Lisp2C

AutoLISP to C (ADS) treanslator user's guide

Jure Spiler,
Jesenkova 5,
61000 Ljubljana, Slovenia
tel: +386 1 314 069
fax: +386 1 318 211
CompuServe: [70541,1765]
e-mail:
Jure Spiler, director jure.spiler@public1.noprmd.mail.si
Joze Marincek joze.marincek@uni-lj.si

version 1.9 (22-June-1993)

Table of Contents

1. INTRODUCTION

Lisp2Cads translates an AutoLISP source file(s) into C source files that can be further compiled using the Watcom or Metaware C compiler.

Why would anyone bother to compile the existing .LISP code?

First, this completely protects your algorithms. If you use ordinary AutoLISP, you have to provide source that can be read by AutoCAD. But then it can also be read by a human. Therefore all of your know-how is exposed to everyone interested.

Second, an ADS environment is gaining more and more acceptance. Lisp2C is a great way to preserve all your investments into Lisp (trainig, coding) and slowly moving to the ADS.

And finally, Lisp2C also includes a debugging tool that is easy to use yet powerful.

Requirements are:

- AutoCAD R12 (Dos, Windows)
- Watcom C/386 9.0 (9.01d required for Windows), or
- Metaware C/C++ 3.1 and PharLap DOS Extender/Linker

You need DOS4GW.EXE to run L2C.EXE!

Lisp2Cads consists of:

1.	QSTART.TXT	How to quick - start LISP2C
	LISP2C.DOC	This file (Word for Windows)
	LISP2C.TXT	This file (ASCII)
2.	L2C.EXE	Lisp to C compiler
3.	L2C.H	Header file, included into source
4.	L2C.LIB	Libs: (Watcom -DOS)
	WINL2C.LIB	(Watcom - Windows)
	MWL2C.LIB	(Metaware - DOS)
	MWWINL2C.LIB	(Metaware - Windows)
5.	DEMO.LSP	Sample Lisp program
6.	DLINE.LSP	Sample LISP from ACAD12

STARTUP.LSP Direct statements from DLINE
DLINE.L2C Project file to compile DLINE.LSP

Other example files may appear in distribution.

2. INSTALLATION

2.1.DOS - WATCOM

Install your Watcom C/386 9.0 or later compiler and compile at least one sample file (eg TOWER.C) from \ACAD\ADS, to ensure that the compiler is set up properly.

Place the files L2C.EXE (Lips2C translator) and DOS4GW.EXE (Watcom DOS extender) into directory pointed by a system variable PATH. We suggest C:\DOS or C:\ACAD directory.

Place L2C.LIB and L2C.H files into \ACAD\ADS directory. Point to this directory with L2C variable:

SET L2C=C:\ACAD\ADS

Change INCLUDE variable to include \ACAD\ADS directory:

SET INCLUDE=C:\WATCOM\H;C:\ACAD\ADS

2.2.DOS - METAWARE

Install your Metaware HighC/C++ 3.1 compiler and compile at least one sample file (eg TOWER.C) from \ACAD\ADS, to ensure that the compiler is set up properly.

Place the files L2C.EXE (Lips2C translator) and DOS4GW.EXE (Watcom DOS extender) into directory pointed by a system variable PATH. We suggest C:\DOS or C:\ACAD directory.

Place MWL2C.LIB and L2C.H files into \ACAD\ADS directory. Point to this directory with L2C variable:

SET L2C=C:\ACAD\ADS

Change IPATH variable to include \ACAD\ADS directory:

SET IPATH=C:\HIGHC\H;C:\ACAD\ADS
Note that paths must be set **before** Lisp file is converted into C source. L2C uses the values of these variables to produce .BAT and .MW files. If those values are not set or set properly, .BAT and .MW files may not compile the C files. This does not, however, corrupt produced C code in any way.

2.3.WINDOWS - WATCOM

Install your Watcom C/386 9.01d or later compiler and compile at least one sample file (eg TOWER.C) from \ACADWIN\ADS, to ensure that the compiler is set up properly.

Place the files L2C.EXE (Lips2C translator) and DOS4GW.EXE (Watcom DOS extender) into directory pointed by a system variable PATH. We suggest C:\DOS or C:\ACADWIN directory.

Place L2CWIN.LIB and L2C.H files into \ACADWIN\ADS directory. Point to this directory with L2C variable:

SET L2C=C:\ACADWIN\ADS

Change INCLUDE variable to include \ACADWIN\ADS:

SET INCLUDE=C:\WATCOM\H;C:\ACADWIN\ADS

Copy the file ADS.ICO from ADS\WIN directory to your working directory.

2.4.WINDOWS - METAWARE

Install your Metaware HighC/C++ 3.1 compiler and compile at least one sample file (eg TOWER.C) from \ACADWIN\ADS, to ensure that the compiler is set up properly.

Tip: Read README.ADS from ACADWin. Use -NOSTUB switch with PharLap linker 5.0 or later. Earlier versions of PharLap do not need this switch.

Place the files L2C.EXE (Lips2C translator) and DOS4GW.EXE (Watcom DOS extender) into directory pointed by a system variable PATH. We suggest C:\DOS or C:\ACAD directory.

Place MWL2CWIN.LIB and L2C.H files into \ACADWIN\ADS directory. Point to this directory with L2C variable:

SET L2C=C:\ACADWIN\ADS

Change IPATH variable to include \ACADWIN\ADS directory:

SET IPATH=C:\HIGHC\H;C:\ACADWIN\ADS

Note that paths must be set **before** Lisp file is converted into C source. L2C uses the values of these variables to produce .BAT and .MW files. If those values are not set or set properly, .BAT and .MW files may not compile the C files. This does not, however, corrupt produced C code in any way.

Copy the file ADS.ICO from \ACADWIN\ADS\WIN directory to your working directory. Again, this only affects the compilation with produced .BAT and other files.

2.5. DOS Example (Watcom)

Place the rest of the files into your working directory. These files are included only as a demonstration and can be deleted altogether.

Example:

Let us assume that you have correctly set up the Watcom C/386 9.0 to the C:\WATCOM directory. Thus, when you type SET, you might see something like

PATH=...C:\ACAD;C:\WATCOM\BIN;C:\WATCOM\BINB;C:\WATCOM\LIB386\DOS;...
WATCOM=C:\WATCOM\.
INCLUDE=C:\WATCOM\H

In order to use ADS, your C:\ACAD\ADS directory should contain at least the following files:

WCADS90.LIB

ADSLIB.H
ADSDLG.H
ADS.H
ADSCODES.H

If those files are missing, you can copy them from our distribution.

Now you can copy L2C.EXE to C:\ACAD directory (listed in path), and L2C.H, L2C.LIB into C:\ACAD\ADS directory. Also, you should change the INCLUDE variable so that Watcom C/386 will search for header (.h) files also in C:\ACAD\ADS directory:

SET INCLUDE=C:\WATCOM\H;C:\ACAD\ADS

And finally, you should set the L2C system variable to C:\ACAD\ADS:

SET L2C=C:\ACAD\ADS

3. THE USAGE

3.1. COMMAND LINE INPUT

The syntax is:

L2C [options] file [file...]

where options are

d	includes debugging information
oname	sets the output file name to 'name', instead to the name of 1st input file name
c	compiler (cWAT = Watcom, cMW = Metaware)
e	compiles every function separately
y	"yes" to all questions (except for the registration)
n	"no" to all questions (except for the registration)
t	target (currently Dos or WIndows)
?	displays simple help

The option must be preceded with either / or - character. If an invalid option is specified, program terminates with a message.

File is the name of AutoLISP source file. L2C produces file.C file (if more than one file is specified, the first name is taken, unless /oname option is used). In addition, one file is produced for each (defun...) and (lambda...) These additional files have the name of the main file file padded with underscores ("_") to the length of total 8 (eight) characters, and then up to the last three characters are replaced with a number, starting from 0.

If two different applications are compiled on the same directory, then the corresponding main file names must differ in first five characters, or the files of one application will tackle with the files from the other application.

The order of files and options is insignificant. They can also be mixed. Example: to compile DLINE.LSP, one could use the following command:

L2C ai_utils startup /odline dline /d

which would compile AI_UTILS.LSP, STARTUP.LSP, and DLINE.LSP into DLINE.C and DLINE__0.C, ..., DLINE_64.C, DLINE.BAT, DLINE.MAK, and DLINE.LNK. For Windows, also DLINE.RC and DLINE.DEF would be generated.

3.2. INTERACTIVE INPUT

Alternatively, one can invoke L2C without parameters:

L2C
Input file name (.lsp):

The user then enters one file name per line. The input is terminated with a blank line. Then the question appears:

Include debugging information (<Yes>/No/?):

The answer "Yes" is equivalent to specifying the /d switch. The answer "No" is equivalent to omiting that switch.

Next, the target environment and compiler are specified:

Target (<Dos>/Windows/?):

Compiler (<Watcom>/Metaware/?)
Currently supported are DOS and Windows 3.1 operating systems. The code produced is the same in both cases. However, the support files (.BAT, .MAK, .LNK, and optionally .RC and .DEF) files are different for those two environments. Those two questions correspond to /t and /c switches.

Next, the way how produced source code will be compiled, is choosen:

Arrange for separate compilation of each file (Yes/<No>/?):

This question corresponds to the /e switch. The answer "Yes" is equivalent to specifying that switch, and the answer "No" is equivalent to omitting it.

Next, an answer to all questions can be specified:

Answer to all questions (Yes/No/<Ask>/?):

This question corresponds to /y and /n switches. The answer "Yes" is equivalent to specifying the /y switch, the answer "No" is equivalent to specifying the /n switch, and the answer "Ask" is equivalent to omitting both two switches.

Next, the code commenting can be disabled:

Comment the produced code (<Yes>/No/?):

This question corresponds to /b switch. The answer "Yes" is equivalent to specifying that switch and the answer "No" is equivalent to omiting it.

Finally, user can specify the output file name:

Output file name (5 chrs significant) <>:

This question corresponds to an /o switch. In angle brackets, the name of the first file appears as a default output file name. If the default file name is longer than 5 characters, then only the first five characters are in the upper case, and the rest are in the lower case letters. One should be aware, that Lisp2Cads will use only the first 5 characters for all files but the main .C file, the .BAT file, and the .LNK file.

Example: to compile DLINE.LSP, one could use the following command:

```
L2C
Input file name (.LSP): AI_UTILS
Input file name (.LSP): STARTUP
Input file name (.LSP): DLINE
Input file name (.LSP):
Include debugging information (<Yes>/No/?): Yes
Target (<Dos>/Windows/?): Dos.
Compiler (<Watcom>/Metaware/?): Watcom
Output file name (5 char..) <AI_UTils>: DLINE
```

which would compile AI_UTILS.LSP, STARTUP.LSP, and DLINE.LSP into DLINE.C and DLINE__0.C, ..., DLINE_64.C, DLINE.BAT, and DLINE.LNK.

3.3. PROJECT FILE

The third option is to invoke the compiler with the name of the project file, preceded with an @ character. Project file is an ASCII file. Each line is either a comment (starts with an * (asterisk) or ; (semicolon) in the 1st column), a file name, or an option. The syntax for options is the same as in the command line case. A sample project file (with no comments) might look like:

```
ai_utils
startup
dline
/odline
/d
/tWIN
/cWAT
/n
```

The default extension for a project file is .L2C. You can specify a full project file name, if necessary.

Example: to compile DLINE.LSP, one could use the following command:

```
L2C @DLINE
```

which would compile AI_UTILS.LSP, STARTUP.LSP, and DLINE.LSP into DLINE.C and DLINE__0.C, ..., DLINE_64.C, DLINE.BAT, DLINE.MAK, and DLINE.LNK.

3.4. SWITCHES

/D - Debugging

When this switch is used, a code is added to every function that prints out the function name and its arguments, and simple debugger is enabled Also, batch and link files are set to include the debugging information. You will find more information about debugger in Section 6.3.

/Oname - Output filename

You can specify the output file name. If none is specified, the first file name is taken.

/C - Compiler

Currently supported are Watcom C/386 compiler (/CWAT) and MetaWare High C/C++ compiler (/CMW). By default, compiler assumes Watcom C/386. If you use MetaWare C/C++, then use /CMW switch.

/E - Separate compiling

By default, all the functions are included to the main file during compile time. In this way, the C compiler has only to be loaded once, and the compilation process is significantly faster. Using /E switch, you force the L2C to produce several source files; one main source and one source file for every LISP function. They are compiled separately and then linked together. This way, you can edit functions without recompiling all the source code over and over again. Note that this is not simply the question of the batch and link files produced by L2C. The headers of C source files are also different.

/Y - YES to all questions
/N - NO to all questions

Specifies that answers to all the questions are Yes or No, respectively. If you specify both switches, the last is used.

> These two switches do not apply to the question on registration.

/T - target system (DOS or Windows)

The target is the system under which the compiled application will be running. Currently two target systems are supported: MS-DOS and MS-Windows. To choose MS-DOS as a targeting environment, use /TDOS switch. To use MS-Window as a targeting system, use /TWIN switch. The former is default.

The code generated is the same for all the targets. The difference is in the files that Lisp2C produces to compile the application.

/B - brief code generator

Normally, Lisp2C inserts corresponding parts of Lisp code as a comment to the produsd C code. This is intended to ease the code modification process. However, the size of produced .C files is expanded significantly. If you don't intend to modify the code or you don't have enough disk space, specify /b switch to surpress the insertion if the comments.

/? - Help

This switch displays a simple remainder of the switches and stops the compiler execution.

3.5. COMPILING THE CODE (WATCOM)

The C source file must be translated with Watcom C/386 compiler. The object (.OBJ) file produced by Watcom C/386 compiler must be further linked together with WCADS90.LIB and L2C.LIB libraries under DOS, or with WINADS.OBJ, WINADS.LIB and WINL2C.LIB libraries under Windows.

To simplify the job, Lisp2C translator, automatically produces several files, a batch file named name.BAT, make file name.MAK, a link file name.LNK, and possibly simple resource file name.RC and a

simple definition file name.DEF. Batch file invokes the make utility to compile and link all files into an .EXP file under DOS, or .EXE file under Windows. The file file.LNK uses the system variable L2C to locate the libraries. If the user hasn't preset the variable, file.BAT sets it to point to a \ACAD\ADS directory (on the current drive).

If a single file is to be compiled, the command

```
wcc386p /mf /3s /fpi87 <file>
```

can be used for a DOS target. The meanings of the options are

/mf	generate the code for the flat memory model,
/3s	pass the arguments on the stack,
/fpi87	generate in-line calls to a math coprocessor.

Similar command for Windows environment would be

```
wcc386p /mf /3s /fpi87 /s /j /opmaxet /dWIN /dWATWIN /zW
<file>
```

where

/mf	generate the code for the flat memory model,
/3s	pass the arguments on the stack,
/fpi87	generate in-line calls to a math coprocessor,
/s	remove stack overflow checks,
/j	change char default from unsigned to signed,
/opmaxet	controls several optimization parameters,
/dWIN	defines WIN symbol (as with #define),
/dWATWIN	defines WATWIN symbol, and
/zW	uses Microsoft Windows entry/exit code.

Note that, during the compilation, a compiler might issue several warning messages. They refer to undefined macro symbols (used in other systems), unreachable statements, and unreferenced variables. This is perfectly OK, as long as no error is produced.

3.6. LINKING (WATCOM)

To link the compiled files together, it is best to use the generated linker file, as all the file names must be listed. A command

```
wlink @file.lnk
```

should do the trick. However, do not forget the "@" symbol or the file extension!

The resulting file.EXP file is ready to be XLOADed.

3.7. BINDING (WATCOM)

When compiling for Windows environment, the linker produces .REX file. This file has to be further binded with resource and definition files, file..RC and file.DEF, respectively. This can be achieved with the command

```
wbind file -R file.rc file.exe
```

Make sure that the ADS.ICO file is placed in the current directory, You can find that file in \acadwin\ads directory.

Example.

To compile DEMO.LSP the command

```
L2C DEMO
```

produces the following C source and some support files:

DEMO.C	contains the main loop
DEMO__0.C	code for (defun qsort ...)
DEMO__1.C	code for (defun c:stat ...)
DEMO__2.C	code for (defun c:gc ...)
DEMO__3.C	code for (defun c:interpreter ...)
DEMO__4.C	code for (defun S::STARTUP ...)
DEMO.BAT	batch files that starts make utility,
DEMO.MAK	make file,
DEMO.LNK	link file used by DEMO.MAK.
DEMO.RC	is produced under Windows only, a resource file.
DEMO.DEF	is produces under Windows only, a definition file.

Then the command

DEMO

produces (among others) ADS module, DEMO.EXP file , which can be later loaded into AutoCAD with the command

(xload "demo").

Under Windows, DEMO.EXE is produced, which can be XLOADed into AUTOCAD for WIndows.

4. SUPPORTED FUNCTIONS

Lisp2Cads *DEMO* supports AutoLISP R12 functions, except AME and ASE support. Here is the brief list of supported functions:

(* *error* + - / /= 1+ 1- < <= = > >= abs alloc and append apply ascii assoc atan atof atoi atom atoms-family(*) boole boundp caaaar caaadr caaar caadar caaddr caadr caar cadaar cadadr cadar caddar cadddr caddr cadr car cdaaar cdaadr cdaar cdadar cdaddr cdadr cdar cddaar cddadr cddar cdddar cddddr cdddr cddr cdr chr cond cons cos defun eq equal eval exit exp expand expt fix float foreach gc gcd getenv if itoa lambda last length list listp load log logand logior lsh mapcar max mem member min minusp not nth null numberp open or prin1 princ print progn quit quote read read-char read-line rem repeat reverse set setq sin sqrt strcase strcat strlen subst substr terpri trace type untrace ver vmon while write-char write-line zerop ~ acad_colordlg acad_helpdlg acad_strsort ads alert angle angtof angtos command cvunit distance distof entdel entget entlast entmake entmod entnext entsel entupd findfile getangle getcorner getdist getfiled getint getkword getorient getpoint getreal getstring getvar graphscr grclear grdraw grread grtext grvecs handent initget inters menucmd nentsel nentselp osnap polar prompt redraw regapp rtos setvar ssadd ssdel ssget sslength ssmemb ssname tablet tblnext tblsearch textbox textscr trans vports wcmatch xdroom xdsize xload xunload load_dialog unload_dialog new_dialog new_dialog start_dialog done_dialog term_dialog action_tile mode_tile get_attr get_tile set_tile start_list add_list end_list dimx_tile dimy_tile start_image vector_image fill_image slide_image end_image client_data_tile)

5. DIFFERENCES AND LIMITATIONS

5.1. EXPRESSIONS OUTSIDE THE FUNCTIONS

Only Lisp expressions inside Lisp functions are compiled. Other expressions are merely skipped (and a warning message is generated). They can be collected automatically into S::L2CSTARTUP function (into the file ?????__S.LSP).

5.2 NUMBER OF ARGUMENTS

User functions in Lisp2Cads can only have up to 32 arguments if they are ever to be evaluated using EVAL, APPLY or MAPCAR function.

5.3. PASSING SYMBOLS, SUBROUTINES ETC.

Currently, there is no (regular) way to exchange SYMbols, SUBRoutines, EXSUBRoutines and some other exotic data types between an ADS application and an AutoLISP environment. Functions that expect symbols as their arguments (as when calling parameters by reference) should be rewritten in a way that they would accept strings as arguments and then READ out the symbol. This only applies to a function that is called from AutoLISP. Function called directly from another L2C function can pass symbols without any limitations.

5.4. PASSING LISTS OF INTEGERS ETC.

The transfer of the objects between AutoLISP and an ADS application is not an exact one. For example, if the function FOO is invoked with a list of two integers as the only argument: Command: (FOO '(1 2)), then an ADS application will receive this as a 2D point, with integers already converted into reals. As most of the functions that expect real value they work well if given an integer argument, while the opposite might not be true, Lisp2Cads application will transform any whole element of 2D or 3D point passed from AutoLISP to integer. Therefore an unexpected results may occur every now and then.

Example:

AutoLisp value is seen by Lisp2C as

AutoLISP	Lisp2C
(1 2 3)	(1 2 3)
(1.0 2.0 3.0)	(1 2 3)

(1.1 2.2 3.3)	(1.1 2.2. 3.3)
(1 2.2 3.3)	(1 2.2 3.3)
(1 2.0 3.3)	(1 2 3.3)

5.5. BUILD-IN PRIMITIVES

Build-in primitives are considered as a keywords in L2C. You should not use them as an ordinary symbols, or the results will be unpredictable. The only exception is the TYPE function, that returns 'SUBR type on *every* build-in primitive, except on ACAD_COLOR.

> Unfortunately, you cannot check whether an external function has to be loaded by matching its type against EXSUBR type symbol.

5.6. GLOBAL SYMBOLS

When a global symbol is set, also its value in AutoLISP is updated, unless a value contains a non-re presentable datum (such as a symbol or a function). However, once a Lisp2C cannot pass a symbol value to an AutoLISP, it doesn't check that symbol's value in AutoLISP until a Lisp2C application assigns this symbol to a value such that it can be passed to an AutoLISP. (This is necessary as otherwise global symbols would either be number, strings or lists of numbers or strings, or they would evaluate to NIL.) This also gives rise to a simple trick that prevents Lisp2C from constantly updating value of a global symbol in AutoLISP. Assume, for example, that a global variable X must hold an integer value and that we don't want to pass this value to AutoLISP every time X is referenced (this passing mechanism can also be time-consuming). Therefore one would set X to a (dotted) pair (X . <number>) instead simply to a number. In this way Lisp2C will quickly determine it cannot pass a symbol's value to AutoLISP (as already it cannot pass the very first element of a list) and will in turn refuse to pass to or import from AutoLISP the value of X as long as X is stored in this format. And, as usually, the value of X can be obtained using CADR function.

Anyway, it is generally much better practise to avoid using global variables as much as possible.

5.7. A SYMBOL TABLE

A symbol table is used to store names (and values) of the symbols. Currently, symbol table has 631 entries. The hashing algorithm guarantees that at least half of the table will be used before Lisp2C will complain.

5.8. NENTSEL, NENTSELP

Functions NENTSEL and NENTSELP will return a three-element list when the argument is a simple entity, with the third element being an identity matrix.

5.9. ATOMS-FAMILY

It was virtually impossible to reproduce the bug in AutoLISP's (atoms-family ...) function. Therefore (atoms-family ...) returns only the list of the symbols (as a list of symbols or as a list of strings) without "nil's being inserted here and there. Sorry.

5.10. COMMAND

In AutoLISP, (command) evaluates its arguments on-fly. To maintain compatibility, Lisp2C compiles the command

(command arg_1 arg_2 ... arg_n)

into

(command arg_1)
(command arg_2)
...
(command arg_n)

5.12. SPEED

Normally, the program runs faster when compiled with Lisp2C. But the global variables slow down the execution speed significantly. Avoid the usage of global variables wherever possible. Not to mention that global variables as a rule reflect a poor programming style.

6. FEATURES

6.1. A GARBAGE COLLECTOR

The garbage collector not only reclaims a node space, but also releases unused selection sets and closes all unused files. When unloading the application, garbage collection is invoked automatically.

6.2. S::L2CSTARTUP FUNCTION

This function, if present, is invoked automatically during XLOAD. It can already use all functions compiled with Lisp2Cads.

6.3. AUTOMATIC S::L2CSTARTUP GENERATION

When the first statement that is not a function is read, you can select whether this and all subsequent statements should be collected into S::L2CSTARTUP function. You have to be careful, though, not to include (load ...) statements. It is generally better idea to compile the file than to (load) it. You are prompted for each statement whether to include it into S::L2CSTARTUP or not (as long as /N or /Y switches are not used). You can check the ?????__S.LSP function and edit it to meet your needs. Then you should rename it and recompile the program, adding this function to the file list (e.g. in project file), and, during second iteration, you should not generate this file again.

Note: if you already have S::L2CSTARTUP on the disk and L2C generates its own one, the latter will be executed (for it is compiled later).

6.4. SCOPING

Lisp2Cads supports entirely the scoping mechanism from AutoLISP interpreter.

6.5. MEM FUNCTION

Function (MEM) now accepts a single optional argument. If present and not nil, MEM scans the symbol table and prints out the values of all the symbols. Note that symbol can have more then a single value, one in each instance of a function.

6.6. DEBUGGER

The /d switch enables a L2C debugger. This has the following features:

- the program execution can be monitored step by step,
- intermediate results are printed out (optionally),
- at the break point, any Lisp expression can be entered, so the symbols can be viewed and even set.

When the break point is reached, the following message appears:

Function <name>, depth <n>. Command <cmd>.

Here, <name> is the Lisp name of the function that is being executed. <n> is a **depth**, or level of the command that will be executed first.

In the following example, the depth of every function is indicated with the corresponding number:

```
(defun depth ()
  (setq1 a (1+2 (*3 (sin4 x) (cos4 (read5 y)))))
)
```

The <cmd> is the command that will be executed next ("User function", if user function will be called). Then, the following commands are available:

Over/Return/Go/Verbose/lisp expr <1>:

- **Over** will step the execution at the first command having the same or smaller depth as the current command, in the scope of the current function. If a user function is called, another command in that function may have the same depth, but the execution will not be broken. If no other command inside this function has this or smaller depth, the execution will break at the end of the function.
- **Go** will break the execution of the program only when returning to an AutoLisp
- **Return** will break the execution of the program only at the end of the function.
- environment. However, the next time some L2C-compiled function is invoked, the debugger is here again.
- **Verbose** turns on the printing of the commands results.
- **Silent** turns off the printing of the commands results.
- **lisp expr** is any Lisp expression. This string is processed by (read...) and (eval..) pair.
- **<1>** is the number of steps that are to be executed before the program execution is broken again. One (1) means single step, and greater numbers skip several steps. This has to be a non-negative integer.

  To effectively use a debugger, one has to have a printed LISP program (or function) being debugged. There is no line information (currently).

7. A PROGRAMMER'S VIEW

7.1. SymExp

The SymExp .is a basic type in Lisp2C. It is similar in part to a resbuf structure in ADS, but more powerful. In short, it is designed to represent any Lisp data structure. The structure of SymExp can be found in L2C.H file. However, to preserve compatibility with the subsequent releases, you should not access parts of the structure directly, but rather through several procedures, provided for this purpose. These procedures include:

```
SymExp MakeNumber (long);
SymExp MakeReal (ads_real);
SymExp MakeString (char *);
SymExp MakeSymbol (char *);
SymExp MakeFunction (SymExp (*f)());
SymExp MakeFile (FILE *);
SymExp MakeEName (ads_name);
SymExp MakePickSet (ads_name);
SymExp MakePoint (ads_point);
SymExp MakeMatrix (ads_matrix);
```

7.2. A FUNCTION CALL MECHANISM

A call to a user written function

(myfunc arg_1 arg_2 ... arg_n)

is translated as follows.

1. The symbol myfunc is fetched from the symbol list table.
2. If the symbol is a function, then the function is called.
3. If the symbol is a list, it is evaluated.
4. In any other case, myfunc is invoked (via ads_invoke).

There are *n*+1 arguments passed to a function. The last argument is always a EndArgList symbol. This convention is used to allow the functions having variable number of arguments. In AutoLisp one cannot write such a function. However, in C, this is no problem. Via va_start, va_arg, and va_end macros, one fetches one SymExp after anoter, until there is a EndArgList argument.

Example: the following function will act as a (print ...), but accepting as many arguments as supplied.

```
SymExp myfunc (SymExp S,...)
{
   va_list  args;
   SymExp   arg;

   if (S == EndArgList)       // no arguments
      return (NULL);
   LSP_Print (S, EndArgList);  // 1st argument
   va_start (args, S);
   while ((arg = va_arg (args, SymExp)) != EndArgList)
      LSP_Print (S, EndArgList);
   va_end (args);
} // myfunc
```

Note that you have to modify either L2C.H file (*highly unrecommended*), or an aplicattion´s header file, and accordingly either modify one of the source files, or the make file, to compile this function and link it to the application. Further releases of Lisp2C will include additional mechanisms to assist you in creating your own libraries.

7.3. NAMING CONVENTIONS

Wherever possible, LISP symbol names are kept in C source files. To avoid name collision, they are being capitalised (first letter in uppercase and the rest in lowercase letters). In addition, all build-in functions have a prefix LSP_ or ADS_, and might have more letters in uppercase. Special characters (everything other than a letter, digit, or underscore) cannot be handled by C. This characters are replaced with a three-letter code (e.g., a colon (:) is replaced with COL). This guarantees that two different names in LISP will translate into two different names in C, and additionally, no LISP name can match the name of functions in Lisp2C library, as all functions in this library have more than one capital letter, and none contains any three-letter sequence mentioned above.

APPENDIX

A. EXAMPLE: DLINE.LSP

As an example, we shall examine closer the translation of DLINE.LSP file (located in your AutoCAD SUPPORT directory) for AUTOCAD/386. The following steps should be followed:

A number of files will be produced during the compilation, so it is best to do all the compilation in a separate directory. Make sure you can run L2C.EXE program (either place a copy of it in the working directory, or place it into a directory listed in PATH). Make sure that Watcom compiler will be able to locate L2C.H header file (Watcom compiler can locate any header file in current directory or in the directory pointed to by a system INCLUDE variable) and make sure that L2C system variable is pointing to a directory containing both WCADS90 and L2C library files. In general, read section 2 again.

Once you have a working directory, copy the file DLINE.LSP from \ACAD\SUPPORT directory (you might have a different name, and perhaps a drive letter is necessary as well). As stated in section 5.1., lisp expressions that are not inside a function are ignored. As those expressions are usually evaluated during load time, it is important to understand what they are doing. In our case, the expression at the line 204 loads the file containing the support functions. Therefore, we must copy the file AI_UTILS.LSP form \ACAD\SUPPORT directory as well. Look further. At the lines 2121-2124 four global variables are set that enables DLINE command to retain several parameters between two successive calls. The obvious solution would be to copy those four expressions into another file, say DL.LSP, and load that file *before* calling DLINE command. Unfortunately this won't work. The reason is that both dl:snp and dl:brk are set to T, and T is a symbol. According to 5.5 and 5.7., one cannot pass a symbol from AutoLisp to Lisp2Cads. The correct solution is to copy these expressions into a file, say STARTUP.LSP, and enclose them into a function S::L2CSTARTUP[1]. In this way those variables will be set during xload.

Your STARTUP.LSP file should now look something like this:

```
(defun S::L2CSTARTUP ()
 (if (null dl:ecp) (setq dl:ecp 4))
; default to auto endcaps
 (if (null dl:snp) (setq dl:snp T))
; default to snapping ON
 (if (null dl:brk) (setq dl:brk T))
; default to breaking ON
 (if (null dl:osd) (setq dl:osd 0))
; default to center align
)
```

and in your working directory you should have the following files:

```
DLINE.LSP
AI_UTILS.LSP
STARTUP.LSP
```

We could now compile those files with a single command:

```
L2C DLINE AI_UTILS STARTUP
```

The order of the parameters is not important. However, we would place DLINE.LSP file the first, to get DLINE??? file names

---

1 STARTUP.LSP is also supplied with Lisp2Cads, in case you don't fill like typing.

But there is an easier way. Using a project file (supplied with a distribution), one only has to say:

```
L2C @L2C
```

During the compilation, L2C encounters the statements that are not inside any functions. After the first statement, (LOAD ...) function is read, L2C asks

```
Create DLINE__S.LSP with S::L2CSTARTUP? (<Yes>/No)
```

As we already have our own S::L2CSTARTUP function, we answer "No". L2C produces files:

```
DLINE.C          main loop
DLINE__0.C         1st function
DLINE__65.C        last function
DLINE.BAT          this one compiles everything
DLINE.LNK
```

We can now compile and link the application using the command

```
DLINE
```

Watcom C/386 9.0 complains every time for a symbols that are not defined. This is perfectly all right as long as there are 5 warnings per file. Anything more is suspicious and should be reported immediately.

If everything went OK (and it should), you are ready to xload the application. Start AutoCAD and at the Command: prompt type:

```
(xload "dline")
```

Ljubljana, 22th June 1993