C--言語からC言語への変換系の実用化

柳　洋輔[*1]　重村 哲至[*2]

Practical realization of the Language Translator that translates C-- code into C code

Yosuke YANAGI [*1]　Tetsuji SHIGEMURA[*2]

Abstract

In this paper we describe how we enabled the standard C language functions in the educational system description language C--. Because of the flexible description ability, the C language is popular in the embedded system developers. But, it is difficult for students to master C language in a short time. So, we defined C-- language for the student. It is easy to master the C-- language that has no difficult C language syntax. And we developed the translator to use C-- with many computer systems. However, it was difficult to use the standard C functions in C-- because they used the types that depended on them. Therefore we introduced new syntax to define a type easily.

Key Words :　Programming Language, System Description Language, Translator

1　背景

C 言語は,プログラム記述に対する自由度の高さから,組み込み業界において広く普及している言語である.しかし，初心者にはわかりにくい仕様が多く，短時間で習得することが難しい．例えば，「二次元配列を宣言する場合，宣言の仕方で四つの構造に変化する」，「配列を関数の引数に渡した場合，型が変わってしまうことがある」などが挙げられる．そのため、学生がマイコンなどでプログラム開発をする際に,C 言語の習得が障害になる.

そこで，初心者が短時間で習得し授業演習で使用できることを目的に教育用システム記述言語 C--[(1)]が開発された．C--言語は，C 言語を参考に，C 言語の分かり難い仕様を取り除いたものである．さらに，「配列や構造体は参照型のみ」，「論理型が存在する」，「型チェックが厳しい」など，プログラミング入門でよく使用される Java 言語の特徴が取り入れてある．C--言語を用いれば，既に他の言語を習得している学生が,マイコン等でプログラムを開発する場合,C 言語に比べ短時間で習得し開発を開始することができる。そのため，様々なコンピュータで，C--言語を使用できることが望まれる.

C--言語は 16bit の教育用パーソナルコンピュータ TaC[(2)]用に開発された言語である．従来の C--コンパイラは，TaC のアセンブリ言語を出力した．TaC 以外のコンピュータでも使用するために，他のマイクロプロセッサ用のアセンブリ言語を出力するバージョンも開発されてきた．しかし，それでは一つのコンピュータについて，一つのコンパイラを開発する必要があり，効率が悪い.

そこで，C--言語から C 言語への変換系(トランスレータ)を開発した．多くのコンピュータには C 言語コンパイラが既に開発されているので，一つのトランスレータを開発することで,C 言語コンパイラが準備されている多くのコンピュータで,C--言語プログラムが使用

[*1] 情報電子工学専攻

[*2] 情報電子工学科

可能になった.

C--言語を実用的に使用するには,入出力などの標準関数を使用できる必要がある．しかし，膨大な C 言語の標準関数をC--言語で記述し直すことは困難である.

C 言語の標準関数を C--言語の文法でプロトタイプ宣言すれば,C--言語からC言語の関数を呼び出すことが可能になる．プロトタイプ宣言で，FILE 型などC言語ライブラリに依存する型が利用できる必要がある.

本研究では，このような型を利用可能にするために騙し型をC--言語に導入した．これにより，C 言語の標準関数が利用可能になり,C--言語がより実用的なものとなった.

2　C--言語

C--言語は，16bit 教育用コンピュータ TaC のシステム記述言語として開発された．以下にその特徴を示す.

2.1　データ型

データ型は，基本型と参照型に分けられる．基本型には，16bit 整数値を表現する int 型，8bit 文字コードを表現する char 型，論理値を表現する boolean 型がある.

参照型には，配列型と構造体型があり，多次元配列も宣言可能である．参照変数は Java 言語のように，配列や構造体の本体とは別に，参照を確保するだけである．そのため，配列や構造体の本体は別の領域に確保する必要がある．一次元配列と構造体はメモリ上では同じ構造をしている．文字列は，C 言語のように char 型の配列を用いて表現する.

2.2　関数

C 言語と同様に関数を定義して使用することができる．バグが発生しないように，返す型を省略することはできない．値を返さない場合も，void を明示しなければならない．また，仮引数は自動変数のように扱われ，可変個引数の関数も宣言することができる.

2.3　変数

静的変数と自動変数に分けられる．関数外部で宣言された変数は静的なグローバル変数，関数内部で宣言された変数はローカルな自動変数として扱われる.

ローカル変数は関数内のどこでも宣言可能で，有効範囲はブロックの終わりまでである．同じ名前があった場合はローカル変数が優先される．有効範囲に重なりのある同名のローカル変数は許されない.

2.4　演算子

C 言語を参考に，一通りの演算子が準備されている．しかし，レパートリーの多い代入演算子や前置後置などの組み合わせが複雑なインクリメント演算子とデクリメント演算子は省略されている.

2.5　文

文には空文，式文，ブロック文，if 文，while 文，do-while 文，for 文，return 文，break 文，continue 文がある．switch 文を除けば，C 言語の文は一通り揃っている.

if 文などの条件式は Java 言語のように論理型である．そのため，条件式に誤って代入式を記述するバクを防ぐことができる．簡単な C--言語プログラムの例をリスト1に示す.

3　トランスレータ

トランスレータの内部構造を図1に示す.C--言語とC言語は文法がよく似ている．しかし，意味的に異なる部分が多いため，字面で変換することは難しい．そこで,C--言語プログラムの意味を表現する構文木を作成した上で,C言語を生成する.字句解析部,構文解析部,最適化部はC--コンパイラのものを流用している.

```
int[] a = {4,2,3,5,7};

int f(int[] c, int n){
  int sum = 0;
  for(int i=0; i<n; i=i+1)
    sum=sum+c[i];
  return sum;
}

public int main(){
  int b = 0;
  b = f(a,5);
  return 0;
}
```

リスト1　C--言語プログラム例

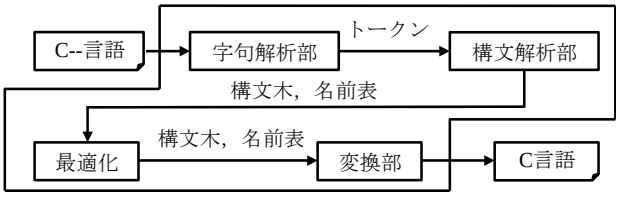

図1　トランスレータの内部構造

3.1 構文木

C 言語を出力するために，C--コンパイラの構文木を改良した．例えば，通し番号を用いて表現されていたローカル変数は，それの名前が C 言語の出力時に利用できるように，名前表のインデックスで表現するように改良されている．

3.2 変換部

構文木と名前表を使用し C 言語を生成する変換部を新規に作成した．変換部は「(1) グローバル変数宣言の完了」，「(2) 構造体宣言の完了」，「(3) 関数定義(プロトタイプ宣言を含む)の完了」のイベントが発生したときに，C 言語プログラムを出力する．以下に C--言語と C 言語の違いが分かりやすい，配列宣言，構造体宣言を変換した例を示す．

3.2.1 配列宣言

リスト 2 に配列宣言を含む C--言語プログラムを，図 2 にリスト 2 で表現される配列の構造を示す．

C--言語の一次元配列 a は，参照変数が配列インスタンスの先頭アドレスを指す構造になる．しかし，C 言語の配列は，変数名が配列インスタンスの先頭アドレスを示すので構造が異なる．プログラムの意味を変化させないために，C--言語と同じ構造の配列を C 言語で出力する必要がある．

トランスレータはリスト 2 の一次元配列 a をリスト 3 の 1～2 行目のような C 言語プログラムに変換する．まず，配列インスタンス_cmm_a_cmm_0 を生成する．次に，配列インスタンスを指すポインタ変数 a を出力する．これにより，C--言語と同じ構造の C 言語の一次元配列が記述できる．

static 修飾子は変数名のスコープを C--言語と同じにするために付加される．_cmm_a_cmm_0 のような，仮の変数名が必要である．仮変数名には，C--言語プログラムの変数と名前が重複しないようにプレフィックス「_cmm_」を付加している．

多次元配列の場合は，配列インスタンスを参照配列が指し，参照配列を参照変数が指す構造になる．この構造を C 言語で表現するために，まず，リスト 3 の 3～4 行目のように，必要数の配列インスタンスを生成する．次に，5 行目のように生成した配列インスタンスを指すポインタ配列を生成する．最後に 6 行目のように，ポインタ配列を指すポインタを宣言する．

```
int[] a = {1,2,3,4,5};
int[][] b = {{1,2,3},{4,5}};
```

リスト 2　配列定義を含む C--言語プログラム

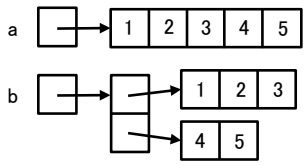

図 2　C--言語の配列の構造

```
1: static int _cmm_a_cmm_0[] = {1,2,3,4,5};
2: static int *a = _cmm_a_cmm_0;
3: static int _cmm_b_cmm_0[] = {1,2,3};
4: static int _cmm_b_cmm_1[] = {4,5};
5: static int *_cmm_b_cmm_2[] =
            {_cmm_b_cmm_0,_cmm_b_cmm_1};
6: static int **b = _cmm_b_cmm_2;
```

リスト 3　配列定義を含むプログラムを変換した結果

3.2.2 構造体宣言

リスト 4 に構造体宣言を含む C--言語プログラムを，リスト 5 にリスト 4 を変換した C 言語プログラムを示す．

リスト 4 の構造体 x と構造体配列 y の構造は図 3 のようになる．図のように，C--言語の構造体は基本型の一次元配列に，構造体配列は二次元配列に似た構造になっている．C--言語の構造体は C 言語とよく似ているが，構造体名が型になることと，変数が参照型である点で異なる．

C--言語の構造体は C 言語では，リスト 5 の 5 行目のような構造体インスタンスと 6 行目のようにポインタ変数により表現される．構造体インスタンスのアドレスを求めるために，アドレス演算子('&')を付加する．

構造体配列の生成では，最初に 7～8 行目のように必要数の構造体インスタンスを宣言する．次に，9 行目のように生成した構造体インスタンスを指すポインタ配列を宣言する．最後に，10 行目のようにそのポインタ配列を指すポインタ変数を宣言する．12 行目のように構造体のメンバーを参照するために，アロー演算子が用いられる．

```
struct S{
  int a;
  char c;
};

S x = {1,'a'};
S[] y = {{1,'a'},{2,'b'}};

public void main(){
  x.a = 3;
  y[1].c = 's';
}
```

リスト 4　構造体を含む C--言語プログラム

```
 1: struct S{
 2:   int a;
 3:   char c;
 4: };
 5: static struct S _cmm_x_cmm_0 = {1,'a'};
 6: static struct S *x = &_cmm_x_cmm0;
 7: static struct S _cmm_y_cmm_0 = {1,'a'};
 8: static struct S _cmm_y_cmm_1 = {2,'b'};
 9: static struct S *_cmm_y_cmm_2[] =
           {&_cmm_y_cmm_0,&_cmm_y_cmm_1};
10: static struct S **y = _cmm_y_cmm_2;
11: void main(){
12:   (x->a=3);
13:   (y[1]->c='s');
14: }
```

リスト5　構造体を含むプログラムを変換した結果

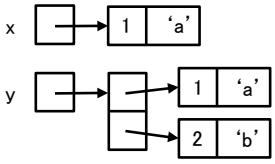

図3　C--言語の構造体の構造

3.3　トランスレータの実行例

リスト6は配列中の最大値を印刷するC--言語プログラムである．これをトランスレータにより変換するとリスト7のようなC言語プログラムになる.C言語の標準関数（printf）は，C--言語の文法でプロトタイプ宣言しておくことで使用可能である.

for文は構文木でwhile文と同様に扱われる.そのため，C言語には8〜13行目のように，while文として出力される.for文の初期化式で宣言された変数の有効範囲を制御するために7〜14行目のfor文全体をブロックで囲んでいる.

C--コンパイラで文字列は，構文解析時にアセンブリ言語のSTRING命令を用いて文字列専用セグメントに出力されていた．トランスレータでも文字列を構文解析部で出力できるように名前付きリテラルとして出力する（17行目）．変換部は名前付きリテラルを参照するC言語プログラム（21行目）を出力する.

12行目のように式には構文木の構造を反映したカッコ付けを行い，構文木と異なる意味の式が出力されないようにする．リスト7のプログラムはC言語コンパイラで正常にコンパイルでき，実行も可能である.

4　トランスレータの改良

開発したトランスレータは，単純なC--言語プログラムをC言語に変換することはできる．しかし，FILE型などのC言語ライブラリに依存する型を宣言する方法がない．そのため実用的に使用することが難しかった.

この問題を解決するために，FILE型などの宣言を可能にすることと，型宣言等を記述したインクルードファイルの扱いに関する改良を行った.

```
public int printf(char[] fmt, ...);

int[] a = {3,7,5,3,1,6,10};

int Max(int[] x, int n){
  int max = 0;
  for(int i=0;i<n;i=i+1)
    if(max<x[i])
      max=x[i];
  return max;
}

public int main(){
  int max = Max(a,7);
  printf("%d\n",max);
  return 0;
}
```

リスト6　最大値を求めるC--言語プログラム

```
 1: int printf(char *fmt,... );
 2: static int _cmm_a_cmm_0[] =
              {3,7,5,3,1,6,10};
 3: static int *a = _cmm_a_cmm_0;
 4: static int Max(int *x,int n){
 5:   int max;
 6:   (max=0);
 7:   {
 8:     int i;
 9:     (i=0);
10:     while ((i<n)){
11:       if ((max<x[i]))(max=x[i]);
12:       (i=(i+1));
13:     }
14:   }
15:   return max;
16: }
17: #define _cmm_str_L0 "%d\n"
18: int main(){
19:   int max;
10:   (max=Max(a,7));
21:   printf(_cmm_str_L0,max);
22:   return 0;
23: }
```

リスト7　最大値を求めるプログラムの変換結果

4.1　騙し型の導入

FILE型などのライブラリで定義される構造体型を使用できるように，新たな文法として騙し型を導入した.騙し型を宣言する構文は以下の通りである.

typedefref 名前;

typedefref構文で宣言した名前は名前表に型名として登録され，プログラム中で参照型として使用できる.

```
typedefref FILE;
public FILE fopen(char[] f, char[] m);
public int main(){
  FILE fp = fopen(“a.txt”,”r”);
  return 0;
}
```

リスト8　騙し型使用例

```
FILE *fopen(char *f,char *m);
#define _cmm_str_L0 “a.txt”
#define _cmm_str_L0 “r”
int main(){
  FILE *fp;
  (fp=fopen(_cmm_str_L0,_cmm_str_L1));
  return 0;
}
```

リスト9　騙し型を使用したプログラムの変換結果

FILE 構造体等のフィールドを C--言語がアクセスできなくても実用上は支障にならないので，名前だけ登録すれば十分である．リスト 8 に騙し型の使用例を，変換結果をリスト 9 に示す．参照型なので，C 言語では型名の後に「*」が付加される．

4.2　インクルードファイル内部の変換出力抑制

C--言語でも，C 言語同様にプリプロセッサが使用できる．プリプロセッサは，ソースコード中の定数名を値に置換する等，前処理を行うプログラムのことである．トランスレータはプリプロセッサにより前処理された C--言語プログラムを入力する

標準関数などの宣言は，リスト 10 のようにインクルードファイルに分離して書くのが一般的である．リスト 10 の C--言語プログラムを GNU の C 言語用プリプロセッサ(cpp)で処理するとリスト 11 のようになる．これをトランスレータで処理すると，リスト 12 のように util.h の内部まで出力されるが，FILE 型の宣言がないのでコンパイルすることはできない．C 言語プログラムでは C 言語用のインクルードファイルを読ませたいので，util.h の内容ではなく C 言語用の#include ディレクティブを出力するべきである．

そこで，トランスレータがリスト 11 のプログラムを読み込んだ際に，1 行目や 5 行目のディレクティブからファイル名を知り，util.h ファイルの内容を出力しないように改良した．改良後の出力をリスト 13 に示す．なお，stdio.h のインクルードは無条件に出力される．

util.h には C--言語で使用できる C 言語の標準関数が宣言されている．util.h はトランスレータ開発者が提供する．ユーザは util.h をインクルードすることで標準関数を使用することができる．

util.h
```
typedefref FILE;
public FILE fopen(char[] f, char[] m);
```

main.cmm
```
#include “util.h”
public int main(){
  FILE fp = fopen(“a.txt”,”r”);
  return 0;
}
```

リスト10　プリプロセッサに入力するファイル

```
1: # 1 “util.h” 1
2: typedefref FILE;
3: public FILE fopen(char[] f, char[] m);
4:
5: # 2 “main.cmm” 2
6: public int main(){
7:   FILE fp = fopen(“a.txt”,”r”);
8:   return 0;
9: }
```

リスト11　プリプロセッサの出力

```
FILE *fopen(char *f,char *m);
#define _cmm_str_L0 “a.txt”
#define _cmm_str_L0 “r”
int main(){
  FILE *fp;
  (fp=fopen(_cmm_str_L0,_cmm_str_L1));
  return 0;
}
```

リスト12　トランスレータの出力

```
#include <stdio.h>
#define _cmm_str_L0 “a.txt”
#define _cmm_str_L0 “r”
int main(){
  FILE *fp;
  (fp=fopen(_cmm_str_L0,_cmm_str_L1));
  return 0;
}
```

リスト13　改良後のトランスレータの出力

4.3　標準関数宣言

リスト 14 に示すように C 言語の標準関数は，util.h 中で C--言語で使用しやすいように，アレンジして宣言している．例えば，C 言語の fgetc 関数は文字の他に EOF も表現するために int 型である．C--言語では int 型と char 型の間で自動型変換がされず，文字コードを int 型の変数に代入すると不便なので char 型にする．

EOF は feof 関数で判別する．feof 関数も C 言語では int 型で宣言されているが，C--言語では boolean 型にする．二つの言語の内部データ表現に注意しながら util.h を作成すれば，C 言語と異なる型で関数を宣言しても支障ない場合が多い．

5　結果

リスト 14 は C 言語の標準ライブラリ関数を用い「a.txt」ファイルの内容を表示する C--言語プログラムである．これをトランスレータで変換した結果はリスト 15 のようになる．リスト 15 のプログラムは，C 言語コンパイラで正常にコンパイルでき実行も可能である．

util.h

```
typedefref FILE;
public FILE fopen(char[] f, char[] m);
public void fclose(FILE fp);
public char fgetc(FILE fp);
public boolean feof(FILE fp);
public int printf(char f, …);
```

main.cmm

```
#include “util.h”
public int main(){
  FILE fp = fopen(“a.txt”, “r”);
  while(!feof(fp)){
    char c = fgetc(fp);
    printf(“%c”, c);
  }
  fclose(fp);
  return 0;
}
```

リスト 14　テストプログラム

```
#include <stdio.h>
#define _cmm_str_L0 "a.txt"
#define _cmm_str_L1 "r"
#define _cmm_str_L2 "%c"
int main(){
  FILE *fp;
  (fp=fopen(_cmm_str_L0,_cmm_str_L1));
  while ((!feof(fp))){
    char c;
    (c=fgetc(fp));
    printf(_cmm_str_L2,c);
  }
  fclose(fp);
  return 0;
}
```

リスト 15　テストプログラムの変換結果

6　まとめ

教育用の習得が容易な C--言語を多くのコンピュータで使用できるように，C--言語から C 言語への変換系を開発した．変換系は C--コンパイラの構文木に改良を加えた上で，コード生成部を C 言語出力用の変換部に置き換えたものである．変換系を用いることで C 言語コンパイラが準備されている多くのコンピュータで，C--言語が使用可能になった．

さらに，C 言語の標準ライブラリ関数が使用できるように C--言語に騙し型を導入した．ライブラリに依存する型は，新しく導入した typedefref 構文を用いて宣言することで C--言語で使用可能になる．これにより C 言語標準関数の大部分が使用できる条件が整い，変換系がより実用的になった．

今後の課題は，より多くの C 言語標準関数を C--言語用のインクルードファイル util.h に宣言することと，C 言語に出力する#include ディレクティブを C--言語のソースコードから制御する方法の検討である．

また今後，配列アクセスに添え字チェックが含まれる C 言語プログラムを出力する改良などを行い，教育用システム記述言語としての完成度を高める．

文献

1）重村他，教育用システム記述言語 C--，教育システム　情報学会．研究報告，Vol.22，No.4，pp.75-80（2007）
2）重村他，機械語教育用マイコン TeC とシリーズ化した教育用パソコン TaC のアーキテクチャ，情報処理学会研究報告，Vol.2012-CE-117，No.9，pp.1-7

（2015.9.25 受理）