Solutions by Prof. Allan Gottlieb, edited by Ang Sun

1. (20 points). Fill in the blank (2 points each).

(i) The theoretical foundation for functional languages is called the ____________

(ii) The time at which variables are assigned to specific memory locations is called the ____________

(iii) Variables with unrestricted lifetimes are allocated from the ____________

(iv) The region of program text where a binding is active is called the ____________ of the binding.

(v) A program whose last action before returning is to invoke itself recursively is called ____________

(vi) The parameter mode in which the parameter is bound to the location of the corresponding argument is referred to as ____________

(vii) A value that can be created at run-time, can be assigned to a variable, can be passed to a procedure, and can be returned by a function is called ____________

(viii) The evaluation rule to perform the (leftmost) innermost reduction first is called ____________

(ix) In Scheme a primitive item such as a number, a string, or a symbol, but not a list is called ____________

(x) A value that can legally occur in given context must have a type that is ____________ with the context.

Solution:

i lambda calculus
ii binding time
iii heap
iv scope
v tail recursive
vi pass/call by reference
vii first class
viii applicative order
ix an atom
x compatible
2. Lambda Calculus (20 points).
Perform ONE normal-order β-reduction on the following expression in the lambda calculus
\((\lambda x.x)(\lambda y.yzy)(\lambda w.ww)\)

**Solution:**

There are two ways to interpret \((\lambda x.x)(\lambda y.yzy)(\lambda w.ww)\)

A. \(((\lambda x.x)(\lambda y.yzy))(\lambda w.ww)\) (default)

B. \((\lambda x.x)((\lambda y.yzy)(\lambda w.ww))\)

Either is correct.

A. \(((\lambda y.yzy)(\lambda y.yzy))(\lambda w.ww)\)

B. \(((\lambda y.yzy)(\lambda w.ww))(\lambda y.yzy)(\lambda w.ww)\)
3. Call-by-name vs. call-by-reference (20 points).
Suppose you have two non-standard compilers, C_R and C_N. C_R accepts standard C syntax as input but
uses non-standard semantics: unlike C, C_R uses call-by-name for passing parameters. Similarly, C_N uses
call-by-reference.
Write the C-like code for a function called printName() such that
• When this code is compiled with C_R, it prints your first name.
• When this code is compiled with C_N, it prints your last name.
You may introduce additional variables and functions as needed.
Technical point (probably not relevant): Both compilers use call-by-value for parameters that are not legal
I-values (e.g., x, y or 3).

Solution:
```c
void printName()
```
```c
{
    int i = 0;
    int A[2] = {0,0};
    F(i, A[i]);
    if (A[1] == 1) {
        printf("Allan\n");
    }
    else {
        printf("Gottlieb\n");
    }
}
```
```c
void F(x, y) {
    x = x + 1;
    y = 1;
}
```

It's also correct to write printName() as the following:
```c
int printName()
```
```c
{
    int i = 0;
    int A[2] = {0,0};
    F(i, A[i]);
    if (A[1] == 1) {
        printf("Allan\n");
    }
    else {
        printf("Gottlieb\n");
    }
    return 0;
}
```

You probably want to review Problem 6 in hw4. Prof. Allan gives a very good ada program
illustrating the 4 different parameter passing modes.
4. Scheme Programming (20 points).
Your Scheme system is very slightly broken, the append library function, which appends one list to the end of another, is missing so you must write it in Scheme. Examples of append:

((append 1 2 3) 0) => (1 2 3)
((append 1 2 3) '0) => '1 2 3
((append 1 2 3) (1 2 3)) => (1 2 3 1 2 3)

Solution:

(define APP
  (lambda (x y)
    (cond
      ((null? x) y)
      (else (cons (car x) (APP (cdr x) y))))))

or

(define APP
  (lambda (x y)
    (if (null? x) y
      (cons (car x) (APP (cdr x) y))))
5. Memory layout (20 points).
Suppose our architecture requires 1-byte for char, 4-bytes for int, and 8-bytes for double. Suppose every record must begin on an address that is a multiple of 8. Suppose every field must begin on an address that is a multiple of its size. Consider the following array of records ARR.

```c
struct rec {
    char c1;
    int i;
    char c2;
    double d;
} ARR[1000];
```

5A (5 points). What is the total amount of data in ARR (not counting padding)?

Solution:

\[
(1 + 4 + 1 + 8) \times 1000 = 14,000
\]

5B (5 points). What is the minimum amount of memory used by ARR if no reordering is done?

Solution:

\[
(1 + 3 + 4 + 1 + 7 + 8) \times 1000 = 24,000
\]

5C (5 points). What is the minimum amount of memory used by ARR if the best reordering is done?

Solution:

The best reordering is `double int char char`

\[
(8 + 4 + 1 + 1 + 2) \times 1000 - 2 = 15,998 \quad // \text{does not need 2 bytes for the last record.}
\]

5D (5 points). What is the total amount of memory used by the record REC if no reordering is done?

```c
struct arr {
    char c1[1000];
    int i[1000];
    char c2[1000];
    double d[1000];
} REC;
```

Solution:

\[
1000 \times 1 + 1000 \times 4 + 1000 \times 1 + 1000 \times 8 = 14,000
\]