

# 概略設計文からの C ソースコードの生成

恐 神 正 博\* ・ 西 田 富士夫\*\*

## A method of automatic generation of C source codes by program design documents

Masahiro OSOGAMI ・ Fujio NISHIDA

### Abstract

In the recent quest to improve software productivity and maintenance, effective methods of automatic program generation or some such system has become necessary. We have currently been studying one of these methods. It is an experimental system of automatic program generation called MAPP (Module Aided Programming system by Prolog).

This paper outlines a method of automatic generation of C source codes by program design documents made by an automated drawing from a structured program diagram. Users of MAPP may fabricate program design documents by using the module dictionary. MAPP customizes parameters and picks up insufficient statements, during the construction of C source codes. Users may also use their native language in program design documents. C program modules have an indefinite number of structure nesting levels (like switch\_case or if\_else\_if function) and customizing parameters. These are supported by recursive call of conversion rules on MAPP. This makes the automatic generating of C source codes more effective.

### 1. はじめに

プログラムの生産性や保守性を改善するために、近年、国内外においてプログラム設計の自動化についての研究が活発に行なわれている<sup>1)</sup>。筆者らも、それらの一環としてプログラム自動生成システムMAPPの開発・改良を重ねてきた。

今回は、設計中のプログラム構造をディスプレイ上で確認しながら、そのプログラムコードを自動的に作成する一つの方法について報告する。MAPPでは、モジュール辞書を

---

\* 事務局    \*\* 経営工学科

用い、対応するステートメントやモジュールについての、自然言語表記を許した設計文を、それらのカスタマイズや・論理チェック・補填等を行いながら作成する。Cの関数はswitch\_case文やif\_else\_if文のように場合や手続きの数が不定であったり、入出力文に含まれる変数の個数が不定の一般的な関数を含むが、それらの述語形をCのソースコードに変換する再帰的な規則を設けることで効率的に変換を行っている。

## 2. 設計文書作成とコード生成の基本的な考え方

与えられた仕様のもとにプログラム設計を行うとき、Cのシステムや利用者のライブラリにどのような基本関数やユーザ作成のモジュールがあるかを表示し、必要なものを指定、利用できれば好都合である。

Cのユーザ定義の関数や手続きは一般に

tf f(t1 x1,...,tn xn) (1)

の形をもっている。ここに、tfはこの関数が返す値のタイプ、fは関数記号、xi, ti(i=1, ..., n)は引数部の変数名とタイプである。その意味構造は関数記号を通して別にソースプログラムコードや注釈文により定義されており、一般のユーザがこれらを参照せずにその機能を推察するのは関数名から想像する以外に方法がない。ところが式(1)の注釈文は、引数部の各変数に役割助辞などを付加すれば、(1)の直訳形で常用の日本語文などを用いて分かりやすく簡潔に表せることが多い。これはCの基本関数などについても同様である。このような観点から注釈文風の設計文よりなる設計文書を作成し、これを構文解析し中間形式に変換しコードを生成する方法が考えられる。この方法は変換規則数を節減化するなどの長所をもつが、表現力に富み紛れが少ない設計文法を構築することは必ずしも容易でなく、また、利用者がこの文法を駆使するためにはある程度の記憶と習熟を必要とする。

これに対し各モジュールの関数やCの基本関数毎に注釈文に相当する呼出文を設け、呼出文毎に関数表現に変換する素朴な方法を採用することにすれば、構文規則の設定や構文解析の必要はなく、上記の目的の多くを達成することとなる。また、マウスなどでコマンドなどを指示入力することにすれば、キーボードからの入力は、変数名など設計文の一部に限られる。

これらの呼出文は式(1)の情報を含むものであれば、言語や文型は日本語、英語などに関せず言語フリーでよい。なお、呼出文中のタイプ情報の指定などは、設計文書のデータ構造定義部で定義しているものなどを、コード生成時にシステムが参照し挿入するので、紛らわしくなければ省略し簡単化することが出来る。

呼出文は機能の種別毎に集めて見出し辞書を作り、read, print, repeat, sumなどの見出しキー毎に一覧表示する。利用者は所要の呼出文番号や引数を指定して呼び出し文を作成する。作成した呼出文は設計文書に加えられ、SPD<sup>2)</sup>風の構造化図に自動変換され、視覚

的にチェックしながら設計を進めることが出来る<sup>4)</sup>。また、所望の機能をもつ呼出文がなければ、より基本的なモジュールや C の基本関数などの呼出文を組み合わせで設計文を作成する。

設計文書ができると、モジュール述語など C のコード辞書を用いて呼出文毎に C の関数表現に自動的に変換する。いま、J, P をそれぞれモジュールの呼出文と対応する C の関数呼出表現を表す変項とし、呼出文のモジュール述語を `module(H,T)` とする。このとき、

```
proc(J):-module(H,T),member(jp(J,P),H),... ,out(P).      (2)
```

のような規則を用いて、原理的に、呼出文からその関数表現を求めることが出来る。ここに `member(X,Y)` は X がリスト Y のメンバであることを表す述語である。C の基本関数に対しても同様な方法で C に変換する。なお、3 節で述べるように、変換に際し、データタイプのチェックを行い、呼出文の処理対象のデータタイプと適合する基本関数やモジュールを自動的に選択している。

また呼出文を関数定義するファイルがインクルードすべきものであれば、そのファイル名をコード辞書などから取り込み記憶し、ソースコード生成時にインクルード文を生成する。

処理関連対象のデータ構造の設計は効率的で体系的な処理のために重要な研究対象となっているが、表現法に関しては、データ構造の種類は少なく、属性も単純であり困難な問題は少ないと思われる。ここでは処理設計に対応して、データ設計文の述語表現を

```
objlist:-obj(x1),obj(x2),...,obj(xn).      (3)
```

の形で取り扱う。各処理対象は、名前とタイプ名、注釈などからなる。処理が構造体のときはタイプを `struct` (タグ名) とし、タグ名毎にメンバリストを定義する。また、配列である場合、例えば、名前が a, タイプが実数、最大要素が 200, 初期値が `v1, v2, ..., vn` の 1 次元配列は、

```
obj([name(a),type(float),array(dim(1),size(n),max(200)),
    val([v1,v2,...,vn]) ]).      (4)
```

で表す。

### 3. コード辞書の構成

設計文書から目的言語のコードを生成するために、コード辞書として呼出文毎にステートメント述語またはモジュール述語を設ける。ここに、ステートメントとは C 言語の基本関数を呼出文風に表した命令文を意味する。C の基本関数の呼出文は、ステートメント述語を用いて述語形に変換し、これから変換述語を用いて C の関数に変換する。他方、モジュールは、利用者がいくつかの C の基本関数を用いて作成したプログラムモジュールであり、これを含むファイルを関数呼出で取り込んで使用する。呼出文はモジュール述語を用

いてCの関数呼出表現に直接変換する。

### 3. 1 ステートメント述語

式(5)にステートメント述語の主要部分を示す。

関数適用のための入出力条件などは省いている。

```
statement([jp(JE,PE),arg_type([ARG_TYPE_LIST]))). (5)
```

項 jp の引数部 JE には、この基本関数の呼出文、PE には、述語的表現を格納する。項 arg\_type の引数、ARG\_TYPE\_LIST には、呼出文中に含まれる変数の名前とデータ構造名を格納する。

式(5a)(5b)にステートメント述語の例を示す。式(5a)はキーボードからのプロンプト表示出力付きで読み込む関数のステートメント辞書であり、入力要求表示と読み込みの2個のCのコードを生成する。また、変数 TYPE は、int, float, char などの中から任意のものを指定できる。式(5b)は分かりやすさの点から式(5)の JE に常用の英語文表現を当てている。T や S1 や S2 などの変数は変項でなく文を表すものであるのでタイプ指定を行っていない。

```
statement([jp([プロンプトにより,OBJ,を読み込む],
prompt_read(OBJ)),
arg_type([name(OBJ),type(TYPE)]))]. (5a)
```

```
statement([jp([if(T),then,S1,else,S2,end_if],
if_else(T,S1,S2))]. (5b)
```

### 3. 2 モジュール述語

各モジュール述語は、

```
module([Head,Tail]). (6)
```

のように引数部を Head 部と Tail 部に分けている。Head 部は呼出文項目や処理対象のデータ構造の項目、Tail 部はC言語における関数形や、関数の本体部を格納したファイル名などからなる。なお、モジュール適用のための入出力条件や関数宣言などの項目は省いている。式(7)に、プロンプトにより1次元配列にキーボードからデータを読み込むモジュールのモジュール述語を示す。

```
module([
jp([プロンプトにより1次元配列,OBJ,に読み込む],
prompt_read_array(OBJ)), ①
argl([[name(OBJ),type(TYPE),
array([dim(1),size(N),max_size(NMAX)]))]]),②
[ funct([[int,[readladi(OBJ,N)],
[float,[readladf(OBJ,N)]])], ③
```

```
file_name([[int,'rdladi.c'],
          [float,'rdladf.c']]) ④ (7)
```

①②は HEAD 部, ③④は TAIL 部である. ①はこのモジュールの日本語文による呼出文とこれに対応する述語形を示している. このモジュールは「read」を見出しキーとして見出し辞書から検索することができる.

②の引数リスト項 argl はこのモジュールの処理対象の名前とデータ構造である. ③は C におけるモジュールの関数名と引数を表わす. ④は, そのモジュールが入っているファイル名を表し, このモジュールが使用するとき, インクルードされる.

## 4. プログラムの生成

設計文書からコード辞書を用いてプログラムを生成する.

### 4. 1 コードの生成 1

処理設計文書の内容は, 扱うデータを spec(NUM) というファイルに書き出してある場合, process 述語の本体の proc\_list 述語の引き数部に式 (8) のようなリスト形 [P1,P2,...,Pn] で記録されている.

```
process(spec(NUM)):-proc_list([P1,P2, ... , Pn]) (8)
```

これらの設計文書の内容は式 (9) により, 一つの呼出文毎に処理される.

```
proc_list([H|T]):-proc(H),proc_list(T).
proc_list([]). (9)
```

呼出文 X を引き数に含む jp(X,Y) が式 (10) に示すようにステートメント述語 S の要素であり, 且つ, この基本関数で処理する対象のデータタイプが, ②③に示すようにデータ設計文書の処理対象のタイプに適合していれば, X の述語形 Y から変換述語 c(Y) により 4.2 節で述べるように C の関数を生成する.

```
proc(X):-statement(S),member(jp(X,Y),S), ①
        member(arg_type([name(OBJ),type(TYPE)]),S), ②
        obj([name(OBJ),type(TYPE)]), ③
        c(Y). (10)
```

呼出文が制御文であるときには, その処理対象は文単位以上であり, 対象のデータ構造とは直接には関係せず, ②③のデータタイプのチェックを行わない.

モジュールの呼出文の処理も同様であるが, C 言語の関数呼出表現を直接書き出す. 式 (11) に処理対象が配列の場合の例を示す.

```
proc(X):-
        module(H,T),member(jp(X,Y),H), ①
```

```

member(arg_type([name(OBJ),type(TYPE),
array([dim(D),size(N),max(NMAX)])),H),           ②
p_obj([name(OBJ),type(TYPE),
array([dim(D),size(N),max(NMAX)])),              ③
member(func(FUNCT),T),                             ④
writelst(FUNCT),write(';'),nl,                     ⑤
member(file_name(FN),T),included(FN).              ⑥      (11)

```

①のように呼出文  $X$  を引き数にもつ項  $jp(X,Y)$  がある module 述語の頭部  $H$  に含まれるものとする。このとき、②③のようにこのモジュールの頭部  $H$  に含まれ項  $arg\_type$  の引数部のタイプが、データ設計文書から作成した  $p\_obj$  述語の引き数のタイプと一致させうる（単一化可能）かどうかをチェックする。一致させることができれば、④⑤によりモジュールの Tail 部の  $func$  の引き数部の  $FUNCT$  を C の関数呼出表現として書き出す。したがって利用者が新しくモジュールを作った場合、その関数表現に対応する呼出文を作って見出し辞書の該当分野に登録し、且つ、(7) のような module 述語をパラメータ指定方式などで作っておけば MAPP で直ちに利用することが出来る。また⑥において Tail 部の  $file\_name$  の引き数部のファイル名  $FN$  を  $included$  述語により取り込む。

タイプ不一致などによりコード生成の不可能な呼出文が現れた場合に、そのメッセージを出す規則を設け、 $proc(X)$  を頭部にもつ規則の最後に配置する。

#### 4. 2 コードの生成 2

基本関数の呼出文の述語形から C 言語の関数を生成するために、呼出文の種類毎に規則形の変換述語を設ける。

式 (12a)(12b) にプロンプト付き変数読み込みの呼出文を述語形から関数表現へ変換する規則を示す。

```

c(prompt_read(OBJ)):-included('stdio.h'),
writelst(['printf"',OBJ,'"');'],nl,
c(readl(OBJ)).                                     (12a)

```

```

c(readl(OBJ)):-included('stdio.h'),
p_obj([name(OBJ),type(TYPE)]),
ptype(TYPE,PTYPE),
writelst(['scanf"',PTYPE,'"'],),
write_type_obj(TYPE,OBJ),write(';'),nl.           (12b)

```

さて、C では標準ライブラリ関数を使用するときには、`stdio.h` などのヘッダファイルを取り込まねばならない。(12a)(12b) の `included` 述語は引数部のヘッダファイルをインクルー

ド文の中にまだ取り込んでいなければ、取り込むことを指示する。(12b)の p\_obj 述語は、処理対象 OBJ のデータ設計文から MAPP が作成する述語で、OBJ のタイプ情報 TYPE を ptype 述語により、%d, %s などの C の変換指定子 PTYPE に変えて読み込み文を作成する。

呼出文の述語形からは、引き数部の長さに関せず変換規則を一般的に表すことができる。これらの理由から、この論文では基本関数の呼出文の C への変換を、述語形への変換と、述語形から C の関数表現への変換に分けている。

例えば switch\_case 文の呼出文の述語形の例では、

```
case(INDEX_NAME,[item(1),H1,end_case],
      ....., [item(n),Hn,end_case]).
```

(13)

であり、中で指定する場合分けの個数は任意である。

このような場合、例えば次のような規則を用意しておけば、任意の個数の場合分けに対して C のソースコードを生成することができる。

```
c(case(INDEX_NAME,BODY)):-
  writelist([' switch(' ,INDEX_NAME,' ){ ' ]),nl,
  case_body(BODY),write(' } '),nl.
```

(13a)

```
case_body([[item(NUM),H,end_case]|T]):-
  writelist([' case ',NUM,' : ' ]),
  proc_list(H),writelist([' ',break; ' ]),nl,case_body(T).
```

(13b)

```
case_body([ ]).
```

(13c)

if\_else\_if 文などの生成においても同様であり、不定個の場合分けや、それぞれの関数の持つ不定個数の変数の処理を、これらの規則を用いることで、C の表現にに変換することができる。

### 4. 3 コードの生成 3

4 節では設計文書の呼出文から対応するコードを生成する手法について述べたが、全プログラム生成の概要をメインプログラムの場合を例にとり述べる。

設計文書は、データ設計 obj\_list(spec(NUM)) と、処理設計 process(spec(NUM)) とからなる。NUM は設計文書番号 である。これらを入力して次の prog 述語によりプログラムに変換する。

```
prog(spec(NUM)):-obj_list(spec(NUM)),
  main_head,process(spec(NUM)),main_end.
```

(14)

```
main_head:-(include([ ]);
  include(FILE),include_sent_gen(FILE)),
  main_header,type_decl.
```

(15)

式(14)の obj\_list 述語により、処理対象のデータ構造を取り込み、main\_head 述語によりメイン部頭部を作成した後、プロセス述語により処理部のコードを生成する<sup>3)</sup>。

メイン頭部では、included 述語により取り込まれた include 述語の引数部に記憶したファイル名を書き出す。また、type\_decl 述語により obj\_list 述語などで取り込んだ処理対象などのデータ構造の宣言文を書き出す。同様に、メイン以外の一般の関数のコードも作成できる。

以上述べたように、関数の処理本体の手続き部に対し、メイン関数を始め一般の関数の頭部の作成は次のような述語により自動的に行う。

```
prog(spec(NUM)):-obj_list(spec(NUM)),
                    function_head,process(spec(NUM)),function_end. (16)
function_head:- (include([]);include(FILE),include_sent_gen(FILE)),
                function_header,type_decl. (17)
```

ここに function\_head 述語では include 述語の引数部に取り込んだファイル名などを書き出す。function\_header 述語は関数のプロトタイプを作成し、type\_decl 述語は自動変数などの宣言文をデータ設計文書より作成する。

#### 4. おわりに

予め定めた日本語表記のモジュールや基本関数の呼出文をカスタマイズして作成した設計文から C のソースコードを自動的に変換・生成する一つの手法について述べた。プロログの記号処理能力により、変換システムは比較的コンパクトに構築することができた。変換や詳細化に要する時間は小規模のモジュールやプログラムであれば数秒から数十秒程度である。

今後は、未指定の部分を残しカスタマイズできる範囲を広げた基本モジュールを上流モジュールとして構築する研究も、重要性を増すものと思われる。

#### 【 参 考 文 献 】

- 1) 原田 実: CASE のすべて, オーム社, 1991.
- 2) 甲斐 義久他: 構造化プログラム設計図法, 共立出版, 1992.
- 3) 恐神, 西田: プロログによるモジュールの検索とプログラムの合成,  
福井工業大学研究紀要第 23 号, pp.313 ~ 320(1993.3).
- 4) 恐神, 西田: フレームワークの一構成法,  
情報処理学会第 49 回全国大会, pp.5-191 ~ 192(1994.3).

(平成 6 年12月14日受理)