プログラムの静的解析技法を用いたプログラム簡素化手法の検討

高谷暢之† 練 林† 井上克郎† 鳥居宏次†‡

† 大阪大学基礎工学部情報工学科
‡ 奈良先端科学技術大学院大学情報科学研究科

年々ソフトウェアは多機能化，大規模化している．一般に，利用者はソフトウェアのすべての機能を利用しておらず，一部の機能のみを利用している場合が多い．多機能なソフトウェアから，必要な機能だけを残し，それ以外の不要な機能を削除したソフトウェアが容易に構築できれば，サイズも小さくなり，その実行効率の向上が期待できる．またプログラム部品として別の用途に用いることもできる．本稿では，既存のプログラムに対して，入出力の値を制限することによってその機能を限定した小さな軽いプログラム(簡素化プログラム)を自動生成する方法を提案する．さらに，この方法を一般化するための枠組についても提案する．

キーワード：プログラム簡素化，部分評価，プログラムスライス

Program Simplification Method Based on Static Analysis of Program

Nobuyuki Takaya†, Lin Lian†, Katsuro Inoue† and Koji Torii†‡

† Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University
‡ Graduate School of Information Science, Nara Institute of Science and Technology

Software systems we are using are becoming larger and larger. However, all the facilities provided to the users would be hardly used, but some specific ones are repeatedly used. In this paper, we propose a program simplification method, which extracts the program statements related to the specific facilities and remove other unnecessary statements.

The extraction is based on the techniques to restrict input and/or output domains. This restriction leads to source program transformation, based on partial evaluation technique and static slicing technique.

The resulting program (simplified program) performs only the restricted facilities, and it would be smaller and more efficient than the original one. Also, the simplified program is used as a program component for other software systems. We propose an idea of a formal and uniform framework for this method.

Keywords : Program Simplification, Partial Evaluation, Program Slice

1 はじめに

年々ソフトウェアは多機能化し，ますます大規模になっている．一般に，利用者はソフトウェアに実装されているすべての機能を利用しているわけではなく，全体のごく一部の機能しか利用しない場合が多い．例えばemacsエディタは何百という機能を有するが，筆者らは通常いくつかの決まった機能しか使っていない．多機能なソフトウェアから，利用者が頻繁に用いる機能だけを残し，それ以外の利用しない機能を削除したソフトウェアが容易に構築できれば，ソフトウェアのサイズが小さくなり，その実行効率の向上が期待できる．更に小さくすることによりプログラム部品として，他の部品と組み合わせて別のシステムの構築に用いることができる．

また，UNIXシステムでは多くのシェルスクリプトが用いられているが，その中ではあるツールに特定の入力を必ず与えて実行させるような形で記述される場合がある．このような場合もその特定の入力に特殊化したツールが生成できれば効率の向上が期待される．

本稿では，既存のプログラムに対してその機能を制限することによって小さく軽いプログラム (ここでは簡素化プログラムと呼ぶ) を自動生成するための方法を提案する．この方法では，システムの機能の制限を入力または出力の値の範囲の制限によって定義する．

次章でプログラムの簡素化とは何かを説明し，3章でプログラムの簡素化に利用する技術と簡素化する際に適用する規則について，4章で適用例について，5章で提案する簡素化手法を統一的に組み合わせて用いる枠組について述べる．

2 プログラムの簡素化手法

プログラムは複数の入力から値を1つずつ取り出した入力組を受けとり，それに基づいて処理を行ない，その結果を複数の出力値を組にして出力する．プログラムの入力，出力の値が取り得る領域について考える．

あるプログラムにおいて，a) 特定の入力だけを処理するような場合，b) 入力の領域すべてではなく，その一部について値が必要な場合，c) 出力の一部だけが必要な場合などはその目的に合わせてプログラムを特殊化することができる．

入力もしくは出力を制限し，プログラムを特殊化することをプログラムの簡素化と呼ぶことにする．プログラムを簡素化すれば，プログラムのサイズが小さくなり，実行効率が上がる，プログラム部品として他の用途に再利用できるなどといった利点がある．

以下ではプログラムの簡素化には，入力を制限して簡素化する場合と出力を制限して簡素化する場合の2通りが考えられるのでそれぞれについて詳しく述べる．

2.1 入力制限からの簡素化

入力の制限には，次のようなものが考えられる．

- ある入力データの値の範囲を制限もしくは固定する
- 逆に特定の値を取らないようにする
- 文字列データであれば，特定の文字列で始まる
- 特定の文字列を含む

一般に入力を制限すればプログラム内の分岐文においてその分岐方向が一意に定まる場合や繰り返し文において繰り返しの回数が定まる場合が起こり得る．このような場合は，プログラム内の不要な文や無駄な条件判定などを削除でき，プログラムを簡素化できる．プログラムの不要な部分を削除すれば，更に文や変数が不要となることも考えられる．

プログラムの部分評価 (3.1節参照) は全体のうちの一部の入力が与えられたとき，それだけで可能な計算を行なう技術である．よって，プログラムに制限された入力を与えて部分評価を行なうことは，入力を制限してプログラムを簡素化したことに他ならない．

具体的には入力を制限してプログラムを簡素化するには，最初に入力を制限することがプログラムの変数にどのように影響するかを調べなければならない．そのためにはプログラムの制御フローとデータフローについて調べる必要がある．入力の制限により，取り得る値の範囲が変化したり，特定の値に固定されるような変数が出てくれば，当然その変数を参照する式の計算結果に影響を与える．評価結果が一意に定まるような式を見つければプログラムを簡素化できる．プログラム内の文や条件判定を削除すれば，プログラムの制御フローとデータフローに影響を与えるので新たに範囲が変化する変数や不要となる変数や文がないか調べなければならない．

入力の制限からプログラムを簡素化する手順は，

1. 制限された入力を定める
2. 範囲が変化したり，値が定まる変数を調べる
3. その変数を参照する文すべてを評価する
4. 書き換え規則を適用し，文を書き換える
5. 書き換えによって新たに範囲が変化する変数や不要となる変数や文がないか調べる．あれば3.に戻る

2.2 出力制限からの簡素化

出力の制限には，次のようなものが考えられる．

- 複数ある出力値の一部だけが欲しい
- 出力値をある範囲に押えたい

出力には必ず対応する出力文がある．ある出力が不要ならばその出力文は不要となるであろうし，その出力を押えたいのであればその出力文から出力に影響する文をたどっていきそれらの文に制限を加える必要があるだろう．

ここではある出力値を出力しないという制限について考える．出力しない値は当然計算する必要はなくなる．よってその値を計算するためだけに記述されている文は不要となるので削除でき，プログラムを簡素化できる．不要な文を削除すれば文や変数が更に不要となることも考えられる．

このような制限を行ないプログラムを簡素化するには，逆に必要な出力に対応する出力文を調べる．それらの出力文に影響を与える文，つまりそれらの出力値の計算に必要な文をすべて取り出せばプログラムを簡素化できる．

プログラムスライス (3.3節参照) はプログラム内のある文の実行に影響を与えるすべての文を抽出する技術である．よって，必要な出力文からプログラムスライスを求めることは，出力を制限してプログラムを簡素化したことと同じになる．

なお，この方法ではエラーが起こったときのエラー処理やファイルへのアクセスの後処理などの部分については考慮されていないので，それらを削除しないようにしなければならない．出力の制限からプログラムを簡素化する手順は，

1. 必要な出力をすべて定める
2. 必要な出力に対応する出力文をすべて調べる
3. 得られた出力文からプログラムスライスを求める
4. どの出力文のスライスにも含まれない文を調べる
5. 得られた文を削除する
6. 不要な文の削除によって書き換え規則が適用可能な文は書き換える

3 利用技術と簡素化規則

本章では，プログラムの簡素化で利用する静的解析技法と簡素化における書き換え規則，不要となる変数の判断基準について述べる．

3.1 部分評価

プログラムはデータを受けとり，それに基づいて処理を行ない，その結果を出力する．プログラムは入力データがすべて揃ってからその処理を行なうのが通常である (これを全計算と呼ぶ)．一部の入力データが与えられたとき，その入力データだけを用いて可能な計算を行なうことを部分評価 (部分計算) と呼ぶ [1, 3]．すなわち与えられた入力に基づいて実行できるものをできるだけ行ない，残りのデータが与えられないと実行できないものはそのまま残しておく．通常部分計算を行なって得られる結果のプログラムは，全計算のプログラムより能率の良い，特殊化したプログラムである．

部分計算して得られたプログラムに残りの入力データを与えて得られる計算結果は，元のプログラムを全計算して得られる計算結果と同じである．

例えば図 1(a) のプログラムに y=1 を与えた場合を考える．

1行目の if 文の条件式 (x>0) を評価しても値は定まらない．2行目の if 文の条件式 (y>0) を評価すると真 (true) となる．よって else 節 (3行目) は実行されることはなくなり，不要となる．x の代入文の右辺 (x+y) は y の値はわかっているので x+1 となる．5行目の代入文の右辺 (-x) は変わらない．したがって，部分評価プログラムは図 1(b) のようになる．

一般に部分評価では繰り返し文を何回展開するかなど評価の戦略を決める必要がある．ここでは次節で述べるように，式の計算，if 文などの簡単な書き換えのみを行なう．

3.2 書き換え規則

プログラムを部分評価すると分岐文，繰り返し文で実行される経路が一意に定まる場合が生じる．

(a) プログラム

```
1  if x > 0 then
2       if y > 0 then x = x + y;
3                else x = x - y;
4  else
5       x = -x;
```

(b) 部分評価プログラム (y=1)

```
1  if x > 0 then
2       x = x + 1;
3  else
4       x = -x;
```

図 1: プログラムの部分評価の例

このような場合実行されない経路は削除できるので，対応する分岐文，繰り返し文を書き換えることが可能となる．

また，書き換えを行なえば，新たに書き換えが可能となる場合も生じる．例えば図2のプログラムに y=1 を与えた場合を考える．

1行目の if 文の条件式 (x>0) を評価しても値は定まらない．2行目の if 文の条件式 (y=0) を評価すると偽 (false) となる．よって then 節が実行されることはない．また else 節もないので2行目の if 文は不要となり，削除できる．2行目の if 文を削除すると1行目の if 文の then 節は空文となる．よって新たにこの if 文も不要となり，削除できる．したがって，得られるプログラムは空となる．

図3に本手法で用いる書き換え規則を示す．

```
1  if x > 0 then
2       if y = 0 then x = x * x;
```

図 2: 新たな書き換えが生じる例

3.3 プログラムスライスとプログラム依存グラフ

プログラムスライシング (Program Slicing) は，直観的には，プログラム内の文の間の依存関係を調べ，プログラム内のある文の実行に影響を与えるすべての文を抽出する技術である [2, 4]．抽出された文の集合をスライス (Slice) と呼ぶ．スライスには，プログラムを静的に解析して得られる Static Slice と，動的に解析して得られる Dynamic Slice の2つ

1. **if** *true* **then** Statement → Statement
2. **if** *false* **then** Statement → ϕ
3. **if** *true* **then** Statement-A **else** Statement-B → Statement-A
4. **if** *false* **then** Statement-A **else** Statement-B → Statement-B
5. **while** *false* **do** Statement → ϕ
6. **if** *exp* **then** ϕ → *exp**
7. **if** *exp* **then** Statement **else** ϕ → **if** *exp* **then** Statement
8. **if** *exp* **then** ϕ **else** Statement → **if not** *exp* **then** Statement
9. **if** *exp* **then** ϕ **else** ϕ → *exp**

*exp の中で変数の定義が行なわれない場合は削除可能

図 3: 書き換え規則

がある．ここでは，プログラムの簡素化に利用する Static Slice に絞って説明する．

ある文に関する Static Slice とは，下記の2つの依存関係をたどってその文で使用しているいずれかの変数に到達するすべての文の集合である．

Data Dependence
文 $s1$ におけるある変数 vの定義が vを使用している文 $s2$ に到達するとき，文 $s1$ から文 $s2$ に Data Dependence 関係があるという．

Control Dependence
文 $s1$ は分岐文か繰り返し文であり，文 $s2$ の実行の有無が文 $s1$ の実行結果に直接依存するとき，文 $s1$ から文 $s2$ に Control Dependence 関係があるという．

上記の関係を用いてプログラムをグラフ化したものをプログラム依存グラフ (Program Dependence Graph) と呼ぶ．

ある文に関する Static Slice はその文の実行結果を保存する完全なプログラムを構成するという特徴を持つ．

3.4 不要な変数の検出

プログラムが書き換えられると元のプログラムにあった変数の定義や参照がなくなることが頻繁に起こる．その結果変数自身が不要となる場合がある．

変数を参照する文があっても次の例のような場合はその変数が不要である．

図4(a)のようなプログラムを考える．x=0を与えて部分評価したものが図4(b)である．変数yは自分自身の再定義にしか参照されておらず，無駄な計算をしているだけで変数yが不要なことがわかるであろう．

よって，変数が不要となるのは次の2つの場合である．

1. 参照する文がない
2. 自分自身の再定義にしか参照されない

これらは通常プログラムの依存グラフの到達性を調べることにより調べることができる．

(a) プログラム

```
1  i = 1; y = 1;
2  while i < N do
3      y = y * i;
4      i = i + 1;
5  end;
6  if x = 1 then print(y);
```

(b) 部分評価プログラム (x=0)

```
1  i = 1; y = 1;
2  while i < N do
3      y = y * i;
4      i = i + 1;
5  end;
```

図 4: 変数が不要となる例

4 適用例

例にはC言語で記述されたwcプログラムを用いる．wcは行数，語数と文字数を計算するプログラムであり，引数にはオプションとファイルをとる．オプションはl，w，cの3種類でそれぞれ行数，語数，文字数の計算を行なうかを指定する．オプションを省略した場合はすべてのオプションを指定したことになる．

付録にwcのソースコード(一部)を示す．大域変数doline，doword，docharはそれぞれ行数，語数，文字数の計算を行なう場合は1，行なわない場合は0となる．関数cntは実際にファイルの行数，語数，文字数の計算を行ない，出力する関数であり，メインの関数から呼び出される．

wcから行数と語数のみを計算するプログラムwc'を作成する場合を考える．入力制限から簡素化する場合と出力制限から簡素化する場合のそれぞれについて述べる．

4.1 入力制限からの簡素化の適用

入力を制限してwcからwc'を作成する過程を説明する．

- wc'では文字数の計算は行なわないのでwcでのオプションcは不要になる．
- オプション cが指定されないので大域変数docharの値は0に固定される．
  docharの値が制限されたのでこれを参照する文を評価する．
- docharを参照しているのは47行目のif文の条件部のみである．
  条件部を評価するとその評価値は0，すなわちfalseとなる．
- 書き換え規則2を適用して47行目のif文は削除される．
- 48行目のprint文が削除されるので変数charctの参照文が1つなくなる．
  charctの参照関係を調べると27行目の自分自身への代入文で参照されるのみであることがわかる．
  charctは変数が不要となるケース2に当てはまるので削除する．同時にcharctに対する代入文(13，27行目)は削除する．
- 27行めの代入文が削除されると変数lenの参照文が1つなくなる．lenの参照関係を調べると23行目のif文，28行目のfor文の条件部で参照されている．
  lenは削除することはできないので簡素化はこれで終了する．

削除されたのは，変数charct，13，27行目のcharctの代入文，47，48行目のif文である．

4.2 出力制限からの簡素化の適用

出力を制限してwcからwc'を作成する過程を説明する．

- wc'では行数と語数を計算し，出力するので行数と語数の出力文が必要となる．
- 行数と語数の出力文となっているのは，それぞれ44，46行目のprint文である．

- 44 行目の print 文からスライスを求めるとスライスに含まれるのは，6-11 行目の宣言文と 13-15，18-23，26，28-31，41-44 行目の実行文となる．
- 同様に 46 行目の print 文からもスライスを求める．
- 2 つのスライスのどちらにも含まれない文は，16，17，24，25，27，47-49 行目の文である．
- 16，17，24，25 行目はエラー処理を行なう部分であり，49 行目は後処理を行なう部分である．削除されるのは，27，47，48 行目の文である．
- これらの文の削除によって変数 dochar，charct，len の参照がなくなる．
  それぞれについて参照関係を調べると，dochar，charct は参照されず，len は 23 行目の if 文，28 行目の for 文の条件部で参照されている．
  charct は変数が不要となるケース 1 に当てはまるので削除する．同時に charct に対する代入文 (13 行目) は削除する．
  len は削除することはできないので簡素化はこれで終了する．

削除されたのは，変数 charct，13，27 行目の charct の代入文，47，48 行目の if 文である．

今回の例ではどちらの簡素化の場合も同じプログラムが得られた．しかし一般には入力を制限して簡素化した場合と出力を制限して簡素化した場合ではその結果は異なる．

5 簡素化手法の統一的な枠組

今まで見てきたいくつかのプログラムの簡素化手法を統一して取り扱えるような枠組について考える．

まず，操作の対象として，簡単なプログラム依存グラフを考える．ここでは，グラフ上の各頂点は，元のプログラムテキスト上の各代入文か if 文の条件式に対応している．この if 文の条件式に対応している頂点から制御依存の辺が出ており，その条件式の値に応じて，制御依存している頂点が実行されたり，されなかったりする．他の構造の文，例えば，while，for，switch 文などは if 構造で表現されるものと考える．関数呼びだし文は，代入文右辺や条件式中に現れうる．

ここで，図 5のような，プログラム依存グラフの中のデータ依存関係に着目する．この頂点 s に対応する文 (代入文，または条件式) は，各変数 $x_1,\ldots,x_n$ の定義域 $d^s_{x_1},\ldots,d^s_{x_n}$ を値域 $r^s_{x_1},\ldots,r^s_{x_n}$ に関数 f_s で変換する．ここで f_s は，直接 s の文の作用を行なう関数ではなく，各定義域 d を引数として，各値域 r へ変換する関数である (各型によって与えられる領域のパワーセットを引数や値とする関数)．f_s^{-1} は，f_s の逆関数である．一般に，最小の d，r を与える f_s，f_s^{-1} は容易に定義できるとは限らないが，特定の定数の代入文や変数値がいくつかの候補値しかない場合，逆に特定の値をとらない場合などは，それらの (制限的な) 情報を関数として与えることができよう．それらの情報がない場合は，型が許す全ての値を与えるようにする．

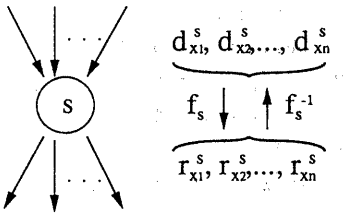

図 5: プログラム依存グラフ内のデータ依存関係

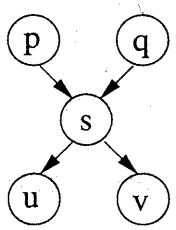

図 6: データ依存関係の例

このようにして，個々の文の d と r の関係が関数として与えられるとして，次にグラフ上でデータ依存関係を考える．今，簡単のため，図 6のようなグラフを考える (一般化は容易)．この場合，p，q で定義されている値のいずれかが s に到達する．したがって，p，q での値の領域の制限は s に伝えられるべきである．逆に s での値の制限は p,q にも影響すべきであろう．したがって，

$$d^s_{x_i} \leftarrow (r^p_{x_i} \cup r^q_{x_i}) \cap d^s_{x_i} \quad (1)$$

$$r^p_{x_i} \leftarrow r^p_{x_i} \cap d^s_{x_i} \quad (2)$$

$$r^q_{x_i} \leftarrow r^q_{x_i} \cap d^s_{x_i} \quad (3)$$

という操作を用いてお互いの制限を反映させることにより，

$$r^p_{x_i} \cup r^q_{x_i} = d^s_{x_i} \qquad (a)$$

とすることができる．

同様に，

$$r^s_{x_i} \leftarrow (d^u_{x_i} \cup d^v_{x_i}) \cap r^s_{x_i} \qquad (4)$$

$$d^u_{x_i} \leftarrow d^u_{x_i} \cap r^s_{x_i} \qquad (5)$$

$$d^v_{x_i} \leftarrow d^v_{x_i} \cap r^s_{x_i} \qquad (6)$$

により，

$$r^s_{x_i} = d^u_{x_i} \cup d^v_{x_i} \qquad (b)$$

とすることができる．

グラフの各頂点について d, rの値を求める．まず，それぞれの d, rの初期値は，式の値が定まる時はその値を要素とする部分領域，それ以外は，その式や変数の型が与える領域全体とする．そしてグラフの全ての頂点，全ての変数について (a), (b) が成り立つよう f_s, f_s^{-1}を使いながら (1) から (6) の操作を行ない，順次，d, rを小さくしていく[1]．

変化がなくなり，いずれの場所でも (a), (b) が成り立ったときのグラフを考える．sが代入文 $x_i := \ldots$ の時，$r^s_{x_i} = \phi$となる場合は，その頂点は不要なので削除できる (副作用があるときはその副作用を与える変数についてもϕとなる必要があろう)．if文の条件文が真または偽になるような各変数の dが与えられる時は，制御依存の変をたどり，不要な頂点を削除できる．

通常，無駄なく記述されたプログラムそのものでは，この種の削除は期待できにくい (一部読みやすさのために，定数伝搬が起こるような記述がなされて削除できる例もあろうが)．

ここで入力制限について考える．これは入力文が読み込む値の領域を制限するのに相当する．入力変数の一つを固定する，すなわち部分評価することは，その変数の領域を，型が与える領域全体からその一つの値にすることで表現できる．入力変数一つを削除することはその変数の領域をϕにすることで表せる．

次に出力制限について考える．必要でない出力変数の領域をϕにすることで，必要な出力変数が依存する文のみ，すなわちスライスを残すことができる．

[1]これは，各頂点に与えられた領域の制限は全て守られるという仮定に基づいている．特定の頂点のある変数の制約のみを守り，他はそれに応じた領域をとる場合などは異なる解の収束のさせた方が必要であろう．

このように，部分評価，スライスなどのプログラムの静的な操作をこの枠組では統一的に扱うことができる．ここでは，入出力のみに制限を与えて，他への波及を調べる方法のみを述べたが，プログラムの途中の特定の文に着目して，そこの領域を制限することによるプログラムの簡素化も考えられよう．

ここでは，各頂点で始めに与えれる制限は全て満たす，という前提で，全体の整合性をとっている．したがって得られたプログラムに，その制限からはずれた入力を与えた場合は，プログラムの動作は保証されない．

6 まとめ

プログラムの簡素化手法について述べた．ここでは，入出力の領域の制限からプログラム変換を行なうことに着目した．入力，出力領域を狭めるのではなく，広げたり，移動したりすることによるプログラム変換もこの手法の延長と考えられる．ただし各式が新たな領域の値を受けとって演算ができるような工夫が必要であろう．

現在，2章から4章で述べた方法についてシステム化を検討している．また5章で述べた枠組の形式的な記述，及び，既存のプログラムの意味定義方法との関係，また，プログラムの抽象解釈などとの関連などは今後の課題である．

参考文献

[1] 二村良彦："部分計算"，淵一博，古川康一，溝口文雄，"プログラム変換"，共立出版，pp.63–79 (Aug. 1987).

[2] S. Horowitz and T. Reps : "The Use of Program Dependence Graphs in Software Engineering", Proc. 14th International Conference on Software Engineering, Melbourne, Australia, pp.392–411 (May 1992).

[3] Uwe Meyer : "Techniques for Partial Evaluation of Imperative Languages", Proc. Symposium on Partial Evaluation and Semantics-Based Program Manipulation, pp.94–105 (Jun. 1991).

[4] 下村隆夫："Program Slicing 技術とテスト，デバッグ，保守への応用"，情報処理，Vol.33, No.9, pp.1078–1086 (Sep. 1992).

付録：wc のソースコード (一部)

```
 1  static int      doline, doword, dochar;
 2
 3  cnt(file)
 4          char *file;
 5  {
 6          register u_char *C;
 7          register short gotsp;
 8          register int len;
 9          register long linect, wordct, charct;
10          int fd;
11          u_char buf[MAXBSIZE];
12
13          linect = wordct = charct = 0;
14          if (file)
15                  if ((fd = open(file, O_RDONLY, 0)) < 0) {
16                          perror(file);
17                          exit(1);
18                  }
19          else
20                  fd = 0;
21
22          for (gotsp = 1; len = read(fd, buf, MAXBSIZE);) {
23                  if (len == -1) {
24                          perror(file);
25                          exit(1);
26                  }
27                  charct += len;
28                  for (C = buf; len--; ++C)
29                          switch(*C) {
30                                  case NL:
31                                          ++linect;
32                                  case TAB:
33                                  case SPACE:
34                                          gotsp = 1;
35                                          continue;
36                                  default:
37                                          if (gotsp) {
38                                                  gotsp = 0;
39                                                  ++wordct;
40                                          }
41                          }
42          }
43          if (doline)
44                  printf(" %7ld", linect);
45          if (doword)
46                  printf(" %7ld", wordct);
47          if (dochar)
48                  printf(" %7ld", charct);
49          close(fd);
50  }
```