修 士 論 文

ソースコードの表示法とその実現

平成 25 年度 修了
三重大学大学院　工学研究科
博士前期課程　情報工学専攻
コンピュータソフトウェア研究室

橋爪　佑起

概 要

プログラムの理解支援において，関数間の関係やクラス間の関係を図で示す研究は多く存在する．しかし，ソースコード自体を見やすくする研究は少なく，既存のソースコード表示法には欠点がある．また，ソフトウェアのメンテナンスにおいて，コードを読むのに少なくとも半分の時間を費やす．そのため，ソースコード表示法の改善が必要である．そこで本研究では，プログラムのブロックをウィンドウ化し，条件分岐を並列に配置する表示法を提案する．これにより，ソースコード自体の可読性を向上させ，プログラム理解支援を行うことを目的とする．

ゲシュタルト心理学の群化の法則を元に，プログラムを理解する原理を考える．人間はあるものを見るときに，個別の要素がまとまって 1 つものとして感じとれる．この現象を群化と言う．これをプログラムの場合にあてはめて考える．プログラムの構成要素として，文やブロック，条件分岐のグループなどがあるが，この構成要素の間には，コメントやスペースを入れることで，群化の支援をすることが一般的である．このように，群化の支援がされているプログラムは見やすい．そこで，一般的なエディタにおいてどのような群化支援がされているかを見る．

一般的なエディタの表示法は欠点がある．一つは，ブロックの範囲が不明確なことである．字下げの量を変えるだけでは，ブロックの開始記号と終端記号の対応が取りにくい．これはブロックが群化しにくい要因となる．もう一つは，ソースコードを縦に長く表示していることである．縦長に表示するため，条件分岐のグループが画面内に収まらないときがあり，見にくさにつながる．さらに，画面の左側にコードが寄っているため，1 画面での情報量が少なくなるという欠点がある．

以上を踏まえて，プログラム構成要素の群化を支援する手法を提案する．まず，ブロックの群化を支援するため，ブロックのウィンドウ化を行う．これにより，ブロックの範囲が明確になる．次に，条件分岐グループの群化を支援するため，条件分岐を並列配置する．これにより，処理の流れにおいて分岐は並列方向に見ると決まるため，制御構造が明確になる．さらに，画面内の情報量も増加する．

提案手法をツールとして実現し，評価に用いる．このツールは C 言語ファイルを入力として，提案手法で説明したウィンドウ表示ができる．また，プレーンテキスト表示との比較評価をするため，プレーンテキスト表示の機能もある．

研究室に所属する学生を対象とし，簡単な C 言語のプログラムを読み，実行結果を答えるまでの時間を計測する評価実験を行った．この実験により，制御構造が複雑で長いプログラムにおいて，本手法を用いることで可読性が向上することが確認できた．

目 次

第1章　背景と目的

1.1　研究の背景

ソフトウェアの作成や，品質の維持，向上をはかるために，開発者はソースコードを読み，プログラムの構造，動作を理解する必要がある．そのために，プログラムのデータ構造や構成を図で表示することで，プログラム理解を支援する研究が行われ，理解支援ツールが開発されている [小泉 11]，[US]．しかし，図を新たに生成する方法はソースコードと図を見比べる必要があり，手間がかかる．そのため，そもそもソースコード自体が読みやすいことが重要であり，既存のソースコード表示法には見やすさを改善する余地がある．また，ソフトウェアのメンテナンスにおいて，コードを読むのに少なくとも半分の時間を費やす [Cor89]．そのため，ソースコード表示法の改善が必要である．

1.2　研究の目的

本研究の目的は，見やすさを改善するソースコード表示法を提案，実装し，既存の表示法と比較することで，その効果を評価することである．まずは，既存のソースコード表示法の欠点がどこにあるかを考えるため，ソースコードが「見やすい」とは何かを，人間の認識の理論について考えているゲシュタルト心理学における群化の法則の視点から考える．見やすいとは，プログラムを構成している 4 つの要素が認識しやすいことと考え，既存の表示法において，この構成要素に関する工夫が足りない部分を考える．ここから，既存のソースコード表示法はブロック，グループに関して工夫が足りないと考え，プログラムのブロックをウィンドウ化し，条件分岐を並列に配置する表示法を提案する．続いて，この手法を評価するため，プログラムの表示ツールとして手法を実装し，それを用いて学生にプログラムに関するテストを受けてもらう．これにより，既存の手法と比べてどのくらいの効果が得られるかを評価する．

第2章　見やすさの要因

既存のソースコード表示法は見やすさにおいて改善の余地がある．見やすいソースコードとは何かを考えるため，まず，人がどのようにプログラムを認識するのかをゲシュタルト心理学 [Wer23] の視点から考える．

2.1　ゲシュタルト心理学における群化の法則

物の形は，人間が物を認識する上で，重要な手掛かりとなる．このとき，人間が知覚するものは，個々の刺激要素ではなく，要素に還元できない全体性をもつ形態（ゲシュタルト，Gestalt）である．図形や形は，それを構成する点や線などの要素がまとまったもので，そのような知覚的なまとまりを形成することを，知覚的体制化という．ここでは，分かりやすさのためにまとまりを形成することを群化または，グループ化と呼ぶ．M. Wertheimer は，ゲシュタルトが知覚される際に働く，次に示すような心理学法則（ゲシュタルトの法則　群化の要因）を挙げた [西堀 09].

- 近接の要因：空間的，時間的に近いものがまとまりとして知覚されやすい
- 類同の要因：色や形などの類似性が高いものどうしがまとまりやすい
- 閉合の要因：閉じた領域を形成するものが知覚されやすい
- よい連続の要因：なめらかに特性が変化するものがまとまりとして知覚されやすい

これ以外にもいくつかの要因が挙げられているが，ここでは，ソースコードの理解において関係があるものだけを取り上げる．この法則について詳しく説明する．

まずは近接の要因を説明する．これは，空間的，時間的に近いものがまとまりとして知覚されやすいというものである．下図の場合，6 つの四角形があるとは認識されにくく，2 つの四角形が空間的に近いため，1 つのグループとして見え，グループが 3 つあるように知覚されやすい．

類同の要因は，色や形などの類似性が高いものどうしがまとまりやすいというものである．下図の場合，緑と青色の図形がグループとして見えるため，3 つのグループがあるように知覚されやすい．

閉合の要因は，閉じた領域を形成するものが知覚されやすいというものである．下図の場合，いくつかの”] [” の集まりではなく，いくつかの”[ ]” の集まりと認識されやすい．

よい連続の要因は，なめらかに特性が変化するものがまとまりとして知覚されやすいというものである．すなわち，連続する線もそのうちに入る．下図の場合，なめらかに変化していっている方向に線が続いているように見えるため，3 本の線が交差していると知覚されやすく，急に曲がっている線がいくつかあるとは知覚されにくい．

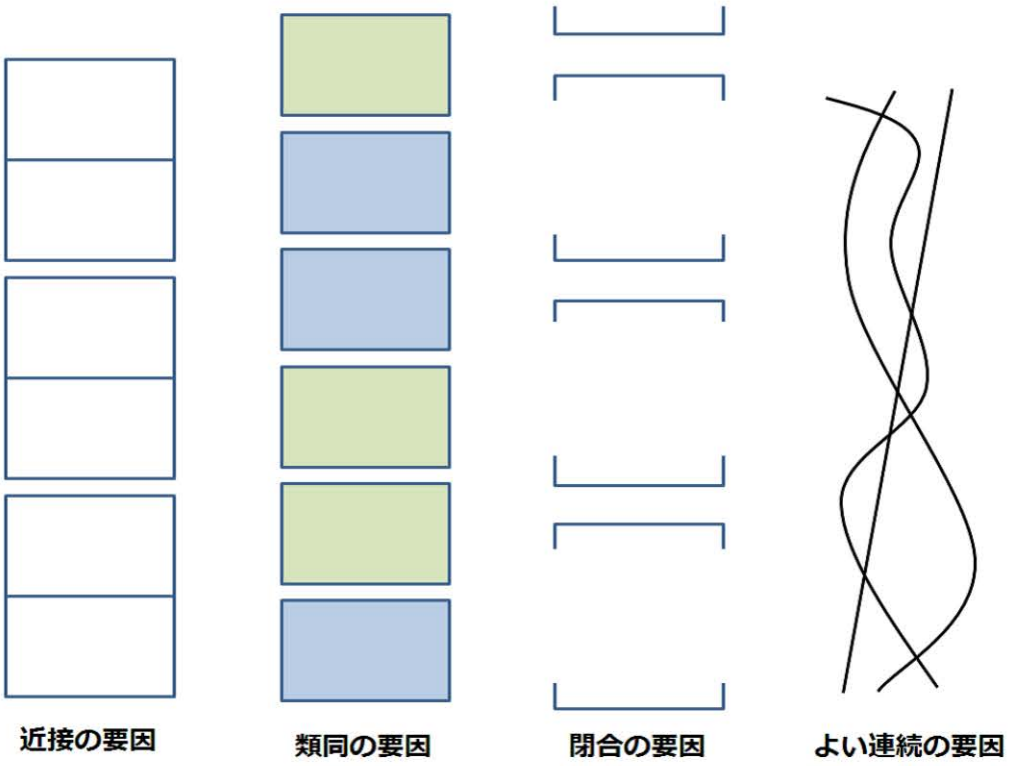


図 2.1: 群化の要因の一部

このように，人は構成要素を群化して，1つのものとして認識する．また，この構成要素に何らかの要因によるまとまりがあれば，群化しやすいと言える．そこで，人はどのような構成要素を群化してプログラムの内容を理解するのかを分析する．

2.2　プログラムの４つの構成要素

プログラムは次の 4 階層の構成要素から成り立つと考える．

- 文
- ブロック
- グループ
- 関数

これらの要素が認識しやすい，すなわち，群化を支援する工夫がされていて，見てすぐに要素が把握できるものは見やすく，情報がまとまっていると言える．これらの要素について図 2.2 を例に詳しく定義する．

```
int main(void){
  int i,j;

  for(i = 0;i < 10;i++){
    test1();
  }
  /**コメント**/
  if(i == 0){
    test2();
  }else if(i == 1){
    test3();
  }else {
    test4();
  }

  while(i !=0){
    test5();
  }
  printf("%d",i);
}
```

図 2.2: プログラム例

2.2.1　文

複合文を含まない単純な文のことを指す．下図のプログラム例で説明すると，`"int i,j;"` のような変数定義の文，`"test1();"`や`"printf("%d",i);"`のような関数呼び出しを指す．

2.2.2 ブロック

複合文を含む文のことを指す．下図のプログラム例で説明すると，"int main(void) { 〜 }" "for(i=0;i<10;i++) { 〜 }", "if(i == 0){ 〜 }", "else if(i == 1){ 〜 }", "else { 〜 }", "while(i != 0){ 〜 }"は 1 つのブロックである．このように，予約語と条件式も含んで 1 つのブロックと呼ぶ．

2.2.3 グループ

プログラムは意味のあるグループが順番に並んでいると考える．ここでグループとは，似たような意味を持つ文とブロックのまとまりとする．

グループは一般的に，スペースで間を区切るか，コメントを挟むことにより，一目でまとまりが判別できるようにすることが多い．下記のプログラム例で説明すると，関数 main は変数定義の部分，for 文の部分，条件分岐の並びの部分，while 文と出力の部分で大体グループにわかれている．また，大きなグループの中に小さなグループを含む場合もある．本研究では，この条件分岐の連なりを 1 つのグループとして考えることが重要である．

2.2.4 関数

関数はグループの集まりによって構成されていると考える．2.2.3 節で説明したように，連続するグループによって構成されているため，その制御構造が明白である関数は内容が伝わりやすいと考える．

2.3 まとめ

一般的にプログラムを書く際に，共通するグループをスペースやコメントにより区切ることが多い．近接の要因から，人間はその区切られたグループを知覚しやすくなる．人間はこのように自然と自分自身の群化を支援していることになる．そこで，4 つのプログラム構成要素 (文，ブロック，グループ，関数) に関して，群化を支援する工夫がされているソースコードは見やすいと考える．

次の章では，既存の手法や研究では 4 つの要素に関してどのような工夫がされているかを分析する．

第3章　既存の手法

プログラム表示の既存の手法としては，プレーンテキスト表示が一般的である．この表示法を使っているツールとして，Microsoft の Visual Studio がある．また，プレーンテキスト表示以外にも，Control Structure Diagram（CSD）を使った表示法が研究されている．この 2 つの項目に関して，4 つのプログラム構成要素においてどのような工夫を行っているかを分析する．

3.1　Visual Studio

図 3.1 は VisualStudioExpress2013[VS] にてプログラムを表示した場合の最初の状態である．

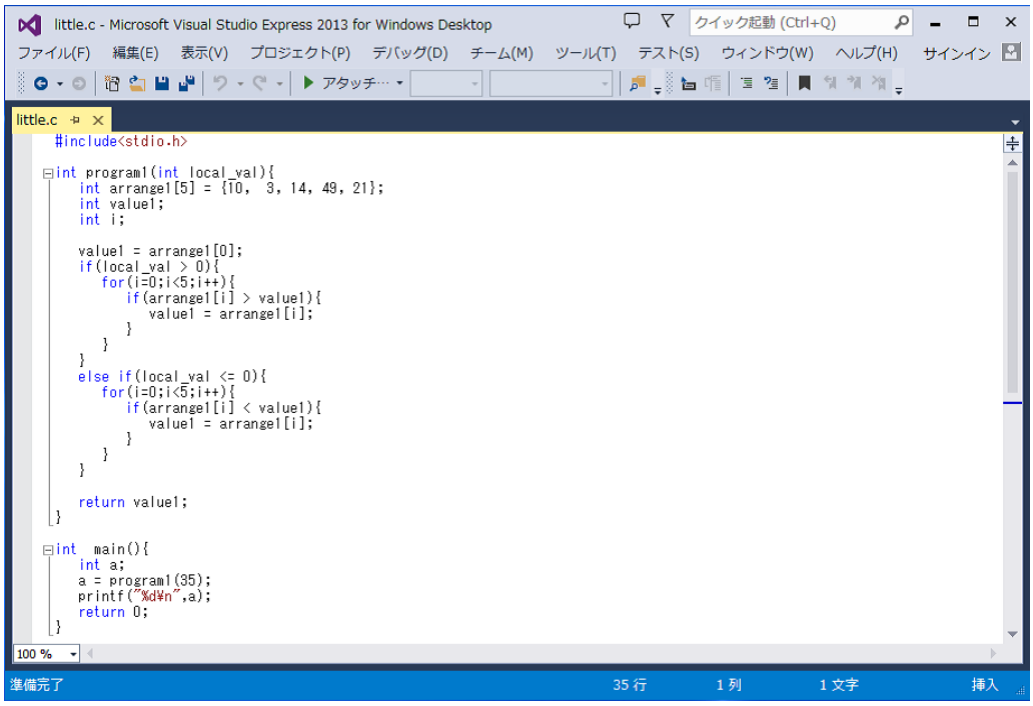


図 3.1: VisualStudioExpress2013

このツールについて，コードの見た目に影響する機能をまとめる．まず，特定の予約語に色をつけることで，他の文字列と区別し，強調している．次に，折りたたみ機能があり，余計な関数を折りたたんで，コンパクトに表示できる．この折りたたみ範囲はユーザによる選択もできる．次に，画面の左側に行番号の表示ができる．デフォルトの設定では，図 3.1 のように表示されていないが，図 3.2 のように画面の左側に行番号の表示ができる．また，画面のスクロールやズームイン，ズームアウトもできる．

これらの機能以外にもソースコードを見やすくする機能がある．その中でも特徴的な機能として，マップモードがある．

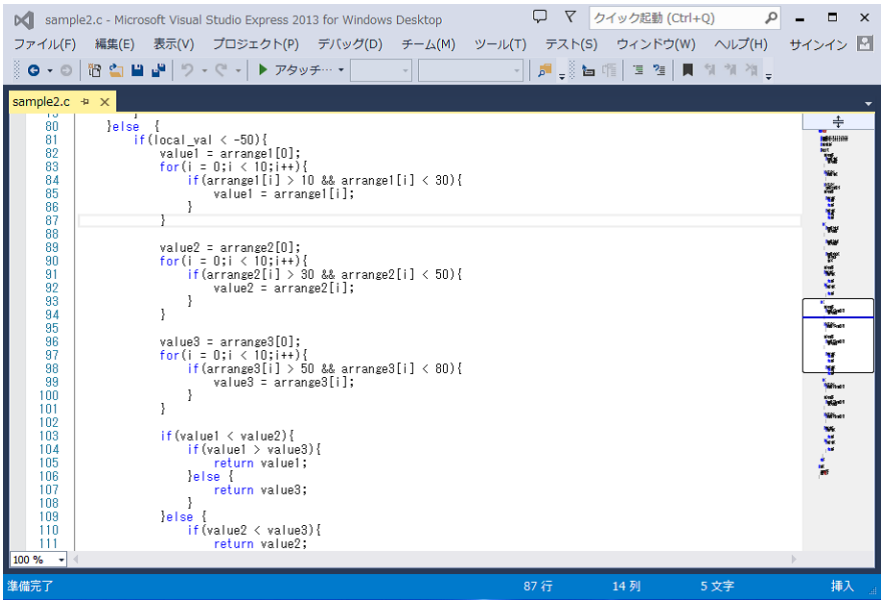


図 3.2: マップモード

マップモードでは図 3.2 のように，右側にコード全体のどの位置を見ているのかがひと目で分かるマップが表示される．マップで見たい位置をクリックすれば，そこにジャンプができる．この機能により，全体において見ている位置を把握しながら詳細を見られる．さらに，図 3.3 のように 2 画面表示ができ，配列の値を見ながらコードが見られる．

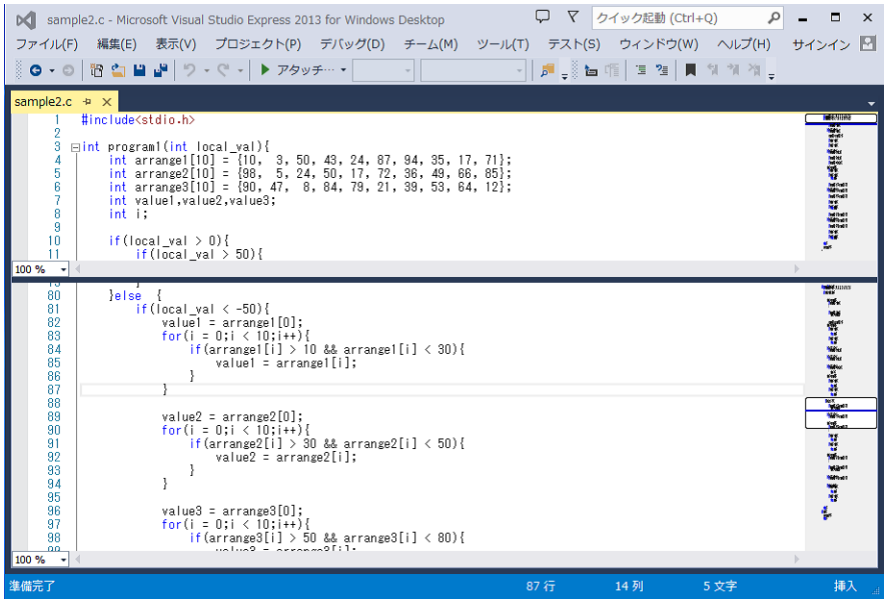


図 3.3: 2 画面表示

3.2 CSD

ソースコードを見やすく表示する既存の手法として，Control Structure Diagram（CSD）がある．CSD は jGRASP[JGR] というツールで表示できる．CSD は自動生成ができ，現在は Java，C，C++，Python，Objective-C，Ada，VHDL といった言語に対応している．各々のプログラム単位において，制御構成，制御パス，全体構造を明確に示すことで，ソースコードの分かり易さを改善できる．手動で使用する場合も自動生成する場合も，簡単でコンパクトな記法であることを目標として考案された記法である [CSD].

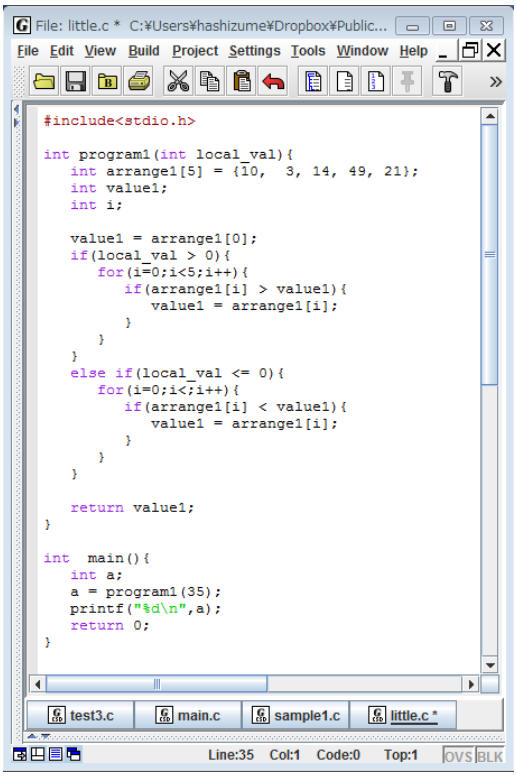


図 3.4: プレーンテキスト表示

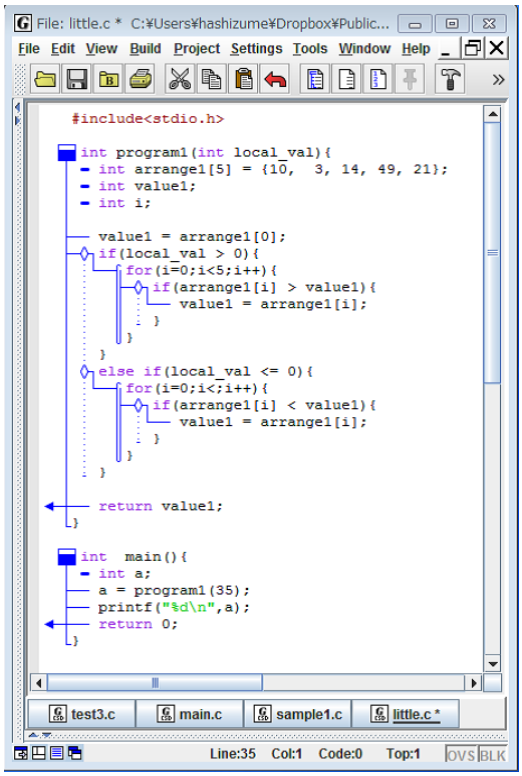


図 3.5: CSD 表示

図 3.4 は jGRASP のプレーンテキスト表示であり，図 3.5 は CSD を付加した表示である．図 3.5 を見ると分かるように，CSD は今までの情報に加えてフローチャートのように，実行の流れに沿って記号と流れ線を用いて，順次実行，分岐および繰り返しを強調している．それぞれの記号について図 3.6 にて説明する．他にも記号は用意されているが，ここでは一部だけを紹介する．

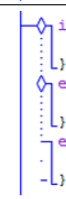
記号	説明
	関数定義を示す.
	変数定義を示す.
	制御構造を示すものであり，順次実行を意味する.
	制御構造の条件分岐を示す. フローチャートのように「真/偽」の分岐部分をひし形で表す. 真の場合はひし形の右の実線の方をたどる. 偽の場合は破線をたどる.
	制御構造の繰り返し実行を示す.
	矢印の方向に抜けるという意味で，return文を示す. break文も同じ表記である.

図 3.6: CSD の記号

CSD の特徴は，従来の表示法を崩すことなく，そのソースコードの表示部分に情報を加えることで理解支援をしていることである．また，折りたたみ機能が充実しており，各ブロックが折りたためるのに加えて，連続する条件分岐を一度に折りたためる（グループの折りたたみ）．また，コードの見た目に影響を与えるような他の機能については，予約語と文字列に色をつける機能と行番号を付与する機能がある．

CSD により制御構造が分かりやすくなっている．しかし，群化の法則から考えると，この表示法はブロックの群化を妨げている．次節でこの欠点について述べる．

3.3 既存の手法の欠点

これら 2 つのツールにおいて，2.2 節で述べたプログラムの 4 つの構成要素に関してどのような工夫がされているかを考察し，これらの表示法の欠点について述べる．

VisualStudio と jGRASP において，4 つのプログラム構成要素に関してどのような工夫を行っているかを図 3.7 にまとめる．

	VisualStudio	jGRASP
文	・予約語，””で囲まれた文字列に色をつける ・画面外に出た場合の自動改行	・予約語，””で囲まれた文字列に色をつける
ブロック	・折りたたみ機能(手動で選択した場合)	・折りたたみ機能
グループ	・折りたたみ機能(手動で選択した場合)	・折りたたみ機能 連続する条件分岐を一度に折りたためる
関数	・折りたたみ機能	・折りたたみ機能
その他の機能	・マップモード ・スクロール ・ズーム ・行番号の表示	・スクロール ・CSDの表示 ・行番号の表示

図 3.7: VisualStudio と jGRASP の比較

どちらも文字に色をつけることで，予約語と通常の文字列を区別している．また，スクロールや行番号の表示といった基本的な機能を持っている．また，折りたたみ機能が充実している．

ソースコードが大きくなり，条件分岐，ネストの深さが増えたときに，画面内に収まらないためブロックやグループが見にくくなる (図 3.8)．このような場合には，折りたたみ機能が有効である（図 3.9）．

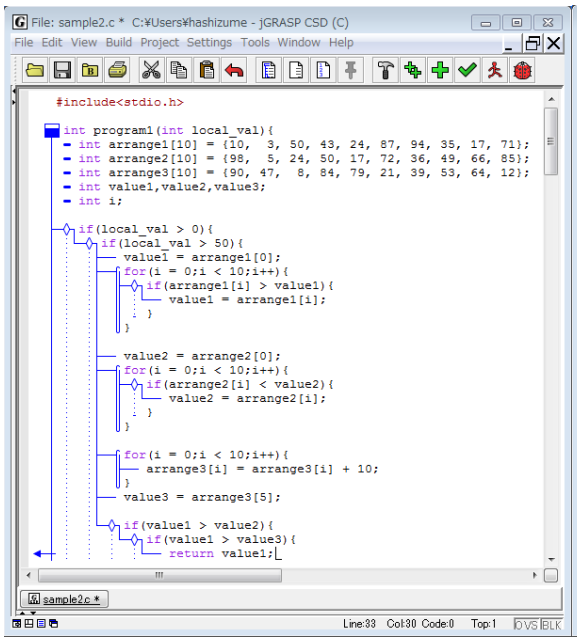


図 3.8: 折りたたみ前

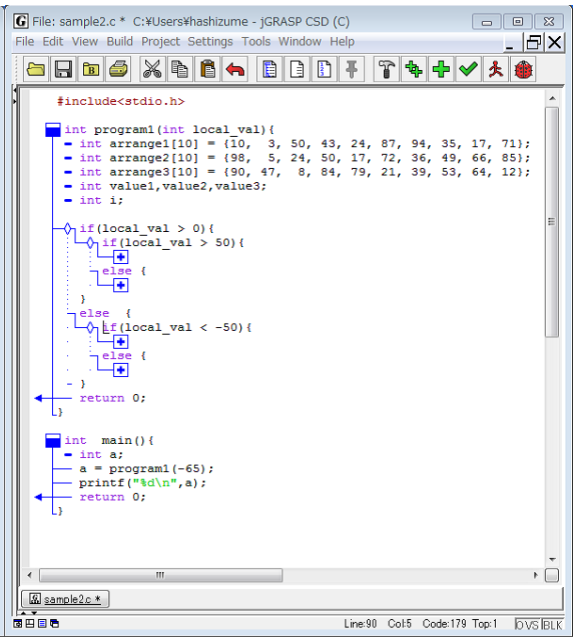


図 3.9: 折りたたみ後

しかし，折りたたみ機能には欠点があり，詳細の情報と全体のアウトラインの情報の両立はできない．図 3.8 のように詳細を見る場合には，全体の制御構造が把握しづらくなり，図 3.9 のように全体の制御構造を見る場合は詳細が見えなくなる．

次の欠点は，ブロックの範囲が分かりづらいことである．図 3.10，3.11 では，字下げの量を変えることで，ブロックの範囲を示している．しかし，このように字下げの量を変えるだけでは，ブロックの開始記号とブロックの終端記号の対応が取りにくいため，ブロックの範囲が把握しづらい．これは，閉合の要因の観点からも言えることである．人は閉じた領域をひとまとまりとして認識しやすい．しかし，この場合は，互いに閉じあっているものが無いため，ブロックの範囲は把握しづらい．また，jGRASP においては，制御の流れ線が加わったことにより，ブロックの開始，終端記号の対応が取りにくくなっている．

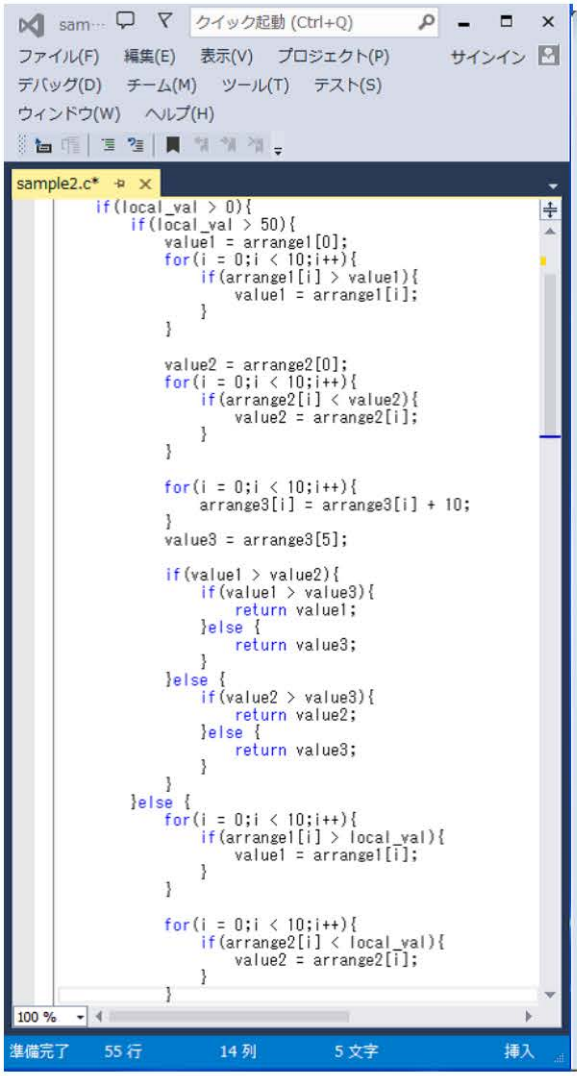


図 3.10: VisualStudio

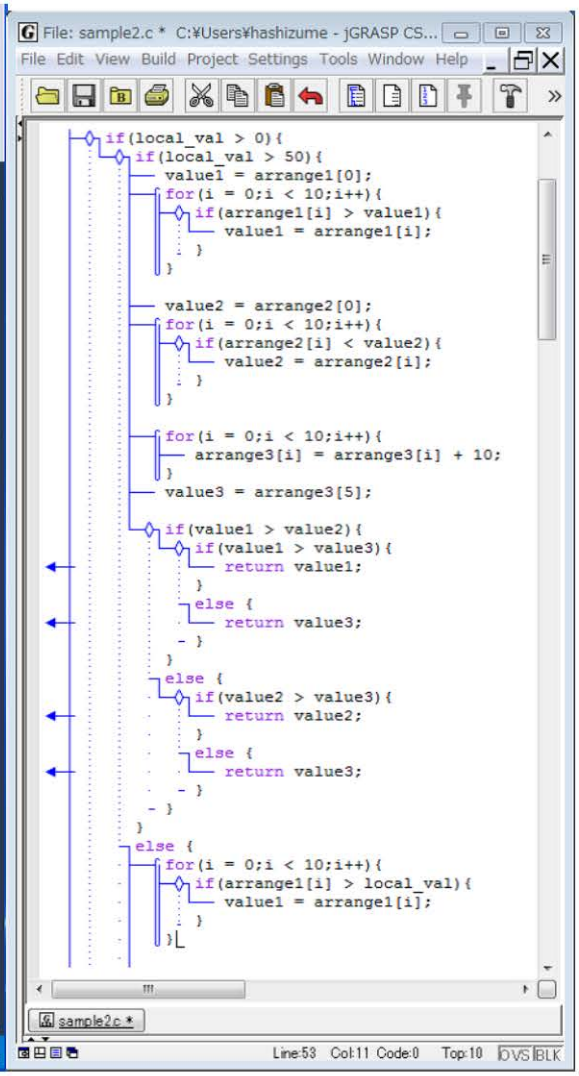


図 3.11: jGRASP

最後の欠点はソースコードを縦長に表示していることである．縦長に表示することで，1つのブロックやグループが分かりづらくなってしまう場合がある．図 4.1 の例を用いて説明する．例えば，条件分岐グループは 3 つのブロックからなっているが，画面に収まらないために，いくつの分岐があるのかがひと目では分からない．すなわち，条件分岐のグループが認識できないと言える．また，制御構造全体としては，for 文の処理があって，3 パターンの分岐があって，while 文の処理があるという構造をしているが，長く表示しているために，ひと目でその制御構造は把握できない．また，ソースコードは左詰めで書くため，必然的に右側に空白部分が多くなってしまい，画面内の情報量が少なくなる．

第4章　提案手法

条件分岐の部分は横に並べることで，縦方向に短く表示でき，右側の空白部分は有効に使える．これにより，縦方向では無理だったものが，面内に収まる場合が出てくる．プログラムの構成要素をなるべく画面内に収めることで，視覚的分離を避けられ，結果プログラム全体の制御構造の把握につながる．例えば，図 4.1 では，赤い枠を見えている画面の部分としたときに，else 文と while 文は画面内に入っておらず，認識できない．しかし，図 4.2 ではすべてのブロックが認識できる．このような場合において，図 4.2 のほうが見やすいのは明らかである．また，図 4.1 に比べて図 4.2 は画面内にあるブロック数が増えるため，単純に情報量が多くなる．

しかし，囲いもなく並列に配置すると，図 4.3 のように見にくくなる．そのため，ブロックを並列に配置するには，上下方向の範囲を示すだけでなく，左右の方向にもそのブロックの範囲を明確にするようなブロックのウィンドウ化が必要である．

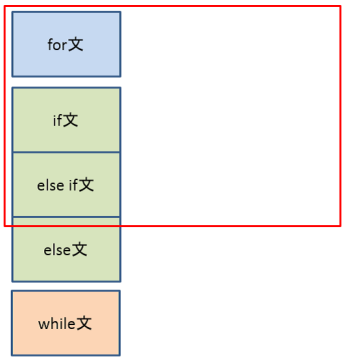


図 4.1: 配置例 1

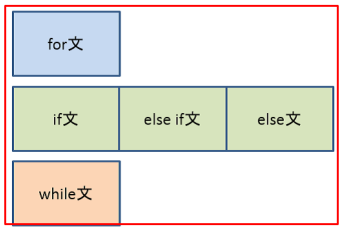


図 4.2: 配置例 2

```
#include<stdio.h>

int program1(int local_val){
   int arrange1[5] = {10,  3, 14, 49, 21};
   int value1;
   int i;

   value1 = arrange1[0];
   if(local_val > 0){
      for(i=0;i<5;i++){
         if(arrange1[i] > value1){
            value1 = arrange1[i];
         }
      }
   }
   else if(local_val <= 0){
      for(i=0;i<5;i++){
         if(arrange1[i] < value1){
            value1 = arrange1[i];
         }
      }
   }
   return value1;
}

int  main(){
   int a;
   a = program1(35);
   printf("%d¥n",a);
   return 0;
}
```

図 4.3: 囲いなしで並列配置した場合

4.1 ブロックのウィンドウ化

ブロックのウィンドウ化では，図 4.5 のように 1 つのブロックを 1 つのウィンドウとして表示し，入れ子構造にする．ウィンドウ化することで，閉合の要因の観点から，閉じた領域はより群化しやすいため，ブロックの範囲は明確になる．また，ウィンドウ上部の文字は通常の文字より大きく表示する．ブロックの始まりを強調することで，ブロックをより明確に表示できる．そして，ウィンドウ内の文は基本的に自動改行をせず，ウィンドウの幅はそのウィンドウ内の文の最大長に依存させる．

4.2 条件分岐の並列配置

ブロックをウィンドウ化したことで，ブロックを並列に配置した際にもそれぞれを区別しやすい．これにより，条件分岐のブロックは並列に配置できる．

並列配置とは，図 4.5 の黄色のウィンドウのように，if 文，else 文のような条件分岐のブロックが続く場合に，ウィンドウを横に連結する．これにより，一般的な書き方に比べ，横のスペースを有効に活用できるため，画面内の情報量が増加する．また，条件分岐の部分は縦方向に小さくなるため，プログラムを縦に読むことで処理の概要が見えやすくなる．また，処理の流れを見る際に，縦長の場合，図 4.6 の 3 色のルートを見るため，制御構造が把握しにくいが，図 4.7 のように並列に配置してあれば，分岐が見やすくなり，流れを追うのが容易になる．そのため図 4.6 に比べて図 4.7 は制御構造が明確である．

```
#include<stdio.h>

int program1(int local_val){
    int arrange1[5] = {10,  3, 14, 49, 21};
    int value1;
    int i;

    value1 = arrange1[0];
    if(local_val > 0){
       for(i=0;i<5;i++){
          if(arrange1[i] > value1){
             value1 = arrange1[i];
          }
       }
    }
    else if(local_val <= 0){
       for(i=0;i<5;i++){
          if(arrange1[i] < value1){
             value1 = arrange1[i];
          }
       }
    }

    return value1;
}

int  main(){
    int a;
    a = program1(35);
    printf("%d¥n",a);
    return 0;
}
```

図 4.4: プレーンテキスト表示

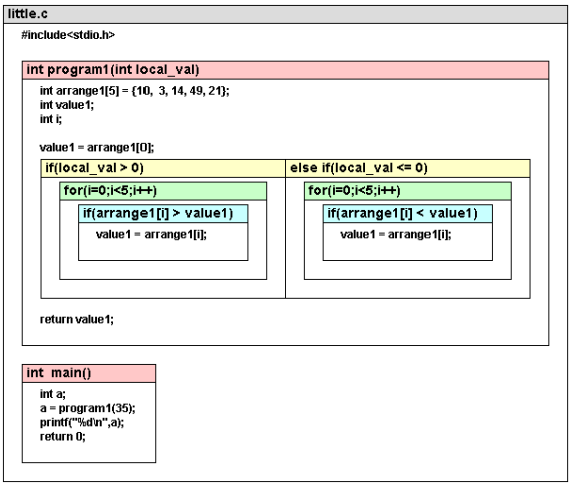


図 4.5: ウィンドウ表示

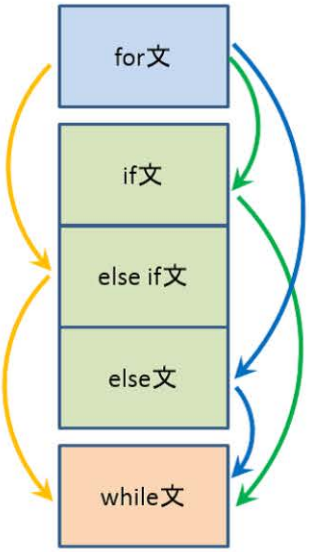


図 4.6: 処理の流れ 1

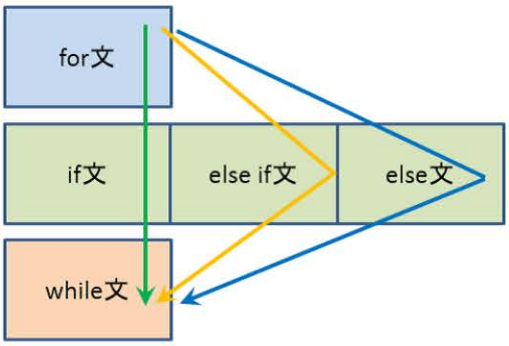


図 4.7: 処理の流れ 2

類同の要因の観点から言うと，色や形，向きが同じものはグループとしてとらえやすい．そのため，一般的な表示法では，図 4.6 のようにすべてのブロックが同じ縦方向に連結し，他のブロックの連なりとの違いがないため，条件分岐のグループが見えにくくなる．条件分岐のブロックは横方向に連結させることで，条件分岐のグループがより強調され，通常のブロックの連なりと区別しやすくなる．これにより，条件分岐が群化しやすくなり，プログラムの制御構造が見やすくなる．

4.3 欠点と対策

4.1，4.2 節の手法により，見やすさは改善できるが，欠点もある．まず，ウィンドウを並列に配置するだけでは，同じ段であるということが分かりづらい（図 4.8）．高さがバラバラであると，全体がグループとして認識しにくい．よい連続の要因から，切れ目や変化の無い線を 1 つのものとしてみるため，ウィンドウの高さをそろえることで，同じ段であることを示す（図 4.9）．

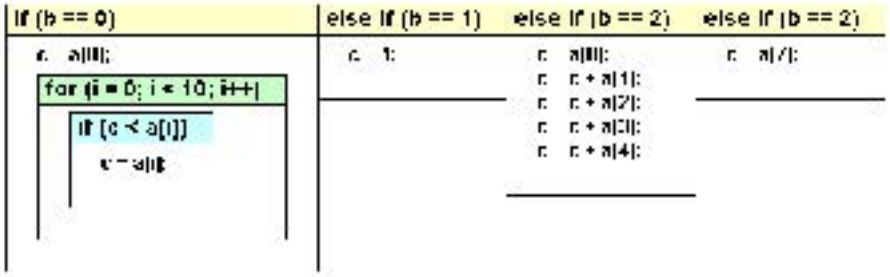


図 4.8: 高さを揃えない場合

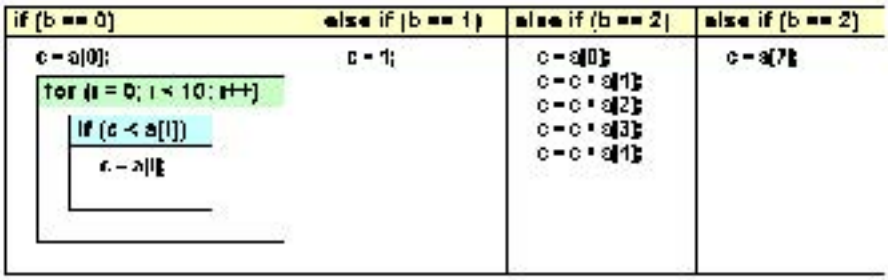


図 4.9: 高さを揃える場合

次の欠点は，条件分岐が多くなったときに，ブロックが横方向に画面外に出てしまうことである（図 4.10）．この場合は，段を 1 つ替えて連結させる．しかし，段を替えることでそれらが群化されにくくなり，別のグループに見える場合がある．そこで近接の要因より，近くにあるブロックほど同じグループとしてとらえられるため，次の段は前の段にくっつけておく（図 4.11）．さらに，ネストの深さに応じてウィンドウ上部に背景色をつけることで，同じグループであることを強調する（図 4.5，4.11）．

段を替えるアルゴリズムについては，5.2 節で説明する．

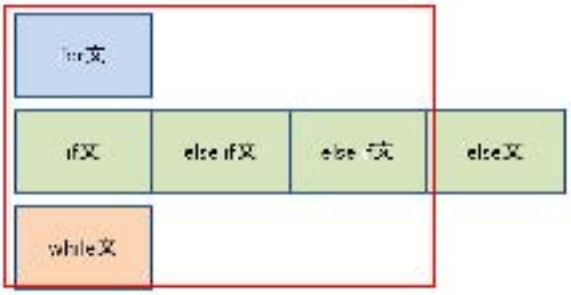


図 4.10: 段替えなし

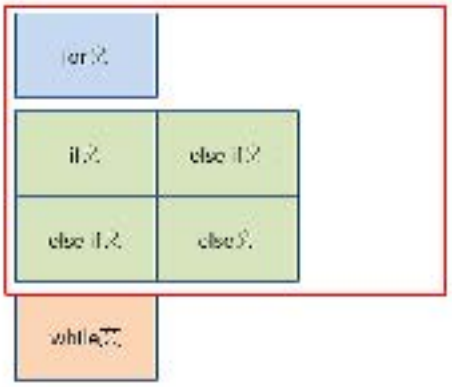


図 4.11: 段替えあり

第5章　実装

4章の手法を評価するため，まず，提案手法をツール（WindowPrograming）として実現した．このツールの基本情報，機能について説明する．

5.1　ツールの説明

図5.1はWindowPrograming にてプログラムを表示したときの状態である．

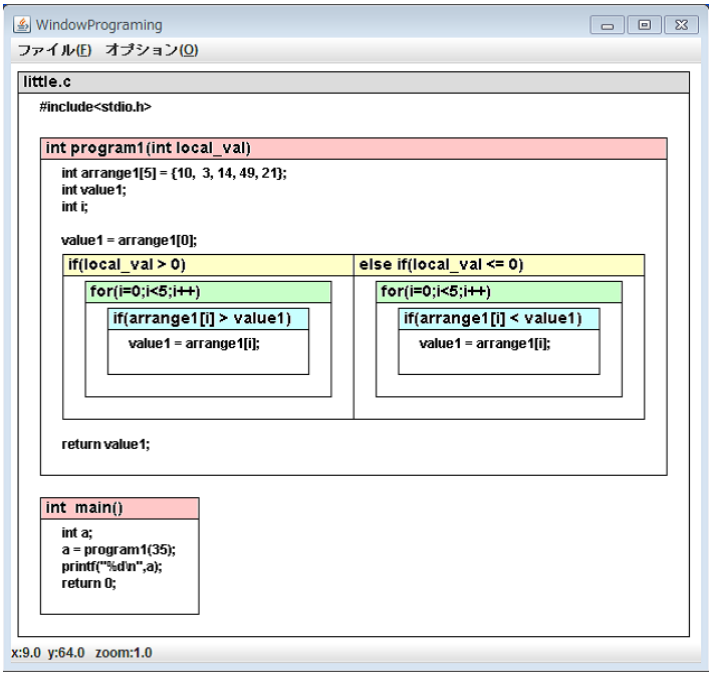


図 5.1: WindowPrograming

このツールはJava言語で作成した．入力できる言語はC言語のみである．構文には制限があり，変数定義の文，代入文，関数定義，if文，else文，for文，while文，複合文は読めるが，switch文，do-while文，include以外のプリプロセッサ制御文，構造体には対応していない．また，現在は編集の機能はなく，表示と簡単な操作のみの機能をもつ．

操作とツールバーについては図5.2，5.3で説明する．

操作説明	
マウス左ボタン長押し ＋ ドラッグ	ウィンドウ全体の移動
ホイールボタン上下	画面の上下スクロール
マウス右ボタン長押し + 上下移動 左Ctrlキー ＋ ホイールボタン上下	ズームイン，ズームアウト
ホイールボタンクリック	並列配置部分の段を変える
左Shiftキー + ホイールボタンクリック	並列配置にする

図 5.2: 操作説明

ツールバー説明		
[ファイル]	[新規]	まっさらな新規ファイルの読み込み
	[ファイル名を指定して読み込み]	C言語ファイルの読み込み
	[再読み込み]	指定したファイルを読み込みしなおす
[オプション]	[並列配置]	条件分岐の並列配置のon/off
	[段替え]	並列配置の際に制限を越えたウィンドウの段を替えるかどうかのon/off
	[段ごとに高さを調整]	並列配置の際に高さをそろえるかどうかのon/off
	[プレーンテキスト表示]	プレーンテキスト形式での表示をするかどうかのon/off

図 5.3: ツールバーの説明

5.2 段替えのアルゴリズム

横幅に制限を設け，その制限を越えるような段が出た場合に，その段を 2 分割，3 分割，…，n 分割の順番で分割していき，制限に収まる n 分割を探索する．横幅の制限は，文字の見える限界まで最大限にズームアウトしたときの画面幅とする．このツールの場合，0.71 倍を文字の見える限界の倍率として設定している．図 5.4 は倍率を約 0.71 倍で表示のときの例である．図 5.5 のように 0.71 倍を下回ると文字がつぶれて見えなくなる．図 5.6 の場合は約 0.71 倍で表示したときに 3 分割で収まった場合の状態である．これは，2 分割を先に実行し収まらなかったため，3 分割を実行し，探索し終えている．

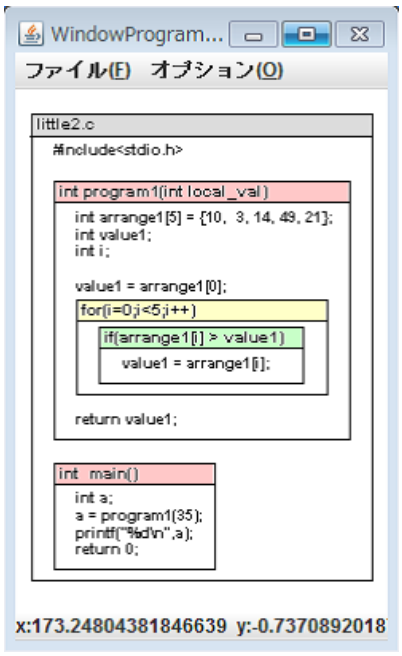


図 5.4: 0.71 倍

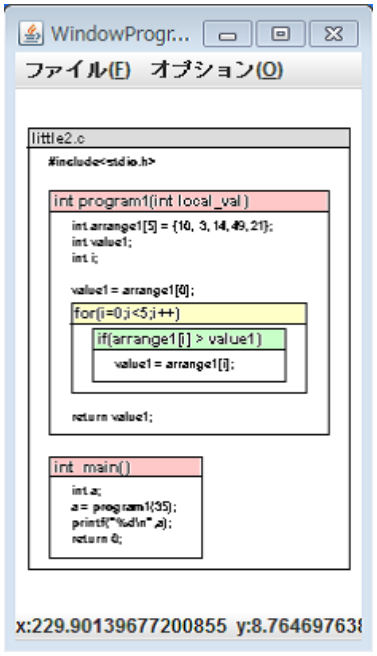


図 5.5: 0.69 倍

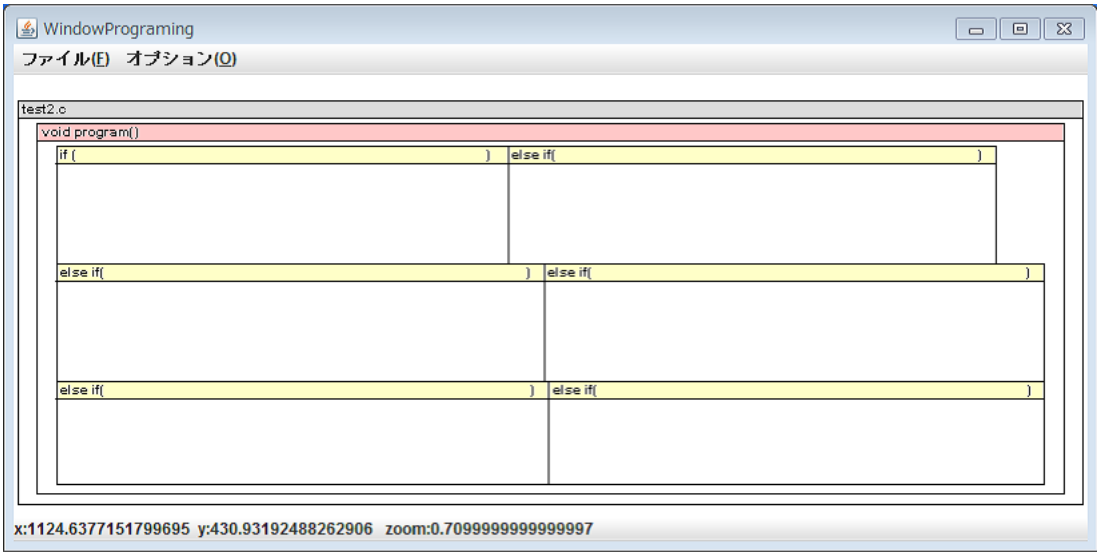


図 5.6: 3 段に分割した場合

5.3　まとめ

4 つの構成要素について，どの機能が群化を支援しているかを下図にまとめる．

	VisualStudio	jGRASP	WindowPrograming
文	・予約語，””で囲まれた文字列に色をつける ・画面外に出た場合の自動改行	・予約語，””で囲まれた文字列に色をつける	・ウィンドウ上部の文字を大きく表示
ブロック	・折りたたみ機能(手動で選択した場合)	・折りたたみ機能	・ブロックのウィンドウ表示 ・ウィンドウ上部の文字を大きく表示 ・ネストの深さに応じたウィンドウ上部の背景色
グループ	・折りたたみ機能(手動で選択した場合)	・折りたたみ機能 連続する条件分岐を一度に折りたためる	・条件分岐の並列配置(画面の大きさに応じて段を替える) ・ネストの深さに応じたウィンドウ上部の背景色
関数	・折りたたみ機能	・折りたたみ機能	・ブロックのウィンドウ表示 ・ウィンドウ上部の背景色
その他の機能	・マップモード ・スクロール ・ズーム ・行番号の表示	・スクロール ・CSDの表示 ・行番号の表示	・スクロール ・ズーム

図 5.7: 3 つのツールの群化に関する機能の比較

従来の手法に比べると，ブロックとグループに対する群化をよく支援していることが分かる．この群化支援が見やすさにどれくらい影響を与えるのかを実験により評価する．

第6章　評価

5章で紹介したツールを用いて，プレーンテキスト表示と本手法による表示の場合で，どのような差が出るかを実験し，評価する．

6.1　評価実験の内容

三重大学の情報工学を専攻する学部4年生と大学院生24人を対象とし，簡単なC言語プログラムを5章で紹介したツールで読み，実行結果を答えるまでの時間を計測する評価実験を1人ずつ順番に行った．このテストの実施日時は，被験者ごとに異なる．

まずテスト1では，全員にプレーンテキスト表示のプログラムを読んでもらい，その都度，平均正答時間がそろうようにAグループとBグループに振り分けてバランスを調整していく．ここでの平均正答時間とは，それぞれのテストにおける各グループの平均時間のことであり，被験者が正解を出すまでにかかった時間の平均である．このテストは全員が別の時間に受けているため，その時々の平均正答時間を見て，振り分けをしている．テスト1で使用したプログラムは，46行の簡単なプログラムである(sample1.c)(付録Aを参照)．Aグループはテスト2から，プレーンテキスト表示で問題に答え，Bグループはウィンドウ表示で答える．

テスト2では，関数が画面内に収まる53行のプログラムを使用した(sample2.c)．このプログラムにおいては，ウィンドウの並列配置はほとんどなく，表示法の差が出ないような工夫をしている．

テスト3では，条件分岐のブロックが画面外におよぶ161行のプログラムを使用した(sample3.c)．このプログラムは，条件分岐，for文のブロックを多く含み，ネストの深さは5が最大である．そのため制御構造が複雑であり，このプログラムにおいては，条件分岐の並列配置によって見やすさを改善できると考える．テスト3は，ブロックのウィンドウ化と条件分岐の並列配置による効果を見るものである．

図6.1はテストの内容をまとめたものである．また，このテストは，全員同じノートパソコンを使用し，紙とペンは自由に使っていいものとした．ソースコードの表示部分の大きさは1200 × 700ピクセルである．データは被験者がキーを押してテストを開始したときから，正解を入力したときまでの時間を秒単位で計った．このとき，時間の計測と答え合わせは，Windowsスクリプトによりコマンドライン上で行った．

	テスト1	テスト2		テスト2	
	全員 プレーンテキスト表示	Aグループ プレーンテキスト表示	Bグループ ウィンドウ表示	Aグループ プレーンテキスト表示	Bグループ ウィンドウ表示
プログラムの行数	46行　関数が画面内に収まる大きさ	53行　関数が画面内に収まる大きさ		161行　ブロックが画面外におよぶ大きさ	
テストの意図	このテスト結果を使って，平均正答時間がなるべくそろうようにグループAとBに分け	表示にほとんど差が出ない場合を想定してプログラムを作成．		条件分岐が見づらい場合を想定してプログラムを作成．	

図 6.1: テストの内容

具体的なテストの流れを以下に示す．

1. テストの概要説明
   ツールの表示法でプログラムを見て，実行結果を答えるまでの時間を測ると説明する．

2. ツールの操作説明
   マウスを用いたズーム，スクロールの説明をし，実際に動かしてもらう．

3. テスト開始から終了までの流れ説明
   コマンドラインに名前を入力した後に，何かキーを入力してテスト開始と説明する．
   答えに関しては数値を答えるよう説明する．

4. テスト開始

5. 答えを入力し，テスト終了

6. グループ分けをし，B グループならば 7 へ．A グループならば 8 へ移る

7. ウィンドウ表示について説明
   ブロックはウィンドウ化し，条件分岐は並列配置をしていることを伝える．また，条件分岐が長い場合は段を替えて接続している場合があることを説明する．

8. 次の問題に移りながら，4，5 を繰り返す

6.2 実験結果に影響を与える要因

何が見やすさに関係するかを調べるために，正答時間に影響を与える要因を絞って実験する必要がある．そのため，この実験では，全てのテストに同じ表示ツールを使うことで，文字の外観（大きさ，色，フォント）やウィンドウの制御機能（ズーム，スクロール）を統一した．画面の大きさに関しては，デフォルトの大きさは統一し，後から大きさを変えるのは自由とした．これだけの要因を排除したが，今回の実験においては本手法の要因を細かくは分けていない．本手法において見やすさを改善する要因として考えられるのは，図 5.7 で示したように，ブロックのウィンドウ化，条件分岐の並列配置，ウィンドウ上部への色づけ，ウィンドウ上部の文字拡大である．これだけの要因が混ざっているが，今回はそれぞれの機能についてどれだけ効果があるかは測定せず，本手法が総合的にどれくらい影響を与えるかを測定する．

また，テストに使うプログラムの内容についても，なるべく個人差が出ないように工夫をしている．そのため，プログラムに出てくる内容は，配列から最大値，最小値を求めるような問題や足し算，比較式のような簡単な計算問題と，if 文，else 文，for 文，return 文，引き数のある関数，標準出力関数である．詳しいテストプログラムの内容については，付録 A に記述する．

6.3　実験結果と考察

図 6.2 はテスト 1 の結果である．このテストで，なるべく全体の平均正答時間がそろうようにグループ分けをしたが，ミスなどによる個人の差が大きく出た．A グループの 4 人目のように大幅にタイムロストする人もいれば，ミスをしても早く終わる人もいた．また，間違えずに答えを出した学生においても差が見られた．これは経験による差と考える．同じような制御構造を普段から扱い，見ている学生は群化の仕方が経験的に分かるため，すぐに全体の構造を把握できる．それとは逆に，久しぶりに C 言語に触れる人は答えを出すのに時間がかかっているように感じた．

図 6.3 はテスト 2 の結果である．このテストでは表示法にほとんど差はなかったが，平均正答時間において約 60 秒の差が出る結果となった．その原因として次のようなことが考えられる．A グループに関しては，1 問目とほとんど内容が一緒だったためミスも少なく，全体的に早く回答を終えている．それに対して B グループは，初めてウィンドウ表示を見てもらい，すぐにテストを行ったため，経験の差が出てしまったと考える．また，勘違いによるミスも増え，このような結果となった．

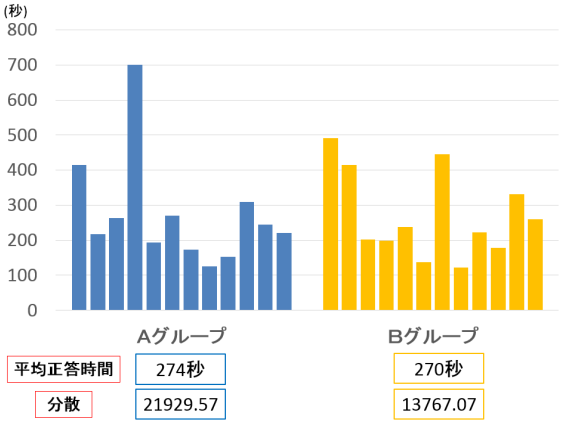


図 6.2: テスト 1　　sample1.c

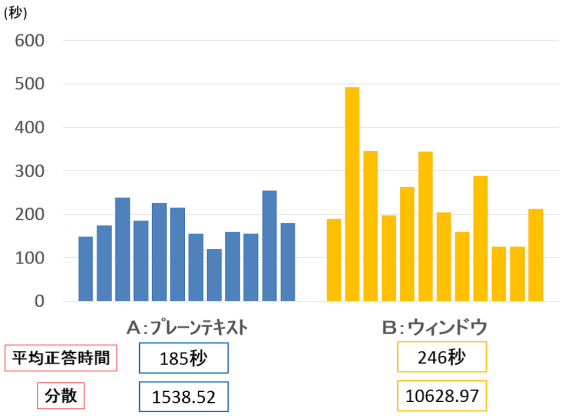


図 6.3: テスト 2　　sample2.c

図 6.4 はテスト 3 の結果である．このテストでは，テスト 2 に比べると倍以上の長さのプログラムであったにも関わらず，B グループの平均正答時間は早くなり，A グループと約 100 秒の差をつける結果となった．このような結果となった要因を考える．A グループに関してはプログラムが縦に長いこととデータの部分と答えに関連するプログラムの位置が遠かったことから，画面をスクロールする時間が多くなったと考える．画面をスクロールした際に，ブロックが多いことと似た構造をしていたことから読み間違いも多発したと考える．B グループに関しては，条件分岐の並列配置による効果が大きかったと考える．まず，画面のスクロールに関しては，A グループの場合よりも縦に短いため，スクロール時間は減る．そして，制御構造にまとまりがあることと，プログラムを 2 次元的に位置把握ができることから，答えに関連するプログラムの位置は覚えるのが簡単であったと考える．テスト 3 においては，ウィンドウ表示が優れていることが確認できた．

t 検定を用いて平均値に有意差があるかを調べた．図 6.5 はテスト 1〜3 における平均正答時間と t 検定の結果を示す．テスト 1 は F 検定により等分散であることを確認できたため，分散が等しいときの t 検定を行った．テスト 1 においては，p 値が 0.1 より大きいためこの平均値に有意差はない．この結果より，ほぼ同じ能力を持ったグループに分けられたと考える．テスト 2 は F 検定により分散が等しくないことを確認し，不等分散のときの t 検定を行った．テスト 1 と同じような結果が出ると考えていたが，結果として，0.1>p>0.05 より有意傾向であることが確認された．テスト 3 もテスト 2 と同様に不等分散であったため，不当分散のときの t 検定を行った．テスト 3 においては，0.05>p により平均値に有意差があることが確認できた．

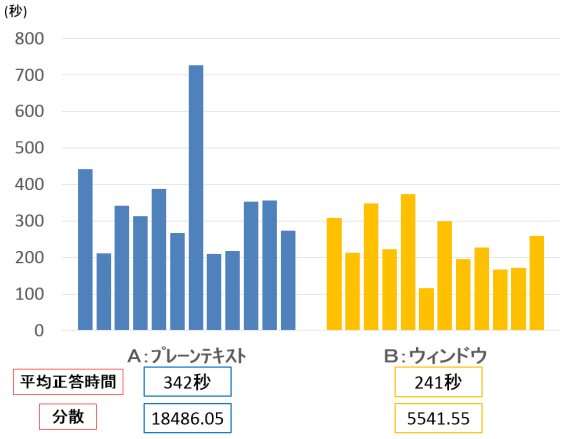


図 6.4: テスト 3　sample3.c

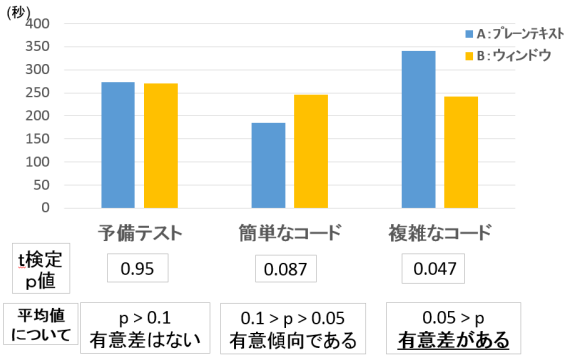


図 6.5: 正答時間の平均値の変化

第7章　結論

7.1　研究成果

本研究では，ゲシュタルト心理学の観点から，プログラム理解支援にはプログラム構成要素の群化支援が必要だと述べた．既存のソースコード表示法は，ブロックの範囲が不明確であり，縦に長い表示をしているため欠点がある．そこで，ブロックのウィンドウ化と条件分岐の並列配置により，ブロックの群化と制御構造の群化支援を提案した．これを評価するため，手法をツールとして実現し，学生にテストをしてもらった．テストの平均正答時間を出し，t 検定により平均値の有意差を確認した．その結果，本手法を使うことで，制御構造が複雑で長いプログラムにおいて可読性が向上することが確認できた．

7.2　今後の課題

- プリプロセッサの制御文
  一般的な C 言語プログラムにおいては，プリプロセッサの制御文が多く含まれるため，見にくさにつながる．本手法は if 文等のブロックをウィンドウするが，プリプロセッサの制御文にもブロックの概念があり，この制御文にも対応することでさらに見やすくできる．

- switch 文
  作成したツールは switch 文や break 文などへ対応していない．switch 文も並列配置できるが，break 文との兼ね合いもあり，表示方法の検討が必要である．

- データの群化支援
  プログラムにおいて，変数定義の部分が縦に長くなることが多いため，これを並列配置することで見やすさを改善できる．

- 多言語に対応
  C 言語だけでなく，jGRASP のように，Java，C++，Python，Objective-C，Ada，VHDL にも対応すべきである．

- 機能の充実
  折りたたみ機能を用いたプログラムの抽象化により，余計なブロックは小さくできる．また，文字に色をつけて，文の群化を支援できる．

- エディタとして完成
  本研究の最終目標はこのツールをエディタとして完成させることである．エディタにした際にも様々な問題があり，大きな課題である．

- 詳細な評価
  本研究では，様々な要因の混ざった実験を行った．そのため，どの要因が効果的であるのかを細かく分析する必要がある．また，制御構造が複雑なプログラムにおいては可読性が向上できたが，現実的なプログラムにおいて複雑なプログラムがどれくらい出てくるのかを調べる必要がある．

謝辞

本論文を作成するにあたり，日頃から研究・発表・講義で丁寧にご指導いただいた山田俊行講師，学会発表時の各種手続きなど様々な場でお世話になりました落合美子事務職員に心より感謝申し上げます．そして日頃研究・講義に関する議論をしていただいた研究室の学生諸氏にも感謝いたします．

参考文献

[小泉 11] 小泉俊幸，橋浦弘明，松浦佐江子，古宮誠一,「プログラミング学習支援環境 AZUR —ブロック構造と関数の可視化—」，電子情報通信学会技術研究報告. KBSE，知能ソフトウェア工学 110(386)，pp. 61–66，2011.

[US] Understand —ソースコード構造解析ツール— | テクマトリックス株式会社，https://www.techmatrix.co.jp/quality/understand/

[Cor89] T. A. Corbi，“Program understanding: challenge for the 1990’s”，IBM Systems Journal，28，pp. 294–306，1989.

[Wer23] M. Wertheimer，“Untersuchungen zur Lehre von der Gestalt. II”，Psychologische Forschung，4，pp. 301–350，1923.

[西堀 09] 西堀研人,「視聴覚事象の対応付けとそれに基づく概念の獲得」，名古屋大学博士学位論文，2009，http://hdl.handle.net/2237/12230

[VS] Visual Studio - ホーム，http://www.visualstudio.com/ja-jp/

[JGR] jGRASP Home Page，http://www.jgrasp.org/index.html

[CSD] jGRASP Control Structure Diagram，http://www.jgrasp.org/csd.html

[SK05] Dabo Sun and Kenny Wong，“On Evaluating the Layout of UML Class Diagrams for Program Comprehension”，Proceedings of the 13th International Workshop on Program Comprehension (IWPC ’05)，2005.

[FSP+11] Janet Feigenspan，Michael Schulze，Maria Papendieck，Christian Kstner，Raimund Dachselt，and Mathias Frisch，“Using Background Colors to Support Program Comprehension in Software Product Lines”，15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011)，pp. 66–75，2011.

[HCM02] Dean Hendrix，James H. Cross II，and Saeed Maghsoodloo，“The Effectiveness of Control Structure Diagrams in Source Code Comprehension Activities”，IEEE Transactions on Software Engineering，28，pp. 463–477，2002.

付 録A　実験に用いたプログラム

ここでは6章の評価に用いたプログラムについて紹介をする．また，テスト2，3についてはウィンドウ表示も紹介する．プログラムに対する設問は以下の通りである．

問題

以下のプログラムは2つの関数からなり，main関数内の標準出力関数により数値を1つ出力する．その数値を答えよ．

A.1　テスト1 sample1.c

```
#include<stdio.h>

int program1(int local_val){
        int arrange1[10] = {10,  3, 14, 49, 21, 39, 54, 35, 17, 41};
        int arrange2[10] = {92, 84, 79, 21, 56, 82,  8, 53, 64, 12};
        int arrange3[10] = {96, 47, 37, 72, 36, 49,  5, 24, 50, 17};
        int value1,value2,value3;
        int i;

        value1 = arrange1[0];
        for(i = 0; i < 10; i++){
                if(arrange1[i] > value1){
                        value1 = arrange1[i];
                }
        }

        value2 = local_val;
        if(value1 < 30){
                for(i = 0; i < 10; i++){
                        if(arrange1[i] > value2){
                                value2 = arrange1[i];
                        }
                }
        }else if(30 <= value1 && value1 < 60){
                for(i = 0; i < 10; i++){
                        if(arrange2[i] < value2){
                                value2 = arrange2[i];
                        }
                }
        }else if(60 <= value){
                for(i = 0; i < 10; i++){
                        if(arrange3[i] == value2){
                                value2 = arrange3[i];
                        }
                }
        }

        return value2;
}

int  main(){
        int a;
        a = program1(35);
        printf("%d\n",a);
        return 0;
}
```

A.2 テスト2 sample2.c

```
#include<stdio.h>

int program1(int local_val){
        int arrange1[10] = {21, 39, 53, 64, 12, 10,  3, 50, 17, 71};
        int arrange2[10] = { 8,  5, 24, 50, 17, 72, 36, 11, 66, 35};
        int arrange3[10] = {90, 47,  8, 84, 79, 21, 39, 53, 64, 12};

        int value1,value2,value3;
        int i;

        value1 = 0;
        for(i = 0; i < 10; i++){
                if(arrange1[i] > local_val){
                        value1++;
                }
        }

        value2 = 0;
        for(i = 0; i < 10; i++){
                if(arrange2[i] < local_val){
                        value2++;
                }
        }

        value3 = 0;
        for(i = 0; i < 10; i++){
                if(arrange3[i] > local_val - 10 && arrange3[i] < local_val + 10){
                        value3++;
                }
        }

        if(value1 > value2){
                if(value1 > value3){
                        return value1;
                }else {
                        return value3;
                }
        }else {
                if(value2 > value3){
                        return value2;
                }else {
                        return value3;
                }
        }
        return 0;
}

int  main(){
        int a;
        a = program1(24);
        printf("%d\n",a);
        return 0;
}
```

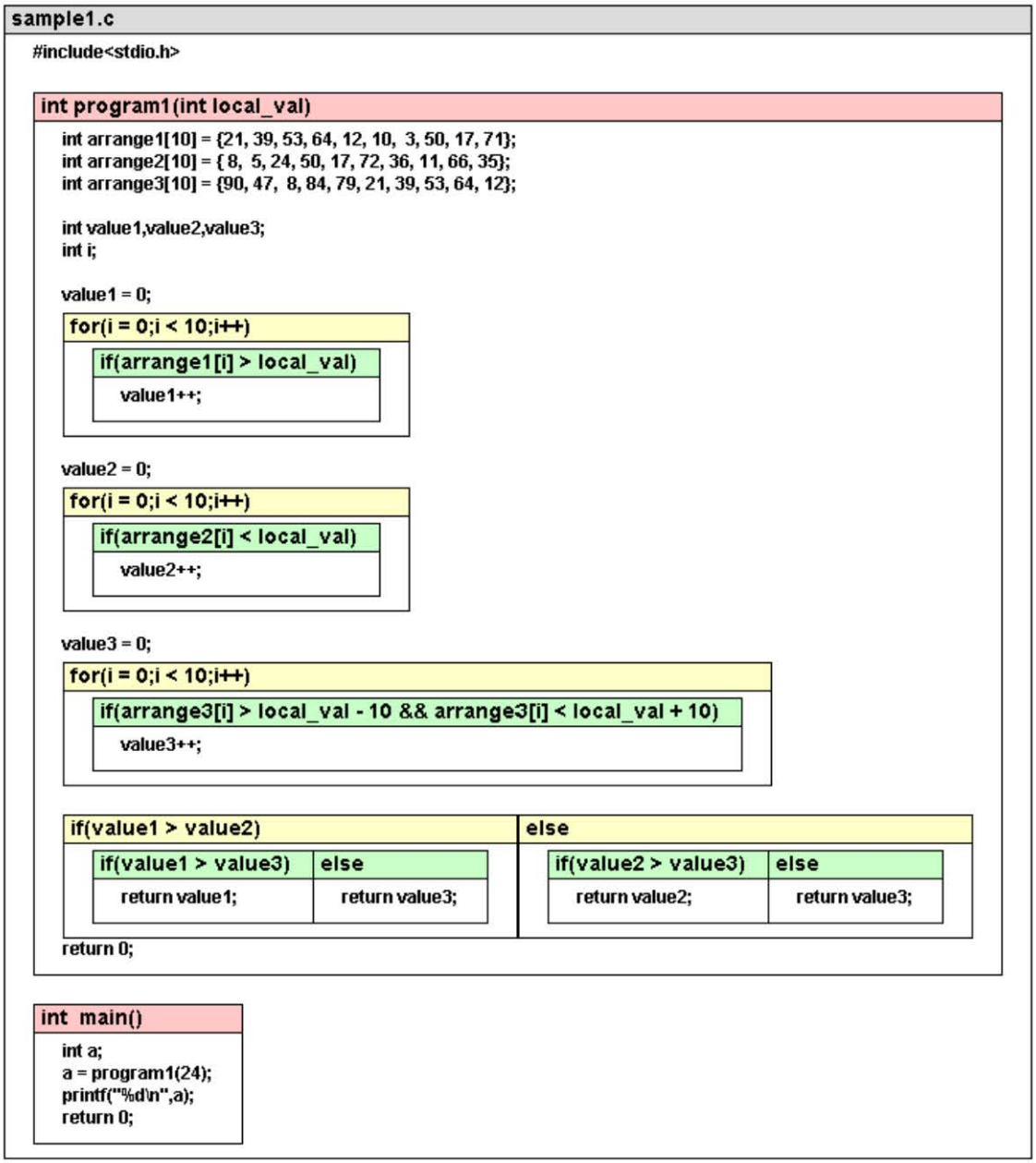


図 A.1: sample1.c のウィンドウ表示

A.3 テスト3 sample3.c

```
#include<stdio.h>

int program1(int local_val){
        int arrange1[10] = {10,  3, 50, 43, 24, 87, 94, 35, 17, 71};
        int arrange2[10] = {98,  5, 24, 50, 17, 72, 36, 49, 66, 85};
        int arrange3[10] = {90, 47,  8, 84, 79, 21, 39, 53, 64, 12};
        int value1,value2,value3;
        int i;

        if(local_val > 0){
                if(local_val > 50){
                        value1 = arrange1[0];
                        for(i = 0; i < 10; i++){
                                if(arrange1[i] > value1){
                                        value1 = arrange1[i];
                                }
                        }

                        value2 = arrange2[0];
                        for(i = 0; i < 10; i++){
                                if(arrange2[i] < value2){
                                        value2 = arrange2[i];
                                }
                        }

                        for(i = 0; i < 10; i++){
                                arrange3[i] = arrange3[i] + 10;
                        }
                        value3 = arrange3[5];

                        if(value1 > value2){
                                if(value1 > value3){
                                        return value1;
                                }else {
                                        return value3;
                                }
                        }else {
                                if(value2 > value3){
                                        return value2;
                                }else {
                                        return value3;
                                }
                        }
                }else {
                        for(i = 0; i < 10; i++){
                                if(arrange1[i] > local_val){
                                        value1 = arrange1[i];
                                }
                        }

                        for(i = 0; i < 10; i++){
                                if(arrange2[i] < local_val){
                                        value2 = arrange2[i];
                                }
                        }

                        for(i = 0; i < 10; i++){
                                if(arrange3[i] == local_val){
                                        value3 = local_val;
                                }else {
                                        value3 = 0;
                                }
                        }
                        value3 = arrange3[5];
```

```
                if(value1 < value2){
                        if(value1 < value3){
                                return value1;
                        }else {
                                return value3;
                        }
                }else {
                        if(value2 < value3){
                                return value2;
                        }else {
                                return value3;
                        }
                }
        }
}else  {
        if(local_val < -50){
                value1 = arrange1[0];
                for(i = 0; i < 10; i++){
                        if(arrange1[i] > 10 && arrange1[i] < 30){
                                value1 = arrange1[i];
                        }
                }

                value2 = arrange2[0];
                for(i = 0; i < 10; i++){
                        if(arrange2[i] > 30 && arrange2[i] < 50){
                                value2 = arrange2[i];
                        }
                }

                value3 = arrange3[0];
                for(i = 0; i < 10; i++){
                        if(arrange3[i] > 50 && arrange3[i] < 80){
                                value3 = arrange3[i];
                        }
                }

                if(value1 < value2){
                        if(value1 > value3){
                                return value1;
                        }else {
                                return value3;
                        }
                }else {
                        if(value2 < value3){
                                return value2;
                        }else {
                                return value3;
                        }
                }
         }else {
                value1 = arrange1[0];
                for(i = 0; i < 10; i++){
                        if(arrange1[i] < 10 || arrange1[i] > 30){
                                value1 = arrange1[i];
                        }
                }

                value2 = arrange2[0];
                for(i = 0; i < 10; i++){
                        if(arrange2[i] < 30 || arrange2[i] > 50){
                                value2 = arrange2[i];
                        }
                }

                value2 = arrange3[0];
```

```
                        for(i = 0; i < 10; i++){
                                if(arrange3[i] < 50 || arrange3[i] > 80){
                                        value3 = arrange3[i];
                                }
                        }

                        if(value1 > value2){
                                if(value1 < value3){
                                        return value1;
                                }else {
                                        return value3;
                                }
                        }else {
                                if(value2 > value3){
                                        return value2;
                                }else {
                                        return value3;
                                }
                        }
                }
        }
        return 0;
}

int  main(){
        int a;
        a = program1(-65);
        printf("%d\n",a);
        return 0;
}
```

図 A.2 に関しては倍率を下げて全体を表示している.

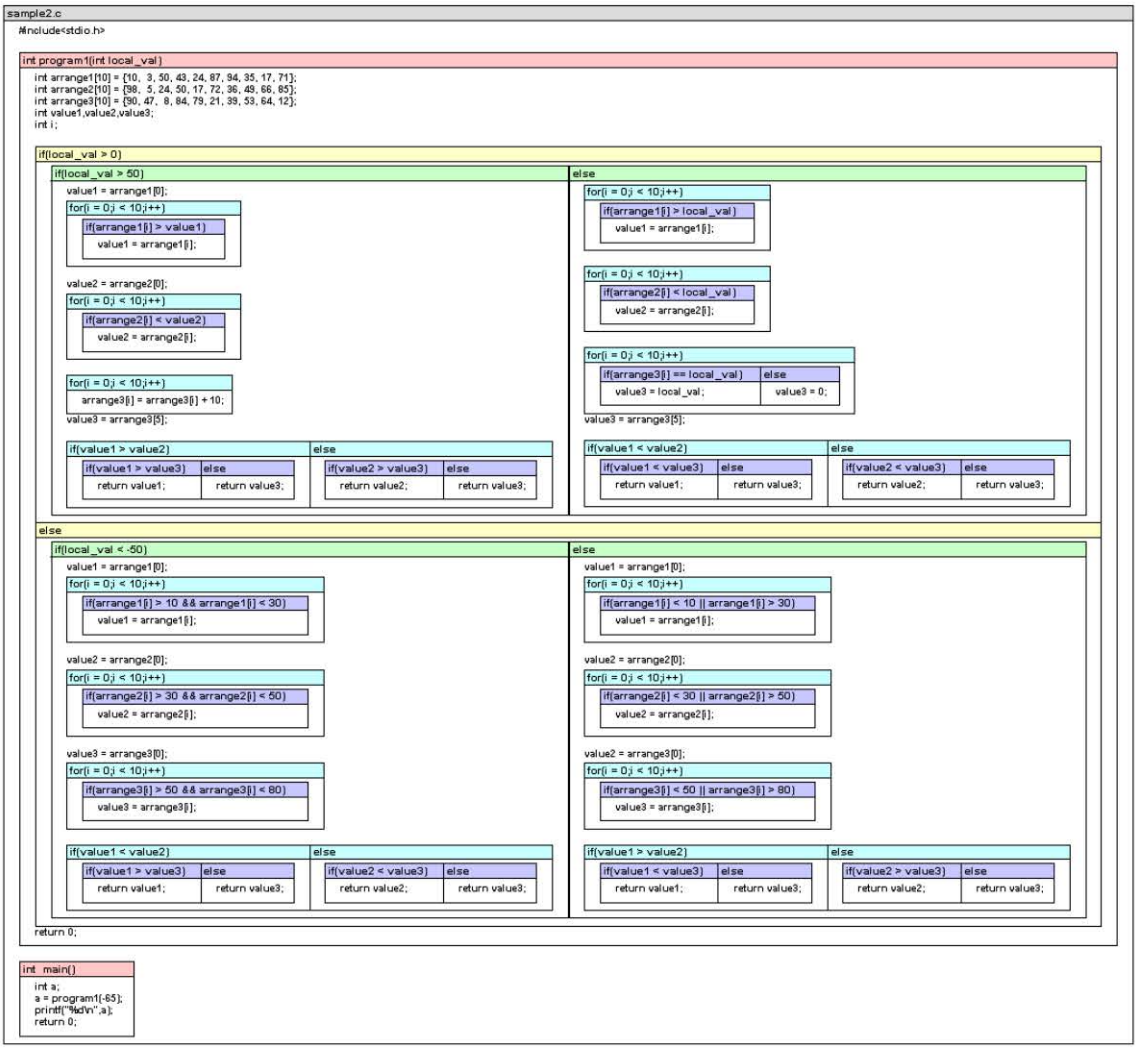


図 A.2: sample2.c のウィンドウ表示