5M-5

Semiautomatic Software Documentation Generator

Yasushi Kambayashi, Yoshitake Mishima

Mitsubishi Research Institute

Introduction

Semiautomatic Software Documentation Generator(SSDG) extracts some design information of a software and interface information of each function in the softwre from existing source progrms by using mechanical parsing.

The importnce of software documentation is appreciated. For the most cases of software development contracts, the software documentation is required as a mjor part of supply goods. On the other hand, the real world programmers are driven by irrational dead-lines and kept busy with coding and debugging.

SSDG is developped with hope to mitigate the documentation burden of programmers. We choose the C programming language for the target language for programs from which informations extracted, because the C language is the most frequently used language in our environment. But this choice is arbitrary and our method is able to applied for any other languages.

SSDG tenders three stages to prepare software documentation. First is to define forms used in the documentation. Second is to extract specification information from existing source programs and prepare document file. And the third is to merge these files and print them.

The Form Generator

The form used for software documentation is vary from each other. A software contractor has ten cliants, then, has to prepare ten kinds of them . Therefore, the documentation generator must be able to produce documents with arbitrary formats. We propose a set of grammars which defines a set of forms used for the software documentation, e.g. the form for common variables, the form for function definition and the form for the relationship between functions.

A form is described through drawing rectangles and lines on the A4 size sheet. Character strings can be stated in rectangles for the purpose of literal constants, i.e the title of form or titles for filling area. A rectangle is also able to be specified as an invisible area, so that one can put character strings at arbitrary position on the sheet without any rectangles or lines.

Figure 1 illustrates a form description. A form definition file is consisted of this kind of descriptions so that one file has arbitrary number of forms,then a form definition file can be considered as a form database. Each rectangle has a name and can be the filling area so that document printing facility of SSDG can make correspondence between forms and filling data in the corresponding document file which has the extracted specification information by the name.

This description is parsed and transformed into a stream of the troff commands. The reason why we choose the troff as the basis is that is widely available in the UNIX environment and it clearly separates the source text and formats.

```
*P
Form4
frame4: frame "¥s16";
box31: box height:12 width:87 at:upperLeft "関数仕様";
box32: box height:12 width:87 at:upperRight;
box33: box invisible height:12 width:40 with:topLeft at:box32 topLeft "¥s0 (関数名) ";
!box34: box invisible height:8  width:40 with:leftCentre at:box33 rightCentre;
box34a:box invisible height:6  width:34 with:topLeft at:box31 bottomLeft " (呼び出し形式) ";
box34b:box height:20 width:174 with: topLeft at:box31 bottomLeft;
!box35: box invisible height:14 width:174 with: topLeft at:box34a bottomLeft;
box35a:box invisible height:6  width:20 with:topLeft at:box35 bottomLeft " (機能) ";
!box36: box invisible height:50 width:174 with:topLeft at:box35a bottomLeft;
box37: box height:12 width: 36 with:topLeft at:box36 bottomLeft " (データ型) ";
box38: box height:12 width: 36 with:topLeft at:box37 topRight " (引数名) ";
box39: box height:12 width:102 with:topLeft at:box38 topRight " (説　明) ";
!box40: box height:90 width: 36 with:topLeft at:box37 bottomLeft;
```

Figure1: An example of the form description

The Specification Generator

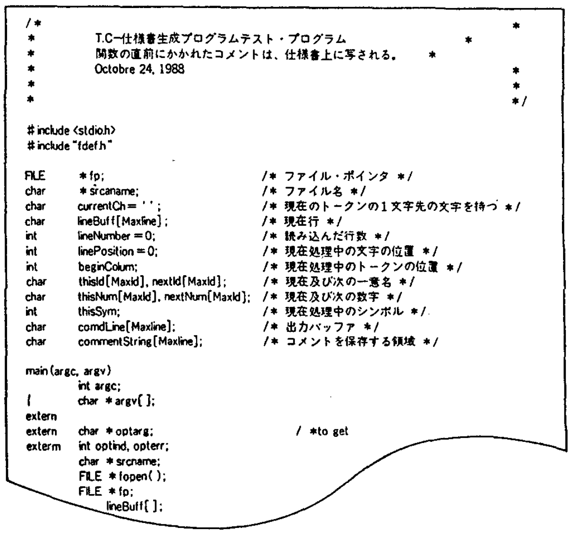
The specification generator extracts software specification information from source program files. The acquired informations are (1) common variables , (2) function definitions, i.e. the name of function, the calling format, names and data types of arguments, return value and names of called functions. They are stored in the document file, so that a user can edit the extracted information by using conventional editor and this document file. We expect a programmer put some information which SSDG could not extract from the source program mechanically. This is because we call our generator "semiautomatic." The printing facility merges this document file and the form definition file into complete document at the later time.

We expect no particular style in the program code or any restriction on the source program to extract specification information. Since SSDG is expected to be applied to existing software, any tiny restrictions or special style will be major obstacles for utilization. Therefore, as long as a program is written syntactically correct, specification information is to be acquired correctly. The only limitation one can regards is the position of the comment for each function. This is extracted as the function explanation, and expected to be stated immediately before the function definition. Describing the comment to explain the function of the function is, however, the convention widely exploited, and considered not to be serious problem. To relax this restriction, the specification generator must have some design knowledge to extract some useful information from arbitrary stated comments in the source program. To put such knowledge would be another burden to programmers. Our purpose is lighten burdens of programmers as much as possible but put nothing to him/her.

The way to format the extracted information into a software documentation is able to be specified by the user as stated before so that same information can be prepared into several different formats and is done at the printing time. For example, from the source program as shown in Figure 2 , SSDG produces documentation shown in Figure 3 automatically.

Conclusion

Since the specification information extracted from existing source program forms the documentation file, and it can be edited by using conventional editor, SSDG's way is said to be flexible. Form generator is not very easy to use now. We intend to utilize the J-STAR workstation to prepare form definition files.



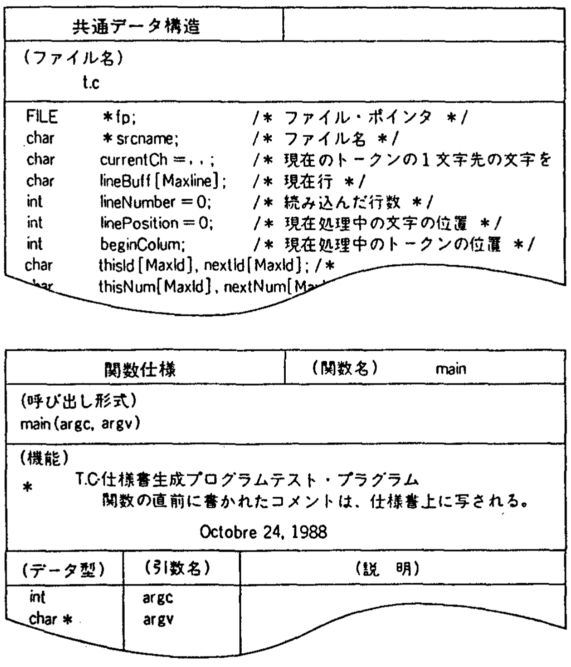
```
/*                                                              *
 *      T.C−仕様書生成プログラムテスト・プログラム                  *
 *      関数の直前にかかれたコメントは、仕様書上に写される。        *
 *      Octobre 24, 1988                                        *
 *                                                              *
 *                                                              */

#include <stdio.h>
#include "fdef.h"

FILE    *fp;                                 /* ファイル・ポインタ */
char    *srcaname;                           /* ファイル名 */
char    currentCh= ' ';                      /* 現在のトークンの1文字先の文字を持つ */
char    lineBuff[Maxline];                   /* 現在行 */
int     lineNumber =0;                       /* 読み込んだ行数 */
int     linePosition =0;                     /* 現在処理中の文字の位置 */
int     beginColum;                          /* 現在処理中のトークンの位置 */
char    thisId[MaxId], nextId[MaxId];        /* 現在及び次の一意名 */
char    thisNum[MaxId], nextNum[MaxId];      /* 現在及び次の数字 */
int     thisSym;                             /* 現在処理中のシンボル */
char    comdLine[Maxline];                   /* 出力バッファ */
char    commentString[Maxline];              /* コメントを保存する領域 */

main (argc, argv)
        int argc;
{       char *argv[ ];
extern
extern  char *optarg;                         / *to get
exterm  int optind, opterr;
        char *srcname;
        FILE *fopen( );
        FILE *fp;
            lineBuff[ ];
```

Figure 2: An example source program



共通データ構造	
(ファイル名) t.c	

```
FILE    *fp;                   /* ファイル・ポインタ */
char    *srcname;              /* ファイル名 */
char    currentCh =. . ;       /* 現在のトークンの1文字先の文字を
char    lineBuff [Maxline];    /* 現在行 */
int     lineNumber = 0;        /* 読み込んだ行数 */
int     linePosition = 0;      /* 現在処理中の文字の位置 */
int     beginColum;            /* 現在処理中のトークンの位置 */
char    thisId [MaxId], nextId [MaxId]; /*
har     thisNum[MaxId], nextNum[M
```

関数仕様	(関数名) main	
(呼び出し形式) main (argc, argv)		
(機能) * T.C仕様書生成プログラムテスト・プラグラム 関数の直前に書かれたコメントは、仕様書上に写される。 Octobre 24, 1988		
(データ型)	(引数名)	(説 明)
int	argc	
char *	argv	

Figure 3: An example of generated document

Reference

Current Topics of Software Techniques in *Journal of Mitubishi Research Institute*, No15, 1988