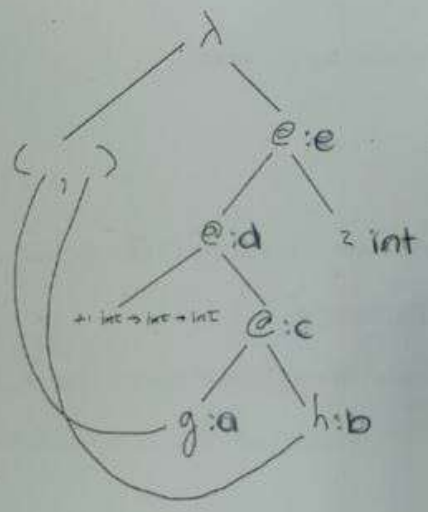
Paradigmas de la Programación – Primer Parcial

9 de Mayo de 2023

Apellido y Nombre: _____

1. **[10 pt.]** Escriba la expresión que se representa en este árbol de tipos y su signatura de tipos.



2. **[20 pt.]** En el siguiente programa, ¿encontraremos una diferencia si el alcance del lenguaje es estático o si el alcance es dinámico? ¿Qué se imprimiría en cada caso?

```
1   procedure p;
2       x: integer;
3       procedure q;
4          begin x := x+1 end;
5       procedure r;
6          x: integer;
7          begin x := 1; q; write(x) end;
8       begin
9       x:= 2;
10      r
11      end;
```

3. [20 pt.] Escriban un programa en pseudocódigo en el que

   a) se define una función que toma una lista y un entero como parámetros, modifica el primer elemento de la lista y devuelve la lista modificada.

   b) se declaran e inicializan una lista `l = [1,2,3]` y un entero `e = 5`, y declaran una lista `ll` que no inicializan

   c) llaman a la función que modifica la lista, con los argumentos `l` y `e`, y asignan el resultado de la función a la lista `ll`

   d) imprimen `l` y `ll`

   Este programa, ¿imprimirá resultados diferentes para pasaje de parámetros por valor, por referencia, por valor-resultado, por nombre y por necesidad? Diga qué imprimirá para los diferentes tipos de pasaje de parámetros.

4. [20 pt.] ¿Qué imprime el siguiente programa?

```
1  class Program
2  {
3      static void Main(string[] args)
4      {
5          try
6          {
7              try
8              {
9                  throw new Exception("exception thrown from try block");
10             }
11             catch (Exception ex)
12             {
13                 Console.WriteLine("Inner catch block handling {0}.", ex.Message);
14                 throw;
15             }
16             finally
17             {
18                 Console.WriteLine("Inner finally block");
19                 throw new Exception("exception thrown from finally block");
20                 Console.WriteLine("This line is never reached");
21             }
22         }
23         catch (Exception ex)
24         {
25             Console.WriteLine("Outer catch block handling {0}.", ex.Message);
26         }
27         finally
28         {
29             Console.WriteLine("Outer finally block");
30         }
31     }
32 }
```

5. [10 pt.] Identifiquen en el siguiente programa dos características no declarativas, subráyenlas en el código y digan cómo se llaman.

```
1 total = 0
2
3 for i in range(1, 6):
4     square = i * i
5     total += square
6
7 print(total)
```

6. [20 pt.] Le preguntamos a chatGPT cómo se usaban correctamente los métodos virtuales en C++, y nos contestó con el siguiente ejemplo. Léanlo y expliquen qué problema hay en el código inicial y qué efecto tiene la sustitución que se propone en el segundo pedazo de código.

```
1  #include <iostream>
2
3  using namespace std;
4
5  class Shape {
6  public:
7      virtual void draw() {
8          cout << "Drawing shape..." << endl;
9      }
10 };
11
12 class Circle : public Shape {
13 public:
14     void draw() {
15         cout << "Drawing circle..." << endl;
16     }
17 };
18
19 int main() {
20     Shape *shape = new Circle();
21     shape->draw();
22
23     delete shape;
24     return 0;
25 }
```

In this example, the `Shape` *class has a virtual function* `draw()` *which is overridden by the* `Circle` *class. The* `draw()` *function is called on an instance of* `Circle` *using a pointer to the base class* `Shape`.

The problem with this code is that the `draw()` *function in* `Circle` *is not declared as virtual. This means that when the* `draw()` *function is called on an instance of* `Circle` *using a pointer to* `Shape`, *the* `draw()` *function in* `Shape` *is called instead of the one in* `Circle`.

To fix this, the `draw()` *function in* `Circle` *should be declared as virtual:*

```
1  class Circle : public Shape {
2  public:
3      virtual void draw() {
4          cout << "Drawing circle..." << endl;
5      }
6  };
```

Now, when the `draw()` *function is called on an instance of* `Circle` *using a pointer to* `Shape`*, the* `draw()` *function in* `Circle` *will be called, as expected.*

The correct use of virtual functions is important in C++, as it enables polymorphism and allows subclasses to override the behavior of their base classes.