計算機言語Cの役割り

大阪大学基礎工学部　辻　野　嘉　宏

1. はじめに

システム記述用言語Cは、ＵＮＩＸオペレーティング・システム（ＯＳ）上で開発され、また、ＵＮＩＸ自身を記述していることで有名である。Cや、その前身であるＢＣＰＬ，Ｂなどの言語が生まれる以前は、ＯＳは通常アセンブリ言語で書かれていた。その主な理由は次の通りである。

(1) アセンブリ言語は、高級言語に対して、実行時間や記憶領域の点で効率のよい目的コードが得られる。もちろん、これは、アセンブラが高級言語コンパイラよりも機能が高いためというわけではなく、単に、アセンブリ言語がその構造上最適化をプログラマに押しつけることができ、実際に押しつけているためである。

(2) ＯＳの持つデータ構造には、しばしばビットフラグが用いられる。これは、記憶容量を少なくするためだけでなく、計算機のハードウェアの構造としてビットごとに別の意味を持つものが多いためでもある。しかし、ほとんどの高級言語には、このようなビット操作を行う機能が用意されていなかった。

(3) 計算機の周辺装置のドライバを作成する際、特定の番地にあるレジスタを参照しなくてはならないが、このような機能を持つ高級言語は少なかった。

(4) 新たに製作された計算機には、アセンブラもコンパイラも最初はないという場合がほとんどである。したがって、ＯＳ作成にとりかかる前にアセンブラかコンパイラを作成しなくてはならないが、当然、コンパイラの作成には、手間も時間も格段に必要である。しかもよい目的コードを出力するコンパイラともなれば、なおのこと大変である。

しかしながら、ＯＳのような巨大なプログラムを作成する場合、すべてをアセンブリ言語で記述することの欠点も大きい。たとえば、

(1) アセンブリ言語で記述されたプログラムは読みにくい。そのため、より多くのドキュメントが必要となり、保守や管理が面倒となる。

(2) 同じ処理を記述するにしてもステップ数が高級言語で記述する場合に比べて多くなり、開発期間、工数が多くなる。

(3) 高級言語で記述されたプログラムは、計算機が置き換えられても、ほとんど変更なしに新しい計算機上で使用できるが、アセンブリ言語で記述されたプログラムは、全体を書きなおさなくてはならない場合が多い。すなわち、移植性はまったくない。

などがある。

以上のような問題を解決するために種々の高級アセンブラとも呼ばれるような言語が開発されてきたが、その中でも比較的高級な言語がCである。Cは、以上に述べたアセンブリ言語の利点を持ちながら、高級言語風にすることによってその欠点を解消するという厚かましい言語である。以下にその特徴を挙げる。

(1) 後に述べるように、よい目的コードを出力するための努力を可能な限りプログラマに押しつけることにより、簡単によい目的コードが得られる。逆にプログラマの立場から見れば、アセンブリ言語で記述した場合と同じく、効率のよいプログラムを書いたという満足感を得ることができる。

(2) ＯＳの記述には必須のビット操作のための演算子を持つ。

(3) ポインタ変数を用いて任意の番地をアクセスできる。さらに、ほとんどのCコンパイラには、Cのプログラム中にアセンブリ言語を混ぜることができ（インライン機能）、それを用いて特権命令なども記述できる。

(4) Cの言語仕様が小さいことと、上に述べたように最適化の努力が最小限ですむことにより、簡単にコンパイラが作成できる。

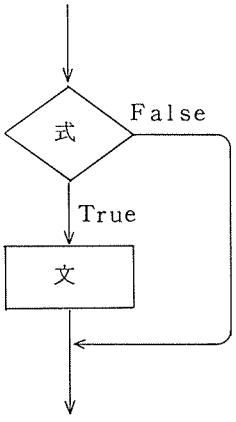
(5) Pascal風の制御構造（実際にはRatforに近い）とデータ構造を含み、他の高級言語と同程度に読みやすく、保守しやすいプログラムを書くことができる。また開発効率もアセンブリ言語を用いる場合に比較してはるかによい。

(6) Cで記述されたプログラムの移植性は、アセンブリ言語の場合に比べて非常に高い。しかし、これは条件付であって、プログラムの機能と書き方による。つまり、ＯＳのように元来計算機に依存した処理を含むプログラムのような場合はPascalのような他の高級言語と同程度というわけにはいかない（もっとも、そのようなプログラムはPascalでは記述できないかもしれない）。また、特定の計算機やCコンパイラに依存した記述（整数型の大きさが4バイトであることを利用するなど）を用いた場合は、移植性は低くなる。

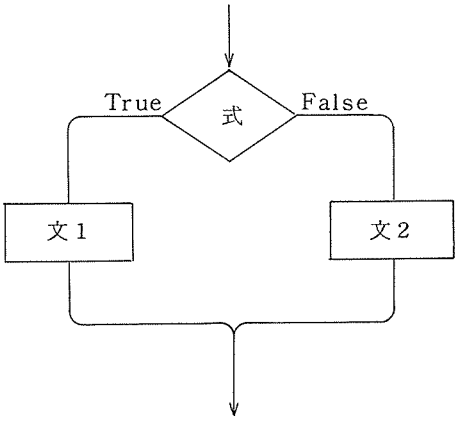
以上の特徴から容易にわかるように、Cは、Pascalのような高級言語のユーザから、アセンブリ言語のユーザまで吸収し得る適用範囲の広い言語である。また、非常に書きやすく、しかも簡潔に書けてしまうので、一度Cに慣れてしまうとアセンブリ言語やPascalは使う気がしなくなるという麻薬的な性質もある。以下にCの具体的な記述例を挙げる。

2. 制　御　文

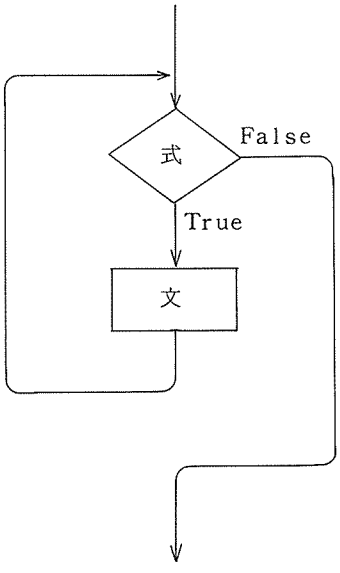
Cは、図1に示すような制御構造を持つ。if 文，switch文は分岐を行い、前判定ループであるwhile文、for 文、後判定グループであるdo 文がループを形作る。この他、制御の流れを



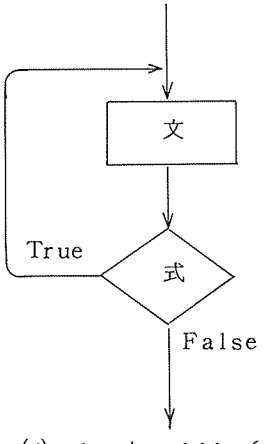
(a) if (式) 文



(b) if (式) 文1 else 文2

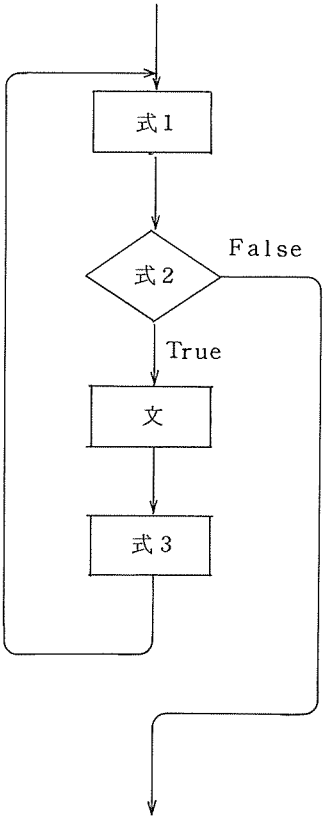


(b) while (式) 文

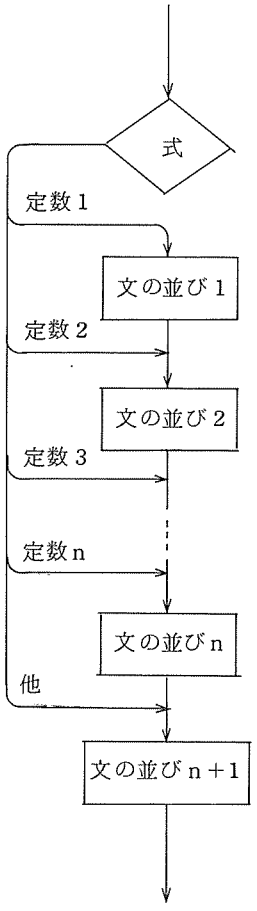


(d) do 文 while (式) ;

図1. Cの制御構造 (続く)



(e) for（式1；式2；式3）文



(f) switch（式）{

case　定数1：

文の並び1

case　定数2：

文の並び2

case　定数n：

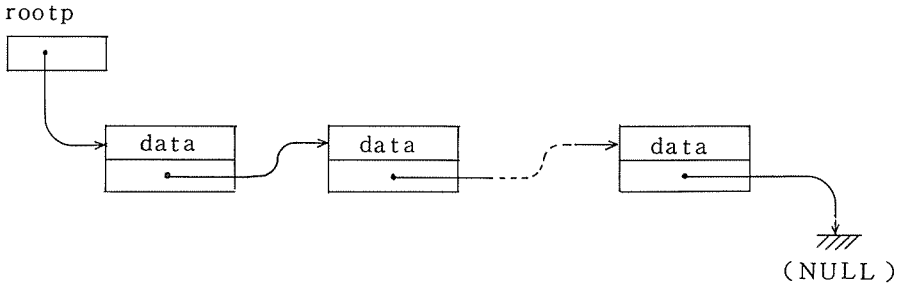
文の並びn

default：

文の並びn+1

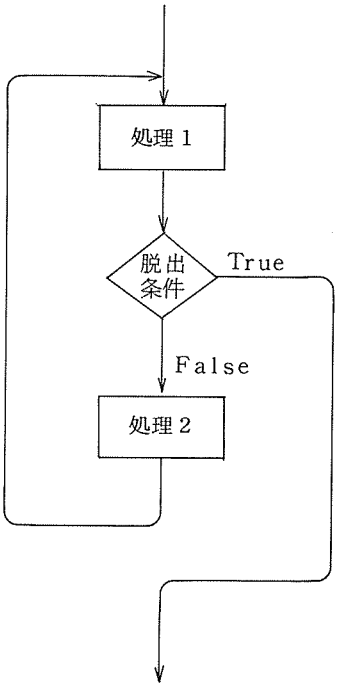
}

図1. Cの制御構造　（続き）




```
for(p=rootp;p!=NULL  &&  P->data !=key ;P=P->next);
return(p);
```

図2. リスト構造をたどるfor文の例




```
for ( ; ; ){
      /* 処理1 */
   if(脱出条件) break ;
      /* 処理2 */
 }
```

図3. break文を用いた中判定ループ

```
switch( c ) {
  case '+' :
  case '-' :
                s = "sign" ;
                break ;
  case '0' :
  case '1' :
  case '2' :
  case '3' :
  case '4' :
  case '5' :
  case '6' :
  case '7' :
  case '8' :
  case '9' :
                s = "digit" ;
                break ;
  default  :
                s = "other character" ;
}
```

図4. switch文の例

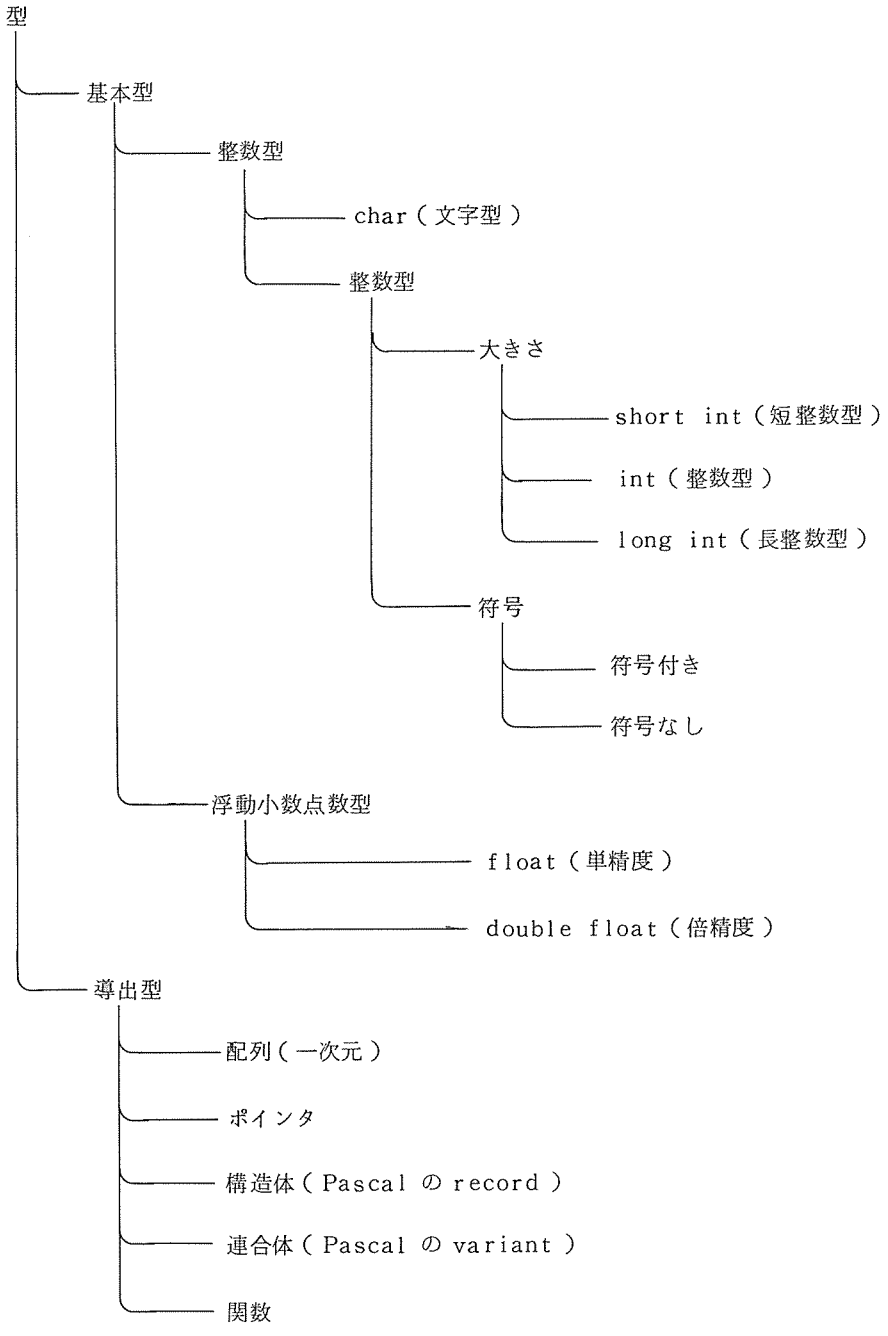

型
基本型
整数型
char（文字型）
整数型
大きさ
short int（短整数型）
int（整数型）
long int（長整数型）
符号
符号付き
符号なし
浮動小数点数型
float（単精度）
double float（倍精度）
導出型
配列（一次元）
ポインタ
構造体（Pascal の record）
連合体（Pascal の variant）
関数

図5. Cの型

変更するbreak文、continue文、goto文などがある。以下、Cに特徴的なfor文とswitch文を中心に説明する。

for文は、Pascalのfor文、FortranのDO文とは違い、制御変数というものを持たない。それ故、柔軟性があり、利用範囲が広い。図2は、リスト構造をたどり、値keyを持つノードへのポインタを返す例である。for文の式1において、ポインタ型変数pの値を初期化した後、式2でリスト構造の最後か否かのチェックと値がkeyと等しいか否かのチェックを行う。リスト構造の最後でなく、値がkeyと異なる場合には、pの値を次のノードへのポインタに更新して判定を繰り返している。もし、値keyを持つノードがない場合にはNULLを返す。また、for文の3個の式はそれぞれ省略することができる。特に式2を省略した場合には、無限ループとなる。この機能とbreak文(while文、for文、do文やswitch文中からその次の文へ制御を移す脱出用goto文)を併用すれば中判定のループを構成することができる(図3)。

switch文は、Pascalのcase文と同様の働きをする。もっとも異なる部分は、一つのcaseラベルに飛び込み、文の並びを実行して次のcaseラベルに達してもswitch文を終わらず、次の文の並びに制御が移る。switch文を終了する場合は、break文を用いて陽にswitch文から脱出する。図4はswitch文の例である。この例は、文字型変数Cの値が符号('+','−')であれば、ポインタsが"sign"を、数字('0','1',…,'9')であれば、"digit"を、それ以外であれば"other character"を指すようにするswitch文の例である。

3. データ構造

Cは、図5に示すように各種の型を持つデータを用いることができる。導出型はほとんどすべての型と組み合わせることができる。同一の型の集まりである配列と異なる型の集まりである構造体型によって、プログラムで必要なデータ構造を構築することができる。この点においては、CはPascalなどと同様の使い方ができる。しかしながら、CはPascalと比べてはるかに型変換が自由である。たとえば、整数型を指すポインタ型は、移植性をほとんど損わずに任意の構造体型を指すポインタ型に変換可能である。また、配列名は、それが変更できないことを除けば、その要素型へのポインタ型と扱いは同じである。すなわち、ポインタ型変数の持つデータはアセンブリ言語で言う番地であり、配列名はラベルである。このように、Cはその下にアセンブリ言語が透けてみえている。

4. Cの式と演算子

Cは、表1に示すように、代入演算子(=など)を含む豊富な演算子を持っている。その特徴

はビット操作用の演算子であり、式の記述性のよさである。

ビット操作用の演算子として、論理和（｜）、論理積（＆）、排他的論理和（へ）、否定（～）、があり（図６）、これらは、条件判定用論理演算子としても使用可能である。

記述性を高めるための演算子として、＋＋，－－演算子や各種の代入演算子がある。図７にその例を示す。また、図８は、文字列の終端子である'＼０'を見つける例である。Ｃの論理値は０（False）か、０でないか（True）で判定されるのでこのような記述が可能である。図９は、空白を読み飛ばす例である。以上のように、Ｃでは強力な式の機能により、簡潔にプログラ

表1. Ｃの演算子

優先順位	演算子	意　味	結合性
1	（　）	関数呼び出し	左→右
	〔　〕	配列の要素参照	
	．	構造体の要素参照	
	－＞	ポインタの指す構造体の要素参照	
2	！	論理否定	右→左
	～	ビットごとの否定	
	＋＋	＋１	
	－－	－１	
	－	符号の反転	
	(type)	型変換	
	＊	間接参照	
	＆	その変数の番地	
	sizeof	その型の大きさ	
3	＊	積	左→右
	／	商	
	％	剰余	
4	＋	和	
	－	差	
5	≫	右シフト	
	≪	左シフト	

優先順位	演算子	意　味	結合性
6	＜	比較（＜）	左→右
	＜＝	〃（≦）	
	＞	〃（＞）	
	＞＝	〃（≧）	
7	＝＝	比較（＝）	
	！＝	〃（≠）	
8	＆	ビットごとの論理積	
9	へ	ビットごとの排他的論理和	
10	｜	ビットごとの論理和	
11	＆＆	論理積（条件付）	
12	｜｜	論理和（条件付）	
13	？：	if～then～else	右→左
14	＝	代入	
	＋＝	加算代入	
	－＝	減算代入	
	＊＝	乗算代入	
	：	：	
15	，	順次演算	左→右

```
getbits(x,p,n)  /* get n bits from position p */
  unsigned x, p, n ;
 {
  return( ( x >> ( p+1-n ) ) & ~( ~0 << n ) ) ;
 }
```

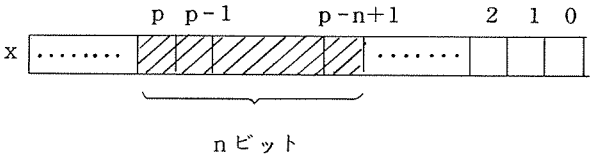


図6. ビット操作の例

```
for( i = 0 ; i < 100 ; i = i+1 ) a[i] = 0 ;
```

(a) C特有の演算子を使わない例

```
for( i = 0 ; i < 100 ; i += 1 ) a[i] = 0 ;
```

(b) 代入演算子の使用例

```
for( i = 0 ; i < 100 ; a[i++] = 0 ) ;
```

(c) ++演算子の使用例

図7. Cによるいろいろな記述例(大きさ100の配列の初期化)

```
for( i = 0 ; a[i] != '¥0' ; i++ ) ;
```

(a) 通常の記述

```
for( i = 0 ; a[i] ; i++ ) ;
```

(b) 論理判定の性質を利用した記述

図8. 論理判定の例

```
read( c ) ;
while c = ' ' do read( c )
```

(a) Pascalによる記述

```
while( ( c = getchar() ) == ' ' ) ;
```

(b) Cによる記述

図9. 空白を読み飛ばす例

ムが記述できる。

また、Cには論理判定用の演算子として、&&と||があるが、この評価法はPascalとは異なる。たとえば、

```
if ( 0 <= i  &&  i <= 9 )
```

なるif文があったとき、もしiの値が負であれば、「i <= 9」は評価しないということである。これは効率のよい目的コードが出力できるというだけでなく次の点で有利である。今、pを構造体型へのポインタ型の変数であるとする。このときPascalであれば、

```
while ( p <> nil ) and ( p^.data = 1 ) do…
```

というwhile文はpの値がnilの場合もp^.dataを参照するので実行時エラーを引き起こす可能性がある。しかもこの判定は単純には正しく書きなおすことはできない。Cにおいては、単に、

```
while ( p != NULL  &&  p->data == 1 )…
```

と書けばよい。

5. Cにおける最適化

前に述べたように、Cにおいては、最適化はある程度プログラマに委ねられている。局所変数のregister宣言はその例である。この宣言は、コンパイラにこの変数がよく用いられることを知らせるものである。また、代入演算子もプログラマに委ねられた最適化の例である。たとえば、

```
a[i][j] = a[i][j] * k ;
```

は、配列の番地計算を2度行うが、同じ処理を

```
a[i][j] *= k ;
```

とも書くことができ、この場合には、配列の番地計算を1度しか行わない。

以下、我々の研究室で作成したMC68000用Cコンパイラを例にとってソースプログラム上の記述の仕方と目的コードの関係を示す。図10～12は、配列を用いた文字列コピーの例である。図13～16は、ポインタを用いた文字列コピーの例である。とくに図16はregister宣言を用いている。これらの例からわかるようにCにおいてはプログラムの書き方により目的コードの性質が大きく変わる場合が多い。

6. おわりに

以上に示した例からも明らかなように、Cは、高級言語の能力を持ちながら、高級言語に特有な種々の制限を持たない。言いかえるならば、CはPascalのようにも使える一方、アセンブリ

```
strcopy1(p,q)
  char p[], q[] ;
 {
  int i ;

  for( i=0 ; p[i] != '¥0' ; i++ )
    q[i] = p[i] ;
  q[i] = p[i] ;
 }
```

(a) ソースプログラム

(関数本体の大きさ：８４バイト
ループの大きさ：５８バイト)

```
                    CLR.W    -10(A6)
.2:                 EQU      *
                    MOVE.L   -4(A6),A4
                    MOVE.W   -10(A6),D7
                    MOVE.B   0(A4+D7.W),D7
                    EXT.W    D7
                    TST.W    D7
                    BEQ.F    .4
                    BRA.F    .3
.5:                 EQU      *
                    ADDQ.W   #1,-10(A6)
                    BRA.F    .2
.3:                 EQU      *
                    MOVE.L   -4(A6),A4
                    MOVE.W   -10(A6),D7
                    MOVE.L   -8(A6),A3
                    MOVE.W   -10(A6),D6
                    MOVE.B   0(A4+D7.W),0(A3+D6.W)
                    BRA.F    .5
.4:                 EQU      *
                    MOVE.L   -4(A6),A4
                    MOVE.W   -10(A6),D7
                    MOVE.L   -8(A6),A3
                    MOVE.W   -10(A6),D6
                    MOVE.B   0(A4+D7.W),0(A3+D6.W)
```

(b) 関数本体に対応する目的プログラム

図１０. 目的プログラムとの関係 (1)

```
strcopy2(p,q)
  char p[], q[] ;
 {
  int i ;

  i = 0 ;
  while( ( q[i] = p[i] ) != '¥0' )
    i++ ;
 }
```

(a) ソースプログラム

(関数本体の大きさ：４４バイト
ループの大きさ：４０バイト)

```
                    CLR.W    -10(A6)
.8:                 EQU      *
                    MOVE.L   -4(A6),A4
                    MOVE.W   -10(A6),D7
                    MOVE.B   0(A4+D7.W),D7
                    MOVE.L   -8(A6),A4
                    MOVE.W   -10(A6),D6
                    MOVE.B   D7,0(A4+D6.W)
                    EXT.W    D7
                    TST.W    D7
                    BEQ.F    .7
                    ADDQ.W   #1,-10(A6)
                    BRA.F    .8
.7:                 EQU      *
```

(b) 関数本体に対応する目的プログラム

図１１. 目的プログラムとの関係 (2)

```
strcopy3(p,q)
  char p[], q[] ;
 {
  int i ;
    ↖
  i = 0 ;
  while( q[i] = p[i] ) i++ ;
 }
```

```
                CLR.W   -10(A6)
.B:             EQU     *
                MOVE.L  -4(A6),A4
                MOVE.W  -10(A6),D7
                MOVE.L  -8(A6),A3
                MOVE.W  -10(A6),D6
                MOVE.B  0(A4+D7.W),0(A3+D6
                BEQ.F   .A
                ADDQ.W  #1,-10(A6)
                BRA.F   .B
.A:             EQU     *
```

(a) ソースプログラム　　(b) 関数本体に対応する目的プログラム

（関数本体の大きさ：38バイト，ループの大きさ：34バイト）

図12. 目的プログラムとの関係 (3)

```
strcopy4(p,q)
  char *p , *q ;
 {
  while( ( *q = *p ) != '¥0' )
    p++ , q++ ;
 }
```

(a) ソースプログラム

（ループの大きさ：32バイト）

```
.E:             EQU     *
                MOVE.L  -4(A6),A4
                MOVE.B  (A4),D7
                MOVE.L  -8(A6),A4
                MOVE.B  D7,(A4)
                EXT.W   D7
                TST.W   D7
                BEQ.F   .D
                ADDQ.L  #1,-4(A6)
                ADDQ.L  #1,-8(A6)
                BRA.F   .E
.D:             EQU     *
```

(b) 関数本体に対応する目的プログラム

図13. 目的プログラムとの関係 (4)

```
strcopy5(p,q)
  char *p , *q ;
 {
  while( ( *q++ = *p++ ) != '¥0' ) ;
 }
```

(a) ソースプログラム

（ループの大きさ：28バイト）

```
.11:            EQU     *
                MOVE.L  -4(A6),A4
                ADDQ.L  #1,-4(A6)
                MOVE.B  (A4),D7
                MOVE.L  -8(A6),A4
                ADDQ.L  #1,-8(A6)
                MOVE.B  D7,(A4)
                EXT.W   D7
                TST.W   D7
                BNE.F   .11
.10:            EQU     *
```

(b) 関数本体に対応する目的プログラム

図14. 目的プログラムとの関係 (5)

```
strcopy6(p,q)
  char *p , *q ;
 {
  while( *q++ = *p++ ) ;
 }
```

(a) ソースプログラム

（ループの大きさ：22バイト）

```
.14:    EQU     *
        MOVE.L  -4(A6),A4
        ADDQ.L  #1,-4(A6)
        MOVE.L  -8(A6),A3
        ADDQ.L  #1,-8(A6)
        MOVE.B  (A4),(A3)
        BNE.F   .14
.13:    EQU     *
```

(b) 関数本体に対応する目的プログラム

図15. 目的プログラムとの関係 (6)

```
strcopy7(p,q)
  register char *p , *q ;
 {
  while( *q++ = *p++ ) ;
 }
```

(a) ソースプログラム

```
        MOVE.L  -4(A6),A1
        MOVE.L  -8(A6),A2
.17:    EQU     *
        MOVE.B  (A1)+,(A2)+
        BNE.F   .17
.16:    EQU     *
```

(b) 関数本体に対応する目的プログラム

（関数本体の大きさ：14バイト，ループの大きさ：6バイト）

図16. 目的プログラムとの関係 (7)

言語のようにも使える。したがって、Cを用いてよいプログラムを作成しようとすると注意が必要である。Cはシステム記述用言語であり、汎用プログラミング言語でもあるが、熟練者用であって初心者用の言語ではないと言える。それ故にプログラミングを楽しむ者にとって、Cはもっとも面白い言語の一つとなっている。

ここで述べたCの各機能の説明は十分ではないし、また機能もすべてを網羅しているわけでもない。文献(1)がCのもっとも基準となる言語仕様を規定しているので詳しくはこれを参照されたい。また、この他、日本語の教科書もいくつか出版されている。[2,3,4,5]

参考文献

(1) Kernighan, B. W. and Ritchie, D. M. : The C Programming Language, Prentice-Hall (1978).
石田晴久訳：プログラミング言語C，共立出版（1981）.

(2) 米田信夫編：C，コンピューターサイエンスライブラリー、産業図書（1982）.

(3) 西田親生、五月女健治：C言語プログラミング入門、啓学出版（1983）.

(4) Hancock, L., Krieger, M. 著，アスキー出版局監訳：C言語入門，アスキー出版局（1984）.

(5) 石田晴久：Cプログラミング、岩波書店（1984）.

センターより：本センターでのC言語の使用方法及び使用例は、速報№104 を御覧ください。