User's and Installation Guide Portable Simula System based on C

by
Sverre Johansen
Stein Krogdahl
and Terje Mjøs
Department of informatics
University of Oslo

January 1991

1 Introduction

Cim is a compiler for the programming language Simula, as approved (not yet) by the SIMULA Standards Group (except unspecified parameters to formal or virtual procedures). It offers a class concept, separate compilation with full type checking, interface to external C-routines, an application package for process simulation and a coroutine concept. Reference is the book: Kirkerud, Object-oriented programming with SIMULA, Addison Wesley 1989.

Cim is a Simula compiler whose portability is based on the C programming language. The compiler and the run-time system is written in C, and the compiler produces C-code, that is passed to a C-compiler for further processing towards machine code.

2 Installation

The system is distributed as a compressed tar file. Take the following actions to install the software:

1. Uncompress the tar file with the tar command:

   ```
   %uncompress cim.tar.Z
   ```

2. Extract the file with the tar command:

   ```
   %tar xf cim.tar
   ```

3. List the files with the ls command to verify that you have gotten the correct files:

   ```
   %ls
   Makefile        cim.1           cim.h           cim.tar
   cimtest.sim cim libcim.a
   ```

4. Read the make file, and create the needed directories to run make install.

5. Install the system by entering super user and make install:

   ```
   %su
   Password:
   %make install
   %exit
   ```

6. A simple test of the installation can be made by make test:

   ```
   %make test
   cim -r test
   Compiling: test.sim
   cc -w -c test.c
   cc -o testinst test.o -lcim -lm
   Executing testinst:
   Installation: No errors found
   ```

3 A simple example

We show in this section a simple example of a SIMULA program and how to compile and run it.

Create a SIMULA program with a text editor, and give it a name with extension "sim". In this example we name it doesit.sim:

```
begin
    Outtext("Simula does it in C");
    Outimage;
end
```

You can now compile the program with the cim command:

```
%cim doesit
Compiling: doesit.sim
cc -w -c doesit.c
cc -o doesit doesit.o -lcim -lm
%
```

The compiler will produce C code that is further processed towards machine code with a standard C compiler. All produced files have the same name as the input file, but with different extensions. The compiled and linked program can be invoked by entering:

```
%doesit
Simula does it in C
%
```

4 Compiling

Cim is a Simula compiler that first compiles the source code into C. The C code will then be compiled with cc, and linked with other modules.

The Cim command will accept one Simula program and other none Simula modules. The specified Simula program will be compiled and linked with the modules. If a main Simula program is compiled, it will automatic be linked with the necessary Simula modules. If a separate Class or Procedure is compiled, then the linking will be suppressed.

The diagnostics produced by the Simula compiler are intended to be self-explanatory.

The following options are accepted by the cim command:

- -c
  Suppress linking of the complete program.
- -C
  Only link the specified files.
- -cc
  The following argument is the name of the C-compiler.
- -D*name*
  Define a symbol *name*.
- -E
  Run only the preprocessor and output the result to standard output.

- -g
  Make the C compiler produce debugging information. This option is useful for debugging the generated code.

- -gcc
  Invoke the Gnu Project C compiler instead of the standard C compiler. This option can be used if the standard C compiler don't generate correct code.

- -I *dir*
  Use the Simula include file located in directory *dir* instead of the standard directory /usr/local/include.

- -l
  Omit line number information in the compiled program. This will make the program smaller and faster.

- -l*library*
  Link with object library *library*. This option is parsed to the link-command.

- -L*dir*
  Use the Simula library located in directory *dir* instead of the standard directory /usr/local/lib.

- -m
  The memory pool size may be set at runtime by an option -m*n*.

- -m*n*
  Set the initial memory pool size to *n* mega bytes.

- -M*n*
  Set the maximal memory pool size to *n* mega bytes.

- -o
  The following argument is the name of the output executable file.

- -oc
  The following argument will be parsed to the CC-command.

- -ol
  The following argument will be parsed to the link-command.

- -q
  Run the compiler in quiet mode.

- -r
  Run the program after compilation.

- -R
  Recompile the module using the same timestamp.
- -s
  Only C-compile and link the specified files.
- -S
  Run the source file through Simula-compiler, only.
- -t
  Do not remove temporary files. If a main program is compiled with option -r, then the executable file will be removed unless this option or option -T is specified.
- -T
  Do not remove the executable file.
- -U*name*
  Remove any initial definition of the symbol *name* (Inverse of the -D option).
- -v
  Run the compiler in verbose mode.
- -w
  Do not print warnings.

4.1 Arguments

The following arguments are accepted by the Cim command:

- *file*.a
  Library of object files and attribute files. Include this simula library when compiling and linking. The simula library is created with ar(1V) and ranlib(1).
- *file*.o
  Object file of other none Simula modules.
- *file*.sim
  Simula source file. A file name without an extension are assumed to be shorthand notation for the corresponding Simula file.

5 Implementation Aspects

5.1 Language restrictions

A formal or virtual procedure must be specified with respect to its type, and type, kind and transmission mode of its parameters.

5.2 Allowed implementation restrictions

- The type short integer and long real is implemented as integer and real.
- The standard access mode SHARED for files is not implemented.
- The only and default byte size of access mode BYTESIZE is 8.

5.3 Implementation dependent characteristics

- Trailing blanks of image are not transferred to the external file on outfile.outimage excepts it's a direct file.
- A parameter to printfile.spacing with value zero gives the standard effect of overprint.
- The procedures lock and unlock are not implemented.
- All open external files are closed when a program is terminated.
- The following directive lines is supported:
  - `%` *whitespace ...*
    A directive line with a whitespace is treated as a comment line.
  - `%nocomment` ...
    The rest of the line is treated as ordinary source text. Some other simula implementations will ignore this line, and give a warning message. But this can be useful as the following example shows. In this implementation formal procedures must be specified, but that should not be done in standard simula. This will work both on Lund (Simula implementation from Lund Software House AB, Sweden) and Cim:

    ```
    PROCEDURE P(i1,P2);INTEGER i1;
    %nocomment PROCEDURE P2 IS
                  INTEGER PROCEDURE P2
    %nocomment      (i,j);INTEGER i,j;
               ;
    ```
  - `%comment`
    Will cause the compiler to strip all lines until the corresponding %endcomment is reached. This directive may be nested.
  - `%eof`
    Will cause the compiler to react as if the end of the source file was reached. Include files that is placed in a archive must be preceded with this directive line.

- `%casesensitive` ON/OFF
  The case sensitivity of identifiers and keywords is turned ON or OFF. Default value is OFF.
- `%define` *name*
  Define a name. Names such as aix, amigados, convex, cray hp9000s800, hppa, hpux, i286, i386, i486, mach, minix, msdos, mc68000, mc68010, mc68020, mc68030, mc68040, m88000, mips, next, ns32000, sony, sparc, sunos, ultrix, unicos, unix, vax and vms are defined dependent of the system. The name cim is defined for implementations that is generating C code.
- `%error` ...
  Will cause the compiler to believe that it has found an error in the source text. The message that is preceded on the line is printed as an error message.
- `%ifdef` *name*
  If *name* is not defined then the compiler will strip all lines until the corresponding `%else` or `%endif` is reached. If *name* is not defined then the compiler will strip all lines between the optional `%else` and `%endif`.
- `%include` *filename*
  Will cause the compiler to include the indicated file in place of the INCLUDE directive line. This directive may be nested, but only to a level of 10.
- `%nameasvar` ON/OFF
  If it is turned ON, then transmission mode for name is implemented as reference. This will produce more efficient code. Default value is OFF.
- `%staticblock` ON/OFF
  If it is turned on, then data objects will be allocated static instead of dynamic, and the compiler may generate more efficient code. This option should be used with care and should not be used for blocks which may have more than one active data object at a given time. The option may not be used for classes that are given as prefix or virtual procedures or procedures that are parameter to other procedures. It may not be used for external classes or procedures.
- `%stripsideeffects` ON/OFF
  If it is turned ON, then the compiler can generate more efficient code, but not necessary correct code due to evaluation order for expressions. Default value is OFF.
- `%undefine` *name*
  Undefine a name. If the name is not defined the directive line has no effect.

- C is the only language supported for none-Simula external procedures. "Kind" is interpreted as "C", and the *external-item* is case sensitive. External C procedures must be specified in the following way:

  `External C procedure` *external-item* `is` *type* `procedure` *procedure-identifier parameter/mode/specification-part* `;` `;`

  The rules for external C procedures are:

  - Avoid global symbols prefixed with "__", it may lead to conflicts with system names in Cim.
  - The procedure may have any type, except *ref*. If the type is *text*, then the null terminated string returned from C is converted to a Simula text object.
  - Parameters may not be a Simula-procedure, switch or label.
  - Parameters transmitted by value are always copied. Text or arrays are allocated by malloc, and are not deallocated by Cim. It's the C-programs responsibility to dealloc the space.
  - Parameters transmitted by reference or name are transmitted to C as pointer to. Array or text are transmitted to C by the location of the first element.
  - External C procedures with variable number of parameters can be specified by use of "..." in the end of the parameter list. Printf and scanf can be specified as follows:

    ```
    EXTERNAL C PROCEDURE printf IS
       INTEGER PROCEDURE printf(t,...);TEXT t;;
    EXTERNAL C PROCEDURE scanf IS
       INTEGER PROCEDURE scanf(t,...);NAME ...;TEXT t;;
    ```

5.4 Implementation defined characteristics

- The internal character are the same as the standard character set.
- Inlength and outlength are equal to 80.
- SYSIN, SYSOUT and SYSERR is connected to standard input, standard output and standard error. If they are closed and reopened they are connected to `/dev/tty` under UNIX, AIX and MINIX and `sys$input` and `sys$output` under VMS.
- The relative value ranges of real are as double in C and ranges of integer are as long.
- Conversion from an integer type to a real type are exact except for implementations where integer have better precision than real (which is the case for the cray implementation.)

- The effect is not defined if the range of a numeric item in a de-editing procedure exceeds the value range of the procedure result.
- The exponent from "putreal" has 5 characters except for the cray implementation where it may be 6 characters.
- A text frame has a maximum length of about 64K characters.
- The return values of "char" and "rank" are as given by the standard character set.
- The exact definitions of the standard mathematical functions are system specific.
- The association between a file object and an external file are standard procedures based on C's FILE. The object is connected to the external file when open is called.
- Several file objects may represent the same external file, but the effect is not defined if some of them is opened for writing.
- A minimum of checks are performed at "locate".
- The default value to LINES_PER_PAGE is MAXINT.
- The "basic random drawing" is implemented as suggested in the standard.
- Two decimals are used for the field for seconds of the function "datetime".
- Evaluation of arithmetic expressions are based on C, but the Simula expression are by default divided up in several expressions, to guarantee correct evaluation order.

5.5 Capacity limitations

The compiler have the following logical limitations:

- The maximal number of Simula-libraries that the compiler can search is 100 libraries.
- Length of a token in the input stream is restricted to about 1000 characters.
- The nesting of compiler directives is limited to 100.
- The level of nesting of include files must not exceed 10.
- The parser is written i YACC and the parser stack have a size equal 1500 elements.

- Block nesting level is limited to 100.
- The compiler builds an expression tree for each Simula-expression, and one tree is limited to 1000 nodes.
- The code generator have a stack of labels with 1000 elements.
- Temporary expressions may not consist of more than 100 value-type, 100 text, or 100 ref-type elements.
- The nesting of temporary expressions may not be deeper than 100 levels.
- The maximum number of dimensions for arrays is 100.
- Text objects may not contain more than about 64K characters.
- Some other limitations that is based on the underlying hardware or the operating system, and that is not checked by the compiler.

5.6 Extension to the environment

The following procedures is added to the Simula environment and may be called directly from Simula:

- `PROCEDURE Gbc;...;`
  The garbage collector is called when the dynamic storage exceeds an implementation dependent limit. The garbage collector traverse and moves all the accessible objects, and leaves the free space as one area initialized to zero. The garbage collector may be called explicitly through the procedure Gbc.
- `INTEGER PROCEDURE Argc;...;`
  Returns the number of command-line arguments that the program was invoked with.
- `INTEGER PROCEDURE Argv;...;`
  Returns a pointer to an array of character strings (in C fashion) that contains the arguments.
- `PROCEDURE Dump(t);TEXT t;...;`
  Dump the state of the Simula-program to file. Before a call on Dump all files except SYSIN, SYSOUT and SYSERR should be closed.
- `PROCEDURE UnDump(t);TEXT t;...;`
  Read a previously stored state from file and start the program in that state. To get these procedures to work, they should be compiled into the same program. The program may not be re-compiled between a call on Dump and UnDump.

- —tt REF(PrintFile) PROCEDURE SysErr;...;
  Returns the file object associated with standard error.

6 Error report

Errors should be reported to cim-bug@ifi.uio.no.

7 Authors

- Terje Møs, Hydro Data, Oslo.
- Sverre Johansen, Department of Informatics, University of Oslo.
- Stein Krogdahl, Department of Informatics, University of Oslo.