Software Engineering 1

Herkansing OP4 - 2017

1 Question 1 (4 pts.)

Grading: *Full points for all correct steps and result. -1 for each wrong step. Zero points with 4 or more errors.*

Given the following lambda program, complete the empty beta reduction steps for this program.

```
((((λf g→(f (g x))) (λx y→y)) (λx→x)) A)
```

1.1 Answer 1

```
((((λf g→(f (g x))) (λx y→y)) (λx→x)) A)
```

```
(( ((λf g→(f (g x))) (λx y→y)) (λx→x)) A)
```

```
(((λg→( (λx y→y) (g x))) (λx→x)) A)
```

```
( ((λg→((λx y→y) (g x))) (λx→x)) A)
```

```
(((λx y→y) ( (λx→x) x)) A)
```

```
(((λx y→y) ((λx→x) x)) A)
```

```
(((λx y→y) x) A)
```

```
( ((λx y→y) x) A)
```

```
((λy→y) A)
```

```
((λy→y) A)
```

```
A
```

2 Question 2 (4 pts.)

Grading: *Full points for all correct steps and result. -1 for each wrong step. Zero points with 4 or more errors.*

Given the following lambda calculus program complete the typing derivation for the program.

```
(λ(f:(Tuple→List)) (g:(List→List→List)) (t:Tuple) (l:List)→((g (f t)) l))
```

2.1 Answer 2

```
(λ(f:(Tuple→List)) (g:(List→List→List)) (t:Tuple) (l:List)→((g (f t)) l))
```

```
(λ(f:(Tuple→List)) (g:(List→List→List)) (t:Tuple) (l:List)→((g (f t)) l))
```

```
(λ(f:(Tuple→List)) (g:(List→List→List)) (t:Tuple) (l:List)→((g ( (Tuple→List) t)) l))
```

```
(λ(f:(Tuple→List)) (g:(List→List→List)) (t:Tuple) (l:List)→((g ((Tuple→List) t)) l))
```

```
(λ(f:(Tuple→List)) (g:(List→List→List)) (t:Tuple) (l:List)→(( (List→List→List) ((Tuple→List)
    t)) l))
```

```
(λ(f:(Tuple→List)) (g:(List→List→List)) (t:Tuple) (l:List)→
    (((List→List→List) ((Tuple→List) t)) l))
```

```
(λ(f:(Tuple→List)) (g:(List→List→List)) (t:Tuple) (l:List)→(((List→List→List) ((Tuple→List)
     Tuple)) l))
```

```
(λ(f:(Tuple→List)) (g:(List→List→List)) (t:Tuple) (l:List)→
    (((List→List→List) ((Tuple→List) Tuple)) l))
```

```
(λ(f:(Tuple→List)) (g:(List→List→List)) (t:Tuple) (l:List)→(((List→List→List) ((Tuple→List)
    Tuple)) List))
```

```
(λ(f:(Tuple→List)) (g:(List→List→List)) (t:Tuple) (l:List)→(((List→List→List)
    ((Tuple→List) Tuple)) List))
```

```
(λ(f:(Tuple→List)) (g:(List→List→List)) (t:Tuple) (l:List)→(((List→List→List) List) List))
```

```
(λ(f:(Tuple→List)) (g:(List→List→List)) (t:Tuple) (l:List)→( ((List→List→List) List) List))
```

```
(λ(f:(Tuple→List)) (g:(List→List→List)) (t:Tuple) (l:List)→( (List→List) List))
```

```
(λ(f:(Tuple→List)) (g:(List→List→List)) (t:Tuple) (l:List)→ ((List→List) List) )
```

```
(λ(f:(Tuple→List)) (g:(List→List→List)) (t:Tuple) (l:List)→ List )
```

```
(λ(f:(Tuple→List)) (g:(List→List→List)) (t:Tuple) (l:List)→ List )
```

```
(λ(f:(Tuple→List)) (g:(List→List→List)) (t:Tuple)→ (List→List) )
```

```
(λ(f:(Tuple→List)) (g:(List→List→List)) (t:Tuple) → (List→List) )
```

```
(λ(f:(Tuple→List)) (g:(List→List→List))→ (Tuple→List→List) )
```

```
(λ(f:(Tuple→List)) (g:(List→List→List)) → (Tuple→List→List) )
```

```
(λ(f:(Tuple→List))→ ((List→List→List)→Tuple→List→List) )
```

```
(λ(f:(Tuple→List))→((List→List→List)→Tuple→List→List))
```

```
((Tuple→List)→(List→List→List)→Tuple→List→List)
```