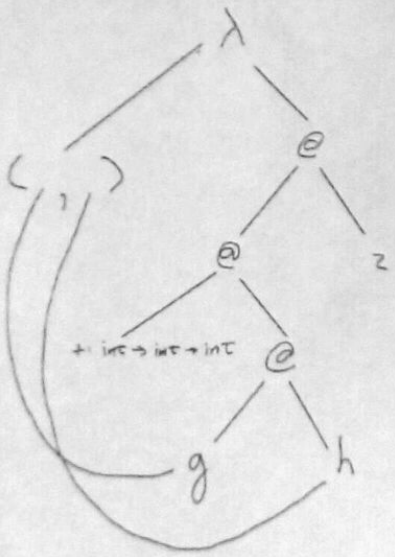
Paradigmas de la Programación – Primer Parcial

3 de Mayo de 2022

Apellido y Nombre: _____

1. **[10 pt.]** Escriba la expresión que se representa en este árbol de tipos. ¿Hay algún error de tipos? Si lo hay, explique dónde se encuentra. Si no lo hay, escriba la signatura de tipos de la expresión.



2. **[10 pt.]** El siguiente código tiene una instrucción . Esta instrucción produce un salto a otra parte del código. Entonces, el código que usa , ¿es una característica del código spaghetti? ¿por qué?

```
1   while (true) {
2     decrease(k);
3     if (k!=0) {
4       set(p,u_k);
5       increasev2(a[k]);
6       if (q != 0) {
7         break;
8       }
9     } else {
10      return;
11    }
12  }
```

3. [10 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;
```

4. [10 pt.] El siguiente texto explica lo que es un *thunk*.

> *A simple implementation of çall by name"might substitute the code of an argument expression for each appearance of the corresponding parameter in the subroutine, known as a "thunk".*

Escriba el *thunk* resultante de compilar en el siguiente programa, donde los argumentos se pasan mediante la estrategia *call-by-name*:

```
1  int i;
2  char array[3] = { 0, 1, 2 };
3
4  i = 0;
5  f(a[i]);
6
7  int f(int j)
8  {
9      int k = j;
10     i = 2;
11     k = j;
12 }
```

5. [10 pt.] El siguiente programa en C++, ¿es declarativo? ¿Por qué?

```
1  int values[4] = { 8, 23, 2, 4 };
2  int sum = SumArray(values);
3  RotateArrayIndices(values, -1);
```

6. [30 pt.] Explique verbalmente qué va sucediendo en la ejecución del siguiente programa, ayudándose de diagramas de la pila de ejecución cuando lo considere necesario. No es necesario representar las variables locales, control links, access links, retorno de función ni ninguna otra información que no sea relevante al proceso de manejo de excepciones.

Ayuda: la ejecución de este programa imprime lo siguiente:

```
Inner catch block handling exception thrown from try block.
Inner finally block
Outer catch block handling exception thrown from finally block.
Outer finally block
```

```
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 }
```

7. **[20 pt.]** Lea el siguiente texto y explique brevemente qué es una colisión de nombres. Atención: en clase hemos usado otro término para referirnos al mismo fenómeno. Si tenemos un lenguaje orientado a objetos, hay un contexto en el que podemos encontrar una mayor cantidad de colisiones de nombres, descríbalo.

   No todos los lenguajes orientados a objetos permiten el tipo de contextos con que es más fácil que suceda una colisión de nombres. Explique algún caso de un lenguaje con orientación a objetos que limitó su expresividad para evitar esos contextos.

   An example of a naming collision

```
1  a.cpp:
2
3  #include <iostream>
4
5  void myFcn(int x)
6  {
7      std::cout << x;
8  }
9  main.cpp:
10
11 #include <iostream>
12
13 void myFcn(int x)
14 {
15     std::cout << 2 * x;
16 }
17
18 int main()
19 {
20     return 0;
21 }
```

When the compiler compiles this program, it will compile a.cpp and main.cpp independently, and each file will compile with no problems.

However, when the linker executes, it will link all the definitions in a.cpp and main.cpp together, and discover conflicting definitions for function myFcn. The linker will then abort with an error. Note that this error occurs even though myFcn is never called!