Appendix C

Module Interface Specifications

C.1 Global Definitions

C.1.1 system.h

```
#include <stdio.h>
#include <string.h>

/*****constants*****/

#define SY_EXCFIL "SHAM.excfil"

/*****types*****/

/*sham instructions*/
typedef enum {
     SY_LOAD,SY_STORE,SY_ADD,SY_SUBTRACT,
     SY_BRANCH,SY_BRANCHZERO,SY_BRANCHPOS,SY_LOADCON,SY_PRINT,SY_HALT
} sy_instr;

/*****macros for number of operands*****/

#define SY_OP0(cmd) \
     ((int)cmd == (int)SY_PRINT || (int)cmd == (int)SY_HALT)
#define SY_OP1(cmd) \
     ((int)cmd >= (int)SY_LOAD && (int)cmd <= (int)SY_LOADCON)

/*****variables*****/

extern FILE *sy_excfilp; /*file for exception messages*/
```

C.2 BSHAM Modules

C.2.1 *absmach* MIS

C.2.1.1 Interface syntax

```
#define AM_MEMSIZ 100
#define AM_MAXINT 999

typedef enum {AM_NORMAL,AM_HALT,AM_PRINT,
   AM_ARITHEXC,AM_ADDREXC,
   AM_OBJECTEXC,AM_NOOPEXC
} am_stat;
```

Routine names	Inputs	Outputs	Exceptions
`am_s_init`			
`am_s_acc`	`int`		`am_int`
`am_g_acc`		`int`	
`am_s_pc`	`int`		`am_addr`
`am_g_pc`		`int`	
`am_s_mem`	`int` `int`		`am_addr` `am_int`
`am_g_mem`	`int`	`int`	`am_addr`
`am_sg_exec`		`am_stat`	

C.2.1.2 Interface semantics

state variables
mem : *sequence* [`AM_MEMSIZ`] *of* [0..`AM_MAXINT`]
acc : [0..`AM_MAXINT`]
pc : [0..`AM_MEMSIZ` $- 1$]

state invariant
none

assumptions
`am_s_init` is called before any other access routine.

access routine semantics
`am_s_init`:
  transition: $acc, pc, mem := 0, 0,$all zeroes
  exceptions: none
`am_s_acc`(i):
  transition: $acc := i$
  exceptions: $exc := (i \notin [0..\texttt{AM\_MAXINT}] \Rightarrow \texttt{am\_int})$
`am_g_acc`:
  output: $out := acc$
  exceptions: none
`am_s_pc`(a):

transition: $pc := a$
exceptions: $exc := (a \notin [0..\texttt{AM\_MEMSIZ} - 1] \Rightarrow \texttt{am\_addr})$

`am_g_pc`:
output: $out := pc$
exceptions: none

`am_s_mem`(a, i):
transition: $mem[a] := i$
exceptions: $exc := \quad (a \notin [0..\texttt{AM\_MEMSIZ} - 1] \Rightarrow \texttt{am\_addr}$
$\quad\quad\quad\quad\quad\quad | \; i \notin [0..\texttt{AM\_MAXINT}] \Rightarrow \texttt{am\_int})$

`am_g_mem`(a):
output: $out := mem[a]$
exceptions: $exc := (a \notin [0..\texttt{AM\_MEMSIZ} - 1] \Rightarrow \texttt{am\_addr})$

`am_sg_exec`:
transition-output:
(an error is specified in the Exec. Phase Exception Table $\Rightarrow$
$out :=$ the error identifier
$| \; mem[pc] = \texttt{SY\_HALT} \Rightarrow out := \texttt{AM\_HALT}$
$| \; mem[pc] = \texttt{SY\_PRINT} \Rightarrow out, pc := \texttt{AM\_PRINT}, pc + 1$
$| \; true \Rightarrow \quad out := \texttt{AM\_NORMAL}$
$acc, pc, mem :=$ values specified in the RS Lang. Sem. Table)
exceptions: none

C.2.1.3 Header file: `absmach.h`

```
/*****constants*****/

#define AM_MEMSIZ 100 /*memory size*/
#define AM_MAXINT 999 /*maximum integer value*/

/*****types*****/

typedef enum {AM_NORMAL,AM_HALT,AM_PRINT,
     AM_ARITHEXC,AM_ADDREXC,AM_OBJECTEXC,AM_NOOPEXC
} am_stat;

/*****access routines*****/

void am_s_init();

void am_s_acc();
/*   void am_s_acc(i)
*    int i;
*/

int am_g_acc();

void am_s_pc();
/*   void am_s_pc(a)
```

```
*    int a;
*/

int am_g_pc();

void am_s_mem();
/*   void am_s_mem(a,i)
*    int a,i;
*/

int am_g_mem();
/*   int am_g_mem(a)
*    int a;
*/

am_stat am_sg_exec();

void am_g_dump(); /*for testing purposes only*/

/*****exception handlers*****/

void am_addr();

void am_int();
```

C.2.2 *exec* MIS

C.2.2.1 Interface syntax

Routine names	Inputs	Outputs	Exceptions
ex_s_init			
ex_s_exec			

C.2.2.2 Interface semantics

environment variables
scn
  the terminal screen
stdout
  UNIX standard output

state variables
none

state invariant
none

assumptions

Before `ex_s_exec` is called, `ex_s_init` has been called and
the *absmach* module has been initialized.
At compile time, exactly one of these preprocessor flags is defined:
`BSHAM`, `ISHAM`

access routine semantics
`ex_s_init`:
  transition:
    if flag `ISHAM` is set then
      initialize the screen
  exceptions: none
`ex_s_exec`:
  transition:
    if flag `BSHAM` is set then
      perform the execution phase as described in the BSHAM RS
    else if flag `ISHAM` is set then
      perform the execution phase as described in the ISHAM RS
    In either case:
      • Use the *mem*, *acc*, and *pc* values stored in the *absmach* module
      • Invoke `am_sg_exec` to execute the next instruction
      • Use the `am_sg_exec` return value to determine whether
        a normal case or exception output is needed
  exceptions: none

C.2.2.3 Header file: `exec.h`

```
/*****constants*****/

/*****types*****/

/*****access routines*****/

void ex_s_init();

void ex_s_exec();

/*****exception handlers*****/
```

C.2.3 *load* MIS

C.2.3.1 Interface syntax

```
#define typedef enum {LD_NORMAL,LD_ERROR} ld_stat;
```

Routine names	Inputs	Outputs	Exceptions
`ld_s_init`			
`ld_sg_load`	`FILE*`	`ld_stat`	`ld_fil`

C.2.3.2 Interface semantics

environment variables
stdout
UNIX standard output

state variables
none

state invariant
none

assumptions
`ld_s_init` is called before any other access routine.
The *absmach* and *token* modules have been initialized.
The argument to `ld_sg_load` points to an open file control block.

access routine semantics
`ld_s_init`:
transition: none
exceptions: none
`ld_sg_load`(f): defined in terms of the SHAM Requirements Specification.
transition/output:
(file f has no load errors $\Rightarrow$
absmach.mem := the object code version of the program in f
out := `LD_NORMAL`
| *true* $\Rightarrow$
write the appropriate messages to *stdout*
out := `LD_ERROR`)
exceptions: *exc* := (error reading file $f \Rightarrow$ `ld_fil`)

considerations
In `ld_sg_load`(f), if f has load errors or if `ld_fil` occurs,
the value of *absmach.mem* is "dontcare."

C.2.3.3 Header file: `load.h`

```
/*****constants*****/

/*****types*****/

typedef enum {LD_NORMAL,LD_ERROR} ld_stat;

typedef FILE *ld_filptr;

/*****access routines*****/

void ld_s_init();
```

```
ld_stat ld_sg_load();
/*   ld_stat ld_sg_load(f);
*    ld_filptr f;
*/

/*****exception handlers*****/

void ld_fil();
```

C.2.4 *sham* MIS

There is no MIS for *sham*.

C.2.5 *token* MIS

C.2.5.1 Interface syntax

```
#define TK_MAXSTRLEN 100
#define TK_MAXIDLEN 10
#define TK_MAXINTLEN 5

typedef enum {TK_ID,TK_INT,TK_BADTOK} tk_toktyp;

typedef struct {
   char val[TK_MAXSTRLEN+1];
   tk_toktyp typ;
} tk_valtyp;
```

Routine names	Inputs	Outputs	Exceptions
`tk_s_init`			
`tk_s_str`	`char*`		`tk_maxlen`
`tk_sg_next`		`tk_valtyp`	`tk_end`
`tk_g_end`		*boolean*	

C.2.5.2 Interface semantics

state variables
toklist : *sequence of string*

state invariant
none

assumptions
`tk_s_init` is called before any other access routine.
All string parameters are legal C strings.

access routine semantics
`tk_s_init`:
  transition: $toklist := \langle\rangle$

exceptions: none

`tk_s_str`(s):

transition: $toklist := tokens(s)$

exceptions: $exc := (|s| > \texttt{TK\_MAXSTRLEN} \Rightarrow \texttt{tk\_maxlen})$

`tk_sg_next`:

transition/output: $toklist, out :=$

$toklist[1..|toklist|-1]$,

$\langle toklist[0], toktyp(toklist[0])\rangle$

exceptions: $exc := (toklist = \langle\rangle \Rightarrow \texttt{tk\_end})$

`tk_g_end`:

output: $out := (toklist = \langle\rangle)$

exceptions: none

local types

$idtoksetT = \{s \mid s$ is a string of alphabetic or numeric characters $\wedge$
$s[0]$ is alphabetic $\wedge$ $|s| \in [1..\texttt{TK\_MAXIDLEN}]\}$

$inttoksetT = \{s \mid s$ is a string of numeric characters $\wedge$ $|s| \in [1..\texttt{TK\_MAXINTLEN}]\}$

local functions

$tokens : string \rightarrow sequence\ of\ string$

$tokens(s)$ returns the sequence of tokens in s where

1. a token is a non-empty subsequence $s[i..j]$ of s
2. $s[i..j]$ contains no blanks
3. $(i = 0 \vee s[i-1] = \text{' '}) \wedge (j = |s| - 1 \vee s[j+1] = \text{' '})$

$toktyp : string \rightarrow \texttt{tk\_toktyp}$

$toktyp(s) :=$

$(s \in idtoksetT \Rightarrow \texttt{TK\_ID}$

$\mid s \in inttoksetT \Rightarrow \texttt{TK\_INT}$

$\mid true \Rightarrow \texttt{TK\_BADTOK})$

C.2.5.3 Header file: `token.h`

```
/*****constants*****/

#define TK_MAXSTRLEN 100 /*maximum length of an input string*/
#define TK_MAXIDLEN 10 /*maximum length of an id token*/
#define TK_MAXINTLEN 5 /*maximum length of an integer token*/

/*****types*****/

typedef enum {TK_ID,TK_INT,TK_BADTOK} tk_toktyp;

typedef struct {
     char val[TK_MAXSTRLEN+1];
     tk_toktyp typ;
} tk_valtyp;
```

```
/*****access routines*****/

void tk_s_init();

void tk_s_str();
/*   void tk_s_str(str)
*    char *str;
*/

void tk_sg_next(); /*out value returned using call-by-reference*/
/*   void tk_sg_next(valtyp);
*    tk_valtyp *valtyp; NOTE: caller must allocate *valtyp
*/

/*boolean*/ int tk_g_end();

void tk_g_dump(); /*for testing purposes only*/

/*****exception handlers*****/

void tk_end();

void tk_maxlen();
```

C.3 ISHAM Modules

C.3.1 *keybdin* MIS

C.3.1.1 Interface syntax

Routine names	Inputs	Outputs	Exceptions
ki_s_init			
ki_sg_next		char	
ki_s_end			

C.3.1.2 Interface semantics

environment variables
stdin : *string*
UNIX standard input

state variables
none

state invariant
none

assumptions
The *curses* module has been initialized.

Calls to *keybdin* obey the following pattern:
(`ki_s_init`.`ki_sg_next` $*$.`ki_s_end`)$*$, where $X*$ indicates zero or more occurrences of X

access routine semantics

`ki_s_init`:
  transition: turn off keystroke echoing
  exceptions: none

`ki_sg_next`:
  transition-output: *out* := the next available character
  exceptions: none

`ki_s_end`:
  transition: turn on keystroke echoing
  exceptions: none

considerations

- Keystrokes are returned by `ki_sg_next` in first-in–first-out order.
- Characters are returned immediately, without waiting for a newline.
- If, on entry, there is no new keystroke available, `ki_sg_next` will not return until another keystroke occurs.

C.3.1.3 Header file: `keybdin.h`

```
/*****constants*****/

/*****types*****/

/*****access routines*****/

void ki_s_init();

char ki_sg_next();

void ki_s_end();

/*****exception handlers*****/
```

C.3.2 *scndr* MIS

C.3.2.1 Interface syntax

Routine names	Inputs	Outputs	Exceptions
sd_s_init			
sd_s_clrscn			
sd_s_con			
sd_s_mem			
sd_s_pc			
sd_s_acc			
sd_s_prt	int		
sd_s_msg	char*		
sd_s_hlt	int *boolean*		

C.3.2.2 Interface semantics

environment variables
scn
 the terminal screen

state variables
none

state invariant
none

assumptions
sd_s_init is called before any other access routine.
The *absmach*, *scnstr*, and *scngeom* modules have been initialized.
The address passed to sd_s_hlt is a legal address.

access routine semantics
Note: *MEM*, *PC*, *ACC*, *PRT*, and *MSG* are screen fields from the ISHAM RS.

sd_s_init:
 transition: none
 exceptions: none
sd_s_clrscn:
 transition: clear terminal screen
 exceptions: none
sd_s_con:
 transition: display the fixed screen fields
 exceptions: none
sd_s_mem:
 transition:
 $(\forall r, c \in [0..9]) MEM[r, c] := \texttt{am\_g\_mem}(10 \times r + c)$,
 converted to ASCII, right justified and padded left with blanks

exceptions: none

`sd_s_pc`:

transition: PC := `am_g_pc`, converted to ASCII,
right justified and padded left with blanks

exceptions: none

`sd_s_acc`:

transition: ACC := `am_g_acc`, converted to ASCII,
right justified and padded left with blanks

exceptions: none

`sd_s_prt`(x):

transition: $PRT := x$, converted to ASCII,
right justified and padded left with blanks

exceptions: none

`sd_s_msg`(s):

transition: $MSG := s$, left justified and padded right with blanks

exceptions: none

`sd_s_hlt`(a, f):

transition:

$(f = true \Rightarrow$ display $MEM[a/10, a\%10]$ in inverse video
$|\ f = false \Rightarrow$ display $MEM[a/10, a\%10]$ normally$)$

exceptions: none

considerations

For each field displayed by *scndr*, the value is truncated to the field length returned by *scngeom*.

C.3.2.3 Header file: `scndr.h`

```
/*****constants*****/

/*****types*****/

/*****access routines*****/

void sd_s_init();

void sd_s_clrscn();

void sd_s_con();

void sd_s_mem();

void sd_s_pc();

void sd_s_acc();

void sd_s_prt();
/*   void sd_s_prt(i)
```

```
*    int i;
*/

void sd_s_msg();
/*   void sd_s_msg(s)
*    char *s;
*/

void sd_s_hlt();
/*   void sd_s_hlt(a,f)
*    int a;
*    (boolean) int f;
*/

/*****exception handlers*****/
```

C.3.3 *scngeom* MIS

C.3.3.1 Interface syntax

```
#define SG_NUMROW 24
#define SG_NUMCOL 80

typedef enum {
   SG_MEM,SG_PC,SG_ACC,SG_PRT,SG_MSG,
   SG_SCNTTL,SG_MEMTTL1,SG_MEMTTL2,SG_MEMCOLHDR,SG_MEMROWHDR,
   SG_PCTTL,SG_ACCTTL,SG_PRTTTL,SG_PROMPTTTL,SG_MSGTTL
} sg_fldnam;

typedef struct {
   sg_fldnam nam;
   int row;
   int col;
} sg_fld;
```

Routine names	Inputs	Outputs	Exceptions
sg_s_init			
sg_g_legfld	sg_fld	*boolean*	
sg_g_row	sg_fld	int	sg_badfld
sg_g_col	sg_fld	int	sg_badfld
sg_g_len	sg_fld	int	sg_badfld
sg_g_val	sg_fld	char*	sg_badfld

C.3.3.2 Interface semantics

Identifier	Legal row values	Legal column values	Associated field in ISHAM RS
Variable fields			
`SG_MEM`	[0..9]	[0..9]	*MEM*
`SG_PC`	0	0	*PC*
`SG_ACC`	0	0	*ACC*
`SG_PRT`	0	0	*PRT*
`SG_MSG`	0	0	*MSG*
Fixed fields			
`SG_SCNTTL`	0	0	Screen title
`SG_MEMTTL1`	0	0	*MEM* title line 1
`SG_MEMTTL2`	0	0	*MEM* title line 2
`SG_MEMCOLHDR`	0	[0..9]	*MEM* column header
`SG_MEMROWHDR`	[0..9]	0	*MEM* row header
`SG_PCTTL`	0	0	*PC* title
`SG_ACCTTL`	0	0	*ACC* title
`SG_PRTTTL`	0	0	*PRT* title
`SG_PROMPTTTL`	0	0	Prompt title
`SG_MSGTTL`	0	0	Error message title

state variables
none

state invariant
none

assumptions
`sg_s_init` is called before any other access routine

access routine semantics
`sg_s_init`:
  transition: none
  exceptions: none
`sg_g_legfld`(fld):
  output: out := (fld is a legal field identifier)
  exceptions: none
`sg_g_row`(fld):
  output: out := starting screen row for fld, zero-relative
  exceptions: exc := (fld is not a legal field identifier $\Rightarrow$ `sg_badfld`)
`sg_g_col`(fld):
  output: out := starting screen column for fld, zero-relative
  exceptions: exc := (fld is not a legal field identifier $\Rightarrow$ `sg_badfld`)
`sg_g_len`(`fld`):
  output: out := length of fld
  exceptions: exc := (fld is not a legal field identifier $\Rightarrow$ `sg_badfld`)
`sg_g_val`(fld):
  output: out :=

(fld is a fixed screen field $\Rightarrow$ as shown in the RS
| fld is a variable screen field $\Rightarrow$ "")
exceptions: exc := (fld is not a legal field identifier $\Rightarrow$ `sg_badfld`)

C.3.3.3 Header file: `scngeom.h`

```
/*****constants*****/

#define SG_NUMROW 24 /*number of rows on the screen*/
#define SG_NUMCOL 80 /*number of columns on the screen*/

/*****types*****/

typedef enum {
     SG_MEM,SG_PC,SG_ACC,SG_PRT,SG_MSG,
     SG_SCNTTL,SG_MEMTTL1,SG_MEMTTL2,SG_MEMCOLHDR,SG_MEMROWHDR,
     SG_PCTTL,SG_ACCTTL,SG_PRTTTL,SG_PROMPTTTL,SG_MSGTTL
} sg_fldnam;

typedef struct {
     sg_fldnam nam;
     int row;
     int col;
} sg_fld;

/*****access routines*****/

void sg_s_init();

/*boolean*/ int sg_g_legfld();
/*   int sg_g_legfld(fld)
*    sg_fld fld;
*/

int sg_g_row();
/*   int sg_g_row(fld)
*    sg_fld fld;
*/

int sg_g_col();
/*   int sg_g_col(fld)
*    sg_fld fld;
*/

int sg_g_len();
/*   int sg_g_len(fld)
*    sg_fld fld;
*/
```

```
char *sg_g_val();
/*   char *sg_g_val(fld)
*    sg_fld fld;
*/

/*****exception handlers*****/

void sg_badfld();
```

C.3.4 *scnstr* MIS

C.3.4.1 Interface syntax

```
#define SS_NUMROW 24
#define SS_NUMCOL 80
```

Routine names	Inputs	Outputs	Exceptions
ss_s_init			
ss_s_clrscn			
ss_s_str	int int char*		ss_row ss_col ss_len
ss_s_hlt	int int int *boolean*		ss_row ss_col ss_len
ss_s_cur	int int		ss_row ss_col
ss_s_ref			
ss_s_end			

C.3.4.2 Interface semantics

environment variables
scn : *sequence* [SS_NUMROW][SS_NUMCOL] *of char*
 $scn[r][c]$ is the character at screen row r and column c,
 with numbering zero-relative and beginning at the upper-left corner
hlt : *sequence* [SS_NUMROW][SS_NUMCOL] *of boolean*
 $hlt[r][c]$ is true if the position at screen row r and column c is highlighted,
 with numbering zero-relative and beginning at the upper-left corner
cur : *tuple of* (row : [0..SS_NUMROW − 1], col : [0..SS_NUMCOL − 1])
 the terminal cursor is at screen row $cur.row$ and column $cur.col$
 with numbering zero-relative and beginning at the upper-left corner

state variables
scnbuf : *sequence* [SS_NUMROW][SS_NUMCOL] *of char*
hltbuf : *sequence* [SS_NUMROW][SS_NUMCOL] *of boolean*

$curbuf$: *tuple of* (row : $[0..\texttt{SS\_NUMROW} - 1]$, col : $[0..\texttt{SS\_NUMCOL} - 1]$)

state invariant
none

assumptions
The *curses* module has been initialized.

Calls to *scnstr* obey the following pattern:
$(\texttt{ss\_s\_init}.T * .\texttt{ss\_s\_end})*$, where
T is any call other than `ss_s_init` or `ss_s_end`
$X*$ indicates zero or more occurrences of X
String parameters are legal C strings.

access routine semantics

`ss_s_init`:
transition: none
exceptions: none

`ss_s_clrscn`:
transition: $scnbuf, hltbuf, curbuf := \texttt{all ' '}, \texttt{all} \mathit{false}, \langle 0, 0\rangle$
exceptions: none

`ss_s_str`(row, col, s):
transition: $(|s| > 0 \Rightarrow scnbuf[row][col..col + |s| - 1] := s)$
exceptions: $exc :=$
$(row \notin [0..\texttt{SS\_NUMROW} - 1] \Rightarrow \texttt{ss\_row}$
$|\ col \notin [0..\texttt{SS\_NUMCOL} - 1] \Rightarrow \texttt{ss\_col}$
$|\ |s| \notin [0..\texttt{SS\_NUMCOL} - col] \Rightarrow \texttt{ss\_len})$

`ss_s_hlt`(row, col, l, f):
transition: $(l > 0 \Rightarrow hltbuf[row][col..col + l - 1] := f)$
exceptions: $exc :=$
$(row \notin [0..\texttt{SS\_NUMROW} - 1] \Rightarrow \texttt{ss\_row}$
$|\ col \notin [0..\texttt{SS\_NUMCOL} - 1] \Rightarrow \texttt{ss\_col}$
$|\ l \notin [0..\texttt{SS\_NUMCOL} - col] \Rightarrow \texttt{ss\_len})$

`ss_s_cur`(row, col):
transition: $curbuf := \langle row, col\rangle$
exceptions: $exc :=$
$(row \notin [0..\texttt{SS\_NUMROW} - 1] \Rightarrow \texttt{ss\_row}$
$|\ col \notin [0..\texttt{SS\_NUMCOL} - 1] \Rightarrow \texttt{ss\_col})$

`ss_s_ref`:
transition: $scn, hlt, cur := scnbuf, hltbuf, curbuf$
exceptions: none

`ss_s_end`:
transition: none
exceptions: none

considerations
`ss_s_str` and `ss_s_hlt` may alter the value of *curbuf*.

C.3.4.3 Header file: scnstr.h

```
/*****constants*****/

#define SS_NUMROW 24 /*number of rows on the screen*/
#define SS_NUMCOL 80 /*number of columns on the screen*/

/*****types*****/

/*****access routines*****/

void ss_s_init();

void ss_s_clrscn();

void ss_s_str();
/*   void ss_s_str(r,c,s)
*    int r,c;
*    char *s;
*/

void ss_s_hlt();
/*   void ss_s_hlt(r,c,l,f)
*    int r,c,l;
*    (boolean) int f;
*/

void ss_s_cur();
/*   void ss_s_cur(r,c)
*    int r,c;
*/

void ss_s_ref();

void ss_s_end();

/*****exception handlers*****/

void ss_row();

void ss_col();

void ss_len();
```

C.4 Demonstration Modules

C.4.1 *stack* MIS

C.4.1.1 Interface syntax

```
#define PS_MAXSIZ 100
```

Routine names	Inputs	Outputs	Exceptions
`ps_s_init`			
`ps_s_push`	`int`		`ps_full`
`ps_s_pop`			`ps_empty`
`ps_g_top`		`int`	`ps_empty`
`ps_g_depth`		`int`	

C.4.1.2 Interface semantics

state variables
s : *sequence of integer*

state invariant
$|s| \leq$ `PS_MAXSIZ`

assumptions
`ps_s_init` is called before any other access routine.

access routine semantics
`ps_s_init`:
  transition: $s := \langle\rangle$
  exceptions: none
`ps_s_push`(x):
  transition: $s := s \mathbin{||} \langle x \rangle$
  exceptions: $exc := (|s| = \texttt{PS\_MAXSIZ} \Rightarrow \texttt{ps\_full})$
`ps_s_pop`:
  transition: $s := s[0..|s|-2]$
  exceptions: $exc := (|s| = 0 \Rightarrow \texttt{ps\_empty})$
`ps_g_top`:
  output: $out := s[|s|-1]$
  exceptions: $exc := (|s| = 0 \Rightarrow \texttt{ps\_empty})$
`ps_g_depth`:
  output: $out := |s|$
  exceptions: none

C.4.1.3 Header file: `stack.h`

```
/*****constants*****/

#define PS_MAXSIZ 100 /*the maximum stack size*/
```

```
/*****types*****/

/*****access routines*****/

void ps_s_init();

void ps_s_push();
/*   void ps_s_push(i);
*    int i;
*/

void ps_s_pop();

int ps_g_top();

int ps_g_depth();

void ps_g_dump(); /*for testing purposes only*/

/*****exception handlers*****/

void ps_empty();

void ps_full();
```

C.4.2 *symtbl* MIS

C.4.2.1 Interface syntax

```
#define ST_MAXSYMS 50
#define ST_MAXSYMLEN 20
```

Routine names	Inputs	Outputs	Exceptions
st_s_init			
st_s_add	char* int		st_maxlen st_exsym st_full
st_g_exsym	char*	*boolean*	
st_s_loc	char* int		st_notexsym
st_g_loc	char*	int	st_notexsym
st_g_siz		int	

C.4.2.2 Interface semantics

state variables
$tbl : set\ of\ tuple\ of\ (sym : string, loc : integer)$

state invariant
1. $|tbl| \leq$ `ST_MAXSYMS`
2. $(\forall t \in tbl)(|t.sym| \leq$ `ST_MAXSYMLEN`$)$
3. $(\forall t_1, t_2 \in tbl)(t_1 \neq t_2 \rightarrow t_1.sym \neq t_2.sym)$

assumptions
`st_s_init` is called before any other access routine.
All string parameters are legal C strings.

access routine semantics
`st_s_init`:
 transition: $tbl := \{\}$
 exceptions: none
`st_s_add`(sym, loc):
 transition: $tbl := tbl \cup \{\langle sym, loc\rangle\}$
 exceptions: $exc := (|sym| >$ `ST_MAXSYMLEN` $\Rightarrow$ `st_maxlen`
 $\mid (\exists loc_1)(\langle sym, loc_1\rangle \in tbl) \Rightarrow$ `st_exsym`
 $\mid |tbl| =$ `ST_MAXSYMS` $\Rightarrow$ `st_full`$)$
`st_g_exsym`(sym):
 output: $out := (\exists loc)(\langle sym, loc\rangle \in tbl)$
 exceptions: none
`st_s_loc`(sym, loc):
 transition: $tbl := (tbl - \{\langle sym, loc_1\rangle\}) \cup \{\langle sym, loc\rangle\}$ where $\langle sym, loc_1\rangle \in tbl$
 exceptions: $exc := (\neg(\exists loc_1)(\langle sym, loc_1\rangle \in tbl) \Rightarrow$ `st_notexsym`$)$
`st_g_loc`(sym):
 output: $out := loc$, where $\langle sym, loc\rangle \in tbl$
 exceptions: $exc := (\neg(\exists loc)(\langle sym, loc\rangle \in tbl) \Rightarrow$ `st_notexsym`$)$
`st_g_siz`:
 output: $out := |tbl|$
 exceptions: none

C.4.2.3 Header file: `symtbl.h`

```
/*****constants*****/

#define ST_MAXSYMS 50 /*maximum number of symbols*/
#define ST_MAXSYMLEN 20 /*maximum symbol length*/

/*****types*****/

/*****access routines*****/

void st_s_init();

void st_s_add();
/*   void st_s_add(sym,loc);
*    char *sym;
*    int loc;
```

```
*/

int st_g_siz();

/*boolean*/ int st_g_exsym();
/*   int st_g_exsym(sym);
*    char *sym;
*/

void st_s_loc();
/*   void st_g_loc(sym,loc);
*    char *sym;
*    int loc;
*/

int st_g_loc();
/*   int st_g_loc(sym);
*    char *sym;
*/

void st_g_dump(); /*for testing purposes only*/

/*****exception handlers*****/

void st_exsym();

void st_maxlen();

void st_notexsym();

void st_full();
```