Principles of Programming Languages
Small examination

Student ID: Name:

Problem 1 Show the type consistency of the program fragment according to (1) and (2). Note that the program fragment is written in the subset of C language shown in the lecture.

```
int *p;
int x[3];
p = x;
```

(1) Rewrite the variable declarations `int *p;` and `int x[3];` in the postfix notation shown in the lecture.

(2) Show the type consistency of the assignment expression `p=x` by applying the inference rules to the declarations of `p` and `x` in the postfix notation obtained in (1).

Typing rules

- Rules for function calls, pointers, arrays

$$\frac{e : \tau[n]}{e[i] : \tau} \qquad \frac{e : \tau()}{e() : \tau} \qquad \frac{e : \tau*}{*\, e : \tau} \qquad \frac{e : \tau[n]}{e : \tau\&}$$

- Rule for assignment operator $=$, where e is an l-value expresssion and not a constant.

$$\frac{e : \tau \quad e' : \tau}{e = e' : \tau}$$

- Rule for the & operator where the outermost part of τ is not &.

$$\frac{e : \tau}{\&e : \tau\&} \qquad \frac{e : \tau\&}{*\, e : \tau} \qquad \frac{e : \tau * \quad e' : \tau\&}{e = e' : \tau\&}$$

Problem 2 A lambda expression $(\lambda x.\ \lambda y.\ x)\ ((\lambda z.\ z)\ w)$ can be transformed to $(\lambda y.\ w)$ by applying the β reductions. Write the each step of the β reductions. (Although there are more than one sequences of β reductions, write one of them.)

Rules for lambda calculus

- β reductions

$$(\lambda x.M)\ N \xrightarrow[\beta]{} M[N/x]$$

$$\frac{M \xrightarrow[\beta]{} N}{\lambda x.M \xrightarrow[\beta]{} \lambda x.N} \qquad \frac{M \xrightarrow[\beta]{} N}{MP \xrightarrow[\beta]{} NP} \qquad \frac{M \xrightarrow[\beta]{} N}{PM \xrightarrow[\beta]{} PN}$$

- Substitutions

$$\begin{array}{rcl}
c[N/x] & = & c \\
x[N/x] & = & N \\
x[N/y] & = & y \quad (x \neq y) \\
(\lambda y.M)[N/x] & = & \begin{cases} \lambda y.M & \textbf{if } x = y \\ \lambda y.(M[N/x]) & \textbf{if } x \neq y,\ y \notin FV(N) \\ \lambda z.((M[z/y])[N/x]) & \textbf{if } x \neq y,\ z \neq x,\ y \in FV(N), \\ & \quad z \notin FV(M),\ z \notin FV(N) \end{cases} \\
(M_1 M_2)[N/x] & = & (M_1[N/x])(M_2[N/x])
\end{array}$$

- Free variables

$$\begin{array}{rcl}
FV(c) & = & \{\} \\
FV(x) & = & \{x\} \\
FV(\lambda x.M) & = & FV(M) \setminus \{x\} \\
FV(M_1 M_2) & = & FV(M_1) \cup FV(M_2)
\end{array}$$

Problem 3 Write the output to the display when executing the following program in C++.

```
#include <stdio.h>

class B {
public:
  virtual char f() { return 'B';}
  char g() { return 'B'; }
  char testF(B *b) { return b->f();}
  char testG(B *b) { return b->g();}
};

class D : public B {
public:
  char f() { return 'D';}
  char g() { return 'D';}
};

int main(void) {
  D *d = new D;
  printf("%c%c\n", d->testF(d), d->testG(d));
  return 0;
}
```

Problem 4 Write the solution (the substitution to the variable `X`) to the query `a(X).` after defining `a`, `b`, `c`, `d`, and `e` in Prolog as follows.

```
a(1) :- b.
a(2) :- e.
b :- !, c.
b :- d.
c :- fail.
d.
e.
```

Problem 5 Write the result of evaluating `length [1,2,3]` after defining `length` in Standard ML.

```
fun length nil = 0
  | length (x::xs) = 1 + length xs;
```