Integrantes:
- Cruz Jiménez Alejandro – 316008488 – alexcj71@ciencias.unam.mx
- Sandoval Mendoza Antonio – 316075725 – tonodx17@ciencias.unam.mx
- Sinencio Granados Dante Jusepee – 316246019 – spartanonix@ciencias.unam.mx

El proceso que lleva a cabo en el compilador es el que se usa comúnmente en casi todos los compiladores (dividir la estructura en 3 partes). Lo primero que hace es tomar un código fuente (en mi caso es el archivo ejemplo.mt), y después se procede a procesar el contenido en ese archivo en el front-end.

Primero se aplica un análisis sintáctico al archivo con el método "read-file"*, para después aplicar los pre procesos del front-end que en el caso del proyecto los pre procesos se aplican en el siguiente orden:*

1) ***parse-LF:*** *Parser para nuestro lenguaje fuente.*
2) ***rename-var:*** *Nombra cada variable de forma distinta para poder identificarlas.*
3) ***remove-one-armed-if:*** *Función para eliminar el if de la expresión dada.*
4) ***remove-string:*** *Función para eliminar las strings como elementos terminales en una expresión dada.*
5) ***traductor-LNI2-L6:*** *Función para traducir el lenguaje LNI2 a L6 en mi código.*
6) ***curry-let:*** *Función que tomará la expresión dada para transformar su let universal a un let por cada asignación.*
7) ***identify-assigments:*** *Función que tomará la expresión dada para transformar sus let's a los que se le están asignando lambdas a un solo letrec.*
8) ***un-anonymous:*** *Función que tomará la expresión dada para cachar las lambdas y asignarlas en un letfun.*
9) ***verify-arity:*** *Función que tomará la expresión dada para verificar si los primitivos están siendo aplicados en sus operadores con la aridad correcta.*
10) ***verify-vars:*** *Función que tomará la expresión dada para verificar que la expresión no contenga variables libres.*
11) ***curry:*** *Función que toma la expresión dada y la anida con su respectivo tipo.*

Después para el middle-end se debe obtener la información de que tipos son los identificadores, para ello se aplicaron los pre procesos:

1) ***type-const:*** *Función que convierte los quots en constantes.*
2) ***type-infer:*** *Función que toma la expresión e infiere el tipo de sus variables.*
3) ***uncurry:*** *Función que elimina expresiones lambda.*

Y por último para el back-end se debe obtener el código de bajo nivel para entendimiento de la máquina, para ello se aplicaron los pre procesos:

1) ***assignment:*** *Función que elimina el valor de los identificadores y el tipo de let, letrec y letfun.*
2) ***list-to-array:*** *Función que traduce las listas en arreglos.*

*Los resultados de cada etapa se registran en archivos diferente para llevar el registro de cómo se va transformando, para el front-end se crea el archivo "*ejemplo.fe*", para el middle-end "*ejemplo.me*" y para el back-end se crea el archivo "*ejemplo.c*".*

Notas:

- *El proyecto está dividido en dos archivos, el primero es el de "*Compilador.rtk*" que contiene las funciones de las prácticas pasadas modificadas con los nuevos requerimientos (incluyendo la función c) y el segundo archivo es el de "*CompiladorC.rtk*" donde se encuentran los pre procesos ordenados por front-end, middle-end y el backend.*
- *Para ejecutar el proyecto lo hice desde DrRacket corriendo el archivo CompiladorC.rtk y mandando a llamar el método compilar de la forma (compilar "*ejemplos/ejemplo.mt*").*
- *Al momento de entregar el proyecto el código está estructurado con las indicaciones de cada inciso que realice, hice el los incisos obligatorios 2, 5 y 7, y también los incisos seleccionables 3 y 4, pero por algunas fallas que tenia mi codigo en las prácticas pasadas y que no logre solucionar al momento de realizar el proyecto, aunque si compilen los dos archivos si se trata de usar el código con algunos ejemplos tiene fallas que evitan que se logre la compilación satisfactoriamente.*