CMPSC 461: Programming Language Concepts
Assignment 5 Solution

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`?

Solution:
`u` occupies max(4+16,1+8)=20 bytes. The start address of `u.s1.b` is 1004; the start address of `u.s2.b` is 1001.

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.

Solution:

$$\frac{\Gamma \vdash e_1 : \texttt{int} \quad \Gamma \vdash e_2 : \texttt{int}}{\Gamma \vdash (e_1 \leq e_2) : \texttt{bool}} \text{ (T-LEQ)}$$

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.

Solution:

$$\dfrac{\dfrac{\dfrac{x : \texttt{int} \vdash x : \texttt{int} \qquad x : \texttt{int} \vdash 1 : \texttt{int}}{x : \texttt{int} \vdash (x \leq 1) : \texttt{bool}}}{\vdash (\lambda x : \texttt{int}\ .\ (x \leq 1)) : \texttt{int} \to \texttt{bool}} \qquad \vdash 2 : \texttt{int}}{\vdash (\lambda x : \texttt{int}\ .\ (x \leq 1))\ 2 : \texttt{bool}}$$

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.

Solution:

$$\begin{aligned}
\texttt{Unify}(\alpha \to \beta = \texttt{bool} \to \alpha; \beta = \gamma) &= \texttt{Unify}(\alpha = \texttt{bool}; \beta = \alpha; \beta = \gamma) \\
&= [\alpha \leftarrow \texttt{bool}] \circ \texttt{Unify}((\beta = \alpha; \beta = \gamma)[\alpha \leftarrow \texttt{bool}]) \\
&= [\alpha \leftarrow \texttt{bool}] \circ \texttt{Unify}(\beta = \texttt{bool}; \beta = \gamma) \\
&= [\alpha \leftarrow \texttt{bool}] \circ [\beta \leftarrow \texttt{bool}] \circ \texttt{Unify}((\beta = \gamma)[\beta \leftarrow \texttt{bool}]) \\
&= [\alpha \leftarrow \texttt{bool}] \circ [\beta \leftarrow \texttt{bool}] \circ \texttt{Unify}(\texttt{bool} = \gamma) \\
&= [\alpha \leftarrow \texttt{bool}] \circ [\beta \leftarrow \texttt{bool}] \circ [\gamma \leftarrow \texttt{bool}] \circ \texttt{Unify}(\emptyset) \\
&= [\alpha \leftarrow \texttt{bool}, \beta \leftarrow \texttt{bool}, \gamma \leftarrow \texttt{bool}]
\end{aligned}$$

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.

Solution:

```
int g(int x) {
    return h(x, 1);
}

int h(int x, int y) {
    if (x == 1) {
        return y;
    }
    return h(x-1,x*y);
}
```

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).

Solution: Before the recursive function call, the compiler can reuse the memory space that stores the caller's parameters for the parameters passed to the callee. Then, the control flow can be switched to the callee without the calling sequences.

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.

Solution: The activation records for different instances of foo will occupy the same space in the stack. Hence, although the local variable i is never initialized, it might "inherit" a value from the previous instance of the routine. If the stack is initialized to zeros by the operating system, then i will start with the value zero in the first iteration, and increase by one in each iteration after that.

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;
}
```

Solution:Here is one possible solution:

```
void A(int m, int n) {
  m = m+1;
  n = n+1;
}
main () {
  int a=0, b=0;
  A(a,a);
  print a,b;
}
```

This program prints 0,0 (call by value), 2,0 (call by reference) and 1,0 (call by value return).