CSCI 3155: Principles of Programming Languages
Exercise sheet #5 (v2)
11th June 2007

Group name:_____

Scoping and Parameter Passing

Exercise 1. Ada 83 allows language implementers to choose between the two possible ways of implementing **in out** parameters.

(a) (**Skill 6.1**) What are the two possible ways of implementing **in out** parameters?

(b) (**Skill 6.1**) Explain how the two can lead to different results. You may use an example.

(c) (Sebesta, Problem Set 9.3, **Skill 6.1**) Argue in support of the Ada 83 designers' decision.

Exercise 2. Many languages used today, such as Java or Haskell, do not provide **out** mode parameter passing. Other languages used today, such as C# or Ada, do provide such a mechanism.

(a) (**Skill 6.1**) What are the two possible ways of implementing **out** mode parameter passing?

(b) (**Skill 6.1**) Give a practical example where **out** mode parameters are useful in a language that already supports regular return values.

(c) (**Skillset 1**) Consider adding **out** mode parameters to a language that doesn't already have any **out** or **in out** mode parameters. Assume that the language does not support tuples or object-oriented features. Choose a position on whether or not to add the feature and justify it, using our criteria and characteristics.

Exercise 3. Sebesta describes two scoping techniques: dynamic scoping and static scoping.

(a) The following MYSTERY program is statically scoped. Mark its static scopes as on page 229 in Sebesta:

```
VAR i : INTEGER;
PROCEDURE p(j : INTEGER) : INTEGER =
BEGIN
    j := j + j;
    PROCEDURE q(k : INTEGER) : INTEGER =
    BEGIN
        IF 16 > k
        THEN i := p(k);
             RETURN k
        ELSE PRINT k;
             RETURN 0
        END
    END
    BEGIN
        RETURN q(j);
    END
END
BEGIN
    i := 0;
    p (i + 1)
END
```

For this task, assume that each ⟨*DeclList*⟩ provides all of the names it defines to its associated ⟨*Block*⟩, as well as to all of the inner scopes of its ⟨*Decl*⟩ members. For procedures, the names defined by such a ⟨*DeclList*⟩ are only the procedure names, but not parameter names.

(b) List all the variables and procedures visible in each of the scopes you marked.

(c) (**Skill 5.1**) Write a MYSTERY program that illustrates the *advantages* of *dynamic* scoping. Explain.

(d) (**Skill 5.1**) Re-write your program to use static scoping. Explain how you converted your program.

(e) (**Skill 6.1**) Did you assume any particular parameter passing mode or modes? If so, which mode(s) and why?

Exercise 4. The C programming language comes with a specialised sublanguage, called the *C preprocessor* (or just `cpp` for sort). `cpp` programs are C programs extended with a small number of constructs. When compiling `cpp` programs, first `cpp` is invoked, then the C compiler.

`cpp` provides the following basic constructs (among others):

```
#define V n
```

Define the name `V` as `n`

```
#define P(X) f(X)
```

Define the subprogram `P` with formal parameter `X` to mean `f(X)` (where the `X` refers to the formal parameter).

```
#undef V
```

"Un-defines" the definition associated with the name `V`; required before re-defining a previously defined name.

Consider the following `cpp` program:

```
int f(int x)
{
        printf("f\n");
        return x;
}

#define V f(2)
#define P(X) printf("%d, %d, %d\n", X, X, V)

#undef V
#define V f(3)

int main(int argc, char **argv)
{
        P(V);
}
```

The program, when compiled and executed, prints the following output:

```
f
f
f
3, 3, 3
```

(a) (**Skills 6.1, 6.3 (applied to a non-Mystery language)**) What parameter passing mode does `cpp` use? Explain.

(b) (**Skills 5.1, 5.3 (applied to a non-Mystery language)**) What scoping mechanism does `cpp` use? Explain.