1. Define y ejemplifica el concepto de continuación (*continuation*), ¿Qué es? ¿Cuáles son sus características? ¿Cómo es su comportamiento en memoria?, su utilidad, etc.

   Solución. Es un estilo de programación en el que el flujo de control se pasa explícitamente en forma de continuación, lo que permite optimizar el proceso de compilación. Algunas de sus características son.

   - Es un estilo de codificación.
   - Las continuaciones escritas, se pasan de forma explícita.
   - Cada función recibe un parámetro adicional, representando la continuación de la función.

   Este tipo de estilo hace que el tamaño de la pila crezca con cada llamada a las funciones, haciendo que aumente el riesgo de desbordamiento de memoria. Para evitar lo anterior, se emplea la optimización TCO (Tail-Call Optimization).

   Veamos el siguiente ejemplo.

Forma directa	Continuation

function(x){
  return x;
}
	function(x,y){
	  y(x);
	}

2. Implementa la siguiente función recursiva usando CPS.

```
filterPos :: [Int] -> [Int]
filterPos [] = []
filterPos (x:xs)
   | (x > 0) = x:filterPos xs
   | otherwise = filterPos xs
```

   Solución.

```
filterPos :: [Int] -> [Int]
filterPos list = filterPosCPS list (x -> x)

filterPosCPS :: [Int] -> ([Int] -> [Int]) -> [Int]
filterPosCPS [ ] a = (a [ ])
filterPosCPS (x:xs) a
| (x > 0) = filterPosCPS xs (y -> (a(x:y)))
| otherwise = filterPosCPS xs a
```

3. Menciona 3 ventajas que presentan los lenguajes tipificados implícitamente sobre los tipificados explícitamente.

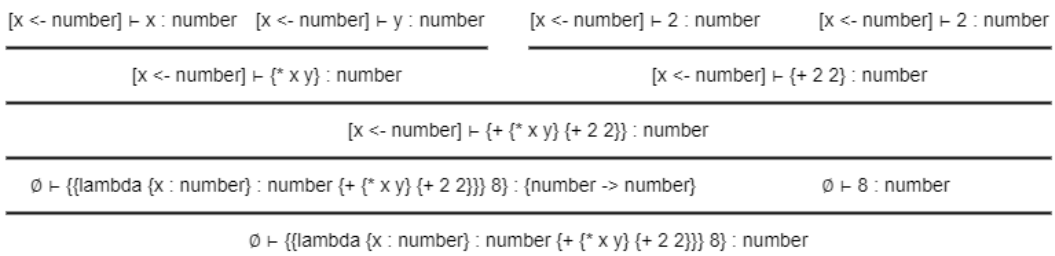
   Solución.

   - Podemos omitir información explícita de tipos.
   - Mayor facilidad en la creación de grandes sistemas.
   - Mayor flexibilidad en tiempo de ejecución.

4. Realiza el juicio de tipo para la siguiente expresión:

```
((lambda (x : number) : number
   (+ (* x y) (+ 2 2)))
 8)
```

Solución.



5. Utiliza el algoritmo de inferencia de tipos con la siguiente expresión:

   `((lambda (x) x) (* 7 1))`.

   Solución. Reescribimos la función como $^{1}((lambda\ ^{2}(x)\ ^{3}x)\ ^{4}(*\ ^{5}7\ ^{6}1))$. Con las siguientes restricciones.

   - $\boxed{1}$ Es una función que tiene como entrada a $\boxed{2}$ y como cuerpo a $\boxed{4}$. Por lo tanto, tenemos la restricción:
   $$\boxed{1} = \boxed{2} \to \boxed{4}$$
   - $\boxed{2}$ Es simplemente el parámetro de la función, sin embargo, notamos que usa el cuerpo de la función por su nombre, por lo tanto, tenemos que:
   $$\boxed{2} = \boxed{3}$$
   - $\boxed{3}$ Es el valor que se tomará después de ejecutar $\boxed{4}$. Por lo tanto tenemos que:
   $$\boxed{3} = \boxed{4}$$
   - $\boxed{4}$ Es un producto, por lo tanto, su tipo el number, así también su lado izquierdo y derecho. Por lo que obtenemos las siguientes restricciones:
   $$\boxed{4} = number$$
   $$\boxed{5} = number$$
   $$\boxed{6} = number$$

     Por lo que $\boxed{2}$ y $\boxed{3}$ de igual forma son de tipo number.

   Para obtener el tipo completo de la expresión, sustituimos las restricciones según convenga. Quedando de la siguiente manera:

   $$\boxed{1} = number \to number$$
   $$\boxed{2} = number$$
   $$\boxed{3} = number$$

Marín Parra José Guadalupe de Jesús

$$\boxed{4} = number$$
$$\boxed{5} = number$$
$$\boxed{6} = number$$

Por lo tanto, el tipo de la función es $number \to number$.

Ahora veamos la unificación de la expresión, quedando ${}^{1}((lambda\ {}^{2}(x)\ {}^{3}x)\ {}^{4}(*\ {}^{5}7\ {}^{6}1))$.

Si tomamos $\boxed{3}$ del algoritmo de unificación, obtenemos la siguiente tabla.

Acción	Pila	Sustituciones
Inicialización	$\boxed{1} = \boxed{2} \to \boxed{4}$ $\boxed{2} = \boxed{3}$ $\boxed{3} = \boxed{4}$ $\boxed{4} = number$ $\boxed{5} = number$ $\boxed{6} = number$	
3	$\boxed{2} = \boxed{3}$ $\boxed{3} = \boxed{4}$ $\boxed{4} = number$ $\boxed{5} = number$ $\boxed{6} = number$	$(\boxed{1} \leftarrow \boxed{2} \to \boxed{4})$
3	$\boxed{3} = \boxed{4}$ $\boxed{4} = number$ $\boxed{5} = number$ $\boxed{6} = number$	$(\boxed{1} \leftarrow \boxed{2} \to \boxed{4})$ $(\boxed{2} \leftarrow \boxed{3})$
3	$\boxed{4} = number$ $\boxed{5} = number$ $\boxed{6} = number$	$(\boxed{1} \leftarrow \boxed{2} \to \boxed{4})$ $(\boxed{2} \leftarrow \boxed{3})$ $(\boxed{3} \leftarrow \boxed{4})$
3	$\boxed{5} = number$ $\boxed{6} = number$	$(\boxed{1} \leftarrow \boxed{2} \to number)$ $(\boxed{2} \leftarrow \boxed{3})$ $(\boxed{3} \leftarrow \boxed{4})$

3	$\boxed{6} = number$	$(\boxed{1} \leftarrow number \rightarrow number)$ $(\boxed{2} \leftarrow number)$ $(\boxed{3} \leftarrow number)$ $(\boxed{4} \leftarrow number)$ $(\boxed{5} \leftarrow number)$
3		$(\boxed{1} \leftarrow number \rightarrow number)$ $(\boxed{2} \leftarrow number)$ $(\boxed{3} \leftarrow number)$ $(\boxed{4} \leftarrow number)$ $(\boxed{5} \leftarrow number)$ $(\boxed{6} \leftarrow number)$

6. Da tres ventajas de contar con polimorfismo explícito en un lenguaje de programación. Utiliza ejemplos.

   Solución.

   - Definición de datos en tiempo de ejecución. Por ejemplo los genéricos en Java.

     ```
     public class Clase<T> {
         ...
     }

     Clase<Int> prueba = new Clase<Int>();
     ```

     Donde T es el tipo de dato a reemplazar.
   - Creación de objetos polimórficos. Por ejemplo la declaración de un objeto de una superclase en una subclase.

     Superclase objeto = new (Subclase);
   - No hay especificación sobre el tipo de dato con el que se está trabajando.