

1. [15] Consider the following EBNF grammar:

$$X ::= \mathbf{a} X \mid \mathbf{a} X \mathbf{c} X \mid \mathbf{d}$$

- a. Show that the grammar is ambiguous by finding a string that has two different leftmost derivations, and show the derivations.

an ambiguous string: **a a d c d**

two derivations:

$X$	$X$
$\mathbf{a} X$	$\mathbf{a} X \mathbf{c} X$
$\mathbf{a} \mathbf{a} X \mathbf{c} X$	$\mathbf{a} \mathbf{a} X \mathbf{c} X$
$\mathbf{a} \mathbf{a} \mathbf{d} \mathbf{c} X$	$\mathbf{a} \mathbf{a} \mathbf{d} \mathbf{c} X$
<b>a a d c d</b>	<b>a a d c d</b>

- b. Change the grammar to eliminate the ambiguity by adding a terminal **z**.

$$X ::= \mathbf{a} X \mathbf{z} \mid \mathbf{a} X \mathbf{c} X \mathbf{z} \mid \mathbf{d}$$

- c. Write two strings in the changed grammar, one reflecting each of the two interpretations of the ambiguous string.

With this change, the string above would be written as one or the other of

**a a d c d z z**  
**a a d z c d z**

2. [10] Translate the following expression as required for evaluation using a stack, and show the steps of its stack evaluation.

$$3 + ((4 * 5) - (9 / 3))$$

The solution (stack contents underlined):

<u>3</u>	3 4 5 * 9 3 / - +
<u>3</u> 4	4 5 * 9 3 / - +
<u>3</u> <u>4</u> 5	5 * 9 3 / - +
<u>3</u> <u>20</u>	* 9 3 / - +
<u>3</u> <u>20</u> 9	9 3 / - +
<u>3</u> <u>20</u> <u>9</u> 3	3 / - +
<u>3</u> <u>20</u> <u>3</u>	/ - +
<u>3</u> <u>17</u>	- +
<u>20</u>	+

3. [16] Write new definitions of this function:

$$f = \backslash xs \rightarrow [ x+3 \mid x <= xs; x > 0 ]$$

- a. using explicit recursion and pattern-matching, without guards. Include the function's most general type.

$$f :: \text{Num } a \Rightarrow [a] \rightarrow [a]$$

$$f [] = []$$

$$f (x:xs) = \text{if } x > 0 \text{ then } x+3 : f xs \text{ else } f xs$$

- b. using explicit recursion and guards, without pattern-matching.

$$f xs = \begin{cases} \text{null } xs & = [] \\ x > 0 & = x+3 : f xs' \\ \text{otherwise} & = f xs' \end{cases}$$

where  
 $x = \text{head } xs$   
 $xs' = \text{tail } xs$

- c. using higher-order functions and function composition, avoiding explicit recursion.

$$f = \text{map } (\backslash x \rightarrow x+3) . \text{filter } (\backslash x \rightarrow x > 0)$$

4. [15] Write a Haskell program `sorter` that inputs lines until it inputs an empty line, at which point it outputs the lines in sorted order.

```
> import List
> sorter :: IO ()
> sorter = gets1 []
> where
>   gets1 :: [String] -> IO String
>   gets1 lines = do s <- getLine
>                   if null s
>                   then putStr (unlines lines)
>                   else gets1 (insert s lines)
```

5. [10] Use the method of weakest preconditions to determine the truth (or otherwise) of

$$\{x < y+3\} \quad x, y := y-1, x+5 \quad \{x < y+2\}.$$

```
wp("x, y := y-1, x+5", x < y+2)
= {def. '='}
(x < y+2)y-1, x+5x, y
= {substitution}
y-1 < x+5+2
= {simplification}
y < x+8
≠ {predicate calculus, counterexample: let x = 0, y = 10}
x < y+3
```

So the proposition is false. ■

6. [10] Given this definition

```
data Tree a = Nil | Node (Tree a) a (Tree a)
```

define the function `mapTree` so that for any `t::Tree a` and function `f :: a -> b`, `mapTree f t` is a tree that has the same shape as `t`, but in which each of `t`'s labels, say `x`, is replaced by `f x`.

```
> mapTree f Nil = Nil
> mapTree f (Node t1 x t2) = Node (mapTree f t1) f x (mapTree f t2)
```

7. [15] Pay rates are typically specified in terms of an amount of money per unit time, where the time unit may be an hour, a week, or a month.

- a. Define a Haskell algebraic type `PayRate` to represent rates of pay on an hourly, weekly, or monthly basis. Assume that monetary amounts are integers.

```
> data PayRate = Hourly Int
>           | Weekly Int
>           | Monthly Int
```

- b. Write Haskell declarations so that if `r1, r2 :: PayRate`, expressions such as `r1 < r2` are valid and give useful results. Assume that a week is equivalent to 40 hours, and that a year is equivalent to 12 months and to 52 weeks.

```
> instance Eq PayRate where
>   r1 == r2 = annual r1 == annual r2

> instance Ord PayRate where
>   r1 <= r2 = annual r1 <= annual r2

> annual :: PayRate -> Int
> annual (Monthly r) = 12 * r
> annual (Weekly r) = 52 * r
> annual (Hourly r) = 40 * 52 * r
```

8. [12] For each of the following code samples, determine (i) whether a static type check would detect an error, and (ii) whether executing the code would result in a dynamic type error. Assume that the code is C, but with either static or dynamic type checking (actual C, of course, has only static checking).

Indicate your answer by circling **one** of **s** static, **d** dynamic, **b** both, **n** neither.

For all of the samples assume the following declarations:

```
int x = -1, y = 2; struct S{ int p, q; }; struct S z = {3,4};
```

- a. `int q = x || z;`     s    d    b    n
- b. `int q = x && z;`     s    d    b    n
- c. `int q = x && y;`     s    d    b    n
- d. `int q = x ? y : z;`    s    d    b    n
9. [10] Predict the following program's output

```
integer x = 10; integer y = 1;
function psi( integer a ) : integer
{
  integer x = 9;
  return a*x + y;
}
function omega( integer p ) : integer
{
  return p*x + psi(x);
}
procedure main()
{
  integer x = 3;
  y := 20;
  print omega( x );
}
```

assuming that arguments are bound to parameters by value, and that functions' free variables are bound

- a. statically.

```
omega(3)
~ 3*x + psi(x) where x = 10
~ 3*10 + psi(10)
~ 30 + 10*x + y where x = 9, y = 20
~ 30 + 10*9 + 20
~ 30 + 90 + 20
~ 140
```

- b. dynamically.

```
omega(3)
~ 3*x + psi(x) where x = 3
~ 3*3 + psi(3)
~ 3*3 + 3*x + y where x = 9, y = 20
~ 3*3 + 3*9 + 20
~ 9 + 3*9 + 20
~ 56
```

10. [12] Given the following fragmentary C program

```
double x;
typedef struct { int a; int b; } S;
int y;
void f()
{
    double* z;
}
int main()
{
    S d;
    int x;
    double b;
    x; y; d.b;
}
```

assuming that `ints` and pointers occupy 4 bytes each, and `doubles` occupy 8 bytes, what offsets would be generated for the three variable references in `main`? Indicate a global reference by appending `G` to its offset.

```
x: 8
y: 8 G
d.b: 4
```

11. [15] Design an Ada task type `MaxTemp` to provide maximum-temperature readings. The task's clients consist of
- temperature reporters, which call `report` to report the current temperature (in integer degrees).
  - temperature requestors, which call `getMax` to obtain the current maximum.
  - a system client, which calls `clear` to reset the maximum (after the maximum is reset, no `getMax` calls are accepted until a new temperature has been reported)

```
task type MaxTemp
  entry report( temp: in integer );
  entry getMax( temp: out integer );
  entry clear;
end MaxTemp

task body MaxTemp
  maxTemp : integer;
  maxValid: boolean := false;
begin
  loop
    select
      accept report( temp: in integer ) do
        if not maxValid or maxTemp < temp then
          maxTemp := temp;
          maxValid := true;
        end if;
      end report;
    or
      when maxValid =>
        accept getMax( temp: out integer ) do
          temp := maxTemp;
        end query;
    or
      accept clear do
        maxValid := false;
      end clear;
    end select;
  end loop;
end MaxTemp;
```

12. [10] Predict the following C++ program's output.

```
#include <iostream.h>

class A
{
public:
    virtual void a() { cout << "1 "; }
    virtual void b() { cout << "2 "; }
    virtual void c() { a(); b(); }
    virtual void d() { a(); b(); }
};

class B : public A
{
public:
    void a() { cout << "4 "; }
    void b() { cout << "5 "; }
    void c() { a(); b(); }
    void d() { cout << "6 "; }
};

void f( A& x, A y, A* z )
{
    x.a(); y.a(); z->a(); cout << endl;
    x.c(); y.c();      cout << endl;
    y.d(); z->d(); cout << endl;
}

int main()
{
    B b; f( b, b, &b );
    return 0;
}
```

The output:

```
4 1 4
4 5 1 2
1 2 4 2
```

13. [15] How does the following C++ class definition fail to satisfy the Law of the Big Three?

```
#include <iostream.h>

class String
{
public:
    String( const char* );
    ~String() { free(); }
    void print() const;
private:
    char* chars;
    int length;
    void free() { delete chars; }
};

String::String( const char* s )
{
    length = strlen( s );
    chars = new char[ length+1 ];
    strcpy( chars, s );
}

void String::print() const
{
    cout << chars << endl;
}
}
```

It has a destructor, but it lacks an assignment operator and a copy constructor.

Add what is needed to satisfy the law.

```
#include <iostream.h>

class String
{
public:
    String( const char* );
    String( const String& );
    ~String(){ free(); }
    String& operator=( const String& );
    void print() const;
private:
    char* chars;
    int length;

    void copy( const String& );
    void free(){ delete chars; }
};

String::String( const char* s )
{
    length = strlen( s );
    chars = new char[ length ];
    strcpy( chars, s );
}

String::String( const String& b )
{
    copy( b );
}

String& String::operator=( const String& b )
{
    if( chars != b.chars )
    {
        free();
        copy( b );
    }
    return *this;
}

void String::copy( const String& b )
{
    length = b.length;
    chars = new char[ length+1 ];
    strcpy( chars, b.chars );
}

void String::print() const
{
    cout << chars << endl;
}
```