プログラミング教育を
支援する教材に関する研究

後藤　孔

目次

第1章　はじめに

近年，文部科学省はSociety5.0時代の学びの実現に向けて抜本的な教育改革案 [1] を打ち出しており，その一部としてプログラミング教育に関する言及も行なっている．具体的には，発達段階に応じたプログラミング教育の充実を図るとの記述があり，まず小学校ではプログラミングの必修化，つぎに中学校ではプログラミングに関する内容の充実，そして高等学校では全ての生徒がプログラミングのほか，それと関連するネットワークや情報セキュリティなどの情報技術も学習することが求められている．また，大学では，文理を問わず全ての学生に数理・データサイエンス教育を行うことが求められている．

このような状況でプログラミングを学習する対象者の年齢層が広がると，様々な発達段階に応じた教材が必要となり，指導者の負担が大きくなる．プログラミング教育に関して，以前から様々な教材 [2–20] が開発されているが，プログラミングの指導者を積極的に支援することを目指したものはほとんど見当たらない．

従来，プログラミングは主に情報系や工学系の学生あるいは情報技術を扱うエンジニアらに限られてきた．プログラミング教育を行う側も大学教員や一部の専門家に限られてきた．一方，昨今は急速な社会変革に呼応してプログラミングの学習者が小学生から大学生まで広がっている．その結果，プログラミングを指導する教員が学習者の発達段階に応じて教材を準備し，授業を行う必要性が生じている．そのような指導者の育成は不十分であり，積極的な支援が求められている．

たとえば，学習者がプログラムをコンパイルする際に，コンパイルエラーに遭遇してデバッグを自力では行えない場合，それを支援するのは指導者の役割である．指導者にとっては，プログラミングの初心者が多数受講する授業において質問が殺到すると，個別対応に追われることになる．このような場合，プログラミングが専門の教員でも授業を円滑に進めるのは困難である．

また，プログラミングの学習者にとって，プログラミングの作業につまずくと，本来の学習目的が達成されなくなる．たとえば，ユークリッド互除法のアルゴリズムをC言語で実装するという課題を授業で扱う場合，主な学習目的はユークリッド

互除法そのものの理解や C 言語の書き方であることが多いが，C 言語のような文字ベースのプログラミング言語を使用すると初学者はコンパイルエラーに遭遇することが多く，そのエラーを自力で解消することも難しい．コンパイルエラーやコンパイラそのものについての学習は，プログラミングの入門的授業においては主な学習目的に含まれないことが多い．そのため，学習者がエラーを自力で解消できないことも多く，アルゴリズムの理解やプログラミングに十分な時間を割くことができなくなる．

ところで，プログラミング教育の対象が広がっていることを考慮すると学習者が自ら興味を持って学べるような工夫も重要である．そこで，身近な技術を題材としてプログラミング教育に活かすことが考えられる．たとえば，音声情報処理技術を利用すれば生徒の声で機器の操作を行うようなことをプログラミングを通して体験することが可能である．

そこで，本研究では学習者と指導者の双方に有用なプログラミング教材を開発する．具体的には，上記に述べた問題を踏まえ，文字ベースのプログラミング言語を用いたプログラミング授業に有用なデバッグ支援システム [21] と身近な技術として音声情報処理技術を活用したプログラミング教材 [22] を開発する．

以下では，第 2 章でプログラミング教育における学習者と指導者の抱える問題点について述べる．その後，本研究で開発したプログラミング教材について述べ，第 3 章ではデバッグ支援システムについて，第 4 章で音声情報処理技術を用いたプログラミング教材について述べる．そして第 5 章で開発教材の考察を行い，第 6 章で本論文のまとめを述べる．

第2章　プログラミング教育における問題点

2.1　プログラミング教育の広がり

以前はプログラミングを学ぶ対象が技術者などに限定され，多くの人々には無縁のように思われることが多かったが，教育改革，政府の答申，さらには昨今の国際社会や情報社会における需要などを背景として，初等教育においてもプログラミング学習が必修の学習内容とされるようになり，プログラミングを学ぶ対象が子どもから大人まで広がっている．その結果，これまでは単にコンピュータの利用方法を学ぶだけであった人もプログラミングを学ぶことになり，プログラミング教育に携わる側も，それに応じた工夫が必要とされている．

初等教育においてはプログラミング教育が必修化され，専門の教科を新設するのではなく，各教科と融合する形か，総合的な学習の時間を活用してプログラミングの授業が行われる．初等教育におけるプログラミング教育の目標は「子供たちに，コンピュータに意図した処理を行うよう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』などを育むこと」[23] とされる．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」[23] であり，小学生にプログラミングを学ばせるのは情報技術が今後ますます社会に浸透し，人々の生活を便利にする時代においては，技術そのものを享受するだけでなく，その仕組みや構造を理解して，自身の人生や新たな社会構築に生かすための能力を身につけさせるためである．

中等教育においては，中学校技術科で，計測・制御およびネットワークの技術を関連させたプログラミングに加え，人工知能などの新しい情報技術を扱うことが求められている．平成 29 年告示の学習指導要領解説に基づくと，技術科の情報の技術

の指導項目では，人工知能などの新しい情報技術の利用方法や優れた点などを利用者と開発者の両方の立場から考えさせるという学習例が提言されている．利用者として技術を評価するには，既製品の調査や使用の範囲で良いが，開発者として技術を評価するためには，ものづくりを通した学習が重要と考えられる．また，高等学校情報科では，従来の情報科目「社会と情報」と「情報の科学」が改訂され，科目名称を「情報 I」として新しく再編された．情報 I の学習内容は平成 30 年 7 月改訂の高等学校学習指導要領解説情報編において記載されており，たとえば「コンピュータとプログラミング」という学習項目の中には，プログラミングの学習を通じて「情報通信ネットワークを活用する方法について理解し技能を身に付けること（抜粋）」という学習事項が定められている．

高等教育においては，たとえば大学で数学を専攻する学生はプログラミングを活用してグラフを描画し，また，心理学を専攻する学生はプログラミングによってデータを統計的に処理している．さらに，教育学部で教員志望の学生も将来の授業に向けて，プログラミングを学習する機会が増えている．このように，文理を問わず様々な学問を専攻する学生がプログラミングを学ぶようになっている．

要するに，昨今のプログラミング教育は，従来目的とされていた技術者育成の観点ではなく，様々な年代の人々がプログラミングを学習する教養の観点で実施する時代へ移行している．この結果，学習者にとっては自身のプログラミングに対する興味や得意度に関わらず学習を余儀なくされることも生じ，さらにその結果，指導者にはそのような学習者を指導しながら，授業を円滑に進行しなければならないなどの問題が生じる．このような状況においては，学習者に対して効率的で，かつ，学習意欲を高める教材の提供が望まれ，指導者に対しても授業の負担を軽減し，理想の授業を実施できるよう支援することが重要である．

2.2　学習者の問題点

プログラミングを学習する際，まずはプログラミングに用いる言語を決定する必要があるが，近年は，従来の文字入力に基づく方法でプログラムを作成する言語だけではなく，画面操作に基づいてプログラムを作成する，ビジュアルプログラミング言語も普及している．本研究では，２種類の言語をプログラミング教材として扱う上で，次に述べるような学習者にとっての問題点に着目している．

文字入力の方式によるプログラミング言語は，入力する文字を 1 文字でも誤るとプログラムは正常に動作しないことが多く，その場合はプログラムが正常実行でき

る状態となる代わりにエラーメッセージがコンパイラなどから出力される．出力されたエラーメッセージはプログラミングに関する専門用語を含んだ意味合いの内容が多く，プログラミングの学習者は指導者の支援なしには意味を解釈することすら容易でない．これ以前に，多くのプログラミングの授業においては，プログラムの作成方法や実行方法を教授することなどが授業の主な内容であり，教育目標から鑑みてもプログラムのエラーメッセージの解釈に指導時間を占めることは難しく，授業で指導していない以上，学習者が自力でエラーメッセージの解釈が出来ないことは想定されることである．

要するに，学習の道具として必要ではあるが，本来の教育目標や授業目標を鑑みると学習者が密接になる必要がない内容によって，学習が阻害されることは問題といえる．

ビジュアルプログラミング言語を用いて行うプログラミング [3] では，作成したプログラムが意図通りに動作しないことはあっても，文法誤りによるエラーが発生しないため，文字入力が困難な小学生などを対象とした授業では好都合である．ただし，ビジュアルプログラミング言語は実用的なプログラムを作成するために設計されたものではなく，作成可能なプログラムの種類に制限がある．

ビジュアルプログラミング言語に特化したことではないが，プログラミングの授業においては，一般的な演習題材としてプログラムの動作確認に終始したものもしばしば扱われる．たとえば，反復の概念を指導するため，繰り返しの機能を持つ命令を用いて，０から１０までの足し算をプログラムで実現するというものである．このような題材を授業で扱うことはプログラミング言語の基礎的な文法を指導することにおいて大切であるが，その題材で授業が終始すると子どもであっても大人であっても学習者の知的好奇心を刺激することは少ないことが実際の授業経験から判明している．プログラミングの専門家を志望する学習者であれば，題材が身近でなくてもプログラムそのものの仕組みや動作原理に注目することがあり，比較的興味を示すことが多いが，教養としてプログラミングを学ぶ学習者は，そのような反応を示すことは少ない．

学習者は身の回りの生活で利用される製品や技術に知的好奇心や興味を示すことが多い．たとえば音声情報処理技術 [24] は，スマートスピーカやスマートフォンなどの情報機器で利用できるようになっており，それらの機器を用いることで機器のON・OFF や情報検索を音声認識で実行したり，メッセージを音声合成で聞くことができるようになっており，このような技術を授業で扱うことは有効と考えられる．また，中学校技術科の学習指導要領には「開発者としての視点や考え方」を生徒へ

育むという趣旨の内容があり，そのような教育目標を実現するにあたっても学習題材として身近なものを扱うことは意義がある．

2.3　指導者の問題点

指導者の抱える問題について，本研究では次に述べるような問題点に着目している．昨今，短期間でプログラミング教育の対象が広がっており，プログラミングを学ぶ学習者は様々な人々へと拡張したため，学校現場の教員がプログラミングを指導する必要が生じているが，プログラミングは元来，一部の技術者が備えていたスキルであり，プログラミングの指導に不安を覚える教員も少なくない．このようなプログラミング教育をはじめとして，英語教育や道徳教育など大幅な教育革新により，学校現場の教員は授業実施の準備に対応を迫られている．初等中等教育に関して述べると，それらに加えてさらに部活動や進路指導など授業以外の教育活動も山積するため，教員の負担は増加している一方である．

教員はその職業上，新たなものを学ぶことに抵抗がない場合が多く，この意味では新たな教育内容が新設されても比較的順応しやすいと考えられる．しかし，多忙であるがゆえに研鑽時間が確保されていないため，能力や意欲の問題以前に時間的な制約が大きな課題となっている．時間的制約がある中でも限られた時間で，校内の代表者が現場の教員を対象としたプログラミング教育の研修へ赴き，研修が終了後，その代表者が校内の他の教員へ研修内容を伝導するということがあると聞くが，これも学校現場の業務が多忙で十分な研鑽時間を確保できていないことの現れである．

校内の複数の教員が業務の都合上，研修へ参加できない状況の場合，プログラミングに堪能な情報科の専任教員を必要人数校内に雇用して，彼らを中心に校内のプログラミング教育を推進することも考えられるが，これは現実的に難しい状況であることがわかっている．この理由としては様々なものが指摘されているが，理由の一つに，情報科の総授業数が専任教員を配置できる数を満たしていないとのことがあげられている．それゆえに，数学や理科を主に担当する専任教員が副教科として情報科の授業を兼任する形態が多く，これもまた兼任する教員の負担を増加しかねない課題といえる．

高等教育や社会人に対して行われるプログラミング教育であればプログラミングのうち，コーディングそのものの能力を高めることを目的として実施される授業もありえる一方で，初等中等教育で実施されるプログラミング教育の目的は，コーディングを覚えさせることではないと学習指導要領に明記されている．それよりも，プ

ログラムにおける複数の命令を組み合わせることで一つの大きな目的を達成したり，アルゴリズムを工夫する学習を通して，論理的思考力や緻密性などを育むという教育目標が掲げられている．

しかし，プログラミングを指導する教員がコーディングを覚えることに精一杯な状況であるとすると，授業の内容はコーディングで終始する恐れがある．従来の情報科の教育内容は，文書作成や表計算ソフトの使用方法を指導するだけにとどまらず，そのようなソフトの使用を通じて情報デザインやデータ分析力の素養を育むべきところを単にソフトの使い方に終始した事例からアプリケーション教育と批判されることもあった．このように，プログラミング教育もコーディング教育とみなされないよう意識する必要がある．

プログラミング教育は開始したばかりであり，現場の教員が授業教材を作成する余裕がないのが現状である．たとえば，指導者にとって身近な製品の機能や技術を取り入れた教材の作成は必ずしも容易ではなく，これは文字入力の方式によるプログラミング言語にも該当するが，ビジュアルプログラミング言語でもプログラミング教材の開発そのものが簡単になることはない．ビジュアルプログラミング言語は文字ベースの言語をもとに作成されており，ビジュアルプログラミング言語の理解に加えて，その言語を実装する文字ベースの言語の理解がなければ，新たに教材を作成することは困難である．

教材というと学習者の学びを支援する意味合いが主であるが，プログラミング教育などのまだ未成熟な教育分野に関して述べれば，とくに，教材が指導者に対しても内容理解や授業運用を支援する役割を有することも重要と考えられる．

2.4 プログラミング教材の先行研究

本研究では，プログラミング教育における問題を整理し，つぎの点に注目する．学習者にとっては，

1. 学習時間が本来学ぶべき学習内容以外のものによって損なわれること
2. 典型的なプログラミングの学習題材では十分な興味を持てないこと

指導者にとっては，

3. 指導時間が優先的に教授すべき内容以外のものによって損なわれること

4. 身近な製品の技術を取り入れたプログラミング教材の作成が容易でないこと

上記の問題 1 と 3 へは，文字ベースのプログラミング言語で記述されたプログラムに対して，システムが自動的にデバッグ支援を行う教材（以後，デバッグ支援システムと記す）を開発して対応する．上記の問題 2 と 4 へは，身の回りの製品にある身近な技術として音声情報処理技術を活用したプログラミング教材を開発して対応する．

まず，デバッグ支援システムに関する先行研究について，文字入力に基づくプログラミング言語を用いたプログラミングの初学者は，作成したプログラムをコンパイルする際にさまざまなエラーメッセージに直面してプログラムが実行できず，デバッグのために苦労する場合が多い．この場合，指導者がプログラムの誤りの修正方法を指導することとなるが，その誤りが単純な文法誤りの場合でもエラー原因の特定，修正方法の説明などに一定の時間を要するため，個別の指導を行なっている間は授業が停止するという問題を抱えることとなる．このように，プログラムのデバッグは学習者だけでなく指導者にとっても負担が大きいため，デバッグ支援を充実させるとともに，その支援によって初学者のプログラム修正能力を高めていく必要がある．

デバッグ支援に関連する研究として，プログラムにおけるバグを自動的に修正しようとする研究 [25–31] やプログラムの依存関係を解析し，プログラムからエラーを引き起こす部分のみを抽出する支援ツール [4] の報告があるが，これらの研究はプログラミング効率を向上させたり，プログラマの負担を軽減するという観点で有用である．また，プログラムのトレース能力を育成しようとする研究 [5] がある．この研究は学習者のプログラムトレース能力を向上させ，デバッグ能力も向上させるという観点で有用である．しかしこれらの研究結果は，コンパイルエラーに悩むようなレベルの初学者や，その指導者に対して必ずしも効果的な支援を行うものとはなっていない．

そこで本研究では，コンパイルエラー [32] が生じる段階でエラー修正に悩む初学者とその指導に悩む指導者を支援することを目的として，プログラムの誤りを修正する際に有益な補助メッセージを提示する手法とそのプログラムを新たに提案する．なお，本研究では文法誤りの修正を手助けすることを狭義の意味でデバッグ支援と記している．

つぎに，音声情報処理技術を活用したプログラミング教材に関する先行研究について，本研究では，初等・中等教育を通じて重要性が高まっているプログラミング

教育 [33–37] に対応する，音声情報処理技術を活かした教材の作成を検討する．音声情報処理技術を用いる理由は，技術そのものが身近となったことに加え，昨今，本技術を活用するには Web API を利用すれば比較的簡便に行えることにある．作成したプログラムを動作させる際は，テストデータを入力してプログラムを実行させたり，データを出力してプログラムが意図通りに動作しているかを確かめさせることがある．そのような際に，入力方法としてデータをあらかじめプログラムに記述しておいたり，出力方法として文字を画面に表示させる方法などでは指導者が学習者の興味を引き出しにくい．これについては，音声認識技術を入力方法として，音声合成を出力方法として採用すれば，学習者が身近で便利な技術を使用しながらプログラミングの学習を行うことができ学習意欲を高めることが可能である．

プログラミング教材は様々なものが報告されており，オブジェクト指向のプログラミングを学習させる教材 [6]，情報システムの技術理解を目的として複数のプログラミング言語を用いて学習を行う教材 [7]，センサを用いて情報を取得するなどハードウェアも利用してプログラミングの学習を行う実践 [8,9]，パズルを用いて情報処理の手順を工夫させるプログラミング教材 [10]，Scratch 1.4 の Mesh 機能を使用してネットワーク間の通信を活用するもの [11]，ドリトルを用いてネットワークの仕組みやサーバを活用したプログラミングを指導するもの [12–14]，PHP を使用して情報通信ネットワークを実践的に活用するもの [15,16]，複数言語を連携させてプログラミング教育を行うもの [17]，Google Blockly を使用して自動返信プログラムを作成するもの [18] などが報告されている．

本研究では，たとえば音声認識をデータ入力，音声合成をデータ出力として活用することを想定しており，上記の教材にはそのようなものが見当たらないが，たとえば Leap Motion を用いてキーボードやマウスに代わる，新たな入力デバイスを活用するプログラミング教材 [19] やプログラムの実行結果をモータに出力する教材 [20] と関連している．これらはいずれも，コンピュータの入出力を工夫して生徒の興味を引き，それによって入出力に関する理解を深めようとするものである．音声情報処理技術の原理を理解してプログラムを作成することは必ずしも容易でないが，開発者向けに公開されている API (Application Programming Interface) の一種である Web API [38–40] を用いれば，アルゴリズムの詳細を知らなくても音声認識や音声合成を用いたプログラムを少ない負担で開発できる．Web API のほかに，音声認識技術をフリーソフトウェアで扱える「Julius」[41] がある．Julius はオフラインで動作するソフトであるため，授業で用いる際は PC へ事前のインストールが必須となる．学校現場においては，授業を行う教員であっても PC へ新規のソフトウェアをイン

ストールすることが許可されていない場合も多いため，Julius を教材に活用することを断念した．一方，Web API の一つである Web Speech API [42] を用いれば，インターネットアクセスが必須であるが，Web ブラウザさえあれば動作し，音声情報処理を行う関数を作成できずとも音声情報処理の利用は可能となる．たとえば，音声認識については関数を呼び出してマイクロフォンに向かって話しかければ結果が文字列として出力される．同様に，音声合成については，文字列を所定の形で設定して関数を呼び出せば音声として出力される．

しかし，Web Speech API は開発者向けに公開されているものであり，開発時間があまり確保されていないなどの問題から現場の教員がそれを用いた教材を作成することは容易でない．そこで，本研究では Web Speech API を用いることで音声情報処理技術を活用したプログラミングが容易に行えることを示し，さらに，マイクロコントローラも併用して音声指示により計測制御動作を行うプログラミング教材を紹介することでフィジカルプログラミング教育の需要も満たしたプログラミングの学習例も示す．

第3章　デバッグ支援システム

3.1　初学者に対するデバッグ支援

プログラミングの初学者にとって，コンパイラのエラーメッセージに基づいてプログラムの誤りを自力で修正することは容易でない．プログラミングの素養がある技術者にとっては，エラーメッセージが出力されない場合のプログラムのデバッグに苦労することがあるが，コンパイルエラーが出力される場合は機械的にデバッグ作業を行えることも多くデバッグ支援の必要性は小さい．しかし，初学者に対しては，それ以前の段階としてエラーメッセージが出力される場合のデバッグ支援が必要となることが多く，本研究ではその点に注目している．エラーメッセージが出力されない場合についてもデバッグ支援が望まれるが，初学者にとってはコンパイラのエラーメッセージのために学習意欲が損なわれることがないように配慮することが重要である．

デバッグ支援として，学習者にプログラムの修正方針を示すことは指導者の役割であるが，たとえば，大学のプログラミング授業で教員ひとりに対し多数の学生が受講している場合，学生一人ひとりに対して教員個別によるデバッグ支援を行うことが困難であるため，指導者にとっては本来扱うべき学習内容を教授するための時間が十分に確保できないこと，学習者にとっては，指導者の支援を待っている間，プログラミングの学習が停滞することが問題である．既存の統合開発環境をそのまま利用すれば，簡単な文法上・構造上の誤りを修正することが可能であるが，本研究で想定している初学者の支援としては不十分と考えられる．

そこで，本研究では学習者がプログラムをコンパイルしてエラーメッセージが出力された際，プログラムにおける文法誤りを指摘して修正に導くような補助メッセージを提示することで，デバッグ支援を行う手法を提案する．エラーメッセージそのものも，デバッグに向けた手がかりであるが，コンパイラが出力するエラーメッセージはプログラムの文法規則に基づく処理結果として検出された不都合を表現したものであり，プログラミング能力の向上を目指したものではない．先にも述べたように，初学者がコンパイルエラーメッセージのみからプログラムの修正案を見出すこ

とは容易でないため，エラー修正に向けた，より具体的な補助メッセージをコンパイルエラーメッセージと別個に提示することとする．

ただし，コンパイラが直接出力したエラーメッセージを隠蔽してデバッグ用のアドバイスである補助メッセージのみを提示するのではなく，コンパイラのメッセージもそのまま表示する方針とする．これにより，学習者が経験を積むとともにコンパイラのエラーメッセージを理解するようになることを妨げず，指導者もエラー発生の状況を適切に把握することが可能である．

3.2　補助メッセージの提示

本研究では，デバッグ支援システムとして補助メッセージを提示する機能をシェルスクリプトを用いて実装しており，このシェルスクリプト内でコンパイラの呼び出しや，コンパイラの出力メッセージの分析，補助メッセージの挿入などを行なうようにしている．

3.2.1　GCC を用いる場合

本研究の手法は特定のプログラミング言語に限定するものではないが，ここでは具体例として C 言語のプログラミングの学習を行い，コンパイラとして GCC 5.4 を用いる場合を述べる．学習者はターミナル (仮想端末) を用いてソースコードの入力や編集，コンパイルを行なうものとする．

プログラムにおいてエラーが発生するとき，コンパイルエラーが 1 つのみ生じる場合と複数生じる場合がある．本研究ではエラーの個数に関わらずエラーメッセージとして最初に表示されたものに対して補助メッセージを提示し，デバッグ支援を行う．これは，1 つの誤りが最初のエラーメッセージだけでなく，その後の複数のエラーメッセージに関係する場合が多いことを考慮したためで，実際のプログラミング教育においても，まずは最初のエラーメッセージに着目するように指導することが多いことと対応している．

エラーメッセージの種類は，コンパイラが指摘した文法誤りを原因とするエラーとするが，細かな区分として，単に文法誤りというだけでなく，文字の綴り誤りが起因してその結果文法誤りとなるエラーも含む．

```
1 #include <stdio.h>
2 int main(void){
3   printf("Hello World!\n");
4   return 0;
5 }
```

図 3.1: 典型的なサンプルプログラム

```
1  <source>: In function 'main':
2  <source>:4:1: error: stray '\343' in
3  program
4   return 0;
5   ^
6  <source>:4:2: error: stray '\200' in
7  program
8   return 0;
9    ^
10 <source>:4:3: error: stray '\200' in
11 program
12  return 0;
13    ^
14 Compiler returned: 1
```

図 3.2: 非アスキー文字が混入したときのエラーメッセージ

3.2.2 文法誤りに対する補助メッセージ

綴り誤りが起因する文法誤りに関しては後述することにして，ここでは単に文法誤りが原因のエラーメッセージに対する補助メッセージを検討する．エラー内容として初学者にありがちな状況を想定し，そのエラーに対する修正案を補助メッセージとして提示させる．

初学者がプログラミング入門の題材として作成する典型的なプログラムの例を図 3.1 に示す．ただし，各行の左端の番号はソースプログラムの行番号である．この例では，プログラム 4 行目の「return 0」と入力する前に非アスキーの空白文字が記入されるなどした場合，コンパイラは図 3.2 に示すようなエラーメッセージを出力する．図 3.2 のエラーメッセージを見ても，初学者は非アスキーの空白文字がプログラムの中の許可されていない部分に混入していると気づくことは少ない．そこでデバッグ支援プログラムでは，文字列「stray」のあとにコンパイラから出力された文字コードの並びを確認し，そのコードに対応する文字がプログラムに混入している可能性を指摘する．この例では，文字コード UTF8 の空白文字がプログラムに混入していることがエラー発生の原因であるため，つぎに示すような補助メッセージを支援プログラムによって提示する．

ひょっとして …「return」の前に全角の空白文字を入力していませんか？

```
1 #include<stdio.h>
2 int main(void){
3   int n = 2;
4   if(n%2=0)
5   printf("偶数です\n");
6   return 0;
7 }
```

図 3.3: 比較文を代入文に書き間違えた例

```
1 #include<stdio.h>
2 int main(void){
3   int i;
4   2=i;
5   printf("i=%d\n",i);
6   return 0;
7 }
```

図 3.4: 左辺と右辺を逆さに書いた例

図 3.2 のエラーに対しては，修正案が唯一であると考えられるため，補助メッセージをより明確な表現で提案している．この例以外のエラーに対しても，修正案が唯一と考えられるものについては同様に補助メッセージを提示する．

つぎに，エラーメッセージを確認するのみでは，プログラムの修正案が唯一に定まらない場合について述べる．文法誤りを含む例を図 3.3 と図 3.4 に示す．

図 3.3 の例は，変数 n に代入された整数が偶数であれば,「偶数です」というメッセージを標準出力に表示させることを意図したプログラムであると推測される．しかし，この例ではプログラム 4 行目において，判定に用いる条件式が「`if(n%2=0)`」と記載されており，コンパイラは代入不能なものへ数値を代入していることをエラーとして検出する．図 3.4 の例は，変数 i に代入された値を標準出力に表示させることを意図したプログラムであると推測される．しかし，この例ではプログラム 4 行目において，左辺値を右辺へ代入しようとするような記述があり，図 3.3 と同様コンパイラは代入不能なものへ数値を代入していることをエラーとして検出する．

これらの例は，プログラムの内容やプログラマの意図はそれぞれ異なるものであるが，コンパイラの文法規則においては，同様のエラーとして検出される．コンパイラが出力するエラーメッセージについて，それぞれの例に共通した部分から抽出したものを図 3.5 に示す．

図 3.3 に対しては，補助メッセージとして比較文の記述を促すものが有効である．たとえばつぎに示すような補助メッセージをデバッグ支援プログラムで提示する．

ひょっとして … if 文の条件式が比較 (==) でなく代入 (=) となっていま

```
1 <source>: In function 'main':
2 <source>:4:11: error: lvalue required
3 as left operand of assignment
```

図 3.5: 代入不能な定数に値を代入したときのエラーメッセージ

せんか？

一方，図 3.4 に対しては，補助メッセージとしては左辺値と右辺値の入れ替えを促すものが有効である．たとえばつぎに示すような補助メッセージをデバッグ支援プログラムで提示する．

ひょっとして … 左辺と右辺が逆に書かれていませんか？

図 3.3 や図 3.4 のプログラムの場合，まず，提案システムでは等号のある行の左辺値と右辺値を置換し，再度コンパイルを実行する．この段階で図 3.4 の例ではエラーが解消するため，左辺値と右辺値の置換を提案する補助メッセージを提示する．一方，図 3.3 の例では左辺値と右辺値を置換してもエラーが解消されないので，次の段階として代入文を比較文に置換し，再度コンパイルを実行する．この例ではエラーが解消するため，代入文を比較文に修正する提案を補助メッセージとして提示する．代入文を比較文に変更してもエラーが解消しない場合は補助メッセージの追加を行わない．

この提案手法ではコンパイルエラーが発生するプログラムをシェルスクリプトで書き換えて再度コンパイルを試みる方法をデバッグ支援の一つとして提案しているが，対象のプログラムが大規模な場合は適切な修正を行なうことが必ずしも容易でなく，コンパイル時間の点でも不都合が生じる可能性が考えられる．しかしながら，初学者が作成する規模のプログラムではこれらが問題になることはない．

以上に述べた以外のエラーメッセージとそれに対する補助メッセージの例は詳細を省略するが，対応させた補助メッセージは，以下に具体例を列挙する．

- `expected '=', ',', ';', 'asm' or '__attribute__' before '{' token`
  ひょっとして … 引数を入れる ( ) を関数名のあとに書き忘れていませんか？
- `expected declaration or statement at end of input`
  ひょっとして … '}' を入力し忘れていませんか？

- `stray '\357' in program`
  `stray '\274' in program`
  `stray '\233' in program`
  ひょっとして … 全角の' ; ' が入力されていませんか？

- `missing terminating " character`
  ひょっとして … 全角の' " ' が入力されていませんか？

- `'i' undeclared (first use in this function)`
  ひょっとして … 変数 i の宣言方法を間違えていませんか？

- `redefinition of 'i'`
  ひょっとして … 変数 i を複数宣言していませんか？

- `expected ';' before '}' token`
  ひょっとして … '}' の前に ';' を入力し忘れていませんか？

- `expected ';' before ':' token`
  ひょっとして … ';'(セミコロン) を ':'(コロン) と間違えて入力していませんか?

どの補助メッセージも，初学者が経験することの多いエラー内容とそのときの状況を想定して，プログラムの誤りを修正に導くようなものを提示することにしている．

3.2.3 綴り誤りに対する補助メッセージ

プログラムの文法誤りを文字列の綴り誤りに特化して指摘し，その修正案を補助メッセージとして提示する方法を検討する．初学者が作成しがちなプログラムとして，関数名の綴りの誤り，その結果文法誤りとなる例を図 3.6 に示す．

この例では，学習者が定義したユーザ関数「func1」に関して，main 関数内のプログラム 4 行目で「funcl」と誤入力されているため，コンパイラは図 3.7 に示すようなエラーメッセージを出力する．この例では，数字の「1（イチ）」と英小文字の「l（エル）」の字体が似ているため，エラーメッセージを見ても何がエラーの原因であるか初学者は気づかないことが多い．そこで，デバッグ支援プログラムでは，コンパイラから出力された未定義の関数名「funcl」と類似している文字列の関数名「func1」を検索し，つぎに示すような補助メッセージを提示する．

```
1  #include <stdio.h>
2  int func1();
3  int main(){
4    funcl();
5    return 0;
6  }
7  int func1(){
8    printf("Hello World!");
9    return 0;
10 }
```

図 3.6: 関数名の綴りを誤って書いた例

```
1  <source>: In function 'main':
2  <source>:4:3: warning: implicit
3  declaration of function 'funcl'
4  [-Wimplicit-function-declaration]
5     funcl();
6     ^
7  Compiler returned: 0
```

図 3.7: 綴りを誤って書いたときのエラーメッセージ

ひょっとして ···· 関数名 funcl の綴りを誤っていませんか？

さらに，具体的に綴りの訂正候補を初学者へ提示することが効果的なデバッグ支援であると考え，上記の補助メッセージに加えてつぎに示すようなメッセージもデバッグ支援プログラムで提示する．

ひょっとして ··· funcl ではなく func1 ですか？

修正案の文字列をデバッグ支援プログラムが提示する方法について述べる．まず，支援プログラムの内部において，文字列の比較に用いるファイルを作成する．以後，そのファイルを辞書ファイルと記す．ファイルには文字列のリストが並び，その項目は学習者が記述したプログラムにおける両端が空白で挟まれる文字列と C 言語であらかじめ定義される関数やデータ型の名称である．つぎに，デバッグ支援プログラムは辞書ファイルを参照して，綴り誤りの文字列に対して類似度が高い文字列を検索する．

2 つの文字列 w_1, w_2 の類似度を考える際によく用いられるレーベンシュタイン距離 $d_L(w_1, w_2)$ は，文字列の編集作業における 1 文字の削除，追加，置換を単位操作としたときに文字列 w_1 を文字列 w_2 に変更するために必要な最小の操作回数である．

本研究でも同様の距離を扱うが，レーベンシュタイン距離をそのまま用いると 図 3.6 に挙げたような字体の類似が考慮できないので，レーベンシュタイン距離におけ

る置換の対象が数字の 1 と英小文字の l，数字の 0 と英大文字の O の場合の単位操作に対する距離を修正する．1 文字の削除，追加，置換に対応する距離をそれぞれ d_d, d_i, d_s とすると，レーベンシュタイン距離では

$$d_d = d_i = d_s = 1 \tag{3.1}$$

であるが，本研究では

$$d_d = d_i = 1 \tag{3.2}$$

$$d_s = \begin{cases} 0.9 & \text{比較対象が 1 と l，0 と O などのとき} \\ 1 & \text{それ以外のとき} \end{cases} \tag{3.3}$$

としており，その場合の文字列 w_1 と文字列 w_2 の距離を $d(w_1, w_2)$ と定義して，以下の類似度の計算に用いることにする．

文字列 w_1, w_2 の長さをそれぞれ l_1, l_2 として，長いほうの長さを

$$l = \max(l_1, l_2) \tag{3.4}$$

とすると，w_1, w_2 の相対距離あるいは類似度 $s(w_1, w_2)$ は

$$s(w_1, w_2) = \frac{l - d(w_1, w_2)}{l} \tag{3.5}$$

で与えられ，通常のレーベンシュタイン距離を用いた場合と同様に，

$$0 \leq s \leq 1 \tag{3.6}$$

となる．

たとえば，図 3.6 の例が拡張されて,「func1」だけでなく「func2」が追加された場合を考えると,「funcl」に対して「func1」と「func2」のレーベンシュタイン距離はいずれも 1 で，文字列の類似度も等しいので，これを用いて綴りの修正案を提示すると,「func1」と「func2」を区別して取り扱うことができない．本研究では，(3.3) 式の修正を行うことにより，func1 への修正を優先的に提案することが可能である．

本機能においては，綴り修正案を提示するための類似度の限度を定めることも重要である．例えば，整数型を表す単語 int を ind と間違えた場合，文字列間の類似度は 2/3 であり，例えば if を it と間違えた場合の類似度は 1/2 である．本研究では，3 文字の単語で 1 文字を間違えた場合を限度と考え，類似度が $s(w_1, w_2) > 0.66$

```
1  #include <stdio.h>
2  int main(){
3    int i = 3;
4    if( 0<i<5 ){
5      i++;
6    }else{
7      i--;
8    }
9    return 0;
10 }
```

図 3.8: 条件式が常に真と評価される例

の場合に綴り修正案の候補を提示することとした．

ユーザが定義した関数名の文字列に限らず，ユーザ定義の変数名やプログラミング言語固有の予約語の綴りを学習者が間違えた場合も同様で，デバッグ支援を目的とした綴りの修正案を補助メッセージとして提示している．

3.2.4 警告メッセージの利用

初心者のプログラミングの学習を対象として，エラーメッセージが出力された場合のデバッグ支援を中心にこれまで述べてきたが，エラーメッセージが出力されない場合のデバッグ支援も重要であることに変わりなく，具体的な支援を行うことが望まれる．

たとえば，図 3.8 に示すような C 言語プログラムを考える．図 3.8 は，変数 i の値に応じて命令を分岐させ，0 <i< 5 の場合は i の値をインクリメント，それ以外の場合は i の値をデクリメントしようとしたプログラムであると推測される．しかしこの例では，プログラム 4 行目において変数 i の値がどの整数値であっても常に if 文の条件式は真と評価されるため，常に 5 行目の命令が実行されのみで 7 行目の命令が実行されることはない．プログラムには文法規則上誤りがないため，コンパイラがエラーメッセージを出力することもない．

ただし，図 3.8 に対して，GCC 5.4 をコンパイラとして利用するのであれば，警告メッセージを出力する「`-Wall`」オプションを付加してプログラムをコンパイルすると，図 3.9 に示すメッセージが出力される．このメッセージは，if 文の条件式が常に真と評価され分岐構造となっていないことや 0 <i< 5 という記述が不適切であることを警告しているが，やはりコンパイラが出力する警告メッセージもコンパイルエラーメッセージと同様，プログラムの修正方法を提示することが主な目的ではないため，初学者にとってはデバッグに必要な情報が十分に得られない．

```
1  <source>: In function 'main':
2  <source>:5:13: warning: comparison of
3  constant '5' with boolean expression
4  is always true [-Wbool-compare]
5       if( 0<i<5 ){
6               ^
7  <source>:5:11: warning: comparisons
8  like 'X<=Y<=Z' do not have their
9  mathematical meaning [-Wparentheses]
10      if( 0<i<5 ){
11            ^
12 Compiler returned: 0
```

図 3.9: コンパイルエラーメッセージが出力されない例

そこで，図 3.9 のような例の場合には，本研究が提案するデバッグ支援システムにおいて,「`-Wall`」オプションを付記したコンパイルを自動的に実行し，そのときコンパイラから得られた警告メッセージを利用して，そのメッセージの内容に応じた補助メッセージを初学者へ提示することがデバッグ支援として有効である．図 3.8 のようなプログラムがコンパイルされた場合は，たとえば，学習者へつぎに示すような補助メッセージを提示する．

`0<i<5` は `0<i && i<5` ということですか?

このように，警告メッセージを利用して補助メッセージを提示すれば，コンパイルエラーが出力されない場合にもデバッグ支援が可能となる．

3.3　統合開発環境への適用例

本研究で提案するデバッグ支援プログラムは，統合開発環境を用いたプログラミングの学習に対して適用することが可能である．本節では，Arduino [43] の統合開発環境 (Arduino IDE) へデバッグ支援システムを適用した例を述べる．

Arduino は Atmel 社が開発する AVR マイコンが搭載されたワンボードマイコンの一つで，プログラミングを行う際に利用する統合開発環境 (Arduino IDE) や C++ 風の Arduino 言語などを含む総称である．ハードウェアである基板とソフトウェアである Arduino IDE はともにオープンソースであり，自由に改変が許される．なお，Arduino を用いた電子工作例やプログラムおよびライブラリは豊富に開発されており，教材開発においても有用である．Arduino IDE で作成したプログラムの中にエラーが存在し，そのままコンパイラを実行すると，Arduino IDE のコンソール画面

図 3.10: Arduino IDE

へエラーメッセージが出力される．プログラミングの際は Arduino IDE（Arduino 統合開発環境）を用いることが一般的である．

図 3.10 にコンパイルエラーメッセージが表示された Arduino IDE の画面例を示す．

図 3.10 の画面下部が Arduino IDE のコンソール領域であり，この領域にコンパイラの出力するエラーメッセージが表示される．Arduino IDE に含まれるコンパイラは C++言語用のクロスコンパイラ avr-g++である．

Arduino IDE では，プログラマからコンパイラの実行命令を受けると，Arduino 言語で書かれたプログラムを IDE 内部で C++言語のプログラムに変換後，プログラムのコンパイルが開始される．IDE では，コンパイラの出力したエラーメッセージがあれば，それがコンソール画面に表示されるようになっている．この仕組みの概念を図 3.11 に示す．

本研究では，図 3.11 で示したような，Arduino IDE がコンパイラを実行しエラーメッセージをコンソール画面へ表示する一連の処理に介入し，デバッグ支援システ

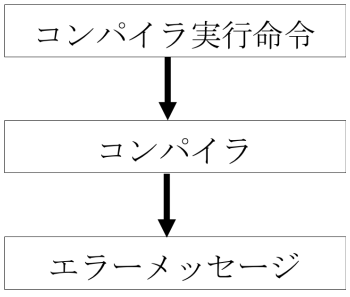

図 3.11: Arduino IDE がコンパイルエラーメッセージを表示する仕組み

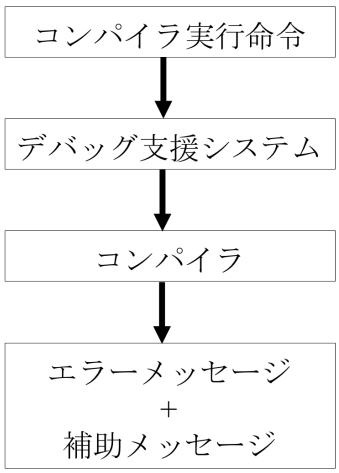

図 3.12: Arduino IDE に対するデバッグ支援システムの適用

ムがエラーメッセージの分析と補助メッセージの提示を行えるよう適用作業を行った．本適用作業の概念を図 3.12 に示す．

具体的に，まず，Arduino IDE のソースプログラムを入手する．つぎに，ソースプログラムにおける avr-g++をリネームし，デバッグ支援システムを avr-g++の名前で登録する．この段階で，Arduino IDE ではコンパイラを動作させる命令を受けると，元の avr-g++ではなく，avr-g++とファイル名を変更したデバッグ支援システムを実行する．なお，デバッグ支援システムにはエラーメッセージを分析し，提示する補助メッセージの作成後，そのメッセージをリネームされた avr-g++に入力する機能を設けることで IDE 一連のエラーメッセージを表示する処理に介入が完了となる．

デバッグ支援システムの介入作業を終えたあとは，Arduino IDE のソースプログラムをビルドして，実行ファイルとしての Arduino IDE を生成する．

Arduino IDE のソースプログラムには開発者のビルド作業を支援する build.xml ファイルが存在する．このファイルには，ビルドに必要な処理が順を追って記述されており，開発者が手動でビルドに必要な処理を一つひとつ実行せずとも，一度の命令でビルド作業を自動的に行う機能がある．本研究においても build.xml を使用してデバッグ支援システムを組み込んだ独自版の Arduino IDE をビルドしている．

ここまでに述べた適用処理を完成させると Arduino IDE にデバッグ支援システムなどの独自のプログラムを適用することができる．デバッグ支援プログラムを組み込んだ Arduino IDE の画面例を図 3.13 に示す．

図 3.13 下部のコンソール画面では，エラーメッセージに加えて，本研究のデバッ

図 3.13: 補助メッセージを提示する Arduino IDE

グ支援プログラムが提示する補助メッセージが出力されている．

上記の通り，本研究で開発したデバッグ支援システムが Arduino の開発環境においても適用できることが確認された．

第4章　音声情報処理技術を活用したプログラミング教材

4.1　プログラミング教材の構成要素

本研究において開発した音声情報処理技術を活用したプログラミング教材の構成要素を図 4.1 に示す.

図 4.1 で示した教材では，音声認識を行う場合，まず，Web Speech API を Web ブラウザの操作によって呼び出す．Web Speech API とは，Web ブラウザで動作する Web ページの機能として音声認識や音声合成を実行することができる API である．Web Speech API の他にも Web ブラウザから音声情報処理を実行することのできる Web API は存在するが，利用にあたって登録作業が必要であったり，有償の提供であることが多い．その点，Web Speech API は登録作業や使用料が必要なく，教材開発において導入しやすい．この API を利用するには JavaScript のコードとして Web ページに記述する．このとき，Web Speech API は Web ブラウザの標準システムで動作する API のため，プログラマが Web ページにおいて JavaScript ライブラリをインポートする必要もない．ただし，普及している Web ブラウザの全てが Web Speech API を実行できるわけでなく，現状では対応しているものとそうでないものが存在する．対応している Web ブラウザの例は，Google Chrome や Chromium，Edge，Mozilla Firefox などである．なお，Mozilla Firefox に関しては標準では API を実行することはできず，Web ブラウザのオプションを手動で変更す

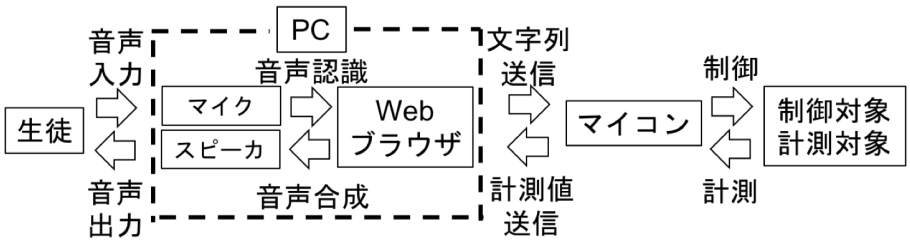

図 4.1: 教材の構成

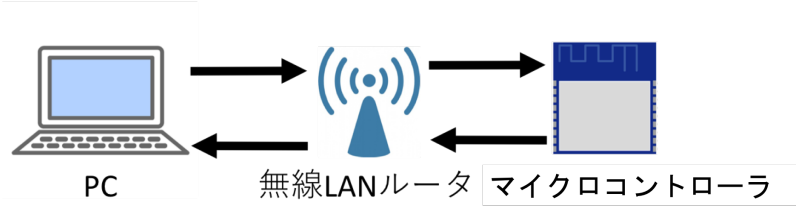

図 4.2: PC とマイクロコントローラ間の通信形態

ることで実行可能となる．Web ブラウザの種類によっては未対応なものがあるものの，Web Speech API は W3C(World Wide Web Consortium) が標準化を進める技術の一つであるため，未対応の Web ブラウザも今後で対応する可能性は考えられる．

さて，Web ブラウザが Web Speech API を正常に呼び出した後，音声の入力を求めるダイアログ画面が Web ブラウザから表示されるため，マイクロフォン（図 4.1 ではマイクと表記）を介して音声を入力する．その後，Web Speech API を機能により音声認識結果の文字列が生成されるため，この文字列はプログラム内で自由に利用することができる．プログラムの実行結果を PC 内で出力するだけにとどまらず，ハードウェアを用いたプログラミングの学習とする場合，本教材では音声認識により得られた文字列をマイクロコントローラ（図 4.1 ではマイコンと表記）へ送信する．マイクロコントローラとは，CPU やメモリなどのコンピュータシステムが 1 つの集積回路に組み込まれた小型の PC であり，用途としては家電を制御するなどの組み込み用が多く，汎用 PC とは性質が異なる．

音声認識で生成した文字列をマイクロコントローラへ送信する際，その通信手段としては様々なものが考えられるが，本教材では通信手段として，IEEE802.11 規格の無線 LAN を選択して教材開発を行った．具体的に PC とマイクロコントローラ間の通信形態を図 4.2 に示す．

図 4.2 の通信形態において，PC から文字列を受信したマイクロコントローラは，文字列の値に応じて計測処理あるいは制御処理を実行することができ，計測処理が実行された場合は，マイクロコントローラが計測値を送信して，そのデータを Web ブラウザで受信することもできる．また，音声合成を行う場合，Web Speech API に引数として音声合成させたい文字列を入力すれば，スピーカを介して音声が出力される．なお，本研究では計測対象を照度，制御対象を LED とし，プログラミングの学習もそれらの計測制御を前提として論じるが，この部分は授業目的に応じて自由に教員が変更可能である．

ここまでは主に，図 4.1 の約左半分の部分であるソフトウェア要素について述べた

図 4.3: ESP-WROOM-02

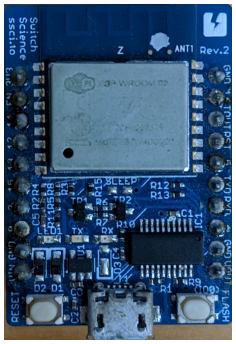

図 4.4: ESPr Developer

が，ここからは約右半分のハードウェア要素について述べる．まず，マイクロコントローラとして使用した ESP-WROOM-02 は Espressif Systems の Wi-Fi モジュールを備えたマイクロコントローラの一つであり，プログラム次第でサーバ機能を持たせることもできる．図 4.3 に ESP-WROOM-02 の外観を示す．

図 4.3 で示した ESP-WROOM-02 は，ピンのピッチが短いため専門技能や設備を持たない者が配線することは困難であり開発の敷居は高い．この問題は，ESP-WROOM-02 のピッチが調節されたスイッチサイエンス社の ESPr Developer (ESP-WROOM-02 開発ボード) を使用すれば解決する．図 4.4 に ESPr Developer の外観を示す．

図 4.4 で示した ESPr Developer は，図 4.3 で示した ESP-WROOM-02 と比べ，USB-シリアル変換 IC や電圧降下に用いるレギュレータなどが標準装備されている分，使い易さは向上するが高価である．そこで，本研究では ESP-WROOM-02 のピンピッチのみが調整された秋月電子通商の AE-ESP-WROOM-02 とその他の必要部品を準備し，それらを組み立てることで教材コストを下げる検討を行った．AE-ESP-WROOM-02 の外観を図 4.5 に示す．

図 4.5 の AE-ESP-WROOM-02 に対して，USB-シリアル変換 IC やレギュレータなどの部品を別途用意し，ESP-WROOM-02 のインターフェース定義に注意して適

図 4.5: AE-ESP-WROOM-02

切に電気回路を組み立てることで ESPr Developer を購入して使用する場合よりも教材コストを削減できる．

ESP-WROOM-02 のインターフェース定義 [44] は表 4.1 のようになっている．

また，ESP-WROOM-02 は「UART ダウンロードモード」と「Flash Boot モード」の 2 つの動作モードがあり，プログラムの書き込み時は前者，プログラムの実行時は後者のモードを利用する．表 4.2 と表 4.3 にインターフェースと入力電圧の値を示す．

表 4.2 と表 4.3 で示した 2 のモードを切り替える方法は，所定のピンの電圧を変化させることで達成される．本研究ではこれらの点を考慮して回路設計を行い，ESPr Developer と同等の機能を有する AE-ESP-WROOM-02 の回路を開発した．その回路を図 4.6 に示す．なお，電気回路はブレッドボードを用いて構成している．

図 4.6 の回路においては，モードを切り替えるスイッチとしてタクトスイッチを利用した．このとき，プルアップ抵抗として 10kΩ の抵抗を回路において用いた．また，USB-シリアル変換 IC を別途設置した．ESPr Developer では USB 電源の 5V を 3.3V に降圧するレギュレータが内蔵されているが，AE-ESP-WROOM-02 を用いる場合はそれがないため，別途三端子レギュレータを設置し，3.3V の回路を作成している．

4.2　計測・制御プログラミング教材への適用例

4.1 節で述べた ESPr Developer や AE-ESP-WROOM-02 を使用したプログラミング題材は様々なものが考えられるが，本節では計測・制御技術の学習を志向した適用例を述べる．具体的には，生徒が CdS を用いて室内の明るさを計測し，その結果

表 4.1: ESP-WROOM-02 のインターフェース定義

番号	Pin 名称	機能説明
1	3V3	3.3V 電源
2	EN	CHIP ENABLE アクティブ・ハイ
3	IO14	GPIO14; HSPI_CLK
4	IO12	GPIO12; HSPI_MISO
5	IO13	GPIO13; HSPI_MOSI; UART0_CTS
6	IO15	GPIO15; MTDO; HSPICS; UART0_RTS
7	IO2	GPIO2; UART1_TXD
8	IO0	GPIO0
9	GND	GND
10	IO4	GPIO4
11	RXD	UART0_RXD; GPIO3
12	TXD	UART0_TXD; GPIO1
13	GND	GND
14	IO5	GPIO5
15	RST	モジュールリセット
16	TOUT	ADC_IN
17	IO16	GPIO16; 低電力モードウェイクアップ
18	GND	GND

表 4.2: UART ダウンロードモード

GPIO15	GPIO0	GPIO2
Low	Low	High

表 4.3: Flash Boot モード

GPIO15	GPIO0	GPIO2
Low	High	High

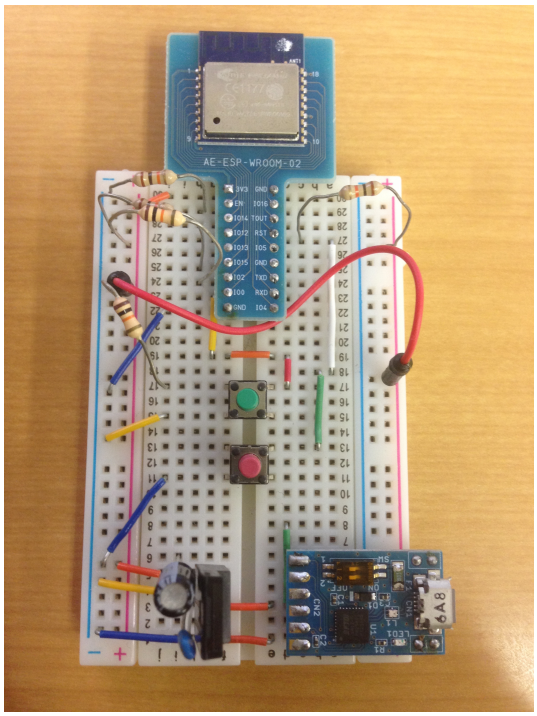

図 4.6: AE-ESP-WROOM-02 で構成した ESPr Developer と同等の回路

に基づいて LED の点灯・消灯を制御するものである．本教材で用いた部品を表 4.4 に示す．

これらの部品を用いて，CdS による照度計測と LED の点灯・消灯を行う電気回路をブレッドボード上に構成する．回路図を図 4.7 に示す．

AD コンバータが内蔵される ESP-WROOM-02 の TOUT ピンは，1.1V 以下の電圧を入力する必要がある．そこで，電源電圧の 3.3V が R1，R2，CdS の抵抗で 3 分の 1 未満となるような分圧回路を構成した．なお，抵抗 R2 に関して 10kΩ の抵抗 3

表 4.4: 回路構成に用いたハードウェア部品

名称	個数	用途
ESP-WROOM-02	1 個	マイクロコンピュータ
LED	1 個	制御対象
CdS	1 個	照度センサ
抵抗 (100Ω)	1 個	電流調整
抵抗 (10kΩ)	3 個	入力電圧調整
モバイルバッテリ (5V 出力)	1 個	電源
ジャンパ線	必要に応じた数	配線

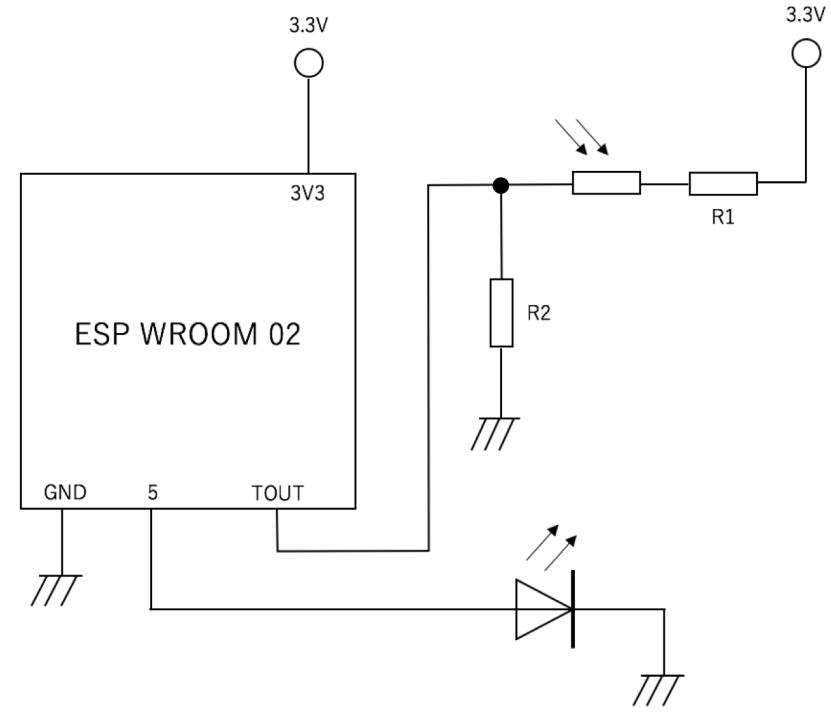

図 4.7: ESP-WROOM-02 を用いて LED の点灯と CdS による照度計測を行う電気回路

つを用いて 3.3kΩ 相当の抵抗を作成した．以上の設計をもとに電気回路を構成した．それを次の図 4.8 に示す．

ESPr Developer は USB 端子を有するため，電源は USB ケーブルで供給可能なモバイルバッテリを利用したが，AC アダプタや乾電池などから電源を供給することも考えられる．

本教材では，ESPr Developer で動作させるソフトウェアとしてサーバ機能を実装した．このサーバの役割は，主に外部のプログラムからの指示に応じて計測と制御動作を実行することである．本サーバプログラムを授業で扱うことも考えられるが，中等教育のプログラミングの学習ではむしろ，既存のプログラムと連携して，その恩恵を享受しながら本来の目的であるアルゴリズムの工夫やプログラミング的志向を指導する方が授業の趣旨に合致する．そこで，本研究ではサーバプログラムを Arduino 言語を用いて開発し，既存のプログラムとして利用させることとした．Arduino 言語を用いる理由は，開発環境の準備が容易であることや開発に必要なライブラリが豊富に用意されているからである．ただし，その場合は ESP-WROOM-02 を Arduino 互換機として動作させる必要が生じる．ESP-WROOM-02 はファームウェアを書き換えることにより，Arduino 互換機として動作させることが認められている．なお，一

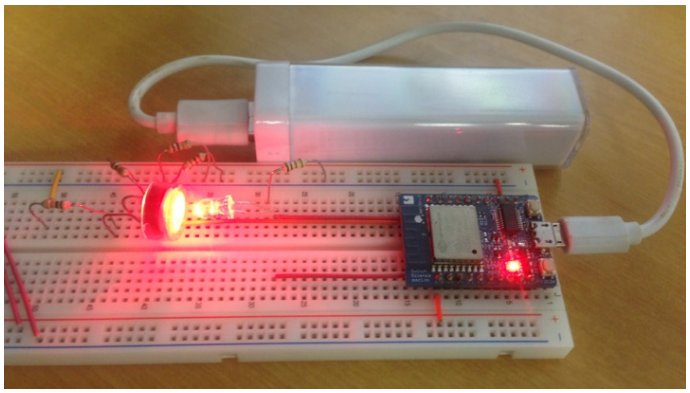

図 4.8: 教材を構成する電気回路

度ファームウェアを書き換えると ESP-WROOM-02 に元々書き込まれていたファームウェアは削除されるが，書き換えたファームウェアを元に戻すこともできる．

ファームウェアを ESP-WROOM-02 元来のものから Arduino のものへ書き換えたあと，ESP-WROOM-02 のユーザプログラム領域へ本研究で開発したサーバプログラムを書き込む．サーバプログラムにおける外部プログラムとの通信方法は WebSocket プロトコルを用いることとした．外部プログラムとして Web Speech API を用いることを前提とした場合，Web Speech API が JavaScript プログラムで記述されるため，生徒がその部分のプログラミングに関与する可能性も踏まえると，サーバプログラムとの通信も JavaScript で記述できるようにしておくことが望ましい．JavaScript では WebSocket プロトコルを記述して比較的容易に IP 通信を実現することができ，授業内で完結が望まれるプログラミングの学習においては適切と考えられる．

また，サーバプログラムと外部プログラムが通信する際の文字列の送受信形式については JSON(JavaScript Object Notation) を用いることとした．JSON は人間にとって読み書きが容易とされ，WebSocket プロトコルを採用した理由と同様に，プログラミングを入門的に始める生徒が文字列の送受信を行うコードをプログラミングする可能性を踏まえると，読み書きが比較的容易という点はプログラミングの要素として好都合である．

```
<!DOCTYPE html>
<head>
  <meta charset="utf-8" />
  <title>計測･制御教材</title>
 </head>
 <body>
 </body>
</html>
```

図 4.9: 基本の Web ページを成す HTML コード

```
<!DOCTYPE html>
<head>
  <meta charset="utf-8" />
  <title>計測･制御教材</title>
</head>
<body>
  <div align="center">
    <button id="rcgbtn">音声認識</button>
  </div>
  <script src="app.js"></script>
</body>
</html>
```

図 4.10: JavaScript やボタンタグを含む HTML コード

4.3 学習者のプログラミング

4.3.1 JavaScript を用いたプログラミングの学習

JavaScript を用いてプログラミングの学習を行う場合，HTML を用いた Web ページの作成を合わせて指導する．たとえば次のような HTML コードを記述するよう教員が生徒へ指導する．図 4.9 にコードの例を示す．

図 4.9 のコードは，Web ページの構造のベースを成すものであり，このコードに JavaScript プログラムやそれを呼び出すボタンの HTML タグなどが追記される．図 4.10 は，追記した HTML コードの例である．

別ファイルとして参照している JavaScript プログラムの app.js に中に音声認識を実行する命令が定義されており，それらを利用して記述するだけでもプログラミングを体験することが可能であるが，app.js の内容を生徒が書き換えることでより発展的な授業を実践することもできる．図 4.11 は，app.js において，音声認識結果に応じて所定の処理を実行するコードの抜粋である．app.js の全コードを付録 A.1 に掲載する．

図 4.11 のコードでは，1 行目で音声認識を行う Web Speech API を呼び出しており，音声認識処理のインスタンスを変数 recog へ代入している．2 行目で認識させる

```
1  recog = new webkitSpeechRecognition();
2  recog.lang = "ja";
3  rcgbtn = document.getElementById( 'rcgbtn');
4  rcgbtn.addEventListner( 'click' ,function(){
5    recog.start();
6  });
7  rcgbtn.addEventListener( 'result' ,function(event){
8  var text = event.results[0][0].transcript;
9    console.log(text);
10 }, false);
```

図 4.11: 音声認識を行うコード

```
1  var text = event.results[0][0].transcript;
2  msg.textContent = "認識結果: " + text;
3  if (text == "点灯"){
4    socket.send("\{\"command\":\"HIGH\"\}");
5  } else if (text == "消灯"){
6    socket.send("\{\"command\":\"LOW\"\}");
7  } else if (text == "計測"){
8    socket.send("\{\"command\":\"MEASU\"\}");
9  }
```

図 4.12: 音声認識により計測制御動作を実行するコード

言語を設定することができ，実際には日本語を指定している．3 行目で HTML コードのボタンタグと結びつけを行い，結びつけたボタンを押された場合の処理を 4 から 6 行目で記述している．その処理を 5 行目の recog.start() のように記述すれば，所定のボタンを押下した際に音声認識を開始することができる．7 から 10 行目では，音声認識が完了した場合の処理が記述されており，8 行目のように変数 text へ認識結果の文字列を代入したり，9 行目のように認識結果を出力して確認することができる．

本教材を併用し，図 4.11 のプログラムを応用すると音声で指示をしたのちは，自動的に計測および制御動作が実行される．図 4.12 は，図 4.11 のプログラムの 9 行目を変更し，認識結果を活用してマイクロコントローラへ計測制御命令を出すようにしたものである．

図 4.12 のコードでは，音声認識結果の文字列が代入される変数 text を文字列比較して，所定の文字列と一致した場合は,「点灯」の場合 LED を点灯させる命令，以下同様に「消灯」の場合 LED を消灯する命令,「計測」の場合は照度を測る命令をマイクロコントローラへ出力する．具体的には処理内容と紐付けた JSON 文字列を送信し，通信相手であるマイクロコントローラは，その文字列を受信した際，その文字列を比較して値に応じた処理を実行する．ところで，音声合成を実行するコー

```
1 var obj = JSON.parse(e.data);
2 if (obj.CdS >= 700) {
3   synth.text = "明るいです";
4   speechSynthesis.speak(synth);
5 } else {
6   synth.text = "暗いです";
7   speechSynthesis.speak(synth);
8 }
```

図 4.13: 音声認識を行う部分

ドは以下のように記述することができる．

図 4.13 のコードでは，1 行目でマイクロコントローラから送信される JSON データを照度として取得し，そのデータを変数 obj に代入している．2 行目以降で照度の計測値を比較するため，obj の CdS プロパティを参照して処理を分岐し，計測値が 700 以上であれば「明るいです」，699 以下であれば「暗いです」と音声合成する処理が記述されている．

4.3.2 Scratch を用いたプログラミングの学習

パズルのような視覚的オブジェクトを組み合わせてプログラムを作成するビジュアルプログラミング言語の一つの Scratch [45] は，プログラミング初学者であっても学習が行いやすい設計となっている．ビジュアルプログラミング言語を用いるプログラマは，内部の処理を意識する必要がなくオブジェクトの組み合わせに集中すればプログラミングが行えるため，ビジュアルプログラミング言語が初学者にも実践しやすいプログラミング言語といわれる．

ビジュアルプログラミング言語の実体は，テキストベースのプログラミング言語で書かれたプログラムであり，ビジュアルプログラミング言語で書かれたプログラムを実行するとアプリケーション内部ではテキストベースのプログラミング言語として処理される．Scratch そのものを構成するプログラムはオープンソースとして公開されており，仕様を変更して自由に利用することが可能である．

Scratch を用いてプログラミングの学習を行う場合，Scratch のコマンドとして音声情報処理を実行するブロックやマイクロコントローラと通信するブロックを作成することで，音声指示により計測制御動作を実行するプログラミングが行えるようになる．

そこで，本研究では Scratch ブロックを新たに作成して，既存の Scratch ブロック群にそれらを追加した．独自に追加した Scratch ブロックの一覧を図 4.14 に示す．

図 4.14: 追加した Scratch ブロック

図 4.15: マイクロコントローラへの接続と接続を切断する Scratch スクリプト

図 4.14 で示した Scratch の拡張ブロックは Web サーバに設置して，プログラミングの際に既存の Scratch システムへ融合させる必要がある．そこで，これに必要な Web サーバを開発した．本 Web サーバは授業での利便性を考慮して，ローカル PC でも動作させることができる．具体的なプログラムは Node.js [46] を用いて作成した．そのソースプログラムを付録 A.2 に記載する．

さて，以降では Scratch スクリプトによりマイクロコントローラを介して LED を制御したり，照度を計測するプログラムを例にして，図 4.14 で示した Scratch ブロックの使い方を述べる．たとえば，図 4.14 のブロック「〜に接続」や「〜から切断」は，Wi-Fi 通信網と WebSocket プロトコルを用いてマイクロコントローラと接続したり，接続を切断する命令である．

図 4.15 にこれらのブロックの使い方を示す．

図 4.15 のスクリプトでは,「マイコン 1」と名付けた変数にマイクロコントローラに割り当てられた IP アドレスなどを文字列で代入し，その変数をブロックの引数として用いることで特定のマイクロコントローラに対して接続したり，その接続を切断することができる．IP アドレスの文字列部分を変更し，同様のプログラムを作成すれば，一つの Scratch プロジェクトから複数のマイクロコントローラを用いて計測制御動作を実行可能である. また， 1 つのマイクロコントローラに複数の Scratch を接続し，ハードウェアを共有して使うこともできる.

つぎに，音声認識によってマイクロコントローラを介し制御動作を実行する Scratch スクリプトを図 4.16 に示す.

図 4.16 のスクリプトでは,「音声認識」ブロックが実行され音声認識が完了すると「音声認識結果」ブロックへ認識結果である文字列が格納される．このブロックの値を文字列比較することで処理を分岐させ，値が「こんにちは」と一致すれば LED を

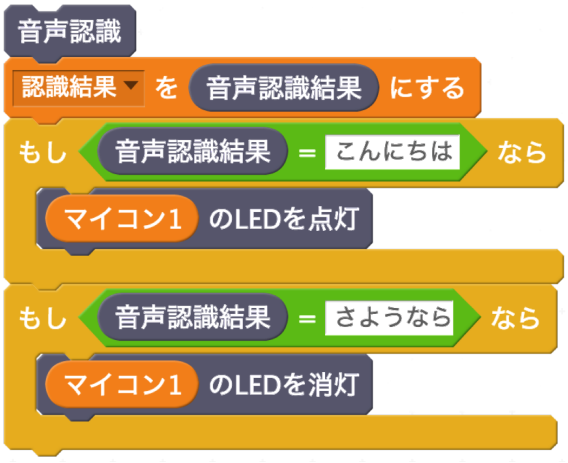

図 4.16: 音声認識の結果を利用して LED を制御する Scratch スクリプト

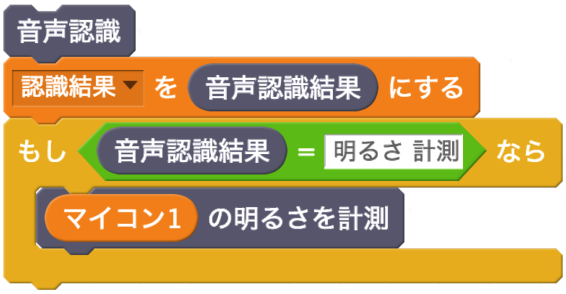

図 4.17: 音声認識の結果を利用して計測動作を実行する Scratch スクリプト

点灯,「さようなら」と一致すれば LED を消灯する命令を「マイコン 1」と名付けたマイクロコントローラへ出力する．なお，変数「認識結果」を参照すれば意図通りに音声認識がなされているか確認可能である．さらに，音声認識によってマイクロコントローラを介し計測動作を実行して計測値を取得する Scratch スクリプトを図 4.17 と図 4.18 に示す.

図 4.17 のスクリプトでは，図 4.16 と同様の処理を行い,「音声認識結果」ブロックの値を文字列比較して,「明るさ 計測」と一致すれば照度を測る命令を「マイコン 1」と名付けたマイクロコントローラへ出力する．この命令を受けたマイクロコントローラは，計測した照度を Scratch に送信する．

一方，Scratch では図 4.18 のスクリプトにおいて，マイクロコントローラから送信される計測値を受信し，受信した計測値に基づいて照度を計測した環境の明暗を判定する．なお，変数値は Scratch 既存のモニターを利用して閲覧可能である．モニターの様子を図 4.19 に示す.

図 4.18: 計測値を取得して明暗を判断する Scratch スクリプト

図 4.19: Scratch における変数閲覧

表 4.5: アンケートの内容

質問番号	質問文
I	プログラミングに対する興味や関心がもてましたか？
II	プログラミングに対する興味や関心が持続しましたか？
III	アルゴリズムを工夫しようとしましたか？
IV	この授業の後も，プログラミングを続けてみたいと思いましたか？
V	音声情報処理技術に対する理解が深まりましたか？
VI	音声情報処理技術以外の新しい情報技術に対しても理解が深まりましたか？
VII	音声情報処理技術はどのようなことに活用されると考えますか？
VIII	その他 (プログラミングについて思ったこと，音声情報処理技術について思ったこと，感想 など)

4.4　授業実践に基づく評価

プログラミング教育において音声情報処理技術を活用することの有効性を検証するため，2018 年 11 月に岐阜県内の高等学校にて授業実践およびアンケート調査を実施した．対象は 2 年生 2 クラス（23 名と 36 名）の合計 59 名とし，実践としては JavaScript を用いる場合の方法を選択して授業を行った．アンケートの質問内容は表 4.5 に示すとおりで，選択式を 6 項目 (質問番号 : I〜VI) と記述式 (質問番号 : VII〜VIII) を 2 項目の合計 8 項目を設定した．選択式の項目については肯定から否定まで 4 つの選択肢（1: とてもそう思う，2: そう思う，3: そう思わない，4: 全くそう思わない）を設定した．

表 4.5 のアンケートを実施し，質問項目 I から VI の回答をクラスごとに集計した結果をそれぞれ表 4.6 と表 4.7 に示す．表 4.6 は理系のクラスに対応しており，全ての質問に対して肯定的な回答が多い．一方，否定的な回答は質問項目ごとに，2 名以下であった．表 4.7 は文理融合のクラスに対応しており，表 4.6 と比べるとやや否定的な回答も散見される．しかし，全体としては全ての質問に対して肯定的な回答が優勢である．

以上をまとめると，両クラスに共通する傾向として，全ての生徒がプログラミングに対する興味や関心をもち，多くの生徒が音声情報処理技術に対する理解が深まったと回答している．質問項目 VII と VIII は記述式であるが，2 クラスの回答内容に目立った傾向が見られなかったので，以下ではクラスを区別せずに考察する．質問

表 4.6: 理系クラスのアンケート結果

	I	II	III	IV	V	VI
1	52.2	34.8	26.1	34.8	52.2	39.1
2	47.8	65.2	65.2	56.5	43.5	60.9
3	0.0	0.0	8.7	8.7	4.3	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0

回答数 23 名，単位：%

表 4.7: 文理融合クラスのアンケート結果

	I	II	III	IV	V	VI
1	33.3	27.8	22.2	36.1	38.9	19.4
2	66.7	63.9	55.6	36.1	55.6	58.3
3	0.0	8.3	22.2	25.0	2.8	22.2
4	0.0	0.0	0.0	2.8	2.8	0.0

回答数 36 名，単位：%

VII には，次のような回答例があった．

- ドアの開閉
- 字が書けなくてもできる
- 字幕
- 医療現場
- 障がいの人の生活支援
- 手が使えない人，文字が書けないが話せる人がパソコン等に文字を打ち込む時

質問 VIII には，次のような回答例があった．

- よりよい未来のために役立つものだと思いました
- プログラミングで自分の思いもよらないものができるとわかった
- 音声認識技術は色々なところに活用されていてこれからも知っていく必要がある
- 知る前はとても難しいものだと感じていたけど，初歩的なものだったら素人でもできることが分かってより興味がわいた

- もっと知識を学びたいと思った
- こういう技術がもっと多くの所で活用され暮らしやすい社会になると思った
- この技術はとても画期的でこれからの将来たくさん使われていく技術だと思います．音声情報技術がどのように発達していくのかとても興味があります
- ものすごく難しいイメージがあったけどみんなでやってすごく楽しかった

上記より，肯定的な反応を示す生徒が多いことがうかがえた．一方，次のような回答もあり，プログラミングなどに困難を感じる生徒もいることがうかがえた．

- 難しいと感じた
- まずやってみてすごく複雑だなと思いました
- コンピュータが話すなんてすごいなと思った．難しかった

以上は生徒に対するアンケート調査の結果であるが，授業実践を行った教員から「いわゆる Print 文のように画面上に文字を表示することや何か計算をさせるような題材と比較し，生徒にとって興味をもちやすい題材だと感じた．特に近年スマートスピーカーが話題になっていること，生徒は自分のスマホを持っていることなど，音声認識機能が身近かつ便利に感じているため，その技術を自分で動かすことができることに達成感を感じているようだった.」などの意見が得られた．

第5章　開発した教材の考察

5.1　教材の拡張性

本研究では，学習者と指導者を支援する教材としてデバッグ支援システムと，音声情報処理技術を活用したプログラミング教材を開発した．これらの教材に共通していることは，プログラミングの学習に用いる言語や技術を他のものに置き換えること，すなわち拡張性を有していることである．

デバッグ支援システムについて述べた第3章では，C言語とArduino言語を用いたプログラミングの学習のデバッグ支援について述べた．第3章でも述べたようにArduinoで使用するコンパイラはC++言語用のg++である．C++言語はC言語の拡張言語であり，両者の文法は類似しており，コンパイラの出力するエラーメッセージは類似している．このように，コンパイラの出力するエラーメッセージが類似している場合，デバッグ支援システムの応用は比較的容易であるが，本デバッグ支援システムは，コンパイラの出力するエラーメッセージの内容に応じて提示する補助メッセージを変化させる仕組みであるため，コンパイラの出力するエラーメッセージがGCCと類似しないものであっても応用可能である．

たとえば，図3.3と同等の結果を出力するプログラムをJavaで記述すると図5.1のようなコードとなる．

図5.1のプログラムをJava用のコンパイラでコンパイルすると図5.2のようなエラーメッセージが表示される．

```
1 import java.util.*;
2
3 public class Main {
4   public static void main(String[] args) throws Exception {
5     int n = 3;
6     if(n%2=0)
7     System.out.println("偶数です\n");
8   }
9 }
```

図 5.1: 比較文を代入文に書き間違えた例 (Java 言語版)

```
Main.java:7: error: unexpected type
       if(n%2=0)
            ^
  required: variable
  found: value
1 error
```

図 5.2: 代入不能な定数に値を代入したときのエラーメッセージ（Java 言語版）

図 5.2 のエラーメッセージをデバッグ支援システムで分析し，エラーメッセージの内容を特定することで，C 言語や GCC を使用したプログラミングと比較した際と同等のデバッグ支援が Java 言語を用いる場合でも可能となる．

また，音声情報処理技術を活用したプログラミング教材について述べた第 4 章では，教材に用いる技術として音声情報処理を取り上げたが，それ以外の例として，脳波情報を用いたプログラミング教材も考えられる．図 5.3 にプロトタイプとして開発した本教材の概要を示す．

図 5.3 の教材では，簡易脳波計を用いて計測された脳波情報を PC で処理し，その PC に接続される IoT 機器へ独自のコマンドを送信することで，最終的にモータを回転させることができる．プログラミングの学習と融合させる場合は，たとえば，PC から IoT 機器へのコマンド送信をプログラミングで実現することが考えられる．

さらに，通信手段の拡張として IEEE802.11 規格の無線 LAN を選択したが，学校環境によっては無線 LAN 環境が整備されていない場合や生徒の無線通信を許可していない場合があるため，USB プロトコルによる有線通信の検討も行っている．昨今，マウスやキーボードのインターフェースとしても普及している USB は学校が導入する PC のインターフェースとして予備があることが多く，それを有効活用すれば教材と連携が行いやすい．図 5.4 に USB の有線通信による PC とマイクロコントローラ間の通信形態を示す．

既に本研究では，図 5.4 で示したような USB の有線通信により，PC とハードウェア (Peripheral Interface Controller，以後 PIC と記す) 間で正常に通信ができること

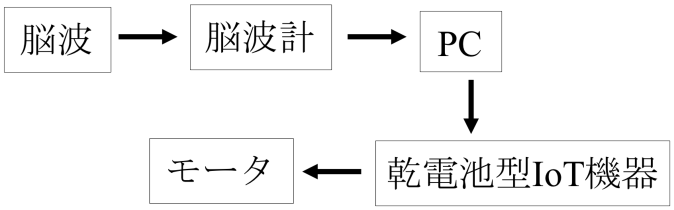

図 5.3: 脳波情報を用いたプログラミング教材の概要

図 5.4: USB による PC とマイクロコントローラ間の有線通信

を確認している．なお，ハードウェアとしては PIC18F14K50 を用いて C 言語でプログラムを記述し，PC から PIC へ情報を送受信するプログラムは WebUSB API と呼ばれる Web API を用いた JavaScript プログラムで動作を確認した．

5.2 最近の GCC コンパイラとの比較

比較的バージョンが新しい GCC では，コンパイルエラーメッセージを出力する際に，プログラムのエラー発生箇所を表示する機能が改善されている．

まず，文法誤りのエラーに関して，たとえば，図 3.2 で示したエラーを含むプログラムを GCC 9.2 でコンパイルすると図 5.5 に示すようなエラーメッセージが出力される．

図 5.5 のエラーメッセージでは，オリジナルのコンパイルエラーメッセージに加えて，エラー発生の要因となったソースコードの具体的な場所が「^ （キャレット）」の記号により示されている．

また，図 3.3 と図 3.4 に示したプログラムを GCC 9.2 でそれぞれコンパイルした場合，図 5.6 と図 5.7 に示すように，図 5.5 と同様，エラー箇所の表示を含むエラーメッセージが出力される．

これらの例に見られる GCC 9.2 の機能は，プログラミングの学習を重ね，ある程

```
1  prog.c: In function ’main’:
2  prog.c:4:1: error: stray ’\343’ in program
3      4 |   return 0;
4        | ^
5  prog.c:4:2: error: stray ’\200’ in program
6      4 |   return 0;
7        | ^
8  prog.c:4:3: error: stray ’\200’ in program
9      4 |   return 0;
10       | ^
```

図 5.5: 非アスキー文字が混入したときの GCC 9.2 が出力するエラーメッセージ

```
1 prog.c: In function 'main':
2 prog.c:4:7: error: lvalue required as left operand of assignment
3     4 | if(n%2=0)
4       |      ^
```

図 5.6: 代入不能な定数に値を代入したときの GCC 9.2 が表示するエラーメッセージ（条件式）

```
1 prog.c: In function 'main':
2 prog.c:4:2: error: lvalue required as left operand of assignment
3     4 | 2=i;
4       |  ^
```

図 5.7: 代入不能な定数に値を代入したときの GCC 9.2 が表示するエラーメッセージ（左辺値と右辺値が逆）

度プログラミングの文法を理解した学習者に対しては有用と考えられる．プログラミングの学習量が蓄積されるとともに，エラーを修正する経験も蓄積されるため，プログラムのエラーの原因が暗示されるのみでも自力で気づくことができる．

しかし，あくまでエラーの原因箇所が暗示されるだけであるため，プログラミングの学習経験が浅い初学者にとっては，エラーの原因箇所を特定できたとしてもエラーの修正方法までは認識できない可能性があり，その点を考慮すると GCC 9.2 の機能だけでは不十分と考えられる．たとえば，図 3.3 に示したプログラムに関して述べると，学習者が数値の代入に用いられる記号は等号であると認知している状態で，エラーの発生箇所を提示するだけでは，知識の再構築には至らずに学習が滞る可能性がある．

一方で，本研究で開発したデバッグ支援システムが提示する補助メッセージは，学習者の学びを促すとともに，指導者の負担を減らすための情報であり，学習者と指導者の支援を積極的に意図している．コンパイラの表示するメッセージは，あくまでプログラマのプログラミング効率を高めることが主な目的であり，この点が本研究との根本的な相違点である．

つぎに，綴り誤りのエラーに関して，たとえば，図 3.6 に示したプログラムを GCC 9.2 でコンパイルすると図 5.8 に示すようなエラーメッセージが出力される．

図 5.8 で示したエラーメッセージのように，GCC 9.2 においても図 3.6 のようなプログラムに対してはコンパイルを実行すると「did you mean 'func1'?」と綴り誤りの可能性を指摘するメッセージが含まれる．

しかし，図 5.9 で示すようなプログラムでは GCC 9.2 のエラーメッセージが初学者に対するデバッグ支援としては不十分な場合がある．

```
prog.c: In function 'main':
prog.c:4:1: warning: implicit declaration of function 'funcl'; did you
     mean 'func1'? [-Wimplicit-function-declaration]
    4 | funcl();
      | ^~~~~
      | func1
/tmp/cczpXTBF.o: In function `main':
prog.c:(.text+0xa): undefined reference to `funcl'
collect2: error: ld returned 1 exit status
```

図 5.8: 綴りを誤って書いたときの GCC 9.2 のエラーメッセージ

```
#include <stdio.h>
int func1();
int func2();
int main(){
 funcl();
 func2();
 return 0;
}
int func1(){
 printf("Hello World!");
 return 0;
}
int func2(){
 printf("Hello World!2");
 return 0;
}
```

図 5.9: 綴り誤りの修正案が適切に表示されない例

```
1 prog.c: In function 'main':
2 prog.c:5:1: warning: implicit declaration of function 'funcl'; did you
      mean 'func2'? [-Wimplicit-function-declaration]
3     5 | funcl();
4       | ^~~~~
5       | func2
6 /tmp/ccyMOKBJ.o: In function `main':
7 prog.c:(.text+0xa): undefined reference to `funcl'
8 collect2: error: ld returned 1 exit status
```

図 5.10: 綴りの訂正案が正確でない場合の GCC 9.2 のエラーメッセージ

図 5.9 のプログラムを GCC 9.2 でコンパイルすると，図 5.10 に示すようなエラーメッセージが出力される．

図 5.8 で示したエラーメッセージのように，GCC 9.2 ではプロトタイプ宣言として最後に書かれた関数名を綴り訂正候補として表示する．仮に関数の個数が増えて，たとえば「func3」と命名された関数がプログラムで定義され，プロトタイプ宣言の「func2」の下の行に「int func3();」の記述が追加されると GCC 9.2 が表示するエラーメッセージは「did you mean 'func3'?」となる．

第 3 章で述べたとおり，プログラミングの初学者にありがちな綴りの誤りは，字体が類似している数字の「 1 (イチ)」と英小文字の「 l (エル)」や数字の「 0 (ゼロ)」と英大文字の「 O (オー)」などの場合が多く見られる．ある程度素養のあるプログラマであれば単に入力ミスの可能性を指摘するコンパイラのメッセージで十分な場合も考えられるが，本研究で開発したデバッグ支援システムでは，初学者にありがちな誤りを意識した補助メッセージを出力する点が特徴である．

5.3 音声情報処理技術を活用したプログラミング教材の指導法

5.3.1 同音異義語の指導

音声認識結果の文字列と比較する文字列は生徒が自由に設定可能である. この点は生徒の興味や関心を引く要素として有効であるが，設定した文字列が同音異義語を持つ場合などは音声認識によって意図通りの文字列を取得できないことがある．そのままでは音声認識結果を用いてプログラムの処理を分岐させることはできないが，その問題を生徒がアルゴリズムを工夫する課題として利用することが考えられる．

```
1 if (text == "点灯"){
2   socket.send("\{\"command\":\"HIGH\"\}");
3 }
```

図 5.11: 生徒が作成するコードの例

```
1 if (text == "点灯"){
2   socket.send("\{\"command\":\"HIGH\"\}");
3 } else if (text == "転倒"){
4   socket.send("\{\"command\":\"HIGH\"\}");
5 }
```

図 5.12: 同音異義語を考慮したプログラムの修正例 1

ここでは，JavaScript を用いたプログラミングの学習を例に述べる．たとえば，以下のようなコードを生徒が記述したとする．

図 5.11 のコードでは，比較文字列が「点灯」と設定されているが，音声認識の結果が「転倒」となることもある．その場合は，その機会を利用して生徒に同音異義語が認識されても正常に動作するプログラムはどのようにかけるか，などと発問してプログラムの改良を指導するとよい．たとえば，

図 5.12 のようなプログラムへの変更が考えられる．

または，図 5.13 のようなプログラムとすることも考えられる．もしくは,「転倒」を「点灯」に書き換える以下のようなコードを指導することも考えられる．

これらのように，プログラムの書き方や処理方法が唯一でないことを実感させながら，生徒がプログラムの論理を考えられるようにすることはプログラミングの学習において重要で教育的に意義がある．図 5.14 の例のほかにも「特定の文字列が所定の順番で記録された場合」「特定の文字列が所定の回数記録された場合」に特定の処理を実行させるなど，生徒が多様なアルゴリズムを考えることにつなげ，プログラミングの学習がさらに発展的なものとなる．

```
1 if (text == "点灯" || text == "転倒"){
2   socket.send("\{\"command\":\"HIGH\"\}");
3 }
```

図 5.13: 同音異義語を考慮したプログラムの修正例 2

```
if (text == "転倒"){
  text = "転倒";
}
if (text == "点灯"){
  socket.send("\{\"command\":\"HIGH\"\}");
}
```

図 5.14: 同音異義語を考慮したプログラムの修正例 3

図 5.15: 音声認識の結果を利用して LED を点滅制御する Scratch スクリプト

5.3.2　出力形式の工夫

プログラムの論理だけでなく，プログラムの出力方法も生徒が工夫する余地がある．ここでは，Scratch を用いたプログラミングの学習を例に述べる．図 4.16 のようなスクリプトを作成した生徒は，それを改変して，図 5.15 に示すような LED を点滅させる Scratch スクリプトを作成することが予想される．

図 5.15 に示した Scratch スクリプトは，プログラミングの重要概念である「順次」「分岐」「反復」を取り入れたものとなっており，このようなスクリプトを作成することを通して，生徒はプログラミング的思考を身につけられる．また，図 11 のような Scratch スクリプトを作成した生徒に「変数の値を目視できない視覚障がい者のような人らに計測した内容を伝えるにはどうしたらよいだろうか」などの発問をすると生徒は，音声合成の活用が有効であることに気づき，図 5.16 のような Scratch スクリプトを作成することが期待される．

図 5.16 に示したようなプログラムの作成を通して，生徒は，音声合成の活用例を認識し，そのほかにどのような場面で社会や生活に役立てることができるか考える

図 5.16: 音声合成を追加した Scratch スクリプト

機会を有することで，他人に役立つ情報技術の在り方やユニバーサルデザインのきっかけも学習することができる．

第6章　おわりに

本研究では，プログラミング教育を対象として，学習者と指導者の双方を支援するプログラミング教材を開発し，それについて述べた．

第2章では，プログラミング教育における学習者と指導者の抱える問題点を明らかにした．学習者の問題は，本来学ぶべき学習内容が非優先的に学習すればよい内容によって学習時間が損なわれることと，典型的なプログラミングの学習題材では学習者の知的好奇心をや興味を引き出せないことであり，指導者の問題は，本来授業で教授する予定の内容が補助的に指導すればよい他の内容の教授に時間を占められることで教えられなくなることと，学習者の意欲を高める身近な技術を用いた教材の作成が容易でないことであった．

これらの問題点に対する2種類の方策を検討した．まず第3章では，初学者が作成したプログラムのコンパイルエラーに着目し，プログラムの修正案を提示することで初学者を支援する，デバッグ支援システムを開発した．本システムが提示する補助メッセージにより，プログラミング初心者が自力でデバッグ作業を進められる可能性が高まった．また，初学者は補助メッセージを活用したデバッグ作業を通して，それぞれのコンパイルエラーに対するプログラムの修正内容を学び，その学びの中で，補助メッセージがなくともコンパイルエラーのみでプログラムをデバッグできる能力を徐々に身につけて行くことが期待される．そして，学生のデバッグ能力が向上していくことにより，教員はデバッグ支援に要する時間を短縮でき，アルゴリズムの指導や発展的なプログラムを紹介するなど，より本質的なプログラミングの教育を展開可能となる．

つづく第4章では，音声情報処理技術をプログラミング教育に活用することを提案した．Web Speech API を用いることで音声情報処理技術を活用したプログラミングが容易に行えることを示し，さらに，マイクロコントローラも併用して音声指示により計測制御動作を行うプログラミング教材を紹介することで音声情報処理技術の活用方法も具体的に示した．この結果，教員は音声認識技術を活用したプログラミング教材を授業で用いることができ，新たな教材を作成することもできる．生

徒はプログラミングに対する興味や関心を持続させながらアルゴリズムを工夫するなど，学習を積極的に行うことができ，新しい情報技術に対する理解も深められる．

第 5 章では，本研究で開発した教材のデバッグ支援システムと音声情報処理技術を活用したプログラミング教材について考察した．デバッグ支援システムについては，最近のコンパイラの表示するエラーメッセージが教育的視点に配慮したものでないことを確認し，本研究の視点が独自のものであることを述べた．音声情報処理技術を活用したプログラミング教材については，教材を用いる際に，音声認識結果の文字列と比較する文字列は生徒が自由に設定可能であることに関して，生徒が同音異義語をもつ言葉を設定した場合は，プログラムの性能を高めるアルゴリズムの工夫として，条件分岐を応用した指導法について述べた．

プログラミングの学習を行った現在の小学生が大人となる頃，プログラミングは人々に広く認知され，国民の教養として存在している可能性が考えられる．その頃には，プログラミング教育に携わる教員もプログラミングの素養が高まっていることが期待されるが，この状況に変化し終えるのは今後何十年も先のことである．少なくともそれまでのプログラミング教育においては，学習者だけでなく指導者の支援も念頭においた教材の開発が必要である．

本研究で開発したデバッグシステムの今後は，初学者が作成しがちなエラーを生成させるソースコードをさらに蓄積して，それぞれのエラーに対応する補助メッセージを検討することで，より多様なケースに対応可能なデバッグ支援プログラムを構築する．また，音声情報処理技術を活用したプログラミング教材の今後は，その主要技術として音声認識と音声合成の基本的な機能を扱ったが，音声認識における話者の識別や音声合成における声質の変化などを活用することは今後の課題である．

謝辞

本研究は広島大学大学院教育学研究科の藤中透教授から様々な観点で多くのご指導を賜わり，博士論文として執筆することが出来たものです．論文執筆の合間には，研究そのものに対する考え方や臨み方，また，研究者としての在り方についてもご教授いただきました．ここに深く感謝の意を表します．

また，副指導教員として本研究をご指導いただきました広島大学大学院教育学研究科の渡辺健次教授，田中秀幸教授，長松正康教授には，終始貴重なご意見を賜りました．厚く御礼申し上げます。

参考文献

[1] 文部科学省. Society5.0 時代の教育・科学技術の在り方について, 2020. https://www5.cao.go.jp/keizai-shimon/kaigi/minutes/2019r/1113/shiryo_06.pdf（2020 年 1 月 18 日最終アクセス）.

[2] 岡本雅子・村上正行・喜多一・吉川直人. 初学者を対象とした自習中心のプログラミング教育の教材開発と評価. 情報教育シンポジウム 2010 論文集, 2010(6), pp. 87–94, 2010.

[3] 酒井統康・長谷川元洋. Sphero を用いた小学校プログラミング学習単元の開発. 日本科学教育学会研究会研究報告, 31(8), pp. 117–122, 2017.

[4] 佐藤慎一・飯田元・井上克郎. プログラムの依存関係解析に基づくデバッグ支援ツールの試作. 情報処理学会論文誌, 37(4), pp. 536–545, 1996.

[5] 江木鶴子・竹内章. プログラミング初心者にトレースを指導するデバッグ支援システムの開発と評価. 日本教育工学会論文誌, 32(4), pp. 369–381, 2009.

[6] 兼宗進・中谷多哉子・御手洗理英・福井眞吾・久野靖. 初中等教育におけるオブジェクト指向プログラミングの実践と評価. 情報処理学会論文誌, 44(13), pp. 58–71, 2003.

[7] 間辺広樹・長島和平・長慎也・並木美太郎・兼宗進. 高等学校における複数言語によるフプログラミング教育の提案〜情報システムの理解を目標としたドリトル，JavaScript，PHP の連携〜. 研究報告コンピュータと教育 (CE), 133(3), pp. 1–10, 2016.

[8] 大見嘉弘・滑川敬章・永井保夫. 情報系高校におけるセンサを利用したプログラミング教育の実践. 研究報告コンピュータと教育 (CE), 114(5), pp. 1–7, 2012.

[9] 滑川敬章・落合秀也・山内正人・高岡詠子・中山雅哉・江崎浩・砂原秀樹. 情報系高校における環境情報を計測・可視化する実用的なプログラミング教育の実践. 研究報告コンピュータと教育 (CE), 116(16), pp. 1–8, 2012.

[10] 村田育也・広田高. 5パズルを用いたプログラミング教材開発と高等学校情報科における授業実践. 北海道教育大学紀要教育科学編, 63(1), pp. 201–209, 2012.

[11] 木下崇・鎌田敏之・本多満正. Scratch の Mesh 機能を用いた双方向性のプログラミング教材の開発〜中学校技術科のネットワークを用いたコンテンツ制作の導入として〜. 愛知教育大学技術教育研究, 6, pp. 7–12, 2018.

[12] 秋山大翼・木下崇・本多満正. ネットワークを用いた双方向性のあるプログラミングの教材開発. 愛知教育大学研究報告芸術・保健体育・家政・技術科学・創作編, 67(2), pp. 15–19, 2018.

[13] 鎌田敏之・本多満正・木下崇・秋山大翼. ネットワークを用いた双方向性のあるプログラミングの授業へ向けた教員支援の試みとその評価. 愛知教育大学研究報告芸術・保健体育・家政・技術科学・創作編, 67(1), pp. 31–36, 2018.

[14] 西ヶ谷浩史・紅林秀治・青木浩幸・保福やよい・原久太郎・久野靖・兼宗進. ITクラフトマンシッププロジェクト〜中学生によるネットワークプログラミング〜. 情報処理学会研究報告コンピュータと教育（CE）, 16, pp. 173–180, 2006.

[15] 川井勝登・荻窪光慈・山本利一. ネットワークを利用した双方向性のあるコンテンツのプログラミングに関する指導過程の提案〜反転学習で活用する学習コンテンツの開発と授業実践〜. 埼玉大学教育学部附属教育実践総合センター紀要, 17, pp. 77–84, 2019.

[16] 滑川敬章・落合秀也・山内正人・高岡詠子・中山雅哉・江崎浩・砂原秀樹. 情報系高校における環境情報を計測・可視化する実用的なプログラミング教育の実践. 研究報告コンピュータと教育 (CE), 2012-CE-116(16), pp. 1–8, 2012.

[17] 間辺広樹・長島和平・長慎也・並木美太郎・兼宗進. 高等学校における複数言語によるプログラミング教育の提案〜情報システムの理解を目標としたドリトル, JavaScript, PHP の連携〜. 研究報告コンピュータと教育 (CE), (3), pp. 1–10, 2016.

[18] 中原久志・村長廉太郎. 双方向性コミュニケーションを行うブロック型プログラミングアプリケーションの開発. 大分大学教育学部研究紀要, 40(1), pp. 69–79, 2018.

[19] 島袋舞子・兼宗進. ドリトル言語における Leap Motion 対応と教育的利用の可能性. 情報教育シンポジウム 2014 論文集, 2, pp. 239–243, 2014.

[20] 鈴木聡. 視覚的プログラミング言語 Scratch を活用した計測制御教材の開発. 木更津工業高等専門学校紀要, 46, pp. 21–26, 2013.

[21] 後藤孔・藤中透. プログラミング教育におけるデバッグ支援. システム制御情報学会論文誌, 32(6), pp. 249–255, 2019.

[22] 後藤孔・藤中透. プログラミング教育における音声情報処理技術の活用. 日本情報科教育学会誌, 11(1), pp. 15–22, 2018.

[23] 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. http://www.mext.go.jp/b_menu/shingi/chousa/shotou/122/attach/1372525.htm（2020 年 1 月 18 日最終アクセス）.

[24] 古井貞熙. 音声情報処理技術. In Advanced Image Seminar ’98, 1998.

[25] 村上寛明・堀田圭佑・肥後芳樹・楠本真二. ソースコードの自動進化に向けて. 電子情報通信学会技術研究報告, 133(422), pp. 107–112, 2014.

[26] 鷲見創一・肥後芳樹・楠本真二. ソースコードの変更予測手法による自動プログラム修正の高速化. 電子情報通信学会技術研究報告, 116(277), pp. 73–78, 2016.

[27] 鷲見創一・肥後芳樹・堀田圭佑・楠本真二. 自動プログラム修正の修正可能バグ数に関する考察. 電子情報通信学会技術研究報告, 33(3), pp. 81–87, 2016.

[28] John Stasko James A. Jones, Mary Jean Harrold. Visualization of Test Information to Assist Fault Localization. In ICSE ’02 Proceedings of the 24th International Conference on Software Engineering, 2002.

[29] Abhik Roychoudhury Sergey Mechtaev, Jooyong Yi. Directfix: Looking for Simple Program Repairs. In ICSE ’15 Proceedings of the 37th International Conference on Software Engineering, 2015.

[30] Martin Rinard Fan Long. Automatic Patch Generation by Learning Correct Code. In POPL '16 Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2016.

[31] Aditya Kanade Rahul Gupta, Soham Pal and Shirish Shevade. Deepfix: Fixing Common C Language Errors by Deep Learning. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17), 2017.

[32] 伏田享平・玉田春昭・井垣宏・藤原賢二・吉田則裕. プログラミング演習における初学者を対象としたコーディング傾向の分析. 電子情報通信学会技術研究報告. SS, ソフトウェアサイエンス, 111(481), pp. 67–72, 2012.

[33] 日本政府. 日本再興戦略-Japan is Back・改訂 2015-. 2015.

[34] 日本政府. 世界最先端 IT 国家創造宣言. 2015.

[35] 教育再生実行会議. 第七次提言. 2015.

[36] 教育課程企画特別部会. 論点整理. 2015.

[37] 山西潤一・竹野英敏・高橋純・杉沼浩司・松原紀男. 諸外国におけるプログラミング教育に関する調査研究報告書. 2016.

[38] 高久雅生. Web API の過去・現在・未来. 情報の科学と技, 64(5), pp. 162–169, 2014.

[39] 鎌田篤慎. Web API 活用の現在 : Hack for Japan の活動の事例から. 情報の科学と技術, 64(5), pp. 175–180, 2014.

[40] 重元康昌・桑名良和・權娟大・菅原秀明. 生物分野における Web API の適用. 情報知識学会誌, 19(2), pp. 86–91, 2009.

[41] Julius. `http://julius.osdn.jp`（2020 年 1 月 18 日最終アクセス）.

[42] Julius Adorf. Web Speech API. In KTH Royal Institute of Technology, 2013.

[43] `https://www.arduino.cc/en/Guide/Introduction`（2020 年 1 月 18 日最終アクセス）.

[44] Espressif. ESP-WROOM-02 Datasheet. p. 4, 2016.

[45] https://scratch.mit.edu/about（2020 年 1 月 18 日最終アクセス）.

[46] https://nodejs.org/ja/about/（2020 年 1 月 18 日最終アクセス）.

付録A　ソースリスト

本研究で作成したプログラムのソースリストを以下に示す.

A.1　JavaScriptで音声情報処理を実行するapp.js

```
var
 rcgbtn = document.getElementById('rcgbtn'),
 socket,

  // 音声認識の利用
  recog = new webkitSpeechRecognition(),
  // 音声合成の利用
  synth = new SpeechSynthesisUtterance(),
  // 言語情報取得
  voices = window.speechSynthesis.getVoices();

  // 日本語を設定
  recog.lang = "ja";
  synth.lang = "ja";

function init(){
    socket = new WebSocket("ws://192.168.2.100:80");
    socket.onopen = function(e){
        alert("open websocket!");
    }
    socket.onmessage = function(e){
        var p = document.getElementById("msg");
        p.innerHTML = "計測データ: " + e.data;
        var obj = JSON.parse(e.data);
        if (obj.CdS >= 700) {
          synth.text = "明るいです";
          speechSynthesis.speak(synth);
        } else {
          synth.text = "暗いです";
          speechSynthesis.speak(synth);
        }
    }
    socket.onerror = function(e){
        alert("error!!");
    }
    socket.onclose = function(e){
        alert("close websocket.");
    }
}
init();

function sendOn(){
    socket.send("\{\"command\":\"HIGH\"\}");
```

```
}
function sendOff(){
    socket.send("\{\"command\":\"LOW\"\}");
}
function receiveValue(){
    socket.send("\{\"command\":\"MEASUREMENT\"\}");
}

rcgbtn.addEventListener('click', function() {

  // 音声認識が始まった時に行う処理を書く
  // 音声認識の開始
  recog.start();

});

recog.addEventListener('result', function(event){

  // 解析された「言葉」を使った処理を書く
  var text = event.results[0][0].transcript;
    console.log(text);
}, false);
```

A.2 Web サーバプログラム

```
var express = require("express");
var fs = require("fs");
var http = require("http");
var https = require("https");

var app = express();

var options = {
  key: fs.readFileSync('./key.pem'),
  cert: fs.readFileSync('./cert.pem')
};

app.use(express.static('htdocs'));

http.createServer(app).listen(80);
https.createServer(options, app).listen(443);
```