Assignment 5

May 14, 1996

1 Due Date

May 21, 1996. Please turn in your solution at the beginning of the lecture.

Start early!!!!!!!!

2 Exceptions and Handlers

I have extended our language with exceptions and handlers.

2.1 Syntax

The concrete syntax of the new constructs is specified using the following clauses. Keywords are in boldface, the meta-variable E refers to the name of an exception, the meta-variable H refers to a procedure (the handler), and the meta-variables $M, M_1, M_2, \ldots$ refer to arbitrary expressions:

newexception E **in** M
raise E M
tryhandle E H **in** M
tryhandlefinally E H **in** M_1 **finally** M_2
resume M

I have already extended the parser to accept the new forms. The abstract syntax trees are:

```
(define-record newexception (name body))
(define-record raise (name exp))
(define-record tryhandle (name handler body))
(define-record tryhandlefinally (name handler body finally))
(define-record resume (exp))
```

The new parser is available from: `/cs/classes/cis624/assignments/ch9/`

2.2 (Informal) Semantics: Termination Model

The following informal semantics is consistent with the "termination model" of Chapter 16 of "The Design and Evaluation of C++" handed out in class. The expression (**resume** M) cannot be used in this context.

The expression (**newexception** E **in** M) introduces a new exception name E that is lexically bound in the expression M. In other words any expression that raises or handles E must occur (syntactically) within M. The initial handler for the exception E simply prints an error message and exits.

The expression (**raise** E M) evaluates the argument M, locates the current handler for exception E, and then calls the handler with the value of M as an argument.

The expression (**tryhandle** E H **in** M) first saves the old handler, and uses H to construct a new handler. It then attempts to evaluate expression M. If the evaluation terminates normally then the value of M is returned and the new handler is discarded. Otherwise if the evaluation of M raises exception E with argument A then the current handler is applied to the argument A.

The last kind of expression (**tryhandlefinally** E H **in** M_1 **finally** M_2) behaves like **tryhandle** except that the expression M_2 must be evaluated whether M_1 raises exceptions, jumps out, or terminates normally. In any case, the final value of the entire expression is the value returned by M_1.
Here is a small example:

```
newexception xxx in
  let a = 3
  in let f = proc (x,y) if equal(x,y)
                        then (raise xxx (proc () a))
                        else (proc () +(+(x,y),a))
     in let v=tryhandle xxx (proc (g) g()) in f(3,3);
            w=(tryhandlefinally xxx (proc (g) g) in f(1,4) finally (a:=99))();
            z=(tryhandlefinally xxx (proc (g) g) in f(5,5) finally (a:=999))()
        in +(+(v,w),z);
==> 1106
```

Note: If you think that this informal semantics is complicated, ambiguous, incomplete, etc, then you have just discovered that the semantics of programming constructs should never be communicated in English but rather using some formal notation, *e.g.* an interpreter like the one you will be writing!

If the informal semantics does not specify something in enough detail, then do someting reasonable. Document your design choices!

2.3 (Informal) Semantics: Resumption Model

The following informal semantics is consistent with the "resumption model" of Chapter 16 of "The Design and Evaluation of C++" handed out in class.

All the old expressions (everything but **resume**) have essentially the same semantics. The additional expression (**resume** M) only makes sense from within a handler H. If the handler H was raised from a program point p, then the effect of (**resume** M) is to jump back to point p with the value of M.
Here is another small example:

```
newexception xxx in
  let f = proc (x) if equal(x,0) then (raise xxx 42) else +(x,x)
  in tryhandle xxx (proc (x) (if equal(x,42) then (resume 0) else x))
     in f(0);
==> 0
```

3 Assignment

You need to complete the following tasks:

1. Write two interpreters: one in CPS and one without CPS that implement the termination model. (Ignore **resume** in this part.)

2. Modify the interpreters to implement the resumption model. (Ignore **tryhandlefinally** in this part.)

4 Puzzles

- Do you buy the arguments in the "The Design and Evolution of C++" about resumption semantics being too expensive and not very useful? Based on your interpreter explain the cost incurred by an implementation.

- Extend the interpreter with your favorite `goto`-like facility (`call/cc` would do). What goes wrong? Fix it! **Hint:** Read Section 7.4.1 from the chapter on Exceptions in the Java book. Think about the `finally` clause.