Homework Assignment 2

1. For each of the following lambda expressions, write its normal form or explain why it does not have one.
   (a) \( + \ 1 \ 2 \)
   (b) \( \lambda x. + x \ 3 \)
   (c) \( (\lambda x. + x \ 3) \ 5 \)
   (d) \( (\lambda x. \lambda y. * ( + x \ 2) \ y) \ 2 \)
   (e) \( (\lambda x. \lambda y. * ( + x \ 2) \ y) \ 2 \ 3 \)
   (f) \( (\lambda x. x \ x)(\lambda x. x \ x) \)

2. Consider the following C-like program.

   ```c
   int w = 3;
   int x = 10;

   int incw() { return ++w; }
   int incx() { return ++x; }

   void foo(y, z) {
     printf("%d\n", y + y);
     x = 1;
     printf("%d\n", z);
   }

   int main() {
     foo(incw(), incx());
     return 0;
   }
   
   What does it print if the language uses
   (a) Applicative-order evaluation?
   (b) Normal-order evaluation?
   
3. In an assembly-language-like notation (e.g., use MIPS or a pseudocode of your own choosing), write what a good optimizing compiler would produce for the following two switch statements. Assume they are in separate functions (i.e., compiled separately).

   ```c
   switch (a) {
     case 1: x = 3; break;
     case 2: x += 5; break;
     case 3: y = 15; break;
     case 4: z = 20; break;
     case 5: x = z + 23; break;
     default: x = 28; break;
   }

   switch (b) {
     case 1: p = 3; break;
     case 10: p = 5; break;
     case 100: q += 15; break;
     case 1000: r = q + 20; break;
     default: r = 25; break;
   }
   ```

4. For a 32-bit little-endian processor with the usual alignment rules, show the memory layout and size in bytes of the following C types.

   ```c
   union {
     struct {
       char b; /* 8-bit */
       int a; /* 32-bit */
     } s;
     short c; /* 16-bit */
   } ul1;

   struct {
     char a;
     short b;
     int c;
     char d;
   } s1;

   struct {
     char a;
     char d;
     short b;
     int c;
   } s2;
   ```