CMPSC 461: Programming Language Concepts

Assignment 5. Due: Nov. 4, NOON (no late submission)

For this assignment, you need to submit your solution as a single PDF file to Canvas.

Problem 1 [6pt] Consider the following C declaration:

```
struct S1 {int a; double b;};
struct S2 {char a; float b;};
union U {
    struct S1 s1;
    struct S2 s2;
} u;
```

Assume the machine has 1-byte characters, 4-byte integers, 8-byte floating numbers, and 16-byte double-precision floating numbers. Assume the compiler does not reorder the fields, and it leaves no holes in the memory layout. How many bytes does `u` occupy? If the memory address of `u` starts from 1000, what are the start addresses of `u.s1.b` and `u.s2.b`?

Problem 2 [12pt] Consider the simply typed λ-calculus and its typing rules defined in Note 3. Suppose we want to add a new operation '$\leq$' which performs the "less or equal than" comparison on two numbers and returns either `true` or `false`. We can define the syntax of this new language as follows:

$$\text{terms} \qquad e ::= \cdots \mid e_1 \leq e_2$$

where the dots represents the terms defined in note 3. The syntax of types remains unchanged.

a) (4pt) Follow the notations in Note 3, write down a typing rule (call it T-Leq) for the new term $e_1 \leq e_2$, so that the comparison takes two integers and returns a Boolean.

b) (8pt) What is the type of the term $((\lambda x : \texttt{int}.\ (x \leq 1))\ 2)$? Justify your answer by writing down the proof tree for this term.

Problem 3 [6pt] Suppose that during type inference, a compiler generates the following constraints to be solved:

$$\alpha \to \beta = \texttt{bool} \to \alpha; \beta = \gamma$$

Solve those constraints by the unification algorithm in Note 4. You need to write down the steps during constraint solving.

Problem 4 [10pt] Consider the following implementation in the C language:

```
int g(int x) {
    if (x == 1) {
        return 1;
    }
    return x*g(x-1);
}
```

a) (6pt) Write down a tail recursive implementation of function `g`. You can use helper function in your solution.

b) (4pt) An "optimizing" compiler will often be able to generate efficient code for recursive functions when they are tail-recursive. Refer to activation record, briefly explain how a compiler may "reuse" the same activation record for your solution in a).

Problem 5 [6pt] Consider the following (erroneous) program in the C language:

```
void foo( ) {
    int i;
    printf("%d ", i++);
}

int main( ) {
    int j;
    for (j=1; j<=10; j++)
        foo();
}
```

Local variable `i` in `foo` is never initialized. On many systems, however, the program will display repeatable behavior: printing 0 1 2 3 4 5 6 7 8 9. Suggest an explanation.

Problem 6 [10pt] Insert one or two statements in function A and insert arguments into the call to A such that the behavior of the program (values printed out) will differ depending upon whether parameters are passed by value, by reference, or by value return. Explain what will be printed out for each parameter passing mode based on your solution.

```
void A(int m, int n) {


}
main () {
  int a=0, b=0;
  A( , );
  print a,b;
}
```