CMPSC 461: Programming Language Concepts
Assignment 6 Solution

Problem 1 [6pt] Assume a 32-bit binary 01000011 01010000 00000000 00000000. Write down what value does the binary represent if its type is (1) 32-bit unsigned integer, (2) four characters in ASCII code, and (3) 32-bit floating point number.

Solution:

32 bit unsigned integer:	1129316352
Char:	CPNulNul
32 bit float IEEE 754:	$1.625 * 2^7$ (sign:0, exp:134-127=7 m: 1+1/2+1/8=1.625)

Problem 2 [10pt] Consider the following C declaration:

```
union U1 {float romo; double posey;};
union U2 {char cain; int posey;};
struct S {
    union U1 sf;
    union U2 giants;
} s;
```

Assume that the machine has 1-byte characters, 4-byte integers, 8-byte single-precision floating numbers, and 16-byte double-precision floating numbers. Assume that the compiler does not reorder the fields, and it leaves no holes in the memory layout.

a) (4pt) How many bytes does `s` occupy?

Solution: `s` occupies max(8,16)+max(1,4)=20 bytes.

b) (6pt) If the memory address of `s` starts from 2021, what are the start addresses of `s.sf.posey` and `s.giants.posey`?

Solution: The start address of `s.u1.posey` is 2021; the start address of `s.u2.posey` is 2021+max(8,16)=2037.

Problem 3 [8pt] Assume that you will design a programming language with Dynamic and Flexible Arrays: arrays whose dimensions are not known until storage is allocated for an array, and the size can change (grow/shrink) after that. Briefly explain among static data area, stack and heap, which memory regions are proper for such arrays. Moreover, briefly describe how bounds checking would be supported. You should consider both one- and multi-dimensional arrays.

Solution: (This is an open question. We only show one possible solution)

The flexible dynamic array needs heap allocation, because the size needs to change at run time. Since the array size might change, a continuous space for all rows and columns is less efficient. For single dimensional array, the complier can request one or multiple blocks of memory on the heap. These memory blocks can be managed as a linked-list, so that array size can change at run time. A similar strategy can be used for multi-dimensional arrays.

Array bounds can be stored in the dope vector. Unlike fixed arrays, the bounds need to be updated whenever the array dimensions change.

Problem 4 [12pt] Consider the simply typed λ-calculus and its typing rules defined in Note 3. Suppose we want to add a new operation '$\geq$' which returns true when the first operand is greater than or equal to the second operand. We can define the syntax of this new language as follows:

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

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

a) (5pt) Follow the notations in Note 3, write down a typing rule (call it T-Geq) for the new term $e_1 \geq e_2$, so that the equality test takes two integers and returns a Boolean value.

Solution:

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

b) (7pt) What is the type of the term $\big((\lambda x{:}\texttt{int}.\ (x \geq (9+4)))\ 15\big)$? Justify your answer by writing down the proof tree for this term.

Solution: The type of the term is `bool`. Here is the proof tree:

$$\dfrac{\dfrac{\dfrac{x : \texttt{int} \vdash x : \texttt{int} \qquad \dfrac{x : \texttt{int} \vdash 9 : \texttt{int} \quad x : \texttt{int} \vdash 4 : \texttt{int}}{x : \texttt{int} \vdash 9 + 4 : \texttt{int}}}{x : \texttt{int} \vdash (x \geq (9+4)) : \texttt{bool}}}{\vdash (\lambda x : \texttt{int}\ .\ (x \geq (9+4))) : \texttt{int} \to \texttt{bool}} \qquad \vdash 15 : \texttt{int}}{\vdash (\lambda x : \texttt{int}\ .\ (x \geq (9+4)))\ 15 : \texttt{bool}}$$

Problem 5 [14pt] Consider the simply typed λ-calculus and its typing rules defined in Note 4 (with type inference).

a) (7pt) What constraints are generated for the term $(\lambda x.\ (\lambda y.\ (x \wedge y)))$ if we use the type system defined in Note 4? Justify your answer by writing down the proof tree for this term.

Solution: The constraints generated are $\alpha_x = \texttt{bool}$ and $\alpha_y = \texttt{bool}$. Here is the proof tree:

$$\dfrac{\dfrac{\dfrac{x : \alpha_x, y : \alpha_y \vdash x : \alpha_x \mid \emptyset \qquad x : \alpha_x, y : \alpha_y \vdash y : \alpha_y \mid \emptyset}{x : \alpha_x, y : \alpha_y \vdash (x \wedge y) : \texttt{bool} \mid \alpha_x = \texttt{bool}; \alpha_y = \texttt{bool}}}{x : \alpha_x \vdash (\lambda y.\ (x \wedge y)) : \alpha_y \to \texttt{bool} \mid \alpha_x = \texttt{bool}; \alpha_y = \texttt{bool}}}{\vdash (\lambda x.\ (\lambda y.\ (x \wedge y))) : \alpha_x \to \alpha_y \to \texttt{bool} \mid \alpha_x = \texttt{bool}; \alpha_y = \texttt{bool}}$$

b) (7pt) Suppose that during type inference, a compiler generates the following constraints to be solved:

$$\alpha \to \beta = \texttt{int} \to \alpha; \beta = \alpha$$

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

Solution:

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