

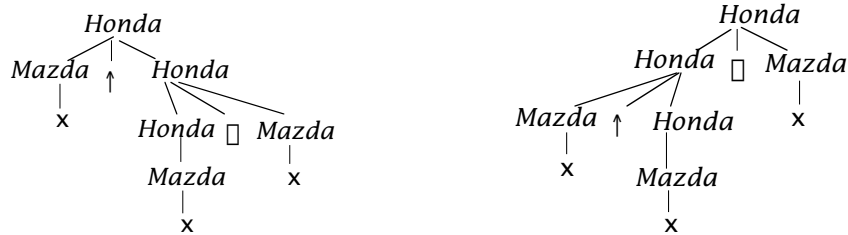
1. [15] Demonstrate, using one or more sample strings and their *parse trees*, that the following grammar is ambiguous.

$Honda ::= Mazda \uparrow Honda \mid Honda \square Mazda \mid Mazda$
 $Mazda ::= x \mid y \mid z$

The string

$x \uparrow x \square x$

can be parsed as either of



$x \uparrow (x \square x)$

$(x \uparrow x) \square x$

This can be seen as either a precedence problem or an associativity problem—two operators of the same precedence have opposite associativities.

Change the grammar to an unambiguous one which describes the same language.

Here's one way to fix it, by changing the precedence relationship between the operators::

$Honda ::= Honda \uparrow Toyota \mid Toyota$
 $Toyota ::= Toyota \square Mazda \mid Mazda$
 $Mazda ::= x \mid y \mid z$

Another way to fix it is to make the operators have the same associativity:

$Honda ::= Mazda \uparrow Honda \mid Mazda \square Honda \mid Mazda$
 $Mazda ::= x \mid y \mid z$

2. [15] Define `merge` in Haskell so that if `as` and `bs` are strictly increasing lists, `merge as bs` is a strictly increasing list containing all the elements of `as` and all the elements of `bs`. Give `merge`'s most general type.

```
> merge :: Ord a => [a] -> [a] -> [a]
> merge [] ys = ys
> merge xs [] = xs
> merge (x:xs) (y:ys)
> = case compare x y of
>   LT -> x : merge xs (y:ys)
>   GT -> y : merge (x:xs) ys
>   EQ -> x : merge xs ys
```

3. [15] Use Newton's method for calculating cube roots, in which successively better approximations to the cube root of N are given by

$$a_{i+1} = \frac{2}{3} \cdot a_i + \frac{N}{3 \cdot a_i^2},$$

to define a function that (a) computes an infinite list of improving approximations and (b) selects the first approximation that differs from its predecessor by at most $1.0e-6$.

```
> cubrt :: Float -> Float
> cubrt n = within 1.0e-6 apps
>   where
>     apps = iterate (next n) n -- alternative: apps = n : [ next n a | a <- apps ]
>     next :: Float -> Float -> Float
>     next n a = (2/3) * a + n/(3 * a*a)
>     within :: Float -> [Float] -> Float
>     within eps (a:b:bs)
>       | abs (a-b) <= eps = b
>       | otherwise       = within eps (b:bs)
```

4. [15] Translate this system of Ada tasks to a unix-shell pipe program. Rewrite each Ada task as a program in C or C++, and give the shell program that connects them in the same relationship as the Ada tasks.

```
task A
  sa : string;
begin
  loop
    ... produce( sa ); ...
    B.input( sa ); ...
  end loop
end A;
```

```
task B
  sb : string;
begin
  loop
    accept input(s : in String) do
      sb := s
    end input;
    ... consume( sb ); ...
  end loop
end B;
```

Assume that Ada "code" like

```
... produce( sa ); ...
```

translates to C/C++ "code" that looks identical:

```
... produce( sa ); ...
```

The C++ programs:

```
// file A.cpp
int main()
{ string sa;
  for(;;)
  {
    ... produce( sa ) ...
    cout << sa << endl;
  }
}
```

```
// file B.cpp
int main()
{ string sb;
  for(;;)
  {
    cin.getLine( sb );
    ... consume( sb ); ...
  }
}
```

The shell program:

```
A | B
```

5. [15] Predict the output of the following program:

```
#include <iostream.h>

class X
{ public:
  X() { cout << 1 << ' '; }
  X( const X& ) { cout << 2 << ' '; }
  ~X() { cout << 3 << ' '; }
  X& operator=( const X& ) { cout << 4 << ' '; }
};

X f( X x ) { return x; }

X& g( X& x ) { return x; }

int main()
{
  X a;
  X b = a;
  cout << endl;
  a = b;
  cout << endl;
  a = f( b );
  cout << endl;
  b = g( a );
  cout << endl;
  return 0;
}
```

The output:

```
1 2
4
2 2 4 3 3
4
3 3
```

6. [10] Define a function `all` so that

```
all( i, iMin, iMax, f(i) )
```

corresponds to

```
( $\forall i \mid iMin \leq i \leq iMax : f(i)$ )
```

What method of binding arguments to parameters is necessary for this to work?

```
all( i, iMin, iMax : integer; exp : Bool ) : Bool
{
  i := iMin;
  while i <= iMax do
  { if not exp then return False;
    i := i+1
  }
  return True
}
```

This technique, known as Jensen's device, requires *by-name* binding.

7. [15] Define in Haskell `NumList`, a type of list in which each element can contain either an `Int` or a `Float`, and define `sumNumList :: NumList -> Float`.

We show two solutions. The first one uses built-in lists:

```
> data IntFlt = AnInt Int | AFloat Float
>
> type NumList = [IntFlt]
>
> sumNumList :: NumList -> Float
> sumNumList [] = 0
> sumNumList (AnInt n : nums) = fromInt n + sumList nums
> sumNumList (AFloat n : nums) = n          + sumList nums
```

The second solution “reinvents” lists:

```
> data NumList = Nil | Inode Int NumList | Fnode Float NumList
>
> sumNumList :: NumList -> Float
> sumNumList Nil = 0
> sumNumList (Inode n nums) = fromInt n + sumList nums
> sumNumList (Fnode n nums) = n          + sumList nums
```

8. [10] If this C program's first output is 4, what is its second output?

```
int main()
{
    int a[] = {0,1,2,3,4,5,6,7,8,9};
    cout << sizeof( int ) << *(a+2*sizeof(int));
}
```

The second output:

8

9. [10] How—and why— would a type checker respond to the statement

```
x := if roman then "XVII" else 17;
```

assuming that the type of `roman` is `Boolean`, and that type checking is...

- a. static?

A static type checker must be able to determine the type of every expression by analyzing its text. In this case, the type of the `if`-expression may be string or integer, depending on the value of `roman`. Hence this expression would be rejected.

- b. dynamic?

A dynamic type checker requires only that every function is applied to arguments of the appropriate type, so this statement would be accepted.

10. [15] The following fragmentary C program text contains a type definition, declarations of several variables, and some references to the variables. The variable references are tabulated in a table following the program.

To answer this question, write the address information that a compiler would generate for each of the variable references in the appropriate line of the table. Assume

- a typical stack implementation
- space is allocated for variables in the order in which they are declared, beginning with offset 0
- no requirement that variables be aligned on word boundaries.

Assume that addresses identify bytes, and that the basic data types' space requirements (in bytes) are as follows:

char 2 **int** 4 **float** 8

The program:

```
int a[10];
float b;
typedef struct{ int a, int b[3]; } S;
char c;

void f()
{
    char x[5]; S r[3]; int t;
    static int b;
    ... t ... c ... b ...
}

int main()
{
    S p[2];
    ... p[1].b[2] ... b ...
    return 0;
}
```

Answer by filling in the table. Indicate the distinction between local and global variables by appending a **G** to global-variable references. If a reference is invalid, explain. Partial credit may be given for incorrect answers, but *only* if you've shown your work *clearly*.

First we calculate the space requirements for values of type S:

```

a      4  0-3
b     3*4 4-15
```

Then we build a table for each block showing each variable's size and its address(es) in its block's activation record.

```

global  a   10*4  0-39
        b    8   40-47
        c    2   48-49
        f::b  4   50-53

f       x    5*2   0-9
        r   3*16 10-57
        t    4   58-61

main   p   2*16  0-31
```

Then we use the address information to fill in the variables' activation-record addresses:

t	58
c	48 G
b (in f)	50 G
p[1].b[2]	1*16 + (4 + 2*4) = 28
b (in main)	40 G

11. [20] Suppose two coordinates, represented by integer variables x and y (both initially 0), are to be incremented repeatedly (by 1 each time) separately by various Ada client tasks. Write the body of a server task XY which (a) maintains the invariant $\text{abs}(x-y) < d$, and (b) enables any client to obtain the variables' current values.

```
task body XY is
  x,y : integer;
begin
  x := 0; y := 0;
  loop
    select
      accept xyout( xo, yo : out integer ) do
        xo := x; yo := y;
      end xyout;
    or
      when x-y < d =>
        accept incx do
          x := x+1;
        end
    or
      when y-x < d =>
        accept incy do
          y := y+1;
        end
    end select;
  end loop
end XY;
```

12. [10] If the following program compiles, what is its output? If it won't compile, what would the diagnostic say?

```
#include <iostream.h>

class Simon
{
public:
  void s1(){ cout << "1 "; }
  virtual void s2(){ cout << "2 "; }
  void f(){ s1(); s2(); }
};

void paul( Simon& r, Simon v )
{
  r.f(); v.f();
}

class Garfunke1 : public Simon
{
public:
  virtual void s1(){ cout << "3 "; }
  void s2(){ cout << "4 "; }
};

int main()
{
  Simon s; Garfunke1 g;

  s.f(); s.s1(); s.s2(); cout << endl;
  g.f(); g.s1(); g.s2(); cout << endl;

  paul( g, g ); cout << endl;

  return 0;
}
```

It compiles OK. The output:

```
1 2 1 2
1 4 3 4
1 4 1 2
```