Optimización de compilador

Se tiene implementado el gencode para la gramática:

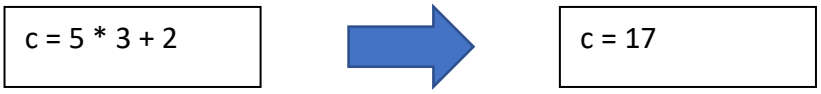
- Program ::= VarDecList FunDecList
- VarDecList ::= (VarDec)*
- FunDecList ::= (FunDec)+
- FunDec ::= **fun** Typ id **(**[ParamDecList] **)** Body **endfun**
- Typ::= **int** | **void**
- Body ::= VarDecList StmtList
- ParamDecList ::= Type id (**,** Type id)*
- VarDec ::= **var** Type VarList **;**
- Type ::= id
- VarList :: id (,id)*
- StmtList ::= Stmt **(;** Stmt **)***
- Stmt ::= **id =** CExp |
  - **print (** CExp **)**
  - **if** CExp **then** Body [**else** Body] **endif**
  - **while** CExp **do** Body **endwhile**
  - **return (** [CExp] **)**
- CExp ::= Exp [(**<**) Exp]
- Exp ::= Term ((**+** | **-**) Term)*
- Term ::= Factor ((*****|**/**) Factor)*
- Factor ::= id | Num **|** Bool | **(** Exp **) | id (** [ArgList] **)**
- ArgList ::= CExp (, CExp)*
- Bool **::= true | false**

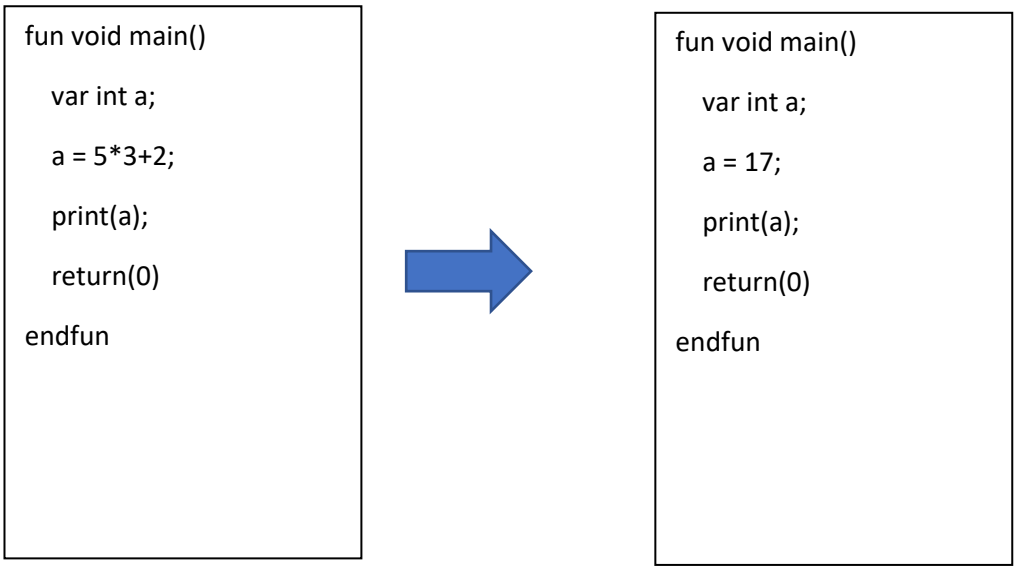
Tarea 1: Optimización de Expresiones Aritméticas

Implemente una optimización en la generación de código para expresiones aritméticas, utilizando el algoritmo de etiquetado de árboles. Este enfoque permite minimizar el número de registros temporales y mejorar la eficiencia del código generado.

Tarea 2: Plegado de Constantes (Constant Folding)

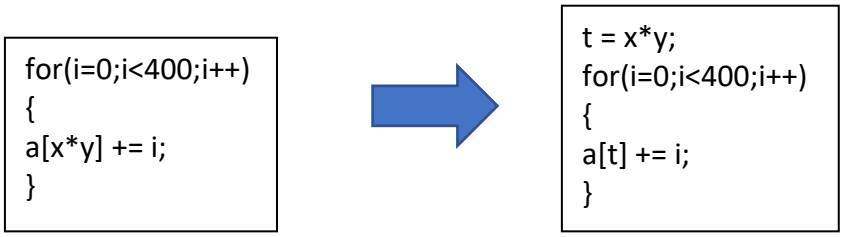
Modifique el compilador para que recorra el Árbol de Sintaxis Abstracta (AST) y evalue en tiempo de compilación las expresiones cuyos operandos sean constantes. Estas expresiones deben ser reemplazadas por su valor resultante, reduciendo así el trabajo en tiempo de ejecución.





Tarea 3: Code Hoisting

Implemente la técnica de Code Hoisting, que consiste en mover expresiones o instrucciones computacionalmente costosas fuera de estructuras de control, como bucles o condicionales, siempre que sus valores permanezcan invariantes. Esto permite evitar cálculos repetidos y mejora la eficiencia del programa.



Tarea 4: Instalador Automático

En la entrega agregar el sh.

Desarrolle un script de instalación automática para el compilador UtecCompiler. El instalador debe:

- Descargar el código fuente desde un repositorio.
- Compilar los ejecutables UtecC y UtecCop.
- Instalarlos en el sistema, dejándolos disponibles desde la terminal.

```
#!/bin/bash

echo "===  UtecCompiler Auto-Build Installer by Luis"

#  Crea carpeta temporal
WORKDIR="$HOME/uteccompiler-build"
mkdir -p "$WORKDIR"
cd "$WORKDIR"

#  Clona tu repo de código fuente
echo "===  Cloning source code..."
git clone https://github.com/LuisEnriqueCortijoGonzales/uteccompiler-repo-source.git .

#  Compila UtecC y UtecCop
echo "===  Compiling UtecC..."
g++ main.cpp parser.cpp scanner.cpp token.cpp visitor.cpp exp.cpp -o UtecC

echo "===  Compiling UtecCop..."
g++ main.cpp parser.cpp scanner.cpp token.cpp visitor.cpp exp.cpp -o UtecCop

#  Instala en /usr/local/bin (requiere sudo)
echo "===  Installing executables..."
sudo mv UtecC /usr/local/bin/
sudo mv UtecCop /usr/local/bin/

#  Limpia
echo "===  Cleaning up..."
cd ~
rm -rf "$WORKDIR"

echo " Done!"
```