木構造化チャートによるプログラム開発・保守技法†

大原 茂之††

これまでのプログラム文書化技法の多くは，どちらかといえばプログラム開発に焦点を合わせてきた．これに比べプログラムの保守性をいかに高めるかという問題についての議論は十分になされていなかったように思われる．プログラム開発はトップダウン的な側面が強く，プログラム保守は作成済みのプログラムから仕様に近付くボトムアップ的な側面が強い．本論文ではプログラムの開発と保守をトップダウンとボトムアップ併用によるプログラミングとして位置づけ，これを統一的に扱える記述法について述べる．この記述法（TS チャートと呼ぶ）では，プログラムの保守を重視した寄生構造という概念を導入した．寄生構造を利用することでプログラムを制御部，被制御部，メイン処理，準備的な処理，後始末的な処理といったレベルまで構造化できること，さらに作成済みプログラムの処理，モジュール，制御構造等の修正を容易にできることを示す．

1. まえがき

構造化プログラミング技法が提案されて以来，各種高級言語やプログラム開発技法の中に構造化の考え方が取り入れられてきた．しかし，二村や河合らが言うように，通常のフローチャートは高級言語の制御構造を直接表すことができず，したがって構造化技法に馴染まないという問題が表面化してきた[1]．そこでロジックの流れを表形式で記述する NS チャート，木図面で記述する PAD，木構造化フローチャート，HCP 等構造化プログラミングの思想を図的表現の中に取り込む試みがなされてきた[1)~5)]．これらの技法はプログラムの制御構造，階層構造を明確にするという点で成果を上げている．

しかし，一連の構造化技法を学生に教えてきた経験からするとまだいくつかの問題点が残っている．

第一はプログラム保守上の基本的な立場の一つである分かりやすさである．ほとんどの学生は自分で作成したプログラムであっても，そのプログラムのポイントを他の学生に説明することが苦手である．逆に，他の学生が作成したプログラムを理解するとなると実行手順を追いかけるのみで，プログラム全体を把握するまでに時間がかかる．経験的にではあるが少なくともこれらの原因は，

（1） メイン処理と非メイン処理の区別．

（2） ループ等に対する初期設定と終結処理の明示．

（3） ループ等の制御部と非制御部の区別．

（4） 本来の処理順序が全順序であるか半順序であるかといった順序関係の明示．

（5） データの受渡しと手続きの流れの関係の明示．

等を構造化する手段が不十分であったと考えられる．ここで言う構造化とは処理内容を見なくても，チャートの接続関係から各処理の役割を把握できるようにすることである．これまでの構造化プログラミングもしくはプログラム図等で上記（1），（3）を解決することは困難であることが指摘されている[6]．

第二はプログラムを作成する場合の立場である．与えられた問題に対し，彼らは何度も同じ箇所の修正をしたり，最初から作成し直す等の行動を見せる．特にトップダウンロジックのみによる作成となると完成までに相当時間がかかる．作成中ではあっても既に記述したプログラム部分の修正，改良等を行うということは，プログラム開発の中に保守の考え方が要求されていると見てよいであろう．プログラム保守から見たプログラミングは，手を加えたいプログラム部分と他の部分とのインタフェース等を考えながらそのプログラムの仕様に近付くボトムアップ的な側面が強い．しかし，これまでのプログラミング技法はトップダウンに記述する開発技法が中心であり，ボトムアップに記述する技法についての議論はあまりなされていなかった[7)~9)]．プログラミングを行う場合両者を併用すると効果的であることが知られている[10]．

本論文では，プログラムを開発と保守によって得られる文書として捉え，これを統一的に扱えるようにした木構造化チャート（Tree Structured Chart, TS チャート）なる文書化技法について述べる．TS チャートでは上述した問題点を，モジュール構造，寄生構

† Program Development and Maintenance Techniques Based on Tree Structured Chart by SHIGEYUKI OHARA (Department of Electronics, Faculty of Engineering, Tokai University).

†† 東海大学工学部電子工学科

造，制御構造という概念によって解決した．以下，TS チャートで表現可能な各種プログラム構造，ボトムアップを中心としたプログラミング技法について述べる．

2. TS チャートの定義

ここでは TS チャートで用いる記号と TS チャートの基本構造としてのモジュールについて述べる．

2.1 記号セット

TS チャートは**図 1** に示す記号セットを用いて構成する．同図において，(1)を開始記号，(2)を終端記号，(3)を定義開始記号，(4)を定義終端記号，(5)を一般処理記号，(6)を選択記号，(7)を排他的処理記号，(8)を条件ループ記号，(9)を無限ループ記号，(10)を離脱記号，(11)を帰還記号，(12)をマクロ記号，(13)を変数記号と言う．これら TS チャートの記号と重複しない通常のフローチャートの記号はそのまま用いる．

2.2 モジュールと実行順序

開始記号で始まり終端記号で終わる構造をモジュールと呼ぶ．モジュールの例を**図 2** に示す．

TS チャート上の任意の二点 u, v に対し $\pi(u, v)$ なる表記をもって u から v への実行順序と呼ぶ．実行順序は正則表現で定義する[11),12)]．例えば，図 2 の s から e への $\pi(s, e)$ は A と定義する．

3. TS チャートの構造

3.1 順序構造

3.1.1 連接構造

図 3 (1)のように処理に処理を接続した構造もしくは同図(2)のようにモジュールにモジュールを接続した構造も TS チャートである．これらを連接構造と言う．同図(1)，(2)の $\pi(u, v)$ は共に AB である．

3.1.2 差し込み構造

図 4 (1)のように処理とモジュールを接続した構造

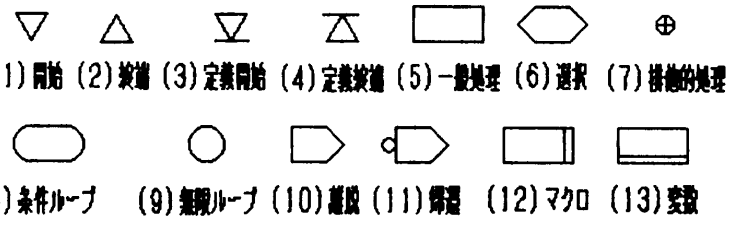

図 1 TS チャートの記号セット
Fig. 1 Set of TS chart symbols.

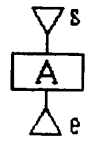

図 2 モジュールの基本構造
Fig. 2 Basic structure of module.

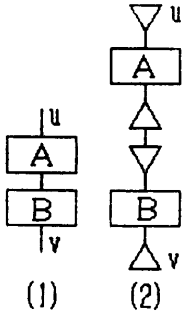

図 3 TS チャートの連接
Fig. 3 Concatenation of TS charts.

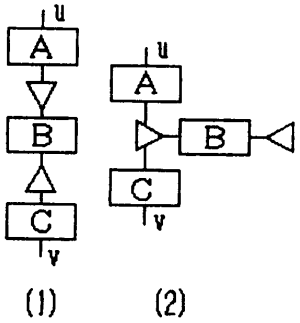

図 4 差し込み構造
Fig. 4 Insert structure.

も TS チャートである．この構造は同図(2)のようにも表す．このように接続したモジュールを差し込みモジュールと言う．$\pi(u, v)$ は ABC である．差し込みモジュールによって非メイン処理とか後に述べる被制御部を表すことができる．

3.1.3 選択構造

選択構造は選択記号を用いて構成されるスイッチ型選択構造と，選択記号と排他的処理記号を用いて構成する排他的選択構造とがある．

(1) スイッチ型選択構造：**図 5** (1)の構造であり P と P_1〜P_n によって一連の処理 A_1〜A_n の実行開始箇所を指定する．例えば，条件 P が P_i として成立した場合は A_i〜A_n を一連の処理として実行に移す．

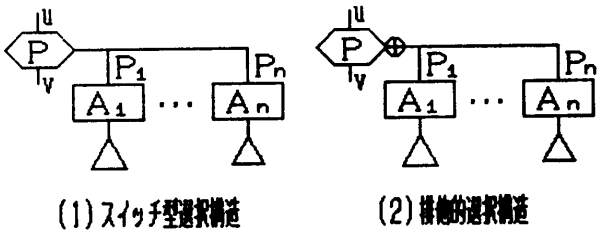

図 5 選択構造
Fig. 5 Structure of selection.

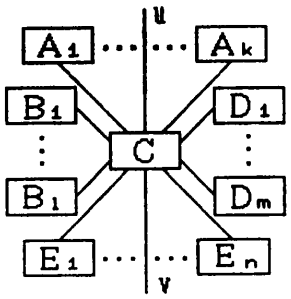

図 6 寄生構造
Fig. 6 Parasitic structure.

$\pi(u, v)$ は $(P_1A_1\cdots A_n)+\cdots+(\bar{P}_1\cdots\bar{P}_{i-1}P_iA_i\cdots A_n)+\cdots+(\bar{P}_1\cdots\bar{P}_{n-1}P_nA_n)$ である．スイッチ型選択構造をC言語で解釈するとPは switch 文となり P_1〜P_n は各々 case 文となる．

（2） 排他的選択構造：図5（2）の構造であり，複数個の処理 A_1〜A_n の中から一つだけ選択し実行に移す．例えば条件Pが P_i として成立したとすると，A_i を実行に移す．したがって $\pi(u, v)$ は $(P_1A_1)+\cdots+(\bar{P}_1\cdots\bar{P}_{i-1}P_iA_i)+\cdots+(\bar{P}_1\cdots\bar{P}_{n-1}P_nA_n)$ である．

特に n を2とし P_1, P_2 をそれぞれ yes, no とした場合を分岐構造と言う．分岐構造であることが明らかな場合は排他的処理記号の記入を省略する．

3.1.4 寄生構造

図6の構造を寄生構造と呼び A_1〜A_k, B_1〜B_l, D_1〜D_m, E_1〜E_n はCに寄生していると言う．このときuからは，先ず処理Cの上に寄生しているグループ A_1〜A_k を任意の順序で実行し，第2番目にCの左側に寄生しているグループ B_1〜B_l を任意の順序で実行し，第3番目にC自身を実行し，第4番目にCの右側に寄生しているグループ D_1〜D_m を任意の順序で実行し，第5番目にCの下側に寄生しているグループ E_1〜E_n を任意の順序で実行した後vへ至る．

実行順序が任意ということを数学的に言えばグループ内での順序関係が半順序になっているということであり，システム的に言えば各グループごとに並列化可能であることを意味する．また通常のプログラミング言語でコーディングする場合は各グループ内でのコーディング順序は任意に定めてよいことを意味する．

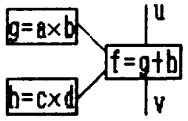

図 7 寄生構造の例
Fig. 7 Example of the parasitic structure.

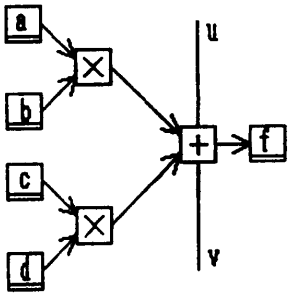

図 8 TS チャートにおけるデータフローの例
Fig. 8 Example of the data-flow in TS chart.

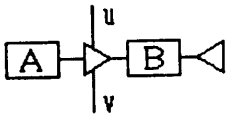

図 9 制御構造
Fig. 9 Control structure.

具体的な寄生構造の例を $f=a\times b+c\times d$ に基づいて説明する．この式の $a\times b$ と $c\times d$ はどちらを先に計算してもよいし同時に計算してもよい．いわば両者の関係は半順序になっており，図7のように寄生構造で表すことができる．保守を考えるならば，原システムの順序関係はプログラム構造の中においても保存されるべきであろう．

3.1.5 データフロー構造

既に示した寄生構造を利用してデータフローチャートとTSチャートを結合することができる．例えば図8は図7の順序関係を変数記号とデータフローで書き換えた構造である．この記法を用いることで手続きを中心とした構造の中でも，変数間のデータの受渡しやデータの入出力を構造的に明確化できる．

3.2 制御構造

3.2.1 制御部と被制御部

図9は差し込みモジュールBに処理Aを寄生させた構造であり，$\pi(u, v)$ は AB となる．ここでAを制御

部，Bを被制御部という．

3.2.2 ループ構造

ループ構造には条件ループ構造と無限ループ構造の二種類がある．いずれもモジュールの開始記号にループ記号を寄生させることによってそのモジュールを制御するという考え方を取っている．

（1） 条件ループ構造：**図 10**(1)に示すように差し込みモジュールに条件ループ記号を寄生させた構造を条件ループ構造と言い，条件ループ記号内のAをループ条件と言う．条件ループ記号はループ条件に基づいてモジュールの実行を繰り返させる．したがって，$\pi(u, v)$ は $(AB)^*\bar{A}$ と定義する．ループ条件としては例えば For, While といったステートメントが考えられる．

（2） 無限ループ構造：図 10(2)のように差し込みモジュールに無限ループ記号を寄生させた構造を無限ループ構造と言う．無限ループ記号はモジュールの実行を無限に繰り返させる．したがって $\pi(u, v)$ は B^ω と定義する．

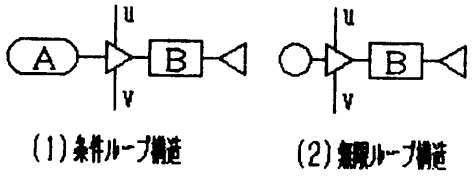

図 10 ループ構造
Fig. 10 Loop structures.

3.2.3 離脱構造

離脱記号もしくは帰還記号を含んだモジュールを離脱構造と言う．

（1） 離脱記号：離脱記号によって現モジュールの実行を打ち切り，次のモジュールもしくは処理等の実行に移行する．**図 11** は離脱記号を含んだ条件ループ構造の例である．離脱記号内のCを離脱条件と言う．処理Eは離脱条件Cが成立しvへ移る時点で実行される．$\pi(u, v)$ は $(AB\bar{C}D)^*(\bar{A}+ABCE)$ となる．

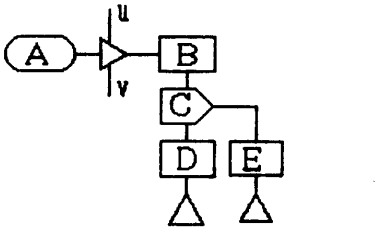

図 11 離脱記号付きループ
Fig. 11 Loop with the escape symbol.

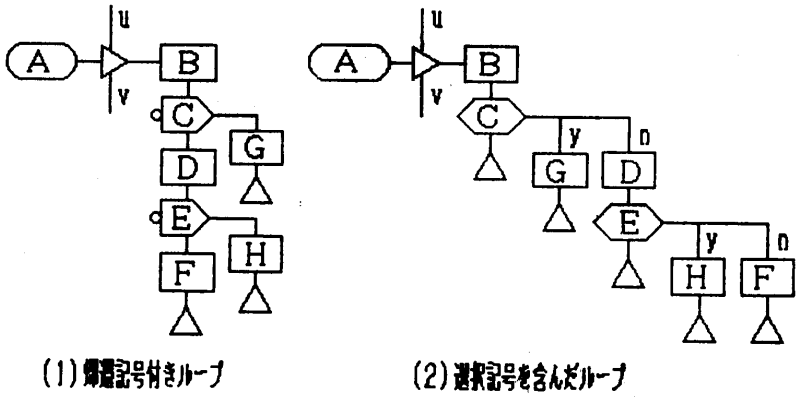

図 12 複数の帰還路を持ったループ
Fig. 12 Loop with feedback paths.

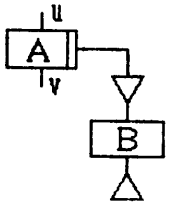

図 13 マクロ構造
Fig. 13 Macro structure.

（2） 帰還記号：帰還記号によって処理の途中からモジュールの制御部へ実行を移行する．**図 12**(1)は帰還記号を含んだループ構造の例であり，帰還記号内のC, Eといった条件を帰還条件という．帰還条件Cが成立した場合はGを実行して制御部へ戻り，帰還条件Eが成立した場合はHを実行して制御部へ戻る．したがって，$\pi(u, v)$ は $(ABCG+AB\bar{C}DEH+AB\bar{C}D\bar{E}F)^*\bar{A}$ となり，三通りの路から成るループであることがわかる．これに対し図 12 (2)は選択記号を用いて同図(1)と同じ働きをするようにしたTSチャートである．同図(1)に比べ同図(2)の方がこのループ構造を把握しがたいことは直感的に理解されよう．このように選択記号による場合分けが深くかつループが重なる構造では帰還記号を利用してループ構造を簡潔にできる．C言語で言えば帰還記号は continue 文として解釈できる．

3.3 マクロ構造

TS チャートの一部またはモジュールをより大きな

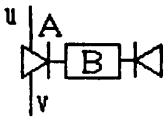

図 14 定義構造
Fig. 14 Definition structure.

意味で表す場合には図 13 の構造を用いる．この構造をマクロ構造と言い，Aをマクロ表現と言う．Aは非実行内容であるがBはAの具体的な実行内容である．したがって，実行順序を表す $\pi(u, v)$ はBとなる．

3.4 定義構造

関数，手続き，データ構造等を定義する場合には図 14 の構造を用いる．Aは定義構造の名称であり，Bは定義の内容である．定義そのものは実行されるものではなく，したがって $\pi(u, v)$ は空である．手続き等の定義構造を実行させるには，JIS の定義済み処理記号を用いる．

4. プログラムの寄生構造化

4.1 前処理と後処理の構造化

プログラムの流れの中には本来行いたい処理，その処理のための準備的な処理と後始末的な処理とがある．ここでは寄生構造によってこれらの処理を区別できることを示す．

4.1.1 処理に対する寄生

図 15 (1)は一般的な処理に対する寄生構造の例であり，同図(2)はモジュールに対する寄生構造の例である．両者共に $\pi(u, v)$ は ABC である．この実行順序は A, B, C が直列に接続されている場合と同じである．しかし，プログラムの流れから見てBはメイン処理であるが，AとCは非メイン処理でそれぞれBの前処理，後処理であると解釈できる場合には，直列表現よりこの構造の方が優れていると言ってよいであろう．このように構造化されたプログラムであれば，メイン処理だけを選択的に追跡していくことでプログラムの概要を素早く掌握することが可能となる．

4.1.2 ループ構造に対する寄生

ここでは，ループ構造に対する初期化と後始末の箇所を構造的に明示できることを示す．

図 16 (1), (2)の TS チャートの実行順序は，共に A(BC)*B̄D である．ただし同図(2)の TS チャートは同図(1)の TS チャートの処理 A, D をそれぞれループに対する初期化，後始末と考えた場合であり，ループモジュールに寄生させた構造になっている．この構造であればループに対する初期化(A)，後始末(D)，ループ制御部(B)，被制御部(C)を明確に区別できる．

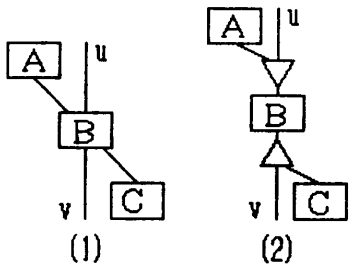

図 15 処理の構造化
Fig. 15 Structured of processings.

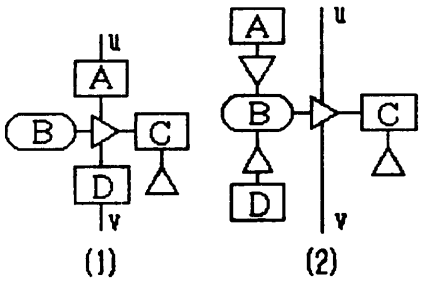

図 16 ループに対する寄生構造
Fig. 16 Parasitic structure on the loop structure.

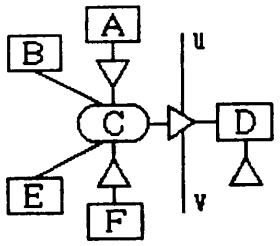

図 17 ループ記号に対する寄生構造
Fig. 17 Parasitic structure on the loop symbol.

4.2 制御部の構造化

図 17 の TS チャートはループ条件に処理を寄生させた構造であり $\pi(u, v)$ は A(BCDE)*BC̄F となる．この構造を利用することで被制御部内の制御的な要素を制御側へ抜き出せる場合がある．

例えば，図 18 (1)は 1 から n までの総和を求める TS チャートの例である．この例では無限ループと離脱記号を用いている．しかし，このループを制御する変数 i に関する増分処理と離脱条件という制御部分が被制御側に含まれている．この制御部分を被制御部から分離させてループ制御側へ寄生させると同図(2)が得られる．このようにするとモジュールを制御する制御側と，被制御モジュールとを明確に区別できる．

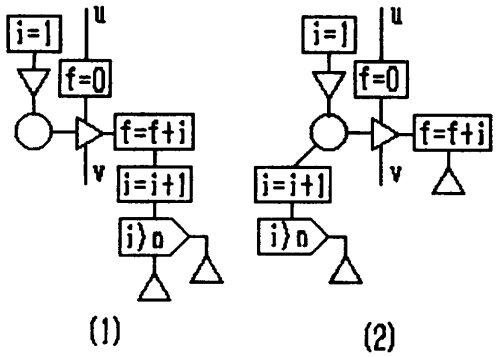

図 18 被制御部からの制御部分の分離
Fig. 18 Separate the part of control from the controlled modules.

5. TS チャートによるプログラミング

5.1 トップダウン記述技法

トップダウンに記述するには既に述べたマクロ構造を用いることで可能となる．プログラムの詳細まで立ち入りたくない場合は，とりあえずマクロ表現で記述し，後からその実行内容を寄生させていけばよい．

5.2 ボトムアップ記述技法

5.2.1 構造の追加

プログラムの開発，保守を行う上で有効となる一連のボトムアップ記述法を図 19 に示す．

（1） 処理の追加：二つの処理の間に他の処理を入れ忘れていた場合，寄生構造を利用して継ぎ足すことができる．図はAとCの間にBを寄生させた例である．

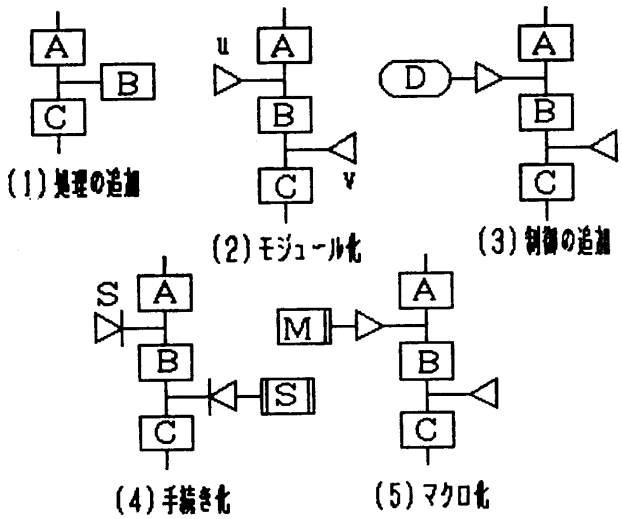

図 19 ボトムアップによる構造化
Fig. 19 Structured by the bottom-up.

（2） モジュール化：プログラム部分をモジュールとして定義したい場合は，開始記号と終端記号を寄生させればよい．図はBのモジュール化の例である．ここではBという処理が一つであるが複数個あっても同じくモジュール化できる．

（3） 制御の追加：プログラム部分を制御構造化する場合はその部分をモジュール化し，このモジュールに制御部を寄生させればよい．図はプログラム部分Bを条件Dの間繰り返す制御構造に変更した例である．

（4） 手続き化：プログラム部分を手続き，あるいは関数として定義したい場合は，定義開始記号と定義終端記号を寄生させればよい．図はBを手続きSとして定義した例である．ここで注意することは手続き化によってプログラム部分Bが定義構造となり実行されなくなることである．したがって，手続きSを実行させる記号を寄生させておく必要がある．

（5） マクロ化：プログラムの保守を考えるならば，プログラムを意味のあるいくつかの部分に区切り，各部分をマクロ構造にしておくべきである．しかし，一般的にプログラムを作成する場合，必ずしも最初からすべての部分をマクロ構造として捉えられるものではない．したがって，作成済みのプログラムの一部を後からマクロ表現の詳細部分として定義しておきたい場合が生じる．このような場合は図のようにモジュール化とモジュールへのマクロ表現の寄生を行うことでボトムアップでのマクロ化が可能となる．

5.2.2 ボトムアッププログラミング

プログラムはトップダウンに開発されていくべきものであるが，部分的あるいは補助的な考え方として見るならばボトムアップも有効となる[10]．ここではこれまで述べてきた記述法のボトムアップ的な使用方法を簡単な例によって示す．

例として終端マークが END になっている配列 $f(c)$ の平均値を av に求めるプログラムを作成する．ただし，c は1以上とする．このプログラムをボトムアップに作成していく過程を図 20 に示す．

（1） 平均値を求めるために，総和と全体の個数を求めることを考える．まず総和 (s) を求めるための基本モジュールを定義する．

（2） $f(c)$ が END マークでなければ基本モジュールの実行を繰り返させる制御条件を与える．なお，図中 != は等しくないという条件を意味する．

（3） このループは変数 c を制御することで動作する．ここでは，変数 c がループの制御変数である

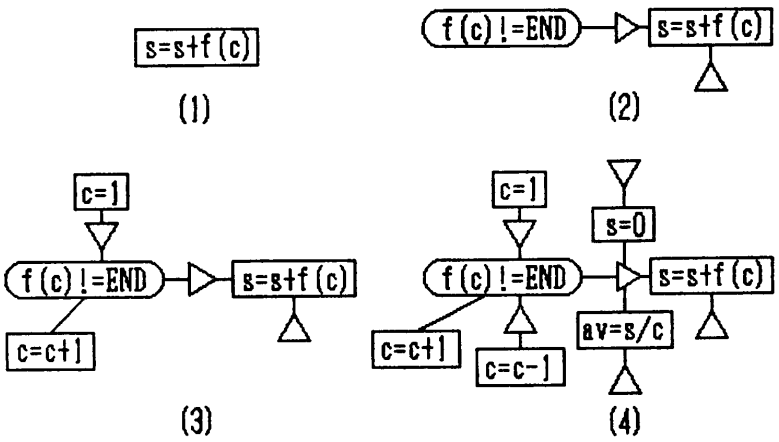

図 20 ボトムアッププログラミングの例
Fig. 20 Example of the bottom-up programming.

ことを強調する意味で変数 c の初期化と増分処理を制御側に寄生させる.

(4) このループから抜け出す時点で変数 c から1を減じると, 変数 c はデータの個数となる. 一方ループの被制御側にある変数 s を0に初期化する. 変数 s は変数 c と異なりループの制御変数ではないことを強調するためメイン側で初期化を行う. s を c で割ることで av に平均値を求めることができる.

なお, 図20(4)からはメイン処理が変数 s の初期化, 総和の計算, 平均値を求める ($av=s/c$) の三つの部分からなっていることがわかる. また制御側, 被制御側, ループに対する初期化, 変数の制御, 後始末といった処理の質的側面を構造的に把握できる点にも注意されたい. ただしこのプログラムにおいて,「変数 c はメインでも使用しているから c の初期化はメインで行うべきである」と考えることも可能である. したがって, 同じ流れのプログラムではあっても設計者によってプログラム構造が異なってしまうことは十分に考えられる. しかし, 少なくとも設計者はプログラム中の変数や処理に対する質的な見方を構造的に主張できる. このことはプログラムの保守において, 保守担当者に設計者の考え方をより詳細に伝達でき, 保守後も設計者の思想を連続させる手助けとなる.

6. TS チャートに対する評価

ここでは TS チャートが実務面でどの程度有効であるかを調査した結果について述べる. 調査対象は TS チャートを社内標準ツールもしくは公認ツールとして使用している一部上場企業を含めた5社20人である. 調査結果を以下に示す. この中の評価点は各個人の TS チャート使用感によるものであり, 統計的な厳密さはないので一応の目安とされたい.

(1) TS チャートの使用期間: 6か月～7年, 平均2年
(2) TS チャートによる開発・保守プログラムの本数: 3～100本, 平均23本.
(3) 使用した言語
アセンブラ, C, BASIC, PASCAL 等.
(4) プログラムの種類
- OS(ファイル管理等).
- ユーティリティー (キー入力の解析, UNIX 用通信ソフト等).
- アプリケーション(歯科用事務処理システム, 生体計測用等).
- ハードの制御 (RS232C, プリンタの制御等).

(5) 通常のフローチャートを1とした評価
- 開発効率: 1.2～2, 平均 1.6
- 保守効率: 1～5, 平均 2.04

(6) 利点
- 制御構造が分かりやすい.
- モジュール化が容易.
- 思考の過程を表現しやすい.

(7) 欠点と要望
- チャートが横に広がりコンパクトに書けない.
- 記号に慣れるまでが大変.
- 構造化されていない言語にも使いやすくなるように.

この調査結果の中で特に興味深いのは, 開発の効率以上に保守の効率が上がっていることであり, このことは利点として上げられた意見からも推察できる. これは TS チャートが目的とする保守性の重視にほぼ沿った結果となった. ただし(7)のような欠点や要望については今後研究の余地がある.

7. おわりに

本論文ではプログラムの開発と保守の両者を効率よく統一的に扱える文書化の技法として TS チャートを提案した. TS チャートはモジュールを基本単位とし

モジュールの差し込み，モジュールの制御，寄生というプログラム構造を導入した．これらの構造を用いることでプログラムの制御構造のみでなく，制御部，被制御部，メイン，準備，初期化，後始末といった処理の意味を構造的に明確化できることを示した．さらにこれらの構造を用いることでトップダウンとボトムアップ併用によるプログラムの開発と保守を可能にした．

TS チャートは研究室の学生が作成するプログラムの引き継ぎを滑らかにすべく保守性を重視して開発してきたものである．TS チャートは今回調査させていただいた企業以外に，個人的使用を許可したり組織的な採用の検討に入る等の企業が増えつつある．したがって今回述べたプログラム構造以外にも興味ある構造が発見できるものと期待している．

謝辞 本論文を作成するに当たり快くアンケート調査等に協力していただいた関係各位，現在の TS チャートの体系にまとめ上げるまでのどろどろとした過程で TS チャート使用者として協力していただいた卒業生ならびに学生諸君，そして筆者の見落としていた誤りや新たな観点を指摘していただいた査読者に対してこの場を借りて心から感謝の意を表する．これらの方々の御協力や御意見がなければこのような論文は作成できなかった．

最後になるが，本研究を進める上で日頃から多くの面で御援助を賜っている東海大学工学部飯田昌盛教授，小高明夫教授に深謝する次第である．

参 考 文 献

1) 二村：構造的プログラム作成のための一方法，電子通信学会総合全国大会論文集，S8-9 (1978).
2) 夜久，二木：フローチャートの木構造型記法，電子通信学会 AL 研資，AL78-47 (1978).
3) 大原，野島，前田：フローチャートの木構造化の一提案，電子通信学会総合全国大会論文集，1499 (1979).
4) 花田，佐藤，松本，長野：コンパクトチャートを用いたプログラム設計法，情報処理学会論文誌，Vol. 22, No. 1, pp. 44-50 (1981).
5) 二村，河合，堀越，堤：PAD (Problem Analysis Diagram) によるプログラムの設計及び作成，情報処理学会論文誌，Vol. 21, No. 4, pp. 259-267 (1980).
6) 国分屋八郎：ソフトウェア製作用ドキュメントの保守面からの考察，情報処理，Vol. 26, No. 3, pp. 246-249 (1985).
7) Ohara, S.: Tree Structured Chart, *International Conference on Artificial Intelligence Methodology Systems Applications 84* (1984).
8) 大原：木構造化フローチャートの制御構造について，第 27 回情報処理学会全国大会論文集，4C-7 (1983).
9) 大原：TS チャートによる並列表現について，電子通信学会回路とシステム研資，CAS84-209 (1985).
10) McClure, C. L. (渡辺ほか訳)：ソフトウェア・保守の管理，p. 61，近代科学社，東京 (1982).
11) 伊藤貴康：プログラム理論，p. 56，コロナ社，東京 (1975).
12) Aho, A. V. (守屋悦朗訳)：最近の計算理論，p. 160，近代科学社，東京 (1976).

(昭和 60 年 8 月 21 日受付)
(昭和 61 年 8 月 27 日採録)

大原 茂之 (正会員)

昭和 22 年生．昭和 44 年東海大学工学部電子工学科卒業．昭和 46 年同大学院工学研究科修士課程修了．同年同大学工学部電子工学科助手，現在同大学助教授．プログラム構造論，並列処理システム等に興味を持つ．著書等「電子計算機通論」「マイクロコンピュータハンドブック」(オーム社)．電子通信学会会員．