

1. [10] Give the types of the following Haskell functions

```
f (x:_) (y1:y2:_) z = if x==z then y1 else y2
g zs = f zs zs
```

Answers:

```
f :: Eq a => [a] -> [b] -> a -> b
g :: Eq a => [a] -> a -> a
```

2. [10] Suppose you are implementing a type NatSet of sets of consecutive natural numbers in C using a single 8-byte unsigned long int.

- a. Assuming that 0 can be a member of a NatSet, what is the largest natural number that can be a member of a NatSet?

63

- b. Define a function subset so that if a and b are NatSets, subset(a, b) returns 1 if a is a subset of b and 0 otherwise.

```
int subset( NatSet a, NatSet b )
{
    return (~(~a | b) == 0);
}
```

3. [10] You are given the following program fragment in C:

```
int a = 0;
int b = (++a || a++) > (a++ && ++a);
printf( "%d %d", a, b );
```

Assuming that all side-effects take effect immediately, what is printed?

In cases where the C definition does not specify the order of operand evaluation, assume that right operands are evaluated first.

b = (++a a++) > (a++ && ++a)	where a = 0
b = (++a a++) > (0 && ++a)	where a = 1
b = (++a a++) > 0	where a = 1
b = (2 a++) > 0	where a = 2
b = 1 > 0	where a = 2
b = 1	where a = 2

So the output is **2 1**

4. [5] Predict the output of this program fragment:

```
struct R { int x; double y; };
R A[20]; cout << &(A[12]) - A;
```

assuming that an int value occupies 4 bytes and a double value occupies 8 bytes.

Output:

12

5. [15] Define a Haskell algebraic type `Course` with fields denoting

```
department  "CS"
number      345
year        2002
semester    Spring, Summer, Fall
```

To denote the semester, define an auxiliary type `Semester`.

Include the definitions and declarations that would be necessary so that the library function

```
sort :: Ord a => [a] -> [a]
```

would sort a list of `Courses` chronologically; courses offered at the same time should be sorted by department and then by number.

```
> data Course = Crs Year Semester Dept Cnum
> deriving (Eq, Ord)
> type Dept = String
> type Cnum = Int
> type Year = Int
> data Semester = Spring | Summer | Fall
> deriving (Eq, Ord)
```

or

```
> data Course = Crs Dept Cnum Year Semester
> type Dept = String
> type Cnum = Int
> type Year = Int
> data Semester = Spring | Summer | Fall
> instance Eq Course where
>   Crs d1 n1 y1 s1 = Crs d2 n2 y2 s2 = d1==d2 && n1==n2 && y1==y2 && s1==s2
> instance Ord Course where
>   Crs d1 n1 y1 s1 <= Crs d2 n2 y2 s2 =
>     y1 < y2 ||
>     y1 == y2 && (s1 < s2 ||
>                 s1 == s2 && (d1 < d2 ||
>                             d1 == d2 && n1 <= n2))
> instance Eq Semester where
>   Spring == Spring = True
>   Summer == Summer = True
>   Fall == Fall = True
>   _ == _ = False
> instance Ord Semester where
>   Spring <= _ = True
>   Summer <= Spring = False
>   Summer <= _ = True
>   Fall <= Fall = True
>   Fall <= _ = False
```

6. [10] Predict the output of the following program

```
integer x, y;  
procedure dee( a : integer )  
{  
  print a; print ' '; print x;  
}  
procedure dum( x : integer )  
{  
  dee( x );  
}  
begin  
{  
  x := 7; y := 4; dee( y );  
  print '\n';  
  x := 2; dum( y );  
}
```

assuming free variables are bound

a. statically

4 7
4 2

b. dynamically

4 7
4 4

7. [10] What is output by the following program

```
int a = 7;  
  
procedure P( int z )  
{  
  z := z - 2; a := a + 2;  
}
```

P(a); print(a);

assuming arguments are bound to parameters

a. by reference?

7

b. by value-result?

5

8. [20] In class we considered a bank-account system in which interference between two concurrent deposits into an account could give an incorrect result. Design an Ada task type that solves the problem of concurrent deposits and withdrawals involving a single checking account by ensuring that all operations on the account are atomic. Equip your task type to handle deposits, balance inquiries, and withdrawals. A single overdraft should be accepted, but after an overdraft no further withdrawals should be accepted until deposits have restored the account to a positive balance. Finally, a request to close the account and return the balance should be accepted, provided the balance is not negative; after accepting a close transaction, the task should terminate itself.

The amounts involved in these transactions should be expressed as type `integer`. Your answer should demonstrate your understanding of the aspects of Ada—and especially its task- related features—that we have studied in class.

```
task type account is
  entry deposit( amt : in integer );
  entry inquiry( bal : out integer );
  entry withdraw( amt : in integer );
  entry close( bal : out integer );
end account;

task body account is
  balance : integer;
  open : boolean;
begin
  balance := 0;
  open := true;
  loop while open
    select
      accept deposit( amt : in integer ) do
        balance := balance + amt;
      end deposit;
    or accept inquiry( bal : out integer ) do
      bal := balance;
    end inquiry;
    or when balance > 0 =>
      accept withdraw( amt : in integer ) do
        balance := balance - amt;
      end withdraw;
    or when balance >= 0 =>
      accept close( bal : out integer )
        do
          bal := balance;
          open := false;
        end close;
    end select
  end loop;
end account;
```