Egzamin z programowania
część II

Zadanie 1. (**6p**) Rozważmy następujący program napisany w C++.

```
class Bazowa {
   int b;
public:
   Bazowa(int bb) {cout << "Tworzę bazowa"; b=bb};
   ~Bazowa() { cout << "Usuwam bazowa (" << b << ")" }
   void drukuj() { cout << b };
};
class Pochodna : public Bazowa {
   int p;
public:
  Pochodna(int bb, int pp) : Bazowa(bb) {cout << "Tworze pochodna "; p=pp};
  ~Pochodna() { cout << "Usuwam pochodna (" << p << ")" }
  void drukuj() { Bazowa::drukuj(); cout << " " << p };
};
```

Opisz działanie następujących fragmentów programów lub powiedz, że są niepoprawne; za każdy podpunkt (**1p**))

a) `{Pochodna p(1,11);}`
b) `Pochodna p(1,11); Bazowa b(2); b=p; b.drukuj();`
c) `Pochodna p(1,11); Bazowa b(2); p=b; p.drukuj();`
d) `Pochodna p(1,11); Bazowa *pb; pb=&p; pb->drukuj();`
e) Ten sam program, co powyżej, ale gdy w metoda `drukuj()` zdefiniowana jest jako wirtualna.
f) `Bazowa *b = new Bazowa(2); b->drukuj(); delete(b);`

Zadanie 2. **(8p)** Implementując kolejkę typu FIFO należy zdefiniować następujące funkcje:

i) `enqueue(q,e)` oznaczającą kolejkę powstałą z kolejki `q` po wprowadzeniu elementu `e`,
ii) `dequeue(q)` oznaczającą kolejkę powstałą z kolejki `q` po wyjściu pierwszego elementu,
iii) `first(q)` oznaczającą pierwszy element kolejki `q`,
iv) `empty(q)` do sprawdzenia, czy kolejka jest pusta.

Zaimplementuj kolejki w SML-u na dwa sposoby:

a) **(3p)** pamiętając kolejkę jako listę,
b) **(5p)** pamiętając kolejkę jako parę list, z których jedna służy do wstawiania, druga do wybierania elementów, jeśli druga lista jest pusta to zastępujemy ją odwróconą listą pierwszą.

Możesz (jeśli chcesz) korzystać z funkcji `@` oraz `reverse`.

Zadanie 3. **(6p)** Odpowiedz na następujące pytania:

a) Jaki najogólniejszy typ ma w SML-u funkcja `f` zdefiniowana jako `fun f x y = x::y@y`?
b) Jaki najogólniejszy typ ma w SML-u funkcja `f` zdefiniowana jako `fun f x y = (x y, x y y)`?
c) Jaka (przykładowo) funkcja ma najogólniejszy typ równy `'a -> 'b`?
d) Jaka funkcja ma najogólniejszy typ równy `('a -> 'a) ->'a -> 'a`?
e) Jaki jest najbardziej ogólny unifikator `app([b],[c,d],L)` oraz `app([X|Xs],Ys,[X|Zs])`?
f) Jaki jest najbardziej ogólny unifikator `h(X,Y,Z)` oraz `h(f(Y,Y),f(Z,Z),f(a,a))`?

W podpunktach c) i d) nie wolno używać więzów typowych.

Zadanie 4. **(9p)** Zdefiniuj w Prologu następujące predykaty:

a) (**1p**) `duplicate(L1,L2)` oznaczający, że lista `L2` składa się z elementów listy `L1` z których każdy wzięty jest dwukrotnie, np. `duplicate([1,2,3],[1,1,2,2,3,3])`.
b) (**2p**) `noDoubles(L1,L2)`, który zachodzi, gdy `L2` powstało z `L1` przez usunięcie powtarzających się elementów, np. `noDoubles([1,2,3,2,1],[1,2,3])`.
c) (**2p**) `substitute(X,Y,L1,L2)` prawdziwy, gdy lista `L2` powstała z `L1` przez zamienienie każdego wystąpienia `X` na `Y`,
d) (**2p**) `path(X,Tree,Path)` prawdziwy, gdy w drzewie `Tree` prowadzi z korzenia ścieżka `Path` do węzła z etykietą `X`. Przykładowo `path(3, br(1,leaf,br(3,leaf,leaf)), [1,3])`.

Wymień (**2p**) trzy sposoby użycia predykatu `substitute` (pomijając ten, w którym wszystkie argumenty są określone, czyli `substitute(+X,+Y,+L1,+L2)`.

Zadanie 5. (7p) Będziemy rozważać termy dwóch rodzajów: definicje zmiennych postaci `def(X,M,N)` oznaczające, że zmienna `X` może przyjmować wartości między `M` a `N`, oraz termy `E1=E2`, gdzie `E1` oraz `E2` są wyrażeniami arytmetycznymi zbudowanymi ze zmiennych, liczb oraz operatowów arytmetycznych.

a) (**3p**) Napisz predykat `defsFirst(L1,L2)` przekształcający listę `L1` termów w listę `L2`, w której występują te same termy, ale definicje są przed równościami

b) (**4p**) Napisz predykat `solve(L)`, który dla listy `L` równań i definicji będzie generował wszystkie rozwiązania problemu zdefiniowanego przez `L`, przy założeniu, że każda zmienna w `L` występuje dokładnie raz w jakiejś definicji. Przykładowo

```
?- solve([def(X,1,3), 2*X=Y, def(Y,2,5)] ).
```

powinno dać odpowiedź `X=1, Y=2`, a po napisaniu średnika `X=2, Y=4`. Kolejny srednik powinien spowodować napisanie `no (more) solutions`.

Zadanie 6. (6p) Drzewo binarne zdefiniowane jest jako

```
datatype a' tree = Leaf | Br of 'a * 'a tree * 'a tree.
```

Zdefiniuj funkcjonał `maptree` (**2p**) spełniający następujące zależności:

i) (maptree f) o reflect = reflect o (maptree f)

ii) (map f) o preorder = preorder o (maptree f)

Udowodnij pierwszą z nich (**4p**). Przypominamy definicję używanych funkcji:

```
fun (f o g) x = f (g x);
fun reflect Leaf = Leaf |
    reflect Br(x,L,P) = Br(x, reflect P, reflect L);
fun map f []  = [] |
    map f (x::xs) = (f x)::(map f xs);
fun preorder Leaf = [] |
    preorder Br(x,l,p) = [x] @ (preorder l) @ (preorder p);
```