拡張されたヒープ領域管理機能をもつ PASCAL 処理系 ELPH†

中内伸二†† 萩原兼一††
都倉信樹†† 鈴木直也†††

PASCAL では，ユーザは，プログラム実行時に動的変数を使用できる．それが確保されるヒープ領域には，ポインタを用いることにより，任意のネットワーク構造が構成可能である．ところで，実行時にヒープ領域に構築された情報を他のプログラムに伝達したり，保存したりするためには，順編成外部ファイルとして出力する必要がある．このための変換は一般に困難であり，PASCAL プログラムのデータ構造の構築に関する優位性をいかしにくくしている．今回設計，作成された ELPH システムは，主としてこの不便さを解消するためのものであり，その特徴としては，1)ヒープ領域中に構成された情報を自動的にファイルの形で残すことができる，2)逆にファイルからヒープの情報を再構成することができる，3)ヒープ領域中の情報をファイルを媒体として複数のプログラムで共有使用することができる，4)プログラムの実行時にユーザは大量のヒープ領域を使用することができる，などがある．ELPH システムにより，ユーザは，ヒープ領域を使用するプログラムを書くことが容易になり，また，入出力のための本質的でない変換処理をコード化する必要がなくなったことにより，プログラムの了解性，読解性も向上する．

1. まえがき

プログラミング言語 PASCAL[1] は豊富なデータ構造が使用可能で，プログラム構造も簡明であり，現在，多くの処理系が作成されている．それにより，アルゴリズム記述，システム記述などに広く利用されている．

しかし，現在の PASCAL については，ユーザが使いにくいと考えられる点がいくつかある．

その一つに，コンパイルの単位がプログラムのみであるということがあるが，現在は，手続きごとのコンパイルが可能である分離コンパイルシステム[2] などにより解決されている．

もう一つは，動的記憶領域とファイルに関してである．PASCAL では，実行時に標準手続き new によって動的記憶領域 (heap area，以下，ヒープ領域と呼ぶ) が活用できる．この機能は有用ではあるが，ユーザの立場から次のような不便な点があげられる．

プログラム P_1 がプログラム P_2 に情報伝達を行う場合，PASCAL においては順編成の外部ファイルを用いる．ところが，ヒープ領域中のデータ構造が自由にとれるのに対し，外部ファイルは一次元的な情報しか保持できないため，ヒープ領域中の情報* を，ファイルを通してプログラム P_2 に伝達するのは簡単なことではない．つまり，ユーザが，自らの責任において，ヒープの自由な構造と順編成ファイルとの間のギャップを埋めようとすれば，P_1 の実行で一般にネットワーク構造に構成されたヒープを順編成ファイルに出力できるように一次元化する処理を記述する必要がある．ところが，その処理は複雑で，そのアルゴリズムもヒープ領域中のデータ構造に依存することが多い．ユーザが P_1 中でこの変換の処理を記述することは煩雑であるばかりか，そのためにプログラムの了解性を損うことにもなりかねない．また，P_1 が出力した順編成ファイルを入力として，P_2 中でヒープを復元する処理に関しても同様である．

以上のような PASCAL におけるヒープ領域に関する不便な点を解消し，PASCAL-∑C** 分離コンパイルシステム[2]を改良して，PASCAL 処理系 ELPH[3),4)] (以後，**ELPH システム**と呼ぶ) を設計，作成した．

本論文では，2章で ELPH システムにおけるヒープの概念，3章で設計方針，4章で機能および応用分野，5章で外部仕様，6章で実現方法，7章で性能評価を中心に述べる．

† ELPH System—PASCAL with Filed Heap Facility by SHINJI NAKAUCHI, KEN'ICHI HAGIHARA, NOBUKI TOKURA (Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University) and NAOYA SUZUKI (SONY Corporation).

†† 大阪大学基礎工学部情報工学科
††† ソニー株式会社

* 連結状態をも含めた，ヒープ領域中の全情報を，以後，ヒープと呼ぶ．

** 言語 PASCAL-∑C と標準 PASCAL との相違点は，おもに次の2点である．
(1) 手続きごとの分離コンパイルのための，手続きの宣言部と定義部の分離．
(2) text 型以外の **file** 型および実数型をもたない．

2. ELPH システムにおけるヒープの概念

PASCAL のヒープはプログラムの実行時にのみ存在する．しかし，ELPH システムでは，ヒープとファイルとの間の相互変換に関する支援を主目的とするので，プログラムの実行終了後も，ヒープに準ずるものを考える．それは，実行時のヒープ領域中の全情報が格納されたファイルという形態をとり，**ヒープファイル** (heapfile) と呼ぶ．これにより，ヒープの寿命は，プログラムの寿命と無関係となる．

ヒープファイルは，プログラム間でのヒープ情報伝達の媒体である．すなわち，あるプログラムが残したヒープファイル中の情報は，他のプログラムによって参照，変更，追加，削除が可能である．このような状態を，プログラム間でヒープを共有しているという．また，一つのプログラムだけが何回も実行されて，しだいにヒープを更新してゆくことも可能である．

3. ELPH システム設計の指針

PASCAL のヒープを扱いやすいものとするため，ELPH システムの設計では以下のことを重視した．

（1） ヒープの内容をファイルの形で伝達でき，そのためのヒープのファイルへの格納，ファイルからヒープへの復元が容易であること．

（2） すでに開発された PASCAL プログラムを，ELPH システムを使用するプログラムにするための変更はまったく不要か，あるいはわずかでよいこと．これに関連して PASCAL の仕様（構文，意味）の変更は少ないほどよいことになる．

（3） プログラムで使用できるヒープ領域の量を大きくできること．

（4） プログラムの実行時間が不必要に長くならないこと．

（5） ヒープを複数のプログラムで共有使用する場合，それらのプログラムに共通な型宣言などは一度ですませられること．

また，ELPH システムを作成する側からの要求としては，既存の PASCAL-∑C 分離コンパイルシステムを利用し，コンパイラおよび実行時ルーチンなどの変更点をできるだけ少なくすることが考えられる．

4. ELPH システムの支援機能とその応用分野

4.1 ヒープの伝達

ELPH システムの下で作成されたプログラムにおけるヒープは，ヒープファイルとして，以降に実行されるプログラム（最初のものと別のプログラムでも同じプログラムでも可）に伝達することができる．その際，ヒープは自動的にヒープファイルに格納され，ヒープファイルからヒープへの復元も自動的にされるので，ユーザは格納や復元に関する処理の記述をする必要がない．

ただ，ヒープをファイルに格納する時期，あるいはファイルからヒープを復元する時期を，ユーザが陽に記述できると便利な場合も考えられる．そこで，ヒープの格納，復元を自動的に行わず，プログラム中の命令によって行う機構も備える．この機能を用いると，任意の時期にヒープの格納，復元ができる（詳細は5章参照）．

一般に，ヒープ領域中には，論理的には独立ないくつかの情報が混在していることがある（**図1**(a)）．そのような場合には，論理上，互いに独立な情報は，異なるヒープファイルに保存することができれば有益である．このように，ヒープを，ヒープファイルに対応するそれぞれの部分に分割したものを**サブヒープ**と呼ぶ（4.2節参照）．また，ヒープの一部のうち，プログラムの実行後は不要となる情報は，ヒープファイルにしないようにできれば効率がよい．これらを考慮に入れると，ELPH システムにおける一つのプログラムによるヒープは，一般に図1(b)のように，サブヒープに分割され，それぞれがヒープファイル(なくても可) に対応していると考えられる．ヒープをどのように分割するかは，ユーザが任意に指定できるものとする．その指定は，ヒープ領域中に取られる動的変数の型名による*．たとえば，型名 T1 の動的変数の情報はヒープファイル1に，型名 T2 と T3 の動的変数はヒープファイル2に格納する，というような指定が可能である（詳細は5章参照）．また，ユーザの利便のため，ヒープ領域の獲得および解放はサブヒープごとに可能とする．

ELPH システムの下で動くプログラムでは，一般に，ヒープファイルからのヒープの復元，ヒープから

* PASCAL-∑C, ELPH とも，プログラム中での型の一致性は構造によって判断している．

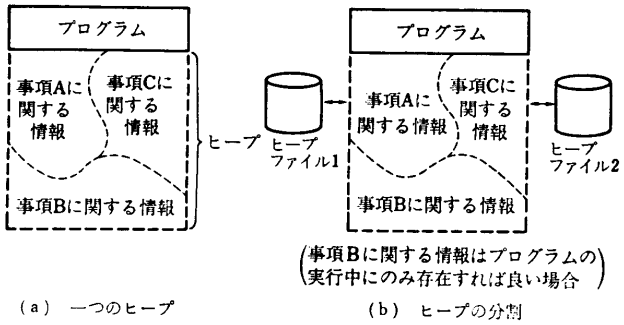


図 1 プログラムとヒープ

Fig. 1 Program and its heap.

ヒープファイルへの格納が行われる．これは，実際には，ヒープファイルと結合* されるオペレーティングシステム上の実ファイルに対しての入出力により行われるのであるが，その実ファイルの指定は，入力，出力とも，同じにすることも，別のものとすることもできる．後者の場合は，入力に用いた実ファイルは保存されるので，以前のものを再利用することも可能である．この両者の選択は実行時にできる．

これらの機能によって，データを更新するために一つのプログラムが何度も実行されるようなシステムや，データの複雑なバージョン管理を必要とするようなシステムを容易に実現できる．具体例としては，PASCAL のヒープ領域を利用した情報管理システム，簡易データベースなどが考えられる．

4.2 ヒープの共有

複数のプログラムは，ヒープ領域中の情報をヒープファイルの形式を用いて共有することができる．この場合，ヒープを共有するプログラムの集合および共有されるヒープの型に関する情報などが一つの大きな機能単位を構成することになる．この単位を**プログラムシステム** (program system) と呼ぶ (**図 2** (a))．さらに，4.1 節で述べたように，ヒープの分割が可能であるので，一般的なプログラムシステム中では，概念的には図 2 (b) のように，サブヒープごとの共有を行う．

プログラムシステムは，共有するヒープの型に関する情報や，各プログラムにおいて共通に使用できる定数に関する情報などを有しており，各プログラム中で別個にそれらを記述する必要はない．

プログラムシステム中に含まれるプログラムの実行順序および実行回数は任意であり，実行時には，一つのプログラムが主記憶上にロードされて実行される．

また，プログラムシステムに新しいプログラムを追加したり，プログラムシステム中のプログラムを削除したりすることができ，追加の場合でも，追加されるプログラムだけをコンパイルすればよい．

ELPH システムにおける，ヒープの共有の機能を用いることにより，複数のプログラムで一連のまとまった仕事を行うようなシステムを簡単に実現できる．また，コンパイラでのシンボル表などをヒープファイルとして保存しておけば，種々のツールで有益に用いられうる．

なお，現在実現されている PASCAL-ΣC 分離コンパイルシステムでは，9 個のプログラムがヒープ領域中の情報を一次元化して順編成ファイルで入出力を行

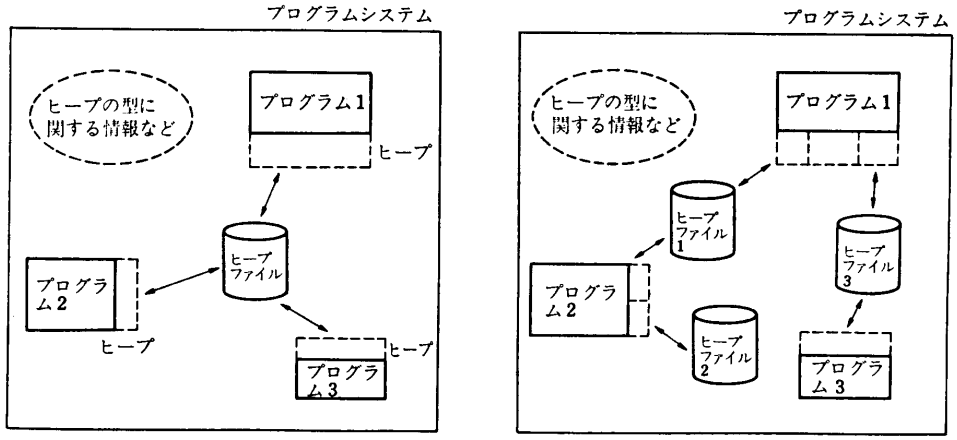


図 2 プログラムシステムの概念図

Fig. 2 Image of program system.

* ヒープファイルの名前と，オペレーティングシステム上でのファイルの名前を対応づけ，入出力可能にすることを，ヒープファイルと実ファイルとの結合という．

っている．このPASCAL-∑C分離コンパイルシステムもELPHシステムを用いることにより，より簡潔なものに改良される．

4.3 ヒープ領域量の拡大

プログラムの実行時には，ヒープ領域は膨大な量が必要となる場合がある．仮想記憶方式を支援していない，とくに小型計算機ではこの問題は深刻であり，ヒープ領域が不足となるような場合，ユーザはヒープ領域を細かく解放したり，途中結果を一時的に外部記憶に退避したりすることになる．しかし，このような処理により，本来行うべき仕事の内容が不明確になりやすい．したがって，自然な処理手順の記述のためには，使用可能なヒープ領域量は多いほうが好ましい．

ELPHシステムでは，実行時のヒープ領域量の制限は事実上撤廃されているので，ユーザはヒープ領域量に関する制約を意識することは不要となる．

実際上，この機能は重要であり，ヒープ領域量に関連して処理能力が限定されていたプログラムに対しては，その能力を向上させることになる．

5. プログラムシステムの仕様

4章で述べたプログラムシステムの，ソーステキスト上での構成は**図3**のように，**システムヘッダ部**および**プログラム群部**から成っている．以下では，それぞれの外部仕様について述べる．

5.1 システムヘッダ部の仕様

システムヘッダ部は，

（1） **共有定数**

（2） **共有型**

（3） ヒープファイル

の定義を行う部分である（**図4**にシステムヘッダ部の構文図，**図5**にその例を示す）．

共有定数および共有型とは，プログラム群部中のどのプログラムからも参照可能な定数および型である．それらの定義は，それぞれ，共有定数宣言部，共有型宣言部で行われる．なお，共有型の一部は，後述するセーブ型宣言により，一部のプログラムにおいてはセーブ型として使用される．

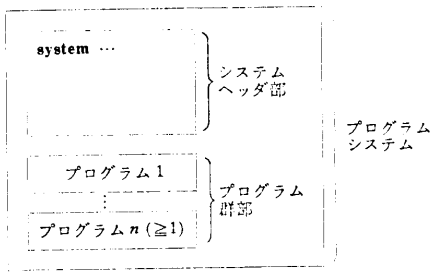


図3 プログラムシステムの構造
Fig. 3 Structure of program system.

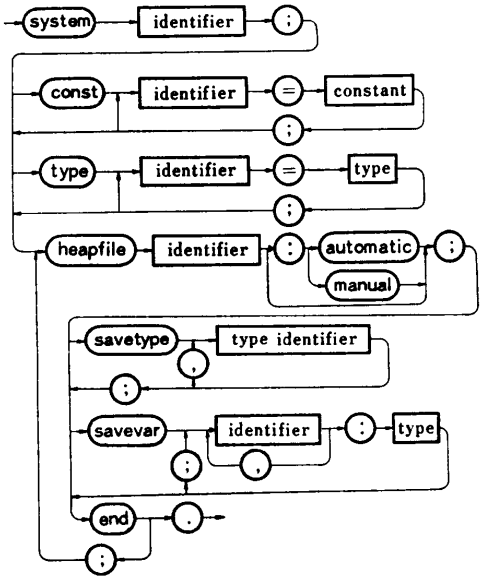


図4 システムヘッダ部の構文
Fig. 4 Syntax of system header part.

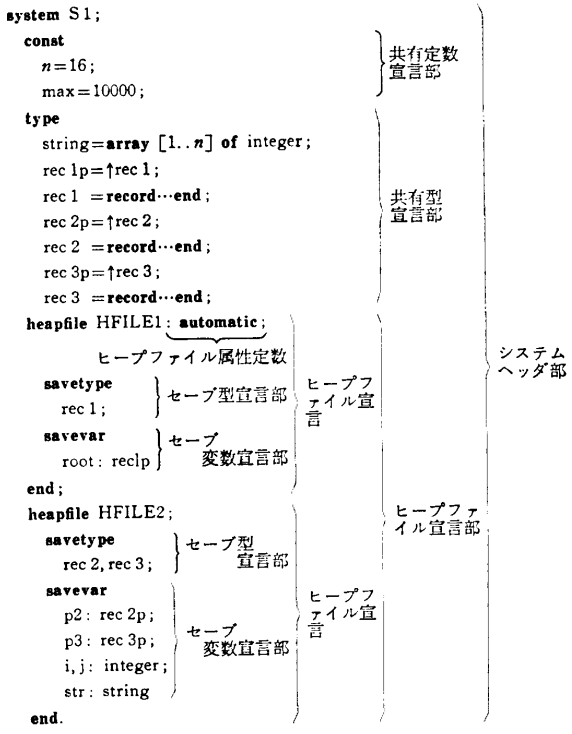


図5 システムヘッダ部の例
Fig. 5 Example of system header part.

共有定数宣言部，共有型宣言部に続くヒープファイル宣言部には1個以上のヒープファイル宣言が含まれる．1個のヒープファイル宣言では，

① ヒープファイル名
② **ヒープファイル属性**
③ **セーブ型** (savetype) 名
④ **セーブ変数** (savevariable)

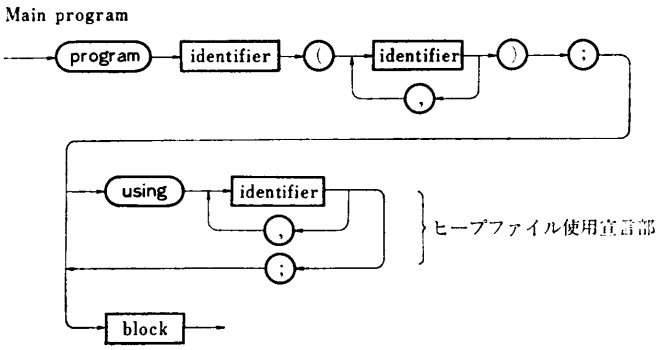
が定義される．

ヒープファイル属性は，ヒープの出入力の方法を指定するもので，**automatic 属性**および**manual 属性**の2種がある．automatic 属性のヒープファイルの入力（ヒープの復元）および出力（ヒープの格納）は，プログラムの実行開始時および実行終了時に自動的に行われる．したがって，automatic 属性のヒープファイルを用いる場合には，ユーザはそのヒープファイルの入出力に関する記述をする必要がない．一方，manual 属性のヒープファイルの入力および出力は，ユーザがプログラム中で，5.3 節に述べる標準手続きを用いて行う．manual 属性の特徴としては，

（i） ヒープファイルの入出力を行うかどうか，行うならばいつ行うかということを実行時に指定できる
（ii） ヒープファイルと実ファイルとの結合を実行中に変更できる

などがあげられる．この機能は，エディタのように，エンドユーザの入力により異なる種々の仕事を行うような場合に便利である．なお，ヒープファイル属性の指定がない場合には，automatic 属性とみなす．また，automatic 属性のヒープファイルの結合は，プログラムの実行開始のコマンドに付随して指定する．

システムヘッダ部ですでに定義されている共有型名は，セーブ型として指定できる．このとき，そのセーブ型は，そのヒープファイルに属するという．セーブ型をさすポインタ変数を引数として標準手続き new が実行されると，その動的変数の値は，そのセーブ型が属するヒープファイルに格納される．図5の例では，ヒープファイル HFILE1 を使用するプログラム中で，rec1 型をさすポインタ変数で標準手続き new が実行されると，そのとき確保された領域の値はヒープファイル HFILE1 に格納される．なお，一つのヒープファイルに属するセーブ型の個数は任意であるが，1個の共有型名を二つ以上のヒープファイルでセーブ型とすることはできない．また，プログラム P が使用する（5.2節参照）ヒープファイルに属するセーブ型以外の共有型は，P においては単なる一般の型とみなされる．さらに，どのヒープファイルにも属さない共有型は，たんなる，各プログラムに共通な型ということになる．



block など他の構文は PASCAL-∑C に準ずる．詳細は文献2)参照．

図 6 プログラムの構文
Fig. 6 Syntax of program.

セーブ変数とは，ヒープファイルにその値が格納される変数である．あるヒープファイル宣言中で定義されたセーブ変数は，そのヒープファイルに属するという．セーブ変数の値は，ヒープがヒープファイルに格納および復元されるのと同じ時期に，それが属するヒープファイルに格納および復元される．図5の例では，ヒープファイル HFILE1 には，セーブ型 rec1 の動的変数とともに，rec1p 型のセーブ変数 root の値も保存される．一般にセーブ変数は，ヒープ構造のどこかの部分をさすポインタ変数とか，ヒープの大きさを表す変数などのように，ヒープと密接に関連する値を保存するために用いられるものである．セーブ変数は，それが属するヒープファイルの使用宣言（5.2節参照）をしたプログラム中でのみ使用可能であり，他のプログラムからは，その名前は不可視である．

システムヘッダ部はプログラムシステムの基幹となるもので，以下のプログラム群部中の各プログラムをコンパイルするときの環境を構成する．

5.2 プログラム群部の仕様

図3のように，プログラム群部は一つ以上のプログラムから構成される．プログラムの構文は，ヒープファイル使用宣言部を除いて，PASCAL-∑C と同一である（**図6**にプログラムの構文概略，**図7**にプログラム群部の例（図5の例に続くもの）を示す）．

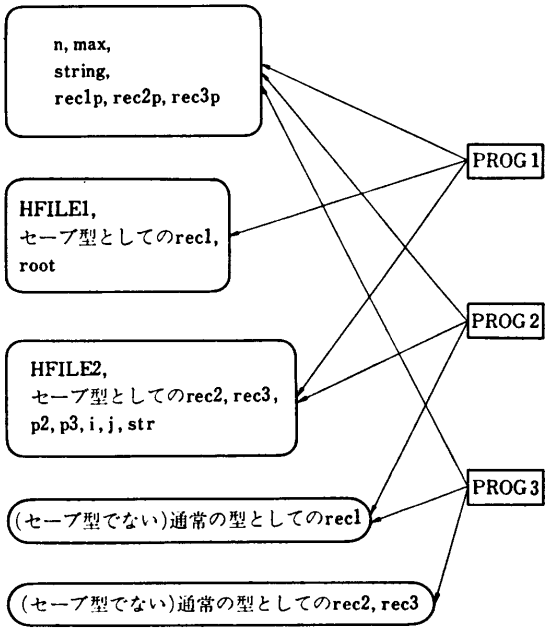
プログラム頭部に続く，予約語 **using** を伴った

```
program PROG 1 (input, output);
  using                          } ヒープファイル
    HFILE 1, HFILE 2;            } 使用宣言部
  label                          } プログラム
    :
  begin                          }
    :                            }
  end.                           }
program PROG 2 (input, output);  }
  using                          } ヒープファイル
    HFILE 2;                     } 使用宣言部
    :                            } プログラム
  begin                          }
    :                            }
  end.                           }
program PROG 3 (input, output);  }
  type T=array [0..1] of rec 3;  }
    :                            } プログラム
  begin                          }
    :                            }
  end.                           }
```

プログラム群部

図 7 プログラム群部の例

Fig. 7 Example of program group part.



←―― は □ のプログラムが ⬭ の定数，型，変数あるいはヒープファイルを使用可能であることを示す．

図 8 図5と図7の例におけるプログラムと名前の参照関係

Fig. 8 Relationship between programs and identifiers according to Fig. 5 and Fig. 7.

ヒープファイル使用宣言によって，そのプログラムがどのヒープファイルを用いるかを指定する．用いるヒープファイルの宣言により，同時に，使用できるセーブ型およびセーブ変数も決定される．

図7におけるプログラム PROG 3 のように，ヒープファイル宣言部が空のプログラムが存在してもよい．このようなプログラムではヒープファイルは取り扱わないが，実行時に使用可能なヒープ領域量は多くなる（数 M 語まで，詳細は 6.1 節参照）．

以上のように，システムヘッダ部の情報と，プログラム中のヒープファイル使用宣言によって，各プログラムが使用するヒープファイルや，その中に格納される変数に関する情報が与えられる．たとえば，図5および図7から構成されるプログラムシステムにおいては，各プログラムが使用できるヒープファイル，型などは**図8**のようになる．

5.3 仕様に関する補足

（1） システムヘッダ部を伴わないプログラムシステム

プログラムシステムの構文は，一般的には図4および図6に示したとおりであるが，特例として，システムヘッダ部をもたない，1個の PASCAL-∑C プログラムもプログラムシステムとして認める．このようなプログラムは，ヒープファイルとは無関係であるが，実行時のヒープ領域量は拡大される（6.1 節参照）．つまり，ELPH システムのこの機能だけを使用する場合には，ユーザはソースプログラムの変更がまったく不要である．

（2） 標準手続き unsave, save

manual 属性のヒープファイルの入出力の時期を明示する機能を果たすのが，標準手続き unsave と save である．それぞれ二つの引数をもち，第1の引数はヒープファイル名，第2の引数はそれに対応づけられるオペレーティングシステム上でのファイル名をバイト詰めにしたもの（文字を要素とした **packed array** に相当）である．標準手続き unsave により，ヒープおよびセーブ変数の値が復元され，標準手続き save により，それらが格納される．

（3） 標準論理関数 newfile

1個の引数をもち，それはヒープファイル名である．引数で示されるヒープファイルと結合されている実ファイルが存在すれば true，そうでなければ false を返す．manual 属性のヒープファイルでは，標準手続き unsave が実行されたときにヒープファイルと実ファイルが結合されるので，それ以前の未結合時には newfile の値は定義されない．この標準論理関数は，主として，プログラムが実行されるのが1回目（すなわち，ヒープには何の情報もない）かどうかを調べるために用いられるものである．

6. ELPH システムの実現

ELPH システムは，小型計算機 NOVA 3（主記憶容量 96K 語（1語は16ビット），磁気ディスク装置容量 20M バイト）のオペレーティングシステム MRDOS の下で稼動中の PASCAL-∑C 分離コンパイルシステムを基本として作成された．PASCAL-∑C コンパイラは2パス方式で，中間言語としてPコード[5]を生成する．Pコードプログラムは相対二進形式に展開，アセンブルされた後，PASCAL-∑C 実行時ルーチンを加えて実行形式に変換される．

以下，実現方法について項目にわけて示す．

6.1 ヒープファイル

ヒープファイルは磁気ディスク上のランダムアクセスファイルとして実現する．また，プログラムの実行時に，主記憶量以上のヒープ領域を使用可能とするため，実行時にもディスクをヒープのために用いる．すなわち，基本的にはヒープ領域はディスク上に設け，主記憶のうち，通常使用されていない拡張アドレス空間と呼ばれる領域（現在，最大 44K 語）をバッファとして用いる．拡張アドレス空間へのアクセスは，MRDOS の提供する窓（window）の機構[6]を用いて高速に行える．また，拡張アドレス空間とディスクとの間の情報転送も 1K 語ごとに比較的短時間で可能である．拡張アドレス空間およびディスクファイルは 1K 語を単位として考え，それをブロックと呼ぶ．ユーザプログラムの実行時の記憶階層は，概念的には**図 9** のようになる．

バッファの管理は需要ページング（demand paging）方式[7]をとる．バッファ中のブロックをディスクに追出す場合，どのブロックにするかを決める，追出しブロック選出のアルゴリズムには多くの種類がある．ELPH システムにおいてはどのアルゴリズムが優れているかに関しては，ユーザのヒープの使用方法に依存するので一概に決められない．そこで，標準的には，最も簡単な巡回法（round robin）[7]を用いる．そして，その部分は差替え可能としておき，種々のアルゴリズムで作動させ，時間的な効率も測定できるようにした．

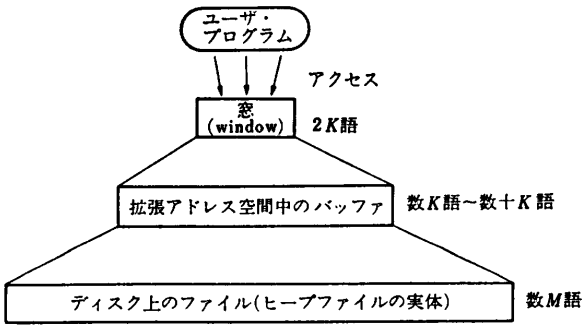


図 9 ユーザ・プログラム実行時の記憶階層
Fig. 9 Hierarchy of storages.

6.2 コンパイラおよび実行時ルーチン

ヒープファイルを扱えるように，PASCAL-∑C コンパイラおよび実行時ルーチンを次のように変更した．

ポインタ変数は従来1語で扱っていたが，ヒープ領域量の拡大に伴って2語に変更し，従来のPコードでは対処できない機能をもつ 10 個のPコードの追加を行った．

実行時ルーチンは，新設されたPコードに関する処理，バッファ管理ルーチン，ヒープファイルとディスクファイルを結合するコマンド処理などが追加された．

7. ELPH システムの評価

ELPH システム実現後の評価として最も問題となるのは，ユーザのプログラム実行時の時間的効率である．

今回は，PASCAL-∑C でのプログラムと実行時の効率を比較するために，簡単な二分探索木（binary search tree）* 作成の問題を選んだ．二分探索木では，データが新しく入力されるたびに二分木を根からたどるので，バッファ中のブロックのスワップが頻繁に起こり，ELPH システムにとってはこれは厳しい例題である．評価のための比較項目は，

① データを入力しつつ木を作成する時間
② ヒープをファイルに出力する時間（ELPH システムでは手続き save の実行時間）
③ ファイルからヒープを復元する時間（ELPH システムでは手続き unsave の実行時間）

の3点であり，**表1**にその結果を示す．表1(a)のバッファ量が 22K 語の項でわかるように，ディスクアクセスによる ELPH システムでの効率低下は妥当な範囲に留まっており，(b)のような，PASCAL-∑C ではヒープ領域があふれて処理できない範囲でも適当な処理時間となっている．

表1の測定は，スワップするブロックの選出アルゴ

* データを入力しながら，より小さなデータは二分木の左の枝へ，より大きなデータは二分木の右の枝へとたどって，新しい葉頂点となる場所を探索する．

表1 効率測定結果
Table 1 Results of the measurement of efficiency.

(a) PASCAL-∑C で扱える最大限度の入力量の場合

システム ＼ 測定量 / バッファ量		①	②	③	①+②+③
ELPH	11K語	5分45秒	3秒	13秒	5分48秒
	22K語	2分57秒	3秒		3分0秒
	31K語	1分38秒	6秒		1分44秒
	43K語	1分38秒	6秒		1分44秒
PASCAL-∑C	—	1分13秒	1分4秒	1分2秒	3分19秒

バッファ量22K語の場合，PASCAL-∑C システムで用いられる主記憶量と ELPH システムで用いられる主記憶量とが同じ値になる．その他の11K語，31K語，43K語という値は，実際上多用されるバッファ量である．

(b) (a)の約2.4倍の入力量の場合

システム ＼ 測定量 / バッファ量		①	②	③	①+②+③
ELPH	11K語	36分44秒	3秒	23秒	37分10秒
	31K語	18分5秒	3秒		18分31秒
	43K語	13分5秒	3秒		13分31秒
PASCAL-∑C	—	処理不能			

リズムが巡回法の場合である．同じ問題に対して，最低使用頻度 (least frequently used)[7] アルゴリズムでは平均約14%の効率向上，最長未使用 (least recently used)[7] アルゴリズムでは平均約7%の効率向上となっている．ただし，最長未使用アルゴリズムは，バッファ量が少ないときの性能がとくに優れている．

8. あとがき

ELPH システムは，ユーザが PASCAL の構造をも含めたヒープを容易に，自由に扱えることを第一目標として設計された．現在の実現において，この目標は十分に満足されていると思われる．ヒープを扱うようなプログラムは作りやすくなり，また，プログラムの書きやすさとともに読解性も向上した．さらに，ELPH システムの周辺の各種ツールやライブラリ手続き群も整備されて，本格的に使用されるようになっている．これには，PASCAL-∑C から引き継いだ分離コンパイル機能も大きく貢献している．ELPH システムは，前述のように，NOVA 3 上で実現されているが，ELPH システムのコンパイラは PASCAL で記述されているため，追加した10個のPコードの機能およびバッファ管理に関する機能などを他の計算機用に書き直せば，容易に移植可能である．

なお，ELPH システムの大きさは，コンパイラフェイズ1が2,150行，13K 語 (PASCAL-∑C の約10%増)，フェイズ2が5,280行，25K 語 (同じく約20%増)，実行時ルーチン (アセンブリ言語) が6K 語 (同じく約50% 増) となっている．

謝辞 本研究に関し，ご助力，ご助言いただいた荒木俊郎講師をはじめとする研究室諸氏に深謝いたします．

参考文献

1) Jensen, K. and Wirth, N.: *PASCAL User Manual and Report*, 2nd ed., Springer-Verlag, New York (1975).
2) 鍵政，荒木，都倉：PASCAL 内部手続きの分離翻訳，電子通信学会論文誌 (D)，Vol. J 63-D, No. 2, pp. 177-182 (1980).
3) 中内，鈴木，萩原，都倉：PASCAL のヒープ領域のある拡張について，情報処理学会第22回全国大会，2B-4 (1981).
4) 中内，萩原，都倉，鈴木：拡張されたヒープ領域管理機能をもつ PASCAL 処理系 ELPH の作成，電子通信学会技術研究報告 (電子計算機)，EC 82-19 (1982).
5) Nori, K. V., Ammann, U., Jensen, K., Nägeli, H. H. and Jacobi, Ch.: *The PASCAL ⟨P⟩ Compiler: Implementation Notes*, Revised Ed., Institut für Informatik, ETH, Zürich (1976).
6) 日本・データゼネラル：NOVA/ECLIPSE リアル・タイム・ディスク・オペレーティング・システム (RDOS), Rev. 06 解説書プログラミング編，第5版 (1979).
7) ハーバマン，A. N.：オペレーティングシステムの基礎，土居範久訳，培風館，東京 (1978).

(昭和57年8月2日受付)
(昭和57年10月4日採録)