II
Semestre
2025

Instituto Tecnológico de Costa Rica

Escuela de Computación

IC-5701 Compiladores e Intérpretes GR 40

Proyecto 02_Casos_de_Uso

Profesor:

José Castro Mora

Trabajo elaborado por:

William Jiménez Venegas - 2023396021

Miguel Alejandro Madrigal Ramírez - 2020219677

Josué Montero Arguedas - 2023152831

25 de setiembre, 2025

1. Usar funciones anónimas para la declaración de variables

```
let
 var f: lambda(Integer): Integer
in
 f:= lambda(n: Integer): Integer ~ n * n
```

Caso de Uso	Declaración de variables de tipo lambda
Descripción	El compilador Triangle permite la declaración de variables cuyo valor sea asignado a partir de una función anónima, identificadas por la palabra reservada “lambda”.
Precondiciones	1. La gramática de Triangle permite la declaración de variables de tipo lambda. 2. El AST (Abstract Syntax Tree) puede identificar los nodos que representan a un tipo lambda. 3. El analizador contextual puede validar los tipos lambda.
Flujo	1. El parser reconoce la declaración de tipo lambda. 2. Se construye el nodo lambda en el AST. 3. El tipo del parámetro es válido. 4. El tipo del dato de retorno es válido. 5. El encoder genera el espacio para almacenar su referencia.
Postcondiciones	1. La variable de tipo lambda fue generada correctamente. 2. Su sección del árbol se generó de forma adecuada.
Excepciones	1. La declaración no sigue la sintaxis esperada. 2. Los tipos de parámetro/retorno no son válidos. 3. La variable ya ha sido declarada. 4. Incompatibilidad entre la función lambda y la variable.

Código con error

```
let
 var f: lambda(Integer): Integer
in
 f:= lambda(n: Boolean): Integer ~ n * n
```

2. Usar la función anónima con un func en el cuerpo

```
let
 var x: Integer;
 var y: Integer;
 var f: lambda(a: Integer, b: Integer, c: Integer): Integer;

 func max(x: Integer, y: Integer, z: Integer): Integer ~
   if x >= y then
     if x >= z then x else z
   else
     if y >= z then y else z
in
 begin
   x:= 5;
   y:= 2;
   f:= lambda(a: Integer, b: Integer, c: Integer): Integer
     ~ max(a, b, c);
   putint(f(x, y, 3));
end
```

Caso de Uso	Utilizar una función anónima que haga uso de una función con func
Descripción	El compilador Triangle permite que una función definida, pueda ser utilizada dentro de una función anónima “lambda”.
Precondiciones	1. La gramática de Triangle permite la declaración de funciones anónimas como parámetros formales.

	2. El AST puede identificar la función func dentro del cuerpo de una función anónima. 3. El analizador contextual puede validar que los argumentos pasados a la función anónima, tiene el mismo tipo que los esperados como parámetros.
Flujo	1. El parser reconoce la declaración de una función func como parte del cuerpo de una función lambda. 2. Se construye la sección del árbol correspondiente a la función lambda. 3. El analizador contextual verifica la compatibilidad entre argumento y parámetro. 4. El encoder genera las referencias correspondientes para el argumento y el parámetro.
Postcondiciones	1. La llamada a la función anónima es representada, adecuadamente, en el AST. 2. El código generado puede ejecutar la función anónima.
Excepciones	1. La declaración no sigue la sintaxis esperada. 2. La cantidad de parámetros no concuerda con los esperados. 3. Hay incompatibilidad entre el tipo del argumento y parámetro esperado.

Código con error

```
let
 var x: Integer;
 var y: Integer;
 var f: lambda(a: Integer, b: Integer, c: Integer): Integer;

 func max(x: Integer, y: Integer, z: Integer): Integer ~
   if x >= y then
     if x >= z then x else z
```

```
      else
        if y >= z then y else z
    in
     begin
      x:= 5;
      y:= 2;
      f:= lambda(a: Integer, b: Integer, c: Integer): Integer
        ~ max(a, b, c);
      putint(f(x, y));
     end
```

3. Usar la función anónima de modo que sea compatible con func

```
    let
      func duplicar(func f (x: Integer): Integer, y: Integer):
Integer ~
        f(y);

      var miFuncion: lambda(Integer): Integer;
      var n: Integer
    in
      begin
        n:= 5;
        miFuncion:= lambda(x: Integer): Integer ~ x * 2;
        putint(duplicar(miFuncion, n))
      end
```

Caso de Uso	Usar la función anónima de modo que sea compatible con `func`
Descripción	El compilador triangle permite que una función lambda sea compatible y asignable a un parámetro formal declarado como func, respetando la equivalencia de tipos.

Precondiciones	1. La gramática de Triangle declara parámetros formales de tipo `func` y de tipo `lambda`. 2. El sistema de tipos considera equivalentes los tipos `lambda` y `func` cuando sus firmas coinciden. 3. El AST puede representar asignaciones y pasos de parámetros entre funciones anónimas y declaradas.
Flujo	1. El parser reconoce la declaración de un parámetro func y una variable de tipo `lambda`. 2. Se construye el nodo AST para la asignación o paso de parámetro con la función anónima. 3. El analizador contextual verifica la compatibilidad de tipos entre `lambda` y func. 4. El generador de código maneja la representación en tiempo de ejecución de la función anónima
Postcondiciones	1. La función anónima fue aceptada correctamente donde se espera un func del mismo tipo. 2. El código generado ejecuta correctamente la función anónima asignada o pasada por un parámetro.
Excepciones	1. La firma de la función anónima no coincide con la esperada por el parámetro func. 2. La variable de tipo `lambda` no es compatible en contextos de asignación.

Código con error

```
let
  func duplicar(func f (x: Integer): Integer, y: Integer):
Integer ~
    f(y);
```

```
  var miFuncion: lambda(Integer): Boolean;
  var n: Integer
in
  begin
    n:= 5;
    miFuncion:= lambda(x: Integer): Boolean ~ x * 2;
    putint(duplicar(miFuncion, n))
  end
```

4. Usar expresiones lambda para simular un map

```
let
 var arrayA: array 4 of Integer;
 var arrayB: array 4 of Integer;
 var i: Integer;
 var f: lambda(x: Integer): Integer
in
 begin
   arrayA[0]:= 1; arrayA[1]:= 2;
   arrayA[2]:= 3; arrayA[3]:= 4;
   f:= lambda(x: Integer): Integer ~ x * 2;

   for i:= 0 to 3 do
     arrayB[i]:= f(arrayA[i]);
   for i:= 0 to 3 do
     putint(arrayB[i])
 end
```

Caso de Uso	Usar expresiones lambda dentro de una estructura for para simular un map

Descripción	El compilador Triangle permite aplicar funciones anónimas (definidas con lambda) dentro de estructuras de control como for. Esto permite recorrer un rango de valores y aplicar dinámicamente una operación definida por una función anónima a cada elemento del rango.
Precondiciones	1. La gramática de Triangle permite la declaración de variables de tipo lambda. 2. El AST (Abstract Syntax Tree) puede identificar los nodos que representan a un tipo lambda. 3. El analizador contextual puede validar los tipos lambda. 4. Las funciones anónimas pueden ser invocadas dentro del cuerpo de un for. 5. El AST (Árbol de Sintaxis Abstracta) puede representar llamadas a funciones anónimas dentro de estructuras de control. 6. El analizador contextual puede verificar la compatibilidad de tipos entre el argumento iterado y el parámetro de la función.
Flujo	1. El parser reconoce la declaración de una variable de tipo lambda. 2. Se construye el nodo en el AST para la función anónima. 3. Se reconoce la estructura for y se agrega su nodo correspondiente en el AST. 4. Dentro del cuerpo del for, se identifica la invocación a la función lambda. 5. El analizador contextual valida: • Que la función lambda se invoca correctamente. • Que el tipo del valor iterado (x) es compatible con el parámetro de la función. 6.El encoder genera el código correspondiente para:

	• La iteración del bucle. • La invocación a la función anónima en cada paso del bucle.
Postcondiciones	1. El programa se ejecuta y aplica la función lambda sobre cada valor del rango definido por el for. 2. El compilador garantiza la seguridad de tipos en la invocación de la lambda.
Excepciones	1. Si la expresión lambda dentro del for no cumple con la gramática de Triangle. 2. Si el for itera enteros pero la función espera otro tipo. 3. Se intenta invocar una función anónima que no tiene asignación previa. 4. Si el compilador no puede resolver la referencia de la lambda dentro del cuerpo del bucle. 5. Número o tipos de parámetros incorrectos.

Código con error

```
let
 var arrayA: array 4 of Integer;
 var arrayB: array 4 of Integer;
 var i: Integer;
 var f: lambda(x: Integer): Integer
in
 begin
   arrayA[0]:= 1; arrayA[1]:= 2;
   arrayA[2]:= 3; arrayA[3]:= 4;
   f:= lambda(x: Integer): Integer ~ x * 2;
```

```
  for i:= 0 to 3 do
    arrayB[i]:= f(arrayA[i], 2);
  for i:= 0 to 3 do
    putint(arrayB[i])
end
```

Nota: Si se quieren ejecutar los scripts, estos se encuentran en: **ide-triangle-v1.1.src\test\Lambda**