AN ABSTRACT OF THE THESIS OF

Mojtaba Mirashrafi for the degree of Master of Science in Computer Science

presented on March 11, 1988.

Title: LIL: A LISP Implementation Language.

Abstract approved: Redacted for Privacy

Michael J. Freiling, Ph.D.

High-level languages provide a convenient environment for program development as well as ease of source code portability. Unfortunately the degree of portability of the programs written in conventional high-level languages change depending on the programming style of the programmer. The Operating System interface also has an effect on the ease of portability.

LIL is a programming language designed to fulfill two goals. The first goal is to provide a high-level language for implementing LISP systems. This is done by providing some special purpose instructions and data structures in LIL which are suitable for LISP interpreter development. The second goal is to increase the portability of the LISP systems written in LIL. This goal is achieved by the fact that LIL programs are not compiled to object code directly, nor is there an interpreter for LIL. All programs written in LIL are translated to some target language, which can be compiled, or assembled using available local tools. The target language may be a high-level language such as C, or assembly language for the target machine.

LIL:
A LISP Implementation Language

by

Mojtaba Mirashrafi

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed March 21, 1988
Commencement June 1988

APPROVED:

Redacted for Privacy

Professor of Computer Science in charge of Major

Redacted for Privacy

Chairman of the Department of Computer Science

Redacted for Privacy

Dean of Graduate School

Date thesis is presented March 11, 1988

Typed by Mojtaba Mirashrafi for Mojtaba Mirashrafi

ACKNOWLEDGMENT

Thank you Dad, for without your support and encouragement I would never have gotten where I am. I would also like to thank my Major professor Dr. Mike Freiling for all the great ideas, and most of all his patience.

My thanks to my Mom, sister, and brother Mori, for being there for me. And thank you J.D.

TABLE OF CONTENTS

LIL: A LISP Implementation Language

1. Introduction

Source code portability is a highly desired feature in computer programs and many person hours are spent porting programs from one system to another. By portability we mean that a source program can be moved from one machine to another with minimum effort to make it run on the new machine. Although most high-level programming languages provide some degree of portability, there are still differences between the compilers that run on different systems. This requires the application programmer to actually modify the source code to make it compatible with the environment to which the program is being transported. Obviously, all the work done on one program will have to be repeated if another program needs to be ported. This is true even when an upgrade on an Operating System takes place. Some system calls may no longer be operational.

The motivation behind developing LIL was to provide a programming language where the source code written in LIL would always be the same regardless of the host environment (i.e., concentrating all the effort of transportation in one place, the translator). This means that the compiler for the language LIL must generate code which runs on the host computer. But writing a complex compiler each time the target machine changes is not a very appetizing idea. Nor is depending on others to write standard compilers. Therefore we must come up with a happy medium. This happy medium should have the following properties:

1. The same source code should be operational at all times.
2. The translator for the language must be easy to implement for a person with modest knowledge of the host environment.
   This criterion leads us to these observations:
   I. The syntax of the language must be very simple.
   II. The translator should not generate object code. It translates the source to some high-level, or assembly-level target language, for possible optimization and final code generation.

III. The constructs of the language, both data and control, must cover the set of constructs which make programming comfortable, yet should be simple enough that they can be translated to a host language with little effort.

IV. The language itself does not provide any run time support routines. Run time support routines normally rely heavily on the execution environment. All run time support modules may be written in LIL. This will make the code highly transportable.

V. A set of standard Operating System interfaces are defined in the language, to provide appropriate linkage for the run time support routines.

1.1 Other portable virtual machine definitions

In order to justify the time and effort spent is developing LIL we must compare it with other 'portables', and point out the differences among them, and LIL. Four systems have been chosen. They are:

1. Pascal P-CODE system [1].
2. University Of Utah's Portable LISP [2] [3].
3. Interlisp's VM [4].
4. Interlisp-D [5] [6] [7]

1.1.1 Pascal P-CODE [1]

The Idea behind the Pascal P-CODE system is to provide a virtual machine so that all Pascal programs are compiled to an instruction set of this machine. The virtual machine provides the data memory, program memory, a set of registers, and the instruction set.

In this system the Pascal code is translated to an intermediate P-CODE language. Then, an interpreter interprets the P-CODE. This makes the Pascal code indeed transportable. This requires the 'new' host machine to have the two other non-portable programs. (i.e., the P-CODE translator, and the interpreter.) Both of these are complex programs. Therefore making the transportation a rather involved process. The instructions have modifiers to work on the different data types available in the system. The data types are: Address, Boolean, Character, Integer (Which includes all user defined scalar types), Real, Set, Multiple word (Only a few instructions can support this data type e.g. string operations), Nil pointer, Set constant. There is no automatic conversion of data types, but

instructions are provided to convert data types. The implementation of the instruction set would be simple yet time consuming, since the data structures are very simple. There are no specific instructions for garbage collection, but instructions to manipulate stack space are available. No segmentation or paging is provided in the system. The only distinction between variable spaces are global variable space and local variable space. Instructions are very simple and *Machine level.* Interpretation of the instructions would be very simple. Also since P-CODE is designed to be the result of translation of Pascal code, Pascal instructions map nicely to P-CODE instructions. The communication between the modules is done via the stack. There is only one calling mechanism: The procedure call, but there are special instructions for calling system routines, and user-defined routines. The unusual instructions are the ones that are taken from Pascal. For example, there are instructions to do 'set' operations. There are also instructions for block compare/move, getting the ordinal value of a non-real value, and instructions for file manipulation. There are no exception handling mechanisms in the P-CODE system.

1.1.2 Utah TLISP [2] [3]

This system also allows writing portable LISP code for mini and micro computers. The TLISP/MTLISP system consists of a hierarchy of virtual levels, where each level of hierarchy offers some level of abstraction in such a way that at top (outer) level, transportable code could be written without any concern about the physical machine environment. As the code descends through the hierarchy the dependency increases. At the bottom (inner) level, the instructions are a set of macros which are translated to a target language.

TLISP is based on a LISP like language called BALM [12]. There are two distinct parts to the TLISP system:

1. A P-code interpreter, garbage collector and I/O system called the MTLISP. This "virtual" LISP-machine supports a number of primitive instructions which can support TLISP. The conventional LISP data types and operations to manipulate the data types

are provided in this machine.

2. This part is a set of compiled TLISP (to MTLISP) which support the LISP like evaluators.

The TLISP interpreter supports these data types and provides instructions to operate on them: Integers, Atoms, Dotted pairs, Strings, Vectors, and Code vectors.

MTLISP objects are tagged objects. The PDP-11's version tag field contains 8 bits, for data type and gc, etc. Since the target machine is an abstract LISP-machine, the data structures can be implemented nicely. There are no special instructions to support the special data structures, but there are macros provided at a higher level to do cons, cdr, etc. The special representation of data types are embedded in the tags, for Garbage Collection, and type checking. The TLISP system contains a garbage collection system in the interpreter. The TLISP system contains a virtual memory page table. Each data type is stored on a different heap for fast access. The instructions for the abstract machine are designed to support LISP, which make the translation easy. The communication may either be done through stacks, or registers. Functions are called via 'standard' function call methods.

1.1.3 The INTERLISP virtual machine. [4]

This package is a sophisticated set of support programs, which provides an environment with support for LISP data types and objects, where a LISP compiler or an interpreter could be written. Provided that the target machine has this 'Virtual Machine', the code written for a LISP interpreter could be ported to that machine.

The virtual machine for the implementation of INTERLISP is a "conventional" LISP environment. All INTERLISP functions are implemented in this environment. The host LISP system and the support facilities for INTERLISP are called the Virtual Machine[4] or VM. The INTERLISP language is implemented in VM, therefore, the INTERLISP may be implemented on any machine which supports VM LISP.

VM provides a very powerful and "complete" set of operations, "P-CODEs" which will allow implementation of INTERLISP in a very simple way. Once the VM is available the INTERLISP definitions which are public domain can be ported to the system. The P-CODEs support:

Logical operations:	such as test for equality of objects, logical "AND", logical "OR", etc.
Data types:	List cells, Literal Atoms, Large and small integers, Strings, Arrays, Stack pointers, Read Tables, Terminal Tables. And a full set of operations on these data types.
Evaluation:	VM provides functions for evaluating expressions, e.g. EVAL, APPLY, COND, PROG, PROGN, etc.
I/O functions:	Functions are provided for opening, closing, reading, and writing files.
Interrupts:	The user is given the ability to interrupt the computation of the VM code by typing certain characters at the terminal. There may be several classes of interrupts associated with several interrupt character codes.

1.1.4 Interlisp-D [5] [6] [7]

Interlisp-D is an implementation of INTERLISP VM for the micro computers, Dolphin and Dorado. The goal of the designers of this system was to implement the VM of the INTERLISP on machines other than large and powerful time sharing computers. The first implementation of Interlisp-D was on the Dorado [13], a microcodable 16 bit computer. The machine was microcoded so it could run the Alto micro computer instructions [14]. The first version of the Interlisp-D was ported directly from Aloto.

After the original port, it was noticed that the system performance could be improved by moving much of the code written in machine language to LISP itself.

As mentioned above INTERLISP is implemented on the VM, the Virtual LISP machine for INTERLISP. The authors of Interlisp-D noticed that the porting of the system was not as easy as they were lead to believe. They observed that the VM specification was very large and not very easy to code. Another observation was that the VM as specified

would be a good starting point to get a first-version-prototype up and running. Then the new system should be tuned to the local environment. In general it felt that the VM was too big, and it is a lot more efficient to make the virtual layer as simple as possible (LIL's goal in life!), and move as much code as possible to the target language (LISP in this case).

1.1.5 Observations

All of the above systems provide source code portability. they do this by encapsulating the source code in an environment with full support of I/O, memory allocation, data types, etc. Basically they provide everything that a **real** machine and its Operating System provide for program development. That is exactly what these systems do: providing a virtual machine. One could write a program which emulates Unix under VMS Operating System, therefore making the Unix code portable to all machines with VMS. Unfortunately, implementation of such complete and powerful virtual machines on each new target machine is very time-consuming!

The LIL approach is not to provide a complete abstract machine environment for program development. Instead, LIL simply provides a programming language which is not at a high-enough-level to make its translation difficult, nor at so low a level as to make it machine-dependent. The non-portable part of LIL is its translator, which can be implemented by using macro definitions in a macro processor, available on most machines as a part of the assembler or high-level language development tools. The main reason for taking this approach is to ease the porting of the translator. It should be noted that in order to create a LIL environment, the implementor does not need to write 'programs'. The existing tools in the host machine are sufficient to create the environment. The most complex 'programming' needed is to represent a LIL statement in some target language, e.g. Pascal or C, or host assembly language. Of course, the code written in LIL may not be as compact and/or elegant as the codes written for some of the above systems.

The spartan simplicity of LIL requires that many "useful" features of more complex virtual machine definitions, such as memory allocation or garbage collection are not supplied and must be implemented in the source code. As mentioned above all these 'useful' features may be kept in a LIL source library to be used by different programs.

2. LIL -- an overview

LIL is a prototype language meant to facilitate writing portable code for all programs in general and interpreters for AI languages like LISP[8] [9] in particular. LIL assumes no particular architecture in terms of underlying software or hardware. There is no need to write a compiler or an interpreter for LIL. LIL can easily be implemented as a set of defined macros. Each LIL statement has an associated macro which expands it into one or more statements of a target language, which might normally be a low-level, machine-dependent assembly language, but could as easily be a high-level language such as Pascal[10] or 'C'[11]. In fact, the first implementation of LIL used 'C' as the target language.

The statements which make up the LIL instruction set are grouped into five categories.

1. Declaration statements
2. Constant statements
3. Definition statements
4. Conditional statements
5. Control statements

In the following sections we will describe the different classes and show some examples of how each class could be used. Wherever possible, we will try to use examples from the CORVALLISP LISP interpreter which has been written in LIL. The complete listing, plus a brief description of CORVALLISP and its available functions are provided later in the paper.

2.1 The LIL virtual machine

Although LIL is not a very complex system it does provide a simple virtual machine environment. The LIL virtual machine has two push-down lists (PDL's). JPDL, the jump push-down list is used for transferring control from one LIL function to another. APDL,

the argument push-down list is used to pass the arguments from one LIL function to another.

Functions are called by using the POPJ instruction. Functions can be scheduled to be called by placing them in the JPDL via JPUSH (push on top of stack) or BJPUSH (put at the bottom of stack, effectively treating the stack as a queue). Another way to invoke a function is to call it via SRJUMP. This instruction is just like normal subroutine calls in other languages, and does not use JPDL. LIL also offers an 'accumulator' which LIL expressions use to return values to LIL instructions.

LIL's statements are divided into two categories.

LIL expressions	These statements are evaluated to a single value (like Pascal functions). Expressions must be used inside statements where their evaluated value may be used. LIL expressions return their value in an accumulator.
LIL instructions	These statements execute and contain expressions, constants, variables, etc. These statements are analogous to Pascal statements and procedures.

Another component of the LIL virtual machine is the 'space' and space management registers. Spaces may be used to implement arrays, free-space for dynamic memory allocation, stacks, and queues. When a space is declared in LIL, some space management registers are created along with the space. These registers are used to keep track of, the top of the space, bottom of the space, and current space pointer. These registers are used by LIL when a push or a pop operation is done on a space or when a LIL free-space allocation, 'mkstr, is performed.

Before we go on, let's look at a simple segment of LIL code.

```
 1 (comment, Define the function)
 2 (fndef,READATOM)
 3 (comment, Set the variable 'counter' to 99)
 4  (set,counter,99)
 5  (tagdef,loopclear) (comment, Define a label)
 6    (comment, store an ascii null in the 'st' field with byte)
 7    (comment, offset 'counter' of the variable pass)
 8    (put,pass,st,counter,(character,0))
 9    (comment, if counter is zero jump to readchars)
10    (jumpz,readchars,counter)
11    (comment, decrement counter and store in counter)
12    (set,counter,(sub,counter,1))
13    (jump,loopclear)
14  (tagdef,readchars)
15    (put,pass,st,0,buff)
16    (set,counter,0)
17    (tagdef,charloop)
18    (comment, read one character from inunit and
19           store in variable buff)
20      (fread,inunit,buff,1)
21   (comment, if buff = C_LPAR then jump to end atom)
22      (jumpif,endatom,(eq,buff,C_LPAR))
23      (jumpif,endatom,(eq,buff,C_RPAR))
24      (jumpif,endatom,(eq,buff,C_BLANK))
25      (jumpif,endatom,(eq,buff,C_NEWLINE))
26      (jumpif,storechar,(le,counter,98))
27    (jump,charloop)
28  (tagdef,storechar)
29      (inc,counter)
30      (put,pass,st,counter,buff)
31      (jump,charloop)
32    (tagdef,endatom)
33      (comment, push a value on argument stack)
34      (apush,pass)
35      (comment, call-return to subroutine intern)
36      (srjump,intern)
37      (comment, pop a value from the argument stack
38            and store in variable result)
39      (set,result,(apop))
40      (comment, transfer control to the function
41            which is on top of the jump stack)
42      (popj)
43 (comment, end of the function)
44 (fnend,readatom)
```

The above code segment is taken from the CORVALLISP interpreter. It is a part of the read function of CORVALLISP. In general the code reads an atom. The job is done by first initializing the atom name to ascii nulls <lines 4-13>. The next step is to read a character. If it is not an atom-terminator and the character count is not greater than 99, then the character is stored as part of the atom name<lines 15-33>. When the atom name is read, it is passed to the call to intern is pushed on the APDL. The result is popped off the APDL and stored in 'result', which is used as a way to return the value to the function which is invoked after READATOM <lines 33-40>. Finally, the control is transferred to the next function on top of the JPDL <line 42>. Line 44 signals the end of the LIL function.

The READATOM function uses the following LIL statements:

APOP	This expression pops a value of the top off the APDL and returns the value popped.
APUSH	This instruction pushes the expression on APDL.
COMMENT	comments.
FNDEF	This instruction defines the beginning of a LIL function.
FNEND	The end of a LIL function.
INC	Increment the value of the given variable.
JUMP	Jump to the given tag.
JUMPIF	This is a conditional jump. If the result of the conditional expression is true the jump to the given tag is taken.
JUMPZ	Also a conditional jump. If the value of the expression is zero the jump is taken.
POPJ	This instruction pops the next LIL function off the JPDL and transfers control to it (last statement executed in a LIL function).
PUT	This instruction stores a value in a filed of a 'structure'
SET	This is the assignment statement in LIL.
SRJUMP	This statement 'calls' a conventional subroutine (like a procedure call in Pascal). The called procedure returns to the caller.
SUB	The subtraction operation.
TAGDEF	Define a tag in the LIL code. The tag is used for conditional and unconditional jumps.

3. Syntax of LIL

LIL is obviously intended to have minimal syntactic complexity. Even constants are surrounded by 'optional' constant declaration expressions. A LIL program consists of a sequence of declarations followed by a series of function statements, conditional statements, and control statements. LIL cannot enforce the above syntax rules and in most cases it does not matter to LIL how things are defined. The following are a set of guidelines for making the LIL code as portable as possible.

3.1 Identifiers

An identifier in LIL consists of a sequence of alphanumeric characters plus '_' where the first character is a letter. The '_' character must be used carefully, since some of the less forgiving languages, e.g. Fortran, do not allow the use of non-alphanumeric characters in identifiers.

Examples

this_is	Valid
a1234	Valid
?foo	Not valid
12x	Not valid

3.2 Statements

A LIL statement must always begin with a LIL open-statement-delimiter, here '(', followed by a LIL keyword followed by the parameters separated by the LIL separator-delimiter, here ',', then the LIL close-statement-delimiter, here ')'. For example:

(id, par1, par2,..., parn)

Where id is a LIL key word and, 'par1' through 'parn' are LIL expression.

If necessary, to permit translation by other macro processors, these delimiters can be changed.

3.3 Declaration statements.

The declaration statements are:

- Comment declaration
- Constant declaration
- Variable declaration
- Space/array declaration
- Structure template declaration

In the following pages LIL expressions are marked as <EXPR>, and LIL instructions are marked as <INST>. Expressions in LIL must evaluate to a valid LIL object.

3.3.1 (*comment*, <text>) <INST>

This statement is used for inserting comments in the LIL source code.

3.3.2 Constant declaration statements <EXPR>

A constant definition expression has the following form:

(<id>,<cons_val>)

Where <id> may be:

- *Integer*, for decimal constants.
- *Real*, for real constants.
- *Octal*, for octal constants.
- *Hex*, for hexadecimal constants.
- *Char*, for character constants.
- *String*, for string constants.

The cons_val is the value of the constant.

Examples:

(integer, 12)
(real, 2.4)
(char, x)
(string, This is a string)

Originally it was decided that all constants must be declared within a constant declaration expression. For numeric type constants such as integers and reals, this is rarely necessary since programming languages recognize these constants, in a nearly universal fashion, whether they are high-level or low-level. Therefore it is optional to declare numeric constants in LIL. Characters and strings are represented differently in different languages, and they must be declared. Other optional expressions are:

1. (*vbl*, <id>) <EXPR>
   This expression indicates that the named identifier is a constant from the variable space.
2. (*space*, <id>) <EXPR>
   The identifier is a space/array identifier.
3. (*struc*, <id>) <EXPR>
   The identifier is name of a structure template.
4. (*field*, <id>) <EXPR>
   identifier is a field of a structure template.
5. (*tag*, <id>) <EXPR>
   The identifier is a defined tag or Label.

3.3.3 Variable declaration statements <INST>

For a LIL source program to run correctly, all LIL variables must be declared. Of course, LIL has no way of enforcing this constraint, because there is no LIL level error checking. Even though LIL might be translated into some language where use of non-declared variables are allowed, it is required that all variables be declared for the simple reason of portability. For example, if we go from a FORTRAN target language which allows use of non-declared variables to a 'C' target language, and we have not included the declarations in the source, we will violate one of the rules that we have defined for LIL: changing the source code. The source code of LIL must never be changed.

Variables in LIL do not have any types associated with them, they are merely a means of storing intermediate-results during execution of the programs. The sizes of these variables should be allocated by the implementation so that the largest data-type allowed, by the host language, can be accommodated.

Examples:

(vdef, x)
(vdef, ptr)

3.3.4 Space/array declaration <INST>

Space, (a physically contiguous sequence of memory locations) is the only data structure supported in the LIL language itself. Space may be used to implement conventional arrays, free-space for generating linked lists, stacks or queues. The size of individual elements of the space may be specified as *byte*, or *word*. A spaces is declared as follows:

(*spacedef*, <id>, <size>, <type>)

Where <id> is the name of the space, <type> is the type of the space element for which its length, in bytes can be determined; <type> may be '(*word*)', or '(*byte*)'. <Size> is the number of elements in the space.

Spaces are one-dimensional. The programmer is responsible for all the mapping from n-dimensional arrays into one-dimension. A set of internal variables (space management registers) are automatically created when a space is defined. These are used by LIL for stack and queue operations, e.g. top-of-space pointer, bottom-of-space pointer. These registers are used when push, pop, mkstr and other instructions which change the 'state' of a space. The programmer must execute an 'initspace' statement for each space which is to be used as free-space or stack/queues. This instruction will initialize the space management registers to their starting values.

Examples:

(spacedef,apdl,1000,(word))
(spacedef,cons_sp,10000,(byte))
(spacedef,(space,print),(integer,100),(byte))
(initspace, (space, cons_sp))

LIL offers commands which support array type access, free-space management and stack/queue operations on spaces.

3.3.5 Structure template declaration statements <INST>

In order to make it possible to build more complex data structures out of the simple *space*, LIL provides a set of instructions for creating templates. These structure template declarations (records for Pascal[10] fans) do not allocate any memory. They can be used as blocks for memory allocation from free-spaces, or they can be used to generate symbolic references to offsets (fields) within memory blocks which are independent of the length of the fields.

A structure template is declared as follows:

```
(strucdef,<id-1>)
  (fielddef,<id-2>, <number-of-elements>, <element-type>)
                .
                .
                .
  (fielddef,<id-n>, <number-of-elements>, <element-type>)
(strucend, <id-1>)
```

Where, <id-1> through <id-n> are the names of the fields, <number-of-elements> indicate the number of elements within the field and <element-type> is the type of each individual element which may be '(*bit*)', '(*byte*)' or '(*word*)'. [Only byte and word are currently implemented.]

Examples:

```
1)
 (strucdef, generic_header)
    (fielddef, cell_type, 1, (byte))
    (fielddef, gc, 1, (bit))
 (strucend, generic_header)

2)
 (strucdef, cons_cell)
    (fielddef, type_cc, 1, (byte))
    (fielddef, gc_cc, 1, (byte))
    (fielddef, car_pointer, 1, (word))
    (fielddef, cdr_pointer, 1, (word))
 (strucend, cons_cell)

3)
 (strucdef,atom_header)
    (fielddef,type_ah,1,(byte))
    (fielddef,gc_ah, 1, (byte))
```

```
  (fielddef,vcell,1,(word))
  (fielddef,plist,1,(word))
  (fielddef,pname,1,(word))
(strucend,atom_header)
```

All the field names and structure names must be unique. If a field name is used it can only be within one structure def. This permits symbolic accesses to expand unambiguously into an offset.

It should be noted that blocks from a *space* are not allocated with respect to any particular type. When an allocation is requested for some structure, only the size of the structure is relevant. After a structure has been allocated, it may be accessed with respect to *any* structure template. In this fashion, variant records can be supported as in Pascal [10] even though field names must be unique to a particular structure definition.

For example ...

Consider these two structure definitions:

```
(strucdef, cons_cell)
  (fielddef,cell_type,1,(byte))
  (fielddef,gc,1,(byte))
  (fielddef,car_pointer,1,(word))
  (fielddef,cdr_pointer,1,(word))
(strucend,cons_cell)

(strucdef,atom_header)
  (fielddef,comm_1,2,(byte))
  (fielddef,vcell,1,(word))
  (fielddef,plist,1,(word))
  (fielddef,pname,1,(word))
(strucend,atom_header)
```

And the code segment:

```
(set, foo, (get, data_ptr, cell_type, 0))
(jumpif, cons_type, (eq, foo, CONS_CELL))
(jumpif, atom_type, (eq, foo, ATOM_HEADER))
(jump, error)
(tagdef, cons_type)
    (set, name, (get, data_ptr, cdr_ptr, 0))
    ...
    ...
(tagdef, atom_type)
    (set, name, (get, data_ptr, plist, 0))
    ...
```

...

The above code uses the *cell_type* to access the first byte of the generic structure pointed to by *data_ptr*. Then based on the value found in the type field the appropriate fields are accessed.

No type checking is performed to determine the applicability of any template, even if it is too large.

3.4 Definition statements

In LIL, one can use definition statements to associate a name with some actions or values. For example one can define a symbol to have a constant value (symbolic constants), or define a symbol to represent a sequence of actions (functions and procedures).

3.4.1 Constant definitions <INST>

A symbolic constant is defined as follows:

(*cdef*, <id>, <val>)

Where, <id> is the name and <val> is the value to be used where identifier appears. Constant definitions associate a symbolic name to a constant. Here are some examples:

```
(cdef,C_INUNIT,(CONSTANT,0))
(cdef,C_OUTUNIT,1)
(cdef,C_EXIT_FLAG,-1)
(cdef,C_ATOM_HEDER,1)
```

3.4.2 Tag definition statements <INST>

These statements define a labeled location in the modules. Control may be transferred to tags via jump statements..

(*tagdef*, <id>)

Where identifier is the name of the tag.

Examples:

```
(tagdef, foo)
(tagdef, jump_here)
```

3.4.3 Module definition statements <INST>

A module in LIL is a sequence of control and conditional statements which define a set of actions to take place when the module is invoked. There are two different types of modules available in LIL:

1. Threaded modules (scheduled to be called)
2. Subroutines (immediately called)

The only difference between threaded modules and subroutines is the way they are invoked. The threaded modules or functions as they are called in LIL are invoked through a special stack, JPDL (Jump Push down List) and its associated LIL operators JPUSH, BJPUSH and POPJ. The subroutines are the conventional subroutines which map to the host language subroutines. Subroutines are invoked directly and not by placing the code address in the LIL JPDL. There may be no recursive calls in subroutines. Neither functions nor subroutines have any parameters. Information can be shared through the global variables or through a special stack called the APDL (Argument Push Down List).

3.4.3.1 Threaded modules

Threaded modules are defined as follows:

```
(fndef, <id>)
  <expr 1>
    .
    .
    .
  <expr n>
(fnend, <id>)
```

Where identifier is the name of the function and expr 1 ... expr n are control statements.

3.4.3.1.1 The jump push-down list (JPDL)

The jump stack is a special-purpose stack which is used in LIL to transfer control to functions. If the user wishes to use this stack he/she must declare and initialize a space called JPDL. Here is how JPDL is used:

Whenever the program decides that a particular action should take place it pushes the function name which performs the task on the JPDL. This action by itself does not invoke the function, it merely schedules the function to be executed in the future. At this point the scheduling module still has the control of execution and may schedule other functions to be executed. A function can be pushed on top of the JPDL by (*JPUSH*, <fn>) operation or queued at the bottom of the stack by (*BJPUSH*, <fn>) operation. When the function which has the control of the flow performs a (*POPJ*) instruction the control is transferred to the function which is on top of JPDL. This will cause the control to be given to the new function and the function is popped off the JPDL. Note that POPJ is not a conventional subroutine call. Once POPJ is performed, control will not automatically return to the function, unless its own address has been placed on the JPDL, either by itself or by another function. This feature is useful to implement recursion and also tail-recursive optimization of recursive functions. In the example below LISP_1 pushes its own address on the JPDL and then schedules some other tasks to be performed. LISP_1 writes a prompt and transfers control via JPDL to the last function which it had pushed on JPDL. Eventually some function will perform a POPJ which transfers control back to LISP_1.

3.4.3.1.2 The argument push-down list (APDL)

Since modules may not have arguments, the APDL is used to pass information to modules in LIL. This space must also be defined and initialized by the user if it is to be used. APUSH, and APOP are special case push and pop instructions which operate on the APDL

Examples:

The following LIL module implements a standard read eval print loop. The module first pushes itself (LISP_1) on the JPDL (schedules itself to be executed later). Then in reverse order it pushes the functions which should be executed. The JPUSH's have the following effect:

1. Read an s-expr. The result is stored in a variable called RESULT.
2. Push the result of the read on the argument stack.
3. EVALuated the s-expr (EVAL_0 assumes it argument is on stack.)
4. Push the result of the EVAL on apdl.
5. Print the result.
6. Go back to LISP_1.

```
(fndef,LISP_1)
  (jpush,LISP_1)
  (jpush,PRINT_1)
  (jpush,PUSH_RESULT_ON_APDL)
  (jpush,EVAL_0)
  (jpush,PUSH_RESULT_ON_APDL)
  (jpush,READ_0)
  (fwrite,C_OUTUNIT,(string,0>>>>),lisp_level)
  (comment, Call the mkcons subroutine to make a cons-cell)
  (srjump,mkcons)
  (set,gc_hold,(apop))
  (popj)
(fnend,lisp_1)


(fndef,PUSH_RESULT_ON_APDL)
  (apush,result)
  (popj)
(fnend,push_result)
```

3.4.3.2 Subroutines

Subroutines are defined as follows:

```
(srdef, <id>)
   <expr 1>
     .
     .
     .
   <expr n>
(srend, <id>)
```

Where identifier is the name of the subroutine and expr's are LIL control or conditional statements

Examples:

cons_1 is a subroutine which takes two arguments from the APDL. Makes a cons-cell. Then sets the CAR and CDR of the new cons-cell equal to the arguments.

```
(srdef,cons_1)

   (set,arg2,(apop))
   (set,arg1,(apop))
   (srjump,mkcons)
   (set,result,(apop))
   (comment, Set the first entry in
           the car_pointer field of
           variable result to arg1
   )
   (put,result,car_pointer,0,arg1)
   (put,result,cdr_pointer,0,arg2)
   (ret)
(srend,cons_1)
```

Hasher gets its argument from APDL and calculates a hash value for it by adding all the bytes in the string mod size of OBLIST. It returns the result via APDL. This routine uses the optional enclosing brackets vbl, integer, space, etc.

```
(srdef,hasher)

(set,(vbl,counter),(integer,0))
(set,(vbl,argu),(pop,(space,apdl)))
(scopy,
      (get,(vbl,argu),(field,st),(integer,0)),
       (aget,(space,print),(integer,0)))
(set,(vbl,hashval),(integer,0))
(tagdef,hloop)
(jumpif,out,
   (eq,(aget,(space,print),(vbl,counter)),(character, )))
(set,(vbl,hashval),
    (add,(vbl,hashval),(aget,(space,print),(vbl,counter))))
(set,(vbl,counter),(add,(vbl,counter),(integer,1)))
(jump,hloop)
(tagdef,out)
(set,(vbl,hashval),(mod,(vbl,hashval),(const,1000)))
(inc,counter)
(apush,counter)
(push,(space,apdl),(vbl,hashval))
(ret)
(srend,hasher)
```

3.5 Conditional statements <EXPR>

Conditional statements are used to compare the value of two LIL statements and return true or false. These statements are used in conditional jumps. Their syntax:

(cond, <expr_1>, <expr_2>)

Where *cond* is one of the following:

1. eq: equal to?
2. ne: not equal to?
3. le: less than or equal?
4. lt: less than?
5. ge: greater or equal?
6. gt: greater that?

Expr's must evaluate to a single value.

Examples:

(eq, x, y)
(ne, man, (add, x, y))

3.6 Control statements

Control statements are statements which use the declarations and the definitions to perform the desired task. The control statements are as follow:

3.6.1 (*codebegin*, <fn>) <INST>

This statement starts the executions of the LIL program by transferring control to function fn. It also allows the implementor to perform any kind of initialization which is needed. Only the last *codebegin* in a LIL program will have any effect.

Example:

(*codebegin*, LISP)

3.6.2 (*codeend*, <label>) <INST>

The end of the LIL code. This statement is provided so that the code for the target language can be 'ended' in a clean fashion. Most compilers and assembler need to know where the end of the code is.

Example:

(*codeend*, LISP)

3.6.3 (*loop*, <label>) <INST>

This statement marks the beginning of an infinite loop. To exit this loop one should execute a jump statement. The end of the loop is marked by (*loopend*,_<label>) statement.

Example:

```
(loop, again)
    .
    .
    .
(loopend, again)
```

3.6.4 (*loopend*, <label>) <INST>

The ending bracket of (*loop*, <label>) statement. The loopend statement must be preceded by a loop statement with the same label name. The loop-loopend expressions are put in the language to make the structure of a loop more obvious to the reader. One could use the unconditional jump statements to create the same effect, but the jump by itself does not depict a loop in the program. By using this looping construct and the conditional jumps any 'structured' looping construct such as while loops or repeat-until loops can be implemented. Notice that a loop_end is translated to a goto, so if there is no loop label defined by a loop statement somewhere else in the code, an error will be detected by the compiler or the assembler of the target language. Another point to keep in mind is that if a loopend statement appears in the code before the loop statement, it would act only as a 'goto' the loop statement and no looping will result. It is possible to define more than one loopend statement for a single loop statement with the same label, but that is not good programming practice.

3.6.5 (*srjump*, <id>) <INST>

This statement does a return-jump to the indicated subroutine. The subroutine must execute a (*ret*) instruction in order to return the control to the calling module. A

subroutine should never do an srjump to itself (recursive calls). This construct is only meant to support non-recursive calls to simple utility subroutines. More general recursive function calling schemes should be implemented via PUSHJ/POPJ.

Example:

(srjump, subroutine)

3.6.6 (*ret*) <INST>

The means to return the control to the module which has called the subroutine which is executing this statement. This statement must only be used in a subroutine. Otherwise the result of this operation is undefined. All subroutines must use this statement to exit. There is no implied return at the end of the subroutine. A *ret* in LIL is translated to a target language return-from-subroutine statement.

Example:

```
(srdef, zip)
  ...
  ...
  (set, result, xx)
(ret)
```

3.6.7 (*call*, <os_routine>, <par1>, <par2>, ..., <parn>) <INST>

This statement is used to invoke Operating System interface modules such as open file, close file, etc. These calls are distinguished from other 'module' calls to allow direct parameter passing ability to the interfaces. The reason for distinguishing this set of calls from calls to LIL code is that many assembly languages require separate operations to be invoked for Operating System routines. Thus a separate LIL instruction name is used. See the section on the Operating System interface, for a description of what OS routines have been defined for current implementation of LIL.

In the current version of LIL we have implemented direct statements in the LIL language. e.g., there is a LIL *fopen* statement.

Example:

(call, write, a, b, c)

3.7 Stack manipulation instructions

ALL the stack/queue operations work in a circular fashion i.e., the stack will never underflow or overflow. If we reach the top of the stack by doing extra pushes, the top-of-stack pointer will wrap around to the bottom of the stack. The only exception is when top of stack pointer collides with the bottom of stack pointer.

3.7.1 (*push*, <id>, <expr>) <INST>

This statement pushes the value of expr on top of the space which is denoted by <id>. <ID> must be declared as a space/array variable and initialized via the *initspace* statement. Pushing values on the stack moves top of stack up and bpush-ing (see next section) moves bottom of stack down.

Example:

(push, apdl, result) Pushes the value on the result on apdl.

3.7.2 (*bpush*, <id>, <expr>) <INST>

The bpush statement pushes the value of the statement, (queues it) on the bottom of the stack.

3.7.3 (*pop*, <id>) <EXPR>

Pop will pop a value off the top of the stack and returns the value. Normally this statement should be used within another statement which captures and uses the value returned. If the stack pointer is pointing to the physical top of the space the top of stack pointer will wrap around and return the value which had been bpush'ed on the stack.

Example:

(set, result, (pop, apdl)) Pops a value from apdl.

3.7.4 (*jpush*, <id>) <INST>

This is a special-purpose push operation which pushes the function denoted by the identifier on the JPDL. Note that JPDL must be declared and initialized by the user, otherwise errors will occur. This statement is used to schedule a function to be executed at a later time.

NOTE

JPOP is quite different from POPJ. POPJ causes a transfer of control, while JPOP only removes an element from the JPDL.

Example:

(jpush, EVAL) Pushes the address of EVAL on the jpdl.

3.7.5 (*apush*, <expr>) <INST>

This is another special form of the push statement which pushes values on the APDL. APDL must be declared and initialized by the user.

Example:

(apush, result) Pushes the value of result on the apdl.

3.7.6 (*jpop*) <EXPR>

JPOP is the special purpose pop for the JPDL. It works exactly like pop except the control does not have to be given to the statement as a parameter.

Example:

(jpop) Pop a value from the jpdl

3.7.7 (*apop*) <EXPR>

Another special purpose pop operation which automatically pops a value from the APDL and returns its value.

Example:

(set, result, (apop))

3.7.8 (*popj*) <INST>

POPJ is not merely another pop operation from the top of the JPDL. It is the means to transfer control from one function to another. The function which executes a POPJ is asking the next function which has been scheduled to run to be executed. In current version of LIL POPJ simply pops the value of the next function off JPDL and 'jumps' to it. POPJ could become much smarter and actually play the role of a scheduler or dispatcher by making some decisions as to who should be run next i.e., assign other priorities to functions. Another function which could be performed by a more sophisticated POPJ is error checking on the function invocations.

Example:

```
(push, jpdl, EVAL)
(popj)                  Transfer control to EVAL
```

3.8 Arithmetic Instructions

3.8.1 (*set*, <id>, <expr>) <INST>

This is the equivalent of an assignment statement in other languages. The value of expr is assigned to identifier. Note that identifier must be a declared variable.

3.8.2 (*add*, <expr>, <expr>) <INST>

The value of the two expressions (*integer*) are added and result is returned.

Example:

```
(set, new, (add, 1, 2))
```

3.8.3 (*inc*, <id>) <INST>

Adds one to the value of the identifier, and sets the identifier to the new value.

Example:

```
(inc, value)
```

3.8.4 (*sub*, <expr>, <expr>) <INST>

The value of the second expression is subtracted from the value of the first expression and the result is returned.

Example:

(set, new, (sub, 1, 2))

3.8.5 (*dec*, <id>) <INST>

Subtracts one from the value of identifier, and sets the identifier to the new value.

Example:

(dec, result)

3.8.6 (*mul*, <expr>, <expr>) <INST>

The the two expressions are multiplied and the result is returned.

Example:

(set, new, (mul, i, 2))

3.8.7 (*div*, <expr>, <expr>) <INST>

The value of the first expression is divided by the value of the second expression and an integer result is returned.

Example:

(set, new, (div, i, 2))

3.8.8 (*mod*, <expr>, <expr>) <INST>

The value of the first expression is divided by the value of the second expression and the integer remainder is returned.

Example:

(set, new, (mod, i, 2))

3.9 Storage management Instructions

3.9.1 (*mkstr*, <structure>, <id>) <EXPR>

This expression is an extremely simple memory allocation instruction. It returns a pointer to a block of memory from the space <id> which can hold the given structure. In the current version of LIL, space is allocated sequentially. The allocation starts at the top of the space and continues down. Keep in mind that this expression is very different from 'allocate' system calls. The memory allocation is totally local to LIL, and within the static 'space' declared in the LIL code.

Example:

```
(set, variable, (mkstr, (struc, atom_header), atom_space))
```

3.9.2 (*checkspace*, <structure>, <id>) <EXPR>

This expression can be used to see if there is enough room in the specified space <id> to accommodate the structure. If this expression returns, false then *mkstr* should not be used to allocate memory for the structure.

Example:

```
(comment, Check the space 'array' FREE_SPACE to see if
        there is enough memory left for a structure
        with the size of ATOM_HEADER)
(set, have_space, (checkspace, ATOM_HEADER, FREE_SPACE))

(comment, If there is room then jump to code which
        allocates the memory.)
(jumpif, ok, (eq, have_space, TRUE))
.
.
.
(tagdef, ok)

(comment, From FREE_SPACE allocate enough memory
          for a structure of the type atom_header.
          (Done by mkstr).
          Then set the variable new_hdr equal to
           where that memory starts.(Done by set.)

(set, new_hdr, (mkstr, atom_header, FREE_SPACE))
```

3.9.3 (*initspace*, <id>) <INST>

All the spaces used for free-memory and stacks must be initialized by initspace. This statement initializes all the space management registers which are needed to keep track of top of stack, bottom of stack, etc. Normally this statement should be used once at the beginning of a program. However if one wishes to reset a space to its original state, this statement could be used. For example, in a stop and copy garbage collection method, after the copy, the old space may be initializes using this statement.

Example:

(initspace, cons_space)

3.9.4 (*vinit*, <id>) <INST>

This is an LISP specific instruction. It *interns* the variable <id> and sets its value to nil.

Example:

(vinit, foo-atom)

3.10 Data manipulation Instructions

3.10.1 (*get*, <id>, <field>, <offset>) <EXPR>

The *get* expression accesses fields within structures. The <id> points to some memory location in a space. <Field> is the secondary offset from the beginning of the structure, and <offset> is an offset within the field. In general, fields are assumed to be arrays of memory. Therefore, if the field consists of a single value, an offset of Zero should be used.

Example:

The expression
(get, cons_cell, car_ptr, 0)
is equivalent of
cons_cell.car_ptr
in Pascal.

3.10.2 (*put*, <id>, <field>, <offset>, <expr>) <INST>

The put statement stores the value of the <expr> in the structure pointed to by <id> in field <field> with offset of <offset>.

Example:

(put, atom_header, pname, 0, (character, a))

3.10.3 (*aget*, <id>, <offset>) <EXPR>

Aget retrieves a value from a space which is being used as an array. <ID> is the name of the space and the <offset> is the entry number in the array.

Example:

(set, bucket_ptr, (aget, oblist, hash_val))

3.10.4 (*aput*, <id>, <offset>, <expr>) <INST>

Aput is used to 'put' values in arrays. The value of <expr> is stored in <id> at location offset.

Example:

(aput, oblist, 10, foo)

3.11 Flow control Instructions

3.11.1 (*jump*, <label>) <INST>

Jumps to the defined <label>. The <label> must be defined within the module.

3.11.2 (*jumpif*, <label>, <cond>) <INST>

This is a conditional jump statement. <Cond> must be a conditional statement, which is evaluated and if the result is true, control is transferred to the first expression after the <label>. The <label> must be defined within the current module.

Example:

(jumpif, end_list, (eq, (get, l_ptr, cdr_ptr, 0), nil))

3.11.3 (*jumpnif*, <label>, <cond>) <INST>

Jumpnif is also a conditional jump statement except that the jump occurs if the value of the conditional expression evaluates to false.

Example:

(jumpnif, not_end_of_list, (eq,(get, l_ptr, cdr_ptr, 0), nil))

3.11.4 (*jumpz*, <label>, <expr>) <INST>

Control is transferred to the statement after the label if the result of the expr is Zero.

3.11.5 (*jumpnz*, <label>, <expr>) <INST>

Jump to the label if the value of the expression is not zero.

3.12 Miscellaneous Instructions

3.12.1 (*vset*, <id>, <EXPR>) <INST>

This statement assigns the value of <EXPR> to the vcell of atom <id>.

Example:

(vset, foo-atom, (apop))

3.12.2 (*vget*, <id>) <EXPR>

This expression evaluates to the vcell of the atom <id>.

Example:

(apush (vget, foo-atom))

3.12.3 (*scopy*, <string1>, <string2>) <INST>

This statement copies the contents of space <string1> to <string2> until it finds an ascii '0'. There is no bound checking. So the programmer must make sure that <string2> has room.

Example:

(scopy,(aget,pnamesp,ploc),(get,argument,string,0))

```
This is the same as:
 i := ploc;
 j := 0;
 repeat
   argument.string[j] = pnamesp[i];
 until (pnamesp[i] != 0);
```

3.12.4 (*scomp*, <string1>, <string2>) <EXPR>

This expression compares the two strings and returns -1 if <string1> is less that <string2>. 0 if they are equal, and 1 if <string1> is greater than <string2>. The strings must end with ascii '0'.

Example:

(scomp, (string, this is), (aget, name, 0))

```
Where the first argument is a string
constant and the second argument is
the first character in the array name.
(Address of name[0] is used as
 the starting location.)
```

3.12.5 (*scat*, <string1>, <string2>) <EXPR>

This statement concatenates <string1> to <string2>. It is assumed that there is enough space in <string2> for <string1>.

Example:

(scat, (aget, name, 0), (string, this is))

3.13 Operating System interfaces

A set of Operating System interfaces are defined in LIL for file manipulation. The routines are:

3.13.1 (*fopen*, <id>, <string>) <INST>

This statement will try to open a file with the name pointed to by <string> and returns a connection to <id>. A negative value indicates an error.

Example:

(fopen,arg1,(aget,pnamesp,ploc))

3.13.2 (*fclose*, <id>) <INST>

Close the file with connection <id>.

Example:

(fclose, arg1)

3.13.3 (*fread*, <connection>, <id>, <number-of-characters>) <EXPR>

Read <number-of-characters> number of characters from the <connection> and store them starting from the location pointed to by <id>. The expression should return the actual number of characters which it read as its value.

Example:

(set, numbers, (fread, inunit, p_name, 12))

3.13.4 (*fwrite*, <connection>, <id>, <number-of-characters>) <INST>

Write <number-of-characters> number of characters to the <connection> file. Get the characters from location pointed to by <id>.

Example:

(fwrite, C_OUTUNIT, (string, CORVALLISP),11)

4. CORVALLISP interpreter

In this section a description of the CORVALLISP interpreter which was written entirely in LIL is presented as a working example for this implementation of LIL.

In the final two appendixes, you will find examples of CORVALLISP functions and a sample run session with CORVALLISP.

4.1 Special features of CORVALLISP

We have taken a new approach for implementing the EVAL in CORVALLISP. EVAL is divided into three distinct segments:

1. Argument handler
2. Body handler
3. Result handler

The argument handler decides what should be done with arguments of a function. e.g whether the arguments should be evaluated, or not.

The Body handler's task is to decide how the body of the function is to be executed. Binding of variables is also done by the body handler.

The result handler takes care of cleaning the stack and restoring the environment.

In the current implementation of CORVALLISP, there is only one handler for each of the above categories. The handlers 'look' at the type of the atoms and based on the type, the method of 'handling' is determined. It would be more appropriate to have the atom-header contain the pointers to its own handlers. With this method the processes of evaluation will be reduced to only 'calling' the three handlers which are specified in the atom-header. There would be no decision making in the EVAL. Some standard handlers could be pre-defined in LIL for conventional function types, but the user can define new function types by defining his/her own handlers.

4.2 Future work

The idea of handlers may be implemented via a special kind of expression which we call 'Z-lambda'. The Z-lambda expressions should be implemented in such a way that when they are invoked the state of the user stacks remain unchanged. This is useful for implementing functions which manipulate the user stacks. Example for these functions are: The handlers mentioned in CORVALLISP, debuggers, etc.

4.2.1 Some rules for Z-lambda implementation

- Z-lambda's may never set global values. Each Z-lambda may only work within its own area and may not affect other functions. In another words, Z-lambdas may only manipulate the user stack and their own variables.
- Z-lambdas may not call other Z-lambdas. This is needed to keep the management of Z-lambda calls. We do not want to have to keep track of a Z-lambda stack.
- Z-lambdas do not take any arguments.
- A special register Z-ret must be maintained to keep track of the return address for Z-lambda expressions.

When the idea of Z-lambda is implemented in LIL, The CORVALLISP interpreter may be extended to allow:

Z-expr — This would be a user implemented Z-lambda which is useful when there is a need to implement new functions in LISP which need different kinds of **handlers**

Z-subr — This is the equivalent of the binary functions in LISP. The Z-subrs would be used to implement the standard handlers for CORVALLISP.

4.3 CORVALLISP

CORVALLISP is a prototype LISP interpreter. The purpose of CORVALLISP is to demonstrate usability of LIL.

CORVALLISP supports a subset of the *standard* LISP expressions.

- There is support for some operation on small numbers, lists, and atoms.
- A very simple one character one argument macro expander is available in CORVALLISP. Currently the only macro implemented is the Quote macro.
- There is no PROG function in the current implementation of CORVALLISP.
- There are separate spaces for each of the following objects:
  - — Cons-cell space: Used for cons-cell spaces allocation. The free cons-cells are linked together.
  - — Atom-header space. Used for atom-header space allocation. This space is a simple heap. Currently this space is also used for small number space allocation.
  - — String space: Used for print name space allocation. This space is a simple heap storage area. The atom-headers point to this space where their print names are stored. The print names are null terminated strings.
- A very simple minded garbage collector is used to reclaim the un-attached cons-cells. It uses the mark and sweep method. The cons-cell space is the only space that is garbage collected.

The *Oblist* or the symbol table is a hash table, where each entry is a linked list of all the atoms which have colliding hash values. The hashing is done on the print name of the atom. Cons-cells are used to implement the link list.

The objects supported by CORVALLISP are:

ATOM Any non-list (cons-cell) object, including small numbers. Each atom is identified by an atom-header. An atom-header contains:

1. Type field, which identifies the type of this object.
2. Garbage collection field, which is used for marking the atom during garbage collection (currently not used).
3. Value field, which points to the value of the atom.
4. Plist field, which points to a linked list of the properties of the atom.
5. Pname field, which points to the location in the string space where the print name of the atom resides.

CONS List building blocks. Each cons-cell has 4 fields

1. Type field, which identifies the type of this object.

2. Garbage collection field, which is used for marking the cons-cell during garbage collection.
3. Car field, points to the car of cell.
4. Cdr field, points to the cdr of cell.

SNUMS Small numbers. (32 bits in the current implementation) Small number or snums use the atom-header structure. Small numbers are not interned and they do not have a print name. They always evaluate to themselves.

SUBR Compiled functions which evaluate their arguments.

FSUBR Compiled functions which do not evaluate their arguments.

LAMBDA Lisp functions which evaluate their arguments.

FLAMBDA Lisp functions which do not evaluate their arguments.

Following is a list of the functions available in CORVALLISP:

4.3.1 quote FSUBR

Format: (quote expr)

Description:

Quote suppresses the evaluation of its argument. The macro ' may be used for quoting expressions.

Examples:

(quote x) 'x evaluates to x
(quote (foo bar)) '(foo bar) evaluates to (foo bar)

4.3.2 set SUBR

Format: (set atom expr)

Description:

Set evaluates both of its arguments, and then 'sets' the value of first argument to the value of the second argument. Set evaluates to its second argument.

Example:

(set (quote foo) (quote bar))

evaluates to bar and the value of foo is set to bar.

4.3.3 setq FSUBR

Format: (setq atom expr)

Description:

The first argument is not evaluated and must be an atom. The second argument is evaluated, and the 1st argument is set to the value of second argument. Setq evaluates to its second argument.

Example:

(setq x (quote (this that))

evaluated to (this that) and x is set to (this that)

4.3.4 gc SUBR

Format: (gc)

Description:

This functions performs garbage collection on the cons-cell space.

Example:

```
(gc)
GC: room: 12345 cons-cells.
```

4.3.5 room SUBR

Format: (room)

Description:

This function returns the number of cons-cells available.

Example:

```
(room)
12345
```

4.3.6 oblist SUBR

Format: (oblist)

Description:

This function prints the names of all the atoms in the oblist. The location in the hash-table and the hex value of the atom is also printed.

Example:

```
(oblist)

48: /, 4a
61: <, 47
62: =, 44
63: >, 48
117: t, a7cecc
628: rplaca, 3c
631: rplacd, 3b
633: append, 2e
```

654: oblist, 51
672: result, 4c4360
673: return, 1b

4.3.7 putp SUBR

Format: (putp atom atom expr)

Description:

The second argument will be put in the property list of the first argument and the value of the property will be set to expr. If the property already exists, its value will change to the new value. Otherwise the property is added to property list of the first argument. Putp evaluates to NIL.

Example:

(putp (quote x) (quote y) (setq zz (quote max)))

Evaluates to NIL.

4.3.8 getp SUBR

Format: (getp atom atom)

Description:

If the first argument has a property with the same pname of the second

argument, getp evaluates to the value of the property otherwise NIL.

Example:

from the above example,
(getp (quote x) (quote y)) evaluates to max.

4.3.9 return SUBR

Format: (return expr)

Description:

When ever CORVALLISP encounters an undefined function, the environment is saved and a new break level of CORVALLISP is entered. At each level, one can 'fake' the

value which is expected from the 'undefined' function and continue the evaluation of the rest of the expression.

Example:

(setq foo (quote (bady)))

System will enter a break level with undefined function bady.
The user can continue by:

(return (quote (a b c)))

The system continues with (quote (a b c)) as the result of function call (bady).

4.3.10 reset FSUBR

Format: (reset)

Description:

Reset puts CORVALLISP in level one and clears the environment.

Example:

(reset)

4.3.11 eval SUBR

Format: (eval expr)

Description:

Eval evaluates the value of its argument.

Example:

(eval (quote (quote x))

evaluates to x

4.3.12 progn FSUBR

Format: (progn expr expr ... expr)

Description:

Eval evaluates its argument and returns the value of its last argument.

Example:

(progn (quote x) (list (quote foo) (quote y)))

evaluates to (foo y)

4.3.13 evlis FSUBR

Format: (evlis expr expr ... expr)

Description:

Evlis evaluates to a list which contains the evaluated elements of its arguments.

Example:

(evlis (list (quote d) (quote z) (quote y)))

Evaluates to (d z y)

4.3.14 list SUBR

Format: (list expr expr ... expr)

Description:

List evaluates to a list containing a list of the values of its arguments.

Example:

(list (quote x) (quote y) (quote (a b c)))
evaluates to (x y (a b c))

4.3.15 append SUBR

Format: (append list list)

Description:

Append evaluates to a list containing elements of both lists.

Example:

(append (list (quote x) (quote y)) (list (quote foo) (quote bar)))
Evaluates to (x y foo bar)

4.3.16 car SUBR

Format: (car list)

Description:

Car evaluates to the car pointer of a list.

Example:

(setq foo (list (quote x) (quote y) (quote z)))
(car foo) will evaluate to x.

4.3.17 cdr SUBR

Format: (cdr list)

Description:

Cdr evaluates to the cdr (rest) pointer of a list.

Example:

From the above example
(cdr foo) will evaluate to (y z)

4.3.18 cons SUBR

Format: (cons expr list)

Description:

first argument and its cdr pointer equal to the value of its second argument. The second argument must evaluate to a list. CORVALLISP does not support dotted pairs.

Example:

(setq zip (quote foo))
(setq xx (list (quote this) (quote that)))
(setq yy (list (quote me) (quote my)))
(cons xx yy) evaluates to ((this that) (me my))
(cons zip xx) evaluates to (foo (me my))

4.3.19 null SUBR

Format: (null expr)

Description:

If the value of expr is non-nil, it evaluates to nil. If the value of expr is nil it evaluates to t.

Example:

(null t) evaluates to nil
(null nil) evaluates to t

4.3.20 + SUBR

Format: (+ expr1 expr2)

Description:

This function evaluates both of its arguments, and returns the sum of the two arguments.

Example:

(+ 3 2) evaluates to 5

4.3.21 - SUBR

Format: (- expr1 expr2)

Description:

This function evaluates both of its arguments, and subtracts the value of expr2 from expr1.

Example:

(- 3 2) evaluates to 1

4.3.22 * SUBR

Format: (* expr1 expr2)

Description:

This function evaluates both of its arguments, and multiplies the value of expr1 by expr2.

Example:

(* 3 2) evaluates to 6

4.3.23 / SUBR

Format: (/ expr1 expr2)

Description:

This function evaluates both of its arguments, and divides the value of expr1 by expr2.

Example:

(/ 3 2) evaluates to 1
(/ 10 2) evaluates to 5

4.3.24 %SUBR

Format: (% expr1 expr2)

Description:

This function evaluates both of its arguments, and returns the modulo expr1 and expr2. i.e., the remainder of the division of expr1 by expr2.

Example:

(% 3 2) evaluates to 1
(% 1 2) evaluates to 1
(% 10 2) evaluates to 0

4.3.25 = SUBR

Format: (= expr1 expr2)

Description:

This function evaluates both of its arguments, and returns **t** if expr1 is numerically equal to expr2, otherwise **nil** is returned.

Example:

(= 3 2) evaluates to nil
(= 3 (+ 1 2)) evaluates to t

4.3.26 > SUBR

Format: (> expr1 expr2)

Description:

This function evaluates both of its arguments, and returns **t** if expr1 is numerically greater than expr2, otherwise **nil** is returned.

Example:

```
(> 3 2)          evaluates to t
(> 3 (+ 1 3))    evaluates to nil
```

4.3.27 >= SUBR

Format: (>= expr1 expr2)

Description:

This function evaluates both of its arguments, and returns **t** if expr1 is numerically greater than or equal to expr2, otherwise **nil** is returned.

Example:

```
(>= 3 2)          evaluates to t
(>= 3 (+ 1 2))    evaluates to t
(>= 2 (add 1 3))  evaluates to nil
```

4.3.28 < SUBR

Format: (< expr1 expr2)

Description:

This function evaluates both of its arguments, and returns **t** if expr1 is numerically less than expr2, otherwise **nil** is returned.

Example:

```
(< 3 2)          evaluates to nil
(< 3 (+ 1 9))    evaluates to t
```

4.3.29 <= SUBR

Format: (<= expr1 expr2)

Description:

This function evaluates both of its arguments, and returns **t** if expr1 is numerically less than or equal to expr2, otherwise **nil** is returned.

Example:

```
(<= 3 2)          evaluates to nil
(<= 3 (+ 1 2))    evaluates to t
(<= 2 (+ 1 3))    evaluates to t
```

4.3.30 atom SUBR

Format: (atom expr)

Description:

Atom evaluates to t if the value of its argument is an atom, otherwise nil.

Example:

```
(atom (quote x))                  evaluates to t.
(atom (list (quote x) (quote y)))  evaluates to nil.
```

4.3.31 eq SUBR

Format: (eq atom atom)

Description:

If the value of both of the arguments point to the same object, eq evaluates to t. Otherwise eq evaluates to nil.

Example:

```
(setq x (quote y))
(setq z (quote y))
(eq t nil)    evaluates to nil.
(eq x y)      evaluates to t.
```

4.3.32 or SUBR

Format: (or expr expr .... expr)

Description:

Expressions are evaluated until the first non-nil value is seen, or the end of the argument list is reached. In the first case t is returned and in the second case nil is

returned.

Example:

```
(setq x (quote this))
(setq s x)
(or x (setq b (car foo)) nil nil) evaluates to t
```

4.3.33 and SUBR

Format: (and expr expr ... expr)

Description:

argument list is reached. In former case 'and' evaluates to nil. In the latter case, value of 'and' is 't'.

Example:

```
(setq x (quote this))
(setq boo nil)
(setq s x)
(or x (setq b (car foo)) boo t t) evaluates to nil
```

4.3.34 cond FSUBR

Format: (cond expr ... expr)

Description:

Starting from the first 'expr', EVAL is applied to 'car' of the 'expr'. If the result returned from EVAL is 'nil', the 'expr' is skipped. If the result is non-nil, PROGN is applied to the 'cdr' of the 'expr' and cond terminates with the value returned form PROGN. If the end of the 'cond list' is reached 'nil' is returned.

Example:

```
(setq x nil)
(cond (x          (setq foo bar))
      ((and x foo) nil)
      ((null x)   (setq y (quote foo)))
      (t          (print ss)))
```

The above expression evaluates to 'foo' because (null x)
is the first non-nil expression encountered.

4.3.35 rplacd SUBR

Format: (rplacd list list)

Description:

Rplacd actually replaces the cdr of a list with the value of its second argument.

Example:

```
(setq x (quote ( a b c)))
(rplacd x (quote (foo bar)))
```

Modifies x to be (a foo bar) and
evaluates to same.

4.3.36 rplaca SUBR

Format: (rplaca list expr)

Description:

Rplaca modifies its first argument to have the value of 'expr' as its car value.

Example:

```
(setq x (quote ( a b c)))
(rplaca x (quote (foo bar)))
```

Modifies x to be ((foo bar) b c) and
evaluates to same.

4.3.37 nconc SUBR

Format: (nconc list list)

Description:

Nconc works the same as append except that it modifies the first list.

Example:

```
(setq x (quote (a b c)))
(setq y (quote (foo bar)))
(nconc x y)     modifies x to be (a b c foo bar)
                and returns same.
```

4.3.38 de FSUBR

Format: (de atom (atom1 ... atom-n) expr ... expr)

Description:

De defines 'atom' to be a lambda expression with arguments 'atom1' ... 'atom-n' and the body '(expr ... expr)'

NOTE

The arguments are evaluated when lambda expr is invoked.

Example:

```
(de print_them (x y)
    (print x)
    (print y)
)
```

4.3.39 df FSUBR

Format: (df atom (atom1 ... atom-n) expr ... expr)

Description:

Df defines 'atom' to be a flambda expression with arguments 'atom1' and the body '(expr ... expr)'

NOTE

The arguments are not evaluated when flambda expr is invoked.

Example:

```
(df print_them (x y)
    (print x)
    (print y)
)
```

4.3.40 read FSUBR

Format: (read)

Description:

Read 'reads' an expression from the terminal. It evaluates to the expression.

Example:

(setq x (read))

4.3.41 print SUBR

Format: (print expr)

Description:

Print 'prints' and expr to the terminal. It evaluates to nil.

Example:

(print (quote (this is a line)))

evaluates to : (this is a line)

4.3.42 exit FSUBR

Format: (exit)

Description:

Exit terminates the CORVALLISP.

Example:

(exit)

5. Conclusion

LIL turned out to be a very easy language to implement and a delight to write the LISP interpreter with. Its constructs are simple, and convenient to use. While writing the interpreter there was not one feature which I wished I had added to LIL. Actually it would be very easy to cheat and extend the language by simply adding more macro definitions to the basic set of LIL definitions. As far as portability goes, LIL was ported from its original home on a PDP-11 under *UNIX* v6 to a VAX-11 under *UNIX* BSD, VAX-11 under VMS, Intel system 310/286 under Xenix, Intel system 301/386 under *UNIX* V.3 and finally, to iPSC/2 concurrent computer under NX. In all cases, the code ran with only few modifications to the definitions of LIL. An attempt was made to generate LIL code for INTEL 8086, using the ASM86 assembly language cross assembler under VMS, but the cross assembler had problems understanding some of the ASM86 macro expansions. It should be mentioned that there was one extra step which was needed for generating LIL code. To make the source code syntax fixed at all times we used the LISP like syntax of enclosing parentheses for expressions. The LIL code must be expanded by a macro expander to be able to be used by the host language compiler. A pre-processor is needed to transform the standard syntax of the LIL source code to that which is required by the macro expander. In most cases, it required changing the open statement bracket " here '(' ", and close statement bracket " here ')' " to the macro-warning and end-macro characters of the macro expander. For the versions of LIL on the PDP-11 and VAX the ED text editor was used to make the LIL code acceptable by the M6 and M4 macro expanders. The 8086 version did not use an outside macro expander. The LIL code was changed to ASM86 macro call format by a Pascal program. It is wise to write a macro expander in LIL so it could be ported along with the source code.

An improvement to this version of LIL which could be suggested, is changing the 'mkstr' expression so it could allocate space in some other fashion than sequential

allocation of free-memory. By adding a 'unmkstr' expression, LIL can support allocation and de-allocation of free-space.

Another improvement would be adding floating point macros.

There is one feature which is missing in LIL which should be brought up, that is, BIT level manipulation of data. There were times during writing of the LISP interpreter that we could have saved a lot of memory space by storing flags in bits instead of bytes. For example, we used up a whole byte for the garbage collection flag. In order to be able to take advantage of bit fields in a word, one must be aware of the word size of the machine. A program which compacts its data into a 60 bit word does not work on an 8 bit machine. It would be nice to have such features. The best way that it could be done is not obvious to me.

6. References

[1] Lane, T.,
"P-code and the p-machine", Hewlette-Packard Desktop
Computer Division, Fort Collins, Colorado, (June 8, 1978).

[2] Griss, M. L., Kessler R. R. and Maguire G. Q. Jr.,
"TLISP - A 'TRANSPORTABLE-LISP' Implementation in pcode",
Computer Science Department, University of Utah,
Salt Lake City, (April 1979).

[3] Griss, M. L. and Hearn A. C.,
"A Portable LISP Compiler", Department of Computer
Science University of Utah, Salt Lake City, (June 1979).

[4] Moore, S. J. II,
"The interlisp Virtual Machine Specification",
XEROX Palo Alto Research Center,
Palo Alto California. (September 1979).

[5] Burton, R. R., Masinter, L. M., Bell, A., Bobrow, D. G.,
Haugeland, W. S., Kaplan, R. M. and Sheil, B. A.,
"Interlisp-D: Overview and Status" XEROX Palo Alto Research Center,
Palo Alto California. (July 1981).

[6] Masinter, L. M. and Deursch L. P.,
"Local Optimization in a Compiler for Stack-based LISP Machines",
Proceedings of the LISP conference, Stanford (1980), and
XEROX Palo Alto Research Center, Palo Alto California. (1980)

[7] Kaplan, R. M. Sheil, B. A. and Burton, R. R,
"The interlisp-D I/O system",
XEROX Palo Alto Research Center, Palo Alto California. (1980)

[8] Allen, J. R.,
"Anatomy of LISP", McGraw-Hill, New York (1978).

[9] McCarty, et al,
"LISP 1.5 Programmers Manual", MIT Press (1962).

[10] Jensen, K. and Wirth, N.
"Pascal USER MANUAL AND REPORT", Springer-Verlag, New York (1974).

[11] Kernighan, W. Ritchie, D. M.,
"The C Programming Language", Prentice-Hall, New Jersey (1978)

[12] Harrison, M. C.,
"A Language Oriented Instruction Set for BALM",
SIGPLAN-SIGMICRO 9 (1974).

[13] Lampson, B. W. and Pier, K. A.,
"A processor for a high-performance personal computer",
Seventh international symposium on computer
architecture, La Baule, France (May 1980).

[14] Deutch, L. P.,
"Experience with a microprogrammed interlisp system",
IEEE Micro-11 conference (1978).

APPENDICES

Appendix A: Implementation notes

LIL to target language translation

The working implementation of the LIL translator uses the m4 macro expander to *map* the LIL statements to C. The translation in done in one pass through the LIL source code. The macro definitions are prepended to the source code and then piped to m4:

```
cat defs.1.8 lisp.l |m4 > lisp.c
```

The resulting 'C' code can then be compiled by the 'C' compiler, and ld loader to produce executable code.

How the code starts up

The 'C' program generated by the LIL translator contains a while loop which executes until the value of the global variable prog_switch becomes equal to the constant EXIT_FLAG. This value is pushed on the JPDL at the beginning of the code, which means if a POPJ operation is done without having pushed anything on the JPDL will end the program. The *codebegin* statement sets up the **main** function of 'C' program and pushes the EXIT_FLAG on JPDL. Then it sets the value of the prog_switch to the argument of *codebegin*, which is the first LIL function to take control.

Implementation of LIL functions

LIL functions are translated to *cases* of a switch statement. Each functions has a constant *address* associated with it. To invoke a function the program has to perform **POPJ** . The **POPJ** statement gets translated to:

```
prog_switch = POPJ     ;pop the address of the function from the JPDL
                       ;set the global variable prog_switch to
                            ;the new value
break                  ;and break out of the case statement.
```

This will give the flow of control back to the main while statement which if the value is not equal to the EXIT_FLAG will enter the case statement and the function takes control.

This approach will potentially create a very large main program. But it is simple to implement in most target languages. e.g., the case statement in Pascal, computed go to in FORTRAN, or a jump table in assembly. Of course if the target language is assembly it is much more efficient to use the actual address of the function entries instead of the constant assigned addresses.

A two pass approach

As I pointed out the above method creates one large main program, which some compilers and/or computer might have a hard time to deal with. There is a very simple solution to this problem.

We keep the idea about the main program with the while loop and the switch statement. But we only call a 'C' function with each case. That is instead of putting the code for the function at the case statement we actually create a 'C' function for the LIL function.

In order to do this, we take our definition of LIL macros and define every thing but the function definitions macros to null. We then change the function definitions to a case which calls the function. This would be the first pass of the translation which will produce the main program, with cases that call the functions.

Then we go back to the original LIL macro definitions and modify the function definition macros to expand to 'C' functions. The code produced by this pass will contain the functions only.

By we can now append the two files together and compile the 'C' code. section follow.

Pass 1

```
#
long poa( );
long poj( );
(changequote,<,>)
(define,ret)
(define,loop)
(define,loopend)
(define,inc)
(define,dec)
(define,srjump)
(define,call)
(define,byte,char)
(define,word,long)
(define,pop)
(define,push)
(define,bpush)
(define,jpush)
(define,apush)
(define,apop)
(define,jpop)
(define,popj)
(define,scopy)
(define,concat)
(define,scomp)
(define,strucdef,struct $1 {)
(define,fielddef, $3 $1[$2] ;)
(define,strucend, };
<
(cdef,size_$1, sizeof( struct $1 ) )
>
)
(define,mkstr)
(define,comment)
(define,c)
(define,vbl)
(define,string)
(define,character)
(define,integer)
(define,real)
(define,oct)
(define,struc)
(define,space)
(define,field)
(define,vdef,<<long>> $1;)
(define,cdef,<<#define>> $1 $2)
(define,add)
(define,sub)
(define,mul)
(define,div)
(define,mod)
```

```
(define,get)
(define,put)
(define,aget)
(define,aput)
(define,set)
(define,srdef)
(define,srend)
(define,fndef,case $1: __$1();)
(define,fnend,prog_switch=poj();break;)
(define,spacedef,<long add_$1, bot_$1 ;
(cdef,len_$1,$2)
$3 $1 [$2] ; >)
(define,initspace)
(define,checkspace)
(define,const)
(define,tagdef)
(define,tag)
(define,jump)
(define,eq)
(define,ne)
(define,gt)
(define,lt)
(define,ge)
(define,le)
(define,jumpif)
(define,jumpnif)
(define,jumpz)
(define,jumpnz)
(define,fopen)
(define,fclose)
(define,fread)
(define,fwrite)
(define,codebegin,<int _debug;
main(argc,argv)
int argc;
char **argv;
{
#define EXIT_FLAG -1
  long prog_switch;
  _debug = 0;
  if (argc > 1) _debug++;
  prog_switch = $1;
  puj(EXIT_FLAG);
  while (prog_switch != EXIT_FLAG)
  switch (prog_switch) {>)
(define,codeend, } })
```

Pass 2

```
(changequote,<,>)
(define,ret,return;)
(define,loop,$1_1:)
(define,loopend,goto $1_1;)
(define,inc,$1++;)
(define,dec,$1--;)
(define,srjump,$1();)
(define,call,$1();)
(define,byte,char)
(define,word,long)
(define,pop,(add_$1 == 0 ? ($1[add_$1=len_$1-1]) : $1[--add_$1]))
(define,push, <if ( add_$1 >= len_$1 ) { add_$1 = 0; };
        $1[add_$1++] = $2;>)
(define,bpush, <if ( bot_$1 < 0 ) { bot_$1 = len_$1-1;};
        $1[bot_$1--] = $2 ;>)
(define,jpush,puj( (long)$1 );)
(define,apush,pua( (long)$1 );)
(define,apop,poa())
(define,jpop,poj())
(define,scopy,strcpy((char *)&$2<<,>>(char *)&$1);)
(define,concat,strcat(((char *)&$2)<<,>>((char *)&$1));)
(define,scomp,strcmp((char *)&$1<<,>>(char *)&$2);)
(define,strucdef,struct $1 {)
(define,fielddef, $3 $1[$2] ;)
(define,strucend, };
<
(cdef,size_$1, sizeof( struct $1 ) )
>
)
(define,mkstr,&$2[(add_$2 += size_$1) - size_$1] )
(define,comment,/**/)
(define,c,$1,$2,$3,$4,$5,$6,$7,$8,$9)
(define,vbl,$1)
(define,string,<">$1<">)
(define,character,<'>$1<'>)
(define,integer,$1)
(define,real,$1)
(define,oct,<0>$1)
(define,struc,((struct $1 *) $2))
(define,space,$1)
(define,field,$1)
(define,vdef,<<extern long>> $1;)
(define,cdef,<<#define>> $1 $2)
(define,add,$1+$2)
(define,sub,$1-$2)
(define,mul,$1*$2)
(define,div,$1/$2)
(define,mod,$1%$2)
(define,get,$1->$2[$3])
(define,codebegin)
```

```
(define,codeend)
(define,put,(get,$1,$2,$3) = $4;)
(define,aget,$1[$2] )
(define,aput,$1[$2] = $3;)
(define,set,$1 = (long)$2;)
(define,srdef,$1() {)
(define,srend,})
(define,fndef,__$1() {)
(define,fnend,})
(define,popj,return;)
(define,spacedef,< extern long add_$1, bot_$1 ;
(cdef,len_$1,$2)
extern $3 $1 [$2] ; >)
(define,initspace,add_$1 = 0;bot_$1=len_$1-1;)
(define,checkspace,( size_$1+ add_$2 >= bot_$2 ? 0 : 1))
(define,const,$1)
(define,tagdef,$1:)
(define,tag,$1)
(define,jump,goto $1;)
(define,eq, (long) $1 == (long) $2)
(define,ne, (long) $1 != (long) $2)
(define,gt, (long) $1 > (long) $2)
(define,lt, (long) $1 < (long) $2)
(define,ge, (long) $1 >= (long) $2)
(define,le, (long) $1 <= (long) $2)
(define,jumpif, if ( $2 ){ goto $1;};)
(define,jumpnif,if ( !( $2 )){ goto $1 ;};)
(define,jumpz,if ( !( $2 )){ goto $1 ;};)
(define,jumpnz,if ( $2 ){ goto $1 ;};)
(define,fopen,$1=open(&$2<,>2);)
(define,fclose,close($1);)
(define,fread,<read((int)$1<,>(char *)&$2<,>$3);>)
(define,fwrite,write((int)$1<,>(char *)&$2<,>$3);)
```

Main program

```
#
long poa();
long poj();
struct common_fields {
   char cell_type[1] ;
   char gc[1] ;
};
#define size_common_fields sizeof( struct common_fields )
struct cons_cell {
   char cell_type[1] ;
   char gc[1] ;
   long car_pointer[1] ;
   long cdr_pointer[1] ;
};
#define size_cons_cell sizeof( struct cons_cell )
struct atom_header {
   char cell_type[1] ;
   char gc[1] ;
   long vcell[1] ;
   long plist[1] ;
   long pname[1] ;
};
#define size_atom_header sizeof( struct atom_header )
struct small_num {
   char cell_type[1] ;
   char gc[1] ;
   long vcell[1] ;
};
#define size_small_num sizeof( struct small_num )
struct string_args {
   char st[100] ;
};
#define size_string_args sizeof( struct string_args )
long add_buff, bot_buff ;
#define len_buff 100
char buff [100] ;
long add_apdl, bot_apdl ;
#define len_apdl 1000
long apdl [1000] ;
long add_jpdl, bot_jpdl ;
         ...
         ...
#define len_print 1000
char  print[1000] ;
long add_pnamesp, bot_pnamesp ;
#define len_pnamesp 1000
char pnamesp [1000] ;
#define LISP 1
#define LISP_1 2
#define MUL 75
```

```
        ...
        ...
#define ADD 76
#define SUB 77
#define MAC_EXP 78
#define MAC_EXP_1 79
long last;
long lisp_level;
long apar;
long apar1;
long fpar;
        ...
        ...
long result;
long pars;
long args;
long call_form;
long keeper;
long keeper2;
long sign;
#define C_MAC_CHAR '?
#define C_MARK -61
        ...
        ...
#define C_MARK_1 -62
#define C_NEWLINE '0
int _debug;
main(argc,argv)
int argc;
char **argv;
{
#define EXIT_FLAG -1
  long prog_switch;
  _debug = 0;
  if (argc > 1) _debug++;
  prog_switch = LISP;
  while (prog_switch != EXIT_FLAG)
  switch (prog_switch) {
case LISP: __LISP();
prog_switch=poj();break;
case LISP_1: __LISP_1();
prog_switch=poj();break;
case PUSH_RESULT_ON_APDL: __PUSH_RESULT_ON_APDL();
prog_switch=poj();break;
case FIND_END: __FIND_END();
prog_switch=poj();break;
case APPEND_1: __APPEND_1();


    ...
    ...

prog_switch=poj();break;
```

```
case LOAD: __LOAD();
prog_switch=poj();break;
prog_switch=poj();break;
case EQNUM: __EQNUM();
prog_switch=poj();break;
case LENUM: __LENUM();
prog_switch=poj();break;
case EXIT: __EXIT();
prog_switch=poj();break;
}}

chkpt(str)
char *str;
{
if (_debug) printf("%s0,str);
}
puj(lip)
long lip;
/*
  This routine pushes a 'LIL' object on the JPDL.
  The module is implemented only for space saving
  purposes.
*/
{
  if ( add_jpdl >= len_jpdl ) { add_jpdl = 0; };
        jpdl[add_jpdl++] = lip;
}



pua(lip)
long lip;
/*
  This routine pushes a 'LIL' object on the APDL.
  The module is implemented only for space saving
  purposes.
*/
{
  if ( add_apdl >= len_apdl ) { add_apdl = 0; };
        apdl[add_apdl++] = lip;
}

long poa()
/*
  This modules pops an object from the apdl
*/
{
  return
  (
    add_apdl == 0 ? apdl[add_apdl=len_apdl-1] : apdl[--add_apdl]
  );
}
```

```
long poj()
/*
  This modules pops an object from the jpdl
*/
{
  return
  (
    add_jpdl == 0 ? jpdl[add_jpdl=len_jpdl-1] : jpdl[--add_jpdl]
  );
}
```

Functions

```
/**/
__LISP() {
  init();
  write((int)C_OUTUNIT,(char *)"CORVALLISP0,11);
  puj( (long)LISP_1 );
  return;
}
/**/
__LISP_1() {
  puj( (long)LISP_1 );
  puj( (long)PRINT_1 );
  puj( (long)PUSH_RESULT_ON_APDL );
  puj( (long)EVAL_0 );
  puj( (long)PUSH_RESULT_ON_APDL );
  puj( (long)READ_0 );
  write((int)C_OUTUNIT,(char *)"0>>>>>>>>",lisp_level);
  mkcons();
  gc_hold = (long)poa();
  return;
}
/**/
__PUSH_RESULT_ON_APDL() {
  pua( (long)result );
return;
}
/**/
__FIND_END() {
  arg = (long)poa();
  pua( (long)arg );
  if ( (long)  arg == (long) nil ){ goto done_fe;};
  loop_until_end:
    temp = (long)((struct cons_cell *) arg)->cdr_pointer[0];
    if ( (long)  temp == (long) nil ){ goto done_fe;};
      arg = (long)temp;
      goto loop_until_end;
  done_fe:
    result = (long)arg;
    return;
}
        ...
        ...
/**/
__EXIT() {
  write((int)C_OUTUNIT,(char *)"CORVALLISP END0,15);
  read((int)inunit,(char *)&buff[0] ,1);
  return;
}
```

Debugging

Another trick which can be used is to have a definition file which for each LIL statement produces some tracing or some debugging information. The debugging macro set could be used during the program development and once the code is debugged the final version can use the no-debug macro set to produce clean code.

An example of the debug macro set might look like this:

```
(define,fndef,case $1: printf("Entering $1 function0);__$1();)
(define,fnend,printf("leaving $1 function0);prog_switch=poj();break;)
```

Appendix B: LIL definitions in C

The following definitions are in 'm4' macro processor format.

```
#
long poa??  ;
long poj??  ;
(changequote,<,>)
(define,ret,return;)
(define,loop,$1_1:)
(define,loopend,goto $1_1;)
(define,inc,$1++;)
(define,dec,$1--;)
(define,srjump,$1??  ;)
(define,call,$1??  ;)
(define,byte,char)
(define,word,long)
(define,pop,??add_$1 == 0 @@  ??$1[add_$1=len_$1-1]   : $1[--add_$1]  )
(define,push, <if ?? add_$1   =  len_$1    { add_$1 = 0; };
        $1[add_$1++] = $2;>)
(define,bpush, <if ?? bot_$1 ?  0    { bot_$1 = len_$1-1;};
        $1[bot_$1--] = $2 ;>)
(define,jpush,puj?? ??long  $1   ;)
(define,apush,pua?? ??long  $1   ;)
(define,apop,poa??  )
(define,jpop,poj??  )
(define,scopy,strcpy????char *  &$2<<,>>??char *  &$1  ;)
(define,concat,strcat??????char *  &$2  <<,>>????char *  &$1    ;)
(define,scomp,strcmp????char *  &$1<<,>>??char *  &$2  ;)
(define,strucdef,struct $1 {)
(define,fielddef,  $3 $1[$2] ;)
(define,strucend,  };
<
(cdef,size_$1, sizeof?? struct $1    )
>
)
(define,mkstr,&$2[??add_$2 += size_$1   - size_$1] )
(define,comment,/**/)
(define,c,$1,$2,$3,$4,$5,$6,$7,$8,$9)
(define,vbl,$1)
(define,string,<">$1<">)
(define,character,<'>$1<'>)
(define,integer,$1)
(define,real,$1)
(define,oct,<0>$1)
(define,struc,????struct $1 *   $2  )
(define,space,$1)
(define,field,$1)
(define,vdef,<<long>> $1;)
(define,cdef,<<#define>> $1 $2)
(define,add,$1+$2)
(define,sub,$1-$2)
```

```
(define,mul,$1*$2)
(define,div,$1/$2)
(define,mod,$1%$2)
(define,vget, (get,(struc,atom_header,$1),vcell,0))
(define,vset, (vget,$1) = ??long  $2;)
(define, vinit, <
  (scopy,(string,$1),(get,(struc,string_args,pass),st,0))
       (apush,pass)
  (srjump,intern)
  (set,arg,(apop))
  (set, $1, arg)
  (put,(struc,atom_header,arg),vcell,0,nil)
  (put,(struc,cons_cell,arg),cell_type,0,C_ATOM_HEDER)
>)
(define,codebegin,<int _debug;
main??argc,argv
int argc;
char **argv;
{
#define EXIT_FLAG -1
  long prog_switch;
  _debug = 0;
  if ??argc   1   _debug++;
  prog_switch = $1;
  while ??prog_switch != EXIT_FLAG
  switch ??prog_switch   {>)
(define,codeend, } })
(define,aget,$1[$2] )
(define,aput,$1[$2] = $3;)
(define,set,$1 = ??long  $2;)
(define,get,$1-  $2[$3])
(define,put,(get,$1,$2,$3) = $4;)
(define,srdef,$1??   {)
(define,srend,})
(define,fndef,case $1 :)
(define,fnend)
(define,popj,prog_switch=poj??   ;break;)
(define,spacedef,< long add_$1, bot_$1 ;
(cdef,len_$1,$2)
$3 $1 [$2] ; >)
(define,initspace,add_$1 = 0;bot_$1=len_$1-1;)
(define,checkspace,?? size_$1+ add_$2   = bot_$2 @@ 0 : 1  )
(define,const,$1)
(define,tagdef,$1:)
(define,tag,$1)
(define,jump,goto $1;)
(define,eq, ??long   $1 == ??long   $2)
(define,ne, ??long   $1 != ??long   $2)
(define,gt, ??long   $1    ??long   $2)
(define,lt, ??long   $1 ?   ??long   $2)
(define,ge, ??long   $1   = ??long   $2)
(define,le, ??long   $1 ? = ??long   $2)
(define,jumpif, if ??  $2   { goto $1;};)
```

```
(define,jumpnif,if ?? !?? $2     { goto $1 ;};)
(define,jumpz,if ?? !?? $2     { goto $1 ;};)
(define,jumpnz,if ?? $2   { goto $1 ;};)
(define,fopen,$1=open??&$2<,>2  ;)
(define,fclose,close??$1  ;)
(define,fread,<read????int  $1<,>??char *  &$2<,>$3  ;>)
(define,fwrite,write????int  $1<,>??char *  &$2<,>$3  ;)
(define,within,<
          ??????$1 ?  ??long  &??$2[0]     |
          ??$1   = ??long  &??$2[len_$2]     |
          ????$1-??long  &??$2[0]     %size_$3      @@ 0 : 1
          >)
```

```
chkpt(str)
char *str;
{
if (_debug) printf("%s0,str);
}
puj(lip)
long lip;
/*
  This routine pushes a 'LIL' object on the JPDL.
  The module is implemented only for space saving
  purposes.
*/
{
  if ( add_jpdl >= len_jpdl ) { add_jpdl = 0; };
        jpdl[add_jpdl++] = lip;
}

pua(lip)
long lip;
/*
  This routine pushes a 'LIL' object on the APDL.
  The module is implemented only for space saving
  purposes.
*/
{
  if ( add_apdl >= len_apdl ) { add_apdl = 0; };
        apdl[add_apdl++] = lip;
}

long poa()
/*
  This modules pops an object from the apdl
*/
{
  return
  (
    add_apdl == 0 ? apdl[add_apdl=len_apdl-1] : apdl[--add_apdl]
  );
}

long poj()
/*
  This modules pops an object from the jpdl
*/
{
  return
  (
    add_jpdl == 0 ? jpdl[add_jpdl=len_jpdl-1] : jpdl[--add_jpdl]
  );
```

```
}
```

Appendix C: Subset of LIL definitions in ASM86

```
      ;************************************
;
%*DEFINE (comment(a))
(;%a
)
;******************** UDI? calls ?
%*define(fread (fil,val,num))
(; read characters
)
%*define(fwrite (fil,val,num))
(; write characters
)
%*define(fopen(fil,nam))
(; open file
)
%*define(fclose(fil))
(;close file
)
%*define(length) (_len)
%*define(ad) (_add)
%*define(bot) (_bot)
%*define(bad) (_badd)
%*define(start) (_start)
%*define(size) (_size)

lildata         segment                 'data'
lil_byte db             1
lil_worddb              2
lildata         ends
%*DEFINE (cdef(con,vl))
(
lildata  segment                'data'
%con            db              %vl
lildata         ends
)
%*DEFINE (doval (val,reg))
(%match(a,b) (%val)
%if(%den(%b) eq 0) then
(               mov             %reg, %val
)
else
(%val
                pop             %reg
)
fi
)
%*DEFINE (ret)
(               ret
```

```
)
%*DEFINE (return)
(               ret
)
%*DEFINE (loop( lab))
(%lab:
)
%*DEFINE (loopend (lab))
(               jmp             %lab
)
%*DEFINE (inc (varbl))
(               inc             %varbl
)
%*DEFINE (dec (varbl))
(               dec             %varbl
)
%*DEFINE (srjump (sr))
(               call            %sr
)
%*DEFINE (call (routine))
(               call            %routine
)
%*DEFINE (byte) (db)
%*DEFINE (word) (dw)
%*DEFINE (bpush (sp,val)) LOCAL ok0
( %doval(%val,ax)
                mov             bx, %sp%bad
                cmp             bx, %sp%start
                jge             %ok0
                mov             bx, %sp%bot
%ok0:           mov             word ptr [bx], ax
                dec             bx
                dec             bx
                mov             %sp%bad, bx
)
%*DEFINE (push (sp,val)) LOCAL ok1
(%doval(%val,ax)
                mov             bx, %sp%ad
                cmp             bx, %sp%bot
                jle             %ok1
                mov             bx, %sp%start
%ok1:           mov             word ptr [bx], ax
                inc             bx
                inc             bx
                mov             %sp%ad, bx
)
%*DEFINE (jpush(val))
(               mov             cx,offset %val
%push(jpdl,cx)
)
%*DEFINE (apush(val))
(%push(apdl,%val))
%*DEFINE (pop(sp)) LOCAL ok2
```

```
(               mov             bx, %sp%ad
                dec             bx
                dec             bx
                cmp             bx, %sp%start
                jge             %ok2
                mov             bx, %sp%bot
%ok2:           mov             ax, word ptr [bx]
                mov             %sp%ad, bx
                push            ax
)
%*DEFINE (jpop)
(%pop(jpdl))
%*DEFINE(apop)
(%pop(apdl))
%*DEFINE (strucdef(struc))
(
lildata         segment                 'data'
%struc          struc
)
%*DEFINE (strucend(struc))
(%struc         ends
%struc%size     equ             size %struc
lildata         ends
)
%*DEFINE (fielddef( fi, siz, tp))
(  %fi          %tp             %siz dup(?)
)
%*DEFINE (mkstr(str, sp))
(               mov             bx, %sp%ad
                mov             cx, %str%size
                push            bx
                mov             %sp%ad, bx
)
%*DEFINE (initspace(sp))
(               lea             bx, %sp
                mov             %sp%ad, bx
                lea             bx, %sp%bot
                mov             %sp%bad, bx
)
%*DEFINE (vdef (vb))
(
lildata  segment                'data'
%vb             dw              ?
lildata         ends
)
%*DEFINE (string (st)) LOCAL stlab
(
lildata  segment                'data'
%stlab          db              '%st'
                db              0
lildata         ends
                mov             bx, offset %stlab
                push            bx
```

```
)
%*DEFINE (character (c))
('%c')
%*DEFINE( real (x)) (%x)
%*DEFINE (integer (x)) (%x)
%*DEFINE (hex (x)) (%x h)
%*DEFINE (struc (x)) (%x)
%*DEFINE (space (x)) (%x)
%*DEFINE (vbl(x)) (%x)
%*DEFINE (oct(x)) (%x q)
%*DEFINE (field(x)) (%x)

%*DEFINE (add (a,b))
(%doval(%a,ax)
%doval(%b,bx)
                add             ax, bx
                push            ax
)
%*DEFINE (sub (a,b))
(%doval(%a,ax)
%doval(%b,bx)
                sub             ax, bx
                push            ax
)
%*DEFINE (mul (a,b))
(%doval(%a,ax)
%doval(%b,bx)
                mul             bl
                push            ax
)
%*DEFINE (div (a,b))
(%doval(%a,ax)
%doval(%b,bx)
                div             bl
                xor             ah, ah
                push            ax
)
%*DEFINE (mod (a,b))
(%doval(%a,ax)
%doval (%b,bx)
                div             bl
                xor             cx,cx
                mov             cl, ah
                push            cx
)
%*DEFINE (get (var,fil,off)) LOCAL do_byte getout
(               lea             si, %var
                lea             bx, [si].%fil
                mov             cx, %off
                cmp             lil_byte, type %fil
                je              %do_byte
                shl             cx, 1
                add             bx, cx
```

```
                mov             ax, word ptr [bx]
                jmp             %getout
%do_byte:       add             bx, cx
                mov             al, byte ptr [bx]
                xor             ah, ah
%getout:        push            ax
)
%*DEFINE (put (var,fil,off,val)) LOCAL dobyte putout
(%doval(%val,dx)
                lea             si, %var
                lea             bx, [si].%fil
                mov             cx, %off
                cmp             lil_byte, type %fil
                je              %dobyte
                shl             cx, 1
                add             bx, cx
                mov             word ptr [bx], dx
                jmp             %putout
%dobyte:        add             bx, cx
                mov             byte ptr [bx], dl
%putout:
)

%*DEFINE (aget (var,off)) LOCAL ado_byte agetout
(               lea             si, %var
                mov             cx, %off
                cmp             lil_byte, type %var
                je              %ado_byte
                shl             cx, 1
                add             bx, cx
                mov             ax, word ptr [bx]
                jmp             %agetout
%ado_byte:      add             bx, cx
                mov             al, byte ptr [bx]
                xor             ah, ah
%agetout:       push            ax
)
%*DEFINE (aput (var,off,val)) LOCAL adobyte aputout
(%doval(%val,dx)
                lea             si, %var
                mov             cx, %off
                cmp             lil_byte, type %var
                je              %adobyte
                shl             cx, 1
                add             bx, cx
                mov             word ptr [bx], dx
                jmp             %aputout
%adobyte:       add             bx, cx
                mov             byte ptr [bx], dl
%aputout:
)
%*DEFINE (set (a,b))
(%doval(%b, ax)
```

```
                mov             %a, ax
)
%*DEFINE (databegin)
(
lildata  segment                'data'
)
%*DEFINE (dataend)
(
lildata  ends
)
%*DEFINE (codebegin (fn))
(; set up stack
lilstack segment                'stack'
                dw      1000    dup (0)
lstack_bot      label   word
lilstack ends
lilcode         segment                 'code'
                assume          cs:lilcode, ds:lildata, ss:lilstack
                mov             ax, lilstack
                mov             ss, ax
                mov             sp, offset lstack_bot
                mov             ax, lildata
                mov             ds, ax
%jpush(lilcodeend)
                jmp             %fn
)
%*DEFINE (codeend(lab))
(lilcodeend:
lilcode         ends
                end)

%*DEFINE (srdef(sr))
(%sr            label           near
)
%*DEFINE (srend(sr))
(;*************** END OF %sr
)
%*DEFINE (fndef (fn))
(%fn            label           near
)
%*DEFINE (fnend(fn))
(;*************** END OF %fn
)
%*DEFINE (popj)
(%jpop
                pop             ax
                jmp             ax
)
%*DEFINE (spacedef(sp,leng,tp))
(
lildata  segment                'data'
%sp%start       label           word
%sp             %tp             %leng dup (0)
```

```
%sp%bot                 label           word
%sp%ad          dw              ?
%sp%bad                 dw              ?
%sp%dength      equ             size %sp
lildata         ends
)
%*DEFINE (const (x) ) (%x)
%*DEFINE (tagdef (tag))
(%tag           label           near
)
%*DEFINE ( tag (x) ) (%x)
%*DEFINE (jump(lb))
(               jmp             %lb
)
%*DEFINE (jumpif(lab,cond))
(%cond          %lab
)
%*DEFINE (jumpnif (lab,cond)) LOCAL outnif
(%cond          %outnif
                jmp             %lab
%outnif         label           near
)
%*DEFINE (eq (a,b))
(%doval(%a,ax)
%doval(%b,bx)
                cmp             ax, bx
                je)
%*DEFINE (ne (a,b))
(%doval(%a,ax)
%doval(%b,bx)
                cmp             ax, bx
                jne)
%*DEFINE (gt (a,b))
(%doval(%a,ax)
%doval(%b,bx)
                cmp             ax, bx
                jg)
%*DEFINE (lt (a,b))
(%doval(%a,ax)
%doval(%b,bx)
                cmp             ax, bx
                jl)
%*DEFINE (ge (a,b))
(%doval(%a,ax)
%doval(%b,bx)
                cmp             ax, bx
                jge)
%*DEFINE (le (a,b))
(%doval(%a,ax)
%doval(%b,bx)
                cmp             ax, bx
                jle)
%*DEFINE (jumpz(lab,var))
```

```
(%doval(%var,ax)
                cmp             ax, 0
                je              %lab
)
%*DEFINE (jumpnz(lab,var))
(%doval (%var,ax)
                cmp             ax, 0
                jne             %lab
)
%*DEFINE (scopy(s1,s2)) LOCAL sc_loop sc_done
(%s1
                pop             ax
                push            bx
%s2
                pop             ax
                push            bx

                pop             di              ;second string
                pop             si              ;first    string
                mov             bx, 0
%sc_loop:       mov             cl, byte ptr ds:[si] [bx]
                mov             byte ptr ds:[di] [bx], cl
                cmp             byte ptr ds:[si] [bx], 0
                je              %sc_done
                inc             bx
                jmp             %sc_loop
%sc_done:
)
%*DEFINE (scomp(s1,s2)) LOCAL scm_loop scm_done scm_eq scm_ne
(%s1
                pop             ax
                push            bx
%s2
                pop             ax
                push            bx

                pop             di              ;second string
                pop             si              ;first string
                mov             bx, 0
%scm_loop:      mov             cl, byte ptr ds:[di] [bx]
                cmp             byte ptr ds:[si] [bx], cl
                jne             %scm_ne
                cmp             byte ptr ds:[si] [bx] , 0
                je              %scm_eq
                inc             bx
                jmp             %scm_loop
%scm_ne:        mov             ax, 1
                jmp             %scm_done
%scm_eq:        mov             ax, 0
%scm_done:      push            ax
)
```

Appendix D: CORVALLISP listing in LIL

```
      (comment,********** Constructs **********)
(strucdef, common_fields)
  (fielddef,cell_type,1,(byte))
  (fielddef,gc,1,(byte))
(strucend, common_fields)
(strucdef, cons_cell)
  (fielddef,cell_type,1,(byte))
  (fielddef,gc,1,(byte))
  (fielddef,car_pointer,1,(word))
  (fielddef,cdr_pointer,1,(word))
(strucend,cons_cell)
(strucdef,atom_header)
  (fielddef,cell_type,1,(byte))
  (fielddef,gc,1,(byte))
  (fielddef,vcell,1,(word))
  (fielddef,plist,1,(word))
  (fielddef,pname,1,(word))
(strucend,atom_header)
(strucdef,small_num)
  (fielddef,cell_type,1,(byte))
  (fielddef,gc,1,(byte))
  (fielddef,vcell,1,(word))
(strucend,small_num)
(strucdef,string_args)
  (fielddef,st,100,(byte))
(strucend,string_args)
(comment,********** Spaces **********)
(spacedef,buff,100,(byte))
(spacedef,apdl,100000,(word))
(spacedef,jpdl,100000,(word))
(spacedef,cons_sp,6000000,(byte))
(spacedef,atom_sp,1000000,(byte))
(spacedef,strg_sp,10000,(byte))
(cdef, C_OBLIST_LEN, 1000)
(spacedef,oblist,C_OBLIST_LEN,(word))
(spacedef,(space,print),(integer,100),(byte))
(spacedef,pnamesp,1000,(byte))
(comment, ********** Function Constansts **********)
(cdef,LISP,1)
(cdef,LISP_1,2)
(cdef,PUSH_RESULT_ON_APDL,3)
(cdef,FIND_END,4)
(cdef,APPEND_1,5)
(cdef,LOAD,6)
(cdef,LOAD_1,7)
(cdef,CHECK_LOAD_END,8)
(cdef,READ_0,9)
(cdef,READ,10)
(cdef,READATOM,11)
```

```
(cdef,RDLST,12)
(cdef,RDLST_1,13)
(cdef,RDLST_2,14)
(cdef,RDLST_3,15)
(cdef,PRINTATOM,65)
(cdef,TERPRINT,16)
(cdef,PRINT,17)
(cdef,PRINT_1,18)
(cdef,PRINT_2,19)
(cdef,ATOM_1,20)
(cdef,NCONC_1,21)
(cdef,EVAL_0,22)
(cdef,EVAL,23)
(cdef,ARG_HANDLER,24)
(cdef,FN_ERROR,25)
(cdef,RESET,26)
(cdef,RETURN,27)
(cdef,BODY_HANDLER,28)
(cdef,RESULT_HANDLER,29)
(cdef,EVAL_ARG,30)
(cdef,NEVAL_ARG,31)
(cdef,PROGN,32)
(cdef,DO_BINDINGS,33)
(cdef,OR,34)
(cdef,AND,35)
(cdef,DE,36)
(cdef,DF,37)
(cdef,LIST,38)
(cdef,COND,39)
(cdef,QUOTE,40)
(cdef,COND_1,41)
(cdef,EVLIS,42)
(cdef,EVLIS_1,43)
(cdef,EVLIS_2,44)
(cdef,EVLIS_3,45)
(cdef,APPEND,46)
(cdef,CP_LIST,47)
(cdef,CP_LIST_1,48)
(cdef,LAST,49)
(cdef,CAR,50)
(cdef,CDR,51)
(cdef,CONS,52)
(cdef,SET,53)
(cdef,SETQ,54)
(cdef,SET0,55)
(cdef,NULL,56)
(cdef,ATOM,57)
(cdef,EQ,58)
(cdef,RPLACD,59)
(cdef,RPLACA,60)
(cdef,NCONC,61)
(cdef,PUTP,62)
(cdef,GETP,63)
```

```
(cdef,EXIT,64)
(cdef,PRINTSMALL,66)
(cdef, MKSMALL, 67)
(cdef, EQNUM, 68)
(cdef, LENUM, 69)
(cdef, GENUM, 70)
(cdef, LNUM, 71)
(cdef, GNUM, 72)
(cdef, MOD, 73)
(cdef, DIV, 74)
(cdef, MUL, 75)
(cdef, ADD, 76)
(cdef, SUB, 77)
(cdef, MAC_EXP, 78)
(cdef, MAC_EXP_1, 79)
(cdef,TERPRINT_1,80)
(cdef,OBLIST,81)
(cdef,ROOM,82)
(cdef,GC,83)
(comment, ********** System Variables **********)
(vdef,gc_warn)
(vdef,roomcnt)
(vdef,sign)
(vdef,last)
(vdef,lisp_level)
(vdef,gctemp)
(vdef,gcsearch)
(vdef,gccounter)
(vdef,gcar)
(vdef,gcdr)
(vdef,gc_old)
(vdef,avail_cons)
(vdef,keep_cons)
(vdef,loc_pnsp)
(vdef,ploc)
(vdef,propname)
(vdef,propval)
(vdef,atomname)
(vdef,pass)
(vdef,eof)
(vdef,old_val)
(vdef,f_par)
(vdef,fncell_type)
(vdef,cell_type_1)
(vdef,counter)
(vdef,front)
(vdef,back)
(vdef,nil)
(vdef,inunit)
(vdef,outunit)
(vdef,t)
(vdef,arg1)
(vdef,arg2)
```

```
(vdef,temp)
(vdef,hashval)
(vdef,argu)
(vdef,loc)
(vdef,search)
(vdef,old_search)
(vdef,pointer)
(vdef,a_found)
(vdef,f_body)
(vdef,apar)
(vdef,apar1)
(vdef,fpar)
(vdef,fpar1)
(vdef,fparval)
(vdef,arg)
(vdef,rest)
(vdef,first)
(vdef,first_case)
(vdef,cond_body)
(vdef,form)
(vdef,fn)
(vdef,val)
(vdef,result)
(vdef,pars)
(vdef,args)
(vdef,call_form)
(vdef, keeper)
(vdef, keeper2)
(vdef, aarg1)
(vdef, aarg2)
(comment,  ******************************************** )
(comment,*************** SYSTEM CONSTANTS ******************)
(cdef,C_MAC_CHAR,'^')
(cdef,C_INUNIT,0)
(cdef,C_OUTUNIT,1)
(cdef,C_ATOM_HEDER,1)
(cdef,C_SUB,2)
(cdef,C_FSUB,3)
(cdef,C_LAM,4)
(cdef,C_FLAM,5)
(cdef,C_CONS_CELL,6)
(cdef,C_SNUM,7)
(cdef,S_LPAR,"??")
(cdef,S_RPAR," ")
(cdef,S_BLANK," ")
(cdef,C_LPAR,'??')
(cdef,C_RPAR,' ')
(cdef,C_BLANK,' ')
(cdef,C_TAB,' ')
(cdef,C_MARK,-5533111)
(cdef,C_MARK_1,-5533112)
(cdef,C_NEWLINE,'0)
(codebegin,LISP)
```

```
(comment,**************************************************)
(fndef,LISP)
  (srjump,init)
  (fwrite,C_OUTUNIT,(string,CORVALLISP0,11)
  (jpush,LISP_1)
  (popj)
(fnend,lisp)
(comment,**************************************************)
(fndef,LISP_1)
  /*
  (jumpif, no_need_to_gc, (gt, gc_warn, 0))
  (srjump, gc_cons)
  (tagdef, no_need_to_gc)
  */
  (jpush,LISP_1)
  (jpush,PRINT_1)
  (jpush,PUSH_RESULT_ON_APDL)
  (jpush,EVAL_0)
  (jpush,PUSH_RESULT_ON_APDL)
  (jpush,READ_0)
  (fwrite,C_OUTUNIT,(string,0>>>>>>>>>>>>>>>>>>>>>>>>>),lisp_level)
  (fwrite,outunit, S_BLANK, 1)
  (popj)
(fnend,lisp_1)
(comment,**************************************************)
(fndef,PUSH_RESULT_ON_APDL)
  (apush,(get, (struc, atom_header, result), vcell, 0))
(popj)
(fnend,push_result)
(comment,**************************************************)
(fndef,FIND_END)
  (set,arg,(apop))
  (apush,arg)
  (jumpif,done_fe,(eq,arg,nil))
  (tagdef,loop_until_end)
    (set,temp,(get,(struc,cons_cell,arg),cdr_pointer,0))
    (jumpif,done_fe,(eq,temp,nil))
      (set,arg,temp)
      (jump,loop_until_end)
  (tagdef,done_fe)
    (put, (struc, atom_header, result), vcell, 0, arg)
    (popj)
(fnend,find_end)
(comment,**************************************************)
(fndef,APPEND_1)
  (set,arg1,(apop))
  (set,arg,(apop))
  (set,arg2,(apop))
  (jumpif,arg1_is_nil,(eq,arg1,nil))
  (put,(struc,cons_cell,arg1),cdr_pointer,0,arg2)
  (put, (struc, atom_header, result), vcell, 0, arg)
  (popj)
  (tagdef,arg1_is_nil)
```

```
      (put, (struc, atom_header, result), vcell, 0, arg2)
      (popj)
(fnend,append_1)
(comment,**********************************************************)
(fndef,LOAD)
   (set,arg,(apop))
   (set,arg,(get,(struc,cons_cell,arg),car_pointer,0))
   (set,ploc,(get,(struc,atom_header,arg),pname,0))
   (fopen,arg1,(aget,pnamesp,ploc))
   (set,inunit,arg1)
   (set,eof,1)
   (jpush,LOAD_1)
   (popj)
(fnend,load)
(comment,**********************************************************)
(fndef,LOAD_1)
   (jpush,LOAD_1)
   (jpush,PRINT_1)
   (jpush,PUSH_RESULT_ON_APDL)
   (jpush,EVAL_0)
   (jpush,PUSH_RESULT_ON_APDL)
   (jpush,CHECK_LOAD_END)
   (jpush,READ_0)
   (popj)
(fnend,load_1)
(comment,**********************************************************)
(fndef,CHECK_LOAD_END)
   (jumpif,doneload,(le,eof,0))
      (popj)
(tagdef,doneload)
   (set,temp,(jpop))
   (set,temp,(jpop))
   (set,temp,(jpop))
   (set,temp,(jpop))
   (set,temp,(jpop))
   (put, (struc, atom_header, result), vcell, 0, nil)
   (fclose,inunit)
   (set,inunit,C_INUNIT)
   (popj)
(fnend,check_load_end)
(comment,**********************************************************)
(fndef,READ_0)
   (tagdef,spaces)
      (set,eof,(fread,inunit,(aget,buff,0),1))
      (jumpif,readone,(le,eof,0))
      (jumpif,spaces,(eq,(aget,buff,0),C_BLANK))
      (jumpif,spaces,(eq,(aget,buff,0),C_TAB))
      (jumpif,spaces,(eq,(aget,buff,0),C_RPAR))
      (jumpif,spaces,(eq,(aget,buff,0),C_NEWLINE))
      (jumpif,aread,(ne,(aget,buff,0),C_LPAR))
      (jpush,RDLST)
      (popj)
   (tagdef,aread)
```

```
    (jpush,READATOM)
  (tagdef,readone)
    (popj)
  (tagdef, macread)
    (jpush, MAC_EXP)
    (popj)
(fnend,read_0)
(comment,**********************************************************)
(fndef, MAC_EXP)
  (srjump, mkcons)
  (vset, keeper, (apop))
  (put,(struc,cons_cell,(vget,keeper)),car_pointer,0,nil)
  (srjump, mkcons)
  (vset, keeper2, (apop))
  (put, (struc, cons_cell,(vget,keeper)), cdr_pointer, 0,
       (vget,keeper2))
  (put,(struc,cons_cell,(vget,keeper2)),car_pointer,0,nil)
  (put,(struc,cons_cell,(vget,keeper2)),cdr_pointer,0,nil)
  (scopy,(string,quote),(get,(struc,string_args,pass),st,0))
  (apush,pass)
  (srjump,intern)
  (put, (struc, atom_header, result), vcell, 0, (apop))
  (put, (struc, cons_cell, (vget,keeper)), car_pointer, 0,
      (get, (struc, atom_header, result), vcell, 0))
  (apush, (vget,keeper))
  (jpush, MAC_EXP_1)
  (jpush,READ_0)
  (popj)
(fnend, MAC_EXP)
(comment,**********************************************************)
(fndef, MAC_EXP_1)
  (vset, keeper, (apop))
  (vset, keeper2, (get, (struc, cons_cell, (vget,keeper)),
     cdr_pointer, 0))
  (put, (struc, cons_cell, (vget,keeper2)), car_pointer, 0,
      (get, (struc, atom_header, result), vcell, 0))
  (put, (struc, atom_header, result), vcell, 0, (vget,keeper))
  (popj)

(fnend, MAC_EXP_1)
(comment,**********************************************************)
(fndef,READ)
  (set,arg,(apop))
  (jpush,READ_0)
  (popj)
(fnend,read)
(comment,**********************************************************)
(fndef,READATOM)
 (set,counter,99)
 (tagdef,loopclear)
  (put,(struc,string_args,pass),st,counter,(character, ))
  (jumpz,readchars,counter)
  (set,counter,(sub,counter,1))
```

```
    (jump,loopclear)
  (tagdef,readchars)
    (put,(struc,string_args,pass),st,0,(aget,buff,0))
    (jumpif,dotheatom,(ne,(aget,buff,0),C_MAC_CHAR))
      (jpush, MAC_EXP)
      (popj)
    (tagdef,dotheatom)
    (set,counter,0)
    (tagdef,charloop)
      (fread,inunit,(aget, buff, 0),1)
      (jumpif,endatom,(eq,(aget,buff,0),C_LPAR))
      (jumpif,endatom,(eq,(aget,buff,0),C_RPAR))
      (jumpif,endatom,(eq,(aget,buff,0),C_BLANK))
      (jumpif,endatom,(eq,(aget,buff,0),C_TAB))
      (jumpif,endatom,(eq,(aget,buff,0),C_NEWLINE))
      (jumpif,storechar,(le,counter,98))
    (jump,charloop)
 (tagdef,storechar)
      (inc,counter)
      (put,(struc,string_args,pass),st,counter,(aget,buff,0))
      (jump,charloop)
    (tagdef,endatom)
      (apush,pass)
      (srjump,intern)
       (put, (struc, atom_header, result), vcell, 0, (apop))
      (popj)
(fnend,readatom)
(comment,**************************************************)
(fndef,RDLST)
 (put,(struc,atom_header,front),vcell,0,nil)
 (jpush,RDLST_1)
 (popj)
(fnend,readlist)
(comment,**************************************************)
(fndef,RDLST_1)
  (tagdef,spaceloop)
  (fread,inunit,(aget, buff, 0),1)
  (jumpif,spaceloop,(eq,(aget,buff,0),C_BLANK))
  (jumpif,spaceloop,(eq,(aget,buff,0),C_TAB))
  (jumpif,spaceloop,(eq,(aget,buff,0),C_NEWLINE))
  (jumpnif,restread,(eq,(aget,buff,0),C_RPAR))
  (put, (struc, atom_header, result), vcell, 0,
    (get,(struc,atom_header,front),vcell,0))
  (aput,buff,0,C_BLANK)
  (popj)
 (tagdef,restread)
  (jumpnif,atomfound,(eq,(aget,buff,0),C_LPAR))
  (apush,(get,(struc,atom_header,front),vcell,0))
  (jpush,RDLST_2)
  (jpush,RDLST)
  (popj)
  (tagdef,atomfound)
   (apush,(get,(struc,atom_header,front),vcell,0))
```

```
    (jpush,RDLST_2)
    (jpush,READATOM)
    (popj)
 (fnend,readlist_1)
 (comment,**********************************************************)
 (fndef,RDLST_2)
  (put,(struc,atom_header,front),vcell,0,(apop))
  (apush,(get, (struc, atom_header, result), vcell, 0))
  (apush,nil)
  (srjump,cons_1)
  (apush,(get, (struc, atom_header, result), vcell, 0))
  (apush,(get,(struc,atom_header,front),vcell,0))
  (jpush,RDLST_3)
  (jpush,NCONC_1)
  (popj)
 (fnend,readlist_2)
 (comment,**********************************************************)
 (fndef,RDLST_3)
  (put,(struc,atom_header,front),vcell,0,
    (get, (struc, atom_header, result), vcell, 0))
   (jumpnif,next1,(eq,(aget,buff,0),C_RPAR))
   (aput,buff,0,C_BLANK)
   (put, (struc, atom_header, result), vcell, 0,
     (get,(struc,atom_header,front),vcell,0))
   (popj)
   (tagdef,next1)
    (jumpnif,next2,(eq,(aget,buff,0),C_LPAR))
    (apush,(get,(struc,atom_header,front),vcell,0))
    (jpush,RDLST_2)
    (jpush,RDLST_1)
    (popj)
   (tagdef,next2)
    (jpush,RDLST_1)
    (popj)
 (fnend,readlist_3)
 (comment,**********************************************************)
 (fndef,PRINTATOM)
  (set,arg,(apop))
  (set,ploc,(get,(struc,atom_header,arg),pname,0))
  (scopy,(aget,pnamesp,ploc),(get,(struc,string_args,pass),st,0))
    (set,counter,0)
    (tagdef,cont_print)
     (jumpif,outpr,(eq,counter,100))
     (aput,buff,0,(get,(struc,string_args,pass),st,counter))
     (jumpif,outpr,(eq,(aget,buff,0),(character, )))
     (fwrite,C_OUTUNIT,(aget,buff,0),1)
     (inc,counter)
     (jump,cont_print)
    (tagdef,outpr)
    (fwrite,C_OUTUNIT,S_BLANK,1)
     (popj)
 (fnend,printatom)
 (comment,**********************************************************)
```

```
(fndef,PRINTSMALL)
  (set,arg,(apop))
  (set, arg, (get,(struc,atom_header,arg),vcell,0))
  (jumpnz, nzeronum, arg)
    (fwrite, outunit, (string, 0), 1)
    (fwrite,outunit, S_BLANK, 1)
    (popj)
  (tagdef, nzeronum)
  (jumpif, posit, (gt, arg, 0))
    (set, arg, (mul, arg, -1))
    (fwrite, outunit, (string, -), 1)
  (tagdef, posit)
  (set, counter, 50)
  (tagdef, snum_loop)
    (set, old_val, (mod, arg, 10))
    (aput, buff, counter, (add, old_val, '0'))
    (dec, counter)
    (set, arg, (div, arg, 10))
    (jumpz, dopoping, arg)
    (jump, snum_loop)
  (tagdef,dopoping)
  (fwrite,outunit,(aget, buff, (add, counter, 1)),(sub, 50,counter))
  (tagdef, donesmalp)
  (fwrite,outunit,S_BLANK,1)
   (popj)
(fnend,printsmall)
(comment,**********************************************************)
(fndef, GC)
  (set, arg, (apop))
  (srjump, gc_cons)
  (put, (struc, atom_header, result), vcell, 0, t)
  (popj)
(fnend, GC)
(comment,**********************************************************)
(fndef, ROOM)
  (set, arg, (apop))
  (set, arg, avail_cons)
  (set, roomcnt, 0)
  (tagdef, looproom)
    (jumpif, doneroom, (eq, arg, nil))
    (inc, roomcnt)
    (set, arg, (get, (struc, cons_cell, arg), car_pointer, 0))
    (jump, looproom)
  (tagdef, doneroom)
  (apush, roomcnt)
  (jpush, MKSMALL)
  (popj)
(fnend, ROOM)
(comment,**********************************************************)
(fndef,TERPRINT)
  (set,arg,(apop))
  (put, (struc, atom_header, result), vcell, 0, nil)
  (jpush,TERPRINT_1)
```

```
    (popj)
  (fndef,TERPRINT_1)
    (fwrite,C_OUTUNIT,(string,0,1)
    (popj)
  (fnend,terprint)
  (comment,**********************************************************)
  (fndef,PRINT)
    (set,arg,(apop))
    (put, (struc, atom_header, result), vcell, 0, nil)
    (set,arg1,(get,(struc,cons_cell,arg),car_pointer,0))
    (apush,arg1)
    (jpush,TERPRINT_1)
    (jpush,PRINT_1)
    (popj)
  (fnend,print)
  (comment,**********************************************************)
  (fndef,PRINT_1)
    (set,arg,(apop))
    (set,cell_type_1,(get,(struc,cons_cell,arg),cell_type,0))
    (jumpif, is_list,(eq,cell_type_1,C_CONS_CELL))
    (jumpif, is_smal_1,(eq,cell_type_1,C_SNUM))
    (apush,arg)
    (jpush,PRINTATOM)
    (popj)
    (tagdef, is_smal_1)
    (apush,arg)
    (jpush,PRINTSMALL)
    (popj)
    (tagdef,is_list)
    (jpush,PRINT_2)
    (jpush,PRINT_1)
    (apush,(get,(struc,cons_cell,arg),cdr_pointer,0))
    (apush,(get,(struc,cons_cell,arg),car_pointer,0))
    (fwrite,C_OUTUNIT,S_LPAR,1)
    (popj)
  (fnend,print_1)
  (comment,**********************************************************)
  (fndef,PRINT_2)
    (set,arg,(apop))
    (jumpif,printend,(eq,arg,nil))
    (apush,(get,(struc,cons_cell,arg),cdr_pointer,0))
    (apush,(get,(struc,cons_cell,arg),car_pointer,0))
    (jpush,PRINT_2)
    (jpush,PRINT_1)
    (popj)
    (tagdef,printend)
    (fwrite,C_OUTUNIT,S_RPAR,1)
    (popj)
  (fnend,print_2)
  (comment,**********************************************************)
  (fndef,ATOM_1)
    (set,arg,(apop))
    (set,cell_type_1,(get,(struc,cons_cell,arg),cell_type,0))
```

```
    (jumpif,isatom,(eq,cell_type_1,C_ATOM_HEDER))
    (jumpif,isatom,(eq,cell_type_1,C_SNUM))
      (put, (struc, atom_header, result), vcell, 0, nil)
      (popj)
    (tagdef,isatom)
      (put, (struc, atom_header, result), vcell, 0, t)
      (popj)
  (fnend,atom_1)
  (comment,**********************************************************)
  (fndef,NCONC_1)
    (jpush,APPEND_1)
    (jpush,PUSH_RESULT_ON_APDL)
    (jpush,FIND_END)
    (popj)
  (fnend,nconc_1)
  (comment,**********************************************************)
  (fndef,EVAL_0)
    (set,form,(apop))
    (jumpif,it_is_atom,(eq,
       (get,(struc,cons_cell,form),cell_type,0),C_ATOM_HEDER))
    (jumpif,it_is_small,(eq,
       (get,(struc,cons_cell,form),cell_type,0),C_SNUM))
    (jumpif,it_is_atom,(eq,
       (get,(struc,cons_cell,form),cell_type,0),C_LAM))
    (jumpif,it_is_atom,(eq,
       (get,(struc,cons_cell,form),cell_type,0),C_FLAM))
    (jumpif,not_cons,(ne,
       (get,(struc,cons_cell,form),cell_type,0),C_CONS_CELL))
      (apush,form)
      (set,fn,(get,(struc,cons_cell,form),car_pointer,0))
      (set,fncell_type,(get,(struc,cons_cell,fn),cell_type,0))
      (set,fn,(get(struc,atom_header,fn),vcell,0))
      (apush,fncell_type)
      (apush,fn)
      (jpush,RESULT_HANDLER)
      (jpush,BODY_HANDLER)
      (jpush,ARG_HANDLER)
      (popj)
    (tagdef,it_is_atom)
      (put, (struc, atom_header, result), vcell, 0,
          (get,(struc,atom_header,form),vcell,0))
      (popj)
    (tagdef, it_is_small)
    (tagdef,not_cons)
      (put, (struc, atom_header, result), vcell, 0, form)
      (popj)
  (fnend,EVAL_0)
  (comment,**********************************************************)
  (fndef,EVAL)
    (set,arg,(apop))
    (set,arg,(get,(struc,cons_cell,arg),car_pointer,0))
    (apush,arg)
    (jpush,EVAL_0)
```

```
    (popj)
  (fnend,EVAL)
  (comment,**************************************************)
  (fndef,ARG_HANDLER)
    (set,f_body,(apop))
    (set,fncell_type,(apop))
    (apush,fncell_type)
    (apush,f_body)
    (jumpif,do_EVAL,(eq,fncell_type,C_LAM))
    (jumpif,dont_EVAL,(eq,fncell_type,C_FLAM))
    (jumpif,do_EVAL,(eq,fncell_type,C_SUB))
    (jumpif,dont_EVAL,(eq,fncell_type,C_FSUB))
    (jpush,FN_ERROR)
    (popj)
    (tagdef,do_EVAL)
      (jpush,EVAL_ARG)
      (popj)
    (tagdef,dont_EVAL)
      (jpush,NEVAL_ARG)
      (popj)
  (fnend,arg_handler)
  (comment,**************************************************)
  (fndef,FN_ERROR)
    (inc,lisp_level)
    (fwrite,C_OUTUNIT,(string,UNKNOWN FUNCTION....),20)
    (set,temp,(apop))
    (set,temp,(apop))
    (set,temp,(apop))
    (apush,C_MARK_1)
    (apush,temp)
    (set,temp,(jpop))
    (set,temp,(jpop))
    (jpush,C_MARK_1)
    (jpush,LISP_1)
    (jpush,PRINT_1)
    (popj)
  (fnend,FN_ERROR)
  (fndef,RESET)
    (set,lisp_level,2)
    (initspace,apdl)
    (initspace,jpdl)
    (jpush,LISP_1)
    (popj)
  (fnend,reset)
  (fndef,RETURN)
    (set,arg,(apop))
    (jumpif,okret,(ne,lisp_level,2))
    (fwrite,C_OUTUNIT,(string,CANNOT RETURN0,14)
    (put, (struc, atom_header, result), vcell, 0, nil)
    (popj)
    (tagdef,okret)
    (set,arg,(get,(struc,cons_cell,arg),car_pointer,0))
    (set,temp,(jpop))
```

```
    (tagdef,lpret1)
      (jumpif,donret1,(eq,temp,C_MARK_1))
      (set,temp,(jpop))
      (jump,lpret1)
    (tagdef,donret1)
    (set,temp,(apop))
    (tagdef,lpret2)
      (jumpif,donret2,(eq,temp,C_MARK_1))
      (set,temp,(apop))
      (jump,lpret2)
    (tagdef,donret2)
    (dec,lisp_level)
    (put, (struc, atom_header, result), vcell, 0, arg)
    (popj)
(fnend,RETURN)
(comment,**********************************************************)
(comment,**********************************************************)
(fndef,BODY_HANDLER)
    (set,pars,(apop))
    (set,f_body,(apop))
    (set,fncell_type,(apop))
    (apush,C_MARK)
    (jumpif,binary,(eq,fncell_type,C_SUB))
    (jumpif,binary,(eq,fncell_type,C_FSUB))
      (set,args,(get,(struc,cons_cell,f_body),car_pointer,0))
      (set,f_body,(get,(struc,cons_cell,f_body),cdr_pointer,0))
      (apush,f_body)
      (apush,args)
      (apush,pars)
      (jpush,PROGN)
      (jpush,DO_BINDINGS)
      (popj)
    (tagdef,binary)
      (apush,pars)
      (jpush,f_body)
      (popj)
(fnend,body_handler)
(comment,**********************************************************)
(fndef,RESULT_HANDLER)
(loop,spin)
    (set,f_par,(apop))
    (jumpif,outr,(eq,f_par,C_MARK))
      (set,old_val,(apop))
      (put,(struc,atom_header,f_par),vcell,0,old_val)
(loopend,spin)
(tagdef,outr)
      (popj)
(fnend,result_handler)
(comment,**********************************************************)
(fndef,EVAL_ARG)
    (set,f_body,(apop))
    (set,fncell_type,(apop))
    (set,call_form,(apop))
```

```
    (apush,fncell_type)
    (apush,f_body)
    (apush,(get,(struc,cons_cell,call_form),cdr_pointer,0))
    (jpush,PUSH_RESULT_ON_APDL)
    (jpush,EVLIS)
    (popj)
  (fnend,EVAL_arg)
  (comment,**********************************************************)
  (fndef,NEVAL_ARG)
    (set,f_body,(apop))
    (set,fncell_type,(apop))
    (set,call_form,(apop))
    (apush,fncell_type)
    (apush,f_body)
    (apush,(get,(struc,cons_cell,call_form),cdr_pointer,0))
    (popj)
  (fnend,nEVAL_arg)
  (comment,**********************************************************)
  (fndef,PROGN)
    (set,f_body,(apop))
    (jumpif,donep,(eq,f_body,nil))
      (apush,(get,(struc,cons_cell,f_body),cdr_pointer,0))
      (apush,(get,(struc,cons_cell,f_body),car_pointer,0))
      (jpush,PROGN)
      (jpush,EVAL_0)
      (popj)
    (tagdef,donep)
      (popj)
  (fnend,progn)
  (comment,**********************************************************)
  (fndef,DO_BINDINGS)
    (set,apar,(apop))
    (set,fpar,(apop))
    (set,f_body,(apop))
    (tagdef,until_done)
      (jumpif,done_binding,(eq,apar,nil))
      (jumpif,done_binding,(eq,fpar,nil))
        (set,fpar1,(get,(struc,cons_cell,fpar),car_pointer,0))
        (set,fpar,(get,(struc,cons_cell,fpar),cdr_pointer,0))
        (set,apar1,(get,(struc,cons_cell,apar),car_pointer,0))
        (set,apar,(get,(struc,cons_cell,apar),cdr_pointer,0))
        (set,fparval,(get,(struc,atom_header,fpar1),vcell,0))
        (apush,fparval)
        (apush,fpar1)
        (put,(struc,atom_header,fpar1),vcell,0,apar1)
        (jump,until_done)
    (tagdef,done_binding)
    (apush,f_body)
    (popj)
  (fnend,do_binding)
  (comment,**********************************************************)
  (fndef,OR)
    (set,arg,(apop))
```

```
    (put, (struc, atom_header, result), vcell, 0, nil)
    (tagdef,loopor)
      (jumpif,doneor,(eq,arg,nil))
      (set,first,(get,(struc,cons_cell,arg),car_pointer,0))
      (set,arg,(get,(struc,cons_cell,arg),cdr_pointer,0))
      (jumpif,loopor,(eq,first,nil))
    (put, (struc, atom_header, result), vcell, 0, t)
    (tagdef,doneor)
      (popj)
  (fnend,or)
  (comment,**********************************************************)
  (fndef,AND)
    (set,arg,(apop))
    (put, (struc, atom_header, result), vcell, 0, t)
    (tagdef,loopand)
      (jumpif,doneand,(eq,arg,nil))
      (set,first,(get,(struc,cons_cell,arg),car_pointer,0))
      (set,arg,(get,(struc,cons_cell,arg),cdr_pointer,0))
      (jumpif,loopand,(ne,first,nil))
    (put, (struc, atom_header, result), vcell, 0, nil)
    (tagdef,doneand)
      (popj)
  (fnend,and)
  (comment,**********************************************************)
  (fndef,DE)
    (set,arg,(apop))
    (comment, name of the func)
    (set,fn,(get,(struc,cons_cell,arg),car_pointer,0))
    (comment, pars and body)
    (set,arg2,(get,(struc,cons_cell,arg),cdr_pointer,0))
    (put,(struc,cons_cell,fn),cell_type,0,C_LAM)
    (put,(struc,atom_header,fn),vcell,0,arg2)
    (put, (struc, atom_header, result), vcell, 0, fn)
    (popj)
  (fnend,de)
  (comment,**********************************************************)
  (fndef,DF)
    (set,arg,(apop))
    (set,fn,(get,(struc,cons_cell,arg),car_pointer,0))
    (set,arg2,(get,(struc,cons_cell,arg),cdr_pointer,0))
    (put,(struc,cons_cell,fn),cell_type,0,C_FLAM)
    (put,(struc,atom_header,fn),vcell,0,arg2)
    (put, (struc, atom_header, result), vcell, 0, fn)
    (popj)
  (fnend,df)
  (comment,**********************************************************)
  (fndef,LIST)
    (put, (struc, atom_header, result), vcell, 0, (apop))
    (popj)
  (fnend,list)
  (comment,**********************************************************)
  (fndef,COND)
    (set,cond_body,(apop))
```

```
    (jumpif,cond_cont,(ne,cond_body,nil))
    (put, (struc, atom_header, result), vcell, 0, nil)
    (popj)
    (tagdef,cond_cont)
      (set,first_case,(get,(struc,cons_cell,cond_body),car_pointer,0))
      (apush,(get,(struc,cons_cell,cond_body),cdr_pointer,0))
      (apush,(get,(struc,cons_cell,first_case),cdr_pointer,0))
      (apush,(get,(struc,cons_cell,first_case),car_pointer,0))
      (jpush,COND_1)
      (jpush,EVAL_0)
      (popj)
(fnend,cond)
(comment,**********************************************************)
(fndef,QUOTE)
  (set,arg,(apop))
  (put, (struc, atom_header, result), vcell, 0,
     (get,(struc,cons_cell,arg),car_pointer,0))
  (popj)
(fnend,quote)
(comment,**********************************************************)
(fndef,COND_1)
  (jumpif,not_nil,(ne,
       (get, (struc, atom_header, result), vcell, 0),nil))
    (set,temp,(apop)) (comment, get rid of the body)
    (jpush,COND)
    (popj)
  (tagdef,not_nil)
    (set,arg,(apop)) (comment, progn this)
    (set,arg1,(apop)) (comment, get rid of the rest of the cond stuff)
    (apush,arg)
    (jpush,PROGN)
    (popj)
(fnend,cond_1)
(comment,**********************************************************)
(fndef,EVLIS)
    (set,arg,(apop))
  (jumpif,cont_evlis,(ne,arg,nil))
    (put, (struc, atom_header, result), vcell, 0, nil)
    (popj)
  (tagdef,cont_evlis)
    (set,first,(get,(struc,cons_cell,arg),car_pointer,0))
    (set,rest,(get,(struc,cons_cell,arg),cdr_pointer,0))
    (apush,rest)
    (apush,first)
    (jpush,EVLIS_1)
    (jpush,EVAL_0)
    (popj)
(fnend,evlis)
(comment,**********************************************************)
(fndef,EVLIS_1)
  (set,rest,(apop))
  (apush,(get, (struc, atom_header, result), vcell, 0))
  (apush,nil)
```

```
    (srjump,cons_1)
    (apush,(get, (struc, atom_header, result), vcell, 0))
    (apush,rest)
    (apush,(get, (struc, atom_header, result), vcell, 0))
    (jpush,EVLIS_2)
    (popj)
  (fnend,evlis_1)
  (comment,**********************************************************)
  (fndef,EVLIS_2)
    (set,temp,(apop))
    (set,rest,(apop))
    (jumpif,done_evlis_2,(eq,rest,nil))
      (set,first,(get,(struc,cons_cell,rest),car_pointer,0))
      (set,rest,(get,(struc,cons_cell,rest),cdr_pointer,0))
      (apush,temp)
      (apush,rest)
      (apush,first)
      (jpush,EVLIS_3)
      (jpush,EVAL_0)
      (popj)
    (tagdef,done_evlis_2)
      (put, (struc, atom_header, result), vcell, 0, (apop))
      (popj)
  (fnend,evlis_2)
  (comment,**********************************************************)
  (fndef,EVLIS_3)
    (set,rest,(apop))
    (set,temp,(apop))
    (apush, (get, (struc, atom_header, result), vcell, 0))
    (apush,nil)
    (srjump,cons_1)
    (put,(struc,cons_cell,temp),cdr_pointer,0,
       (get, (struc, atom_header, result), vcell, 0))
    (apush,rest)
    (apush,(get, (struc, atom_header, result), vcell, 0))
    (jpush,EVLIS_2)
    (popj)
  (fnend,evlis_3)
  (comment,**********************************************************)
  (fndef,APPEND)
    (set,arg,(apop))
    (set,arg1,(get,(struc,cons_cell,arg),car_pointer,0))
    (set,arg2,(get,(struc,cons_cell,arg),cdr_pointer,0))
    (set,arg2,(get,(struc,cons_cell,arg2),car_pointer,0))
    (set,cell_type_1,(get,(struc,cons_cell,arg1),cell_type,0))
    (jumpif,ok_app,(eq,cell_type_1,C_CONS_CELL))
      (comment, check for error , later)
    (tagdef,ok_app)
      (apush,arg2)
      (apush,arg1)
      (jpush,APPEND_1)
      (jpush,PUSH_RESULT_ON_APDL)
      (jpush,FIND_END)
```

```
      (jpush,PUSH_RESULT_ON_APDL)
      (jpush,CP_LIST)
      (popj)
 (fnend,append)
 (comment,**********************************************************)
 (fndef,CP_LIST)
   (set,arg,(apop))
   (jumpif,cont_cp,(ne,arg,nil))
     (put, (struc, atom_header, result), vcell, 0, nil)
     (popj)
   (tagdef,cont_cp)
     (set,first,(get,(struc,cons_cell,arg),car_pointer,0))
     (set,rest,(get,(struc,cons_cell,arg),cdr_pointer,0))
     (apush,first)
     (apush,nil)
     (srjump,cons_1)
     (apush,(get, (struc, atom_header, result), vcell, 0))
     (apush,rest)
     (apush,(get, (struc, atom_header, result), vcell, 0))
     (jpush,CP_LIST_1)
     (popj)
 (fnend,copy_list)
 (comment,**********************************************************)
 (fndef,CP_LIST_1)
   (set,temp,(apop))
   (set,rest,(apop))
   (jumpif,done_cp,(eq,rest,nil))
     (set,first,(get,(struc,cons_cell,rest),car_pointer,0))
     (set,rest,(get,(struc,cons_cell,rest),cdr_pointer,0))
     (apush,rest)
     (apush,first)
     (apush,nil)
     (srjump,cons_1)
     (put,(struc,cons_cell,temp),cdr_pointer,0,
         (get, (struc, atom_header, result), vcell, 0))
     (apush,(get, (struc, atom_header, result), vcell, 0))
     (jpush,CP_LIST_1)
     (popj)
   (tagdef,done_cp)
     (put, (struc, atom_header, result), vcell, 0, (apop))
     (popj)
 (fnend,copy_list_1)
 (comment,**********************************************************)
 (fndef,LAST)
   (set,arg,(apop))
   (set,arg,(get,(struc,cons_cell,arg),car_pointer,0))
   (apush,arg)
    (comment, jpush,POPAPDL)
   (jpush,FIND_END)
   (popj)
 (fnend,last)
 (comment,**********************************************************)
 (fndef,CAR)
```

```
    (set,arg,(apop))
    (set,arg,(get,(struc,cons_cell,arg),car_pointer,0))
    (set,cell_type_1,(get,(struc,cons_cell,arg),cell_type,0))
    (jumpif,carok,(eq,cell_type_1,C_CONS_CELL))
    (fwrite,C_OUTUNIT,(string,TRIED TO TAKE CAR OF NON LIST....),33)
    (apush,arg)
    (jpush,RESET)
    (jpush,PRINT_1)
    (popj)
    (tagdef,carok)
    (put, (struc, atom_header, result), vcell, 0,
            (get,(struc,cons_cell,arg),car_pointer,0))
    (popj)
(fnend,car)
(comment,**********************************************************)
(fndef,CDR)
    (set,arg,(apop))
    (set,arg,(get,(struc,cons_cell,arg),car_pointer,0))
    (set,cell_type_1,(get,(struc,cons_cell,arg),cell_type,0))
    (jumpif,cdrok,(eq,cell_type_1,C_CONS_CELL))
    (jumpif,nilcdr,(eq,arg,nil))
    (fwrite,C_OUTUNIT,(string,TRIED TO TAKE CDR OF NON LIST....),33)
    (apush,arg)
    (jpush,RESET)
    (jpush,PRINT_1)
    (popj)
    (tagdef,cdrok)
      (put, (struc, atom_header, result), vcell, 0,
              (get,(struc,cons_cell,arg),cdr_pointer,0))
      (popj)
    (tagdef,nilcdr)
       (put, (struc, atom_header, result), vcell, 0, nil)
      (popj)
(fnend,cdr)
(comment,**********************************************************)
(fndef,CONS)
    (set,arg,(apop))
    (set,arg1,(get,(struc,cons_cell,arg),car_pointer,0))
    (set,arg2,(get,(struc,cons_cell,arg),cdr_pointer,0))
    (set,arg2,(get,(struc,cons_cell,arg2),car_pointer,0))
    (apush,arg1)
    (apush,arg2)
    (srjump,cons_1)
    (popj)
(fnend,cons)
(comment,**********************************************************)
(fndef,SET)
    (set,arg,(apop))
    (set,arg1,(get,(struc,cons_cell,arg),car_pointer,0))
    (set,arg2,(get,(struc,cons_cell,arg),cdr_pointer,0))
    (set,arg2,(get,(struc,cons_cell,arg2),car_pointer,0))
    (apush,arg1)
    (apush,arg2)
```

```
    (jpush,SET0)
    (popj)
  (fnend,set)
  (comment,**************************************************)
  (fndef,SETQ)
    (set,arg,(apop))
    (set,arg1,(get,(struc,cons_cell,arg),car_pointer,0))
    (set,arg2,(get,(struc,cons_cell,arg),cdr_pointer,0))
    (set,arg2,(get,(struc,cons_cell,arg2),car_pointer,0))
    (apush,arg1)
    (apush,arg2)
    (jpush,SET0)
    (jpush,PUSH_RESULT_ON_APDL)
    (jpush,EVAL_0)
    (popj)
  (fnend,setq)
  (comment,**************************************************)
  (fndef,SET0)
    (set,arg2,(apop))
    (set,arg1,(apop))
    (put,(struc,atom_header,arg1),vcell,0,arg2)
    (put, (struc, atom_header, result), vcell, 0, arg2)
    (popj)
  (fnend,set0)
  (comment,**************************************************)
  (fndef,NULL)
    (set,arg,(apop))
    (set,arg,(get,(struc,cons_cell,arg),car_pointer,0))
    (jumpif,isnil,(eq,arg,nil))
      (put, (struc, atom_header, result), vcell, 0, nil)
      (popj)
    (tagdef,isnil)
      (put, (struc, atom_header, result), vcell, 0, t)
      (popj)
  (fnend,null)
  (comment,**************************************************)
  (fndef,ATOM)
    (set,arg,(apop))
    (set,arg,(get,(struc,cons_cell,arg),car_pointer,0))
    (apush,arg)
    (jpush,ATOM_1)
    (popj)
  (fnend,atom)
  (comment,**************************************************)
  (fndef,EQ)
    (set,arg,(apop))
    (set,arg1,(get,(struc,cons_cell,arg),car_pointer,0))
    (set,arg2,(get,(struc,cons_cell,arg),cdr_pointer,0))
    (set,arg2,(get,(struc,cons_cell,arg2),car_pointer,0))
    (jumpif,noteq,(ne,arg1,arg2))
      (put, (struc, atom_header, result), vcell, 0, t)
      (popj)
    (tagdef,noteq)
```

```
      (put, (struc, atom_header, result), vcell, 0, nil)
      (popj)
(fnend,eq)
(comment,*****************************************************)
(fndef,RPLACD)
   (set,arg,(apop))
   (set,arg1,(get,(struc,cons_cell,arg),car_pointer,0))
   (set,arg2,(get,(struc,cons_cell,arg),cdr_pointer,0))
   (set,arg2,(get,(struc,cons_cell,arg2),car_pointer,0))
   (put,(struc,cons_cell,arg1),cdr_pointer,0,arg2)
   (put, (struc, atom_header, result), vcell, 0, arg1)
   (popj)
(fnend,replacd)
(comment,*****************************************************)
(fndef,RPLACA)
   (set,arg,(apop))
   (set,arg1,(get,(struc,cons_cell,arg),car_pointer,0))
   (set,arg2,(get,(struc,cons_cell,arg),cdr_pointer,0))
   (set,arg2,(get,(struc,cons_cell,arg2),car_pointer,0))
   (put,(struc,cons_cell,arg1),car_pointer,0,arg2)
   (put, (struc, atom_header, result), vcell, 0, arg1)
   (popj)
(fnend,replaca)
(comment,*****************************************************)
(fndef,NCONC)
   (set,arg,(apop))
   (set,arg1,(get,(struc,cons_cell,arg),car_pointer,0))
   (set,arg2,(get,(struc,cons_cell,arg),cdr_pointer,0))
   (set,arg2,(get,(struc,cons_cell,arg2),car_pointer,0))
   (apush,arg2)
   (apush,arg1)
   (jpush,NCONC_1)
   (popj)
(fnend,nconc)
(comment,*****************************************************)
(fndef,PUTP)
   (set,arg,(apop))
   (set,atomname,(get,(struc,cons_cell,arg),car_pointer,0))
   (set,arg,(get,(struc,cons_cell,arg),cdr_pointer,0))
   (set,propname,(get,(struc,cons_cell,arg),car_pointer,0))
   (set,arg1,(get,(struc,cons_cell,arg),cdr_pointer,0))
   (set,propval,(get,(struc,cons_cell,arg1),car_pointer,0))
   (set,search,(get,(struc,atom_header,atomname),plist,0))
   (tagdef,putp_1)
      (jumpif,putnf,(eq,search,nil))
      (set,temp,(get,(struc,cons_cell,search),car_pointer,0))
      (jumpif,putf,(eq,propname,temp))
      (set,search,(get,(struc,cons_cell,search),cdr_pointer,0))
      (set,search,(get,(struc,cons_cell,search),cdr_pointer,0))
      (jump,putp_1)
   (tagdef,putnf)
      (put,(struc,cons_cell,arg1),
            cdr_pointer,0,(get,(struc,atom_header,atomname),plist,0))
```

```
      (put,(struc,atom_header,atomname),plist,0,arg)
      (put, (struc, atom_header, result), vcell, 0, nil)
      (popj)
   (tagdef,putf)
      (set,temp,(get,(struc,cons_cell,search),cdr_pointer,0))
      (put,(struc,cons_cell,temp),car_pointer,0,propval)
      (put, (struc, atom_header, result), vcell, 0, nil)
      (popj)
(fnend,putp)
(comment,**************************************************)
(fndef,GETP)
   (set,arg,(apop))
   (set,atomname,(get,(struc,cons_cell,arg),car_pointer,0))
   (set,arg,(get,(struc,cons_cell,arg),cdr_pointer,0))
   (jumpif,getpall,(eq,arg,nil))
   (set,propname,(get,(struc,cons_cell,arg),car_pointer,0))
   (set,search,(get,(struc,atom_header,atomname),plist,0))
   (tagdef,getp1)
      (jumpif,getpnf,(eq,search,nil))
      (set,temp,(get,(struc,cons_cell,search),car_pointer,0))
      (jumpif,getpf,(eq,propname,temp))
      (set,search,(get,(struc,cons_cell,search),cdr_pointer,0))
      (set,search,(get,(struc,cons_cell,search),cdr_pointer,0))
      (jump,getp1)
   (tagdef,getpnf)
      (put, (struc, atom_header, result), vcell, 0, nil)
      (popj)
   (tagdef,getpf)
      (put, (struc, atom_header, result), vcell, 0,
            (get,(struc,cons_cell,search),cdr_pointer,0))
      (jpush,QUOTE)
      (jpush,PUSH_RESULT_ON_APDL)
      (popj)
   (tagdef,getpall)
      (put, (struc, atom_header, result), vcell, 0,
            (get,(struc,atom_header,atomname),plist,0))
      (popj)
(fnend,getp)
(comment, **********************************************)
(fndef,MKSMALL)
   (set, arg, (apop))
   (set, temp, (mkstr,atom_header,atom_sp))
   (put, (struc, atom_header, result), vcell, 0, temp)
   (put, (struc, atom_header,
      (get, (struc, atom_header, result), vcell, 0) ) , vcell, 0, arg)
   (put, (struc,atom_header,
      (get, (struc, atom_header, result), vcell, 0) )
         , cell_type, 0, C_SNUM)
   (popj)
(fnend,MKSMALL)
(comment, **********************************************)
(fndef,ADD)
   (set,arg,(apop))
```

```
   (set,arg1,(get,(struc,cons_cell,arg),car_pointer,0))
   (set,arg2,(get,(struc,cons_cell,arg),cdr_pointer,0))
   (set,arg2,(get,(struc,cons_cell,arg2),car_pointer,0))
   (set, arg1, (get,(struc, atom_header, arg1), vcell, 0))
   (set, arg2, (get,(struc, atom_header, arg2), vcell, 0))
   (put, (struc, atom_header, result), vcell, 0, (add, arg1, arg2))
   (apush,(get, (struc, atom_header, result), vcell, 0))
   (jpush,MKSMALL)
   (popj)
(fnend, add)
(comment, **********************************************)
(fndef,SUB)
   (set,arg,(apop))
   (set,arg1,(get,(struc,cons_cell,arg),car_pointer,0))
   (set,arg2,(get,(struc,cons_cell,arg),cdr_pointer,0))
   (set,arg2,(get,(struc,cons_cell,arg2),car_pointer,0))
   (set, arg1, (get,(struc, atom_header, arg1), vcell, 0))
   (set, arg2, (get,(struc, atom_header, arg2), vcell, 0))
   (put, (struc, atom_header, result), vcell, 0, (sub, arg1, arg2))
   (apush,(get, (struc, atom_header, result), vcell, 0))
   (jpush,MKSMALL)
   (popj)
(fnend, sub)
(comment, **********************************************)
(fndef,MUL)
   (set,arg,(apop))
   (set,arg1,(get,(struc,cons_cell,arg),car_pointer,0))
   (set,arg2,(get,(struc,cons_cell,arg),cdr_pointer,0))
   (set,arg2,(get,(struc,cons_cell,arg2),car_pointer,0))
   (set, arg1, (get,(struc, atom_header, arg1), vcell, 0))
   (set, arg2, (get,(struc, atom_header, arg2), vcell, 0))
   (put, (struc, atom_header, result), vcell, 0, (mul, arg1, arg2))
   (apush,(get, (struc, atom_header, result), vcell, 0))
   (jpush,MKSMALL)
   (popj)
(fnend, mul)
(comment, **********************************************)
(fndef,DIV)
   (set,arg,(apop))
   (set,arg1,(get,(struc,cons_cell,arg),car_pointer,0))
   (set,arg2,(get,(struc,cons_cell,arg),cdr_pointer,0))
   (set,arg2,(get,(struc,cons_cell,arg2),car_pointer,0))
   (set, arg1, (get,(struc, atom_header, arg1), vcell, 0))
   (set, arg2, (get,(struc, atom_header, arg2), vcell, 0))
   (put, (struc, atom_header, result), vcell, 0, (div, arg1, arg2))
   (apush,(get, (struc, atom_header, result), vcell, 0))
   (jpush,MKSMALL)
   (popj)
(fnend, div)
(comment, **********************************************)
(fndef,MOD)
   (set,arg,(apop))
   (set,arg1,(get,(struc,cons_cell,arg),car_pointer,0))
```

```
    (set,arg2,(get,(struc,cons_cell,arg),cdr_pointer,0))
    (set,arg2,(get,(struc,cons_cell,arg2),car_pointer,0))
    (set, arg1, (get,(struc, atom_header, arg1), vcell, 0))
    (set, arg2, (get,(struc, atom_header, arg2), vcell, 0))
    (put, (struc, atom_header, result), vcell, 0, (mod, arg1, arg2))
    (apush,(get, (struc, atom_header, result), vcell, 0))
    (jpush,MKSMALL)
    (popj)
 (fnend, mod)
 (comment, ***********************************************)
 (fndef,GNUM)
    (set,arg,(apop))
    (set,arg1,(get,(struc,cons_cell,arg),car_pointer,0))
    (set,arg2,(get,(struc,cons_cell,arg),cdr_pointer,0))
    (set,arg2,(get,(struc,cons_cell,arg2),car_pointer,0))
    (set, arg1, (get,(struc, atom_header, arg1), vcell, 0))
    (set, arg2, (get,(struc, atom_header, arg2), vcell, 0))
    (jumpif, gnumt, (gt, arg1, arg2))
      (put, (struc, atom_header, result), vcell, 0, nil)
      (popj)
    (tagdef, gnumt)
    (put, (struc, atom_header, result), vcell, 0, t)
    (popj)
 (fnend, GNUM)
 (comment,*****************************************************)
 (fndef,LNUM)
    (set,arg,(apop))
    (set,arg1,(get,(struc,cons_cell,arg),car_pointer,0))
    (set,arg2,(get,(struc,cons_cell,arg),cdr_pointer,0))
    (set,arg2,(get,(struc,cons_cell,arg2),car_pointer,0))
    (set, arg1, (get,(struc, atom_header, arg1), vcell, 0))
    (set, arg2, (get,(struc, atom_header, arg2), vcell, 0))
    (jumpif, lnumt, (lt, arg1, arg2))
      (put, (struc, atom_header, result), vcell, 0, nil)
      (popj)
    (tagdef, lnumt)
    (put, (struc, atom_header, result), vcell, 0, t)
    (popj)
 (fnend, LNUM)
 (comment,*****************************************************)
 (fndef,GENUM)
    (set,arg,(apop))
    (set,arg1,(get,(struc,cons_cell,arg),car_pointer,0))
    (set,arg2,(get,(struc,cons_cell,arg),cdr_pointer,0))
    (set,arg2,(get,(struc,cons_cell,arg2),car_pointer,0))
    (set, arg1, (get,(struc, atom_header, arg1), vcell, 0))
    (set, arg2, (get,(struc, atom_header, arg2), vcell, 0))
    (jumpif, genumt, (ge, arg1, arg2))
      (put, (struc, atom_header, result), vcell, 0, nil)
      (popj)
    (tagdef, genumt)
    (put, (struc, atom_header, result), vcell, 0, t)
    (popj)
```

```
(fnend, GENUM)
(comment,**************************************************)
(fndef,EQNUM)
  (set,arg,(apop))
  (set,arg1,(get,(struc,cons_cell,arg),car_pointer,0))
  (set,arg2,(get,(struc,cons_cell,arg),cdr_pointer,0))
  (set,arg2,(get,(struc,cons_cell,arg2),car_pointer,0))
  (set, arg1, (get,(struc, atom_header, arg1), vcell, 0))
  (set, arg2, (get,(struc, atom_header, arg2), vcell, 0))
  (jumpif, eqnumt, (eq, arg1, arg2))
    (put, (struc, atom_header, result), vcell, 0, nil)
    (popj)
  (tagdef, eqnumt)
  (put, (struc, atom_header, result), vcell, 0, t)
  (popj)
(fnend, EQNUM)
(comment,**************************************************)
(fndef,LENUM)
  (set,arg,(apop))
  (set,arg1,(get,(struc,cons_cell,arg),car_pointer,0))
  (set,arg2,(get,(struc,cons_cell,arg),cdr_pointer,0))
  (set,arg2,(get,(struc,cons_cell,arg2),car_pointer,0))
  (set, arg1, (get,(struc, atom_header, arg1), vcell, 0))
  (set, arg2, (get,(struc, atom_header, arg2), vcell, 0))
  (jumpif, lenumt, (le, arg1, arg2))
    (put, (struc, atom_header, result), vcell, 0, nil)
    (popj)
  (tagdef, lenumt)
  (put, (struc, atom_header, result), vcell, 0, t)
  (popj)
(fnend, LENUM)
(comment,**************************************************)
(fndef,EXIT)
  (fwrite,C_OUTUNIT,(string,CORVALLISP END0,15)
  (fread,inunit,(aget, buff, 0),1)
  (jpush,EXIT_FLAG)
  (popj)
(fnend,exit)
(comment,**************************************************)
(fndef, OBLIST)
  (set, arg, (apop))
  (set, loc, 0)
  (tagdef, ob_loop)
  (jumpif, doneob, (eq, loc, C_OBLIST_LEN))
  (set,(vbl,search),(aget,(space,oblist),(vbl,loc)))
  (inc, loc)
  (tagdef,(tag,obsloop))
    (jumpif,(tag,ob_loop),(eq,(vbl,search),nil))
    (set,(vbl,pointer),(get,(struc,cons_cell,search),
         (field,cdr_pointer),(integer,0)))

    (printf,"%d: %s, %x0,
        loc, &(aget,pnamesp,(get,(struc,atom_header,pointer),pname,0))
```

```
        ,(get, (struc, atom_header, pointer), vcell, 0));
      (set,(vbl,search),(get,(struc,cons_cell,search),
          (field,car_pointer),(integer,0)))
      (jump,(tag,obsloop))
    (tagdef, doneob)
    (put, (struc, atom_header, result), vcell, 0, t)
    (popj)
(fnend, OBLIST)

(codeend,lisp)
(srdef,cons_1)
    (vset,aarg2,(apop))
    (vset,aarg1,(apop))
    (srjump,mkcons)
    (put, (struc, atom_header, result), vcell, 0, (apop))
    (put,(struc,cons_cell,
      (get, (struc, atom_header, result), vcell, 0)),
          car_pointer,0,(vget,aarg1))
    (put,(struc,cons_cell,
      (get, (struc, atom_header, result), vcell, 0)),
          cdr_pointer,0,(vget,aarg2))
    (ret)
(srend,cons_1)
(comment,********** INIT. routine **********)
(srdef,init)
    (initspace,atom_sp)
    (initspace,cons_sp)
    (initspace,strg_sp)
    (initspace,apdl)
    (initspace,jpdl)
    (initspace,pnamesp)
    (set,nil,(mkstr,atom_header,atom_sp))
    (set,keep_cons,(mkstr,atom_header,atom_sp))
    (set,keeper,(mkstr,atom_header,atom_sp))
    (set,keeper2,(mkstr,atom_header,atom_sp))
    (set,aarg1,(mkstr,atom_header,atom_sp))
    (set,aarg2,(mkstr,atom_header,atom_sp))
    (initspace,atom_sp)
    (set, inunit, C_INUNIT);
    (set, outunit, C_OUTUNIT);
(comment, LINK UP THE CONS CELL SPACE)
    (set,arg,(mkstr,cons_cell,cons_sp))
    (set, gc_warn, 0)
(tagdef,initlop)
    (inc, gc_warn)
    (jumpz,doneinit,(checkspace,cons_cell,cons_sp))
    (set,temp,(mkstr,cons_cell,cons_sp))
    (put,(struc,cons_cell,arg),car_pointer,0,temp)
    (set,arg,temp)
    (jump,initlop)
    (tagdef,doneinit)
      (set, gc_warn, (div, gc_warn, 4))
      (initspace,cons_sp)
```

```
    (set,avail_cons,(mkstr,cons_cell,cons_sp))
    (put,(struc,cons_cell,arg),car_pointer,0,nil)
  (set,loc_pnsp,0)
  (set,lisp_level,2)
  (set,pass,(mkstr,string_args,strg_sp))
  (set,counter,0)
  (tagdef,initloop)
    (jumpif,donelp,(eq,counter,C_OBLIST_LEN))
  (aput,oblist,counter,nil)
  (set,counter,(add,counter,1))
  (jump,initloop)
 (tagdef,donelp)
  (scopy,(string,nil),(get,(struc,string_args,pass),st,0))
       (apush,pass)
  (srjump,intern)
  (set,arg,(apop))
  (set,nil,arg)
  (put,(struc,atom_header,arg),vcell,0,nil)
  (put,(struc,cons_cell,arg),cell_type,0,C_ATOM_HEDER)
  (scopy,(string,keep_cons),(get,(struc,string_args,pass),st,0))
       (apush,pass)
  (srjump,intern)
  (set,arg,(apop))
  (set,keep_cons,arg)
  (put,(struc,atom_header,arg),vcell,0,nil)
  (put,(struc,cons_cell,arg),cell_type,0,C_ATOM_HEDER)
  (vinit, keeper)
  (vinit, keeper2)
  (vinit, aarg1)
  (vinit, aarg2)
  (scopy,(string,t),(get,(struc,string_args,pass),st,0))
       (apush,pass)
  (srjump,intern)
  (set,arg,(apop))
  (set,t,arg)
  (put,(struc,atom_header,arg),vcell,0,t)
  (put,(struc,cons_cell,arg),cell_type,0,C_ATOM_HEDER)
  (scopy,(string,result),(get,(struc,string_args,pass),st,0))
       (apush,pass)
  (srjump,intern)
  (set,arg,(apop))
  (set,result,arg)
  (put,(struc,atom_header,arg),vcell,0,t)
  (put,(struc,cons_cell,arg),cell_type,0,C_ATOM_HEDER)
  (scopy,(string,front),(get,(struc,string_args,pass),st,0))
       (apush,pass)
  (srjump,intern)
  (set,arg,(apop))
  (set,front,arg)
  (put,(struc,atom_header,arg),vcell,0,t)
  (put,(struc,cons_cell,arg),cell_type,0,C_ATOM_HEDER)
  (scopy,(string,oblist),(get,(struc,string_args,pass),st,0))
       (apush,pass)
```

```
(srjump,intern)
(set,arg,(apop))
(put,(struc,atom_header,arg),vcell,0,OBLIST)
(put,(struc,cons_cell,arg),cell_type,0,C_FSUB)
(scopy,(string,gc),(get,(struc,string_args,pass),st,0))
     (apush,pass)
 (srjump,intern)
 (set,arg,(apop))
 (put,(struc,atom_header,arg),vcell,0,GC)
 (put,(struc,cons_cell,arg),cell_type,0,C_FSUB)
(scopy,(string,room),(get,(struc,string_args,pass),st,0))
     (apush,pass)
 (srjump,intern)
 (set,arg,(apop))
 (put,(struc,atom_header,arg),vcell,0,ROOM)
 (put,(struc,cons_cell,arg),cell_type,0,C_FSUB)
(scopy,(string,putp),(get,(struc,string_args,pass),st,0))
     (apush,pass)
 (srjump,intern)
 (set,arg,(apop))
 (put,(struc,atom_header,arg),vcell,0,PUTP)
 (put,(struc,cons_cell,arg),cell_type,0,C_SUB)
 (scopy,(string,=),(get,(struc,string_args,pass),st,0))
      (apush,pass)
(srjump,intern)
(set,arg,(apop))
(put,(struc,atom_header,arg),vcell,0,EQNUM)
(put,(struc,cons_cell,arg),cell_type,0,C_SUB)
(scopy,(string,? =),(get,(struc,string_args,pass),st,0))
     (apush,pass)
(srjump,intern)
(set,arg,(apop))
(put,(struc,atom_header,arg),vcell,0,LENUM)
(put,(struc,cons_cell,arg),cell_type,0,C_SUB)
(scopy,(string, =),(get,(struc,string_args,pass),st,0))
     (apush,pass)
(srjump,intern)
(set,arg,(apop))
(put,(struc,atom_header,arg),vcell,0,GENUM)
(put,(struc,cons_cell,arg),cell_type,0,C_SUB)
(scopy,(string,? ),(get,(struc,string_args,pass),st,0))
     (apush,pass)
(srjump,intern)
(set,arg,(apop))
(put,(struc,atom_header,arg),vcell,0,LNUM)
(put,(struc,cons_cell,arg),cell_type,0,C_SUB)
(scopy,(string, ),(get,(struc,string_args,pass),st,0))
     (apush,pass)
(srjump,intern)
(set,arg,(apop))
(put,(struc,atom_header,arg),vcell,0,GNUM)
(put,(struc,cons_cell,arg),cell_type,0,C_SUB)
(scopy,(string,%),(get,(struc,string_args,pass),st,0))
```

```
        (apush,pass)
    (srjump,intern)
    (set,arg,(apop))
    (put,(struc,atom_header,arg),vcell,0,MOD)
    (put,(struc,cons_cell,arg),cell_type,0,C_SUB)
    (scopy,(string,/),(get,(struc,string_args,pass),st,0))
        (apush,pass)
    (srjump,intern)
    (set,arg,(apop))
    (put,(struc,atom_header,arg),vcell,0,DIV)
    (put,(struc,cons_cell,arg),cell_type,0,C_SUB)
    (scopy,(string,+),(get,(struc,string_args,pass),st,0))
        (apush,pass)
    (srjump,intern)
    (set,arg,(apop))
    (put,(struc,atom_header,arg),vcell,0,ADD)
    (put,(struc,cons_cell,arg),cell_type,0,C_SUB)
    (scopy,(string,-),(get,(struc,string_args,pass),st,0))
        (apush,pass)
    (srjump,intern)
    (set,arg,(apop))
    (put,(struc,atom_header,arg),vcell,0,SUB)
    (put,(struc,cons_cell,arg),cell_type,0,C_SUB)
    (scopy,(string,*),(get,(struc,string_args,pass),st,0))
        (apush,pass)
    (srjump,intern)
    (set,arg,(apop))
    (put,(struc,atom_header,arg),vcell,0,MUL)
    (put,(struc,cons_cell,arg),cell_type,0,C_SUB)
    (scopy,(string,getp),(get,(struc,string_args,pass),st,0))
        (apush,pass)
    (srjump,intern)
    (set,arg,(apop))
    (put,(struc,atom_header,arg),vcell,0,GETP)
    (put,(struc,cons_cell,arg),cell_type,0,C_SUB)
(scopy,(string,reset),(get,(struc,string_args,pass),st,0))
      (apush,pass)
(srjump,intern)
(set,arg,(apop))
(put,(struc,atom_header,arg),vcell,0,RESET)
(put,(struc,cons_cell,arg),cell_type,0,C_FSUB)
  (scopy,(string,return),(get,(struc,string_args,pass),st,0))
        (apush,pass)
  (srjump,intern)
  (set,arg,(apop))
  (put,(struc,atom_header,arg),vcell,0,RETURN)
  (put,(struc,cons_cell,arg),cell_type,0,C_SUB)
  (scopy,(string,EVAL),(get,(struc,string_args,pass),st,0))
        (apush,pass)
  (srjump,intern)
  (set,arg,(apop))
  (put,(struc,atom_header,arg),vcell,0,EVAL)
  (put,(struc,cons_cell,arg),cell_type,0,C_SUB)
```

```
(scopy,(string,progn),(get,(struc,string_args,pass),st,0))
      (apush,pass)
(srjump,intern)
(set,arg,(apop))
(put,(struc,atom_header,arg),vcell,0,PROGN)
(put,(struc,cons_cell,arg),cell_type,0,C_FSUB)
(scopy,(string,cond),(get,(struc,string_args,pass),st,0))
      (apush,pass)
(srjump,intern)
(set,arg,(apop))
(put,(struc,atom_header,arg),vcell,0,COND)
(put,(struc,cons_cell,arg),cell_type,0,C_FSUB)
(scopy,(string,evlis),(get,(struc,string_args,pass),st,0))
      (apush,pass)
(srjump,intern)
(set,arg,(apop))
(put,(struc,atom_header,arg),vcell,0,EVLIS)
(put,(struc,cons_cell,arg),cell_type,0,C_FSUB)
(scopy,(string,append),(get,(struc,string_args,pass),st,0))
      (apush,pass)
(srjump,intern)
(set,arg,(apop))
(put,(struc,atom_header,arg),vcell,0,APPEND)
(put,(struc,cons_cell,arg),cell_type,0,C_SUB)
(scopy,(string,last),(get,(struc,string_args,pass),st,0))
      (apush,pass)
(srjump,intern)
(set,arg,(apop))
(put,(struc,atom_header,arg),vcell,0,LAST)
(put,(struc,cons_cell,arg),cell_type,0,C_SUB)
(scopy,(string,car),(get,(struc,string_args,pass),st,0))
      (apush,pass)
(srjump,intern)
(set,arg,(apop))
(put,(struc,atom_header,arg),vcell,0,CAR)
(put,(struc,cons_cell,arg),cell_type,0,C_SUB)
(scopy,(string,cdr),(get,(struc,string_args,pass),st,0))
      (apush,pass)
(srjump,intern)
(set,arg,(apop))
(put,(struc,atom_header,arg),vcell,0,CDR)
(put,(struc,cons_cell,arg),cell_type,0,C_SUB)
(scopy,(string,cons),(get,(struc,string_args,pass),st,0))
      (apush,pass)
(srjump,intern)
(set,arg,(apop))
(put,(struc,atom_header,arg),vcell,0,CONS)
(put,(struc,cons_cell,arg),cell_type,0,C_SUB)
(scopy,(string,SET),(get,(struc,string_args,pass),st,0))
      (apush,pass)
(srjump,intern)
(set,arg,(apop))
(put,(struc,atom_header,arg),vcell,0,SET)
```

```
(put,(struc,cons_cell,arg),cell_type,0,C_SUB)
(scopy,(string,setq),(get,(struc,string_args,pass),st,0))
      (apush,pass)
(srjump,intern)
(set,arg,(apop))
(put,(struc,atom_header,arg),vcell,0,SETQ)
(put,(struc,cons_cell,arg),cell_type,0,C_FSUB)
(scopy,(string,null),(get,(struc,string_args,pass),st,0))
      (apush,pass)
(srjump,intern)
(set,arg,(apop))
(put,(struc,atom_header,arg),vcell,0,NULL)
(put,(struc,cons_cell,arg),cell_type,0,C_SUB)
(scopy,(string,atom),(get,(struc,string_args,pass),st,0))
      (apush,pass)
(srjump,intern)
(set,arg,(apop))
(put,(struc,atom_header,arg),vcell,0,ATOM)
(put,(struc,cons_cell,arg),cell_type,0,C_SUB)
(scopy,(string,EQ),(get,(struc,string_args,pass),st,0))
      (apush,pass)
(srjump,intern)
(set,arg,(apop))
(put,(struc,atom_header,arg),vcell,0,EQ)
(put,(struc,cons_cell,arg),cell_type,0,C_SUB)
(scopy,(string,rplacd),(get,(struc,string_args,pass),st,0))
      (apush,pass)
(srjump,intern)
(set,arg,(apop))
(put,(struc,atom_header,arg),vcell,0,RPLACD)
(put,(struc,cons_cell,arg),cell_type,0,C_SUB)
(scopy,(string,rplaca),(get,(struc,string_args,pass),st,0))
      (apush,pass)
(srjump,intern)
(set,arg,(apop))
(put,(struc,atom_header,arg),vcell,0,RPLACA)
(put,(struc,cons_cell,arg),cell_type,0,C_SUB)
(scopy,(string,nconc),(get,(struc,string_args,pass),st,0))
      (apush,pass)
(srjump,intern)
(set,arg,(apop))
(put,(struc,atom_header,arg),vcell,0,NCONC)
(put,(struc,cons_cell,arg),cell_type,0,C_SUB)
(scopy,(string,de),(get,(struc,string_args,pass),st,0))
      (apush,pass)
(srjump,intern)
(set,arg,(apop))
(put,(struc,atom_header,arg),vcell,0,DE)
(put,(struc,cons_cell,arg),cell_type,0,C_FSUB)
(scopy,(string,df),(get,(struc,string_args,pass),st,0))
      (apush,pass)
(srjump,intern)
(set,arg,(apop))
```

```
    (put,(struc,atom_header,arg),vcell,0,DF)
    (put,(struc,cons_cell,arg),cell_type,0,C_FSUB)
    (scopy,(string,quote),(get,(struc,string_args,pass),st,0))
          (apush,pass)
    (srjump,intern)
    (set,arg,(apop))
    (put,(struc,atom_header,arg),vcell,0,QUOTE)
    (put,(struc,cons_cell,arg),cell_type,0,C_FSUB)
    (scopy,(string,list),(get,(struc,string_args,pass),st,0))
          (apush,pass)
    (srjump,intern)
    (set,arg,(apop))
    (put,(struc,atom_header,arg),vcell,0,LIST)
    (put,(struc,cons_cell,arg),cell_type,0,C_SUB)
    (scopy,(string,or),(get,(struc,string_args,pass),st,0))
          (apush,pass)
    (srjump,intern)
    (set,arg,(apop))
    (put,(struc,atom_header,arg),vcell,0,OR)
    (put,(struc,cons_cell,arg),cell_type,0,C_SUB)
    (scopy,(string,and),(get,(struc,string_args,pass),st,0))
          (apush,pass)
    (srjump,intern)
    (set,arg,(apop))
    (put,(struc,atom_header,arg),vcell,0,AND)
    (put,(struc,cons_cell,arg),cell_type,0,C_SUB)
  (comment,**********************************************)
    (scopy,(string,load),(get,(struc,string_args,pass),st,0))
    (apush,pass)
    (srjump,intern)
    (set,arg,(apop))
    (put,(struc,atom_header,arg),vcell,0,LOAD)
    (put,(struc,cons_cell,arg),cell_type,0,C_FSUB)
    (scopy,(string,read),(get,(struc,string_args,pass),st,0))
          (apush,pass)
    (srjump,intern)
    (set,arg,(apop))
    (put,(struc,atom_header,arg),vcell,0,READ)
    (put,(struc,cons_cell,arg),cell_type,0,C_FSUB)
    (scopy,(string,print),(get,(struc,string_args,pass),st,0))
          (apush,pass)
    (srjump,intern)
    (set,arg,(apop))
    (put,(struc,atom_header,arg),vcell,0,PRINT)
    (put,(struc,cons_cell,arg),cell_type,0,C_SUB)
    (scopy,(string,exit),(get,(struc,string_args,pass),st,0))
          (apush,pass)
    (srjump,intern)
    (set,arg,(apop))
    (put,(struc,atom_header,arg),vcell,0,EXIT)
    (put,(struc,cons_cell,arg),cell_type,0,C_FSUB)
  (srend,init)
  (srdef,hasher)
```

```
  (set,(vbl,counter),(integer,0))
  (set,(vbl,argu),(apop))
  (scopy,(get,(struc,string_args,argu),
    (field,st),(integer,0)),(aget,(space,print),(integer,0)))
  (set,(vbl,hashval),(integer,0))
  (tagdef,hloop)
    (jumpif,out,(eq,(aget,(space,print),(vbl,counter)),(character, )))
    (set,(vbl,hashval),(add,(vbl,hashval),
        (aget,(space,print),(vbl,counter))))
    (set,(vbl,counter),(add,(vbl,counter),(integer,1)))
    (jump,hloop)
  (tagdef,out)
  (set,(vbl,hashval),(mod,(vbl,hashval),(const,C_OBLIST_LEN)))
  (inc,counter)
  (apush,counter)
  (apush,(vbl,hashval))
(srend,hasher)

(srdef,intern)
  (set,(vbl,arg),(apop))
  (comment, Check for small numbers, They are not internrd )
  (set, counter, 0)
  (set, temp, 0)
  (set, sign, 1)
  (jumpif, check_snum, (ne,
    (get, (struc, string_args, arg),st,counter), '-'))
    (set, sign, -1)
    (inc, counter)
    (jumpif,not_snum,(gt,
        (get, (struc, string_args, arg),st,counter), '9'))
    (jumpif,not_snum,(lt,
        (get, (struc, string_args, arg),st,counter), '0'))
  (tagdef, (tag, check_snum))
    (jumpif,snum_done,(eq,
        (get, (struc, string_args, arg),st,counter), ' '))
    (jumpif,not_snum,(gt,
        (get, (struc, string_args, arg),st,counter), '9'))
    (jumpif,not_snum,(lt,
        (get, (struc, string_args, arg),st,counter), '0'))
        (set, temp,
          (add, (mul, temp, 10),
              (sub, (get, (struc, string_args, arg),st, counter), '0')))
        (inc, counter)
        (jump, (tag, check_snum))
  (tagdef, snum_done)
    (set, temp, (mul, temp, sign))
    (set,pointer,(mkstr,atom_header,atom_sp))
    (put, (struc, atom_header, pointer), vcell, 0, temp)
    (put, (struc,atom_header, pointer), cell_type, 0, C_SNUM)
    (jump, found)
  (tagdef, (tag, not_snum))
  (set, counter, 0)
  (apush,(vbl,arg))  (comment, pass argument to hasher)
```

```
    (srjump,hasher)
    (set,(vbl,loc),(apop)) (comment, get loc from hasher)
    (set,counter,(apop))
    (set,(vbl,search),(aget,(space,oblist),(vbl,loc)))
    (set,(vbl,old_search),(vbl,search))
    (tagdef,(tag,sloop))
      (jumpif,(tag,notfound),(eq,(vbl,search),nil))
      (set,(vbl,pointer),(get,(struc,cons_cell,search),
           (field,cdr_pointer),(integer,0)))
      (set,ploc,(get,(struc,atom_header,pointer),pname,0))
      (set,a_found,(scomp,(aget,pnamesp,ploc),
           (get,(struc,string_args,arg),st,0)))
      (jumpz,(tag,found),(vbl,a_found))
      (set,(vbl,old_search),(vbl,search))
      (set,(vbl,search),(get,(struc,cons_cell,search),
           (field,car_pointer),(integer,0)))
      (jump,(tag,sloop))

    (tagdef,(tag,notfound))
      (srjump,mkcons)
      (set,temp,(apop))
      (put,(struc,cons_cell,temp),(field,car_pointer),(integer,0),nil)
      (set,arg1,(mkstr,atom_header,atom_sp))
      (scopy,(get,(struc,string_args,arg),st,0),(aget,pnamesp,loc_pnsp))
      (put,(struc,atom_header,arg1),pname,0,loc_pnsp)
      (set,loc_pnsp,(add,loc_pnsp,counter))
      (put,(struc,cons_cell,temp),
          (field,cdr_pointer),(integer,0),(vbl,arg1))
      (put,(struc,atom_header,arg1),(field,vcell),(integer,0),nil)
      (put,(struc,cons_cell,arg1),cell_type,0,C_ATOM_HEDER)
      (put,(struc,atom_header,arg1),plist,0,nil)
      (jumpif,(tag,emptybucket),(eq,(vbl,search),(vbl,old_search)))
        (put,(struc,cons_cell,old_search),
            (field,car_pointer),(integer,0),(vbl,temp))
        (set,(vbl,pointer),(vbl,arg1))
        (jump,(tag,found))
      (tagdef,(tag,emptybucket))
        (aput,(space,oblist),(vbl,loc),(vbl,temp))
        (set,(vbl,pointer),(vbl,arg1))
    (tagdef,(tag,found))
      (apush,(vbl,pointer))
  (srend,intern)
  (srdef,mkcons)
    (jumpif,nosp,(eq,avail_cons,nil))
    (tagdef,makecell)
    (dec, gc_warn)
    (put,(struc,cons_cell,avail_cons),cell_type,0,C_CONS_CELL)
    (put,(struc,atom_header, keep_cons),vcell,0,avail_cons)
    (apush,avail_cons)
    (set,avail_cons,(get,(struc,cons_cell,avail_cons),car_pointer,0))
    (put, (struc, cons_cell,
      (get, (struc, atom_header, keep_cons), vcell, 0)),
          car_pointer, 0, nil)
```

```
    (put, (struc, cons_cell,
       (get, (struc, atom_header, keep_cons), vcell, 0)),
           cdr_pointer, 0, nil)
    (ret)
  (tagdef,nosp)
    (fwrite,C_OUTUNIT,(string,OUT OF CONS CELL SPACE GC CALLED0,33)
    (srjump,gc_cons)
    (fwrite,C_OUTUNIT,(string,DONE GC0,8)
    (jumpnif,makecell,(eq,avail_cons,nil))
    (fwrite,C_OUTUNIT,(string,CANNOT GC CONS_SPACE0,21)
    (jpush,EXIT_FLAG)
    (ret)
  (srend,mkcons)
  (srdef,gc_cons)
    (initspace,cons_sp)
    (tagdef,put1)
      (jumpz,doneput1,(checkspace,cons_cell,cons_sp))
      (set,gctemp,(mkstr,cons_cell,cons_sp))
      (put,(struc,common_fields,gctemp),gc,0,1)
      (jump,put1)
      (tagdef,doneput1)
    (srjump, mark_atoms)
    (srjump,mark_cons)
    (srjump,mark_apdl)
    (set,last,nil)
    (set,avail_cons,nil)
    (initspace,cons_sp)
    (set, roomcnt, 0)
    (tagdef,loopgc)
      (jumpz,donegc,(checkspace,cons_cell,cons_sp))
      (set,gctemp,(mkstr,cons_cell,cons_sp))
      (jumpz,loopgc,(get,(struc,common_fields,gctemp),gc,0))
      (jumpif,notfirst,(ne,avail_cons,nil))
          (inc, roomcnt)
        (set,avail_cons,gctemp)
        (set,last,gctemp)
        (put,(struc,cons_cell,avail_cons),car_pointer,0,nil)
        (jump,loopgc)
      (tagdef,notfirst)
          (inc, roomcnt)
        (put,(struc,cons_cell,last),car_pointer,0,gctemp)
        (set,last,gctemp)
        (put,(struc,cons_cell,gctemp),car_pointer,0,nil)
        (jump,loopgc)
      (tagdef,donegc)
      (set, gc_warn, (div, roomcnt, 4))
          (printf, "GC: Room %d availcons = %x0, roomcnt, avail_cons);
        (ret)
  (srend,gc_cons)
  (srdef,mark_cons)
      (set,gccounter,0)
      (tagdef,loopmark)
          (jumpif,donemark,(eq,gccounter,C_OBLIST_LEN))
```

```
        (set,gcsearch,(aget,oblist,gccounter))
        (inc,gccounter)
        (jumpif,loopmark,(eq,gcsearch,nil))
        (apush,C_MARK)
        (apush,gcsearch)
        (tagdef,lpmark)
          (set,gcsearch,(apop))
          (jumpif,loopmark,(eq,gcsearch,C_MARK))
          (jumpif,lpmark,(eq,
                (get,(struc,cons_cell,gcsearch),gc,0),0))
          (put,(struc,cons_cell,gcsearch),gc,0,0) (comment, do no gc)
          (set,cell_type_1,
                (get,(struc,cons_cell,gcsearch),cell_type,0))
          (jumpif,lpmark,(eq,cell_type_1,C_FSUB))
          (jumpif,lpmark,(eq,cell_type_1,C_SUB))
          (jumpif,lpmark,(eq,cell_type_1,C_SNUM))
          (jumpif,lpmark,(eq,gcsearch,nil))
          (jumpif,lpmark,(eq,gcsearch,t))
          (jumpif,not_cons_cell,(eq,cell_type_1,C_FLAM))
          (jumpif,not_cons_cell,(eq,cell_type_1,C_LAM))
            (jumpif,not_cons_cell,(eq, cell_type_1, C_ATOM_HEDER))
            (jump, is_cons_cell)
            (tagdef, not_cons_cell)
                (apush, (get, (struc, atom_header, gcsearch), vcell, 0))
            (jump, lpmark)
            (tagdef, is_cons_cell)
          (apush,(get,(struc,cons_cell,gcsearch),cdr_pointer,0))
          (apush,(get,(struc,cons_cell,gcsearch),car_pointer,0))
        (jump,lpmark)
      (tagdef,donemark)
        (ret)
(srend,mark_cons)
(srdef, mark_atoms)
  (set, loc, 0)
  (tagdef, ob_loop)
  (jumpif, doneob, (eq, loc, C_OBLIST_LEN))
  (set,(vbl,search),(aget,(space,oblist),(vbl,loc)))
  (inc, loc)
  (tagdef,(tag,obsloop))
    (jumpif,(tag,ob_loop),(eq,(vbl,search),nil))
    (set,(vbl,pointer),(get,(struc,cons_cell,search),
        (field,cdr_pointer),(integer,0)))
    (set,(vbl,search),(get,(struc,cons_cell,search),
        (field,car_pointer),(integer,0)))
    (put, (struc, atom_header, pointer), gc, 0, 1)
    (jump,(tag,obsloop))
  (tagdef, doneob)
  (ret)
(srend, mark_atoms)
(srdef,mark_apdl)
(vdef, saveapdl)
    (set,gccounter,0)
    (set, saveapdl, add_apdl)
```

```
    (tagdef,loopmark)
        (jumpif,donemark,(eq,gccounter,saveapdl))
        (set,gcsearch,(aget,apdl,gccounter))
        (inc,gccounter)
          (jumpz, loopmark, (within, gcsearch, cons_sp, cons_cell))
        (set,cell_type_1,(get,(struc,cons_cell,gcsearch),cell_type,0))
          (jumpif, loopmark,(ne, cell_type_1, C_CONS_CELL))
        (apush,C_MARK)
        (apush,gcsearch)
        (tagdef,lpmark)
          (set,gcsearch,(apop))
          (jumpif,loopmark,(eq,gcsearch,C_MARK))
          (set,cell_type_1,(get,(struc,cons_cell,gcsearch),
                cell_type,0))
          (jumpif,lpmark,(ne,cell_type_1,C_CONS_CELL))
          (jumpif,lpmark,(eq,
                (get,(struc,cons_cell,gcsearch),gc,0),0))
          (put,(struc,cons_cell,gcsearch),gc,0,0) (comment, do no gc)
          (apush,(get,(struc,cons_cell,gcsearch),cdr_pointer,0))
          (apush,(get,(struc,cons_cell,gcsearch),car_pointer,0))
         (jump,lpmark)
      (tagdef,donemark)
        (ret)
(srend,mark_apdl)
```

Appendix E: CORVALLISP LISTING IN C

```
      #
long poa();
long poj();
/**/
struct common_fields {
  char cell_type[1] ;
  char gc[1] ;
};
#define size_common_fields sizeof( struct common_fields )
struct cons_cell {
  char cell_type[1] ;
  char gc[1] ;
  long car_pointer[1] ;
  long cdr_pointer[1] ;
};
#define size_cons_cell sizeof( struct cons_cell )
struct atom_header {
  char cell_type[1] ;
  char gc[1] ;
  long vcell[1] ;
  long plist[1] ;
  long pname[1] ;
};
#define size_atom_header sizeof( struct atom_header )
struct small_num {
  char cell_type[1] ;
  char gc[1] ;
  long vcell[1] ;
};
#define size_small_num sizeof( struct small_num )
struct string_args {
  char st[100] ;
};
#define size_string_args sizeof( struct string_args )
/**/
 long add_buff, bot_buff ;
#define len_buff 100
char buff [100] ;
 long add_apdl, bot_apdl ;
#define len_apdl 100000
long apdl [100000] ;
 long add_jpdl, bot_jpdl ;
#define len_jpdl 100000
long jpdl [100000] ;
 long add_cons_sp, bot_cons_sp ;
#define len_cons_sp 6000000
char cons_sp [6000000] ;
 long add_atom_sp, bot_atom_sp ;
#define len_atom_sp 1000000
```

```
char atom_sp [1000000] ;
 long add_strg_sp, bot_strg_sp ;
#define len_strg_sp 10000
char strg_sp [10000] ;
#define C_OBLIST_LEN 1000
 long add_oblist, bot_oblist ;
#define len_oblist C_OBLIST_LEN
long oblist [C_OBLIST_LEN] ;
 long add_print, bot_print ;
#define len_print 100
char print [100] ;
 long add_pnamesp, bot_pnamesp ;
#define len_pnamesp 1000
char pnamesp [1000] ;
/**/
#define LISP 1
#define LISP_1 2
#define PUSH_RESULT_ON_APDL 3
#define FIND_END 4
#define APPEND_1 5
#define LOAD 6
#define LOAD_1 7
#define CHECK_LOAD_END 8
#define READ_0 9
#define READ 10
#define READATOM 11
#define RDLST 12
#define RDLST_1 13
#define RDLST_2 14
#define RDLST_3 15
#define PRINTATOM 65
#define TERPRINT 16
#define PRINT 17
#define PRINT_1 18
#define PRINT_2 19
#define ATOM_1 20
#define NCONC_1 21
#define EVAL_0 22
#define EVAL 23
#define ARG_HANDLER 24
#define FN_ERROR 25
#define RESET 26
#define RETURN 27
#define BODY_HANDLER 28
#define RESULT_HANDLER 29
#define EVAL_ARG 30
#define NEVAL_ARG 31
#define PROGN 32
#define DO_BINDINGS 33
#define OR 34
#define AND 35
#define DE 36
#define DF 37
```

```
#define LIST 38
#define COND 39
#define QUOTE 40
#define COND_1 41
#define EVLIS 42
#define EVLIS_1 43
#define EVLIS_2 44
#define EVLIS_3 45
#define APPEND 46
#define CP_LIST 47
#define CP_LIST_1 48
#define LAST 49
#define CAR 50
#define CDR 51
#define CONS 52
#define SET 53
#define SETQ 54
#define SET0 55
#define NULL 56
#define ATOM 57
#define EQ 58
#define RPLACD 59
#define RPLACA 60
#define NCONC 61
#define PUTP 62
#define GETP 63
#define EXIT 64
#define PRINTSMALL 66
#define MKSMALL 67
#define EQNUM 68
#define LENUM 69
#define GENUM 70
#define LNUM 71
#define GNUM 72
#define MOD 73
#define DIV 74
#define MUL 75
#define ADD 76
#define SUB 77
#define MAC_EXP 78
#define MAC_EXP_1 79
#define TERPRINT_1 80
#define OBLIST 81
#define ROOM 82
#define GC 83
/**/
/**/
long gc_warn;
long roomcnt;
long sign;
long last;
long lisp_level;
long gctemp;
```

```
long gcsearch;
long gccounter;
long gcar;
long gcdr;
long gc_old;
long avail_cons;
long keep_cons;
long loc_pnsp;
long ploc;
long propname;
long propval;
long atomname;
long pass;
long eof;
long old_val;
long f_par;
long fncell_type;
long cell_type_1;
long counter;
long front;
long back;
long nil;
long inunit;
long outunit;
long t;
long arg1;
long arg2;
long temp;
long hashval;
long argu;
long loc;
long search;
long old_search;
long pointer;
long a_found;
long f_body;
long apar;
long apar1;
long fpar;
long fpar1;
long fparval;
long arg;
long rest;
long first;
long first_case;
long cond_body;
long form;
long fn;
long val;
long result;
long pars;
long args;
long call_form;
```

```
long keeper;
long keeper2;
long aarg1;
long aarg2;
/**/
/**/
#define C_MAC_CHAR '?
#define C_INUNIT 0
#define C_OUTUNIT 1
#define C_ATOM_HEDER 1
#define C_SUB 2
#define C_FSUB 3
#define C_LAM 4
#define C_FLAM 5
#define C_CONS_CELL 6
#define C_SNUM 7
#define S_LPAR "("
#define S_RPAR ")"
#define S_BLANK " "
#define C_LPAR '('
#define C_RPAR ')'
#define C_BLANK ' '
#define C_TAB '        '
#define C_MARK -5533111
#define C_MARK_1 -5533112
#define C_NEWLINE '0
int _debug;
main(argc,argv)
int argc;
char **argv;
{
#define EXIT_FLAG -1
  long prog_switch;
  _debug = 0;
  if (argc > 1) _debug++;
  prog_switch = LISP;
  while (prog_switch != EXIT_FLAG)
  switch (prog_switch) {
/**/
case LISP :
  init();
  write((int)C_OUTUNIT,(char *)"CORVALLISP0,11);
  puj( (long)LISP_1 );
  prog_switch=poj();break;
/**/
case LISP_1 :
  /*
  if ( (long) gc_warn > (long) 0 ){ goto no_need_to_gc;};
  gc_cons();
  no_need_to_gc:
  */
  puj( (long)LISP_1 );
  puj( (long)PRINT_1 );
```

```
  puj( (long)PUSH_RESULT_ON_APDL );
  puj( (long)EVAL_0 );
  puj( (long)PUSH_RESULT_ON_APDL );
  puj( (long)READ_0 );
  write((int)C_OUTUNIT,(char *)"0>>>>>>>>>>>>>>>>>>>>>>>",lisp_level)
  write((int)outunit,(char *)&S_BLANK,1);
  prog_switch=poj();break;
/**/
case PUSH_RESULT_ON_APDL :
  pua( (long)((struct atom_header *) result)->vcell[0] );
prog_switch=poj();break;
/**/
case FIND_END :
  arg = (long)poa();
  pua( (long)arg );
  if ( (long) arg == (long) nil ){ goto done_fe;};
  loop_until_end:
    temp = (long)((struct cons_cell *) arg)->cdr_pointer[0];
    if ( (long) temp == (long) nil ){ goto done_fe;};
      arg = (long)temp;
      goto loop_until_end;
  done_fe:
    ((struct atom_header *) result)->vcell[0] = arg;
    prog_switch=poj();break;
/**/
case APPEND_1 :
  arg1 = (long)poa();
  arg = (long)poa();
  arg2 = (long)poa();
  if ( (long) arg1 == (long) nil ){ goto arg1_is_nil;};
  ((struct cons_cell *) arg1)->cdr_pointer[0] = arg2;
  ((struct atom_header *) result)->vcell[0] = arg;
  prog_switch=poj();break;
  arg1_is_nil:
    ((struct atom_header *) result)->vcell[0] = arg2;
    prog_switch=poj();break;
/**/
case LOAD :
  arg = (long)poa();
  arg = (long)((struct cons_cell *) arg)->car_pointer[0];
  ploc = (long)((struct atom_header *) arg)->pname[0];
  arg1=open(&pnamesp[ploc] ,2);
  inunit = (long)arg1;
  eof = (long)1;
  puj( (long)LOAD_1 );
  prog_switch=poj();break;
/**/
case LOAD_1 :
  puj( (long)LOAD_1 );
  puj( (long)PRINT_1 );
  puj( (long)PUSH_RESULT_ON_APDL );
  puj( (long)EVAL_0 );
  puj( (long)PUSH_RESULT_ON_APDL );
```

```
  puj( (long)CHECK_LOAD_END );
  puj( (long)READ_0 );
  prog_switch=poj();break;
/**/
case CHECK_LOAD_END :
  if ( (long) eof <= (long) 0 ){ goto doneload;};
    prog_switch=poj();break;
doneload:
  temp = (long)poj();
  temp = (long)poj();
  temp = (long)poj();
  temp = (long)poj();
  temp = (long)poj();
  ((struct atom_header *) result)->vcell[0] = nil;
  close(inunit);
  inunit = (long)C_INUNIT;
  prog_switch=poj();break;
/**/
case READ_0 :
  spaces:
    eof = (long)read((int)inunit,(char *)&buff[0] ,1);;
    if ( (long) eof <= (long) 0 ){ goto readone;};
    if ( (long) buff[0] == (long) C_BLANK ){ goto spaces;};
    if ( (long) buff[0] == (long) C_TAB ){ goto spaces;};
    if ( (long) buff[0] == (long) C_RPAR ){ goto spaces;};
    if ( (long) buff[0] == (long) C_NEWLINE ){ goto spaces;};
    if ( (long) buff[0] != (long) C_LPAR ){ goto aread;};
    puj( (long)RDLST );
    prog_switch=poj();break;
  aread:
    puj( (long)READATOM );
  readone:
    prog_switch=poj();break;
  macread:
    puj( (long)MAC_EXP );
    prog_switch=poj();break;
/**/
case MAC_EXP :
  mkcons();
  ((struct atom_header *) keeper)->vcell[0] = (long)poa();
  ((struct cons_cell *)
    ((struct atom_header *) keeper)->vcell[0])->car_pointer[0] = nil;
  mkcons();
  ((struct atom_header *) keeper2)->vcell[0] = (long)poa();
  ((struct cons_cell *) ((struct atom_header *) keeper)->vcell[0])
    ->cdr_pointer[0] = ((struct atom_header *) keeper2)->vcell[0];
  ((struct cons_cell *) ((struct atom_header *) keeper2)->vcell[0])
    ->car_pointer[0] = nil;
  ((struct cons_cell *) ((struct atom_header *) keeper2)->vcell[0])->
    cdr_pointer[0] = nil;
  strcpy((char *)&((struct string_args *) pass)->st[0],
    (char *)"quote");
  pua( (long)pass );
```

```
    intern();
    ((struct atom_header *) result)->vcell[0] = poa();
    ((struct cons_cell *) ((struct atom_header *) keeper)->vcell[0])->
      car_pointer[0] = ((struct atom_header *) result)->vcell[0];
    pua( (long)((struct atom_header *) keeper)->vcell[0] );
    puj( (long)MAC_EXP_1 );
    puj( (long)READ_0 );
    prog_switch=poj();break;
/**/
case MAC_EXP_1 :
    ((struct atom_header *) keeper)->vcell[0] = (long)poa();
    ((struct atom_header *) keeper2)->vcell[0] =
      (long)((struct cons_cell *)
      ((struct atom_header *) keeper)->vcell[0])->cdr_pointer[0];
    ((struct cons_cell *) ((struct atom_header *)
      keeper2)->vcell[0])->car_pointer[0] =
        ((struct atom_header *) result)->vcell[0];
    ((struct atom_header *) result)->vcell[0] =
      ((struct atom_header *) keeper)->vcell[0];
    prog_switch=poj();break;
/**/
case READ :
    arg = (long)poa();
    puj( (long)READ_0 );
    prog_switch=poj();break;
/**/
case READATOM :
   counter = (long)99;
   loopclear:
     ((struct string_args *) pass)->st[counter] = ' ';
     if ( !( counter )){ goto readchars ;};
     counter = (long)counter-1;
     goto loopclear;
   readchars:
     ((struct string_args *) pass)->st[0] = buff[0] ;
     if ( (long) buff[0] != (long) C_MAC_CHAR ){ goto dotheatom;};
       puj( (long)MAC_EXP );
       prog_switch=poj();break;
     dotheatom:
     counter = (long)0;
     charloop:
       read((int)inunit,(char *)&buff[0] ,1);
       if ( (long) buff[0] == (long) C_LPAR ){ goto endatom;};
       if ( (long) buff[0] == (long) C_RPAR ){ goto endatom;};
       if ( (long) buff[0] == (long) C_BLANK ){ goto endatom;};
       if ( (long) buff[0] == (long) C_TAB ){ goto endatom;};
       if ( (long) buff[0] == (long) C_NEWLINE ){ goto endatom;};
       if ( (long) counter <= (long) 98 ){ goto storechar;};
     goto charloop;
storechar:
       counter++;
       ((struct string_args *) pass)->st[counter] = buff[0] ;
       goto charloop;
```

```
    endatom:
      pua( (long)pass );
      intern();
       ((struct atom_header *) result)->vcell[0] = poa();
      prog_switch=poj();break;
/**/
case RDLST:
 ((struct atom_header *) front)->vcell[0] = nil;
 puj( (long)RDLST_1 );
 prog_switch=poj();break;
/**/
case RDLST_1 :
  spaceloop:
  read((int)inunit,(char *)&buff[0] ,1);
  if ( (long)  buff[0]  == (long) C_BLANK ){ goto spaceloop;};
  if ( (long)  buff[0]  == (long) C_TAB ){ goto spaceloop;};
  if ( (long)  buff[0]  == (long) C_NEWLINE ){ goto spaceloop;};
  if ( !( (long)  buff[0]  == (long) C_RPAR )){ goto restread ;};
  ((struct atom_header *) result)->vcell[0] =
    ((struct atom_header *) front)->vcell[0];
   buff[0] = C_BLANK;
   prog_switch=poj();break;
  restread:
   if ( !( (long)  buff[0]  == (long) C_LPAR )){ goto atomfound ;};
   pua( (long)((struct atom_header *) front)->vcell[0] );
   puj( (long)RDLST_2 );
   puj( (long)RDLST );
   prog_switch=poj();break;
   atomfound:
    pua( (long)((struct atom_header *) front)->vcell[0] );
    puj( (long)RDLST_2 );
    puj( (long)READATOM );
    prog_switch=poj();break;
/**/
case RDLST_2 :
 ((struct atom_header *) front)->vcell[0] = poa();
 pua( (long)((struct atom_header *) result)->vcell[0] );
 pua( (long)nil );
 cons_1();
 pua( (long)((struct atom_header *) result)->vcell[0] );
 pua( (long)((struct atom_header *) front)->vcell[0] );
 puj( (long)RDLST_3 );
 puj( (long)NCONC_1 );
 prog_switch=poj();break;
/**/
case RDLST_3 :
 ((struct atom_header *) front)->vcell[0] =
   ((struct atom_header *) result)->vcell[0];
 if ( !( (long)  buff[0]  == (long) C_RPAR )){ goto next1 ;};
 buff[0] = C_BLANK;
 ((struct atom_header *) result)->vcell[0] =
   ((struct atom_header *) front)->vcell[0];
 prog_switch=poj();break;
```

```
  next1:
   if ( !( (long)  buff[0]  == (long) C_LPAR )){ goto next2 ;};
   pua( (long)((struct atom_header *) front)->vcell[0] );
   puj( (long)RDLST_2 );
   puj( (long)RDLST_1 );
   prog_switch=poj();break;
  next2:
   puj( (long)RDLST_1 );
   prog_switch=poj();break;
/**/
case PRINTATOM :
 arg = (long)poa();
 ploc = (long)((struct atom_header *) arg)->pname[0];
 strcpy((char *)&((struct string_args *)
  pass)->st[0],(char *)&pnamesp[ploc] );
  counter = (long)0;
  cont_print:
   if ( (long)  counter == (long) 100 ){ goto outpr;};
   buff[0] = ((struct string_args *) pass)->st[counter];
   if ( (long)  buff[0]  == (long) ' ' ){ goto outpr;};
   write((int)C_OUTUNIT,(char *)&buff[0] ,1);
   counter++;
   goto cont_print;
  outpr:
  write((int)C_OUTUNIT,(char *)&S_BLANK,1);
   prog_switch=poj();break;
/**/
case PRINTSMALL :
  arg = (long)poa();
  arg = (long)((struct atom_header *) arg)->vcell[0];
  if ( arg ){ goto nzeronum ;};
    write((int)outunit,(char *)"0",1);
    write((int)outunit,(char *)&S_BLANK,1);
    prog_switch=poj();break;
  nzeronum:
  if ( (long) arg >  (long) 0 ){ goto posit;};
    arg = (long)arg*-1;
    write((int)outunit,(char *)"-",1);
  posit:
  counter = (long)50;
  snum_loop:
    old_val = (long)arg%10;
    buff[counter] = old_val+ '0';
    counter--;
    arg = (long)arg/10;
    if ( !( arg )){ goto dopoping ;};
    goto snum_loop;
  dopoping:
  write((int)outunit,(char *)&buff[counter+ 1] ,50-counter);
  donesmalp:
  write((int)outunit,(char *)&S_BLANK,1);
   prog_switch=poj();break;
/**/
```

```
case GC :
  arg = (long)poa();
  gc_cons();
  ((struct atom_header *) result)->vcell[0] = t;
  prog_switch=poj();break;
/**/
case ROOM :
  arg = (long)poa();
  arg = (long)avail_cons;
  roomcnt = (long)0;
  looproom:
    if ( (long)  arg == (long) nil ){ goto doneroom;};
    roomcnt++;
    arg = (long)((struct cons_cell *) arg)->car_pointer[0];
    goto looproom;
  doneroom:
  pua( (long)roomcnt );
  puj( (long)MKSMALL );
  prog_switch=poj();break;
/**/
case TERPRINT :
  arg = (long)poa();
  ((struct atom_header *) result)->vcell[0] = nil;
  puj( (long)TERPRINT_1 );
  prog_switch=poj();break;
case TERPRINT_1 :
  write((int)C_OUTUNIT,(char *)"0,1);
  prog_switch=poj();break;
/**/
case PRINT :
  arg = (long)poa();
  ((struct atom_header *) result)->vcell[0] = nil;
  arg1 = (long)((struct cons_cell *) arg)->car_pointer[0];
  pua( (long)arg1 );
  puj( (long)TERPRINT_1 );
  puj( (long)PRINT_1 );
  prog_switch=poj();break;
/**/
case PRINT_1 :
 arg = (long)poa();
 cell_type_1 = (long)((struct cons_cell *) arg)->cell_type[0];
 if ( (long)  cell_type_1 == (long) C_CONS_CELL ){ goto is_list;};
 if ( (long)  cell_type_1 == (long) C_SNUM ){ goto is_smal_1;};
 pua( (long)arg );
puj( (long)PRINTATOM );
prog_switch=poj();break;
is_smal_1:
 pua( (long)arg );
puj( (long)PRINTSMALL );
prog_switch=poj();break;
is_list:
 puj( (long)PRINT_2 );
 puj( (long)PRINT_1 );
```

```
   pua( (long)((struct cons_cell *) arg)->cdr_pointer[0] );
   pua( (long)((struct cons_cell *) arg)->car_pointer[0] );
   write((int)C_OUTUNIT,(char *)&S_LPAR,1);
   prog_switch=poj();break;
/**/
case PRINT_2 :
  arg = (long)poa();
  if ( (long)  arg == (long) nil ){ goto printend;};
  pua( (long)((struct cons_cell *) arg)->cdr_pointer[0] );
  pua( (long)((struct cons_cell *) arg)->car_pointer[0] );
  puj( (long)PRINT_2 );
  puj( (long)PRINT_1 );
  prog_switch=poj();break;
  printend:
  write((int)C_OUTUNIT,(char *)&S_RPAR,1);
  prog_switch=poj();break;
/**/
case ATOM_1 :
   arg = (long)poa();
   cell_type_1 = (long)((struct cons_cell *) arg)->cell_type[0];
   if ( (long)  cell_type_1 == (long) C_ATOM_HEDER ){ goto isatom;};
   if ( (long)  cell_type_1 == (long) C_SNUM ){ goto isatom;};
     ((struct atom_header *) result)->vcell[0] = nil;
     prog_switch=poj();break;
   isatom:
     ((struct atom_header *) result)->vcell[0] = t;
     prog_switch=poj();break;
/**/
case NCONC_1 :
  puj( (long)APPEND_1 );
  puj( (long)PUSH_RESULT_ON_APDL );
  puj( (long)FIND_END );
  prog_switch=poj();break;
/**/
case EVAL_0 :
  form = (long)poa();
  if ( (long)  ((struct cons_cell *) form)->cell_type[0]
    == (long) C_ATOM_HEDER ){ goto it_is_atom;};
  if ( (long)  ((struct cons_cell *) form)->cell_type[0]
    == (long) C_SNUM ){ goto it_is_small;};
  if ( (long)  ((struct cons_cell *) form)->cell_type[0]
    == (long) C_LAM ){ goto it_is_atom;};
  if ( (long)  ((struct cons_cell *) form)->cell_type[0]
    == (long) C_FLAM ){ goto it_is_atom;};
  if ( (long) ((struct cons_cell *) form)->cell_type[0]
    != (long) C_CONS_CELL ){ goto not_cons;};
     pua( (long)form );
     fn = (long)((struct cons_cell *) form)->car_pointer[0];
     fncell_type = (long)((struct cons_cell *) fn)->cell_type[0];
     fn = (long)((struct atom_header *) fn)->vcell[0];
     pua( (long)fncell_type );
     pua( (long)fn );
     puj( (long)RESULT_HANDLER );
```

```
      puj( (long)BODY_HANDLER );
      puj( (long)ARG_HANDLER );
      prog_switch=poj();break;
    it_is_atom:
      ((struct atom_header *) result)->vcell[0] =
          ((struct atom_header *) form)->vcell[0];
      prog_switch=poj();break;
    it_is_small:
    not_cons:
      ((struct atom_header *) result)->vcell[0] = form;
      prog_switch=poj();break;
  /**/
  case EVAL :
    arg = (long)poa();
    arg = (long)((struct cons_cell *) arg)->car_pointer[0];
    pua( (long)arg );
    puj( (long)EVAL_0 );
    prog_switch=poj();break;
  /**/
  case ARG_HANDLER :
    f_body = (long)poa();
    fncell_type = (long)poa();
    pua( (long)fncell_type );
    pua( (long)f_body );
    if ( (long)  fncell_type == (long) C_LAM ){ goto do_EVAL;};
    if ( (long)  fncell_type == (long) C_FLAM ){ goto dont_EVAL;};
    if ( (long)  fncell_type == (long) C_SUB ){ goto do_EVAL;};
    if ( (long)  fncell_type == (long) C_FSUB ){ goto dont_EVAL;};
    puj( (long)FN_ERROR );
    prog_switch=poj();break;
    do_EVAL:
      puj( (long)EVAL_ARG );
      prog_switch=poj();break;
    dont_EVAL:
      puj( (long)NEVAL_ARG );
      prog_switch=poj();break;
  /**/
  case FN_ERROR :
    lisp_level++;
    write((int)C_OUTUNIT,(char *)"UNKNOWN FUNCTION....",20);
    temp = (long)poa();
    temp = (long)poa();
    temp = (long)poa();
    pua( (long)C_MARK_1 );
    pua( (long)temp );
    temp = (long)poj();
    temp = (long)poj();
    puj( (long)C_MARK_1 );
    puj( (long)LISP_1 );
    puj( (long)PRINT_1 );
    prog_switch=poj();break;
  case RESET :
    lisp_level = (long)2;
```

```
    add_apdl = 0;bot_apdl=len_apdl-1;
    add_jpdl = 0;bot_jpdl=len_jpdl-1;
    puj( (long)LISP_1 );
    prog_switch=poj();break;
  case RETURN :
    arg = (long)poa();
    if ( (long) lisp_level !=  (long) 2 ){ goto okret;};
    write((int)C_OUTUNIT,(char *)"CANNOT RETURN0,14);
    ((struct atom_header *) result)->vcell[0] = nil;
    prog_switch=poj();break;
    okret:
    arg = (long)((struct cons_cell *) arg)->car_pointer[0];
    temp = (long)poj();
    lpret1:
      if ( (long)  temp == (long) C_MARK_1 ){ goto donret1;};
      temp = (long)poj();
      goto lpret1;
    donret1:
    temp = (long)poa();
    lpret2:
      if ( (long)  temp == (long) C_MARK_1 ){ goto donret2;};
      temp = (long)poa();
      goto lpret2;
    donret2:
    lisp_level--;
    ((struct atom_header *) result)->vcell[0] = arg;
    prog_switch=poj();break;
  /**/
  /**/
  case BODY_HANDLER :
    pars = (long)poa();
    f_body = (long)poa();
    fncell_type = (long)poa();
    pua( (long)C_MARK );
    if ( (long)  fncell_type == (long) C_SUB ){ goto binary;};
    if ( (long)  fncell_type == (long) C_FSUB ){ goto binary;};
      args = (long)((struct cons_cell *) f_body)->car_pointer[0];
      f_body = (long)((struct cons_cell *) f_body)->cdr_pointer[0];
      pua( (long)f_body );
      pua( (long)args );
      pua( (long)pars );
      puj( (long)PROGN );
      puj( (long)DO_BINDINGS );
      prog_switch=poj();break;
    binary:
      pua( (long)pars );
      puj( (long)f_body );
      prog_switch=poj();break;
  /**/
  case RESULT_HANDLER :
  spin_1:
    f_par = (long)poa();
    if ( (long)  f_par == (long) C_MARK ){ goto outr;};
```

```
      old_val = (long)poa();
      ((struct atom_header *) f_par)->vcell[0] = old_val;
goto spin_1;
outr:
      prog_switch=poj();break;
/**/
case EVAL_ARG :
   f_body = (long)poa();
   fncell_type = (long)poa();
   call_form = (long)poa();
   pua( (long)fncell_type );
   pua( (long)f_body );
   pua( (long)((struct cons_cell *) call_form)->cdr_pointer[0] );
   puj( (long)PUSH_RESULT_ON_APDL );
   puj( (long)EVLIS );
   prog_switch=poj();break;
/**/
case NEVAL_ARG :
   f_body = (long)poa();
   fncell_type = (long)poa();
   call_form = (long)poa();
   pua( (long)fncell_type );
   pua( (long)f_body );
   pua( (long)((struct cons_cell *) call_form)->cdr_pointer[0] );
   prog_switch=poj();break;
/**/
case PROGN :
   f_body = (long)poa();
   if ( (long)  f_body == (long) nil ){ goto donep;};
      pua( (long)((struct cons_cell *) f_body)->cdr_pointer[0] );
      pua( (long)((struct cons_cell *) f_body)->car_pointer[0] );
      puj( (long)PROGN );
      puj( (long)EVAL_0 );
      prog_switch=poj();break;
   donep:
      prog_switch=poj();break;
/**/
case DO_BINDINGS :
   apar = (long)poa();
   fpar = (long)poa();
   f_body = (long)poa();
   until_done:
      if ( (long)  apar == (long) nil ){ goto done_binding;};
      if ( (long)  fpar == (long) nil ){ goto done_binding;};
         fpar1 = (long)((struct cons_cell *) fpar)->car_pointer[0];
         fpar = (long)((struct cons_cell *) fpar)->cdr_pointer[0];
         apar1 = (long)((struct cons_cell *) apar)->car_pointer[0];
         apar = (long)((struct cons_cell *) apar)->cdr_pointer[0];
         fparval = (long)((struct atom_header *) fpar1)->vcell[0];
         pua( (long)fparval );
         pua( (long)fpar1 );
         ((struct atom_header *) fpar1)->vcell[0] = apar1;
         goto until_done;
```

```
    done_binding:
    pua( (long)f_body );
    prog_switch=poj();break;
  /**/
  case OR :
    arg = (long)poa();
    ((struct atom_header *) result)->vcell[0] = nil;
    loopor:
      if ( (long)  arg == (long) nil ){ goto doneor;};
      first = (long)((struct cons_cell *) arg)->car_pointer[0];
      arg = (long)((struct cons_cell *) arg)->cdr_pointer[0];
      if ( (long)  first == (long) nil ){ goto loopor;};
    ((struct atom_header *) result)->vcell[0] = t;
    doneor:
      prog_switch=poj();break;
  /**/
  case AND :
    arg = (long)poa();
    ((struct atom_header *) result)->vcell[0] = t;
    loopand:
      if ( (long)  arg == (long) nil ){ goto doneand;};
      first = (long)((struct cons_cell *) arg)->car_pointer[0];
      arg = (long)((struct cons_cell *) arg)->cdr_pointer[0];
      if ( (long) first !=  (long) nil ){ goto loopand;};
    ((struct atom_header *) result)->vcell[0] = nil;
    doneand:
      prog_switch=poj();break;
  /**/
  case DE :
    arg = (long)poa();
    fn = (long)((struct cons_cell *) arg)->car_pointer[0]; /**/
    arg2 = (long)((struct cons_cell *) arg)->cdr_pointer[0]; /**/
    ((struct cons_cell *) fn)->cell_type[0] = C_LAM;
    ((struct atom_header *) fn)->vcell[0] = arg2;
    ((struct atom_header *) result)->vcell[0] = fn;
    prog_switch=poj();break;
  /**/
  case DF :
    arg = (long)poa();
    fn = (long)((struct cons_cell *) arg)->car_pointer[0];
    arg2 = (long)((struct cons_cell *) arg)->cdr_pointer[0];
    ((struct cons_cell *) fn)->cell_type[0] = C_FLAM;
    ((struct atom_header *) fn)->vcell[0] = arg2;
    ((struct atom_header *) result)->vcell[0] = fn;
    prog_switch=poj();break;
  /**/
  case LIST :
    ((struct atom_header *) result)->vcell[0] = poa();
    prog_switch=poj();break;
  /**/
  case COND :
    cond_body = (long)poa();
    if ( (long) cond_body !=  (long) nil ){ goto cond_cont;};
```

```
    ((struct atom_header *) result)->vcell[0] = nil;
    prog_switch=poj();break;
    cond_cont:
      first_case = (long)
          ((struct cons_cell *) cond_body)->car_pointer[0];
      pua( (long)((struct cons_cell *) cond_body)->cdr_pointer[0] );
      pua( (long)((struct cons_cell *) first_case)->cdr_pointer[0] );
      pua( (long)((struct cons_cell *) first_case)->car_pointer[0] );
      puj( (long)COND_1 );
      puj( (long)EVAL_0 );
      prog_switch=poj();break;
  /**/
  case QUOTE :
    arg = (long)poa();
    ((struct atom_header *) result)->vcell[0] =
      ((struct cons_cell *) arg)->car_pointer[0];
    prog_switch=poj();break;
  /**/
  case COND_1 :
    if ( (long) ((struct atom_header *) result)->vcell[0]
      != (long) nil ){ goto not_nil;};
      temp = (long)poa(); /**/
      puj( (long)COND );
      prog_switch=poj();break;
    not_nil:
      arg = (long)poa(); /**/
      arg1 = (long)poa(); /**/
      pua( (long)arg );
      puj( (long)PROGN );
      prog_switch=poj();break;
  /**/
  case EVLIS :
      arg = (long)poa();
    if ( (long) arg != (long) nil ){ goto cont_evlis;};
      ((struct atom_header *) result)->vcell[0] = nil;
      prog_switch=poj();break;
    cont_evlis:
      first = (long)((struct cons_cell *) arg)->car_pointer[0];
      rest = (long)((struct cons_cell *) arg)->cdr_pointer[0];
      pua( (long)rest );
      pua( (long)first );
      puj( (long)EVLIS_1 );
      puj( (long)EVAL_0 );
      prog_switch=poj();break;
  /**/
  case EVLIS_1 :
    rest = (long)poa();
    pua( (long)((struct atom_header *) result)->vcell[0] );
    pua( (long)nil );
    cons_1();
    pua( (long)((struct atom_header *) result)->vcell[0] );
    pua( (long)rest );
    pua( (long)((struct atom_header *) result)->vcell[0] );
```

```
    puj( (long)EVLIS_2 );
    prog_switch=poj();break;
  /**/
  case EVLIS_2 :
    temp = (long)poa();
    rest = (long)poa();
    if ( (long)  rest == (long) nil ){ goto done_evlis_2;};
      first = (long)((struct cons_cell *) rest)->car_pointer[0];
      rest = (long)((struct cons_cell *) rest)->cdr_pointer[0];
      pua( (long)temp );
      pua( (long)rest );
      pua( (long)first );
      puj( (long)EVLIS_3 );
      puj( (long)EVAL_0 );
      prog_switch=poj();break;
    done_evlis_2:
      ((struct atom_header *) result)->vcell[0] = poa();
      prog_switch=poj();break;
  /**/
  case EVLIS_3 :
    rest = (long)poa();
    temp = (long)poa();
    pua( (long)((struct atom_header *) result)->vcell[0] );
    pua( (long)nil );
    cons_1();
    ((struct cons_cell *) temp)->cdr_pointer[0] =
      ((struct atom_header *) result)->vcell[0];
    pua( (long)rest );
    pua( (long)((struct atom_header *) result)->vcell[0] );
    puj( (long)EVLIS_2 );
    prog_switch=poj();break;
  /**/
  case APPEND :
    arg = (long)poa();
    arg1 = (long)((struct cons_cell *) arg)->car_pointer[0];
    arg2 = (long)((struct cons_cell *) arg)->cdr_pointer[0];
    arg2 = (long)((struct cons_cell *) arg2)->car_pointer[0];
    cell_type_1 = (long)((struct cons_cell *) arg1)->cell_type[0];
    if ( (long)  cell_type_1 == (long) C_CONS_CELL ){ goto ok_app;};
      /**/
    ok_app:
      pua( (long)arg2 );
      pua( (long)arg1 );
      puj( (long)APPEND_1 );
      puj( (long)PUSH_RESULT_ON_APDL );
      puj( (long)FIND_END );
      puj( (long)PUSH_RESULT_ON_APDL );
      puj( (long)CP_LIST );
      prog_switch=poj();break;
  /**/
  case CP_LIST :
    arg = (long)poa();
    if ( (long) arg != (long) nil ){ goto cont_cp;};
```

```
      ((struct atom_header *) result)->vcell[0] = nil;
      prog_switch=poj();break;
    cont_cp:
      first = (long)((struct cons_cell *) arg)->car_pointer[0];
      rest = (long)((struct cons_cell *) arg)->cdr_pointer[0];
      pua( (long)first );
      pua( (long)nil );
      cons_1();
      pua( (long)((struct atom_header *) result)->vcell[0] );
      pua( (long)rest );
      pua( (long)((struct atom_header *) result)->vcell[0] );
      puj( (long)CP_LIST_1 );
      prog_switch=poj();break;
/**/
case CP_LIST_1 :
   temp = (long)poa();
   rest = (long)poa();
   if ( (long)  rest == (long) nil ){ goto done_cp;};
      first = (long)((struct cons_cell *) rest)->car_pointer[0];
      rest = (long)((struct cons_cell *) rest)->cdr_pointer[0];
      pua( (long)rest );
      pua( (long)first );
      pua( (long)nil );
      cons_1();
      ((struct cons_cell *) temp)->cdr_pointer[0] =
          ((struct atom_header *) result)->vcell[0];
      pua( (long)((struct atom_header *) result)->vcell[0] );
      puj( (long)CP_LIST_1 );
      prog_switch=poj();break;
   done_cp:
      ((struct atom_header *) result)->vcell[0] = poa();
      prog_switch=poj();break;
/**/
case LAST :
   arg = (long)poa();
   arg = (long)((struct cons_cell *) arg)->car_pointer[0];
   pua( (long)arg );
    /**/
   puj( (long)FIND_END );
   prog_switch=poj();break;
/**/
case CAR :
   arg = (long)poa();
   arg = (long)((struct cons_cell *) arg)->car_pointer[0];
   cell_type_1 = (long)((struct cons_cell *) arg)->cell_type[0];
   if ( (long)  cell_type_1 == (long) C_CONS_CELL ){ goto carok;};
   write((int)C_OUTUNIT,(char *)"TRIED TO TAKE CAR OF NON LIST....",33);
   pua( (long)arg );
   puj( (long)RESET );
   puj( (long)PRINT_1 );
   prog_switch=poj();break;
   carok:
   ((struct atom_header *) result)->vcell[0] =
```

```
    ((struct cons_cell *) arg)->car_pointer[0];
   prog_switch=poj();break;
 /**/
 case CDR :
   arg = (long)poa();
   arg = (long)((struct cons_cell *) arg)->car_pointer[0];
   cell_type_1 = (long)((struct cons_cell *) arg)->cell_type[0];
   if ( (long)  cell_type_1 == (long) C_CONS_CELL ){ goto cdrok;};
   if ( (long)  arg == (long) nil ){ goto nilcdr;};
   write((int)C_OUTUNIT,(char *)"TRIED TO TAKE CDR OF NON LIST....",33);
   pua( (long)arg );
   puj( (long)RESET );
   puj( (long)PRINT_1 );
   prog_switch=poj();break;
   cdrok:
    ((struct atom_header *) result)->vcell[0] =
     ((struct cons_cell *) arg)->cdr_pointer[0];
    prog_switch=poj();break;
   nilcdr:
     ((struct atom_header *) result)->vcell[0] = nil;
    prog_switch=poj();break;
 /**/
 case CONS :
   arg = (long)poa();
   arg1 = (long)((struct cons_cell *) arg)->car_pointer[0];
   arg2 = (long)((struct cons_cell *) arg)->cdr_pointer[0];
   arg2 = (long)((struct cons_cell *) arg2)->car_pointer[0];
   pua( (long)arg1 );
   pua( (long)arg2 );
   cons_1();
   prog_switch=poj();break;
 /**/
 case SET :
   arg = (long)poa();
   arg1 = (long)((struct cons_cell *) arg)->car_pointer[0];
   arg2 = (long)((struct cons_cell *) arg)->cdr_pointer[0];
   arg2 = (long)((struct cons_cell *) arg2)->car_pointer[0];
   pua( (long)arg1 );
   pua( (long)arg2 );
   puj( (long)SET0 );
   prog_switch=poj();break;
 /**/
 case SETQ :
  arg = (long)poa();
  arg1 = (long)((struct cons_cell *) arg)->car_pointer[0];
  arg2 = (long)((struct cons_cell *) arg)->cdr_pointer[0];
  arg2 = (long)((struct cons_cell *) arg2)->car_pointer[0];
  pua( (long)arg1 );
  pua( (long)arg2 );
  puj( (long)SET0 );
  puj( (long)PUSH_RESULT_ON_APDL );
  puj( (long)EVAL_0 );
  prog_switch=poj();break;
```

```
/**/
case SET0 :
  arg2 = (long)poa();
  arg1 = (long)poa();
  ((struct atom_header *) arg1)->vcell[0] = arg2;
  ((struct atom_header *) result)->vcell[0] = arg2;
  prog_switch=poj();break;
/**/
case NULL :
  arg = (long)poa();
  arg = (long)((struct cons_cell *) arg)->car_pointer[0];
  if ( (long)  arg == (long) nil ){ goto isnil;};
    ((struct atom_header *) result)->vcell[0] = nil;
    prog_switch=poj();break;
  isnil:
    ((struct atom_header *) result)->vcell[0] = t;
    prog_switch=poj();break;
/**/
case ATOM :
  arg = (long)poa();
  arg = (long)((struct cons_cell *) arg)->car_pointer[0];
  pua( (long)arg );
  puj( (long)ATOM_1 );
  prog_switch=poj();break;
/**/
case EQ :
  arg = (long)poa();
  arg1 = (long)((struct cons_cell *) arg)->car_pointer[0];
  arg2 = (long)((struct cons_cell *) arg)->cdr_pointer[0];
  arg2 = (long)((struct cons_cell *) arg2)->car_pointer[0];
  if ( (long) arg1 != (long) arg2 ){ goto noteq;};
    ((struct atom_header *) result)->vcell[0] = t;
    prog_switch=poj();break;
  noteq:
    ((struct atom_header *) result)->vcell[0] = nil;
    prog_switch=poj();break;
/**/
case RPLACD :
  arg = (long)poa();
  arg1 = (long)((struct cons_cell *) arg)->car_pointer[0];
  arg2 = (long)((struct cons_cell *) arg)->cdr_pointer[0];
  arg2 = (long)((struct cons_cell *) arg2)->car_pointer[0];
  ((struct cons_cell *) arg1)->cdr_pointer[0] = arg2;
  ((struct atom_header *) result)->vcell[0] = arg1;
  prog_switch=poj();break;
/**/
case RPLACA :
  arg = (long)poa();
  arg1 = (long)((struct cons_cell *) arg)->car_pointer[0];
  arg2 = (long)((struct cons_cell *) arg)->cdr_pointer[0];
  arg2 = (long)((struct cons_cell *) arg2)->car_pointer[0];
  ((struct cons_cell *) arg1)->car_pointer[0] = arg2;
  ((struct atom_header *) result)->vcell[0] = arg1;
```

```
    prog_switch=poj();break;
  /**/
  case NCONC :
    arg = (long)poa();
    arg1 = (long)((struct cons_cell *) arg)->car_pointer[0];
    arg2 = (long)((struct cons_cell *) arg)->cdr_pointer[0];
    arg2 = (long)((struct cons_cell *) arg2)->car_pointer[0];
    pua( (long)arg2 );
    pua( (long)arg1 );
    puj( (long)NCONC_1 );
    prog_switch=poj();break;
  /**/
  case PUTP :
    arg = (long)poa();
    atomname = (long)((struct cons_cell *) arg)->car_pointer[0];
    arg = (long)((struct cons_cell *) arg)->cdr_pointer[0];
    propname = (long)((struct cons_cell *) arg)->car_pointer[0];
    arg1 = (long)((struct cons_cell *) arg)->cdr_pointer[0];
    propval = (long)((struct cons_cell *) arg1)->car_pointer[0];
    search = (long)((struct atom_header *) atomname)->plist[0];
    putp_1:
      if ( (long)  search == (long) nil ){ goto putnf;};
      temp = (long)((struct cons_cell *) search)->car_pointer[0];
      if ( (long)  propname == (long) temp ){ goto putf;};
      search = (long)((struct cons_cell *) search)->cdr_pointer[0];
      search = (long)((struct cons_cell *) search)->cdr_pointer[0];
      goto putp_1;
    putnf:
      ((struct cons_cell *) arg1)->cdr_pointer[0] =
          ((struct atom_header *) atomname)->plist[0];
      ((struct atom_header *) atomname)->plist[0] = arg;
      ((struct atom_header *) result)->vcell[0] = nil;
      prog_switch=poj();break;
    putf:
      temp = (long)((struct cons_cell *) search)->cdr_pointer[0];
      ((struct cons_cell *) temp)->car_pointer[0] = propval;
      ((struct atom_header *) result)->vcell[0] = nil;
      prog_switch=poj();break;
  /**/
  case GETP :
    arg = (long)poa();
    atomname = (long)((struct cons_cell *) arg)->car_pointer[0];
    arg = (long)((struct cons_cell *) arg)->cdr_pointer[0];
    if ( (long)  arg == (long) nil ){ goto getpall;};
    propname = (long)((struct cons_cell *) arg)->car_pointer[0];
    search = (long)((struct atom_header *) atomname)->plist[0];
    getp1:
      if ( (long)  search == (long) nil ){ goto getpnf;};
      temp = (long)((struct cons_cell *) search)->car_pointer[0];
      if ( (long)  propname == (long) temp ){ goto getpf;};
      search = (long)((struct cons_cell *) search)->cdr_pointer[0];
      search = (long)((struct cons_cell *) search)->cdr_pointer[0];
      goto getp1;
```

```
    getpnf:
      ((struct atom_header *) result)->vcell[0] = nil;
      prog_switch=poj();break;
    getpf:
      ((struct atom_header *) result)->vcell[0]
          = ((struct cons_cell *) search)->cdr_pointer[0];
      puj( (long)QUOTE );
      puj( (long)PUSH_RESULT_ON_APDL );
      prog_switch=poj();break;
    getpall:
      ((struct atom_header *) result)->vcell[0]
          = ((struct atom_header *) atomname)->plist[0];
      prog_switch=poj();break;
 /**/
 case MKSMALL :
    arg = (long)poa();
    temp = (long)&atom_sp[(add_atom_sp += size_atom_header)
     - size_atom_header] ;
    ((struct atom_header *) result)->vcell[0] = temp;
    ((struct atom_header *) ((struct atom_header *) result)->vcell[0] )
          ->vcell[0] = arg;
    ((struct atom_header *) ((struct atom_header *) result)->vcell[0] )
          ->cell_type[0] = C_SNUM;
    prog_switch=poj();break;
 /**/
 case ADD :
    arg = (long)poa();
    arg1 = (long)((struct cons_cell *) arg)->car_pointer[0];
    arg2 = (long)((struct cons_cell *) arg)->cdr_pointer[0];
    arg2 = (long)((struct cons_cell *) arg2)->car_pointer[0];
    arg1 = (long)((struct atom_header *) arg1)->vcell[0];
    arg2 = (long)((struct atom_header *) arg2)->vcell[0];
    ((struct atom_header *) result)->vcell[0] = arg1+ arg2;
    pua( (long)((struct atom_header *) result)->vcell[0] );
    puj( (long)MKSMALL );
    prog_switch=poj();break;
 /**/
 case SUB :
    arg = (long)poa();
    arg1 = (long)((struct cons_cell *) arg)->car_pointer[0];
    arg2 = (long)((struct cons_cell *) arg)->cdr_pointer[0];
    arg2 = (long)((struct cons_cell *) arg2)->car_pointer[0];
    arg1 = (long)((struct atom_header *) arg1)->vcell[0];
    arg2 = (long)((struct atom_header *) arg2)->vcell[0];
    ((struct atom_header *) result)->vcell[0] = arg1-arg2;
    pua( (long)((struct atom_header *) result)->vcell[0] );
    puj( (long)MKSMALL );
    prog_switch=poj();break;
 /**/
 case MUL :
    arg = (long)poa();
    arg1 = (long)((struct cons_cell *) arg)->car_pointer[0];
    arg2 = (long)((struct cons_cell *) arg)->cdr_pointer[0];
```

```
    arg2 = (long)((struct cons_cell *) arg2)->car_pointer[0];
    arg1 = (long)((struct atom_header *) arg1)->vcell[0];
    arg2 = (long)((struct atom_header *) arg2)->vcell[0];
    ((struct atom_header *) result)->vcell[0] = arg1*arg2;
    pua( (long)((struct atom_header *) result)->vcell[0] );
    puj( (long)MKSMALL );
    prog_switch=poj();break;
  /**/
  case DIV :
    arg = (long)poa();
    arg1 = (long)((struct cons_cell *) arg)->car_pointer[0];
    arg2 = (long)((struct cons_cell *) arg)->cdr_pointer[0];
    arg2 = (long)((struct cons_cell *) arg2)->car_pointer[0];
    arg1 = (long)((struct atom_header *) arg1)->vcell[0];
    arg2 = (long)((struct atom_header *) arg2)->vcell[0];
    ((struct atom_header *) result)->vcell[0] = arg1/arg2;
    pua( (long)((struct atom_header *) result)->vcell[0] );
    puj( (long)MKSMALL );
    prog_switch=poj();break;
  /**/
  case MOD :
    arg = (long)poa();
    arg1 = (long)((struct cons_cell *) arg)->car_pointer[0];
    arg2 = (long)((struct cons_cell *) arg)->cdr_pointer[0];
    arg2 = (long)((struct cons_cell *) arg2)->car_pointer[0];
    arg1 = (long)((struct atom_header *) arg1)->vcell[0];
    arg2 = (long)((struct atom_header *) arg2)->vcell[0];
    ((struct atom_header *) result)->vcell[0] = arg1%arg2;
    pua( (long)((struct atom_header *) result)->vcell[0] );
    puj( (long)MKSMALL );
    prog_switch=poj();break;
  /**/
  case GNUM :
    arg = (long)poa();
    arg1 = (long)((struct cons_cell *) arg)->car_pointer[0];
    arg2 = (long)((struct cons_cell *) arg)->cdr_pointer[0];
    arg2 = (long)((struct cons_cell *) arg2)->car_pointer[0];
    arg1 = (long)((struct atom_header *) arg1)->vcell[0];
    arg2 = (long)((struct atom_header *) arg2)->vcell[0];
    if ( (long) arg1 > (long) arg2 ){ goto gnumt;};
      ((struct atom_header *) result)->vcell[0] = nil;
      prog_switch=poj();break;
    gnumt:
    ((struct atom_header *) result)->vcell[0] = t;
    prog_switch=poj();break;
  /**/
  case LNUM :
    arg = (long)poa();
    arg1 = (long)((struct cons_cell *) arg)->car_pointer[0];
    arg2 = (long)((struct cons_cell *) arg)->cdr_pointer[0];
    arg2 = (long)((struct cons_cell *) arg2)->car_pointer[0];
    arg1 = (long)((struct atom_header *) arg1)->vcell[0];
    arg2 = (long)((struct atom_header *) arg2)->vcell[0];
```

```
    if ( (long) arg1 < (long) arg2 ){ goto lnumt;};
      ((struct atom_header *) result)->vcell[0] = nil;
      prog_switch=poj();break;
    lnumt:
    ((struct atom_header *) result)->vcell[0] = t;
    prog_switch=poj();break;
  /**/
  case GENUM :
    arg = (long)poa();
    arg1 = (long)((struct cons_cell *) arg)->car_pointer[0];
    arg2 = (long)((struct cons_cell *) arg)->cdr_pointer[0];
    arg2 = (long)((struct cons_cell *) arg2)->car_pointer[0];
    arg1 = (long)((struct atom_header *) arg1)->vcell[0];
    arg2 = (long)((struct atom_header *) arg2)->vcell[0];
    if ( (long) arg1 >= (long) arg2 ){ goto genumt;};
      ((struct atom_header *) result)->vcell[0] = nil;
      prog_switch=poj();break;
    genumt:
    ((struct atom_header *) result)->vcell[0] = t;
    prog_switch=poj();break;
  /**/
  case EQNUM :
    arg = (long)poa();
    arg1 = (long)((struct cons_cell *) arg)->car_pointer[0];
    arg2 = (long)((struct cons_cell *) arg)->cdr_pointer[0];
    arg2 = (long)((struct cons_cell *) arg2)->car_pointer[0];
    arg1 = (long)((struct atom_header *) arg1)->vcell[0];
    arg2 = (long)((struct atom_header *) arg2)->vcell[0];
    if ( (long) arg1 == (long) arg2 ){ goto eqnumt;};
      ((struct atom_header *) result)->vcell[0] = nil;
      prog_switch=poj();break;
    eqnumt:
    ((struct atom_header *) result)->vcell[0] = t;
    prog_switch=poj();break;
  /**/
  case LENUM :
    arg = (long)poa();
    arg1 = (long)((struct cons_cell *) arg)->car_pointer[0];
    arg2 = (long)((struct cons_cell *) arg)->cdr_pointer[0];
    arg2 = (long)((struct cons_cell *) arg2)->car_pointer[0];
    arg1 = (long)((struct atom_header *) arg1)->vcell[0];
    arg2 = (long)((struct atom_header *) arg2)->vcell[0];
    if ( (long) arg1 <= (long) arg2 ){ goto lenumt;};
      ((struct atom_header *) result)->vcell[0] = nil;
      prog_switch=poj();break;
    lenumt:
    ((struct atom_header *) result)->vcell[0] = t;
    prog_switch=poj();break;
  /**/
  case EXIT :
    write((int)C_OUTUNIT,(char *)"CORVALLISP END0,15);
    read((int)inunit,(char *)&buff[0] ,1);
    puj( (long)EXIT_FLAG );
```

```
  prog_switch=poj();break;
/**/
case OBLIST :
  arg = (long)poa();
  loc = (long)0;
  ob_loop:
  if ( (long)  loc == (long) C_OBLIST_LEN ){ goto doneob;};
  search = (long)oblist[loc] ;
  loc++;
  obsloop:
    if ( (long)  search == (long) nil ){ goto ob_loop;};
    pointer = (long)((struct cons_cell *) search)->cdr_pointer[0];

    printf("%d: %s, %x0,
        loc, &pnamesp[((struct atom_header *) pointer)->pname[0]]
        ,((struct atom_header *) pointer)->vcell[0]);
    search = (long)((struct cons_cell *) search)->car_pointer[0];
    goto obsloop;
  doneob:
  ((struct atom_header *) result)->vcell[0] = t;
  prog_switch=poj();break;
}}
cons_1() {
  ((struct atom_header *) aarg2)->vcell[0] = (long)poa();
  ((struct atom_header *) aarg1)->vcell[0] = (long)poa();
  mkcons();
  ((struct atom_header *) result)->vcell[0] = poa();
  ((struct cons_cell *) ((struct atom_header *) result)->
   vcell[0])->car_pointer[0] = ((struct atom_header *)
       aarg1)->vcell[0];
  ((struct cons_cell *) ((struct atom_header *) result)->
   vcell[0])->cdr_pointer[0] =
       ((struct atom_header *) aarg2)->vcell[0];
  return;
}
/**/
init() {
  add_atom_sp = 0;bot_atom_sp=len_atom_sp-1;
  add_cons_sp = 0;bot_cons_sp=len_cons_sp-1;
  add_strg_sp = 0;bot_strg_sp=len_strg_sp-1;
  add_apdl = 0;bot_apdl=len_apdl-1;
  add_jpdl = 0;bot_jpdl=len_jpdl-1;
  add_pnamesp = 0;bot_pnamesp=len_pnamesp-1;
  nil = (long)&atom_sp[(add_atom_sp += size_atom_header)
   - size_atom_header] ;
  keep_cons = (long)&atom_sp[(add_atom_sp += size_atom_header)
   - size_atom_header] ;
  keeper = (long)&atom_sp[(add_atom_sp += size_atom_header)
   - size_atom_header] ;
  keeper2 = (long)&atom_sp[(add_atom_sp += size_atom_header)
   - size_atom_header] ;
  aarg1 = (long)&atom_sp[(add_atom_sp += size_atom_header)
   - size_atom_header] ;
```

```
  aarg2 = (long)&atom_sp[(add_atom_sp += size_atom_header)
   - size_atom_header] ;
  add_atom_sp = 0;bot_atom_sp=len_atom_sp-1;
  inunit = (long)C_INUNIT;;
  outunit = (long)C_OUTUNIT;;
/**/
  arg = (long)&cons_sp[(add_cons_sp += size_cons_cell)
   - size_cons_cell] ;
  gc_warn = (long)0;
initlop:
  gc_warn++;
  if ( !( ( size_cons_cell+ add_cons_sp >= bot_cons_sp ? 0 : 1) ))
   { goto doneinit ;};
  temp = (long)&cons_sp[(add_cons_sp += size_cons_cell)
   - size_cons_cell] ;
  ((struct cons_cell *) arg)->car_pointer[0] = temp;
  arg = (long)temp;
  goto initlop;
  doneinit:
    gc_warn = (long)gc_warn/4;
    add_cons_sp = 0;bot_cons_sp=len_cons_sp-1;
    avail_cons = (long)&cons_sp[(add_cons_sp += size_cons_cell)
        - size_cons_cell] ;
    ((struct cons_cell *) arg)->car_pointer[0] = nil;
  loc_pnsp = (long)0;
  lisp_level = (long)2;
  pass = (long)&strg_sp[(add_strg_sp += size_string_args)
   - size_string_args] ;
  counter = (long)0;
  initloop:
    if ( (long)  counter == (long) C_OBLIST_LEN ){ goto donelp;};
  oblist[counter] = nil;
  counter = (long)counter+ 1;
  goto initloop;
donelp:
  strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"nil");
       pua( (long)pass );
  intern();
  arg = (long)poa();
  nil = (long)arg;
  ((struct atom_header *) arg)->vcell[0] = nil;
  ((struct cons_cell *) arg)->cell_type[0] = C_ATOM_HEDER;
  strcpy((char *)&((struct string_args *) pass)->st[0],
   (char *)"keep_cons");
       pua( (long)pass );
  intern();
  arg = (long)poa();
  keep_cons = (long)arg;
  ((struct atom_header *) arg)->vcell[0] = nil;
  ((struct cons_cell *) arg)->cell_type[0] = C_ATOM_HEDER;

  strcpy((char *)&((struct string_args *) pass)->st[0],
   (char *)"keeper");
```

```
      pua( (long)pass );
intern();
arg = (long)poa();
keeper = (long)arg;
((struct atom_header *) arg)->vcell[0] = nil;
((struct cons_cell *) arg)->cell_type[0] = C_ATOM_HEDER;

strcpy((char *)&((struct string_args *) pass)->st[0],
  (char *)"keeper2");
      pua( (long)pass );
intern();
arg = (long)poa();
keeper2 = (long)arg;
((struct atom_header *) arg)->vcell[0] = nil;
((struct cons_cell *) arg)->cell_type[0] = C_ATOM_HEDER;

strcpy((char *)&((struct string_args *) pass)->st[0],
  (char *)"aarg1");
      pua( (long)pass );
intern();
arg = (long)poa();
aarg1 = (long)arg;
((struct atom_header *) arg)->vcell[0] = nil;
((struct cons_cell *) arg)->cell_type[0] = C_ATOM_HEDER;

strcpy((char *)&((struct string_args *) pass)->st[0],
  (char *)"aarg2");
      pua( (long)pass );
intern();
arg = (long)poa();
aarg2 = (long)arg;
((struct atom_header *) arg)->vcell[0] = nil;
((struct cons_cell *) arg)->cell_type[0] = C_ATOM_HEDER;
strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"t");
      pua( (long)pass );
intern();
arg = (long)poa();
t = (long)arg;
((struct atom_header *) arg)->vcell[0] = t;
((struct cons_cell *) arg)->cell_type[0] = C_ATOM_HEDER;
strcpy((char *)&((struct string_args *) pass)->st[0],
  (char *)"result");
      pua( (long)pass );
intern();
arg = (long)poa();
result = (long)arg;
((struct atom_header *) arg)->vcell[0] = t;
((struct cons_cell *) arg)->cell_type[0] = C_ATOM_HEDER;
strcpy((char *)&((struct string_args *) pass)->st[0],
  (char *)"front");
      pua( (long)pass );
intern();
arg = (long)poa();
```

```
front = (long)arg;
((struct atom_header *) arg)->vcell[0] = t;
((struct cons_cell *) arg)->cell_type[0] = C_ATOM_HEDER;
strcpy((char *)&((struct string_args *) pass)->st[0],
  (char *)"oblist");
      pua( (long)pass );
intern();
arg = (long)poa();
((struct atom_header *) arg)->vcell[0] = OBLIST;
((struct cons_cell *) arg)->cell_type[0] = C_FSUB;
strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"gc");
      pua( (long)pass );
 intern();
 arg = (long)poa();
 ((struct atom_header *) arg)->vcell[0] = GC;
 ((struct cons_cell *) arg)->cell_type[0] = C_FSUB;
strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"room");
      pua( (long)pass );
 intern();
 arg = (long)poa();
 ((struct atom_header *) arg)->vcell[0] = ROOM;
 ((struct cons_cell *) arg)->cell_type[0] = C_FSUB;
strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"putp");
      pua( (long)pass );
intern();
arg = (long)poa();
((struct atom_header *) arg)->vcell[0] = PUTP;
((struct cons_cell *) arg)->cell_type[0] = C_SUB;
strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"=");
      pua( (long)pass );
intern();
arg = (long)poa();
((struct atom_header *) arg)->vcell[0] = EQNUM;
((struct cons_cell *) arg)->cell_type[0] = C_SUB;
strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"<=");
      pua( (long)pass );
intern();
arg = (long)poa();
((struct atom_header *) arg)->vcell[0] = LENUM;
((struct cons_cell *) arg)->cell_type[0] = C_SUB;
strcpy((char *)&((struct string_args *) pass)->st[0],(char *)">=");
      pua( (long)pass );
intern();
arg = (long)poa();
((struct atom_header *) arg)->vcell[0] = GENUM;
((struct cons_cell *) arg)->cell_type[0] = C_SUB;
strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"<");
      pua( (long)pass );
intern();
arg = (long)poa();
((struct atom_header *) arg)->vcell[0] = LNUM;
((struct cons_cell *) arg)->cell_type[0] = C_SUB;
strcpy((char *)&((struct string_args *) pass)->st[0],(char *)">");
```

```
        pua( (long)pass );
    intern();
    arg = (long)poa();
    ((struct atom_header *) arg)->vcell[0] = GNUM;
    ((struct cons_cell *) arg)->cell_type[0] = C_SUB;
    strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"%");
        pua( (long)pass );
    intern();
    arg = (long)poa();
    ((struct atom_header *) arg)->vcell[0] = MOD;
    ((struct cons_cell *) arg)->cell_type[0] = C_SUB;
    strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"/");
        pua( (long)pass );
    intern();
    arg = (long)poa();
    ((struct atom_header *) arg)->vcell[0] = DIV;
    ((struct cons_cell *) arg)->cell_type[0] = C_SUB;
    strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"+ ");
        pua( (long)pass );
    intern();
    arg = (long)poa();
    ((struct atom_header *) arg)->vcell[0] = ADD;
    ((struct cons_cell *) arg)->cell_type[0] = C_SUB;
    strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"-");
        pua( (long)pass );
    intern();
    arg = (long)poa();
    ((struct atom_header *) arg)->vcell[0] = SUB;
    ((struct cons_cell *) arg)->cell_type[0] = C_SUB;
    strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"*");
        pua( (long)pass );
    intern();
    arg = (long)poa();
    ((struct atom_header *) arg)->vcell[0] = MUL;
    ((struct cons_cell *) arg)->cell_type[0] = C_SUB;
    strcpy((char *)&((struct string_args *) pass)->st[0],
     (char *)"getp");
        pua( (long)pass );
    intern();
    arg = (long)poa();
    ((struct atom_header *) arg)->vcell[0] = GETP;
    ((struct cons_cell *) arg)->cell_type[0] = C_SUB;
strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"reset");
      pua( (long)pass );
intern();
arg = (long)poa();
((struct atom_header *) arg)->vcell[0] = RESET;
((struct cons_cell *) arg)->cell_type[0] = C_FSUB;
  strcpy((char *)&((struct string_args *) pass)->st[0],
   (char *)"return");
       pua( (long)pass );
  intern();
  arg = (long)poa();
```

```
((struct atom_header *) arg)->vcell[0] = RETURN;
((struct cons_cell *) arg)->cell_type[0] = C_SUB;
strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"EVAL");
      pua( (long)pass );
intern();
arg = (long)poa();
((struct atom_header *) arg)->vcell[0] = EVAL;
((struct cons_cell *) arg)->cell_type[0] = C_SUB;
strcpy((char *)&((struct string_args *) pass)->st[0],
  (char *)"progn");
      pua( (long)pass );
intern();
arg = (long)poa();
((struct atom_header *) arg)->vcell[0] = PROGN;
((struct cons_cell *) arg)->cell_type[0] = C_FSUB;
strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"cond");
      pua( (long)pass );
intern();
arg = (long)poa();
((struct atom_header *) arg)->vcell[0] = COND;
((struct cons_cell *) arg)->cell_type[0] = C_FSUB;
strcpy((char *)&((struct string_args *) pass)->st[0],
  (char *)"evlis");
      pua( (long)pass );
intern();
arg = (long)poa();
((struct atom_header *) arg)->vcell[0] = EVLIS;
((struct cons_cell *) arg)->cell_type[0] = C_FSUB;
strcpy((char *)&((struct string_args *) pass)->st[0],
  (char *)"append");
      pua( (long)pass );
intern();
arg = (long)poa();
((struct atom_header *) arg)->vcell[0] = APPEND;
((struct cons_cell *) arg)->cell_type[0] = C_SUB;
strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"last");
      pua( (long)pass );
intern();
arg = (long)poa();
((struct atom_header *) arg)->vcell[0] = LAST;
((struct cons_cell *) arg)->cell_type[0] = C_SUB;
strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"car");
      pua( (long)pass );
intern();
arg = (long)poa();
((struct atom_header *) arg)->vcell[0] = CAR;
((struct cons_cell *) arg)->cell_type[0] = C_SUB;
strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"cdr");
      pua( (long)pass );
intern();
arg = (long)poa();
((struct atom_header *) arg)->vcell[0] = CDR;
((struct cons_cell *) arg)->cell_type[0] = C_SUB;
```

```
strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"cons");
      pua( (long)pass );
intern();
arg = (long)poa();
((struct atom_header *) arg)->vcell[0] = CONS;
((struct cons_cell *) arg)->cell_type[0] = C_SUB;
strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"SET");
      pua( (long)pass );
intern();
arg = (long)poa();
((struct atom_header *) arg)->vcell[0] = SET;
((struct cons_cell *) arg)->cell_type[0] = C_SUB;
strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"setq");
      pua( (long)pass );
intern();
arg = (long)poa();
((struct atom_header *) arg)->vcell[0] = SETQ;
((struct cons_cell *) arg)->cell_type[0] = C_FSUB;
strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"null");
      pua( (long)pass );
intern();
arg = (long)poa();
((struct atom_header *) arg)->vcell[0] = NULL;
((struct cons_cell *) arg)->cell_type[0] = C_SUB;
strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"atom");
      pua( (long)pass );
intern();
arg = (long)poa();
((struct atom_header *) arg)->vcell[0] = ATOM;
((struct cons_cell *) arg)->cell_type[0] = C_SUB;
strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"EQ");
      pua( (long)pass );
intern();
arg = (long)poa();
((struct atom_header *) arg)->vcell[0] = EQ;
((struct cons_cell *) arg)->cell_type[0] = C_SUB;
strcpy((char *)&((struct string_args *) pass)->st[0],
 (char *)"rplacd");
      pua( (long)pass );
intern();
arg = (long)poa();
((struct atom_header *) arg)->vcell[0] = RPLACD;
((struct cons_cell *) arg)->cell_type[0] = C_SUB;
strcpy((char *)&((struct string_args *) pass)->st[0],
 (char *)"rplaca");
      pua( (long)pass );
intern();
arg = (long)poa();
((struct atom_header *) arg)->vcell[0] = RPLACA;
((struct cons_cell *) arg)->cell_type[0] = C_SUB;
strcpy((char *)&((struct string_args *) pass)->st[0],
 (char *)"nconc");
      pua( (long)pass );
```

```
    intern();
    arg = (long)poa();
    ((struct atom_header *) arg)->vcell[0] = NCONC;
    ((struct cons_cell *) arg)->cell_type[0] = C_SUB;
    strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"de");
          pua( (long)pass );
    intern();
    arg = (long)poa();
    ((struct atom_header *) arg)->vcell[0] = DE;
    ((struct cons_cell *) arg)->cell_type[0] = C_FSUB;
    strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"df");
          pua( (long)pass );
    intern();
    arg = (long)poa();
    ((struct atom_header *) arg)->vcell[0] = DF;
    ((struct cons_cell *) arg)->cell_type[0] = C_FSUB;
    strcpy((char *)&((struct string_args *) pass)->st[0],
      (char *)"quote");
          pua( (long)pass );
    intern();
    arg = (long)poa();
    ((struct atom_header *) arg)->vcell[0] = QUOTE;
    ((struct cons_cell *) arg)->cell_type[0] = C_FSUB;
    strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"list");
          pua( (long)pass );
    intern();
    arg = (long)poa();
    ((struct atom_header *) arg)->vcell[0] = LIST;
    ((struct cons_cell *) arg)->cell_type[0] = C_SUB;
    strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"or");
          pua( (long)pass );
    intern();
    arg = (long)poa();
    ((struct atom_header *) arg)->vcell[0] = OR;
    ((struct cons_cell *) arg)->cell_type[0] = C_SUB;
    strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"and");
          pua( (long)pass );
    intern();
    arg = (long)poa();
    ((struct atom_header *) arg)->vcell[0] = AND;
    ((struct cons_cell *) arg)->cell_type[0] = C_SUB;
/**/
    strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"load");
    pua( (long)pass );
    intern();
    arg = (long)poa();
    ((struct atom_header *) arg)->vcell[0] = LOAD;
    ((struct cons_cell *) arg)->cell_type[0] = C_FSUB;
    strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"read");
          pua( (long)pass );
    intern();
    arg = (long)poa();
    ((struct atom_header *) arg)->vcell[0] = READ;
```

```
    ((struct cons_cell *) arg)->cell_type[0] = C_FSUB;
    strcpy((char *)&((struct string_args *) pass)->st[0],
      (char *)"print");
          pua( (long)pass );
    intern();
    arg = (long)poa();
    ((struct atom_header *) arg)->vcell[0] = PRINT;
    ((struct cons_cell *) arg)->cell_type[0] = C_SUB;
    strcpy((char *)&((struct string_args *) pass)->st[0],(char *)"exit");
          pua( (long)pass );
    intern();
    arg = (long)poa();
    ((struct atom_header *) arg)->vcell[0] = EXIT;
    ((struct cons_cell *) arg)->cell_type[0] = C_FSUB;
}
hasher() {
    counter = (long)0;
    argu = (long)poa();
    strcpy((char *)&print[0] ,(char *)&
      ((struct string_args *) argu)->st[0]);
    hashval = (long)0;
    hloop:
      if ( (long)  print[counter]  == (long) ' ' ){ goto out;};
      hashval = (long)hashval+ print[counter] ;
      counter = (long)counter+ 1;
      goto hloop;
    out:
    hashval = (long)hashval%C_OBLIST_LEN;
    counter+ + ;
    pua( (long)counter );
    pua( (long)hashval );
}
intern() {
    arg = (long)poa();
    /**/
    counter = (long)0;
    temp = (long)0;
    sign = (long)1;
    if ( (long) ((struct string_args *) arg)->st[counter] !=
      (long) '-' ) { goto check_snum;};
      sign = (long)-1;
      counter+ + ;
      if ( (long) ((struct string_args *) arg)->st[counter] >
          (long) '9' ){ goto not_snum;};
      if ( (long) ((struct string_args *) arg)->st[counter] <
          (long) '0' ){ goto not_snum;};
   check_snum:
      if ( (long)  ((struct string_args *) arg)->st[counter]
          == (long) ' ' ){ goto snum_done;};
      if ( (long) ((struct string_args *) arg)->st[counter] >
          (long) '9' ){ goto not_snum;};
      if ( (long) ((struct string_args *) arg)->st[counter] <
          (long) '0' ){ goto not_snum;};
```

```
        temp = (long)temp*10+ ((struct string_args *) arg)->st[counter]-'0';
        counter++;
        goto check_snum;
    snum_done:
      temp = (long)temp*sign;
      pointer = (long)&atom_sp[(add_atom_sp += size_atom_header)
          - size_atom_header] ;
      ((struct atom_header *) pointer)->vcell[0] = temp;
      ((struct atom_header *) pointer)->cell_type[0] = C_SNUM;
      goto found;
    not_snum:
    counter = (long)0;
    pua( (long)arg );  /**/
    hasher();
    loc = (long)poa(); /**/
    counter = (long)poa();
    search = (long)oblist[loc] ;
    old_search = (long)search;
    sloop:
      if ( (long)  search === (long) nil ){ goto notfound;};
      pointer = (long)((struct cons_cell *) search)->cdr_pointer[0];
      ploc = (long)((struct atom_header *) pointer)->pname[0];
      a_found = (long)strcmp((char *)&pnamesp[ploc] ,
          (char *)&((struct string_args *) arg)->st[0]);;
      if ( !( a_found )){ goto found ;};
      old_search = (long)search;
      search = (long)((struct cons_cell *) search)->car_pointer[0];
      goto sloop;
    notfound:
      mkcons();
      temp = (long)poa();
      ((struct cons_cell *) temp)->car_pointer[0] = nil;
      arg1 = (long)&atom_sp[(add_atom_sp += size_atom_header) -
          size_atom_header] ;
      strcpy((char *)&pnamesp[loc_pnsp] ,(char *)
          &((struct string_args *) arg)->st[0]);
      ((struct atom_header *) arg1)->pname[0] = loc_pnsp;
      loc_pnsp = (long)loc_pnsp+ counter;
      ((struct cons_cell *) temp)->cdr_pointer[0] = arg1;
      ((struct atom_header *) arg1)->vcell[0] = nil;
      ((struct cons_cell *) arg1)->cell_type[0] = C_ATOM_HEDER;
      ((struct atom_header *) arg1)->plist[0] = nil;
      if ( (long)  search === (long) old_search ){ goto emptybucket;};
        ((struct cons_cell *) old_search)->car_pointer[0] = temp;
        pointer = (long)arg1;
        goto found;
      emptybucket:
        oblist[loc] = temp;
        pointer = (long)arg1;
    found:
      pua( (long)pointer );
}
mkcons() {
```

```
  if ( (long)  avail_cons == (long) nil ){ goto nosp;};
  makecell:
  gc_warn--;
  ((struct cons_cell *) avail_cons)->cell_type[0] = C_CONS_CELL;
  ((struct atom_header *) keep_cons)->vcell[0] = avail_cons;
  pua( (long)avail_cons );
  avail_cons = (long)((struct cons_cell *) avail_cons)->car_pointer[0];
  ((struct cons_cell *) ((struct atom_header *) keep_cons)->vcell[0])
    ->car_pointer[0] = nil;
  ((struct cons_cell *) ((struct atom_header *) keep_cons)->
    vcell[0])->cdr_pointer[0] = nil;
  return;
nosp:
  write((int)C_OUTUNIT,(char *)
    "OUT OF CONS CELL SPACE GC CALLED0,33);
  gc_cons();
  write((int)C_OUTUNIT,(char *)"DONE GC0,8);
  if ( !( (long)  avail_cons == (long) nil )){ goto makecell ;};
  write((int)C_OUTUNIT,(char *)"CANNOT GC CONS_SPACE0,21);
  puj( (long)EXIT_FLAG );
  return;
}
gc_cons() {
  add_cons_sp = 0;bot_cons_sp=len_cons_sp-1;
  put1:
    if ( !( ( size_cons_cell+ add_cons_sp >= bot_cons_sp ? 0 : 1) ))
        { goto doneput1 ;};
    gctemp = (long)&cons_sp[(add_cons_sp + = size_cons_cell)
        - size_cons_cell] ;
    ((struct common_fields *) gctemp)->gc[0] = 1;
    goto put1;
    doneput1:
  mark_atoms();
  mark_cons();
  mark_apdl();
  last = (long)nil;
  avail_cons = (long)nil;
  add_cons_sp = 0;bot_cons_sp=len_cons_sp-1;
  roomcnt = (long)0;
  loopgc:
    if ( !( ( size_cons_cell+ add_cons_sp >= bot_cons_sp ? 0 : 1) ))
        { goto donegc ;};
    gctemp = (long)&cons_sp[(add_cons_sp + = size_cons_cell)
        - size_cons_cell] ;
    if ( !( ((struct common_fields *) gctemp)->gc[0] ))
        { goto loopgc ;};
    if ( (long) avail_cons !=  (long) nil ){ goto notfirst;};
        roomcnt+ + ;
      avail_cons = (long)gctemp;
      last = (long)gctemp;
      ((struct cons_cell *) avail_cons)->car_pointer[0] = nil;
      goto loopgc;
    notfirst:
```

```
        roomcnt++;
      ((struct cons_cell *) last)->car_pointer[0] = gctemp;
      last = (long)gctemp;
      ((struct cons_cell *) gctemp)->car_pointer[0] = nil;
      goto loopgc;
    donegc:
    gc_warn = (long)roomcnt/4;
        printf( "GC: Room %d availcons = %x0, roomcnt, avail_cons);
      return;
}
mark_cons() {
    gccounter = (long)0;
    loopmark:
        if ( (long)  gccounter === (long) C_OBLIST_LEN )
            { goto donemark;};
        gcsearch = (long)oblist[gccounter] ;
        gccounter++;
        if ( (long)  gcsearch === (long) nil ){ goto loopmark;};
        pua( (long)C_MARK );
        pua( (long)gcsearch );
        lpmark:
          gcsearch = (long)poa();
          if ( (long)  gcsearch === (long) C_MARK ){ goto loopmark;};
          if ( (long)  ((struct cons_cell *) gcsearch)
                ->gc[0] === (long) 0 ){ goto lpmark;};
          ((struct cons_cell *) gcsearch)->gc[0] = 0; /**/
          cell_type_1 = (long)
                ((struct cons_cell *) gcsearch)->cell_type[0];
          if ( (long)  cell_type_1 === (long) C_FSUB )
                { goto lpmark;};
          if ( (long)  cell_type_1 === (long) C_SUB ){ goto lpmark;};
          if ( (long)  cell_type_1 === (long) C_SNUM )
                { goto lpmark;};
          if ( (long)  gcsearch === (long) nil ){ goto lpmark;};
          if ( (long)  gcsearch === (long) t ){ goto lpmark;};
          if ( (long)  cell_type_1 === (long) C_FLAM )
                { goto not_cons_cell;};
          if ( (long)  cell_type_1 === (long) C_LAM )
                { goto not_cons_cell;};
            if ( (long)  cell_type_1 === (long) C_ATOM_HEDER )
                { goto not_cons_cell;};
            goto is_cons_cell;
            not_cons_cell:
                pua( (long)((struct atom_header *) gcsearch)->vcell[0] );
            goto lpmark;
            is_cons_cell:
          pua( (long)((struct cons_cell *)
                gcsearch)->cdr_pointer[0] );
          pua( (long)((struct cons_cell *)
                gcsearch)->car_pointer[0] );
        goto lpmark;
    donemark:
      return;
```

```
}
mark_atoms() {
  loc = (long)0;
  ob_loop:
  if ( (long)  loc == (long) C_OBLIST_LEN ){ goto doneob;};
  search = (long)oblist[loc] ;
  loc++;
  obsloop:
    if ( (long)  search == (long) nil ){ goto ob_loop;};
    pointer = (long)((struct cons_cell *) search)->cdr_pointer[0];
    search = (long)((struct cons_cell *) search)->car_pointer[0];
    ((struct atom_header *) pointer)->gc[0] = 1;
    goto obsloop;
  doneob:
  return;
}
mark_apdl() {
long saveapdl;
    gccounter = (long)0;
    saveapdl = (long)add_apdl;
    loopmark:
        if ( (long)  gccounter == (long) saveapdl ){ goto donemark;};
        gcsearch = (long)apdl[gccounter] ;
        gccounter++;
          if ( !(
          (((gcsearch < (long)&(cons_sp[0])) |
          (gcsearch >= (long)&(cons_sp[len_cons_sp])) |
          ((gcsearch-(long)&(cons_sp[0])) % size_cons_cell )) ? 0 : 1)
           )){ goto loopmark ;};
        cell_type_1 =
            (long)((struct cons_cell *) gcsearch)->cell_type[0];
          if ( (long) cell_type_1 !=  (long) C_CONS_CELL ){ goto loopmark;};
        pua( (long)C_MARK );
        pua( (long)gcsearch );
        lpmark:
          gcsearch = (long)poa();
          if ( (long)  gcsearch == (long) C_MARK ){ goto loopmark;};
          cell_type_1 = (long)((struct cons_cell *)
                gcsearch)->cell_type[0];
          if ( (long) cell_type_1 !=
                (long) C_CONS_CELL ){ goto lpmark;};
          if ( (long)  ((struct cons_cell *)
                gcsearch)->gc[0] == (long) 0 ){ goto lpmark;};
          ((struct cons_cell *) gcsearch)->gc[0] = 0; /**/
          pua( (long)((struct cons_cell *)
                gcsearch)->cdr_pointer[0] );
          pua( (long)((struct cons_cell *)
                gcsearch)->car_pointer[0] );
         goto lpmark;
    donemark:
      return;
}
chkpt(str)
```

```
char *str;
{
if (_debug) printf("%s0,str);
}
puj(lip)
long lip;
/*
  This routine pushes a 'LIL' object on the JPDL.
  The module is implemented only for space saving
  purposes.
*/
{
  if ( add_jpdl >= len_jpdl ) { add_jpdl = 0; };
       jpdl[add_jpdl++] = lip;
}



pua(lip)
long lip;
/*
  This routine pushes a 'LIL' object on the APDL.
  The module is implemented only for space saving
  purposes.
*/
{
  if ( add_apdl >= len_apdl ) { add_apdl = 0; };
       apdl[add_apdl++] = lip;
}

long poa()
/*
  This modules pops an object from the apdl
*/
{
  return
  (
    add_apdl == 0 ? apdl[add_apdl=len_apdl-1] : apdl[--add_apdl]
  );
}



long poj()
/*
  This modules pops an object from the jpdl
*/
{
  return
  (
    add_jpdl == 0 ? jpdl[add_jpdl=len_jpdl-1] : jpdl[--add_jpdl]
  );
}
```

Appendix F: Some CORVALLISP examples

This appendix contains some examples of the code written for CORVALLISP. The next appendix contains a sample run of these functions.

```
(de cddr (x) (cdr (cdr x)))
(de caar (x) (car (car x)))
(de cdar (x) (cdr (car x)))
(de cadr (x) (car (cdr x)))
(de 1- (x) (- x 1))
(de 1+ (x) (+ x 1))
(de not (x) (null x))
(de first (x) (car x))
(de second (x) (car (cdr x)))
```

(nthelement lis n) <LAMBDA>

This function returns the nth element of list 'lis'.

```
(de nthelement (lis n)
  (cond
    ((null lis) nil)
    ((atom lis) nil)
    ((> n (length lis)) nil)
    ((= 1 n) (car lis))
    (t (nthelement (cdr lis) (- n 1)))))
```

(equal a b) <LAMBDA>

This Function determines whether all the elements in 'a' are "eq" with all the elements in 'b'.

```
(de equal (a b)
  (cond
    ((and (atom a) (atom b)) (cond ((eq a b))))
    ((or (atom a) (atom b)) nil)
    (t (and (equal (car a) (car b)) (equal (cdr a) (cdr b))))))
```

(atom-count x) <LAMBDA>

This Function counts the number of atoms in list 'x'.

```
(de atom-count (x)
 (cond
  ((null x) 0)
  ((atom x) 1)
  (t (+ (atom-count (car x)) (atom-count (cdr x))))))
```

(length x) <LAMBDA>

This functions counts the number of elements in list 'x'.

```
(de length (x)
 (cond
  ((null x) 0)
  ((atom x) 1)
  (t (+ 1 (length (cdr x))))))
```

(last x) <LAMBDA>

This function returns the last element of the list 'x'.

```
(de last (x)
  (cond
    ((null x) nil)
    ((atom x) x)
    ((null (cdr x)) (car x))
    (t (last (cdr x)))))
```

(apply fn arg) <LAMBDA>

This function runs function 'fn' with 'arg' as its argument.

```
(de apply (fn arg)
 (cond
   ((atom arg) (eval (append (list fn) (list arg))))
   (t (eval (append (list fn) arg)))))
```

(if con true false) <FLAMBDA>

This functions evaluates 'true' if 'con' is non-nil, otherwise 'false' is evaluated.

```
(df if (con then else)
 (cond
   ((eval con) (eval then))
   (t (eval else))))
```

(match a b) <LAMBDA>

This function is equivalent to the function "equal", with the exception that 'a' may contain the wild-card atom '?', which matches any object.

```
(de match (a b)
  (cond
    ((and (atom a) (atom b)) (cond ((or (eq a '?) (eq a b))t)))
    ((or (atom a) (atom b)) nil)
```

```
    (t (and (match (car a) (car b)) (match (cdr a) (cdr b))))))
```

(member lis at) <LAMBDA>

This function returns true if 'at' is a member of the list 'lis'

```
(de member (lis at)
  (cond
    ((atom lis) (eq lis at))
    ((null lis) nil)
    ((eq (car lis) at) t)
    (t (member (cdr lis) at))))
```

(substitute lis at new) <LAMBDA>'

This function substitutes the atom 'new', with atom 'at', in the list 'lis'.

```
(de substitute (lis at new)
   (cond
    ((atom lis) nil)
    ((eq (car lis) at) (cons new (substitute (cdr lis) at new)))
    (t (cons (car lis) (substitute (cdr lis) at new)))))
```

(reverse x) <LAMBDA>

This function reverses the order in which atoms appear in the list 'x'.

```
(de reverse (x)
   (cond
     ((null (cdr x)) x)
     (t (append (reverse (cdr x)) (list (car x))))))
```

(upto lis x) <LAMBDA>

This function returns a list which contains all the atoms in 'lis'
up to, not including the the atom 'x'.

```
(de upto (lis x)
   (cond
     ((null lis) nil)
     ((atom lis) nil)
     ((eq (car lis) x) nil)
     (t (append (list (car lis)) (upto (cdr lis) x)))))
```

(after lis x) <LAMBDA>

This function returns a list which contains all the atoms in 'lis'
from the first appearance of 'x' to the end.

```
(de after (lis x)
  (cond
    ((null lis) nil)
    ((atom lis) nil)
    (t (reverse (upto (reverse lis) x)))))
```

(remove lis x) <LAMBDA>

This function removes atom 'x' from the list 'lis'.

```
(de remove (lis x)
  (cond
    ((null lis) nil)
    ((atom lis) nil)
    (t (append (upto lis x) (after lis x)))))
```

(power a b) <LAMBDA>

This functions returns 'a' to the 'b'th power.

```
(de power (a b)
  (cond
    ((= b 0) 1)
    (t (* a (power a (- b 1))))))
```

(fact x) <LAMBDA>

This function returns the factorial of 'x'.

```
(de fact (x)
    (cond
     ((null x) nil)
     ((null (atom x)) nil)
     ((= 0 x) 1)
     (t (* x (fact (1- x))))))
```

(setfun x y) <LAMBDA>

This function defines a <LAMBDA> function 'x' to be the same as <LAMBDA> function y.

```
(de setfun (x y)
  (eval (append (list 'de x) y)))
```

(sort> x) <LAMBDA>

This function sorts a list of numbers 'x' in descending order.

```
(de sort> (x) (reverse (sort< x)))
```

(sort< x) <LAMBDA>

This function sorts a list of numbers 'x' in ascending order.

```
(de sort< (x)
  (cond
   ((null x) nil)
   ((atom x) x)
   ((null (cdr x)) x)
   (t (merge (sort< (first-half x)) (sort< (second-half x))))))
```

(first-half x) <LAMBDA>

This function returns a list which contains the atoms in the first half of the list 'x'.

```
(de first-half (x)
  (append (upto x (nthelement x (/ (length x) 2)))
    (list (nthelement x (/ (length x) 2)))))
```

(second-half x) <LAMBDA>

This function returns a list which contains the atoms in the second half of the list 'x'.

```
(de second-half (x)
  (after x (nthelement x (/ (length x) 2))))
```

(merge l1 l2) <LAMBDA>

This function merges two sorted lists 'l1' and 'l2'.

```
(de merge (l1 l2)
  (cond
    ((null l1) l2)
    ((null l2) l1)
    (t (cond
      ((> (car l1) (car l2)) (append (list (car l2)) (merge l1 (cdr l2))))
      (t (append (list (car l1)) (merge (cdr l1) l2)))))))
```

(fib x) <LAMBDA>

This functions computes the fibonacci function of 'x'.

```
(de fib (x)
  (cond ( (= x 0) 0)
```

```
  ( (= x 1) 1)
  ( t (+ (fib (1- x)) (fib (- x 2))))))
```

(han num source dest int) <LAMBDA>

This function solves the towers of hanoi puzzle.

```
(de han (num source dest int)
  (cond ((= 0 num) 'done)
    (t (han (- num 1) source int dest)
      (move num source dest)
      (han (- num 1) int dest source))))

(de move (disk from to)
  (print (list 'move disk 'from from 'to to)))
```

DIV function from Gabriel Benchmarks

```
(de nlist (n)
  (cond ((= 0 n))
   (t (cons n (nlist (1- n))))))

(de divrec (l)
  (cond  ((null l) ())
    (t (cons (car l) (divrec (cddr l))))))
```

Tak function From Gabriel Benchmarks

This function runs in 5 minutes and 43 seconds real time with
Franz LISP, and 7 minutes and 4 seconds real time with CORVALLISP.
The "1-" function is interpreted in CORVALLISP, and compiled in Franz.

```
(de tak (x y z)
 (cond ( (null (< y x)) z)
   (t (tak (tak (1- x) y z)
     (tak (1- y) z x)
     (tak (1- z) x y)))))
```

A different variation of Tak function From Gabriel Benchmarks

```
(de tak0 (x y z)
 (cond ((null (< y x)) z)
  (t (tak1 (tak37 (1- x) y z)
    (tak11 (1- y) z x)
    (tak17 (1- z) x y)))))

(de tak1 (x y z)
 (cond ((null (< y x)) z)
  (t (tak2 (tak74 (1- x) y z)
    (tak22 (1- y) z x)
```

```
    (tak34 (1- z) x y)))))

    ....
    ....  tak2 - tak97 follow the same format.
    ....

(de tak98 (x y z)
  (cond ((null (< y x)) z)
   (t (tak99 (tak63 (1- x) y z)
      (tak89 (1- y) z x)
      (tak83 (1- z) x y)))))

(de tak99 (x y z)
  (cond ((null (< y x)) z)
   (t (tak0 (tak0 (1- x) y z)
     (tak0 (1- y) z x)
     (tak0 (1- z) x y)))))
```

Appendix G: A sample run session with CORVALLISP

This appendix contains a sample run session with CORVALLISP.

```
CORVALLISP
> (load ex)
cddr caar cdar cadr 1- 1+ 2- not first second
nthelement equal atom-count length last apply if match
member substitute reverse upto after remove power fact
setfun sort> sort< first-half second-half merge fib nlist
han move nil

> (nthelement '(this is a line ) 4 )
line
> (equal '(this is a line) '(this is a line ))
t
> (equal '(this was a line ) '(this is a line ))
nil
> (atom-count '(this is a line (of (line here ) there ) every-where ))
9
> (length '(this is a line (of (line here ) there ) every-where ))
6
> (last '(this is a line (of (line here ) there ) every-where ))
every-where
> (apply + '(1 3 ))
4
> (if (eq 'this 'that ) (print 'Yes ) (print 'No ))
No
> (if (= 1 1 ) (print 'Yes ) (print 'No ))
Yes
> (match '(this is a line ) '(this is a line ))
t
> (match '(this ? a ? ) '(this is a line ))
t
> (match '(this is a line ) '(that ))
nil
> (member '(this is a line ) 'line )
t
> (substitute '(this is a line ) 'is 'was )
(this was a line )
> (reverse '(1 2 3 4 5 6 7 8 ))
(8 7 6 5 4 3 2 1 )
> (upto '(this is a line of atoms here ) 'a )
(this is )
> (after '(this is a line of atoms here ) 'a )
(line of atoms here )
> (remove '(this is a line of atoms here ) 'a )
(this is line of atoms here )
> (power 10 5 )
100000
> (fact 9 )
362880
```

```
> (sort> '(1 9 2 8 3 7 4 6 5 10 20 11 19 20 40 80 22 33 ))
(80 40 33 22 20 20 19 11 10 9 8 7 6 5 4 3 2 1 )
> (sort< '(1 9 2 8 3 7 4 6 5 10 20 11 19 20 40 80 22 33 ))
(1 2 3 4 5 6 7 8 9 10 11 19 20 20 22 33 40 80 )
> (first-half '(1 2 3 4 5 6 7 8 9 0 ))
(1 2 3 4 5 )
> (second-half '(1 2 3 4 5 6 7 8 9 0 ))
(6 7 8 9 0 )
> (merge '(1 3 5 7 20 19 ) '(2 4 8 22 77 ))
(1 2 3 4 5 7 8 20 19 22 77 )
> (fib 10 )
55
> (nlist 20 )
(20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1)
> (han 5 'First-pole 'Last-pole 'Middle-pole )
(move 1 from First-pole to Last-pole )
(move 2 from First-pole to Middle-pole )
(move 1 from Last-pole to Middle-pole )
(move 3 from First-pole to Last-pole )
(move 1 from Middle-pole to First-pole )
(move 2 from Middle-pole to Last-pole )
(move 1 from First-pole to Last-pole )
(move 4 from First-pole to Middle-pole )
(move 1 from Last-pole to Middle-pole )
(move 2 from Last-pole to First-pole )
(move 1 from Middle-pole to First-pole )
(move 3 from Last-pole to Middle-pole )
(move 1 from First-pole to Last-pole )
(move 2 from First-pole to Middle-pole )
(move 1 from Last-pole to Middle-pole )
(move 5 from First-pole to Last-pole )
(move 1 from Middle-pole to First-pole )
(move 2 from Middle-pole to Last-pole )
(move 1 from First-pole to Last-pole )
(move 3 from Middle-pole to First-pole )
(move 1 from Last-pole to Middle-pole )
(move 2 from Last-pole to First-pole )
(move 1 from Middle-pole to First-pole )
(move 4 from Middle-pole to Last-pole )
(move 1 from First-pole to Last-pole )
(move 2 from First-pole to Middle-pole )
(move 1 from Last-pole to Middle-pole )
(move 3 from First-pole to Last-pole )
(move 1 from Middle-pole to First-pole )
(move 2 from Middle-pole to Last-pole )
(move 1 from First-pole to Last-pole )
done
> (exit)
CORVALLISP END
```