CS 3721: Programming Languages Lab
Lab #14: Continuation Passing, PHP Programming, and HTML forms.

Continuation passing is a programming style which packs up the rest of the computation after making a function call as an extra parameter to the callee, so that the call does not need to return to the caller. For example, given the following ML code.

```
open TextIO;
fun chainInput() =
   let val x = input(stdIn)
   in if (x = "END\n") then []
      else x :: chainInput()
   end;
chainInput();
```

The above code keeping reading user input until reaching the string "END
n" and then returns a list which contains all the input lines. It can be converted to continuation passing in the following.

```
open TextIO;
fun chainInput(K) =
   let val x = input(stdIn)
   in if (x = "END\n") then K([])
      else chainInput(fn y => K(x :: y))
   end;
chainInput(fn x=>x);
```

PHP is a web programming language which can be used to read input via html forms and then dynamically generate html pages which may read additional input. If using web programming via PHP and HTML forms, the above program becomes

```
<?php
  class EmptyList { public function toString() { return "NULL"; } };
  class myList {
    private $x; private $next;
    public function myList($val_x, $val_next)
        { $this->x = $val_x; $this->next=$val_next; }
    public function toString()
        { return $this->x."::".$this->next->toString();}
  };

  class Identity
  { public function invoke($y) { return $y; }
    public function toString() { return strval(0);}
  };
  class myContinuation
  {
     private $x = "";
     private $K=NULL;
     public function invoke($y)
           {  return $this->K->invoke(new myList($this->x, $y)); }
     public function myContinuation($val_x, $val_K)
```

```
        { $this->x = $val_x; $this->K=$val_K; }
    public function toString() {
        if ($this->x=="") return $this->K->toString();
        else return strval(strlen($this->x)).$this->x.$this->K->toString(); }
    public static function fromString($k)
      {
        $i=0;
        for ($j=0; ctype_digit($k[$j]); $j++) {
           $i=$i*10 + (int)$k[$j];
        }
        if ($i==0 || $k == "") return new Identity();
        if ($i > 0) {
            $x=substr($k,$j,$i);
            $k=myContinuation::fromString(substr($k,$i+1,strlen($k)-$i));
            return new myContinuation($x,$k);
        }
      }
  };

   $x = $_POST['user'];
   $K = myContinuation::fromString($_POST['continuation']);
   if ($x == "END")  echo $K->invoke(new EmptyList())->toString();
   else  {
      $K = new myContinuation($x,$K);
      $K_string = $K->toString();
      echo "
      <form action='lab14.php' method='post'>
        <input type='hidden' name='continuation' value='$K_string' />
        Enter an input: <input type='text' size=40 name=\"user\"/>
        <input type='submit' name='submit'/>
      </xform> ";
   }
?>
```

Note that both the ML list and the continuation representation are translated to object-oriented classes in PHP. Further, since the invocation of the PHP script goes through the HTML form, the generation of the form has to be the last component of the PHP script (the tail recursion part). Finally, because the elements of the HTML form can only take strings as values, the continuation (passed through the *hidden* element) has been converted to and from string repetitively.

Save the ML script and the PHP script into separate files. Run the ML script inside the SML environment, and run the PHP script using a standard browser. Modify both the ML and the PHP script so that instead of simply outputing the strings you entered, they duplicate each string in the output list.