

You may use, without defining them, any functions, types, or other items defined in class lecture notes, published libraries, and published homework solutions.

1. [15] Given the following grammar,

$$\langle \text{larry} \rangle ::= \langle \text{curly} \rangle \mid \langle \text{larry} \rangle \bullet \langle \text{larry} \rangle$$

$$\langle \text{curly} \rangle ::= \langle \text{moe} \rangle \mid \langle \text{curly} \rangle \blacktriangledown \langle \text{moe} \rangle$$

$$\langle \text{moe} \rangle ::= ( \langle \text{larry} \rangle ) \mid 1 \mid 2 \mid 3$$

determine, if possible, the associativity and precedence of the operators, and parenthesize the following expression to make its structure explicit. If the grammar is ambiguous, show two different parenthesizations.

The grammar is ambiguous—the associativity of  $\bullet$  is both left and right. Hence the expression can be parenthesized in either of the following ways:

$$1 \bullet ((2 \blacktriangledown 1) \bullet ((2 \blacktriangledown 3) \blacktriangledown 2))$$

$$(1 \bullet (2 \blacktriangledown 1)) \bullet ((2 \blacktriangledown 3) \blacktriangledown 2)$$

2. [10] Building a document-indexing application in Haskell, you need to define a function `wordNum` which takes two lists—a list of words and a separate list of the page numbers on which the words occur—and creates a list of index entries in which each word is followed by a space and then its page number. For example,

```
wordNum ["hurricane", "tornado"] [22,16] ~~~ ["hurricane 22", "tornado 16"]
```

A solution:

```
> wordNum :: [String] -> [Int] -> [String]
> wordNum words nums = [ w ++ " " ++ show n | (w,n) <- zip words nums ]
```

3. [15] Rewrite the following Haskell function definition

```
f xs n = if not (null xs) && n==0 then head xs
         else if not (null xs) && n > 0 then f (tail xs) (n-1)
         else error "f"
```

in three different ways, preserving the order of the tests:

- a. [5] using guards only

```
f xs n
  | not (null xs) && n==0 = head xs
  | not (null xs) && n > 0 = f (tail xs) (n-1)
  | otherwise             = error "f"
```

- b. [5] using parameter patterns only

```
f (x:_ ) 0      = x
f (_:xs) n | n>0 = f xs (n-1)
f _      _      = error "f"
```

- c. [5] using a **case** expression only

```
> f xs n = case (xs, compare n 0) of
>           (x:_, EQ) -> x
>           (_:xs, GT) -> f xs (n-1)
>           _         -> error "f"
>
> f xs n = case (xs, n==0, n>0) of
>           (x:_, True, _) -> x
>           (_:xs, _, True) -> f xs (n-1)
>           _             -> error "f"
```

4. [15] Define a parser combinator `sepBy` so that `sepBy p q` recognizes zero or more items recognized by `q`, separated from each other by items recognized by `p`. The parser `sepBy p q` should deliver a list containing the items recognized by `q`.

For example, if `dash` is a parser that recognizes hyphens, and `integer` is a parser that recognizes integers, then the item delivered by

```
papply (sepBy dash nat) "32-24-16-23"
```

would be

```
[32,24,16,23]
```

Then use `sepBy` to define a parser `listOf` so that `listOf p` recognizes a sequence of items recognized by `p`, separated by commas and enclosed in braces. For example,

```
papply (listOf nat) "{3,12,88,9}"
```

would deliver

```
[3,12,88,9]
```

A solution:

```
> sepBy :: Parser a -> Parser b -> Parser [b]
> sepBy p q = do x <- q
>              xs <- many (do p; q)
>              return (x:xs)
>
>          +++
>          return []
>
> listOf :: Parser a -> Parser [a]
> listOf p = do char '{'
>              xs <- sepBy (char ',') p
>              char '}'
>              return xs
```

5. [10] Translate this C fragment to Guarded Command notation.

```
for( k = 0; k < n; k++ )
{
  if( M[k] < 0 )
    { m = k; break; }
  M[k] += k;
}
```

```
k := 0;
```

```
do k < n  $\wedge$  M[k]  $\geq$  0  $\rightarrow$  M[k] := M[k] + k; k := k+1 od;
```

```
if k < n  $\wedge$  M[k] < 0  $\rightarrow$  m := k  $\parallel$  k  $\geq$  n  $\vee$  M[k]  $\geq$  0  $\rightarrow$  skip fi
```

6. [10] Determine the most general type of the following Haskell function:

```
> bizarre (x,y)
>   | y == 3    = 7
>   | x < 17   = y
>   | otherwise = y * 2
```

Its type is

```
bizarre :: (Num a, Num b, Ord b) => (b,a) -> a
```

7. [10] Suppose that the first output from this program

```
#include <iostream.h>
int main()
{
    typedef struct { double y; double x; } T;
    T array[8];
    cout << sizeof(T) << &(array[8]) - &(array[2]);
    return 0;
}
```

is 16. What is its second output, if any?

Answer: 6.

8. [10] Here is a program in a programming language that's like C++ except that for binding arguments to parameters it uses the *call-by-name* method. Predict what this program would print.

```
#include <iostream.h>
int p( int x )
{
    return x + x;
}
int f( int y )
{
    y++; return y;
}
int main()
{
    for( int k = 0; k < 7; k++ )
        cout << p( f(k) ) << " ";
    return 0;
}
```

Output:

3 9 15

Explanation: Procedure *f* increments its argument and returns the argument's new value. Procedure *p* calls its argument twice and returns the sum of the two calls; each call of *p* therefore increments *k* by 2 and returns  $k-1 + k$ , or  $2*k-1$ . Hence the successive values of *k*, and the values printed, are:

k	p	printed	after k++
0	2	3	3
3	5	9	6
6	8	15	9

9. [25] Certain software packages are offered under site licenses which allow them to be installed on file servers, but impose an upper bound on the number of copies that may be running at any given time. Design an Ada “permission server” task type that would limit the number of concurrently running copies of a given software package. Assume that
- some administrator process (not your responsibility) will create an instance of your server task type, and will then communicate to it the maximum number of copies of its software package that are to be allowed to run concurrently;
  - each running copy of the software package will request permission from your server task when it begins execution, and will inform your server when it quits.
  - from time to time the administrator process will query your server to find out (1) how many copies of the software package have started execution and (2) the number of copies currently running.

Your answer should consist of

- a. [20] an Ada definition of the permission-server task type

```
task type PermissionServer is
  entry init( limit : in integer );
  entry please();
  entry thanks();
  entry query( nStarts, nRunning : out integer );
end PermissionServer;
task body PermissionServer is
  running : integer;
  maxRunning : integer;
  starts : integer;
begin
  accept init( limit : in integer ) do
    maxRunning := limit;
    starts := 0;
    running := 0;
  end init;
  loop
    select
      when running < maxRunning =>
        accept please() do
          running := running + 1;
          starts := starts + 1;
        end please;
      or
        accept thanks() do
          running := running - 1;
        end thanks;
      or
        accept query( nStarts, nRunning : out integer ) do
          nStarts := starts;
          nRunning := running;
        end query;
    end select;
  end loop;
end PermissionServer;
```

- b. [5] the Ada statements by which a permission-server task would be called.

Assume that in the administrator task the statements

```
ps : access PermissionServer;
ps := new PermissionServer;
```

have established ps as (a pointer to) a PermissionServer, then the administrator calls

```
ps.init( upperBound ); -- sets server's upper bound
ps.query( usage, current ); -- sets usage to number of starts
-- sets current to number of copies currently running
```

The software package that is monitored by the server calls

```
ps.please(); -- when starting up
ps.thanks(); -- when shutting down
```

- 10.** [15] Predict the output of the following program, and annotate each output with
- the line number of the main statement which caused it
  - a brief explanation of the event in the program's execution that caused it.

```
#include <iostream.h>

class A
{
public:
    A( int k = 5 ){ n = k; cout << "A\n"; }
    A( const A& r ){ n = r.n; cout << "B\n"; }
    A& operator=( const A& r ){ n = r.n; cout << "C\n"; return *this; }
    ~A(){ cout << "D\n"; }
    int n;
};

A f( A x ){ cout << "f\n"; return x; }

void g( A& r, A& s )
{
    cout << "g\n";
}

int main()
{
    A a;           // 1
    A b = a;      // 2
    A c(3);       // 3
    b = f(b);     // 4
    g(a,b);      // 5
    return 0;    // 6
}
```

The output, annotated:

- A 1. a is initialized by A::A(int)
- B 2. b is initialized from a by copy constructor A::A(const A&)
- A 3. c is initialized by A::A(int)
- B 4. f's by-value parameter x is copied from b by A::A(const A&)
- f 4. execution of cout << "f " in f
- B 4. anonymous value returned by f is copied from x by A::A(const A&)
- C 4. assignment of f's return value to b
- D 4. destructor for f::x
- D 4. destructor for f's return value
- g 5. execution of cout << "g "
- D 6. destructor for c
- D 6. destructor for b
- D 6. destructor for a

11. [10] If the following program would compile, what would be its output? If it would not compile, what would the diagnostic say?

```
#include <iostream.h>

class A
{
public:
    void bO{ cout << "1 "; }
    virtual void cO{ cout << "2 "; }
    void fO{ aO; cO; }
};

class B : public A
{
public:
    void aO{ cout << "3 "; }
    void cO{ cout << "4 "; }
};

void play( A v, A& r )
{
    v.aO; v.cO; r.aO; r.cO;
}

int mainO
{
    A x; B y;

    x.bO; x.cO; cout << endl;
    y.fO; y.aO; y.cO; cout << endl;

    play( y, y ); cout << endl;

    return 0;
}
```

It doesn't compile. The error message is

```
S04B1.cpp: In member function `void A::fO':
S04B1.cpp:8: error: `a' undeclared (first use this function)
```

12. [10] Write a program in C or C++ which could be used as a process in a Unix pipeline to add line numbers to lines of input. For example,

<u>input</u>	<u>output</u>
first line	0 first line
second line	1 second line

You may assume that no line will be longer than 100 characters.

```
#include <iostream.h>

int mainO
{
    int n = 0;
    char line[100];

    while( cin )
    {
        cin.getline( line, 100 );
        cout << n << ' ' << line << endl;
        n++;
    }

    return 0;
}
```

13. [15] The following C++ Stack class has reference semantics. Add and remove definitions as necessary to obtain value semantics. Wherever possible, use existing code; indicate with arrows where your new code would be added or existing code would be moved, and cross out code that should be removed.

```
#include <iostream.h>

class Stack
{
public:
    Stack(){ top = 0; }
    void push( int& x ){ top = new Cell( x, top ); }
    int pop(){ int x = top->datum; Cell* t = top; top = top->next; delete t; return x; }
    bool isEmpty(){ return top == 0; }
    Stack clone(){ Stack s; s.copy( *this ); return s; }
    void dispose();
private:
    class Cell
    {
    public:
        Cell( int d, Cell* n ){ datum = d; next = n; }
        int datum;
        Cell* next;
    };
    Cell* top;
    void copy( const Stack& );
};

void Stack::copy( const Stack& b )
{
    if( b.top == 0 ) top = 0;
    else
    {
        Cell* p = b.top;
        Cell* q = top = new Cell( p->datum, 0 );
        p = p->next;
        while( p != 0 )
        {
            q = q->next = new Cell( p->datum, 0 );
            p = p->next;
        }
    }
}

void Stack::dispose()
{
    while( top )
    {
        Cell* t = top;
        top = top->next;
        delete t;
    }
}
```

Modified code (new code in **bold**, deleted code shown by ~~strikethrough~~):

```
class Stack
{
public:
    Stack(){ top = 0; }
    Stack( const Stack& b ){ copy(b); }
    ~Stack(){ dispose(); }
    Stack& operator=( const Stack& b );
    void push( int& x ){ top = new Cell( x, top ); }
    int pop(){ int x = top->datum; Cell* t = top; top = top->next; delete t; return x; }
    bool isEmpty(){ return top == 0; }
    Stack clone(){ Stack s; s.copy( *this ); return s; }
    void dispose();
private:
    class Cell
    {
    public:
        Cell( int d, Cell* n ){ datum = d; next = n; }
        int datum;
        Cell* next;
    };
    Cell* top;
    void copy( const Stack& );
    void dispose();
};

Stack& Stack::operator=( const Stack& b )
{
    if( top != b.top )
    {
        dispose();
        copy( b );
    }
    return *this;
}

void Stack::copy( const Stack& b )
{
    if( b.top == 0 ) top = 0;
    else
    {
        Cell* p = b.top;
        Cell* q = top = new Cell( p->datum, 0 );
        p = p->next;
        while( p != 0 )
        {
            q = q->next = new Cell( p->datum, 0 );
            p = p->next;
        }
    }
}

void Stack::dispose()
{
    while( top )
    {
        Cell* t = top;
        top = top->next;
        delete t;
    }
}
```