表 15　日本語記述

図表 14 の流れ図，図表 15 の日本語記述をコードに置き換えたのが図表 16 である。今回の例では 10 を 5 回加えるが，i が奇数の時だけ x の値を表示しているので，x = 10,30,50 の場合だけ値が画面に表示される。

```
1  x = 0
2  print("x = " , x)
3  for i in range(1, 6, 1):
4        x = x + 10
5        if i%2==1:            #i を 2 で割ったときの余りが 1 のとき
6              print("x = " , x)
```

図表 16　コード

図 3.1　高等学校情報科「情報 I」教員研修用教材 [8]

仕事の流れ・手順や処理などを示す図で日本語では「流れ図」といわれており, コンピュータのプログラミングや作業管理に利用されている. フローチャートには国際的なフローチャートと JIS(日本工業規格) のフローチャートがある. 作業やデータの流れおよび処理を図式化すると, 問題解決の方法やプロセスが視覚化されるので, 作業内容の整理や正確な把握に役立っている. 処理の流れは基本的に上から下へ移動し, 線は交差させてはいけない. また, フローチャートには幾つかの欠点が指摘されている.

- フローチャートでは全て同じループという意味で表現しているが, コードで記述する際は何種類かの表現方法が存在する.
- 矢印を多用する事により, プログラムの制御構造が不明確になる. (細分化が困難)

従って, 図 3.1 の教員研修用教材に示すように, フローチャートからコードを記述して理解を促すだけではデータや処理の流れは把握しきれない [6].

3.2　構造化チャート (PAD : Problem Analysis Diagram)

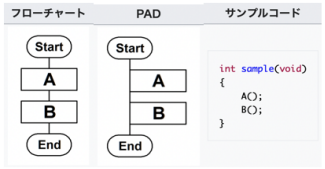
PAD(問題分析図) は日立製作所で考案された構造化チャートの一種である. フローチャートでは, 矢印と流れ線により上下左右の 4 方向にアルゴリズムが展開できることから, アルゴリズム全体の制御構造が複雑になっていた. しかし, PAD では処理の流れである基本線を上から下へ, アルゴリズムの構造を左から右へ表現しているので, シンプルな 2 次元的表現が可能である [6]. PAD は初心者にとって, アルゴリズムの構造を理解するのに取り組みやすいと考えられるが, 学校教育ではあまり馴染みがない. PAD はプログラムの構造を

- 順次 : 単純な処理の連続
- 選択 : 条件による処理の選択 (分岐)
- 反復 : 条件による処理の繰り返し

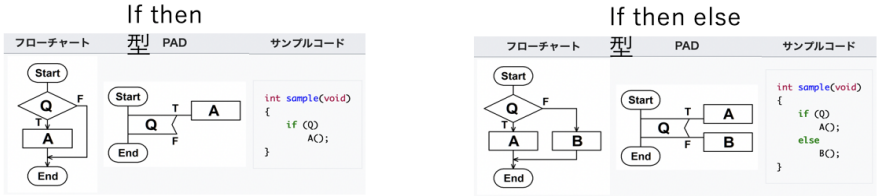
という内容に細分化して, 明確化することを目的としている. 図 3.2 は, フローチャートと PAD と擬似コードを並べている.

3.2　構造化チャート (PAD : Problem Analysis Diagram)

- 順次



- 選択



- 反復

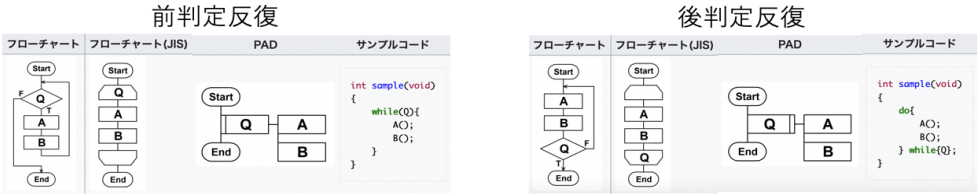


図 3.2　構造化プログラミングの 3 要素 [9]

図 3.3 にプログラミングの順次と分岐の機能を例にフローチャートと PAD の表現を比較して示す [6]. フローチャートは矢印で処理の流れを示し, PAD は上から下へ伸びる線で処理の流れを表す. 分岐については, フローチャートでは菱形で, 左右上下方向への分岐を表現する. PAD では上下 2 つに分岐し, 分岐後は左から右へ行くほど分岐が深くなり, 分岐後の処理を抜けると上から下へ延びる処理の流れの線に戻るので, 分岐と処理のアルゴリズムが 2 次元的に表現されていて分かりやすい.

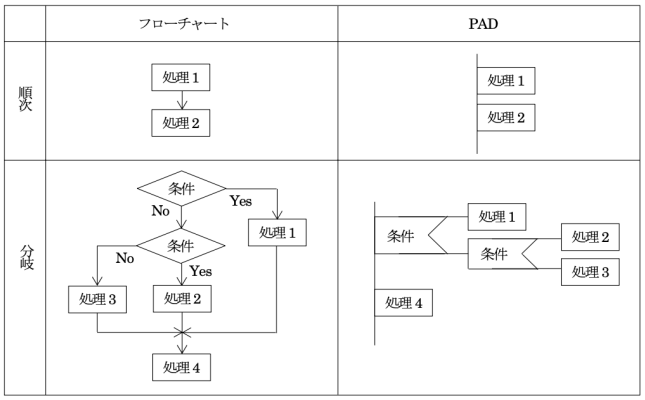


図 3.3　フローチャートと PAD の比較 [6]

3.3　本研究で取り扱うフローチャートおよび構造化チャート

本研究で取り扱うフローチャートを図 3.4 に, 構造化チャートを図 3.5 に示す．「Start」はフローチャートや PAD の初めを意味する．「Connect」は条件分岐する際にダミーな接点として扱う．「End」はフローチャートや PAD の終了を意味する．赤色の「Operation」は命令を意味する．黄色の「Loop_start」と「Loop_end」は繰り返しを意味する．青色の「if_Condition」は if 文, 「ifelse_Condition」は ifelse 文の条件分岐を意味する．

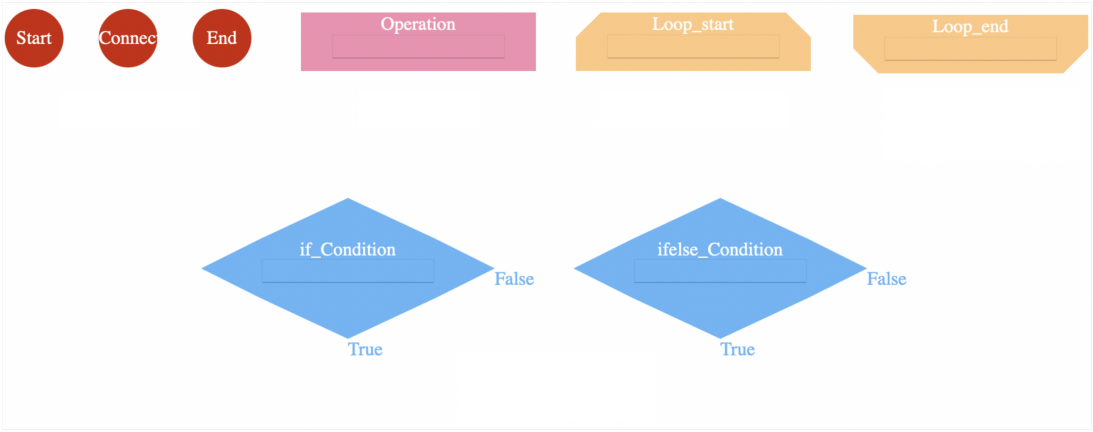


図 3.4　フローチャート

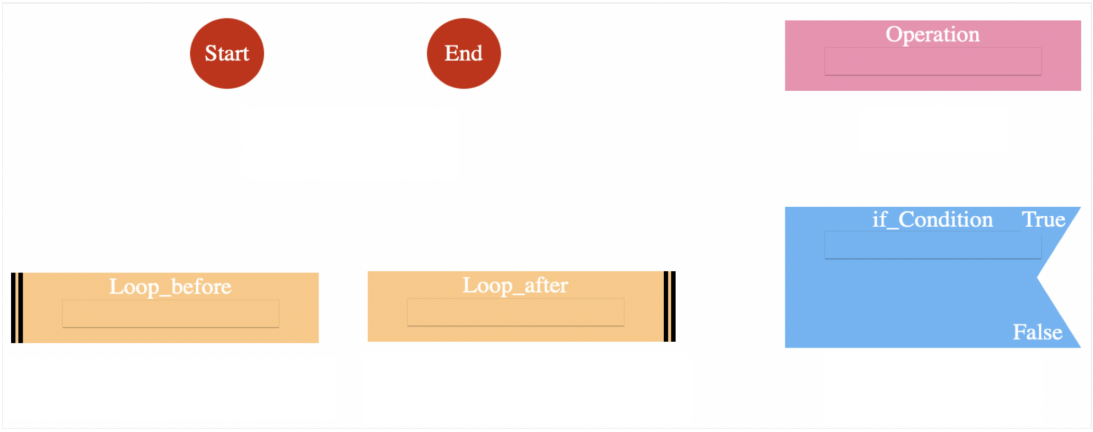


図 3.5　PAD

第 4 章

システムの構築

4.1　システム概要

図 4.1 に本システムの概要を示す．教員にとってタブレットや PC で利用しやすいシステムを考慮し, ブラウザ上で作動する．本システムではフローチャート作成部から開始する．まず, フローチャート作成部ではフローチャートの DOM(Document Object Model) を作成する．フローチャート作成部ではインタラクティブにフローチャートの各パーツを選択でき, ドラッグ&ドロップでのマウス操作により簡単にフローチャート DOM を作成することができる．フローチャート作成部の基本構成や技術的な詳細部分については, 4.2 節で説明する．次に, リストコード作成部ではフローチャート DOM を入力し, フローチャート DOM をリストコードに変換する．このリストコードとはフローチャートの処理順番を形式的な文字列で表現したコードであり, フローチャートから PAD へ変換するための中間表現として利用される．このリストコード変換部はフローチャート DOM が完成した後, フローチャート作成部から呼び出される流れになる．リストコード作成部の詳細については, 4.2.4 節で説明する．次に, PAD 変換部ではリストコードを入力とし, フローチャートに対応した PAD DOM を生成する．この PAD 変換部はリストコードが生成された後で呼び出される流れになる．PAD 変換部の詳細については, 4.3 節で説明する．最後に, 疑似コード変換部では PAD DOM を読み込み, フローチャートや PAD に対応した C 言語の疑似コードを作成する．この疑似コード変換部は PAD DOM が完成した後, PAD 変換部から呼び出される流れになる．疑似コード変換部の詳細については, 4.4 節で説明する．

本システムのファイル構成と動作内容を表 4.1 に示す．なお, 本システムが稼働する対応

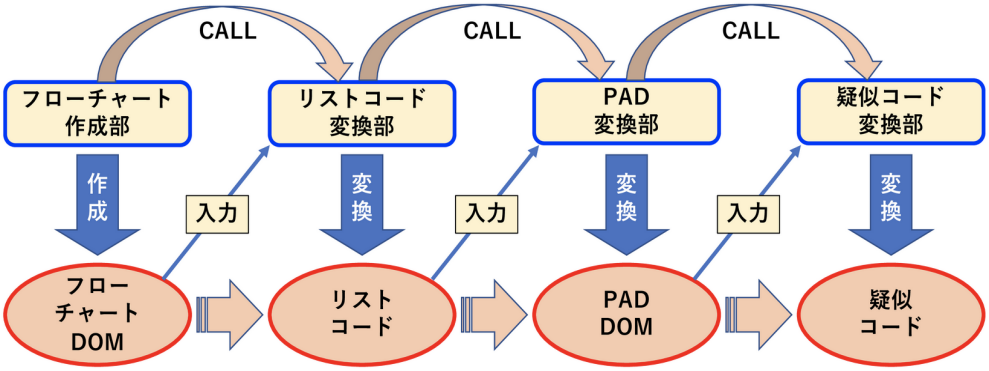


図 4.1　システム概要

ブラウザは, Safari 16.2, 及び Google Chrome 109.0.5414.87 である.

表 4.1　プログラムファイル構成

ファイル名	動作内容
flowchart.html	HTML ページ本体 (初期画面)
style.css	各ノード要素のスタイルを定義
flowchart.js	フローチャート作成部の JavaScript
output.js	リストコード変換部, PAD 変換部, 擬似コード変換部の JavaScript

次に, システムの画面構成について説明する. 図 4.2 にフローチャート作成部の初期画面を示す. 画面の左側にフローチャートを作成するための各種パーツを列挙している. このメニューから利用したいパーツをクリックすると, 右側の空白部分にクリックしたパーツの新しいノード要素が出現する. 初期画面の上部には, フローチャートを PAD に変換するための中間表現となるリストコードが出力されるテキストボックスを用意している. このボックスの右側にフローチャートを変換するための 2 種類のボタンを設置している. 上側の「リストを出力する」のボタンを押すことでリストコード変換部が呼び出され, 先に説明したフローチャートをリストコードに変換する. 下側の「PAD を出力する」のボタンを押すと PAD 変換部が呼び出され, リストコードを中間表現にしてフローチャートを PAD に変換する. な

お, PAD は新規の別ウィンドウに出力される.

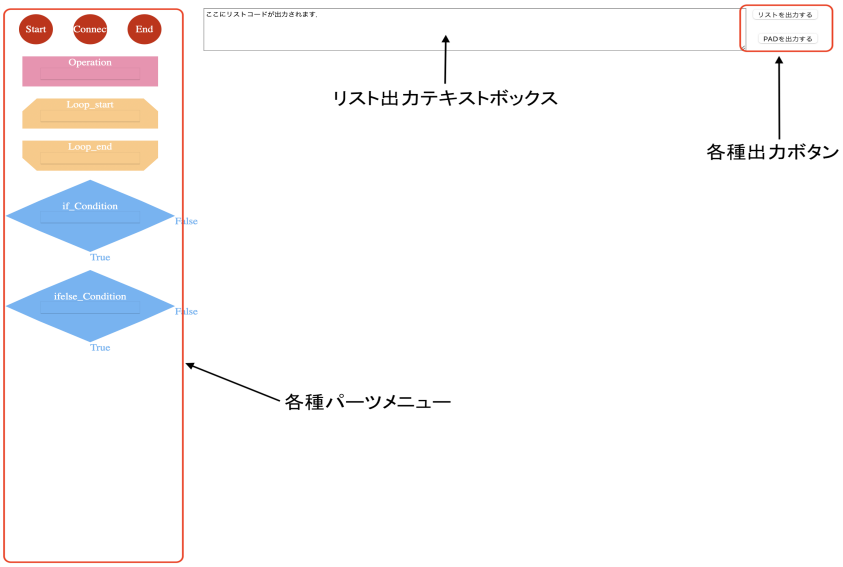


図 4.2　初期画面

図 4.3 にリストコード変換部によって, フローチャートからリストコードを出力した画面を示す. 画面中央にループ処理を伴うフローチャートが作成されている. この状態で「リストを出力する」ボタンを押すと, 画面上部のテキストボックスエリアにフローチャートから変換されたリストコードが出力される. このリストコードはフローチャート内での処理順番を表すリスト, 及び各ノード要素中央のテキストボックス内に入力された内容を表すコンテンツから成る. なお, フローチャートを部分的に変更した場合にも, 変更後ボタンを押すと, 変更後の内容を反映したリストコードを出力することができる.

4.1　システム概要

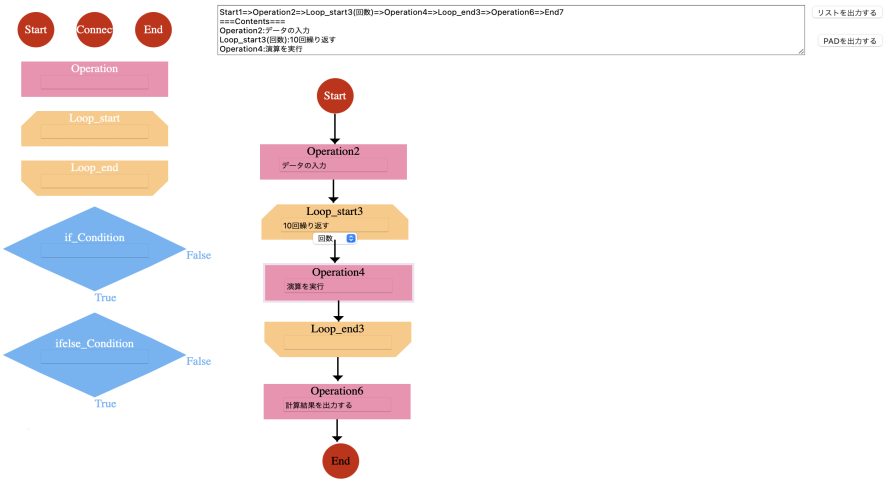


図 4.3　リスト出力画面

リストコードを作成した後,「PAD を出力する」ボタンを押すと, 図 4.4 に示すように別ウィンドウにリストコードから変換された PAD DOM が表示される. 図 4.4 の左側には PAD DOM で使用されるパーツが表示されており, 画面上部には図 4.3 と同様のリストコードがテキストボックスエリア内に表示されている. なお, この画面はフローチャート作成部と異なり, パーツの選択やドラッグでのパーツの移動などはできない仕組みになっている.

図 4.4 の右上にある「疑似プログラムを出力」のボタンを押すと疑似コード変換部が呼び出され, PAD DOM やフローチャートに対応した (C 言語に基づく) 疑似コードが出力される. 図 4.5 に示すように, 疑似コードは PAD DOM が表示されているページ内の別フレーム (下側) に表示され, 疑似コードと PAD DOM とを簡単に見比べることができる. また, フローチャートや PAD のテキストボックス内に記載されていた文字列は, 疑似プログラム内ではコメント文として表示されている. なお, 今回のフローチャートでは, ループ処理は 3 種類に分類されており, 図 4.5 のループ処理では for 文として疑似コードに表示されている. 更に, 図 4.5 の PAD で処理が深くなっている部分 (Operation4) については, 疑似コードでは tab キーを用いて処理の深さの違いを表現している.

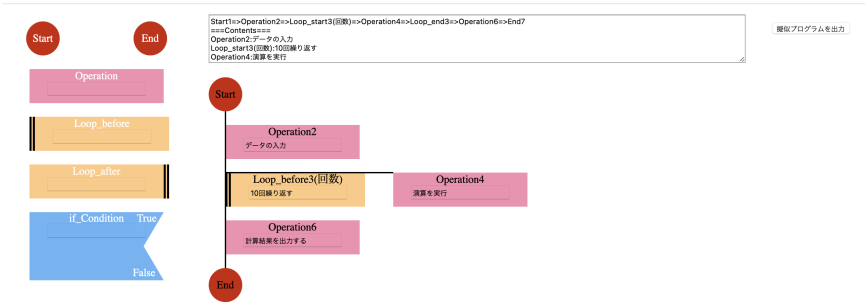


図 4.4　PAD 出力画面

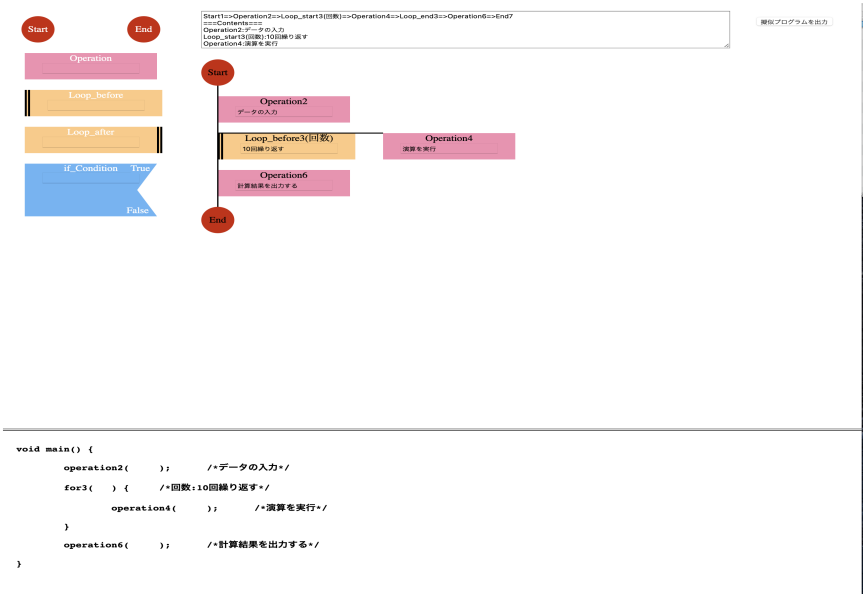


図 4.5　擬似プログラムコード出力画面

4.2　フローチャート作成部

4.2.1　フローチャート DOM の構成

本システムで用いるフローチャート DOM の各ノード要素について説明する．まず，フローチャートの各パーツの中で矢印以外のものをボックス系ノード要素と呼ぶ．各ボックス

系ノード要素のクラス名, 形状, id, 用途 (名称), 実装方法などについて表 4.2 に示す．また，各ボックス系ノード要素を描画した際の形状を図 4.6 に示す．

表 4.2　フローチャートで使用するボックス系ノード要素の各種情報

クラス名	形状	id	用途（名称）	実装に関する特記事項
circle	直径 50px の円形	start_circle	開始ノード（Start）	正方形を描画後に border-radius 属性の値を 50%に設定
		connect_circle	接続ノード(Connect)	
		end_circle	終了ノード（End）	
box	幅 200px 縦 50px の長方形	red_box	処理ノード（Operation）	長方形を描画
loop_start	幅 200px 縦 50px の長方形，左右上部の角切り落とし	loop_start	ループ開始ノード（Loop_start）	linear-gradient 関数と transparent 属性を使用し，左右上部角の透明化を実装
loop_end	幅 200px 縦 50px の長方形，左右下部の角切り落とし	loop_end	ループ終了ノード（Loop_end）	linear-gradient 関数と transparent 属性を使用し，左右下部角の透明化を実装
diamond	幅 150px 縦 50px の長方形，周囲 4 辺に三角形を付与した菱形	if_diamond	条件分岐 if ノード（if_Condition）	::before, ::after の擬似要素を用いて長方形の上下部に同型の三角形を付与，更にタグに擬似要素を用いて長方形の左右に同型の三角形を付与することで菱形を実装
		ifelse_diamond	条件分岐 ifelse ノード（ifelse_Condition）	

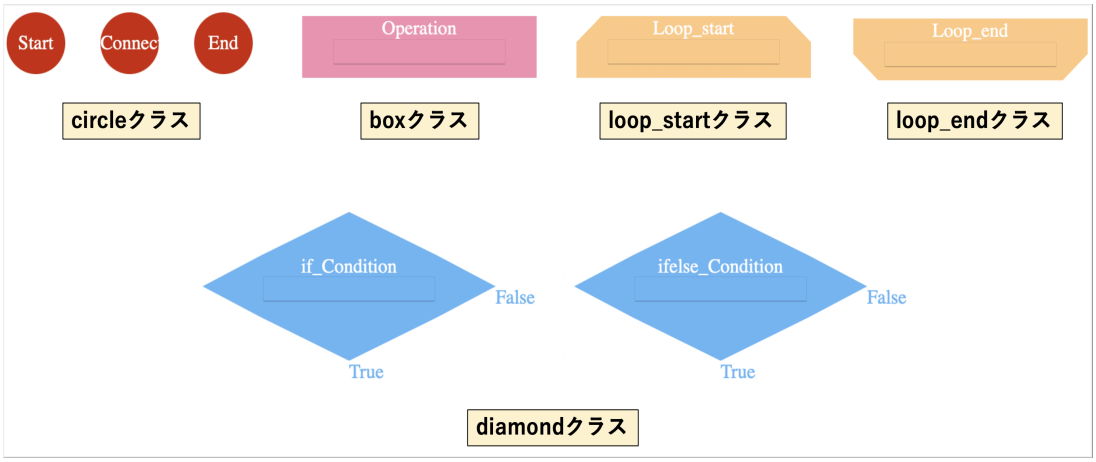


図 4.6　フローチャートで使うボックス系ノード要素

次に, フローチャートで使われる矢印系のノード要素のクラス名, 形状, id, 用途 (名称), 実装方法などについて表 4.3 に示す.

表 4.3　フローチャートで使用する矢印ライン系ノード要素の各種情報

クラス名	形状	id	用途	実装に関する特記事項
h_r_line	水平方向右向き矢印ライン	ライン毎に個別の id 番号を付与	ifelse ノードの False 側の矢印	SVG 画像として<path>タグを用いてラインの始点終点座標を指定することで描画
h_l_line	水平方向左向き矢印ライン		今回は未使用	同上
v_d_line	垂直方向下向き矢印ライン		各ボックス系ノードの下向き矢印	同上
v_u_line	垂直方向上向き矢印ライン		今回は未使用	同上
return_line	コの字型右向き矢印ライン		if ノードの False 側の矢印	4.2.3 に詳細を記載

矢印系ノード要素はボックス系ノード要素同士を接続する際に使用する. 矢印ラインは SVG 画像として描画しており, 描画方法は<path>タグを用いている. 例えば, h_r_line クラスの水平方向右向きの矢印ラインは下記のような<path>タグで描画できる.

【水平方向右向き矢印来の描画例】

```
<svg xmlns="http://www.w3.org/2000/svg" width="70px" height="20px">
<path d="M 0 10 L 70 10 L 65 5 V 15 L 70 10 Z" stroke="black"
stroke-width="2px">
</path>
</svg>
```

<path>タグ内の d=”…”の部分にパス描画の命令を記述する. ”M 0 10”は, x = 0, y = 10 に移動 (Move to), ”L 70 10”は, 現在地から x = 70, y = 10 に線を引く (Line to) を意味する. また, ”V 15”は, 垂直方向に y = 15 まで線を引くことを意味する. 水平方向に線を引く場合には, ”H x 座標”とする. 最後の”Z”はパスを閉じることを意味している.

次に, 表 4.4 にボックス系ノード要素に設定している属性に関する情報を示す. box_id 属性の値は, 図 4.2 の左側パーツメニューから使用したいボックス系ノード要素が選択され, 選択されたものが新たに複製される毎に連番の番号が付与される. top 属性から right 属性までの値は, ボックス系ノード要素の接続状態を把握するための情報であり, フローチャート DOM からリストコードに変換する際に利用される. top_link 属性から right_link 属性までの値は, 該当するボックス系ノード要素に向けて指されている矢印ライン (ノード要素に向かって入ってきている矢印ライン) の自動伸縮を制御するために用意している. 矢印ラインの自動伸縮機能については, 4.2.3 で詳細を説明する. なお, 表 4.4 の属性以外にスタイル属性として, 配置位置 (left, top など) や大きさ (width, height), 文字色 (white, black など) に関する値も保持している.

表 4.4 フローチャートで使用するボックス系ノード要素の属性

属性名	内容
box_id	ボックス系ノード要素を識別する id 番号
top	ボックス系ノード要素の上向き矢印が接続するボックス系ノード要素の box_id
bottom	ボックス系ノード要素の下向き矢印が接続するボックス系ノード要素の box_id
left	ボックス系ノード要素の左向き矢印が接続するボックス系ノード要素の box_id
right	ボックス系ノード要素の右向き矢印が接続するボックス系ノード要素の box_id
top_link	ボックス系ノード要素の上側に接続している矢印ライン系ノード要素の id
bottom_link	ボックス系ノード要素の下側に接続している矢印ライン系ノード要素の id
left_link	ボックス系ノード要素の左側に接続している矢印ライン系ノード要素の id
right_link	ボックス系ノード要素の右側に接続している矢印ライン系ノード要素の id

表 4.5 に矢印ライン系ノード要素に設定している属性に関する情報を示す. 矢印の id 属性についても, 新しく矢印が生成される毎に連番の番号が付与される. 表 4.5 の属性以外にスタイル属性として, 配置位置 (left, top など) や大きさ (width, height) に関する値も保持している.

表 4.5 フローチャートで使用する矢印ライン系ノード要素の属性

属性名	内容
id	矢印ライン系ノード要素を識別する id 番号
direction	矢印の方向（bottom, top, right, left のいずれか）
skip	return_line クラスの要素でのみ使用 上側のラインと下側のラインの長さの差(px)，上側が長いときには正の値，下側が長いときには負の値

4.2.2　フローチャート作成の流れ

フローチャート作成の流れを説明する．図 4.2 で示した初期画面の左側にある各種パーツメニューから必要なパーツ (ボックス系ノード要素) を選択する．図 4.7 にすべてのパーツを選択した際の画面を示す．選択された要素は複製され，複製された要素は画面の左上周辺に出現する．また，End ノード要素以外はフローチャート作成に必要な矢印ラインが子要素として付与された状態で出現する．各ノード要素には，フローチャート内での役割を表すテキスト (白色) が表示され，Start, End 以外のノード要素には，同種 (同じ形状) のノード要素と区別できるように識別番号 (box_id 属性の値) をテキストに付与している．ループ処理開始用のノード要素 (Loop_start) の下側中央には，ループ処理の種類として「回数」「前条件」「後条件」が選択可能なプルダウンメニューを用意している．これらは最終的に擬似プログラムコードに変換する際に，「回数」は for 文，「前条件」は while 文，「後条件」は do-while 文に対応することになる．

画面内に出現した要素は，マウスによるドラッグ操作で移動させることができる．ボックス系ノード要素本体部分や矢印ラインの先端付近以外をマウスでホールドした場合，矢印ライン要素はボックス系ノード要素の子要素となっているため，ボックス系ノード要素と共に矢印ラインも移動させることができる．一方，図 4.8 に示すように，矢印ラインの先端付近以外をホールドした状態でマウスをドラッグさせると，マウスの移動に伴って矢印ラインを伸縮させることができる．更に，矢印の向きに対して 90 度水平もしくは垂直方向にマウスをドラッグさせると，マウス進行方向に矢印を 90 度折れ曲げることができる．

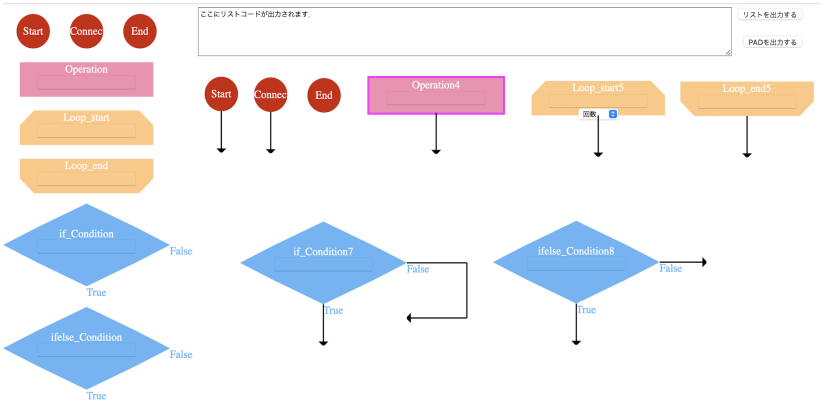


図 4.7　パーツ選択画面

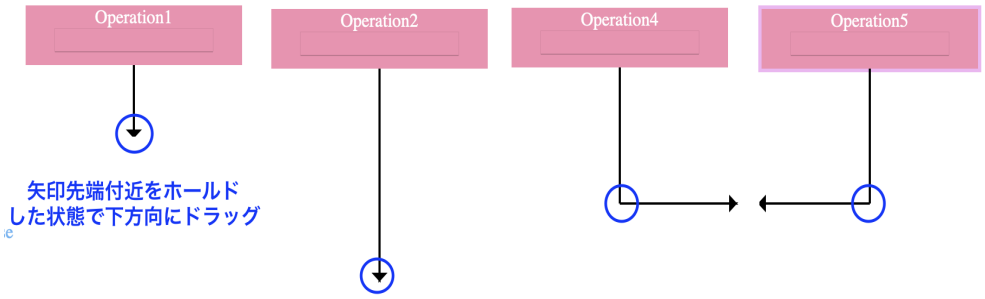


図 4.8　パーツ選択画面

図 4.9 にフローチャート作成した画面を示す．矢印ラインをドラッグし, 接続したい他のボックス系ノード要素の上部中央付近まで矢印を伸ばすことでボックス系ノード要素同士を接続させることができる．ボックス系ノード要素同士が接続しているか否かを視覚的に確認できるように, 矢印が接続側ノード要素の上部中央付近に届くと接続側ノード要素が点滅するように設定している．また, 矢印ラインのドラッグを終了した時点で, 接続状態になっているボックス系ノード要素内のテキストも文字色を白色から黒色に変更する処理も行っている．また, 接続状態の矢印を再度ドラッグし, 相手側ノード要素の上部中央付近から矢印を離すことで接続状態を解除することができる．接続状態が解除されたボックス系ノード要素は, テキスト文字が黒色から白色に変化する．なお, Start ノードだけは接続される相手側ノード要素がないためテキスト文字は白色のままになる．つまり, フローチャートの作成を完了す

る際には, Start ノード以外のテキストの文字色が黒色 (ノードが接続状態) になっていることを確認する必要がある. 更に, 接続状態にあるボックス系ノード要素をマウス操作で移動させると, 付随する矢印ラインも自動伸縮する. この機能については, 4.2.3 で詳細を述べる.

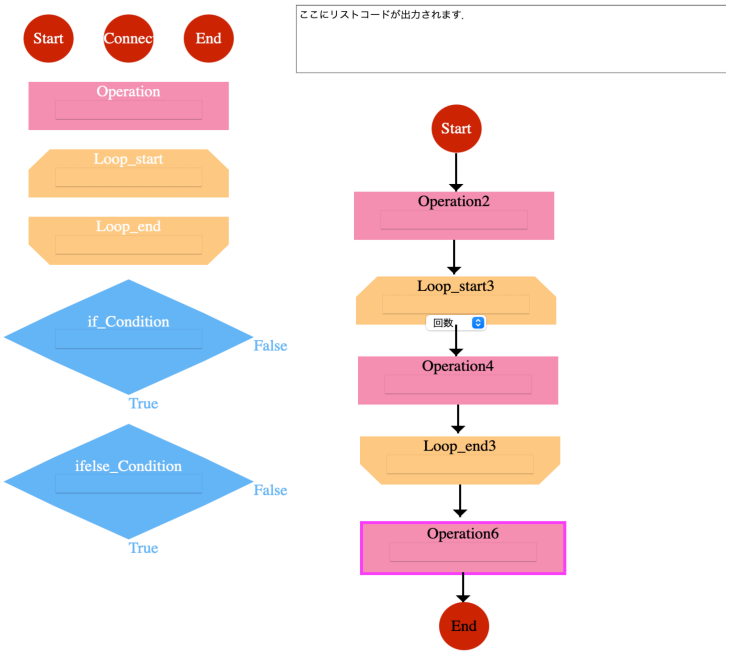


図 4.9　フローチャート作成画面

図 4.10 にフローチャート作成部の全体の処理の流れを示す. 図 4.10 に示す処理を実装した関数は, JavaScript の即時関数として定義しており, HTML 本体を読み込んだ後, 継続的に実行される. また, 図内の青文字は検知すべき各種マウスイベントを表す. まず, マウスが押されたことを示す mousedown イベントを検知すると, メニュー内のボックス系ノード要素が選択されたのか, canvas 内のノード要素が選択されたのかを判定する. ここで, 選択された要素がメニュー内か canvas 内かの区別については, HTML 本体で以下のように定義しており, getElementById 関数の引数として id 名を指定することで, 該当するノード要素を検索することができる.

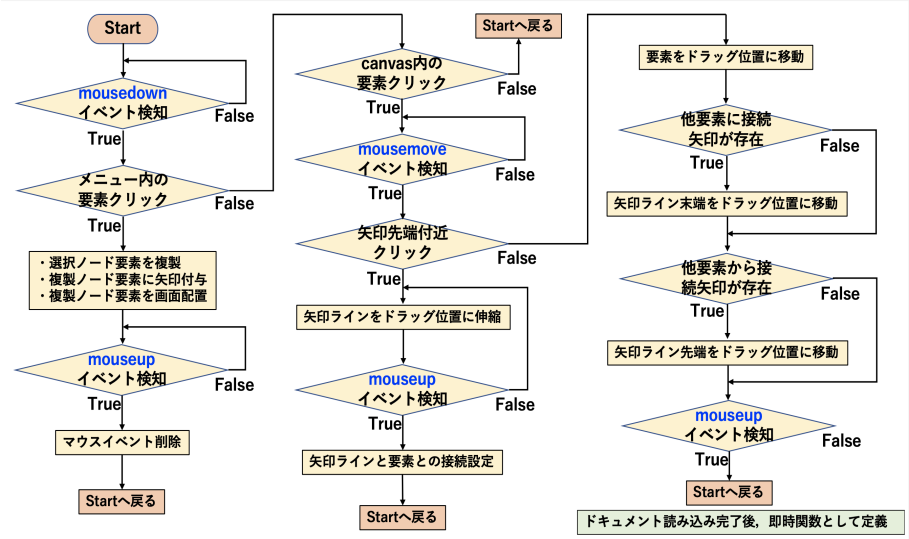


図 4.10　フローチャート作成処理の流れ

【HTML 本体の構成】

```
<body>
    <div id="canvas">    <!-- 画面本体の内容 -->
    ••••••••••••••
   </div>
   <div id="relative">    <!-- メニュー部分の内容 -->
    ••••••••••••••
   </div>
</body>
```

メニュー内のボックス系ノード要素が選択された場合には, 図 4.7 に示したように選択されたノード要素を複製し, 対応する矢印要素を子要素として追加した後, それらを canvas 内の所定の位置に配置する. その後, マウスが離されたことを示す mouseup イベントを検知すると, 各種マウスイベントそを初期化して Start に戻る.

canvas 内のボックス系ノード要素が選択された場合には, マウスがドラッグされたことを

示す mousemove イベントを検知すると, ホールドしているボックス系ノード要素を移動, もしくは矢印ライン要素を伸縮する処理を行う. ここで, 矢印の先端付近をホールドしている場合には, ドラッグされる位置に従って矢印ライン要素を伸縮する. その後, マウスが離されたことを示す mouseup イベントを検知した時点で, 矢印の先端付近が他のボックス系ノード要素と接触していれば, 双方を接続状態に設定する. 具体的には, 相手側ボックス系ノード要素の box_id 属性値を検索し, 自身矢印ライン要素の親要素であるボックス系ノード要素の top, bottom, left, right 属性のいずれか (接続方向に応じた属性) に検索した box_id を設定する. 更に, 相手側ボックス系ノード要素の top_link, bottom_link, left_link, right_link 属性のいずれか (接続方向に応じた属性) に矢印ライン要素の id を設定する. なお, 矢印の先端付近が相手側ボックス系ノード要素から離れた場合には, すでに接続状態になっている値はクリアーされる. このような矢印ラインと要素との接続設定を行なった後, Start に戻る.

一方, ボックス系ノード要素本体や矢印先端付近以外をホールドしている場合には, ホールドしているボックス系ノード要素を (矢印ライン要素とともに) 移動する. 同時に, 移動対象のボックス系ノード要素から他のボックス系ノード要素に接続している矢印ライン (矢印 A と呼ぶ), 及び他のボックス系ノード要素から移動対象のボックス系ノード要素に接続している矢印ライン (矢印 B と呼ぶ) が存在する場合には, 矢印 A の末端座標, 及び矢印 B 先端座標を移動対象のボックス系ノード要素の座標に応じて変更し, それぞれの矢印を再描画している. その後. mouseup イベントを検知した時点で Start に戻る.

4.2.3　各種機能

フローチャートを作成し易くするために以下の機能を実装している.

- 矢印ラインを 90 度折れ曲げる機能
- コ文字型矢印ラインを伸縮する機能
- 矢印ラインの自動伸縮する機能

まず, 図 4.11 に示すように矢印ラインの向きを途中で 90 度折れ曲げることができる．図 4.11 上部の Operation ノード要素の場合には， 下向き矢印ラインを途中で右方向に変更している．また， 図 4.11 中部の ifelse ノード要素では, 右側（False 側）に右向き矢印ラインが付与されるが， 下向きに折れ曲げることができる．次に，図 4.11 下部に示す if ノード要素については， 右側（False 側）に他のノード要素が接続しないため， 下側（True 側）に繋げるためのコ文字型矢印ラインを実装している．コ文字型矢印ラインは矢印付近をドラッグすると矢印を左右に伸縮することができるだけでなく， 右下角付近をドラッグすると図 4.11 に示すように幅や高さを調整することも可能である．

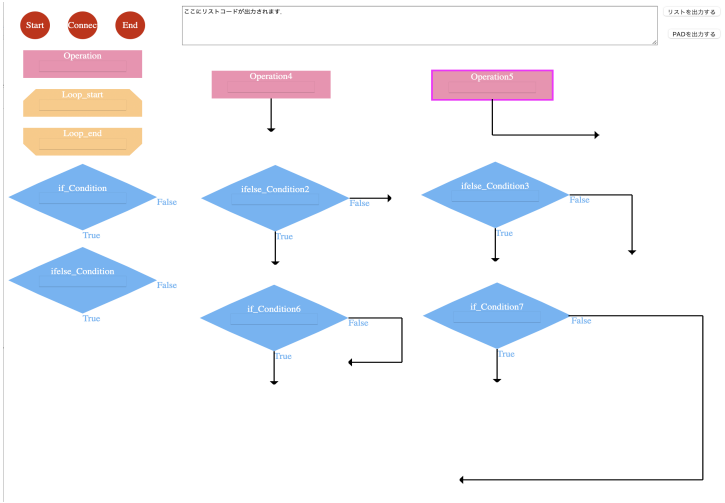


図 4.11　フローチャート機能画面

更に， 接続状態にあるノード要素間に存在する矢印ラインを自動で伸縮することもできる．図 4.12 の左の状態ように各ノード要素が接続状態にある場合， 各ノード要素を上下方向にドラッグして移動するだけで， ノード要素間の矢印ラインが自動伸縮する．この機能を用いると， 矢印ラインの調整をしなくてもノード要素の縦方向配置位置を簡単に調整することが可能になる．

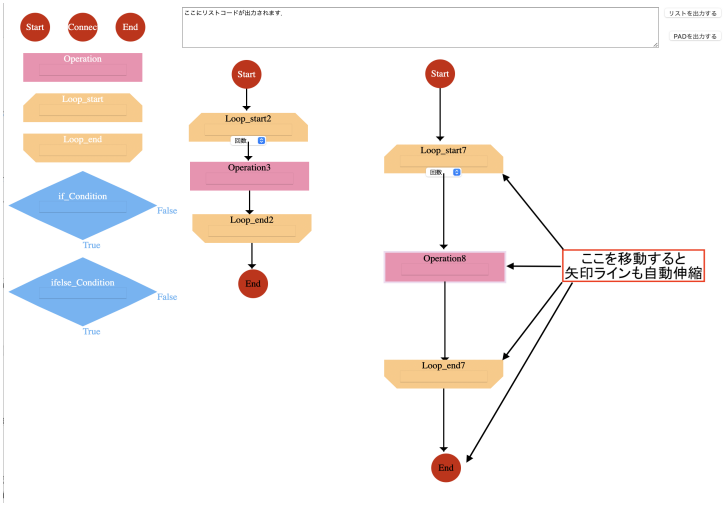


図 4.12　矢印の縦方向自動伸縮

また，図 4.13 のように横方向に矢印ラインを自動伸縮することもできるため，ifelse ノード要素の右側（False 側）の横幅調整が簡単に行うことができる．最後に，Connect ノード要素を導入している．これは if 系ノード要素の右側（False 側）の矢印が下側（True 側）とつながる場所を判定するためであり，後のリストコード生成時に役立てている．

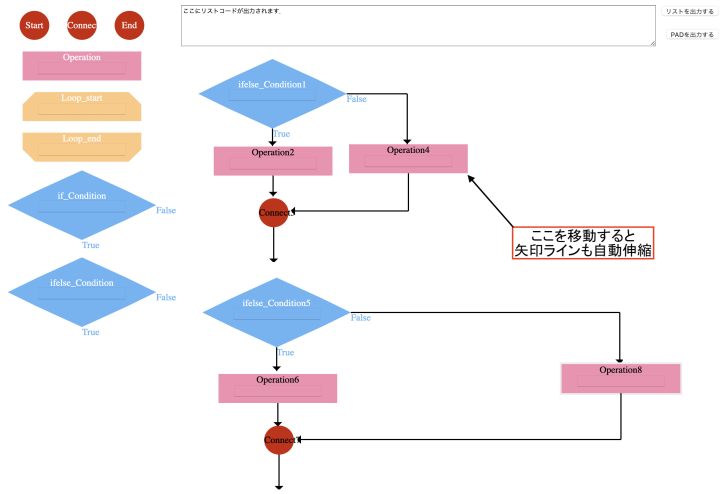


図 4.13　矢印の横方向自動伸縮

4.2.4　リストコード変換部

リストコード変換部では, 完成したフローチャートを PAD に変換するための中間表現としてリストコードを生成する．このリストコードはフローチャート内での処理順番を表すリスト, 及び各ノード要素中央のテキストボックス内に入力された内容 (各処理のコメントや条件式など) を表すコンテンツから成る．リストは PAD の生成順番に深く関わってくるため, PAD を生成しやすくするような工夫が必要になる．特に, if 文のように True 側と False 側に処理が分岐する場合には, どちらの処理を優先して先に出力するかの検討が必要になる．

図 4.14 に入れ子状態の if 文を含むフローチャートから PAD に変換した例を示す．図 4.14 内の丸数字は PAD を描画する際の各種ノード要素の描画順番を表している．特に, PAD の if ノード要素 (if_Condition3 と if_Condition6) の高さは，それぞれ True 側 (右側) に存在する処理ノード数によって決まる．つまり, PAD の if ノード要素を描画するためには, 事前に True 側のすべての処理を描画しておく必要がある．例えば, 図内の 6 番の if ノード要素を描画するためには, 4 番と 5 番を先に描画する必要があり, 7 番の if ノード要素に対しては, 2 番から 6 番すべてを先に描画しておかないと, 7 番の if ノード要素の高さが決まらない．そのため, フローチャートからリストを生成する際には, False 側よりも True 側を優先して先に出力する必要がある．上記の理由により, リストを生成する際には, フローチャートを幅優先ではなく, 深さ優先で各種処理を出力する．図 4.14 のフローチャートから生成されたリストを以下に示す．左から右へ順に処理が進む．if 文については, スタックを用いて PAD での描画順を調整する．詳しい内容については, 4.3.2 で説明する．

【図 4.14 のフローチャートから生成されたリスト】

Start=>Operation2=>if_Condition3=>Operation4=>Operation5=>if_Condition6=>
Operation7=>Operation8=>Connect9=>Connect10=>End

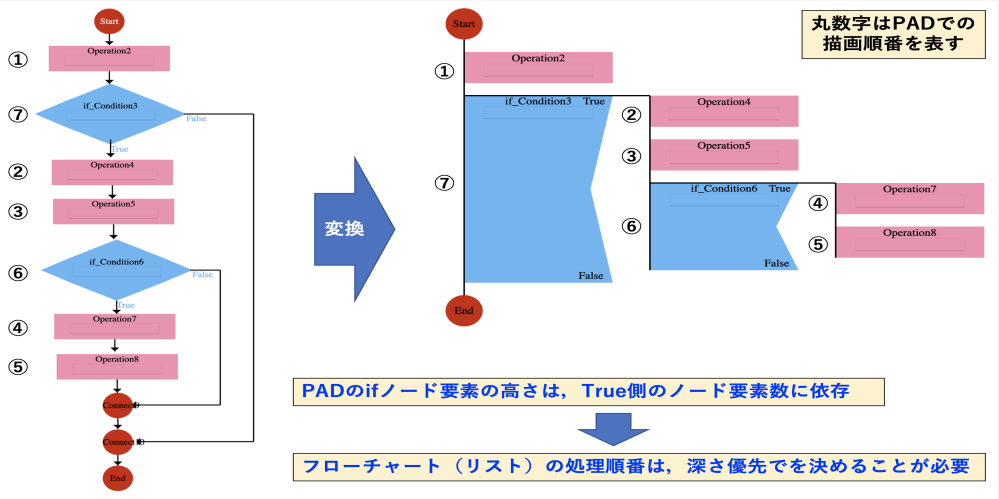


図 4.14　if 文を含むフローチャートから PAD への変換例

図 4.15 に入れ子状態の if-else 文を含むフローチャートから PAD に変換した例を示す．図 4.15 内の丸数字は PAD を描画する際の各種ノード要素の描画順番を表している．if-else 文の場合には，False 側にもノード要素が存在するが，PAD の if ノード要素の False 部分が if ノード要素の下側に描画されるため，PAD の if ノード要素の高さには影響を与えない．このことから，if-else 文に対する描画順は，True 側のノード要素，次に if ノード要素，最後に False 側のノード要素となる．例えば，図内の 6 番の if-else ノード要素に対しては，最初に 4 番と 5 番のノード要素を描画し，次に 6 番の if ノード要素，最後に False 側の 7 番と 8 番のノード要素を描画する．9 番の if-else ノード要素に対しては，True 側の 2 番から 8 番までの各ノード要素を描画し，次に 9 番の if ノード要素が描画された後，最後に False 側のノード要素が描画される．このように False 側の一連の処理は最後に描画することができるので，深さ優先でたどったリストとは別の系統として出力することにする．図 4.15 のフローチャートから生成されたリストを以下に示す．二つの if-else 文の False 側の処理は，それぞれ 2 行目と 3 行目のリストに対応する．ここで，False 側のリストは End ノードまで出力せず，深さ優先の 1 行目のリストに繋がる箇所までにしている．その理由は，それぞれのリストに重複があると PAD 変換の際にエラーを生じるため，及びループ系ノード要素のプルダウンメニューの値を取得ができないためである．なお，if-else 文についてもスタックを用いて PAD

での描画順を調整する．詳しい内容については，4.3.2 で説明する．

【図 4.15 のフローチャートから生成されたリスト】

Start=>Operation2=>ifelse_Condition3=>Operation4=>Operation5=>ifelse_Condition6=>Operation7=>Operation8=>Connect13=>Connect14=>End

ifelse_Condition6(False)=>Operation11=>Operation12

ifels_Condition3(False)=>Operation9=>Operation10

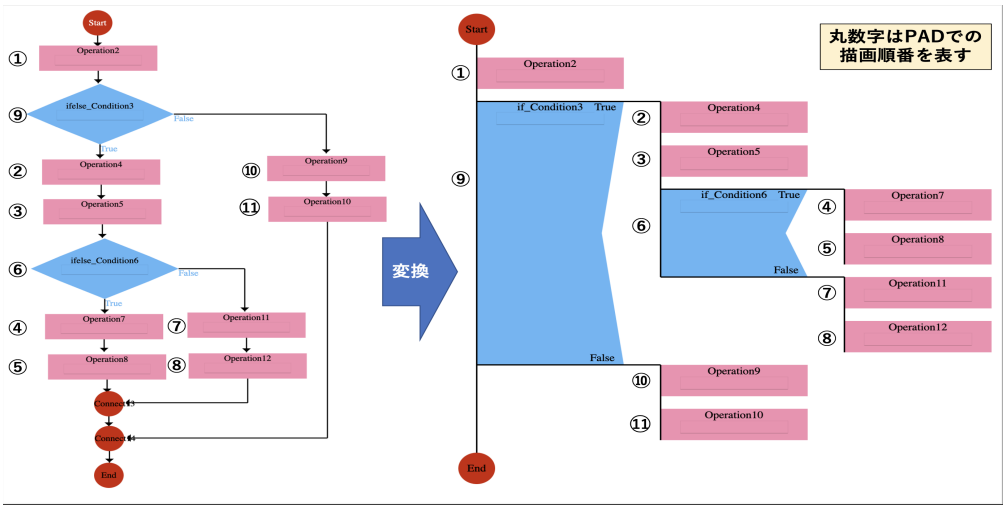


図 4.15　if-else 文を含むフローチャートから PAD への変換例

4.3　PAD 生成部

4.3.1　PAD DOM の作成

本システムで用いる PAD DOM の各ノード要素について説明する．まず，PAD で使用する各ボックス系ノード要素のクラス名，形状，id，用途 (名称)，実装方法などについて表 4．6 に示す．また，各ボックス系ノード要素を描画した際の形状を図 4.16 に示す．

表 4.6　PAD で使用するボックス系ノード要素について

class 名	形状	id	用途（名称）	実装に関する特記事項
circle	直径 50px の円形	start_circle	開始ノード（Start）	特になし
		end_circle	終了ノード（End）	
box	幅 200px 縦 50px の長方形	red_box	処理ノード（Operation）	特になし
pad_loop_before	幅 200px 縦 50px の長方形，左側に縦線	pad_loop_before	前条件ループノード（Loop_before）	border-left 属性を使用し，長方形の左側に縦線を描画
pad_loop_after	幅 200px 縦 50px の長方形，右側に縦線	pad_loop_after	後条件ノード（Loop_after）	border-right 属性を使用し，長方形の右側に縦線を描画
pad_if	幅 200px 縦 100px の長方形から右辺上下部分が三角形のような尖った形状	pad_if	条件分岐 if ノード 条件分岐 ifelse ノード（if_Condition）	clip-path 属性で長方形の周囲を polygon で指定した座標に切り取ることで目的とする形状を実装

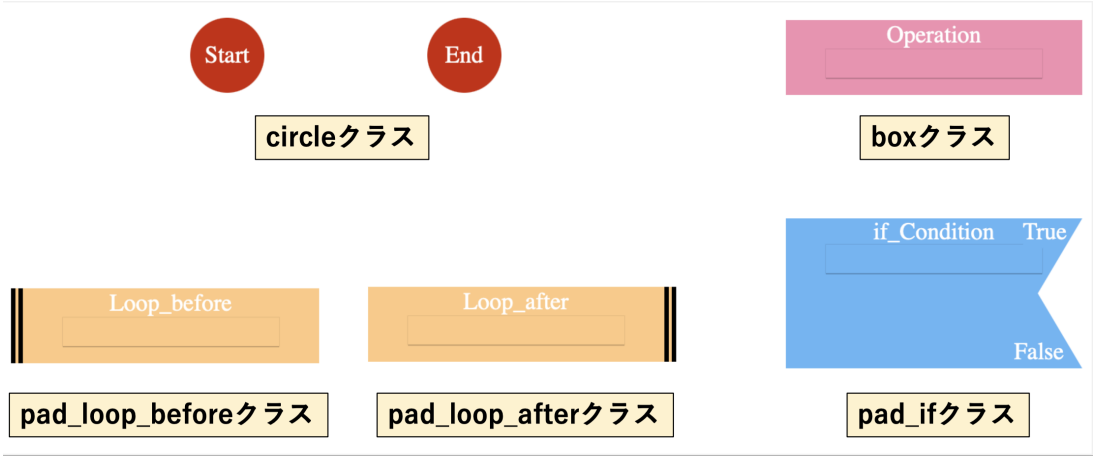


図 4.16　PAD で使うボックス系ノード要素

次に，表 4.7 に PAD で使用するボックス系ノード要素に設定している属性に関する情報を示す．ほとんどフローチャートで使用するボックス系ノード要素と同じだが，pad_if クラスのノード要素には同じ向き (right) の上下に True 側と False 側の処理が繋がるため，属性

を分けている．表 4.7 の属性以外にスタイル属性として, 配置位置 (left, top など) や大きさ (width, height), 文字色 (white, black など) に関する値も保持している．

表 4.7　PAD で使用するボックス系ノード要素の属性

属性名	内容
box_id	ボックス系ノード要素を識別する id 番号
bottom	ボックス系ノード要素の下側に接続するボックス系ノード要素の box_id
right	ボックス系ノード要素の右側に接続するボックス系ノード要素の box_id
top_right	pad_if クラスの要素でのみ使用，ボックス系ノード要素の右上側（True 側）に接続している矢印ライン系ノード要素の id
bottom_right	pad_if クラスの要素でのみ使用，ボックス系ノード要素の右下側（False 側）に接続している矢印ライン系ノード要素の id

4.3.2　リストから PAD DOM の変換方法

図 4.17 にリストから PAD へ変換するアルゴリズムのうち, if 文, if-else 文を除いた内容を示す．図内の青色部分は各処理の実装内容を詳しく述べている．リストはフローチャートの処理順を文字列で表現したコードであるが, 各処理を「=>」の記号で連結している．そのため, 「=>」の記号をデリミターとして, リストを処理毎の文字列に分割し, 配列 list[] に格納するといった前処理を施した状態を仮定したアルゴリズムになっている．なお, アルゴリズム内で「list[i+1] の box_id 値」といった記載があるが, リストコード内の文字列は, 各処理の名称と box_id 値が連結しているので, 正規表現を用いた照合処理により, 後半の数字の部分を box_id 値として取り出すことができる．

まず, “Start”, “End”, “Operation” については, 対応する PAD のボックス系要素を複製し, posX, posY の座標に配置する．その後, 要素同士が重なって描画しないように, 実の要素の配置位置を変更した後, 下方向 (bottom 属性) の接続設定を行う．なお, “End” については接続設定を行わず終了する．

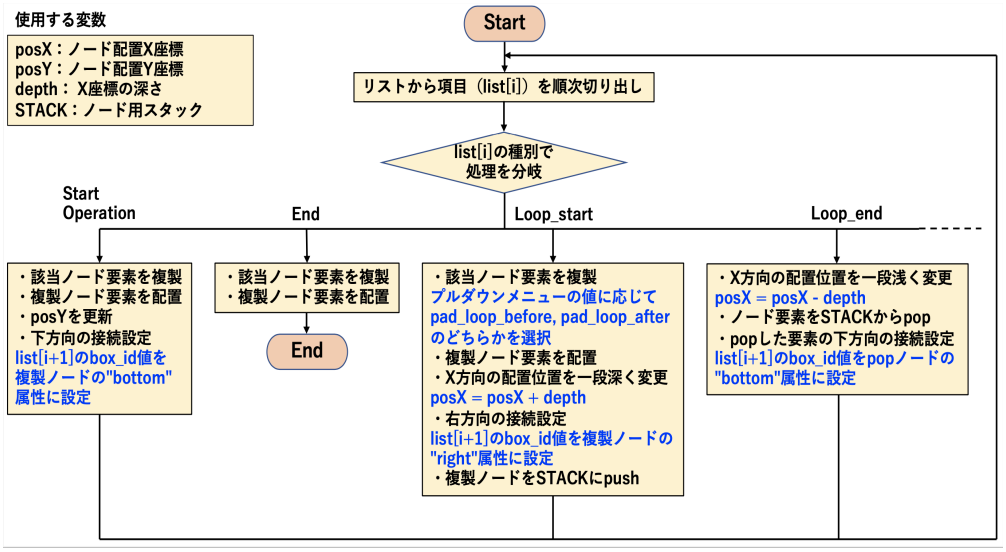


図 4.17　リストから PAD 変換アルゴリズム 1

次に, “Loop_start”, “Loop_end” のループ処理については, 例に従って処理内容の説明を行う. 図 4.18 にループ処理を含むリストから PAD への変換処理の例を示す. “Loop_start” の場合には, まず, 該当するボックス系ノード要素を複製する. この際に, フローチャート作成の際にプルダウンメニューに設定された値に応じて, pad_loop_before クラスか pad_loop_after クラスかいずれかのノード要素が選択される. なお, プルダウンメニューの値はリストの文字列にも反映しているので, リストから取得可能である. 次に, 複製したノード要素を配置した後, ループ処理が現れると PAD は横方向 (X 座標軸方向) に処理が深くなるため, posX 座標を depth 分だけ増加する. 本システムでは, ノード要素同士が重ならないように, depth の値はノード要素の横幅より少し大きい 250px に設定している. 最後に, 右方向 (right 属性) の接続設定をし, 複製ノードをスタックに格納する. スタックを導入する理由は, ループ処理が入れ子になった場合, 最も深いノード要素から (後入れ先出し) 順にスタックら出力し, 処理の深さを順次浅くし, かつ, 下方向に接続を順次設定するためである. 図 4.18 には, 3 回入れ子になったループ処理をスタックを用いて配置の深さと下方向の接続を順次行っている過程を示す.

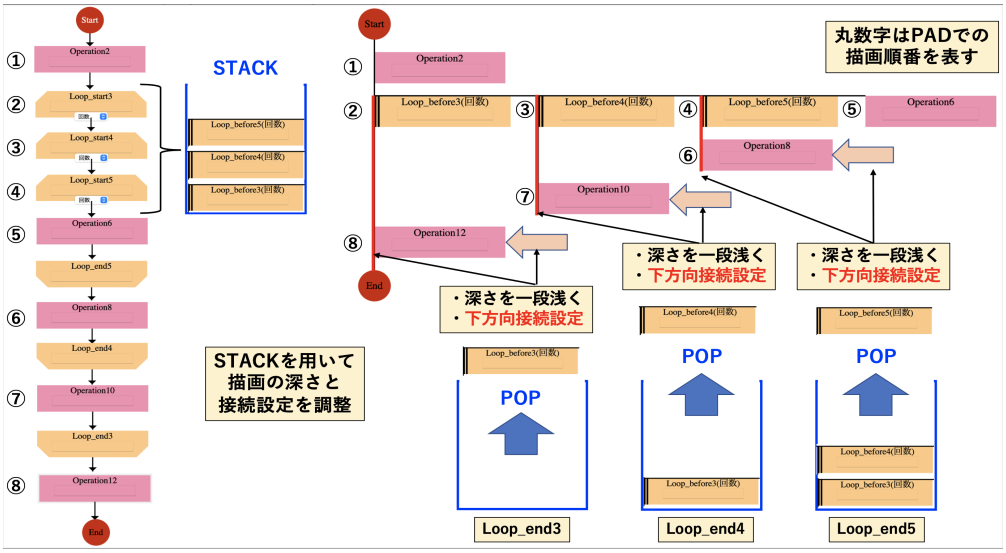


図 4.18　ループ処理を含むリストから PAD 変換例

図 4.19 にリストから PAD へ変換するアルゴリズムのうち, if 文, if-else 文の内容を示す. 図内の青色部分は各処理の実装内容を詳しく述べている. “if_Condition”, “ifelse_Condition” ともに PAD のボックス系ノード要素では pad_if クラスの要素になる. この要素の縦方向の長さは, True 側に配置される要素の個数から決定される. そのため, pad_if クラスの要素を画面内に配置するタイミングは, True 側の要素の配置が終了した時点, 言い換えると, リスト上で if 文, if-else 文に対応する “Connect” が出現した時点になる. そのため, 変換アルゴリズムにおいて, list[i] の内容が “if_Condition”, “ifelse-Condition” の場合には, 該当ノードの複製, X 軸方向の配置位置を変更, True 側に続く処理との接続設定などは行うが, 複製ノード要素の配置は list[i] の内容が “Connect” の際に行われる.

list[i] が “Connect” の場合, 具体的な処理の流れを例とともに説明する. 図 4.20 に条件分岐処理 (if) を含むリストから PAD へ変換する例を示す. pad_if クラスのノード要素を配置するためには, 二種類のスタックを用いる. 一つは, ループ処理と同様, 複製されたノード要素を格納するノード用スタック (nodeSTACK) , もう一つは, リスト内で “if_Condition”, “ifelse-Condition” が出現した時点でのノード配置 Y 座標 (posY) を記憶するための座標用スタック (posSTACK) である. nodeSTACK は描画の深さの調整, 及び下方向 (bottom 属

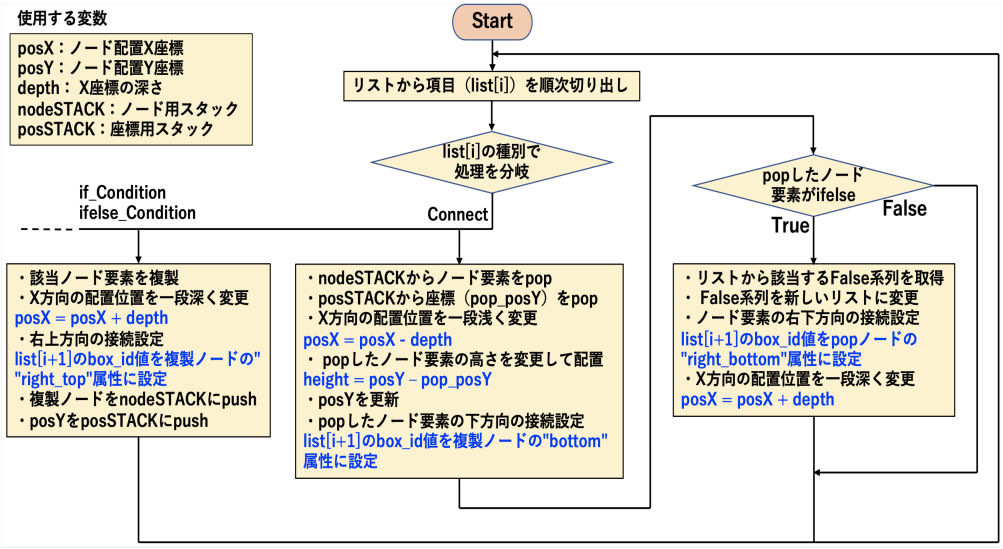


図 4.19　リストから PAD 変換アルゴリズム 2

性) の接続設定を行う役割がある．一方, posSTACK は pad_if クラスのノード要素の高さを決める役割がある．図 4.20 では, 二つの if 文が現れた時点で, 各スタックは図に示すようにそれぞれ二つの要素, データが格納された状態になる．この時点では, pad_if クラスのノード要素は描画されず, 2 番から 5 番の True 側の各要素が順次描画される．その後, list[i] が “Connect9” になった際に, 各スタックからノード要素と配置座標 (pop_posY) が出力される．特に, 座標については, この時点での posY が 350px と仮定すると, スタックから出力された pop_posY との差を求めることで, 6 番のノード要素の高さを 150px に変更し, posX と pop_posY で指定される座標にノード要素を配置する．つまり, pad_if クラス要素の左上の初期位置を posSTACK に記憶させておき, True 側の全要素の配置が終了した時点での配置位置との差を求めることで, pad_if クラス要素の高さを求めている．同様に, list[i] が “Connect10” になると, 7 番の pad_if クラスのノード要素が配置される．ここでは, pop_posY から出力される pop_posY が 100 であり, この時点での posY が 370px と仮定すると, 7 番のノード要素の高さは 270px に変更される．

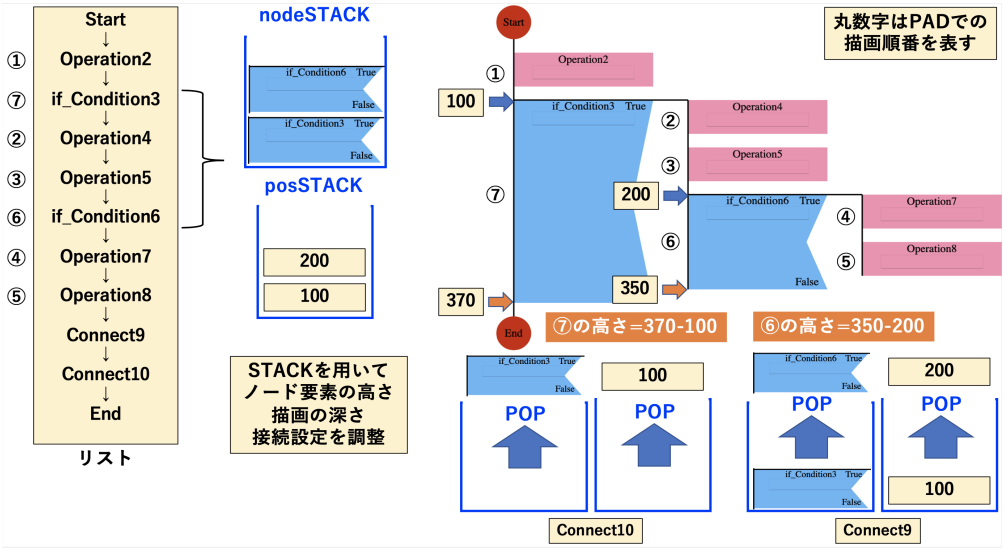


図 4.20　条件分岐処理 (if) を含むリストから PAD 変換例

次に, if-else 文に対する処理を説明する．図 4.21 に条件分岐処理 (if-else) を含むリストから PAD へ変換する例を示す．if-else 文に対する処理も基本的に if 文の処理と同じであるが, False 側の要素が追加される．リストコード内で False 側のリストは True 側のリストとは別の行に記載されているため, まず最初に, リストコードから該当する False 側のリストを取得する．その後, 取得したリストに対して同様の処理を行う．ここで, False 側の要素は pad_if クラス要素の右下側から下方向に配置されるため, 自身の pad_if クラス要素の高さには影響を与えない．ただし, 入れ子になっている場合, 内側の if 文における False 側の要素数は, 外側の pad_if クラス要素の高さに影響する．例えば, 図 4.21 において 7 番と 8 番の要素は 6 番の高さには影響しないので, 6 番, 7 番, 8 番の順番で描画する．一方, 7 番と 8 番は 9 番の pad_if クラス要素の True 側になるので, 9 番の高さには影響する．そのため, 7 番, 8 番, 9 番の順番で描画する．その後, 10 番以降の False 側要素を描画する．

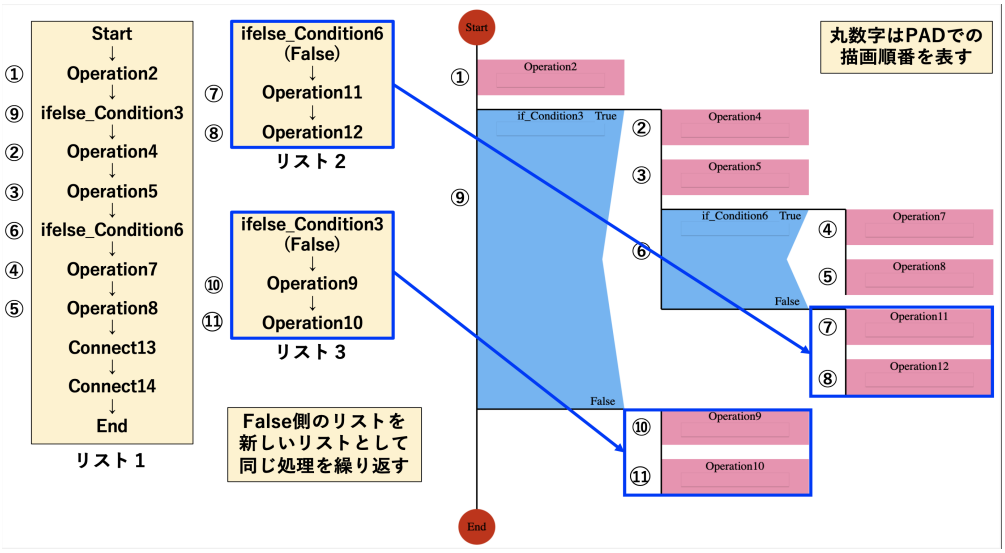


図 4.21　条件分岐処理 (if-else) を含むリストから PAD 変換例

4.4　PAD から擬似プログラムの作成

PAD DOM から疑似コードを作成する方法について説明する．本システムで出力する疑似コードは C 言語をベースとしている．PAD 要素と疑似コードとの対応を表 4.8 に示す．なお, PAD の深さが浅くなる時点で文字「}」を出力する．また, PAD 要素の名称が “Loop_after” の場合には, その時点は「do {」のみを出力し, “Loop_after” の実行部分が終了した時点で残りの「} while( )」を出力する．

表 4.8　PAD 要素と疑似コードとの対応

PAD 要素の名称	疑似コード
Start	void main() {
Operation	operation(　　);
Loop_before(回数)	for(　　) {
Loop_before(前条件)	while(　　) {
Loop_after	do {　　} while(　　)
if_Condition True	if(　　) {
if_Condition False	else {
End	}

PAD から擬似コードへの変換順序について説明する．PAD 自体が横方向が処理の深さを表し，縦方向が処理量を表すと構造化モデルであるため，プログラム言語の構造と関係が深い．特に，PAD の横方向に連接する要素は，プログラム言語では連続した入れ子状態に対応する．そこで，PAD の Start 要素から変換処理を開始し，各要素を幅優先で辿りながら疑似コードに変換する．図 4.22 に入れ子になったループ処理を含む PAD から疑似コードへの変換した例を示す．図内の丸数字が疑似コードへの変換順番を表す．PAD の深さに応じて，疑似コードのインデントも変化させている．なお，PAD 要素のテキストボック内の文字列は，疑似コード内ではコメント行 (/**/で囲まれた部分) として出力している．

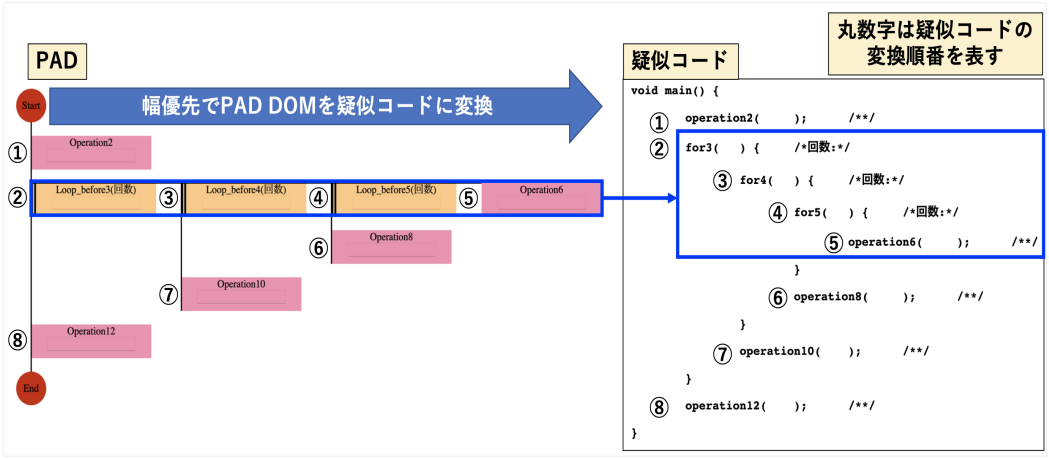


図 4.22　ループ処理を含む PAD から疑似コードへの変換例

図 4.23 に条件分岐処理 (if) を含む PAD から疑似コードへの変換例，図 4.24 に条件分岐処理 (if-else) を含む PAD から疑似コードへの変換例を示す．PAD の各要素を幅優先で辿ることで，疑似コードを上から順に出力できていることが分かる．

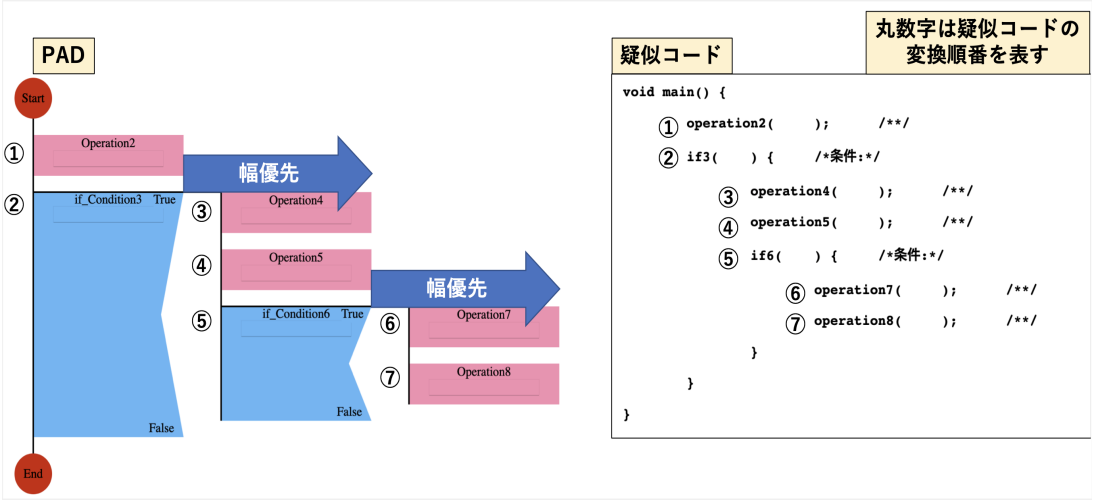


図 4.23　条件分岐処理 (if) を含む PAD から疑似コードへの変換例

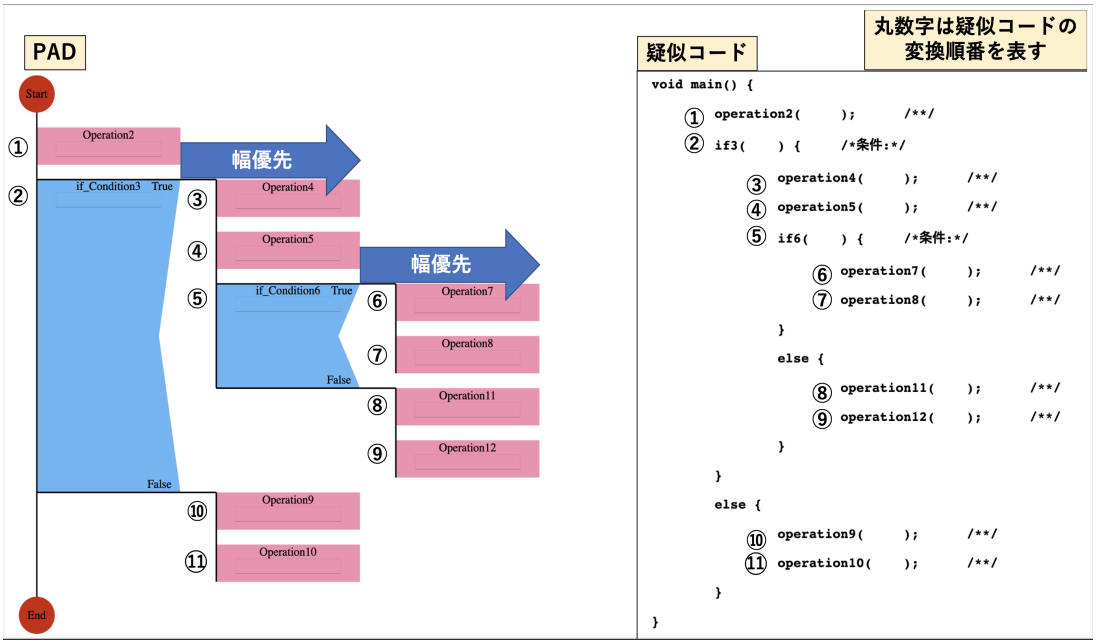


図 4.24　条件分岐処理 (if-else) を含む PAD から疑似コードへの変換例

4.5　複雑なフローチャートに対する変換

PAD と疑似コードは親和性が高いので PAD から疑似コードへの変換は容易である．一方，フローチャートの欠点を補ったのが PAD であり，双方の構造が異なるためにフローチャートから PAD への対応は工夫が必要になる．本システムでは，4.3 でも説明したように if 文の False 側の系列に対しては変換関数を再起呼び出しすることで対応している．そのため，入れ子が深くなったり，if 文とループ処理が組み合わさっても正しく PAD に変換することができる．図 4.25 に複雑なフロチャートの例を示す．図 4.26 にそれを変換した PAD，図 4.27 に対応する疑似コードを示す．

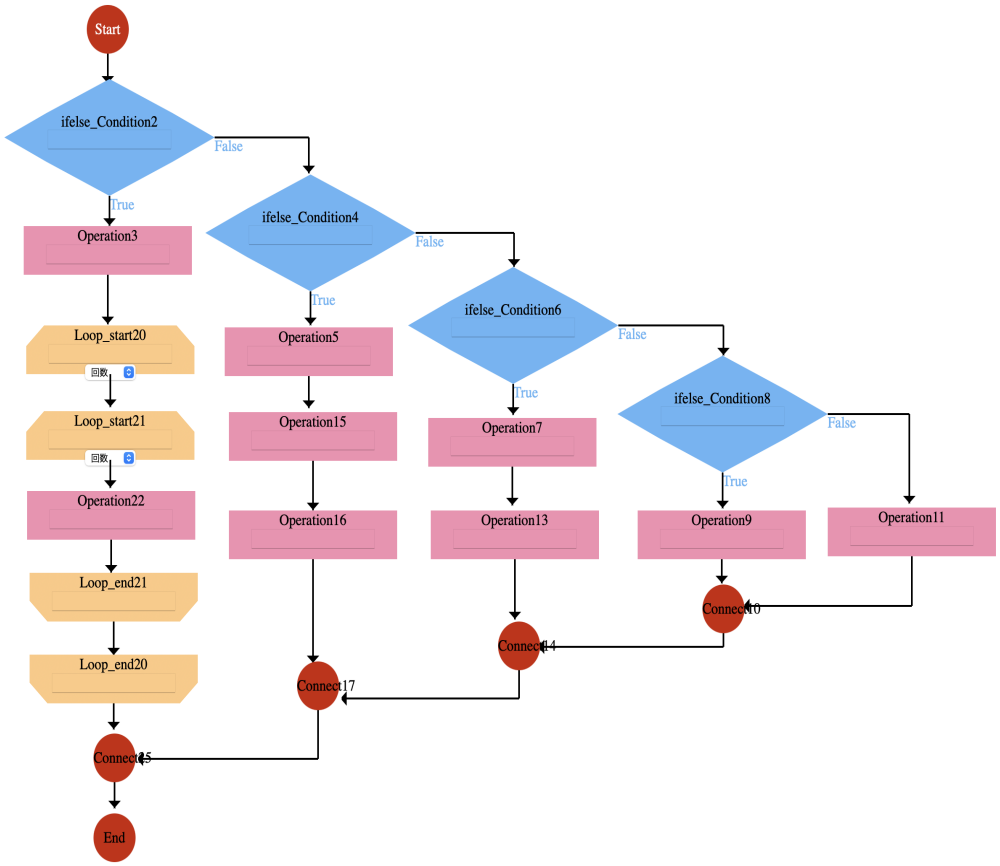


図 4.25　複雑なフローチャートの例

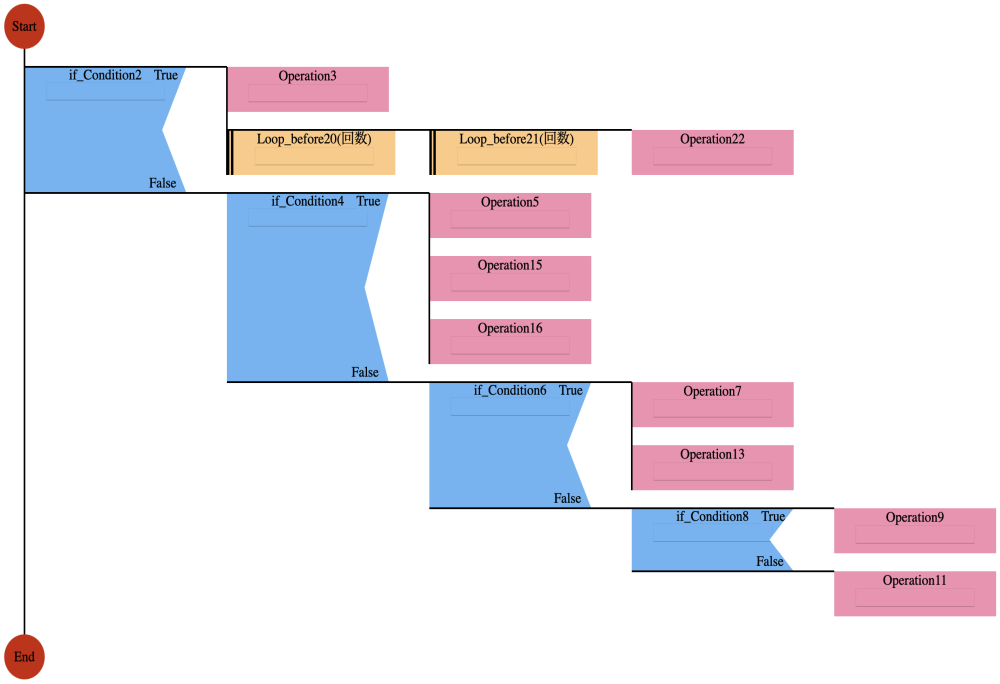


図 4.26　図 4.25 のフローチャートを変換した PAD

```
void main() {
        if2(    ) {     /*条件:*/
                operation3(     );      /**/
                for20(  ) {     /*回数:*/
                        for21(  ) {     /*回数:*/
                                operation22(    );      /**/
                        }
                }
        }
        else {
                if4(    ) {     /*条件:*/
                        operation5(     );      /**/
                        operation15(    );      /**/
                        operation16(    );      /**/
                }
                else {
                        if6(    ) {     /*条件:*/
                                operation7(     );      /**/
                                operation13(    );      /**/
                        }
                        else {
                                if8(    ) {     /*条件:*/
                                        operation9(     );      /**/
                                }
                                else {
                                        operation11(    );      /**/
                                }
                        }
                }
        }
}
```

図 4.27　図 4.26 の PAD を変換した疑似コード

第 5 章

評価と考察

5.1 評価方法

本研究で構築した授業支援システムは, プログラムを書けない, あるいは苦手意識のある教師が授業で使用することを想定しているため, 教職関係者, 教員免許状取得者, 家庭教師や塾講師などの形で児童・生徒と関わっている者 (教育関係者) を対象者としてアンケートを実施した. また, 構築したシステムの利便性や挙動に関しても評価するため, プログラムの知識がある情報系の学生 (プログラミング経験者) にもアンケートを実施した. アンケート手順は以下の通りである.

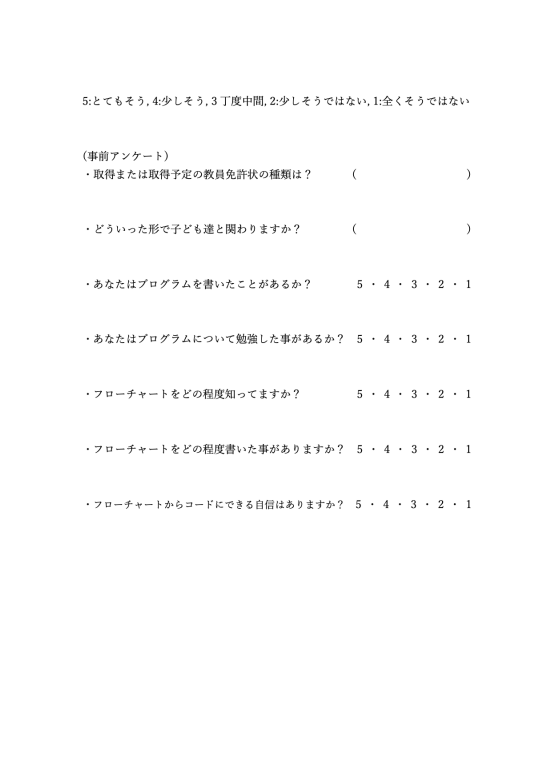
1. 事前アンケートに回答する.
2. 事前質問を回答する.
3. 本システムの目的, 意図, 使い方, 仕様の説明を理解した上で, 質問を回答する.
4. 事後アンケートに回答する.

アンケート用紙は全部で 4 ページ分あり, 図 5.1 から図 5.4 にアンケートの内容を示す. 図 5.1 の事前アンケートで, 回答者が教育関係者であるか, そうでないかを区別した. また, 回答者のフローチャートやプログラムに対する理解度を事前に尋ねた上で, 図 5.2 の事前質問では, 回答者のフローチャートやコードに対する知識がどの程度あるのかを知る為に, 幾つかの問題を用意した. 図 5.3 は本研究で構築した授業支援システムを提示した上で, PAD を通じてどのくらい手順化に役立っているかを確認する為に, 図 5.2 と似た類題を準備した. 図 5.4 の事後アンケートでは, 本システムを通じてフローチャートとプログラムの理解のし

5.1　評価方法

やすさ, どういった形で学校現場で役に立つか, 感想や意見等を尋ねた.

5.1 評価方法



5:とてもそう, 4:少しそう, 3 丁度中間, 2:少しそうではない, 1:全くそうではない

(事前アンケート)
・取得または取得予定の教員免許状の種類は？　　(　　)

・どういった形で子ども達と関わりますか？　　(　　)

・あなたはプログラムを書いたことがあるか？　　5 ・ 4 ・ 3 ・ 2 ・ 1

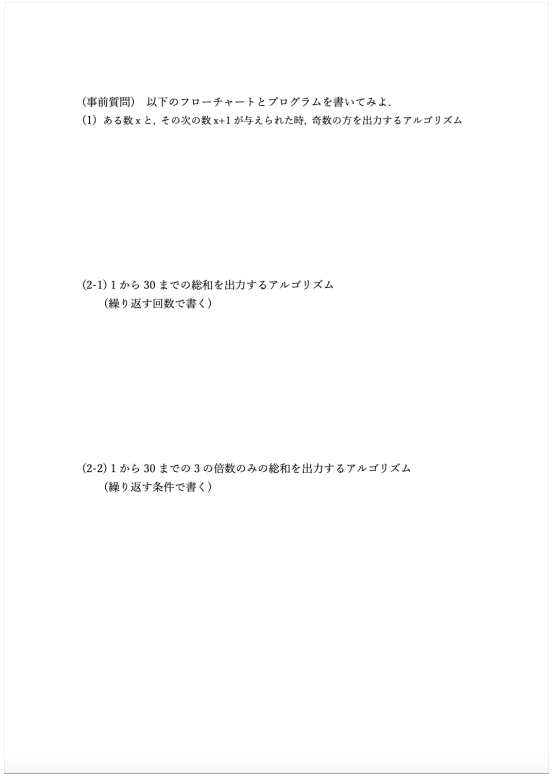
・あなたはプログラムについて勉強した事があるか？　5 ・ 4 ・ 3 ・ 2 ・ 1

・フローチャートをどの程度知ってますか？　　5 ・ 4 ・ 3 ・ 2 ・ 1

・フローチャートをどの程度書いた事がありますか？　5 ・ 4 ・ 3 ・ 2 ・ 1

・フローチャートからコードにできる自信はありますか？　5 ・ 4 ・ 3 ・ 2 ・ 1

図 5.1　1 ページ目 (事前アンケート)

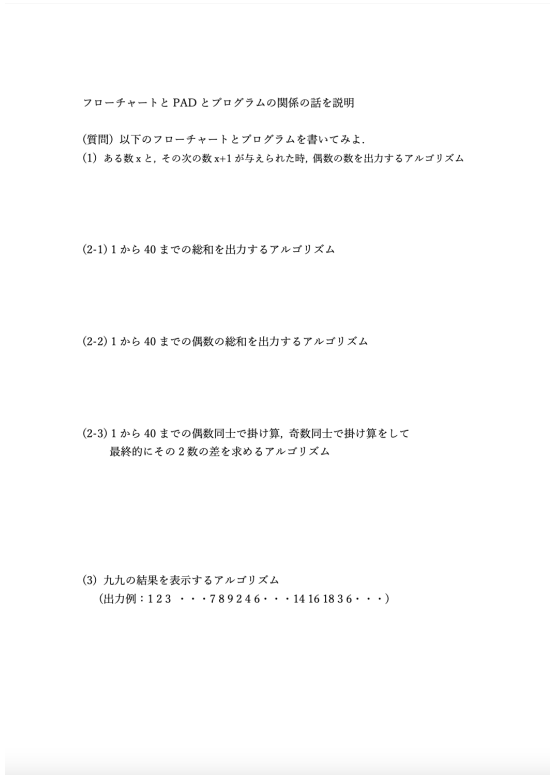


(事前質問)　以下のフローチャートとプログラムを書いてみよ.
(1) ある数 x と, その次の数 x+1 が与えられた時, 奇数の方を出力するアルゴリズム

(2-1) 1 から 30 までの総和を出力するアルゴリズム
　　(繰り返す回数で書く)

(2-2) 1 から 30 までの 3 の倍数のみの総和を出力するアルゴリズム
　　(繰り返す条件で書く)

図 5.2　2 ページ目 (事前質問)



フローチャートと PAD とプログラムの関係の話を説明

(質問) 以下のフローチャートとプログラムを書いてみよ.
(1) ある数 x と, その次の数 x+1 が与えられた時, 偶数の数を出力するアルゴリズム

(2-1) 1 から 40 までの総和を出力するアルゴリズム

(2-2) 1 から 40 までの偶数の総和を出力するアルゴリズム

(2-3) 1 から 40 までの偶数同士で掛け算, 奇数同士で掛け算をして
　　最終的にその 2 数の差を求めるアルゴリズム

(3) 九九の結果を表示するアルゴリズム
　(出力例：1 2 3 ・・・7 8 9 2 4 6・・・14 16 18 3 6・・・)

図 5.3　3 ページ目 (質問)

(事後アンケート)
・フローチャートの仕組みは理解できましたか？　　5 ・ 4 ・ 3 ・ 2 ・ 1

・フローチャートとプログラムとの関係は分かりましたか？　5 ・ 4 ・ 3 ・ 2 ・ 1

・PAD があるとプログラムは理解しやすいですか？　5 ・ 4 ・ 3 ・ 2 ・ 1

・このシステムは授業準備に役立つと思いますか？　5 ・ 4 ・ 3 ・ 2 ・ 1

・このシステムは授業中に役立つと思いますか？　　5 ・ 4 ・ 3 ・ 2 ・ 1

・このシステムは授業以外でも役立つと思いますか？　5 ・ 4 ・ 3 ・ 2 ・ 1

・このシステムを積極的に使いたいと思いますか？　5 ・ 4 ・ 3 ・ 2 ・ 1

・どのようにすると, より良いシステムになるか？
　感想や意見も含めて自由に書いてください.

図 5.4　4 ページ目 (事後アンケート)

5.2　アンケート集計結果

5.2.1　教育関係者を対象とした集計結果

回答者は 10 名であった. 具体的には, 本大学の教職経験者, 免許取得 (見込み) 者, 塾でアルバイトしている大学生, 塾長, 大学の教授である. 教育関係者の中で, 高校情報の免許を保有している学生は 2 名であった. 以下, 事前アンケートの集計結果を表 5.1 に, 事後アンケートの集計結果を表 5.2 に示す.

表 5.1　教育関係者 (事前アンケート)

(教育関係者_事前質問)	5 (とてもそう)	4 (少しそう)	3 (丁度中間)	2 (少しそうではない)	1 (全くそうではない)
あなたはプログラムを書いたことがあるか？	3	3	2	1	1
あなたはプログラムについて勉強した事があるか？	3	3	3	0	1
フローチャートをどの程度知っていますか？	1	3	2	2	2
フローチャートをどの程度書いた事がありますか？	0	3	0	4	3
フローチャートからコードにできる自信はありますか？	0	1	2	3	4

表 5.2　教育関係者 (事後アンケート)

(教育関係者_事後アンケート)	5 (とてもそう)	4 (少しそう)	3(丁度中間)	2 (少しそうではない)	1 (全くそうではない)
フローチャートの仕組みは理解できましたか？	8	1	1	0	0
フローチャートとプログラムの関係は分かりましたか？	5	4	1	0	0
PADがあるとプログラムは理解しやすいですか？	7	3	0	0	0
このシステムは授業準備に役立つと思いますか？	6	4	0	0	0
このシステムは授業中に役立つと思いますか？	6	3	1	0	0
このシステムは授業以外でも役立つと思いますか？	7	2	1	0	0
このシステムを積極的に使いたいと思いますか？	4	4	1	1	0

また, 感想や意見を記述する項目では以下の事柄が挙げられた.

- PAD の方が確かに分かりやすかった.
- 行う内容と処理が構造的で分かりやすかった.
- 生徒に教える前に教師がフローチャートとプログラムを理解して共通認識を持つことが必要だと思うので, 全ての人にとって分かりやすく無駄のないシステムにすればいいと考える.
- フローチャートからプログラムよりも, プログラムからフローチャートの方が使いやすかった.

- フローチャートから, プログラムにする際に型 (int, boolean 等) が分からない人もいるのではないかと感じた.
- PAD の見方も一緒に書いていると自学自習に役立ちそうと感じた.
- フローチャートは分かるが, 言語で書くことは難しいと感じた.
- フローチャートを理解した上でのツールでしょうけど, 矢印があることでの分かりやすさもあるのではないかと思った.
- フローチャートを理解すると, 論理的な思考ができるようになると思う. 深く考えることにも繋がるし, 数学の理解も深まりそうと思った.
- マウスオンで対応する部分がハイライトされるとわかりやすい.
- 入れ子が多くなって右に深くなるとスクロールが必要になるので, ズームインアウトやドラッグで画面をずらせると良い.
- コードからフローチャートへ逆変換する機能が欲しい.
- あり得ない (エラーの) フローチャートを入力した時のシステムの出力はどうなりますか?
- フローチャートが長くなるとドラッグで作成するのが煩わしいので, 教科書に載っているフローチャートを写メって, その画像からフローチャートに変換できる機能が欲しい.

5.2.2　プログラミング経験者を対象とした集計結果

回答者は 17 名であった. 具体的には, 本大学院の情報学コースの学生, 他大学でプログラミングを専門としている学生である. 以下,　事前アンケートの集計結果を表 5.3 に, 事後アンケートの集計結果を表 5.4 に示す.

表 5.3　プログラミング経験者 (事前アンケート)

(プログラミング経験者_事前質問)	5 (とてもそう)	4 (少しそう)	3 (丁度中間)	2 (少しそうではない)	1 (全くそうではない)
あなたはプログラムを書いたことがあるか？	14	1	2	0	0
あなたはプログラムについて勉強した事があるか？	13	2	2	0	0
フローチャートをどの程度知っていますか？	3	9	5	0	0
フローチャートをどの程度書いた事がありますか？	0	8	5	4	0
フローチャートからコードにできる自信はありますか？	2	4	7	4	0

5.2 アンケート集計結果

表 5.4 プログラミング経験者 (事後アンケート)

(プログラミング経験者_事後アンケート)	5(とてもそう)	4(少しそう)	3(丁度中間)	2(少しそうではない)	1(全くそうではない)
フローチャートの仕組みは理解できましたか？	16	1	0	0	0
フローチャートとプログラムの関係は分かりましたか？	15	2	0	0	0
PADがあるとプログラムは理解しやすいですか？	9	6	0	2	0
このシステムは授業準備に役立つと思いますか？	15	2	0	0	0
このシステムは授業中に役立つと思いますか？	14	3	0	0	0
このシステムは授業以外でも役立つと思いますか？	8	5	4	0	0
このシステムを積極的に使いたいと思いますか？	7	7	2	1	0

また, 感想や意見を記述する項目では以下の事柄が挙げられた.

- 2 重ループや条件分岐のようなプログラムにおいては処理の視覚的な配置がプログラムに近く, 直感的に理解しやすかったです. より複雑なプログラムにおいても直感的な理解しやすさが維持できるのか気になります.
- PAD から擬似コードを生成する際は, 処理の流れをそのままコードに落とし込むイメージでフローチャートより短い時間で開発できそうなので, フローチャートより PAD からコードを生成する方が良い.
- PAD があると理解しやすいかという点では, フローチャートで充分という感もあるが, 考えをまとめる材料として利用できるため良いかもしれない.
- 授業などでは, このシステムがある方が理解しやすい生徒もいるだろう.
- 簡単なプログラムであれば使いやすいが, もっと複雑なプログラムになると余計に分かりづらそうと思ってしまった. ただ, この場合も使いやすい人にとっては使いやすいシステムだと思った.
- フローチャートよりも理解しやすいと思った.
- もっと複雑なプログラムの時の PAD も見てみたい.
- PAD が馴染みがないので, フローチャートから直接疑似コードが出るとより分かりやすい.
- フローチャートからコードが分かるのは良かった.
- 逆にコードからフローチャートができればより使いやすい.
- 逆方向の変換もできると, より良いシステムになると思った.

- 1 年次の初めに使用できれば, より理解できると思う．また，苦手意識が解消されると思う.
- PAD と疑似コードの構造が似ていて, あまり違いが分からなかった.
- パワポみたいにノードなどの複製などが出来るのか疑問.
- PAD の表示は処理順に番号を入れるか, ハイライトをつけるなどしないとフローチャートより処理順が分かりにくいと感じた.
- 個人的には, デバッグ時のように, 変数の値や出力値を見えるようにした方が処理の中身が分かりやすいと思う.
- 他の言語でも疑似コードを出力できるようにすれば良いと感じました.
- 長めのフローチャートがぱっと見で理解しやすくなっていると感じた.
- 個人的には if 文はフローチャートの方が分かりやすいと感じた.
- 疑似コードが一緒に出力されるのは魅力的だった.
- プログラミング初心者であれば，コードを考える・書くことは一苦労．フローチャートが出来れば, まだ書き易いかもしれないが, フローチャートも作り慣れていなければ難しいため, 直感的に操作しながらプログラミングをすることはプログラムを学ぶ上で非常に役に立つものだと感じた.
- やっぱり自分が情報系で, ある程度経験があるので, もっと複雑なプログラムになると逆に分かりづらいのではないかと思う．なので, プログラミングの初心者にはコツを掴むきっかけになると思うので, 導入には適していると思った.
- 構文の種類で色を変えているのが分かりやすかった.
- 逆にプログラムからフローチャートを作成するものがあっても良いかと感じた.
- プログラム初心者には, こういったフローチャート変換システムがあった方がイメージを掴みやすく学びやすいと思ったので, 非常に良いシステムだと感じました.
- if 文の T or F, for 文の i の値を指定することで, 実行されるフローチャートの流れがどうなるのか, また, 疑似コードのどの部分が実行されるのかなどが視覚的に分かりやすいと, 1 年生などのプログラム初期の人が使いやすいのではないかと思いました.

- 複数あるモジュールの処理を頭の中だけでイメージすることは難しいと感じていましたが, PAD によりモジュール毎の処理を図解化することで, プログラム全体の流れやアルゴリズムが把握しやすくなると感じました.
- 関数やクラスなども使えたら, より良いシステムになると思った.
- オブジェクト指向の重要さを伝えることができそうと思った.

5.3 考察

5.3.1 教育関係者を対象とした考察

本節では, 教育関係者の感想や意見を基に考察を行う. アンケートでは「フローチャートと比べて PAD の方がわかりやすい」「フローチャートより PAD の方が行う内容と何をどのように繰り返すのかという処理が構造的で分かりやすかった」「PAD の見方も一緒に書いていると自学自習に役立ちそうと感じた」という意見が得られた. これは, フローチャートの繰り返し構造は全て同じノードで表現しているが, PAD で記述する際は 3 種類のループを明示的に表現する必要がある. そして, フローチャートとプログラムの両方を理解するための手がかりとして PAD が役立っている. 教科書レベルでは PAD は取り扱われていないが, PAD についての知識があまり無くても理解でき, 自学自習に役立つ可能性がある. また, 表 5.1 と表 5.2 によると, 事前質問でフローチャートがどんな仕組みか知らなかった被験者が本システムを通じてフローチャートの仕組みを理解できたとの回答が多かった. このように, ポジティブな意見が多く見受けられたことから, プログラミング経験のない教師でもフローチャートから PAD を介して手順化支援できることが示唆された.

また, アンケートのコメントから「コードからフローチャートへ逆変換する機能が欲しい」との意見があった. プログラムが無くフローチャートだけが示される場合は大学入学共通テスト等でも想定される. これに対し, 本システムを使うことで, フローチャートが書ければ一つの解に近づけるためのツールとして使える. しかし, プログラムからフローチャートを書くような問題も将来的に出題される可能性がある. 従って, 今後の課題としてプログラムが

書ければ, フローチャートを出力するシステムを作成するシステムの必要性が示唆される.

さらに, 本システムのインタフェースについて, 「マウスオンで対応する部分がハイライトされるとわかりやすい」「入れ子が多くなって右に深くなるとスクロールが必要になるので, ズームインアウトやドラッグで画面をずらせると良い」「あり得ない (エラーの) フローチャートを入力した時のシステムの出力」との意見があった. これは, 今後の実装で取り組むべき課題である. エラー処理も考慮できておらず, 間違ったフローチャートを作成した際にはエラー箇所を指摘する, または実行できないようにするなどを検討する必要がある.

5.3.2 プログラミング経験者を対象とした考察

本節では, プログラミング経験者の感想や意見を基に考察を行う. アンケートでは「逆にコードからフローチャートができればより使いやすい」「逆にプログラムからフローチャートを作成するものがあっても良いかと感じた」との意見があった. 教育関係者の意見と同様に逆変換機能できるシステムが欲しいという意見が見られた. また, アンケートでは「フローチャートよりも PAD が理解しやすいと思った」「PAD から擬似コードを生成する際は, 処理の流れをそのままコードに落とし込むイメージでフローチャートより短い時間で開発できそうなので, フローチャートより PAD からコードを生成する方が良い」との意見があった. プログラミング経験者は制御構造から考える癖が身に付いているため, フローチャートから考えること自体が逆に難しい経験者が少なからず存在することがわかった. 従って, フローチャートを見てすぐに制御構造が思い浮かぶ人には本システムを使う必要ないが, フローチャートを見て困惑する人には PAD を介した本システムを使う価値があることが示唆された.

5.3.3 全体的な考察

事前アンケートや事前質問の回答状況から, フローチャートを理解している教育関係者が極端に少なかった. 本システムは, フローチャートが書けることを前提として, フローチャー

トからプログラムを書くことができない教員が使用することを想定していたため, 教育関係者にはフローチャートの説明を最初に行わなければならないケースが多かった. 従って, 教育関係者であっても, 情報科を指導しない教師であれ, フローチャートの理解が困難だと考えられ, 本システムを使用する前にフローチャートを理解できる支援を行う必要がある. そして, 本システムを使うことによってフローチャートの理解にも役立てることが分かった. 更に, PAD の構造とアルゴリズムとの対応関係を理解することで, 論理的思考力も身に付くため, 他教科にも活かせる可能性がある.

プログラミング経験者に関して, 表 5.3 によると, プログラムを書く事に対して自信を持っている人は, 逆にフローチャートを書くことに不安感を抱いていることが伺えた. この状況は, プログラミング経験者のみではなく, 教育関係者であっても起こりうるケースである. 疑似コードは記述できるが, フローチャートの仕組みが分からず記述できない教師のために, 疑似コードから構造化チャートを介してフローチャートが理解できるシステムを作成することも効果的であると示唆された.

本研究で扱った構造化チャートは PAD のみだが, 構造化チャートは一種類のみではない. PAD の理解が難しい教師のために, PAD とは別の構造化チャート用いて, 同じような授業支援システムを作成することも検討する必要がある.

また, 本システムは情報が必修化となり, 入学試験で「情報」を受験する高校生を指導する教師のみでなく, プログラムを勉強する大学においても, 1, 2 年生の講義の授業準備や自学自習にも使用できる可能性も示唆された.

プログラミングに慣れていない先生や生徒にとって, PAD があることによってフローチャートの難しい部分やコードの難しい部分を繋ぎ, 理解する手がかりとなる. 本システムを作成する前は, PAD からコードを書くのに役立つという話だったが, コードを書くだけでなく, フローチャートを理解することにも役立つと考えられる. フローチャートからコードを書くことも難しく, コードからフローチャートを書くのも難しい. このような状況に対して, PAD を介することでコードを理解することにも繋がり, フローチャートを理解するのにも繋がることが示唆された.

第 6 章

おわりに

本研究では, フローチャートから PAD を介して擬似コードを理解できることを想定し, プログラミング経験のない教師が授業で使用できる授業支援システムを構築した. 構築したシステムは, フローチャートから手順を生成するのでは無く, フローチャートから構造化チャートを表しつつ, 手順を生成できるようにすることで, データとプログラミングの流れを明確に意識して授業を行うことができるようにした.

本システムに関するアンケート評価の結果, プログラミング経験のない教師でもフローチャートから PAD を介して手順化支援できることが示唆された. プログラミングに慣れていない先生や生徒にとって, PAD があることによってフローチャートの難しい部分や手順化の難しい部分を繋ぎ, 理解する手がかりとなる. フローチャートからコードを書くことも難しく, コードからフローチャートを書くのも難しいといった状況に対して, PAD を介することで手順を理解することにも繋がり, フローチャートを理解するのにも繋がることが示唆された.

今後の展望としては, フローチャートの知識がない教師に対して, フローチャートの理解を支援するシステムを構築する必要がある. また, 疑似コードは記述できるが, フローチャートが記述できない教師のために, 疑似コードから構造化チャートに変換し, そしてフローチャートを生成するシステムを作成する必要性が示唆された. 今後は, 一人でも多くの教師がプログラミングに対して苦手意識を解消し, プログラミングに不安を抱いている教師が本システムを利用するようになるために, 評価結果や感想・意見を基に, 修正や必要な機能の追加を検討する.

謝辞

本研究および本論文に関して, ご多忙な中, 多大なるご指導賜りました高知工科大学情報学群妻鳥貴彦准教授に心より御礼申し上げます.

本研究において, ご多忙な中, 副査をお引き受け頂き, 適切な助言, ご指導頂いた同学群高田善朗教授, 竹内聖悟講師に心から感謝いたします.

本研究の遂行において, 様々なご協力して頂いた教職関係者の方々に心から感謝いたします.

最後に, 私を支えてくれた友人, 大学生活の全てにおいて支えとなって頂いた両親に心から感謝いたします.

参考文献

[1] 文部科学省, “小学校段階におけるプログラミング教育の在り方について (議論の取りまとめ)”, https://www.mext.go.jp/b_menu/shingi/chousa/shotou/122/attach/1372525.htm

[2] 青森彩, “プログラミング的思考に着目した授業支援システム”, 高知工科大学学士学位論文, 2019.

[3] 全国 KOSEN 支援機器開発ネットワーク, “高等学校共通教科情報科の変遷と課題”, 情報処理 Vol59. No.10 Oct. 2018 https://www.ipsj.or.jp/magazine/9faeag0000005al5-att/5910peta.pdf

[4] 独立行政法人大学入試センター, “平成 30 年度告示高等学校学習指導要領に対応した令和 7 年度大学入学共通テストからの出題教科・科目 情報 サンプル問題”, https://www.dnc.ac.jp/albums/abm.php?d=33&f=abm00000307.pdf&n=%E3%82%B5%E3%83%B3%E3%83%97%E3%83%AB%E5%95%8F%E9%A1%8C%E3%80%8E%E6%83%85%E5%A0%B1%E3%80%8F_%E5%95%8F%E9%A1%8C.pdf

[5] 讀賣新聞, “高校「情報」必修 専門教員不足の解消が急務だ”, https://www.yomiuri.co.jp/editorial/20210604-OYT1T50319/

[6] 李月・尾崎誠・但馬文昭, “初心者を対象とした C 言語プログラミング教育における学習法改善のための基礎的研究ーフローチャートと PAD の学生に対する学習支援の効果比較ー”, 教育デザイン研究第 11 号 -2020 年 1 月- (横浜国立大学紀要論文).

[7] 斐品正照・徳岡健一・河村一樹, “構造化チャートを用いたアルゴリズム学習支援システム”, 情報処理学会論文誌, vol.45 No.10 pp.2454-2467 Oct. 2004.

[8] 文部科学省, ‘高等学校情報科「情報 I」教員研修用教材’, p114-118, https://www.mext.go.jp/content/20200722-mxt_jogai02-100013300_005.pdf

[9] Wikipedia, “PAD (ソフトウェア設計)”, https://ja.wikipedia.org/wiki/PAD_(

ソフトウェア設計）