C言語のメモリ管理機能の拡張　"D"

4Y-3

小林　茂　　落合　正雄

㈱東芝　府中工場

1．はじめに

C言語のためのメモリ管理機能「D」を開発した。Dを用いることにより、動的データ構造の生成やアクセスがC言語の自然な構文で行なえる。また、自動的に発生するガーベジ・コレクションについてもプログラマはほとんど意識する必要がない。

以下にDの機能と実現法について紹介する。

2．D開発の目的と方針

ソフトウェアの会話性・柔軟性が重要化するのに伴い、その内部で扱われるデータも構造・大きさが不定であることがむしろ一般的になってきている。しかし、C言語をはじめとする従来のシステム記述言語では、こうした動的データ構造に対する記述力が十分ではない。一方、LISPのように動的データ構造の操作に適した言語も存在するが、これらの多くは性能の面、あるいはプログラマ人口の面から実用的ソフトウェアへの適用には制約がある。そこで、C言語の性能や細かな制御に関する記述性はそのままに保ちながら、データ構造の操作を手軽に行なえるようにするためのメモリ管理機能「D」を開発した。

上記のような目的から開発にあたっては特に次の点を考慮した。

①データ要素の構造をプログラマが自由に定義できること。
②自動的なガーベジ・コレクション（GC）を行なうこと。
③なるべくC言語本来の記法で記述できること。また、記述上プログラマが意識しなければならない制約を設けないこと。
④最小限のメモリ空間で実行でき、かつ大きなデータ構造を扱うプログラムにも対応できること。
⑤移植性が高いこと。

3．利用形態と機能

3.1　利用形態

Dの機能は、関数とマクロを通して利用される。その一部はライブラリ関数として与えられ、他はプログラマによるデータ定義ファイル（以下dファイル）から、変換ユーティリティ（dtrans）によって生成される。dtransの出力は、hファイル、cファイルと呼ぶ2つのCソースファイルである。hファイルは、Dの提供するマクロ類を含み、cファイルはデータ要素生成関数を主な内容とする。

このほかにDには必要に応じてリンクされるいくつかの拡張ライブラリがある。（図1）

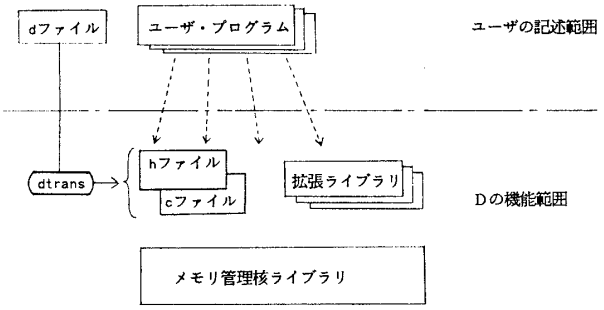

図1．Dの機能構成

3.2　メモリ・インタフェース

Dの最も基本的かつ重要な機能は、dファイルで定義されたデータ要素の構造に基いて生成されるメモリ・インタフェース（要素生成・アクセス機能）である。dファイルにおけるデータ要素の定義は、次のようにC言語の構造体風に記述される。

型名｛メンバ型名1 メンバ名1;　メンバ型名2 メンバ名2;…｝

メンバ型として指定できる型は表1の通りである。

integer（または int）	整数型メンバ
flonum （または float）	実数型メンバ
dflonum（または double）	倍精度実数型メンバ
pointer（または ptr）	ポインタ型メンバ

表1．データ要素のメンバ型

例として、データ要素を次のように定義したとする。

```
node {int nval; ptr left; ptr right;}
```

この場合、次のインタフェースが生成される。

NODE　……定義データ要素の型名
"if(TYPE(p)==NODE) …" のように使用できる。

node(i,P1,P2) …メンバ値がそれぞれ、i,P1,P2 であるようなNODE型データ要素の生成。

nval(p)
left(p)
right(p)　……メンバのアクセス。参照・代入のいずれの側にも使用可。

以上のようなプログラマによる定義型のほかに唯一のメンバを持つデータ型とn個の同一型メンバを持つデータ型（chunk）、および文字列型が標準で利用できる。その生成・アクセスインタフェースと型名を表2に示す。

生成	メンバアクセス	型名
integer(i)	VI(p)	INTEGER
flonum(f)	VF(p)	FLONUM
dflonum(d)	VD(p)	DFLONUM
pointer(p)	VP(p)	POINTER
chunk(INTEGER, n)	XI(p, k)	
chunk(FLONUM, n)	XF(p, k)	[ISCHUNK(p)]
chunk(DFLONUM, n)	XD(p, k)	
chunk(POINTER, n)	XP(p, k)	
string(s)	——	[ISSTRING(p)]

注). []は型名ではなく型判定述語

表2．標準のデータ型

An enhancement of memory management facility for C language —D—

S. Kobayashi,　M. Ochiai

TOSHIBA Corp.

4．メモリ管理機能の実現

前述のｄファイルから生成されるメモリ・インタフェースの説明からも想像できるように、データ要素の生成は必要な大きさの領域を確保して初期値を設定する関数として、またメンバのアクセスは、与えられたポインタにキャストをかけ、該当するメンバをアクセスするマクロとして実現できる。Ｄの実現上のポイントは、ヒープからのデータ要素の切出しと型付け、およびＧＣの方式である。次にこの２点について述べる。

4.1 データ要素の型とヒープ管理

データの型名（先の例における"NODE"や、"INTEGER" など）の実体は、「型情報」と呼ぶ構造体へのポインタである。型情報の中には、その型に関するメモリ管理上必要な全ての情報（要素のサイズや自由要素リストのアドレスなど）が含まれる。

データ要素に型を付加する方式としては、メモリの使用量を最小限に抑え、かつヒープ領域の拡張に上限を生じないという観点から図２のような方式を採用した。

即ち、メモリをブロックと呼ぶ一定の大きさ（現在は１KB）の領域に分割し、それぞれのブロックを１つのデータ型に割当てる。ある型の要素が必要になったとき、自由要素が得られなければ新たに１つのブロックをその型に割当て、その中を自由要素に分割する。ブロックはすべてブロック境界に置かれるので任意のポインタ値に対して、その値をブロックサイズで丸めることによりデータ型を取出すことができる。

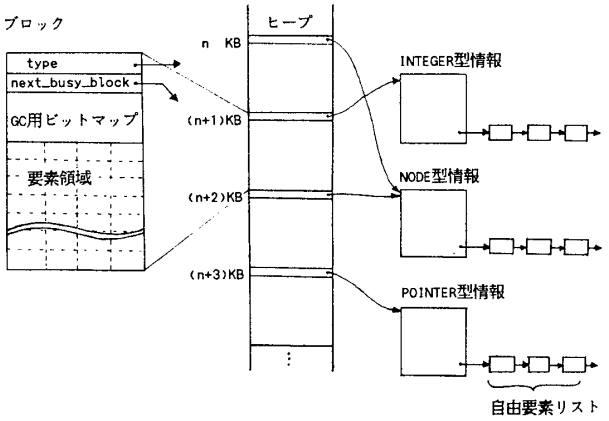

図２．ヒープとデータ型の管理

4.2 ガーベジ・コレクション（GC）

ＧＣは、必要になったデータ型の自由要素が空のときに自動的に発生する。（ＧＣによっても自由要素を回収できなければ、前述のブロック割当てが行なわれる。但し、これはデフォールトの処理で、自由要素リストが尽きた場合などメモリ管理上の特定事象が検出されたときの処理は、実行時に切換えることができる。）ＧＣはマーキング+かき集めによる一括方式で行なっている。マーキングの開始点となるのは、「ポインタ型」の変数である。この型は、Ｃ言語本来のポインタ型とは異なりＤが生成した任意のデータ要素を指すことができる。ポインタ型の変数には、次の３つの種類がある。

(1) グローバルポインタ：すべての関数から参照できるポインタ変数で、ｄファイルで宣言される。その実体は、大域変数として宣言されたポインタ型配列の要素である。

(2) ローカルポインタ：関数内で自動変数的に使用されるポインタ変数で、特別なマクロを使って宣言される。このマクロは関数の開始時にポインタ変数の領域を「ローカルスタック」に確保する。ローカルポインタは、関数からの復帰時に解放される。

(3) ポインタ型引数：ポインタ型の引数を持つ関数は、ｄファイル中で名前と引数型を宣言しなければならない。これによって、その関数の呼出しは、ポインタ型の実引数を関数の呼出しに先立って評価し、「引数保護スタック」にpushするようなコードに展開される。（したがって引数の評価順序が問題になる場合には注意を要する。）引数保護スタックも関数からの復帰時に解放される。

以上のポインタ変数から開始されるマーキング処理は、指されているデータ要素のポインタ型メンバに対して、再帰的に適用される。マーキング用の再帰関数は、ｄファイルにおける構造体の定義をもとにＣファイルに生成される。

5．拡張ライブラリ

Ｄの機能の中核はメモリの管理であるが、LISPに見られる有用な機能の一部を拡張ライブラリとして提供している。その代表的なものは、リスト処理ライブラリと入出力ライブラリである。２進木リストは非常に汎用性の高いデータ構造であるため、これを対象としたコピー・反転・検索等の機能レパートリを揃えておくことは、プログラムをコンパクトにし可読性を高める。入出力ライブラリは、整数、実数、文字列および「シンボル」の並びからなるテキストファイルを対象として、内部データとの間で変換を行なうものであり、データ要素の動的生成、型管理の機能により可能となっている。

6．おわりに

Ｃ言語のためのメモリ管理機能Ｄについてその機能と実現法を紹介した。変換ユーティリティdtrans自身をはじめ、高速プロダクションシステムなど既に数例の開発にＤを適用しており、有効性を確認している。今後、Ｃ言語処理系との緊密な融合をはかることによりＣ言語との親和性の改善（特にポインタ型変数の宣言など）や実行効率の向上が可能であると考えている。

参考値：malloc関数との性能比較

プログラム

```
case-1 (malloc)
  構造体:  struct mstruct {int m1; int m2;}
  ループ:  for (i=0; i<LOOPCNT; i++) {
             p=(struct mstruct *)malloc(sizeof(struct mstruct));
             p->m1=1; p->m2=2;
           }

case-2 (D/ポインタなし)
  構造体:  dstruct {int d1; int d2;}
  ループ:  for (i=0; i<LOOPCNT; i++) p=dstruct(1,2);

case-3 (D/ポインタあり)
  構造体:  cons {ptr car; ptr cdr;}
  ループ:  for (i=0; i<LOOPCNT; i++) p=cons(NULLPTR,NULLPTR);
```

実行時間比

case-1	1
case-2	0.42
case-3	0.53

（AS3160にて測定．LOOPCNT=100000，GC抑止）