

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] Define a BNF grammar for expressions in which
- operands are single lower-case letters
 - parentheses are permitted
 - the operators are
 - § precedence 1, right-associative
 - precedence 2, left-associative

A solution:

```

<expr> ::= <term> § <expr> | <term>
<term> ::= <term> • <factor> | <factor>
<factor> ::= ( <expr> ) | <letter>
<letter> ::= a | b | ... | z

```

2. [10] Define a function `numLines` so that if `lines` is a list of `Strings` and `n0` is an `Int`, `numLines n0 lines` is a list of `Strings` in which successive `Strings` are prefixed by successive integers, beginning with `n0`. For example,

```
numLines 3 ["aa","bb","cc"] ~ ["3 aa","4 bb","5 cc"]
```

Your solution should not use recursion explicitly.

```

> numLines :: Int -> [String] -> [String]
> numLines n0 lines = [ show n ++ " " ++ line | (n,line) <- zip [n0..] lines ]

```

3. [15] Rewrite the following Haskell function definition

```

f :: [a] -> Int -> a
f (x:_) 0      = x
f (_:xs) n | n>0 = f xs (n-1)

```

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

- a. [5] using guards

```

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 conditional expressions

```

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"

```

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

```

> f :: [a] -> Int -> a
> f xs n = case (xs,compare n 0) of
>   (x:_, EQ) -> x
>   (_:xs,GT) -> f xs (n-1)

```

4. [10] Determine the most general type of the following Haskell function:

```

> strange x [y]
>   | x == 3    = 7
>   | y < 17    = x
>   | otherwise = x * 2

```

Its type is

```
strange :: (Num a, Num b, Ord b) => a -> [b] -> a
```

5. [15] Given this grammar for expressions:

```

Exp ::= Num | '(' Op Space Exp { Space Exp } ')'
Num ::= [-] Digit { Digit }
Op ::= + | - | *
Digit ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
Space ::= a space character

```

use the definitions in `ParseLib.hs` to define a parser that parses these expressions and delivers their values. The value of an expression that is not simply a numeral is obtained by applying the operator to the values of the expressions that follow it, i.e., its operands. For example, the value of the expression

```
(+ 20 (* 5 6) -8)
```

is 42.

Look for an opportunity to use `map`, `filter`, and/or `foldr`.

```

> sExp :: Parser Int
>
> sExp = do char '('
>          op <- char '+' +++ char '-' +++ char '*'
>          char ' '
>          a <- sExp
>          as <- many (do char ' '; sExp)
>          char ')'
>          let f = case op of
>                  '+' -> (+)
>                  '-' -> (-)
>                  '*' -> (*)
>          return (foldr f a as)
>
>          +++
>          int

```

6. [10] Translate the following C code to Guarded Command notation.

```

switch( x = y = z )
{ case 0: a++;
  case 1: ++b;
  case 2: c += 6;
}
y := z; x := y;
if x = 0          → a := a+1; b := b+1; c := c+6
  || x = 1        → b := b+1; c := c+6
  || x = 2        → c := c+6
  || x ≠ 0 ∧ x ≠ 1 ∧ x ≠ 2 → skip
fi

```

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

```

#include <iostream.h>

int main()
{
    typedef struct { double y; double x; } T;
    T* array = 0;
    cout << sizeof(T) << &(array[8]) - &(array[4]);
    return 0;
}

```

is 16. What is its second output?

Answer: 4.

8. [10] What is the output of this program:

```
k : int ;
function pr( x, y, n : int ) : int
{
  a : int ;
  a := 0;
  while( y < n )
  {
    a := a + x;
    y := y+1;
  }
  return a;
}

function f( j : int ) : int
{
  return j*k;
}

procedure main()
{
  k := 5;
  print pr( f(2), k, 8 );
}
```

if the parameter-argument binding mechanism being used in it is

- a. by name? 36
b. by value? 30

9. [15] 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 a() { cout << "1 "; }
    virtual void c() { cout << "2 "; }
    void f() { a(); c(); }
};

class B : public A
{
public:
    void a() { cout << "3 "; }
    void c() { cout << "4 "; }
};

void play( A v, A& r )
{
  v.a(); v.c(); r.a(); r.c();
}

int main()
{
  A x; B y;

  x.f(); x.a(); x.c(); cout << endl;
  y.f(); y.a(); y.c(); cout << endl;

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

  return 0;
}
```

It compiles OK. The output:

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

- 10.** [20] Suppose a program you're writing needs lists containing mixtures of element types—some are integers, others are strings. Show how you would solve this problem, defining a type named `MixList` for the lists, along with any other necessary definitions, both in Haskell and in C++.

Also define, for both C++ and Haskell, a function `printSeq` which prints a `MixList`—one element per line—on standard output. Finally, show how `MixList` would be used by defining a 3-element `MixList` named `s0` containing both integer and string elements.

a. Haskell

```
> data ListEl = I Int | S String
> type MixList = [ListEl]
> s0 :: MSeq
> s0 = [ I 3, S " blind" , S " mice." ]
> printSeq :: [ListEl] -> IO ()
> printSeq [] = return ()
> printSeq (a:as) = do case a of
>                       (I n) -> putStrLn (show n)
>                       (S s) -> putStrLn s
>                       printSeq as
```

b. C++

```
#include <iostream.h>

class ListEl
{
public:
    virtual void printEl() = 0;
    ListEl* next;
};

class I : public ListEl
{
public:
    I( int c, ListEl* nxt ) : n(c) { next = nxt; }
    void printEl(){ cout << n << endl; }
private:
    int n;
};

class S : public ListEl
{
public:
    S( char* c, ListEl* nxt ) :s(c) { next = nxt; }
    void printEl(){ cout << s << endl; }
private:
    char* s;
};

typedef ListEl* MixList;

void printSeq( MixList s )
{
    for( ListEl* p = s; p != 0; p = p->next )
        p->printEl();
}

MixList s0 = new I( 3, new S( " blind", new S( " mice.", 0 ) ) );
```

11. [20] Define, in Ada a server task called Gate which acts like a gate through which tasks pass. Gate handles the following calls from its clients:
- **open** The caller (presumably some kind of system administration task) specifies the number of tasks to be allowed to pass through the gate (if zero, the gate stays open until further notice).
 - **close** The gate is closed (also by a system administration task) until further notice, regardless of how many tasks have passed through it; Gate tells its caller how many tasks have passed through the gate during the immediately preceding open period.
 - **passThrough** Called by tasks wishing to pass through the gate; accepted only if the gate is open.

A solution:

```
task Gate is
  entry open( lim : in integer );
  entry close( ct : out integer );
  entry passThrough;
end Gate

task body Gate is
  limit : integer := 0;
  count : integer := 0;
  isOpen : boolean := false;
begin
  loop
    select
      accept open( lim : in integer ) do
        isOpen := true;
        limit := lim;
        count := 0;
      end open;
    or
      accept close( ct : out integer ) do
        isOpen := false;
        ct := count;
      end close;
    or
      when isOpen =>
        accept passThrough do
          count := count + 1;
          isOpen := limit = 0 or count < limit;
        end passThrough;
      end select
    end loop;
end Gate;
```

12. [15] Predict the output of the following program, and annotate each output with a brief explanation of what event in the program's execution caused it.

```
#include <iostream.h>

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

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

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

int main()
{
    A a;
    A b = a;
    A c(3);
    b = f(b);
    g(a,b);
    return 0;
}
```

The output, annotated:

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